Power-Based Diagnosis of Node Silence in Remote High-End Sensing Systems

YONG YANG, Philips Research North America
LU SU, State University of New York at Buffalo
MOHAMMAD KHAN, University of Connecticut
MICHAEL LEMAY, TAREK ABDELZAHER, and JIAWEI HAN, University of Illinois, Urbana-Champaign

Troubleshooting unresponsive sensor nodes is a significant challenge in remote sensor network deployments. While prior work often targets low-end sensor networks, this article introduces a novel diagnostic tool, called the telediagnostic powertracer, geared for remote high-end sensing systems. Leveraging special properties of high-end systems, this in situ troubleshooting tool uses external power measurements to determine the internal health condition of an unresponsive node and the most likely cause of its failure. We develop our own low-cost power meter with low-bandwidth radio, propose both passive and active sampling schemes to measure the power consumption of the host node, and then report the measurements to a base station, hence allowing remote (i.e., tele-) diagnosis. The tool was deployed and tested in a remote solar-powered sensing system for acoustic and visual environmental monitoring. It was shown to successfully distinguish between several categories of failures that cause unresponsive behavior including energy depletion, antenna damage, radio disconnection, system crashes, and anomalous reboots. It was also able to determine the internal health conditions of an unresponsive node, such as the presence or absence of sensing and data storage activities (for each of multiple applications). The article explores the feasibility of building such a remote diagnostic tool from the standpoint of economy, scale, and diagnostic accuracy. The main novelty lies in its use of power consumption as a side channel, which has more availability than other I/O ports, to diagnose sensing system failures.

Categories and Subject Descriptors: C.2.3 [**Computer Systems Organization**]: Network Operations—*Network monitoring*; D.2.5 [**Software**]: Testing and Debugging—*Diagnostics*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Time series analysis*

General Terms: Algorithms, Design, Experimentation, Measurement, Reliability

---

A preliminary version of this article appeared in ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN) 2010 [Khan et al. 2010].
This work was done when Yong Yang was at the University of Illinois at Urbana-Champaign. Now he is with Philips Research North America.
This work was supported in part by NSF CNS 06-26342, NSF CNS 09-05014, NSF CNS 09-17218, NSF CNS 07-16626, NSF CNS 07-16421, ONR N00014-08-1-0248, NSF CNS 05-24695, and grants from the MacArthur Foundation, Boeing Corporation, and Lockheed Martin.
M. LeMay was with the University of Illinois at Urbana-Champaign while performing the work described herein, but he was employed by Intel Corporation at the time of submission. The views expressed are those of the authors only.
Authors' addresses: Y. Yang, Philips Research North America, 345 Scarborough Road, Briarcliff Manor, NY 10510; email: yong.yang_2@ philips.com; L. Su, State University of New York at Buffalo, 321 Davis Hall, Buffalo, New York 14260; email: lusu@buffalo.edu; M. Khan, University of Connecticut, 363 ITE Building, University of Connecticut, Storrs, CT 06269; email: maifi.khan@engr.uconn.edu; M. LeMay, Intel Corporation, 2111 NE 25th Ave. JF2-75, Hillsboro, OR 97124; email: m@lemays.org; T. Abdelzaher and J. Han, Siebel Center for Computer Science, 201 N Goodwin Avenue, Urbana, IL 61801; emails: zaher@illinois.edu, hanj@cs.uiuc.edu.

DOI: http://dx.doi.org/10.1145/2661639

Additional Key Words and Phrases: Energy, sensor networks, troubleshooting

ACM Reference Format:
Yong Yang, Lu Su, Mohammad Khan, Michael LeMay, Tarek Abdelzaher, and Jiawei Han. 2014. Power-based diagnosis of node silence in remote high-end sensing systems. ACM Trans. Sensor Netw. 11, 2, Article 33 (December 2014), 33 pages.
DOI: http://dx.doi.org/10.1145/2661639

1. INTRODUCTION

This article introduces a novel diagnostic system, called the *telediagnostic powertracer*, geared for remote high-end sensor network deployments, where the cost of in situ node maintenance is high. This telediagnostic system uses external low-bandwidth measurements of power consumed by sensor nodes to distinguish between several types of node failures, such as failures induced by energy depletion, radio failures, and software failures (e.g., system crashes). It is also able to infer the state of the application from power traces, such as whether or not sensing and data storage are operational on an unresponsive node. It is common that nodes of dedicated high-end sensor deployments constitute valuable assets such as embedded computers, high-end sensors, and expensive energy supplies and storage. For example, in this article, we use a testbed [Yang et al. 2009] that is composed of sensor nodes that cost more than $ 2,000 each. Therefore, one can easily accrue a total cost that justifies investment in additional diagnostic mechanisms and components.

The work described in this article is motivated by the need to reduce the cost of troubleshooting remotely deployed sensing systems. When remotely deployed nodes become unresponsive, it is generally hard to determine what caused some node to become silent without sending a person to the field. If the cost of such field trips is high, remote damage assessment becomes highly desirable to assess the need for intervention. For example, if the cause of the problem is energy depletion (in a solar-powered system), there may not be much that can be done about it until the energy source is restored (e.g., weather improves). On the other hand, if the cause is attributed to a transient error (e.g., a system crash), power-cycling the system remotely may fix the problem. If the cause is attributed to a hardware malfunction (e.g., a radio failure), the urgency of repair may depend on whether or not the failure has affected the ability of the application to sample and store data. If the application continues to sample and locally store data, then there may be no need for immediate intervention. In contrast, some failures may require urgent attention. For instance, it is urgent to intervene if there is evidence of water damage that may cascade to other nodes or devices. Another example, experienced by the authors on one occasion, was a node that entered a cycle of repeated reboots. The cycle ultimately led to a hardware failure. Early intervention could have saved the node. Our telediagnostic system provides strong clues as to what might be wrong with a node, making it possible to plan intervention accordingly.

Prior work on sensor network troubleshooting [Whitehouse et al. 2006; Yang et al. 2007; Liu et al. 2010; Tolle and Culler 2005; Krunic et al. 2007; Sundaram et al. 2009; Wang et al. 2011] often focused on bugs that alter system behavior but do not render nodes unresponsive. Hence, a diagnostic system could monitor and communicate node states via regular radio. We cannot use techniques that require the participation of failed nodes in this article since we investigate *silent* nodes that, by definition, cannot communicate. Some work [Ramanathan et al. 2005; Rost and Balakrishnan 2006] concerned itself with localizing which node or link failed or malfunctioned when sensor network performance is impaired. Indeed, while other network nodes can localize a peer's failure, in this article, we take the next step of understanding *why* the identified peer went silent, as well as retrieving its coarse-grained internal state. Moreover, most of these aforementioned works target low-end sensor nodes (e.g., Tmote and MicaZ motes), while we focus on troubleshooting high-end sensing systems, whose special properties (e.g., relatively high power consumption and high cost) make some solutions that are infeasible for low-end systems become feasible.

When the primary communication channel of a node goes silent due to a failure, a secondary channel is needed to communicate further information on its state. By "secondary channel," we mean any mechanism (not necessarily a radio) that conveys or "leaks" information. Several tools [HP iLO 2010; IBM Remote Supervisor 2010; Dyer et al. 2007] were proposed to diagnose target systems using different kinds of I/O ports, such as serial ports or PCI. The applicability of these solutions is, however, restricted by the availability of such ports on the system. Deployed sensor nodes might, for example, be optimized for power, enclosure cost, or waterproofing ease. Hence, unnecessary I/O ports might be removed. Considering that sensing, communication, computation, and storage necessarily need power, power consumption may be used as a side channel to infer the states of nodes to be monitored. Compared to other I/O ports, it has a more universal applicability.

Therefore, in this work, we investigate the degree to which power consumption measurements of an unresponsive node can be used as a side channel to help diagnose the causes of node silence. Specifically, for energy and cost efficiency, we attach an external *low-end* power meter with its own radio to each deployed node to sample the node's power consumption, and then the traces of power consumption are wirelessly transmitted to a diagnostic base station, where diagnostic schemes use the received traces to infer the cause of node silence, as well as the health status of the applications on the node.

In this work, we propose two diagnostic schemes. The first scheme, which we initially introduced in Khan et al. [2010], relies on the passive sampling of power consumption and identifies node states based on their built-in power signatures. With this scheme, before runtime diagnosis, we collect power traces and train a classifier for each possible combination of applications and failure modes, and then use the classifier to classify the received power traces during runtime. Note that the number of diagnostic states grows exponentially with the number of applications. However, this classifier-based diagnostic scheme is still applicable in many deployments that have very specific purposes and thus do not run a wide range of different applications concurrently.

In case a deployment does run a large number of concurrent applications, we propose another diagnostic scheme that, instead of just passively measuring the host's power consumption, places a module into each host to actively inject unique power patterns (watermarks) into the power consumption traces based on the current system status. Since watermarks adhere to a pre-agreed-upon code, there is no need for prior training. We have implemented both these passive sampling and active watermarking schemes and compare their performance in this article.

While the approach of exploiting power traces to diagnose problems using our telediagnostic system is applicable, in principle, to a wide array of high-end sensing systems, we present it and evaluate its performance on a specific deployed platform, called *SolarStore* [Yang et al. 2009]. The platform is intended for high-bandwidth sensing applications such as structural, acoustic, or video monitoring. It bundles higher-end solar-powered sensor node hardware with data storage and communication services. It therefore serves as a good example of the types of high-end sensing systems that our powertracer is designed to help troubleshoot. We show that, by remotely analyzing low-bandwidth power traces collected by cheap wireless power meters attached to the deployed SolarStore nodes, it is possible to infer useful information about the nature of node failures when they occur, as well as recognize some coarse-grained application states.

The rest of this article is organized as follows. Section 2 introduces related work on sensor network troubleshooting. Section 3 discusses general design guidelines. Section 4 presents the implementation of powertracer on a SolarStore testbed. Section 5 and Section 6 explore various diagnostic algorithms based on passive sampling and active watermarking, respectively. Section 7 discusses the limitations of our current work along with possible future extensions and improvements. The article concludes with Section 8.

2. RELATED WORK

Reliability of deployed systems has always been a great challenge in wireless sensor network research. Traditional troubleshooting tools, including simulation and emulation [Levis et al. 2003; Girod et al. 2004; Wen and Wolski 2006; Polley et al. 2004; Eriksson et al. 2009; Du et al. 2010], are good at predeployment testing in a controlled lab environment, where one can afford the luxury of changing and inspecting code, restarting nodes, and replacing broken hardware as necessary [Cao et al. 2008; de Jong et al. 2009; Lodder et al. 2008]. In contrast, troubleshooting deployed systems, especially those deployed in remote outdoor environments [Werner-Allen et al. 2005; Zhang et al. 2004; Yang et al. 2009; Ingelrest et al. 2010; Dyo et al. 2010], is a significantly harder problem.

Quite a few tools were developed to troubleshoot deployed systems remotely. Some of this work focuses on identifying the cause of anomalous network behavior by discovering which node or link failed or malfunctioned. For example, Sympathy [Ramanathan et al. 2005] infers node or link failures from reduced network throughput. Memento [Rost and Balakrishnan 2006] implements a distributed failure detector by having nodes in the network cooperatively monitor each other. Besides localizing failures, other tools were designed to inspect system states and identify programming bugs [Whitehouse et al. 2006; Yang et al. 2007; Sundaram et al. 2009; Wang et al. 2011]. PAD [Liu et al. 2010] tries to reconstruct a global view of the network from partial information provided by different nodes and identify problems such as software crashes, hardware failures, and network congestion. SNMS [Tolle and Culler 2005] can be used to gather system-wide performance statistics such as packet loss and throughput. NodeMD [Krunic et al. 2007] runs a runtime detection algorithm for early detection of failure symptoms. Agnostic Diagnosis [Miao et al. 2011] is designed to identify unknown failures through analyzing the correlations among different system metrics such as the radio-on time and the number of packets transmitted. Kamal et al. [2014] propose a framework that exploits in-network packet tagging using the Fletcher checksum and server-side network path analysis to detect persistent path changes that are indicative of network failures. Nie and Ma [2011] and Nie et al. [2012] propose a couple of content-centric diagnostic strategies that study the relationship between sensory readings and node failures/misbehaviors without introducing additional diagnostic traffic. A common underlying assumption of these tools is often that the deployed system is accessible through the normal data channel and can provide information when requested. When deployed sensor nodes become unresponsive, prior approaches can, at best, localize the failed node. In contrast, in this article, our goal is to infer further information on the unresponsive node, such as whether any applications are still running (albeit unable to communicate).

When the main communication channel is down, tools are needed to retrieve diagnostic information from side channels. For example, for the purpose of managing computer servers in a remote data center, out-of-band tools, such as remote supervisor adapters[1] for IBM x86-based servers and iLO[2] for HP servers, have been used. They offer access to servers that are otherwise compromised. These tools are usually based on a System-on-Chip architecture, having their own processor, memory, battery, and network connections. They access the host via I/O ports (e.g., PCI or IPMI). While

[1]ftp://ftp.software.ibm.com/systems/support/system_x_pdf/48p9832.pdf.
[2]http://h18013.www1.hp.com/products/servers/management/remotemgmt.html.

such ports are ubiquitous in server farms, it is not clear that they are as popular with specialized embedded devices optimized for other deployment considerations such as form-factor, waterproofing, or enclosure cost.

The idea of out-of-band management has also been utilized in telecommunications [Horak 2007] with the very different purpose of exchanging call control information. In the sensor network community, researchers have also investigated an out-of-band methodology for developing and testing deployed systems. For instance, DSN [Dyer et al. 2007] can be deployed as an additional wireless backbone network for an existing system. A DSN node (e.g., a mote) is attached to each sensor node through a dedicated connection (e.g., via UART ports) to observe, control, and reprogram the sensor node.

Compared to the previously mentioned approaches, power consumption is a more general channel that is available on every system that needs power to perform. It has already been demonstrated [Kansal and Zhao 2008; Chuang et al. 2005] that power signatures can provide helpful information for the system designer to detect and diagnose problems. Another example is the Nonintrusive Load Monitoring (NILM) algorithm that analyzes the energy consumption of a segment of a building and determines what appliances are in use in that segment without requiring meters to be individually attached to those appliances. The seminal work on NILM [Hart 1992] classifies loads into categories based on their power consumption profiles and presents a clustering-based algorithm for detecting transitions between discrete appliance states based on predetermined profiles of those states. Many other NILM algorithms have been developed, but they typically require electric meters with high sampling rates [Sultanem 1991; Li et al. 2005]. However, meters with high sampling frequencies are typically too expensive to be deployed as add-ons to sensor nodes. Besides the cost, more energy would be required to transmit and process their measurements. Fortunately, we are interested only in gross-level assessment of state, to tell whether or not applications are still running. In fact, it has been shown [Zhou and Xing 2013] that even high-throughput data can be transmitted between the power meter and the host node through voltage/current load modulation. However, the communication only happens during node inactive periods, within which most onboard components fall asleep (i.e., power consumption is relatively stable) and thus there is very little interference on the modulation and demodulation process. This inactive period usually does not exist on high-end sensing systems, where multiple applications and operating system processes are concurrently running and complicated hardware components are present. We show in this article that one can infer gross-level state information of high-end sensing systems using low-end meters with low sampling rates.

This work is also inspired by security research. In the field of computer security, side channels have long been pursued for the purpose of obtaining confidential information from computers. Electromagnetic emanations can be a rich source of data [Kuhn 2005]. Particular attention has been paid to analyzing the emanations of smartcards, and it has been demonstrated that private keys can be extracted from smartcards by analyzing their power consumption and radiation [Gandolfi et al. 2001]. Recent smartcard designs explicitly prevent such attacks. Acoustic emanations from certain machines also carry information, although it is not clear if it can be used to compromise cryptographic material without the assistance of a malicious agent on the machine in question [Asonov and Agrawal 2004; LeMay and Tan 2006]. We borrow the idea of exploiting side channels from security research. Favoring simplicity, we investigate the merits of using low-frequency power traces as the side channel.

3. GENERAL DESIGN OF TELEDIAGNOSTIC POWERTRACER

Our objective is to perform remote gross-level damage assessment on unresponsive nodes in high-end sensing systems, such as what may have caused them to stop

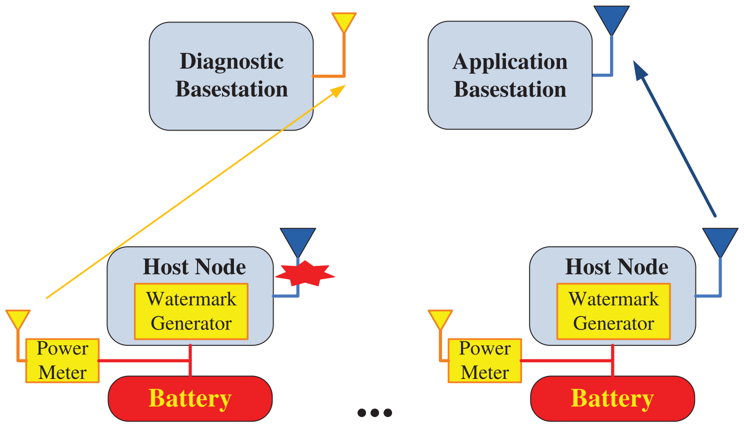


Fig. 1. A power-based telediagnostic system as an "add-on" to a sensing system. The watermark generator is needed only when the active sample scheme is used.

communicating and what the status of the applications might be. Our results show that a high-end sensing node does indeed have a different low-frequency power consumption signature in different normal and abnormal states, leading to the design of a telediagnostic powertracer. Our design follows two main objectives:

Diagnostic Subsystem Generality: It should operate as an external tool and should be generally applicable to most high-end host systems. Since power consumption is a very general channel that is available on every system that needs power to perform, in this work, we study the possibility of using a power-based telediagnostic tool. Moreover, such a tool should require as few changes as possible to the host system. The hardware installation of the tool is simply plugging in a power meter, while one should design its software to be as simple as possible to make it easier to migrate on different host systems.

Diagnostic Subsystem Efficiency: A diagnostic subsystem should not cost, in either components or energy, a sizable fraction of original sensing node cost. Although high sampling frequencies can increase the accuracy of system state estimations, high-frequency Analog-to-Digital Converters (ADCs) are more expensive than low-frequency ADCs, and more energy is required to transmit and process their measurements. Therefore, we aim to devise diagnostic algorithms that can accurately identify node states using meters with low sampling rates.

Following these objectives, we design the telediagnostic powertracer as in Figure 1. It includes a low-cost power meter, one per sensor node, that periodically samples the current and voltage of its host node. These meters are wirelessly connected via direct, low-bandwidth links to a low-end base station, called the *diagnostic base station*, that collects the power traces and runs various diagnostic algorithms to infer the status of unresponsive nodes based on the collected power traces.

Specifically, there are two ways to sample the node's power consumption. One is passive sampling, which just passively samples the host node without requiring any support from the host. The other is active watermarking, which places a *watermark generator* (as shown in Figure 1) into each host to actively inject unique power patterns (watermarks) into the power consumption traces based on the current system status. These two schemes have their own pros and cons, so we investigate both in the article.

Power watermarks can be generated by manipulating the power consumption rate of its host node. For instance, it can alter the CPU load, generate memory or disk operations, or even toggle some peripheral devices. However, for the design objective of generality, the means (e.g., hardware device) chosen for manipulating power consumption should not be specific to a particular sensing system. More importantly, it is desirable for the watermark generator to manipulate the power consumption of a key piece of hardware (e.g., the CPU rather than peripheral devices) that is necessary for application execution as well. This ensures that failure of the watermark generator due to failure of this piece of hardware can still be correctly interpreted as an indication of application failure, since the application would then fail as well.

In principle, the availability of independent low-bandwidth wireless communication on the power meter can also be exploited by the monitored node to send a distress signal if the node's main radio fails. We do not exploit it in this article for two reasons. First, if node failure is brought about by a system crash or energy depletion, having an extra radio on the failed node would not help as the node would not be able to use it anyway. Second, and more importantly, it requires connecting the monitored system to the radio of the diagnostic system and developing a driver for such a radio on the hardware of the monitored system, both of which make this solution deeply coupled with the monitored system and thus violates our design goal of diagnostic subsystem generality.

One should understand that adding a diagnostic subsystem to a remotely deployed sensor network necessarily increases the number of components that are deployed in the field and hence increases the odds of component failure. The simplicity of the meter, however, where it merely measures and communicates power samples at a low rate, makes it likely that the more complex monitored sensing application will fail first. For example, residential power meters are presently contemplated that should operate uninterrupted for approximately 10 years. Failure of diagnosis, from a user's perspective, occurs only when both systems have failed, which has a lower probability than failure of the monitored system alone.

Finally, we emphasize that the diagnostic system described in this article is intended to help a remote operator determine the status of deployed, *unresponsive* nodes. Nodes that remain responsive can, in general, use other solutions for health monitoring. For example, they can run a local diagnostic routine and report its outcome periodically. Such solutions have been discussed at length in previous literature and hence are not a part of the contribution of the work presented in this article.

4. IMPLEMENTATION ON SOLARSTORE TESTBED

We deployed the telediagnostic subsystem in conjunction with an outdoor solar-powered testbed called SolarStore [Yang et al. 2009], which aims at high-end data acquisition for environmental monitoring applications. Local storage is available on each node to cache the sensory data when the base station is disconnected. This testbed is a good example of high-end, multiapplication sensing systems that our tool aims to troubleshoot.

4.1. Hardware

The testbed consists of nine sensor nodes with sensing, communication, and storage capabilities. Figure 2 shows an outside view of a node, as well as the components inside the node enclosure. Each node is powered by one 12-volt deep cycle battery (DEKA 8G31). For the purpose of perpetual operation, the battery is charged by a set of two solar panels. An Asus EEE PC is used as the computing device of each node. It has a 900 MHz Intel Celeron CPU, 1GB DDR2 RAM, and 16GB solid-state disk and runs a Linux operating system. In addition, on each node, a Linksys WRT54GL router is configured to the ad hoc mode to support high-bandwidth communication between nodes. Moreover, one of its two antennas is extended outside of the node enclosure for better signal reception, and thus may be damaged by lightning, rain, or wild animals. The router and PC inside the enclosure are also subject to all sorts of failures caused by

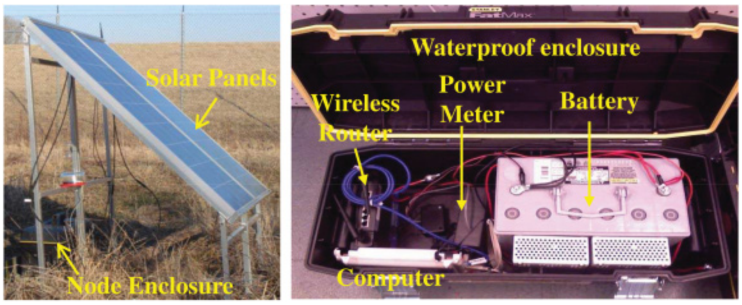


Fig. 2. Outside and inside look of a node in the solar-powered sensor network testbed.

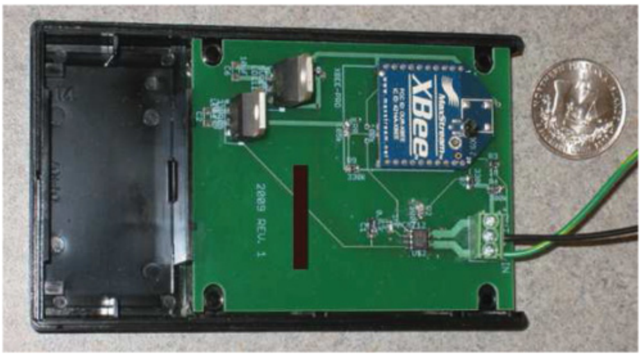


Fig. 3. Our power meter with a wireless radio.

natural events (e.g., overheating or water damage) in the harsh outdoor environment. In order to avoid over discharging the battery, a controller (Xantrex C35) watches the battery voltage, disconnects the power of the host node when the battery voltage is below the threshold 10.5 volt, and switches it on when the battery voltage is restored above 12.5 volt. A wake-up timer on the EEE PC motherboard is used to power on the PC automatically when the power is back.

On each sensor node, separately from the previous components, the telediagnostic powertracer system is installed. Its power meter intercepts the connection between the energy subsystem and the computing subsystem of each node and reports readings back to a diagnostic base station. As shown in Figure 3, the meter is composed of two circuit boards. The first is a custom design that incorporates an Allegro ACS712 hall effect current sensor capable of measuring current consumption of up to 5 amps and an op-amp-based difference amplifier to enhance the precision of the meter. The output from the amplifier is connected to an ADC on an off-the-shelf Digi XBee radio, which is itself the second circuit board. The XBee radio we selected is the basic XBee that uses the 802.15.4 protocol and has a 1mW maximum transmit power. The base station has a matching XBee radio to receive measurements. If the power meters cannot reach the diagnostic base station in one hop, their XBee radios can be configured to use the built-in firmware modules to form a mesh network [Digi XBee]. Therefore, the transmission of power consumption samples to the diagnostic base station is solely through the network formed by the power meters and is independent from the monitored subsystem. The meter samples at 1kHz and averages the measurements in batches to compensate for the noise in the measurements. For example, the batch size is 220 by default, which entails an effective sampling rate of 4.5Hz. By varying the size of sample batches, we

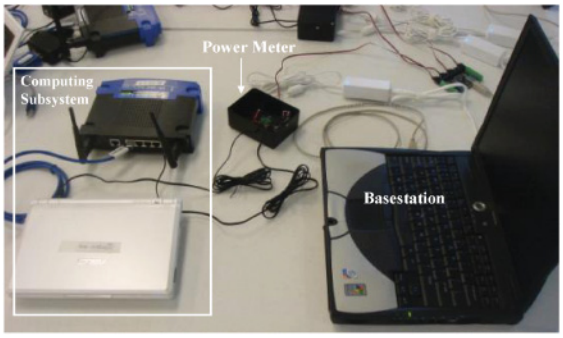


Fig. 4. An indoor node with a power meter measuring its power consumption.

can achieve different sampling rates. The entire meter consumes about 871mW, which is only 6% of the host node. The total cost for the parts in each meter is around $59.41, which is about 3% of the cost of the host node (about $2,000).

Per our design guidelines, the diagnostic subsystem must be independent from the monitored subsystem. Thus, it is ideal that the energy needed for the power meter itself comes from an independent battery. This is needed to reduce the chances of correlated failures such as energy depletion that causes both the host node and its power meter to fail. However, in our solar-powered testbed, we connect both the meter and the monitored system to the same battery, charged by the solar cell, leveraging the fact that the power meter needs a lower voltage to operate, compared to the monitored system. For example, in our case, the lowest voltage at which the power meter operates reliably is 6.2 volt, whereas the voltage threshold for the monitored system is 10.5 volt. We connect the meter directly to the battery, bypassing the discharge controller. In this way, the meter continuously reports readings even after the host node is shut down due to energy depletion.

For the sake of this experimental study, we also set up an indoor testbed, where the computing subsystem of each node is a clone of the one in the outdoor testbed, while the energy subsystem is replaced by power supplies that can be connected or disconnected using X10 modules to emulate the action of the load controller of a solar cell. Figure 4 shows an indoor node with a power meter measuring its power consumption. Due to the difficulties of accessing the remote outdoor testbed, we always test any newly proposed scheme or application before we deploy it on the outdoor testbed. In this case, we tested and validated our proposed telediagnostic powertracer first on the indoor testbed and then deployed it on the outdoor testbed (i.e., every node of the outdoor testbed is connected to one telediagnostic power meter). The experiments are conducted based on the power consumption traces collected from the outdoor testbed.

4.2. Diagnosis Objectives

The objective of our diagnostic subsystem is to determine the cause of connection failures that happen to deployed host nodes and the current status of the applications that are supposed to be running on those nodes. Namely, hardware and software failures that do not impair the connectivity of a node to the outside world are not the focus of our subsystem. Given a network connection to the node, one can always run all sorts of advanced diagnostic and debugging tools to achieve more fine-grained system diagnosis.

Typically, the common failure causes in remote deployments are known from past experience. For example, our initial experience with SolarStore suggests that the most common failure cause is energy depletion. In fact, energy depletion is not an actual “failure” state on this solar-powered platform. No human intervention is needed as a

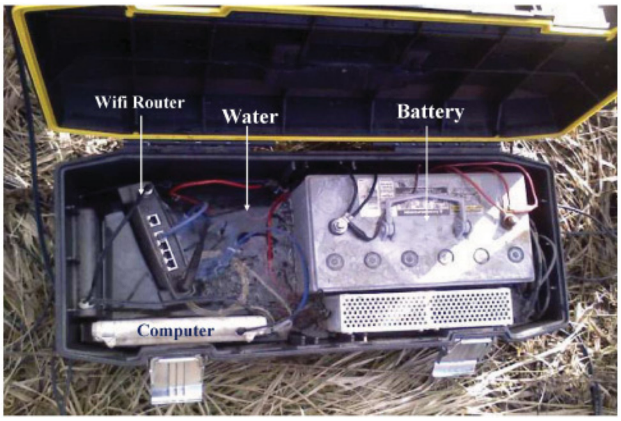


Fig. 5. One node drowned in the flood by heavy rains.

node will eventually come back to life when its energy is restored. However, energy depletion also makes nodes unresponsive, so we need to tell it from other failure cases when it occurs.

Other causes of unresponsive behavior of nodes include operating system crash, infinite loops involving a node reboot, and short circuit due to water damage (shown in Figure 5). Under those failures, all applications on the failed node would be dead and stop collecting new data. Therefore, it is desirable to make a field trip as soon as possible to repair the node to minimize the data loss and contain the hardware damage.

A node also becomes silent if its WiFi router fails or the router is still on but the outside antenna is damaged (e.g., by an animal). These two types of failures may not affect the ability of applications to sample and cache data locally. In such cases, there may be no need for immediate intervention, so we can batch multiple such repairs into one trip to save the cost. Therefore, in addition to distinguishing the hardware failures noted earlier, we also target identifying the states of applications on unresponsive nodes (i.e., whether or not applications are still functioning correctly and storing data). There are two applications that run on our current deployment performing acoustic and video recording of local wildlife near a forest, namely, (*App-I*) collection of bird vocalizations in CD-quality sound, and (*App-II*) detection of predators of bird eggs using infrared cameras.

To test the accuracy of the diagnostic techniques, we therefore set, as a benchmark, the goal of distinguishing among the 12 failure states shown in Figure 6. We take these cases as a proof-of-concept portfolio of failures that we purport to distinguish. In general, as more failures are observed during deployment and their power traces recorded, they can be added to the portfolio.

Note that, given the two applications, there are four application subcases that need to be distinguished. In general, if there are n applications that may fail, there are 2^n possible application failure states. Therefore, diagnosis based on only passively measured power consumption is not likely to scale to a large number of applications. However, in a sensor network context, it is still useful for many cases of dedicated deployments that have very specific purposes and do not run a wide range of different applications concurrently. For example, only two applications are running in our current deployment.

To handle the cases with multiple concurrently running applications, we propose another scheme called power watermarking. We place a watermark generator into each host node that actively injects unique power signatures (watermarks) into the

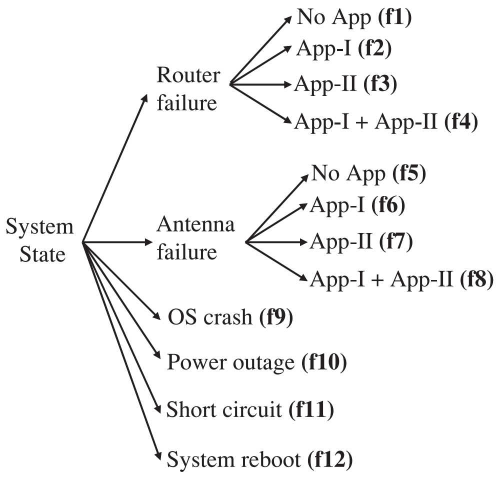


Fig. 6. Possible failure states in our system that cause a node to become unresponsive. App-I is the application responsible for sensing sound using microphone. App-II is the application responsible for recording images.

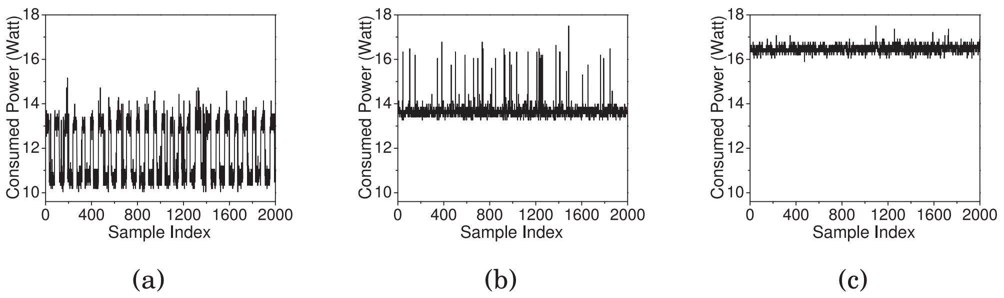


Fig. 7. Power traces of a sensor node in three states: (a) router fails and application is sampling sound, (b) antenna fails and application crashes, and (c) OS crash.

consumed power traces according to the current system state. At the remote diagnostic base station, a watermark detector infers the node state by identifying the embedded power watermarks. Since we are only interested in binary per-application states (i.e., running or failed), even with a large number of applications, few bits are enough to code all the possible states of interest. In fact, a simple coding and modulation technique is sufficient for watermark generation, which makes it easier to be implemented on sensor nodes and ported to different systems.

Passive sampling has limitations in scalability but is truly nonintrusive to the host system, while power watermarking is more scalable to the number of applications but indeed needs to place a simple software module into the host system. In the following, we will present and compare the two schemes in detail.

5. DIAGNOSTICS BASED ON PASSIVE SAMPLING

This section presents an exploration of different design parameters for diagnosing different node failure states from passively measured power consumption traces. The goal is to understand the tradeoffs between algorithm complexity and diagnostic accuracy.

As demonstrated in Figure 7, we observe obvious differences in the power measurements of our solar power sensor node under different execution scenarios. This suggests the possibility of using a pretrained classifier to identify the current system state based purely on the measured power consumption. A classifier is an algorithm that identifies to which of a set of classes a new observation belongs, on the basis of a training set of data containing observations whose class membership is known. Specifically, for each system state of interest, we collect the consumed power traces in advance and use the collected labeled data to train a classifier.

When a failure occurs, diagnosing it within a few minutes is considered good enough. We thus use the classifier to determine the state of the system every τ minutes, which we call the *detection period*. When invoked, the classifier uses a window of size δ samples to determine the system state. The following subsections explore the space of possible power trace analysis algorithms from simplest to more complicated that can be used for classification in order of increasing complexity, as well as hybrid schemes that avoid their individual limitations.

To test the accuracy of the classification schemes, we collected 80,000 samples at the rate of 4.5 samples per second for each of the system states shown in Figure 6. We used the first 40,000 samples to extract the static features and the remaining 40,000 samples to test the accuracy of the model. The 40,000 testing samples are divided into traces of length δ, each of which is fed into the classifier to infer the class label. The diagnostic accuracy for a failure case is calculated as the ratio of the number of times that the correct label is identified over the total number of runs.

5.1. Classification Based on Static Power Consumption Features

A simple way to characterize the power consumption pattern is to use the parameters of the probability distribution of the sampled power time series. For the time series of state k, we use the mean, μ_k, and the standard deviation, σ_k. In other words, rather than modeling how exactly the values in the time series change, we lump such changes into a feature vector (μ_k, σ_k) for each state k.

Assume the series of power consumption measurements for system state k is given by the power samples $x_1, x_2, x_3, \ldots, x_n$. The mean, μ_k, and standard deviation, σ_k, over a training window of n samples in state k are calculated as follows:

$$\mu_k = \left(\sum_{i=1}^{n} x_i\right) / n \tag{1}$$

$$\sigma_k = \sqrt{\left(\sum_{i=1}^{n} (x_i - \mu_k)^2\right) / n}. \tag{2}$$

After training, each state is represented using a (μ_k, σ_k) pair and stored for later reference. At runtime, when a node becomes unresponsive, our diagnostic system is launched. The power trace data recorded for the unresponsive node are inspected. The vector (μ, σ) is computed over a sliding time window (e.g., 30 minutes) for the observed data and matched to the nearest known failure state by finding k such that the Euclidean distance between feature vectors is minimized: $min_k(\sqrt{(\mu_k - \mu)^2 + (\sigma_k - \sigma)^2})$. The observed state is thus classified as failure state k.

To investigate the effect of window size on accuracy, we use window sizes of 1, 5, 10, 15, 20, and 30 minutes, respectively. The diagnostic accuracy for different window sizes is given in Figure 8. Our experiments show that the improvement in classification accuracy with increased window size diminishes after a size of approximately 10 minutes. We show the confusion matrix for the classifier based on static features, for a window size of 20 minutes, in Table I. A confusion matrix, also known as a contingency table or

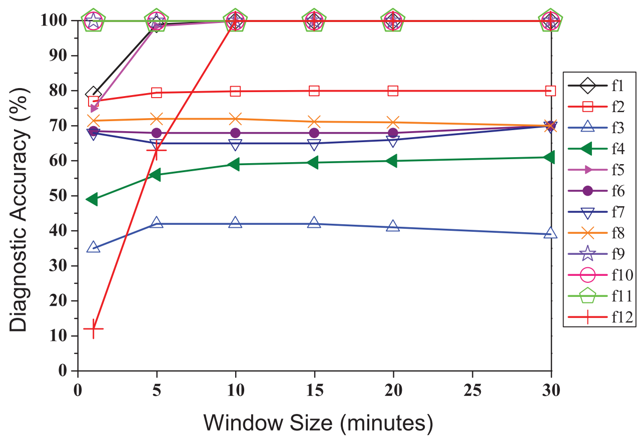


Fig. 8. Effect of the window size on classification accuracy for static feature-based classification.

Table I. Confusion Matrix for Classification Based on Static Features (Window Size = 30 Minutes)

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12
f1	1											
f2		.8	.2									
f3			.4	.4	.2							
f4		.2		.6	.2							
f5					1							
f6			.3			.7						
f7							.7	.3				
f8								.7				.3
f9									1			
f10										1		
f11											1	
f12												1

an error matrix, visualizes the performance of a classification algorithm. The matrix makes it easy to see if the algorithm is confusing two classes (i.e., mislabeling one as another). Specifically, a cell (i,j) in the confusion matrix represents the probability of system state i (the row) to be classified as system state j (the column). A perfect classification algorithm will have a confusion matrix with all 1s on the main diagonal. State f3 (router failure but camera is recording) is confused with state f4 (router failure but camera and microphone are recording) and there is a high chance that f7 (antenna failure but camera is recording) is confused with state f6 (antenna failure but microphone is recording) or state f8 (antenna failure but camera and microphone are recording). It is worth noting that these misclassifications do not affect the ability to recognize which component failed. However, they err in inferring which application is running. In addition, state f12 (system reboot) is confused with antenna failure (states f7 and f8). In the rest of this section, we seek to remedy these inaccuracies using more involved classifiers.

5.2. Classification by Capturing Power Consumption Dynamics

A disadvantage of static features is that they do not capture the dynamics of the sampled power time series. Fortunately, analyzing dynamic time-series data and identifying time-varying patterns are very mature research areas in the machine-learning and data mining communities. Several techniques that vary in complexity, accuracy, and efficiency can be borrowed from the literature [Lin et al. 2003; Patel et al. 2002; Shasha and Zhu 2004; Rafiei and Mendelzon 1997]. We explore the use of Markov models.

A Markov model is very commonly used to model a stochastic process, where the conditional probability distribution of future states of the process depends only on the present state; that is, given the present, the future does not depend on the past. A Markov model consists of system states and probabilities of state transitions that best describe a particular time series. In this case, we use a simplified version of Markov models, where the states are predetermined. To build a Markov model for the power trace of a given failure scenario, we process the power trace corresponding to the failure scenario using the following three stages:

Preprocessing: As the meter is somewhat noisy, we always perform a outlier filtering step. For collected data for each system state k, we first calculate the mean (μ_k) and standard deviation (σ_k) and discard any value that is outside the range of $[\mu_k - 5 * \sigma_k, \mu_k + 5 * \sigma_k]$ as outlier. Next, we can perform an optional step where we may perform smoothing or normalization to input data based on system configuration (will be elaborated in Section 5.2.4).

Discretization: Power consumption produces continuous-valued time-series data, which are hard to analyze as the possible values for continuous data are infinite. To address this issue, we discretize the power measurements, reducing the numeric values of power samples to a small finite number of symbols. For example, 0–1 watt can be called an “a,” and 1–2 watt called a “b,” and so forth. These symbols represent measured power consumption levels, henceforth called *power states*. The entire power trace is therefore converted to a string of symbols. Besides this static discretization method, we also examine a dynamic method based on clustering in Section 5.2.5.

Computing Transition Probability: We build a state transition diagram that expresses which states are followed by which other states. For example, a substring “ab” in the power trace string represents a state transition from “a” to “b.” By observing how often “ab” occurs in the string, we can determine the probability of state transition *ab*. For instance, in string “aaabbcd,” there are a total of six transitions (e.g., the first “a” is followed by the second “a,” the second “a” is followed by the third “a,” the third “a” is followed by the “b,” and so on). Hence, the transition probability p(aa) = 2/6 (i.e., there are two transitions from state “a” to “a”), and p(cb) = 1/6. Any trace can be summarized by a two-dimensional probability matrix that states the probabilities of state transitions from any power state i to any power state j in the trace. The aforementioned state transition diagram is also known as a Markov model. For each system state, we build a model that represents that state.

The models built earlier are subsequently used for classifying system states during runtime diagnosis. When a node becomes unresponsive, we use the δ samples that have just been reported by the power meter. Next, by using the transition matrix of each system state, we calculate the probability that the observed sequence of samples is generated under this model. Specifically, this probability is the product of the transition probability $p(x_i, x_{i+1})$ for all $i = 1, \ldots \delta - 1$, where x_i is the i^{th} sample in the trace. The system state that has the highest probability of generating the observed sequence is returned as the classification result.

In the following, we conduct a series of experiments to answer the following questions regarding the classifier design. For brevity, we refer to the Markov model as MM.

—What is a sufficient number of model states to use?
—What is an acceptable sampling frequency of the power trace?

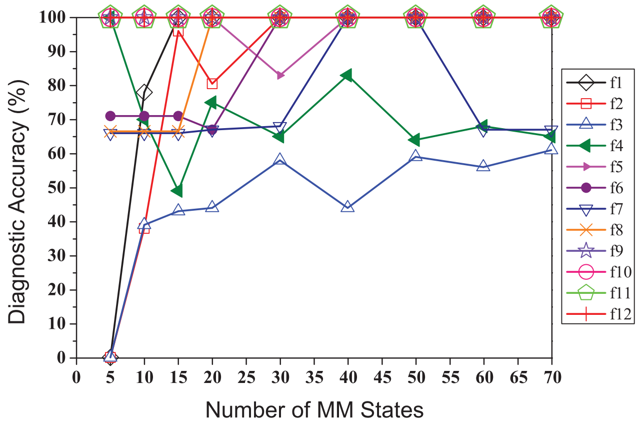


Fig. 9. Effect of number of MM states on classification accuracy (window size = 30 minutes, data preprocessing used: outlier filtering).

—What is the effect of the data window size used for diagnosis?
—What are the pros and cons of different data preprocessing techniques?
—What are the pros and cons of improved data discretization techniques?

5.2.1. Effect of MM Size. To see the effect of the number of MM states on classifier accuracy, we varied the number of states as 5, 10, 15, 20, 30, 40, 50, 60, and 70 and tested the MM with a window size of 30 minutes. The effect of the number of states on accuracy is given in Figure 9. For this experiment, we trained the MM on raw data (after noise reduction). As we can see from Figure 9, the accuracy increases with number of states and becomes stable after the number of states reaches 50. More interestingly, the figure highlights the fact that increasing the number of MM states far beyond that value is a "bad" idea as well, because accuracy starts to drop if the number of states becomes "too large." This is because with too many states, the amount of data available in the used window might become insufficient to accurately determine all the state transition probability. In the rest of the article, we use 50 states for MMs unless we specify otherwise.

5.2.2. Effect of Sampling Frequency. Since reducing the sampling interval increases energy consumption, we evaluate the effect of accuracy of the MM classifier with various sampling intervals. We train the MM classifier at the sampling interval of 222ms, 444ms, 888ms, 1,776ms, 3,552ms, 7,104ms, 14,208ms, 28,416ms, and 56,832ms, respectively. The lower sampling intervals were obtained from the same data by downsampling the original time series (i.e., selecting one every N original samples for $N = 1, 2, 4, \ldots, 256$). We present the effect of the sampling interval on accuracy in Figure 10. As we can see, if the sampling interval is reduced to 444ms, accuracy starts to drop, and after that point the accuracy decreases monotonically due to the loss of information on finer-grained dynamics.

5.2.3. Effect of Window Size. To test the effect of window size on accuracy, we trained the MM on the original data (after outliers are removed) with 50 states and tested its accuracy with window sizes of 1, 5, 10, 15, 20, and 30 minutes, respectively. Regardless of window size, we considered all windows shifted by 1-minute intervals. We show the effect of varying window size on accuracy in Figure 11. In general, increasing window size helps increase the overall accuracy. The amount of improvement varies between different failure states.

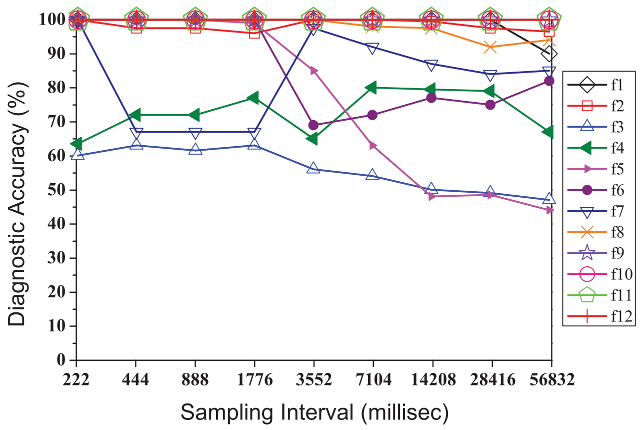


Fig. 10. Effect of sampling rate on the classification accuracy of MM (window size = 30 minutes, number of states = 50, data preprocessing used: outlier filtering).

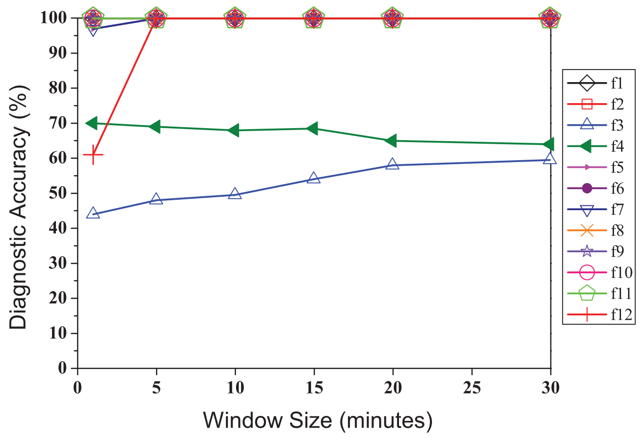


Fig. 11. Effect of window size on the classification accuracy of MM (number of states = 50, data preprocessing used: outlier filtering).

We also show the confusion matrix to determine the probability of misclassification and illustrate which states are most likely to get confused. Table II gives the confusion matrix for a window size of 30 minutes. The performance of the MM with a 30-minute window size is significantly better than the static feature-based classification scheme. We have 100% accuracy for all the states except f3 and f4. f3 occasionally gets misclassified as f4 and vice versa. It is worth noting that these misclassifications do not affect the ability to recognize which component failed. However, they err in inferring which application is running.

5.2.4. Effect of Data Preprocessing. In this section, we consider techniques for data preprocessing that have the potential to eliminate extraneous information from the sampled power signal, allowing us to focus on essential features. For example, the status of a CPU fan ("on" or "off") can affect power consumption by adding or subtracting a constant offset. An MM trained with the fan on may lead to misclassifications if the fan is turned off. To address this problem, we resort to data normalization prior to discretization. We explore two alternatives for normalization, namely, (a) z-score-based

Table II. Confusion Matrix for Classification Using MM (Window Size = 30 Minutes, Number of States = 50, Normalization Used: Outlier Filtering)

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12
f1	1											
f2		1										
f3			.6	.4								
f4			.36	.64								
f5					1							
f6						1						
f7							1					
f8								1				
f9									1			
f10										1		
f11											1	
f12												1

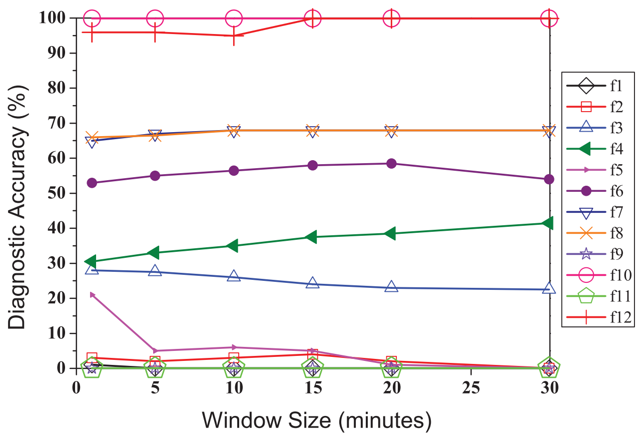


Fig. 12. Effect of window size on the classification accuracy of MM based on z-score (number of states = 50, data preprocessing used: outlier filtering, z-score normalization).

normalization and (b) normalization based on relative difference. We describe each of these techniques next.

Normalization based on z-score: To normalize the data using the z-score, we use the following formula:

$$x_i' = (x_i - \mu_k)/\sigma_k, \tag{3}$$

where x_i is the raw data, μ_k is the mean, and σ_k is the standard deviation for the training data for a particular system state k. Intuitively, the z-score represents the distance between the raw score and the population mean in units of the standard deviation. The z-score is negative when the raw score is below the mean and positive when it is above. It is a very common technique for data normalization in the data mining literature. In Figure 12, we present the impact of varying window size on accuracy of an MM trained based on z-score data. In Table III, we present the confusion matrix. The window size is 30 minutes. It turns out that the accuracy of MMs using z-score normalization is not encouraging and cannot be used for diagnosis effectively.

Normalization based on difference signal: As an alternative, we normalize the data using a simpler scheme that uses the difference signal obtained from the following

Table III. Confusion Matrix for Classification Using MM Based on z-Score (Window Size = 30 Minutes, Number of States = 50, Normalization Used: Outlier Filtering, z-Score)

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12
f1										1		
f2			.3			.5		.1			0.1	
f3			.2	.1			.4	.3				
f4				.4			.3	.2		.1		
f5										1		
f6						.54		.12		.34		
f7				.13			.67	.2				
f8							.3	.7				
f9										1		
f10										1		
f11										1		
f12												1

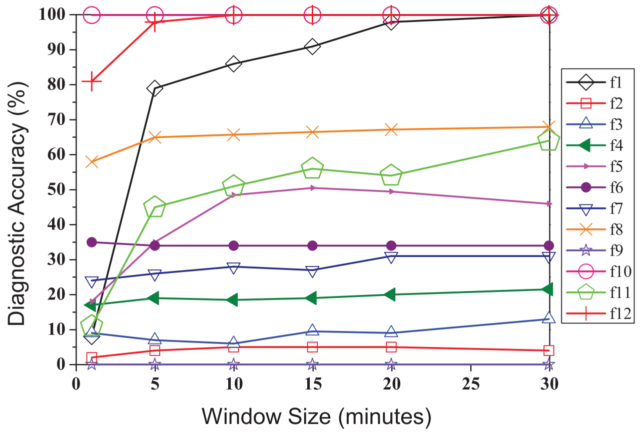


Fig. 13. Effect of window size on the classification accuracy of MM based on signal difference (number of states = 50, data preprocessing used: outlier filtering, difference between consecutive signal).

formula:

$$x'_i = x_i - x_{i-1}, \tag{4}$$

where x_i is the raw data. Note that this scheme is similar to obtaining the derivative of the sampled signal. Hence, any constant bias (such as the power consumption of an irrelevant fan) is eliminated due to differentiation. In Figure 13, we present the impact of window size on the accuracy of the trained MM. As we can see from Figure 13, the window size has a significant impact on MM classifier accuracy. In Table IV, we present the confusion matrix for a window size of 30 minutes. It has a better classification accuracy compared to the z-score normalization technique, but as good as the MM based on original data. The intuition behind such poor performance when normalization is used is that it is because the absolute power consumption level does play an important role in identifying what is running and what is not. Data normalization causes information loss.

5.2.5. Discretization by Clustering. Discretization of the power signal is an important step toward computing the MM. In all previous sections, we used a simple discretization technique that simply cut the range of input data into uniform bins and assigned them names. In reality, the power measured for a system in different states may

Table IV. Confusion Matrix for Classification Based on MM Based on Signal Difference (Window Size = 30 Minutes, Number of States =50, Data Preprocessing used: Outlier Filtering, Difference between Consecutive Signal)

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12
f1	1											
f2	.2		.1	.5				.1			.1	
f3			.1	.6		.05	.1	.1			.05	
f4			.1	.2		.2		.3		.1	.1	
f5					.5						.5	
f6				.3	.2	.3					.2	
f7			.1	.5			.3	.1				
f8				.2		.1		.7				
f9										1		
f10										1		
f11					.4						.6	
f12												1

Table V. Confusion Matrix for MM with Clustering (Window Size = 30 Minutes, Number of Clusters = 50, Normalization Used: Outlier Filtering)

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12
f1	1											
f2		1										
f3		.01	.94	.05								
f4			.14	.86								
f5					1							
f6						1						
f7							1					
f8								1				
f9									1			
f10										1		
f11											1	
f12												1

not necessarily be uniformly distributed across its range. The discretization algorithm introduced earlier does not capture nor take advantage of this knowledge. For example, it may put different clusters of power measurements into the same bin. Conversely, there may be bins into which no measurements fall. The first case causes information loss, while the latter produces unnecessary states for the MM.

In this section, instead of using even ranges, we employ the hierarchical clustering technique [Johnson 1967] to identify representative power levels and use those representative levels as anchor points to discretize the time-series input. Hierarchical clustering is a common technique used in statistical data analysis. It uses a bottom-up approach, where the scheme starts with each data point as a cluster and repeatedly merges each two closest clusters to one until the desired number of clusters is left. The distance between two clusters is given by the average distance between points belonging to the different clusters. Table V shows the confusion matrix for the MM with the clustering scheme with 50 clusters. As the results show, the MM classifiers with clustering perform better than earlier schemes. This result is encouraging since with the same number of states, the MM classifier with clustering performs better. Figure 14 shows the detailed accuracy of each failure state of the system versus the window size.

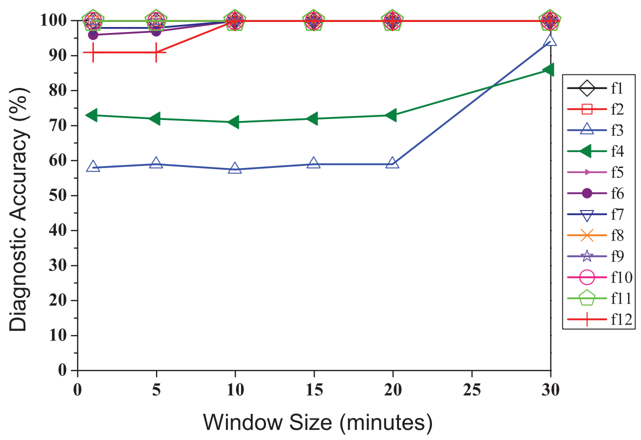


Fig. 14. Effect of window size on the classification accuracy of MM with clustering (number of clusters = 50, data preprocessing used: outlier filtering).

5.3. A Hybrid Scheme

The previous subsections indicated no clear winner in terms of the ability to diagnose system states, although some classifiers were better at some states. This leads to the idea of developing a hybrid scheme. Ensemble classifier approaches, studied in machine learning, can greatly improve performance of a classification system [Asker and Maclin 1997; Giacinto et al. 2000; Bauer and Kohavi 1999] by assembling a good classifier from a group of weaker ones, called subclassifiers. In this work, we employ such a method to achieve nearly perfect accuracy. There are two decisions in designing an ensemble classifier. The first design decision is what the different subclassifiers should be. In our case, the subclassifiers are simply the imperfect classifiers we presented earlier, based on static features and MMs. The second choice is how to combine classification results from these subclassifiers. We implement two hybrid schemes that differ in how they combine subclassifier results. As we explain later, one is based on the implicit assumption that misclassifications of different subclassifiers are fully correlated. The other is based on the assumption that they are independent.

More specifically, our hybrid classifier uses the following schemes for subclassifiers: (1) static feature-based classification, (2) MM-based classification without data preprocessing, (3) MM-based classification based on z-scores, (4) MM-based classification based on the difference signals, and (5) MM-based classification using clustering, without data preprocessing (which was shown to be the best MM+clustering scheme). The confusion matrices for these schemes were previously shown in Table I, Table II, Table III, Table IV, and Table V. When a failure occurs, each subclassifier returns a result, indicating its diagnosis. Say, subclassifier j returned state f_i as its answer. The confidence that f_i is really the correct system state can be computed from the i^{th} column in the subclassifier's confusion matrix by dividing the i^{th} diagonal entry (corresponding to a correct classification of f_i) by the sum of all i^{th} column entries (corresponding to all the cases where the classifier returned f_i, correctly or not). Let us call it the confidence of classifier j in its answer, denoted $Conf(j)$.

Two flavors of the hybrid classifier are described here. The two algorithms differ in how they combine the results based on confidence.

—*Correlated classifier scheme:* In our first scheme, given different answers by different subclassifiers, the subclassifier j with the highest confidence, $Conf(j)$, in its answer wins. The scheme implicitly assumes that if two or more subclassifiers returned the

Table VI. Hybrid Classifier I: Correlated Assumption

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12
f1	1											
f2		1										
f3			.63	.35				.02				
f4				.95				.05				
f5					1							
f6						1						
f7							.9	.1				
f8							.04	.96				
f9									1			
f10										1		
f11											1	
f12												1

Table VII. Hybrid Classifier II: Independent Assumption

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12
f1	1											
f2		1										
f3			.67	.33								
f4				1								
f5					1							
f6						1						
f7							.9	.1				
f8							.04	.96				
f9									1			
f10										1		
f11											1	
f12												1

same result, we do not know this result with any higher confidence than if it was returned only by the one with the highest confidence among this group. This, in turn, assumes that classifier results are fully correlated; whatever causes one to err causes the other to err too, hence not offering extra information when both return the same result. The confusion matrix of this hybrid classifier, computed for 30 states and a 20-minute window, is shown in Table VI. Looking at the diagonal entries, the table demonstrates that this scheme is somewhat better overall, in that the average of the diagonal is higher than that for the used subclassifiers.

—*Independent classifier scheme:* In our second scheme, if a set S_i of subclassifiers agree on the same answer, f_i, a combined confidence is computed in f_i using the formula $CombinedConf(f_i) = 1 - \Pi_{j \in S_i}(1 - Conf(j))$. In other words, when answers coincide, the scheme combines them as if they are independent and hence the previous formula computes a higher confidence in the result. The result f_i with the highest $CombinedConf(f_i)$ is then chosen. The confusion matrix of this hybrid classifier, computed for 30 states and a 20-minute window, is shown in Table VII. The figure demonstrates that this scheme is nearly perfect.

The rationale for testing both of abovethese hybrid classifiers is to see how independent the set of features used by their subclassifiers is. Recall that these subclassifiers differed in the use of static features versus MM, use of different preprocessing techniques, and use of simple discretization versus clustering. It appears, based on the winning hybrid classifier, that these features were closer to being independent than correlated.

6. DIAGNOSTICS BASED ON POWER WATERMARKING

As shown in the results of Section 5, the classifier based on passive sampling has a problem in distinguishing some application states (e.g., f3, f7, and f8 in Table VII). As the number of applications increases, the number of possible system states grows exponentially, making them even more difficult to classify. Although one can expect that the number of applications on dedicated sensing systems is small, to improve scalability, we propose another scheme called power watermarking, which offers a solution by artificially shaping the power trace by adding predefined power watermarking. In the following subsections, we present how watermarks are generated on the host node and how they are detected from the noisy power traces that also contain the power variations attributed to all sorts of operations on the host, and evaluate this scheme on our SolarStore testbed.

6.1. Watermark Generator

The mission of the watermark generator is to inject power watermarks into consumed power traces to carry system state information. Its coupling with the host node calls for a close look at the original power consumption trace of the host. Figure 7(a) shows power measurements of one node, on which App-I is running. App-I continuously samples a microphone and buffers the acoustic samples in the memory. When a predefined size of data has been collected, it processes the raw data into a standard format and then stores it in the local disk. A closer study discloses that those prominent power peaks take place during the raw data processing, which brings the CPU from almost idle to busy.[3]

Based on this observation, a natural way for generating power watermarks is to alter the CPU state between busy and idle artificially. This approach has the following advantages. First, as shown in Figure 7(a), the node power consumption rate jumps from 10.5 watts, when the CPU is idle, to 13.5 watts, when the CPU is busy. This difference is substantial enough for most low-end power meters to detect. Second, the CPU is a key resource that is needed by all applications. Failure of altering the CPU state indicates only one possible system state—the system crash. This ensures that failure of the watermark generation due to failure of this piece of hardware can still be correctly interpreted as an indication of application failures, since the applications would then fail as well. Third, a CPU is generally available on any sensing system, and this makes the watermark generator portable to other systems. Additionally, we observe that other operations (e.g., memory or I/Os) of the applications on the node have nondetectable power changes to the power meter. This is because those operations happen too fast (e.g., within 1ms) to be detected by low-end power meters, whose sampling rates are usually less than 10Hz. For other sensing systems where the CPU states are indistinguishable to low-end power meters (e.g., in low-end computing devices), the proposed diagnostic tool is not suitable and not recommended.

In addition to eliminating spurious hardware dependencies of the watermark generator, one should also reduce spurious dependencies on software, preventing software failures of the watermark generator from being falsely interpreted as application failures. Therefore, we propose a decentralized scheme that works as an add-on library to each application, periodically generating heartbeats with a period unique to this application. Since it shares the same address space with its application, it will fail to

[3]Note that a CPU generally switches between only two states: busy and idle. The CPU utilization as a percentage is actually an average over a time interval.

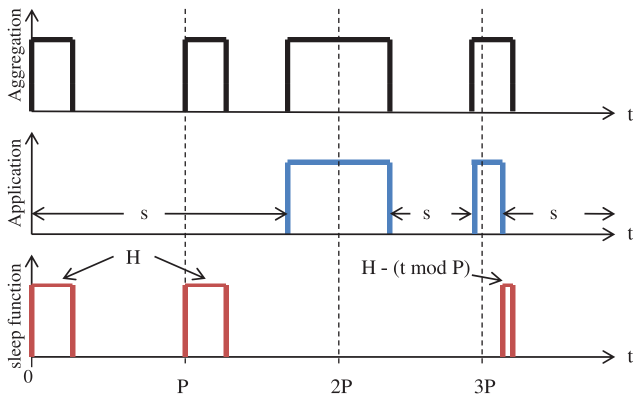


Fig. 15. The CPU usage of the `sleep` function, an application, and their aggregation.

produce heartbeats if the application crashes. Thus, the absence of particular heartbeats indicates the failure of corresponding applications.

Specifically, most applications (not only the applications in this article but also applications like environment monitoring, infrastructure monitoring, and surveillance) in remote deployment sensing systems work in a duty-cycled manner, as shown in Algorithm 1. Then within each cycle we have the application call an API provided by the watermark generator, and thus we can create heartbeats periodically to indicate whether the application is still running. This is the goal of our diagnostic system, retrieving the coarse-grained application states, particularly whether the applications are still running, so that the system operators can evaluate the urgency of repair in case of node silence. It is desired that the generation of heartbeats does not interfere with the normal operation of the application that it belongs to. Obviously, a good occasion for generating heartbeats is the time when the application is sleeping. Therefore, our watermark generator provides a new `sleep` function, which can be called by each application to perform normal sleeping action and has one side capability of generating heartbeats.

ALGORITHM 1: A typical sensing application

```
while True do
    take samples from sensors of interest;
    process samples;
    sleep(s);
end while
```

We devise the `sleep` function to generate heartbeats at one designated period chosen by each application. Of course, this designated period can only be kept approximately as the heartbeat generation also depends on how the `sleep` function is called. Specifically, let P be the designated heartbeat period of an application, and let the `sleep` function attempt to produce a heartbeat of width H at the beginning of each period. When the function is called, it checks the current time t and then decides when to run the dummy code on the CPU to generate heartbeats, when to sleep, and for how long. If the function is called with the sleeping time $s > P$, as shown in Figure 15 , at $t = 0$, it generates heartbeats at the beginning of each period within s and sleeps for the rest of the time. However, if it is called with a short sleeping time at the time when $(t \mod P) \geq H$, it misses the occasion of generating the heartbeat for the current cycle (e.g., the cycle between $2P$ and $3P$ in Figure 15), and hence simply goes to sleep for a duration of s. It may also happen that the function is called at time t when $(t \mod P) < H$, and it then only produces a partial heartbeat with a length of $H - (t \mod P)$.

However, one should note that the heartbeats observed by the power meter are actually the aggregate CPU usage of both the `sleep` function and the application. As shown in Figure 15, when the sleep function has no chance to produce the heartbeat (e.g., in $[2P, 3P]$), it entails that the application is busy using the CPU and thus makes up the wanted heartbeat. The same thing happens when only a partial heartbeat is produced by the `sleep` function. This observation inspires our design for the watermark detector, which will be presented in the next subsection.

In order to reduce overlap among heartbeats of different applications, we only use prime numbers as heartbeat periods. Note that the library interposition technique can be used to intercept the original sleep function calls from applications, so that there is no need to modify the source code of applications to have them call the redefined `sleep` function.

To ensure that the heartbeat generation will not interfere with other applications, we devise the `sleep` function such that it switches the scheduling priority of the calling process to the lowest level when it is called and switches back to the original level when it returns. On Linux, we implement this by using the scheduling system calls sched_getscheduler and sched_setscheduler. The Linux process scheduler considers two classes of priority when scheduling processes onto the CPU. One is static priority, and the other is dynamic priority. The scheduler will start running processes of dynamic priority only when there is no process of static priority in the TASK_RUNNING state. Therefore, by switching to dynamic priority, the `sleep` function will always yield the CPU to other applications that are running with static priority.

6.2. Watermark Detection

On the diagnostic base station, we continuously collect the samples from the meter attached to each monitored node and identify the embedded watermarks based on the dynamics of sampled power time series.

Given a power trace, it is first preprocessed to classify what samples represent the CPU busy state and what samples represent the idle state. A naive way is to measure in advance the power consumption rates for both the CPU states. Instead, we use a clustering-based method that is more resilient to system changes. Since the two power levels for the two CPU states occur very frequently, we observe two masses of samples in the traces, which correspond to the two CPU states. Therefore, we adopt the k-means (i.e., $k = 2$) clustering method to classify each sample into one of the two classes, $\mathbf{B}$ (CPU busy) and $\mathbf{I}$ (CPU idle). k-means [MacQueen 1967] is a method of clustering that aims to partition the observations into k clusters in which each observation belongs to the cluster with the nearest mean.

After preprocessing, a power trace $\mathcal{X}$ of length W is converted into a series of $\mathbf{B}$s and $\mathbf{I}$s. We determine if a particular pattern exists in $\mathcal{X}$ based on its similarity score that is calculated as follows. We denote a candidate heartbeat watermark by $\mathcal{Y}$ and denote the length of $\mathcal{Y}$ by $|\mathcal{Y}|$ (i.e., the period of the heartbeats). Since the candidate $\mathcal{Y}$ may not be synchronized with the trace $\mathcal{X}$, we denote their offset by θ where $1 \leq \theta \leq |\mathcal{Y}|$. The similarity between $\mathcal{X}$ and $\mathcal{Y}$ is calculated as the maximum of the aggregated sample-wise similarity under all possible θs. Formally, we have

$$sim(\mathcal{X}, \mathcal{Y}) = \max_{1 \leq \theta \leq |\mathcal{Y}|} \sum_{i=1}^{W} \mathcal{X}[i] \odot \mathcal{Y}[(i + \theta) \mod |\mathcal{Y}|], \tag{5}$$

where the sample-wise operator $\odot$ measures the similarity of two samples.

Similar schemes have long been adopted in pattern matching problems, where the operator $\odot$ is usually defined as the exclusive NOR that returns 1 if the two operands are the same, or 0 otherwise. However, we define our own $\odot$-based characteristics of our power watermark detection problem. Besides watermarks generated artificially, a power trace also contains noise from applications, back-end system processes, and the measurement process. The first two types of noise originate from the CPU usage, so they can only corrupt **I**s into **B**s, but not **B**s into **I**s. The noise from the measurement has no bias, and hence it could turn **I**s into **B**s or **B**s into **I**s.

Based on these features of noise, we define the operator $\odot$ as

$$x \odot y = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x = \mathbf{B} \text{ and } y = \mathbf{I} \\ -1 & \text{if } x = \mathbf{I} \text{ and } y = \mathbf{B} \end{cases}. \tag{6}$$

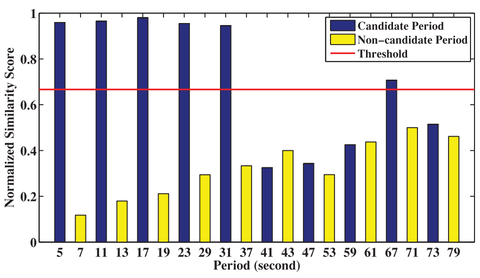
By this definition, a sample in the power trace contributes 1 to the similarity score if it matches the pattern. When a B is wanted by the pattern but an I is observed in the trace, a penalty -1 is given to the similarity measure, because **B**s have immunity to the noise and hence are not supposed to turn into **I**s. On the other hand, as an **I** is pruned to be corrupted into a **B**, there is neither reward nor penalty when $x = \mathbf{B}$ and $y = \mathbf{I}$.

Let $\mathbb{S}$ be the set of all possible patterns that could occur in $\mathcal{X}$. Each pattern in $\mathbb{S}$ is the heartbeat pattern with the designated period for each application installed on the host node. If the similarity score of a heartbeat pattern is high enough, we claim that it exists in the trace and hence the corresponding application is still running. Because heartbeats with a shorter period (i.e., a higher rate) occur more often within a time window, their absolute similarity score will be higher than that of heartbeats with a longer period. Thus, for a fair comparison, we normalize similarity scores with respect to heartbeat rates. Ideally, a symbol **B** should always be produced by either the `sleep` function or applications at the designated time. It could be corrupted into an **I** only by measurement noise. Thus, the zero score gives a very conservative lower bound for declaring existence, which tolerates a measurement error rate of 50%. On the other hand, **I**s are pruned to be turned into **B**s by applications, and hence form some patterns that could be mistaken as heartbeats, resulting in false positives. Thus, in order to obtain a right threshold on the similarity score, we utilize another set of heartbeat patterns, which are not used by any application on the host node and hence are supposed not to exist in the trace. Let μ and σ be the mean and standard deviation of the similarity scores of these nonexisting patterns. We use $\max\{0, \mu + \sigma\}$ as the threshold. With a score above this threshold, a heartbeat pattern is claimed to exist.

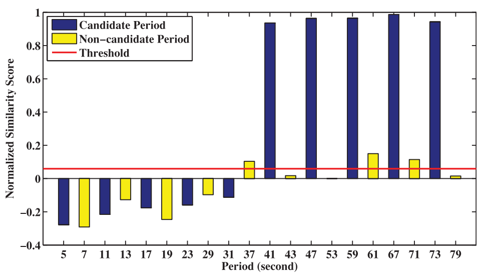
6.3. Evaluation

In this section, we study and compare how accurately the power watermarking can diagnose failures, how it performs when the number of sensing applications increases, and how efficient it is in terms of energy consumption. Since hardware failures can be accurately identified by the classifiers with passive sampling, in this section, we will focus on identifying applications states. As shown in Figure 6, the application states become unknown only in case of router failure and antenna failure. We define these two types of hardware failures as $F1$ and $F2$, respectively.

In order to test scalability, besides the two applications used earlier, we install three other new applications: (*App-III*) an application that logs the output current and voltage of the solar panel, (*App-IV*) an application that monitors the internal temperature of the node, and (*App-V*) an application that collects readings from environmental sensors (i.e., temperature, humidity, and light sensors). Furthermore, by running multiple instances of each application, we are able to test scalability to a larger state space. For



(a) F1 with all 5 applications alive



(b) F2 with all 5 applications alive

Fig. 16. Similarity scores and detection thresholds for three cases.

comparison, we also report the overall diagnostic performance of the passive-sampling-based classifier under each of $F1$ and $F2$ when more than two applications are running.

Let S be the set of prime numbers in the range of [5, 79]. We choose $S_1 = \{5, 11, 17, 23, 31\}$ and $S_2 = \{41, 47, 59, 67, 73\}$ as the two sets of prime numbers, which are used by the five applications as the heartbeat periods (in seconds) for F1 and F2, respectively. We use the remaining 10 prime numbers in S to calculate the detection thresholds. For purposes of obtaining an accurate threshold, we choose the two groups of numbers (i.e., $S_1 \cup S_2$ and $S - (S_1 \cup S_2)$) to be interleaving. The heartbeat duration H is 1 second, the default sampling rate R is 10Hz, and the default window size W is 900 seconds.

First, we study how heartbeats with designated periods can be identified by using their similarity scores. Figure 16 shows the normalized similarity scores of all the periods in S for two power traces collected under two different cases, respectively. Note that the periods in $S - (S_1 \cup S_2)$, denoted by the yellow bars in Figure 16, are never used by any application in case of F1 and F2. The mean and standard deviation of the similarity scores of these periods are used to obtain the detection thresholds, which are the red horizontal lines in the figure.

In case **(1)**, the router fails (F1) but all applications are running. This is one of the most complicated cases because we are supposed to identify all the periods in S_1. As shown in Figure 16(a), the detection threshold is 0.67. The scores of all the candidate periods in S_1 are above this threshold. Although the score of the period 67 in S_2 is also above the threshold, all other candidates in S_2 are below the threshold. The majority existence of candidates in S_1 indicates that there is a router failure, and all applications are running. In case **(2)**, the antenna is broken (F2) but all applications are running. As shown in Figure 16(b), the detection threshold is 0.06. The scores of all the periods in S_2 are above this threshold. Even though some periods, such as 37, 61, and 71, are also above this threshold, we know that these periods are never used. Therefore, we also identify this case correctly.

To investigate the effect of window size on accuracy, we run the detection module with a window size ranging from 100 seconds to 1,000 seconds. Figure 17 shows the average diagnostic accuracy for F1 and F2 with various window sizes. Overall, increasing the window size helps increase the accuracy, but this also implies a longer response time. As we are not aiming at instantaneous failure detection, we choose a reasonably large window size $W = 900$ seconds in the following experiments. Moreover, one may observe that F1 has a higher performance than F2. The only difference between these two failures, with respect to power watermarking, is that applications in F1 have smaller heartbeat periods than those in F2. This indicates that accuracy decreases with the increase of period P.

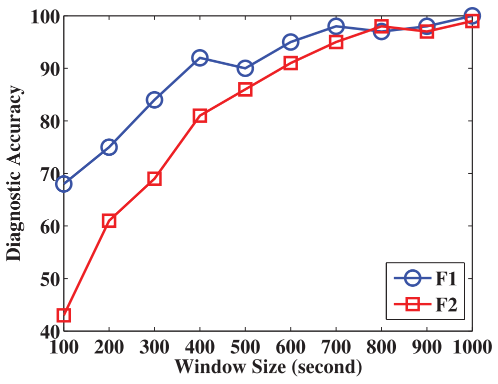


Fig. 17. Effect of window size on accuracy.

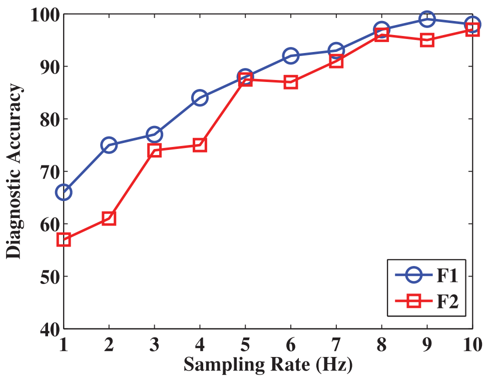


Fig. 18. Effect of sampling rate on accuracy.

As collecting samples can be expensive in terms of consumed power, we present the effect of sampling rate R on accuracy in Figure 18. As we can see, if the sampling rate is decreased, accuracy starts to drop, due to the reduction in the number of samples per window, as well as the loss of information on finer-grained dynamics. However, the achieved accuracy is still above 85% at 5Hz, which implies that a low-end power meter, efficient in both energy and cost, is sufficient for power watermark detection.

When a new application is installed, we need to assign two new prime numbers to it as its heartbeat periods under F1 and F2, respectively. In addition, we add two more neighboring prime numbers into S to help determine the detection threshold. For example, as shown previously, we use 10 prime numbers in $[3, 79]$ as the two sets of heartbeat periods for the five applications and use the other 10 prime numbers left in $[3, 79]$ to calculate detection thresholds. Moreover, for the purpose of obtaining an accurate threshold, we choose two group of numbers that are interleaved. Accordingly, with 10 applications, a period as large as 179 is needed. CPU load, which is about 20% with five applications running, also increases monotonically with the number of applications. Therefore, the scalability of this scheme to the number of applications can be evaluated by studying the effect of period length and CPU load.

To study the scalability while isolating the impact of changes in CPU load, we increase the number of applications by running multiple instances of App-III, App-IV, and App-V that are not CPU intensive. Figure 20 shows that the accuracy of F1 and

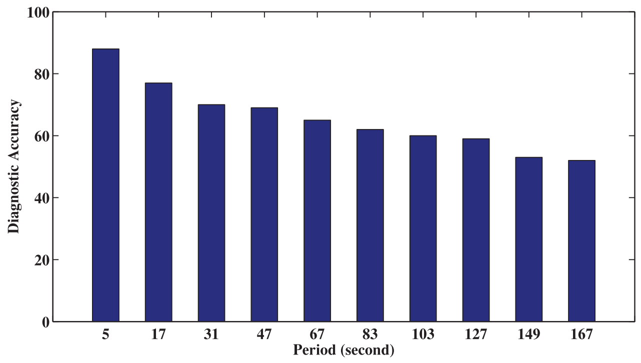


Fig. 19. Effect of heartbeat period on accuracy.

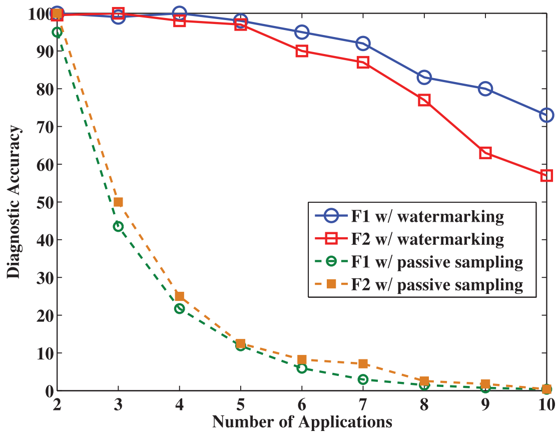


Fig. 20. Effect of number of applications on accuracy.

F2 goes down with the increase of the number of applications. The performance degeneration is mainly caused by the errors in detecting heartbeats with large periods. In Figure 19, we show the accuracy for heartbeats with different periods for 10 applications. Accuracy drops from 88% for heartbeats of period 5 to 52% for heartbeats of period 167. This is because heartbeats with larger periods appear less within a time window and thus their similarity scores are more vulnerable to noise. Thus, overall accuracy is inversely related to the period. Also, this raises a fairness issue that applications with shorter periods are more favored in diagnosis. One may therefore assign short periods to applications with higher priority in diagnosis. On the other hand, we can see that the diagnostic accuracy of the passive sampling scheme drops dramatically as the number of applications increases. This is because App-III, App-IV, and App-V are not CPU intensive, and thus do not generate obvious power signatures that could be captured by the low-end power meter. Hence, the passive sampling scheme has a problem in distinguishing their states.

To study the effect of CPU load without changing the heartbeat patterns, we still use the five applications and their designated periods. A dummy process is used to bring extra load to the CPU. As shown in Figure 21, the diagnostic accuracy for F1 and F2 drops with the increase of CPU load, due to the increasing false positives introduced by the dummy application. In fact, CPU load of sensor nodes is not expected to be high

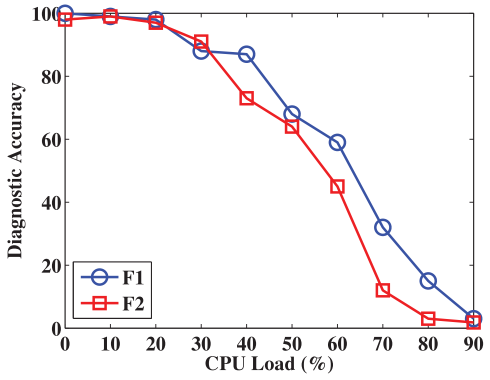


Fig. 21. Effect of CPU load on accuracy.

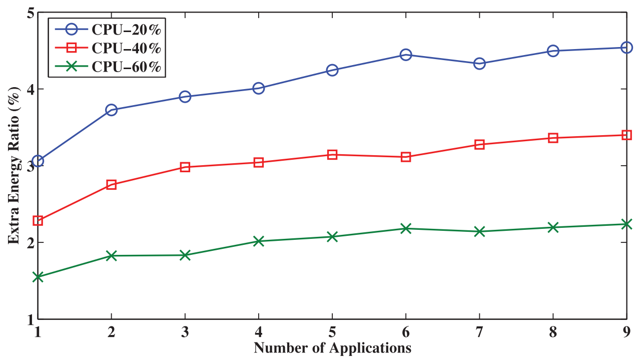


Fig. 22. Energy efficiency of heartbeat generation.

as typical sensing applications are not CPU intensive. This means that our scheme can achieve a reasonably high diagnostic accuracy in most sensor deployments.

Last, we evaluate the energy cost of power watermarking. In Figure 22, we show the energy consumption of heartbeat generation with different numbers of applications. It shows the worst case that all applications are alive and generating heartbeats. The y-axis is the ratio of the extra energy consumption brought up by power watermarking over the normal energy consumption of the host node when no power watermarking is used. When the number of applications increases, heartbeats with more different periods are generated. Consequently, as shown in the figure, more energy is consumed. The energy consumption shown here is normalized by the energy consumption of the host system with no heartbeat, and it becomes relatively small when CPU load increases. In general, the extra energy consumed in this worst case is less than 5% of the original system. This percentage is acceptable for high-end sensing systems that our powertracer targets. Recall that we activate heartbeat generation only in case of failure, and this further conserves energy.

As confirmed by the experiment results, power watermarking is a good supplemental to the passive sampling scheme to identify application states when the number of applications is large. The passive sampling scheme only has about 12% accuracy in distinguishing application states with the five applications, while power watermarking achieves more than 95% accuracy. This is acceptable, as far as scalability goes, considering that the number of applications that run on sensing systems is usually not large.

This performance can be further improved if we increase the window size or sampling rate.

7. LIMITATIONS AND FUTURE WORK

In principle, the approach used in this article to troubleshoot unresponsive sensor nodes remotely based on power consumption characteristics can be applied to a wide range of systems. However, it is crucial to understand the assumptions made in our current work before it can be extended to other contexts.

We developed the tool presented in this article specifically for high-end sensing systems such as SolarStore [Yang et al. 2009] that justify the cost of using an additional power meter. For low-end systems that use low-power devices, such as Tmote and MicaZ motes, we may need to develop power meters that can run at a lower voltage (our current meter needs a minimum of 6.2 volts to operate reliably). For such low-power devices, noise may become a problem. Exploring efficient algorithms for identifying patterns in the presence of a low signal-to-noise ratio will be an important challenge to investigate.

The current system does not consider resource efficiency of upload of meter readings. We simply note that the upload requirements are low due to the low sampling frequency of the meter. Efficient and optimized mechanisms for uploading meter readings could present another avenue of future improvement. Different optimizations may depend on the availability of infrastructure, such as connectivity to wired backbone networks or wireless networks including 3G technologies and WiMax.

From an algorithmic perspective, one limitation of our current diagnostic analysis is that it has no notion of "unknown" state. Currently, it classifies any state as either a "normal" state or a "failed" state depending on its distance measure from the trained states. To fix this, one can use a threshold-based scheme, such that if the distance measure from all the known states is larger than a predefined value, it is classified as an "unknown" state. Exploring classification algorithms that admit explicit unknown states is a worthwhile avenue for future work.

In the current article, the authors artificially restricted the system to 1 meter. Obviously, measuring the power used by different components of the system separately can significantly enrich the set of identifiable energy features, hence making classification more accurate and effective. If the price of meters is small enough to allow using more than one device, this can be an important direction for improving the current scheme.

In summary, the diagnostic powertracer presents an initial proof of concept that demonstrates how energy measurements can be indicative of the nature of failures. We hope that this initial investigation will set the stage for many future extensions and improvements that address the limitations outlined earlier. We should stress that the previous extensions mostly increase the scope of applicability of the approach. In our own outdoors deployment, the approach has already proven adequate and valuable in identifying sources of failures in our system. The powertracer, as it currently stands, is good at diagnosing failures in static, high-end sensing systems.

8. CONCLUSIONS

This article presented a case and a proof of concept for the use of power as a side channel for remote diagnostics, where damage assessment is performed on unresponsive, remotely deployed nodes. An independent power-metering subsystem was used to collect power consumption traces of high-end sensor nodes. A number of algorithms were compared in their ability to determine possible causes of failure and infer application states by analyzing the power traces. It is shown that accurate diagnostics are possible by using a pretrained classifier when the number of states to be classified is not large. This is acceptable for many sensor network deployments where the number of concurrent applications is small. To improve the scalability, we also propose a power watermarking scheme, which achieves 95% accuracy with five applications and more than 70% accuracy with 10 applications. Power watermarking is a good supplemental to the passive sampling scheme to identify application states, while it indeed requires one to run the watermark generator inside the host node, which is less nonintrusive than the passive sampling scheme. The cost of such a system is likely to decrease given the modern trends in metering hardware. The powertracer is currently deployed in conjunction with an outdoor solar-powered high-end sensor system for acoustic and video monitoring. Future work will focus on algorithm robustness as described in the previous section, as well as on experiences with the deployed system.

REFERENCES

L. Asker and R. Maclin. 1997. Ensembles as a sequence of classifiers. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 860–865.

Dmitri Asonov and Rakesh Agrawal. 2004. Keyboard acoustic emanations. In *Proceedings of IEEE Symposium on Security and Privacy*. 3–11.

Eric Bauer and Ron Kohavi. 1999. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning* 36, 1–2 (1999), 105–139.

Qing Cao, Tarek Abdelzaher, John Stankovic, Kamin Whitehouse, and Liqian Luo. 2008. Declarative tracepoints: A programmable and application independent debugging system for wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*.

Chen-Ting Chuang, Chin-Fu Kuo, Tei-Wei Kuo, and Ai-Chun Pang. 2005. A multi-granularity energy profiling approach and a quantitative study of a web browser. In *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05)*.

Adriaan de Jong, Matthias Woehrle, and Koen Langendoen. 2009. MoMi: Model-based diagnosis middleware for sensor networks. In *Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks (MidSens'09)*.

Digi XBee. Wireless Mesh Networking: ZigBee vs. Digi Mesh. Retrieved from http://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf.

Wan Du, David Navarro, Fabien Mieyeville, and Frédéric Gaffiot. 2010. Towards a taxonomy of simulation tools for wireless sensor networks. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*.

Matthias Dyer, Jan Beutel, Thomas Kalt, Patrice Oehen, Lothar Thiele, Kevin Martin, and Philipp Blum. 2007. Deployment support network a toolkit for the development of WSNs. In *Proceeedings of the 4th European Workshop on Wireless Sensor Networks (EWSN'07)*.

Vladimir Dyo, Stephen A. Ellwood, David W. Macdonald, Andrew Markham, Cecilia Mascolo, Bence Pásztor, Salvatore Scellato, Niki Trigoni, Ricklef Wohlers, and Kharsim Yousef. 2010. Evolution and sustainability of a wildlife monitoring sensor network. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*.

Joakim Eriksson, Fredrik Österlind, Niclas Finne, Nicolas Tsiftes, Adam Dunkels, Thiemo Voigt, Robert Sauter, and Pedro José Marrón. 2009. COOJA/MSPSim: Interoperability testing for wireless sensor networks. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*.

Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic analysis: Concrete results. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems*. Springer-Verlag, London, UK, 251–261.

Giorgio Giacinto, Fabio Roli, and Giorgio Fumera. 2000. Design of effective multiple classifier systems by clustering of classifiers. In *Proceedings of the 15th International Conference on Pattern Recognition (ICPR'00)*. 3–8.

Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin. 2004. EmStar: A software environment for developing and deploying wireless sensor networks. In *Proceedings of the ATEC*. 24–24.

G. W. Hart. 1992. Nonintrusive appliance load monitoring. *Proceedings of the IEEE* 80, 12 (Dec. 1992), 1870–1891.

Ray Horak. 2007. *Telecommunications and Data Communications Handbook*. Wiley-Interscience.

HP iLO. 2010. Server Remote Management with HP Integrated Lights Out. Retrieved from http://h18013.www1.hp.com/products/servers/management/remotemgmt.html.

IBM Remote Supervisor. 2010. Remote Supervisor Adapter: Installation Guide. Retrieved from ftp://ftp.software.ibm.com/systems/support/system_x_pdf/48p9832.pdf.

François Ingelrest, Guillermo Barrenetxea, Gunnar Schaefer, Martin Vetterli, Olivier Couach, and Marc Parlange. 2010. SensorScope: Application-specific sensor network for environmental monitoring. *ACM Transactions on Sensor Networks* 6, 2 (March 2010), 17:1–17:32.

Stephen C. Johnson. 1967. Hierarchical clustering schemes. In *Psychometrika*. Springer, New York, 241–254.

Abu Raihan M. Kamal, Chris J. Bleakley, and Simon Dobson. 2014. Failure detection in wireless sensor networks: A sequence-based dynamic approach. *ACM Trans. Sensor Netw.* 10, 2 (Jan. 2014), 35:1–35:29.

Aman Kansal and Feng Zhao. 2008. Fine-grained energy profiling for power-aware application design. *SIGMETRICS Perform. Eval. Rev.* 36, 2 (Aug. 2008), 26–31.

Mohammad Maifi Hasan Khan, Hieu K. Le, Michael LeMay, Parya Moinzadeh, Lili Wang, Yong Yang, Dong K. Noh, Tarek Abdelzaher, Carl A. Gunter, Jiawei Han, and Xin Jin. 2010. Diagnostic powertracing for sensor node failure analysis. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'10)*.

Veljko Krunic, Eric Trumpler, and Richard Han. 2007. NodeMD: Diagnosing node-level faults in remote wireless sensor systems. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys'07)*.

Markus G. Kuhn. 2005. Security limits for compromising emanations. In *Proceedings of CHES 2005, Vol. 3659 of LNCS*. Springer.

Michael LeMay and Jack Tan. 2006. Acoustic surveillance of physically unmodified PCs. In *Proceedings of Security and Management*. 328–334.

Philip Levis, Nelson Lee, Matt Welsh, and David Culler. 2003. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*.

Baoshu Li, Chao Quan, Shutao Zhao, Weiguo Tong, and Peng Tao. 2005. The research of electric appliance running status detecting based on DSP. In *Proceedings of Transmission and Distribution Conference and Exhibition: Asia and Pacific, 2005 IEEE/PES*. 1–4.

Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. ACM, 2–11.

Yunhao Liu, Kebin Liu, and Mo Li. 2010. Passive diagnosis for wireless sensor networks. *IEEE/ACM Trans. Netw.* 18 (2010), 1132–1144.

Menno Lodder, Gertjan Halkes, and Koen Langendoen. 2008. A global-state perspective on sensor network debugging. In *Proceedings of the 5th Workshop on Embedded Networked Sensors (HotEmNetS'08)*.

J. B. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 281–297.

Xin Miao, Kebin Liu, Yuan He, Yunhao Liu, and D. Papadias. 2011. Agnostic diagnosis: Discovering silent failures in wireless sensor networks. In *Proceeding of the 2011 IEEE INFOCOM*.

Jiangwu Nie and Huadong Ma. 2011. Content based pre-diagnosis for wireless sensor networks. In *Proceedings of the 2011 7th International Conference on Mobile Ad-hoc and Sensor Networks (MSN'11)*.

Jiangwu Nie, Huadong Ma, and Lufeng Mo. 2012. Passive diagnosis for WSNs using data traces. In *Proceedings of the 2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems (DCOSS'12)*.

Pranav Patel, Eamonn Keogh, Jessica Lin, and Stefano Lonardi. 2002. Mining motifs in massive time series databases. In *Proceedings of IEEE International Conference on Data Mining (ICDM'02)*. 370–377.

J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras. 2004. ATEMU: A fine-grained sensor network simulator. In *Proceedings of the 1st Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON'04)*. 145–152.

D. Rafiei and A. Mendelzon. 1997. Similarity-based queries for time series data. In *Proceedings of SIGMOD*. ACM, New York, 13–25.

Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, and Deborah Estrin. 2005. Sympathy for the sensor network debugger. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys'05)*. 255–267.

S. Rost and H. Balakrishnan. 2006. Memento: A health monitoring system for wireless sensor networks. In *Proceedings of the 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON'06)*.

Dennis Elliott Shasha and Yunyue Zhu. 2004. *High Performance Discovery in Time Series: Techniques and Case Studies*. Springer.

F. Sultanem. 1991. Using appliance signatures for monitoring residential loads at meter panel level. *IEEE Trans. Power Delivery* 6, 4 (1991), 1380–1385.

Vinaitheerthan Sundaram, Patrick Eugster, and Xiangyu Zhang. 2009. Lightweight tracing for wireless sensor networks debugging. In *Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks (MidSens'09)*.

Gilman Tolle and David Culler. 2005. Design of an application-cooperative management system for wireless sensor networks. In *Proceedings of the 2nd EWSN*. 121–132.

Man Wang, Zhiyuan Li, Feng Li, Xiaobing Feng, Saurabh Bagchi, and Yung-Hsiang Lu. 2011. Dependence-based multi-level tracing and replay for wireless sensor networks debugging. In *Proceedings of the 2011 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*. 91–100.

Ye Wen and Rich Wolski. 2006. S^2DB: A novel simulation-based debugger for sensor network applications. In *Proceedings of the 6th EMSOFT*. ACM, 102–111.

Geoff Werner-Allen, Jeffrey Johnson, Mario Ruiz, Jonathan Lees, and Matt Welsh. 2005. Monitoring volcanic eruptions with a wireless sensor network. In *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN'05)*.

Kamin Whitehouse, Gilman Tolle, Jay Taneja, Cory Sharp, Sukun Kim, Jaein Jeong, Jonathan Hui, Prabal Dutta, and David Culler. 2006. Marionette: Using RPC for interactive development and debugging of wireless embedded networks. In *Proceedings of the 5th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'06) (SPOTS track)*. 416–423.

Jing Yang, Mary Lou Soffa, Leo Selavo, and Kamin Whitehouse. 2007. Clairvoyant: A comprehensive source-level debugger for wireless sensor networks. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*. 189–203.

Yong Yang, Lili Wang, Dong Kun Noh, Hieu Khac Le, and Tarek F. Abdelzaher. 2009. SolarStore: Enhancing data reliability in solar-powered storage-centric sensor networks. In *Proceedings of the 7th International Conference on Mobile Systems, Applications and Services (MobiSys)*. 333–346.

Pei Zhang, Christopher M. Sadler, Stephen A. Lyon, and Margaret Martonosi. 2004. Hardware design experiences in ZebraNet. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*. 227–238.

Ruogu Zhou and Guoliang Xing. 2013. Nemo: A high-fidelity noninvasive power meter system for wireless sensor networks. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*.

Received July 2013; revised March 2014; accepted August 2014