Diagnostic Powertracing for Sensor Node Failure Analysis

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, Xin Jin

{mmkhan2, hieule2, mdlemay2,moinzad1,liliwang,
yang25, dnoh, zaher, cgunter, hanj, xinjin3}@illinois.edu

Department of Computer Science
University of Illinois, Urbana-Champaign
201 N. Goodwin Ave, USA

ABSTRACT

Troubleshooting unresponsive sensor nodes is a significant challenge in remote sensor network deployments. This paper introduces the tele-diagnostic powertracer, an in-situ troubleshooting tool that uses external power measurements to determine the internal health condition of an unresponsive host and the most likely cause of its failure. We developed our own low-cost power meter with low-bandwidth radio to report power measurements and findings, hence allowing remote (i.e., tele-) diagnosis. The tool was deployed and tested in a remote solar-powered sensing network 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 sensors). The paper explores the feasibility of building such a remote diagnostic tool from the standpoint of economy, scale and diagnostic accuracy. To the authors' knowledge, this is the first paper that presents a remote diagnostic tool that uses power measurements to diagnose sensor 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

IPSN'10, April 12–16, 2010, Stockholm, Sweden.

Keywords

Energy, Sensor networks, Troubleshooting

1. INTRODUCTION

This paper introduces a novel diagnostic system, called the *tele-diagnostic powertracer*, geared for remote high-end sensor network deployments, where the cost of in-situ node maintenance is high. The tele-diagnostic 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.

The work described in this paper 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 tele-diagnostic system provides strong clues as to what might be wrong with a node, making it possible to plan intervention accordingly.

While the approach of exploiting power traces to diagnose problems using our tele-diagnostic system is applicable, in principle, to a wide array of sensing systems, we present it and evaluate its performance on a specific deployed platform, called *SolarStore* [34]. 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.

We emphasize that the diagnostic system, described in this paper, 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 paper.

The rest of this paper is organized as follows. Section 2 introduces related work on sensor network troubleshooting. Section 3 presents the design of the tele-diagnostic powertracer on SolarStore. Section 4 explores several power-based diagnostic algorithms that vary in complexity and efficacy. It empirically evaluates algorithmic trade-offs in implementing power-based diagnostic solutions. Section 5 discusses the limitations of our current work along with possible future extensions and improvements. The paper concludes with Section 6.

2. RELATED WORK

Debugging wireless sensor network applications attracted significant attention in the research community recently [14, 20, 12, 13, 4, 25, 33, 32]. Various techniques that are developed to aid sensor network application debugging and troubleshooting at various steps of the development cycle include simulation- and emulation-based tools [17, 30, 23], laboratory testbeds [31, 5, 8], techniques that are developed to assist in finding programming bugs or code-level bugs on real hardware [32, 33], techniques to diagnose protocol bugs such as design errors or corner case bugs [14, 12, 13], real-time network monitoring and troubleshooting tools [25, 20, 28], and formal verification tools [29, 3, 9, 21]. This paper is the first to explore the use of low-bandwidth power traces for diagnostic purposes.

Simulation- and emulation-based tools, such as S^2DB [30], TOSSIM [17], and Atemu [23] are good at the early stages of development to diagnose problems. Emstar [8] is a hybrid pre-deployment tool that can provide visibility into system states by using a callback mechanism during emulation. It allows system states inside an application node to be sent back to a central station, where they can be inspected for possible problems and anomalies. To facilitate debugging sensor network software after deployment, several recent tools [32, 33] provide powerful runtime troubleshooting support, such as breakpoints and watchpoint-style debugging primitives. These tools facilitate inspecting system variables and memory states.

Passive diagnosis techniques [14, 20] have been developed that require minimal help from the application developer to diagnose the root cause of problems with a deployed system. SNTS [14] deploys additional hardware to overhear and record radio communication in the network to troubleshoot protocol bugs by performing offline analysis. Pad [20] collects data on a central PC to diagnose problems based on the partial information provided by each node. SNMS [28] is a sensor network management service that collects and summarizes different types of measurements such as packet loss and radio energy consumption. SNMS provides passively stored information on the node when asked.

Sympathy [25] identifies and localizes failures based on reduced throughput using a decision-tree approach. It is close in spirit to the goals of our approach in that it attempts to determine the root cause of "silence" from nodes in the field. Sympathy can attribute the silence to either a node or a link failure. It nicely complements our approach in that we provide the next level of detail trying to determine the type of node failure when such a failure is responsible for the silence.

Finally, formal methods [29, 9, 21] have been used to verify component correctness. Their techniques are orthogonal to ours in that a system with verified code can also fail, for example, due to a hardware malfunction.

Unlike techniques that require the application nodes to assist in the diagnosis process, for example, by providing internal system states [28, 25, 8], we consider the complementary problem of troubleshooting nodes that become *unresponsive*. The novelty of our approach comes from using low-bandwidth power traces as a side-channel for diagnosis, which has not been attempted in prior work.

A variety of side-channels have been used to extract information from systems. For example, Non-Intrusive Load Monitoring (NILM) algorithms analyze the energy consumption of a segment of a building and determine what appliances are in use in that segment without requiring meters to be individually attached to those appliances. The seminal work on NILM classifies loads into categories based on their power consumption profiles [10] 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 [27, 18]. The Energy Detective (TED) is an example of a product that enables monitoring of home energy consumption to help home owners determine how they use electrical power, primarily to give ideas about how to save on energy costs[1]. Such monitoring enables real-time graphical feeds and profiling of individual appliances, enabling owners to find unnecessary loads like appliances that run when they are not needed or discover appliances that consume the most power. At a grander scale, there is a prospect that power companies will use wireless power meters to aid outage management. For instance, the extent of a blackout can be known much more precisely through advanced meters than by information collected at power substations. This suggests investigating advantages of power diagnostics as a way to gather information about sensor nodes.

[1] `http://www.theenergydetective.com`

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 [15]. 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 [6]. 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 [2, 16]. 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. TELE-DIAGNOSTIC SYSTEM DESIGN

This section presents, in respective subsections, the general design of the tele-diagnostic powertracer and its specific use in the SolarStore system. The objective of the powertracer is to perform remote gross-level damage assessment on unresponsive nodes, such as what may have caused them to stop communicating and what the status of the application might be. This is to be contrasted, for example, with fine-grained debugging tools that attempt to find bugs in lines of code. Fine-grained attribution of causes of error is not a goal of powertracer.

3.1 General Design

Unresponsive behavior of deployed sensor nodes can occur for many reasons, such as broken antennas (e.g., due to storm), software crashes, environmentally-induced damage, or simply energy depletion. When a node stops communicating, an external measurement tool is needed that exploits side-channels to infer further information on node state. Perhaps the most obvious side-channel to exploit is one of power measurement. Sensing, communication, computation, and storage need power in order to be properly carried out. This suggests the possibility of using a power-based tele-diagnostic system to identify the causes of node failure. The question investigated in this work is the degree to which low-frequency sampling of power consumption of an unresponsive node can be used as a side-channel to help diagnose the causes of node silence. Our results show that a sensing node does indeed have a different low-frequency power consumption signature in different normal and abnormal states, leading to the design of a tele-diagnostic powertracer. Our design follows two main objectives:

Diagnostic subsystem independence: The diagnostic subsystem should operate as an independent external measurement tool. It should therefore be self-sufficient and should not require any changes to the system being monitored.

Diagnostic subsystem cost: The diagnostic subsystem should not cost, in either components or energy, a sizable fraction of original sensing node cost. Costs in the range of 1-5% are deemed acceptable, although we demonstrate that costs of around 3% are currently achievable for single units.

Figure 1 shows the tele-diagnostic powertracer. 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

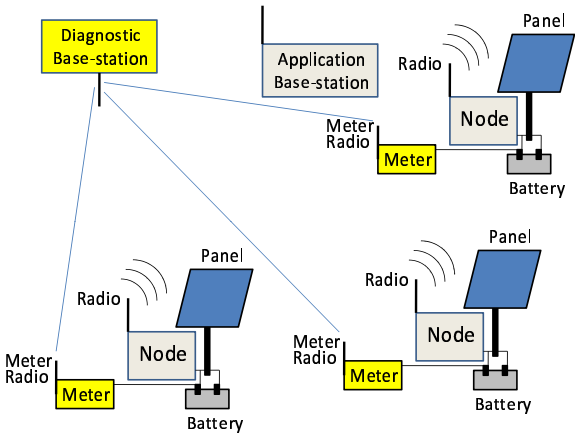


Figure 1: A power-based tele-diagnostic system as an "add-on" to a sensing system

links to a low-end base-station, called the *diagnostic base-station*, that collects the power traces and makes them available to a human operator on request. In our testbed, the deployed diagnostic base-station has wired Internet access.

High sampling frequencies can increase the accuracy of system state estimations. However, 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. The meters we use simply record and relay power measurements to the base-station at a 4.5 Hz effective rate.

Per our design guidelines, the diagnostic subsystem must be independent from the monitored subsystem. Thus, the energy needed for the power meter itself must come 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. Hence, in the common case, when the battery is depleted to a point where its voltage can no longer sustain the monitored system, the meter can still operate, reporting the battery depletion. For example, in our case, the lowest voltage at which the power meter operates reliably is 6.2V whereas the voltage threshold for the monitored system is 11V. To date, no battery failure was observed that would affect both the meter and the monitored system simultaneously, although such a correlated failure remains possible.

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 paper for two reasons. First, it is not a general solution. If node failure is brought about, for example, 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, such

a design choice would violate our design goal of diagnostic subsystem independence.

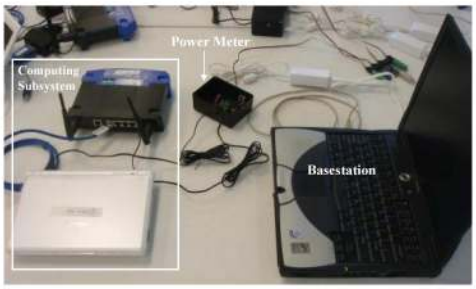
Finally, one should understand that adding a diagnostic subsystem to a remotely-deployed sensor network, necessarily increases the number of components that are deployed in the field and hence increases the odds of component failure. The simplicity of the meter, however, where it merely measures and 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.

3.2 A Specific Testbed

We deployed the tele-diagnostic system in conjunction with an environmental monitoring application that runs on SolarStore. SolarStore [34] is a previously published sensor network service that views the network as a data storage system in which the sensory data collected are the most valuable system output. SolarStore manages these sensory data in the network, when disconnected, and delivers the data to the base station, when one comes in contact with the sensing nodes.

The applications that run on top of SolarStore, in our current deployment, perform acoustic and video recording of local wildlife near a forest. Two actual studies are being performed on the testbed; namely, collection of bird vocalizations in CD-quality sound, and detection of predators of bird eggs using infrared cameras. From a hardware perspective, each node on SolarStore can be broken into (i) an energy delivery subsystem that includes two solar panels to harvest solar energy, and (ii) a computing subsystem that comprises of an embedded PC-grade computer to provide the node's computing capabilities, a Wi-Fi router to support high-bandwidth communication between nodes, and various sensors (e.g., microphones and cameras) to serve different applications. Figure 2(b) shows an outside view of a node.

Separately from the above components, the tele-diagnostic 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. 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 1 mW maximum transmit power. The base station has a matching XBee radio to receive measurements. The entire meter requires around 71mA of current, which means that the meter draws 871mW at the 12.27V provided by our indoor prototype. We use a linear voltage regulator, which should theoretically draw a constant current regardless of voltage, meaning that the efficiency of the system would be higher if the supply voltage were lower. Another mechanism to increase efficiency would be to reduce the sampling rate of the meter and permit the radio to sleep between samples.



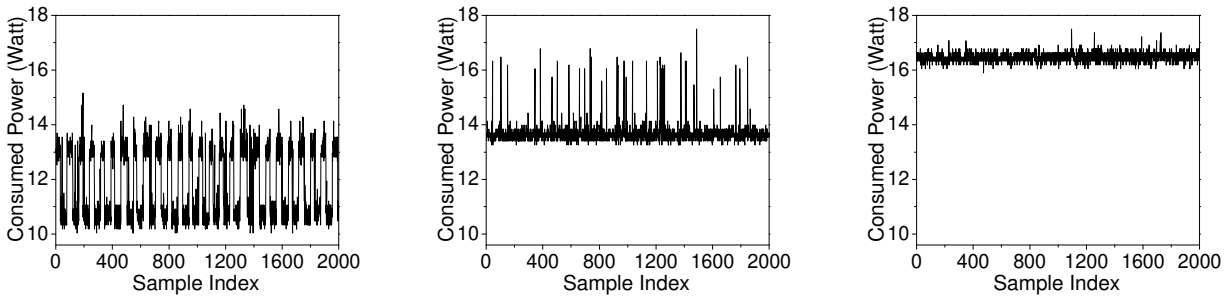
(a)

(b)

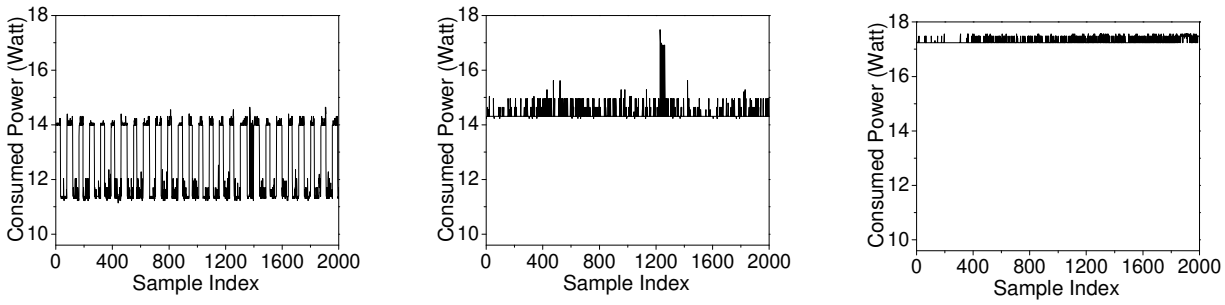
Figure 2: (a)Calibrate the power consumption of the computing subsystems indoor for the initial training of our tele-diagnostic system, (b)Outside look of a node in the solar-powered sensor network testbed

Currently we sample at 1 kHz from the ADC and average the measurements in batches of 220 to achieve an effective sampling rate of about 4.5 Hz. This compensates for the noise in the measurements. Regardless, the current level of power consumption is acceptable.

The total cost for the parts in each powertracer is around $59.41. This figure includes an enclosure and the cost of custom-producing a single PCB. We excluded minor parts such as wire and solder, as well as labor and shipping from this cost metric. This cost of wireless power meters, in general, should soon be much lower, brought about by economy of scale due to the impending proliferation of smart residential power meters, intended to allow remote (wireless) real-time collection of subscriber power consumption by utility providers. It is true that residential meters are AC, not DC, but a large category of AC meters can measure DC current as a side-effect. These AC meters contain circuitry to rectify the sinusoidal waveform of AC current, average it, then scale the result to convert the average to the root mean square (RMS) value. The scaling factor is approximately 1.1, or $RMS(sin(\theta))/AVG(|sin(\theta)|)$. If an AC meter is used to measure DC current, rectification circuits have no effect. The result will therefore be a somewhat smoothed waveform that is about 1.1 of the real DC value. This is demonstrated in Figure 3, where we show the results of measuring the same sensor node current using both a DC and an AC meter simultaneously, for three different software execution scenarios. Indeed, the AC meter yields a somewhat smoother output that is magnified by a factor of



(DC1) f2 (DC2) f5 (DC3) f9



(AC1) f2 (AC2) f5 (AC3) f9

Figure 3: Power traces by a DC/AC meter in three failure states: f2 (router fails and application is sampling sound), f5 (antenna fails and application crashes), and f9 (OS crash).

approximately 1.1, compared to the DC meter. This output is equally acceptable for purposes of classifier training. The availability of low-cost AC and DC meters suggests that a diagnostic subsystem, built from off-the-shelf components is indeed feasible both technically and economically.

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 2 shows an indoor node with a power meter measuring its power consumption.

4. POWER-BASED DIAGNOSTICS

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

4.1 Problem Statement

Our goal is to determine whether low-frequency power traces can be used to (i) distinguish among a range of common failure causes and (ii) infer gross-level application state on unresponsive nodes. Typically, the common failure causes in remote deployments are known from past experience. For example, our initial experience with a solar-powered deployment suggests that the most common failure cause is energy depletion. Other causes of unresponsive behavior of nodes include software crashes and communication device failures. We have also encountered cases of infinite loops involving a node reboot, and short-circuit due to water damage (shown in Figure 4).

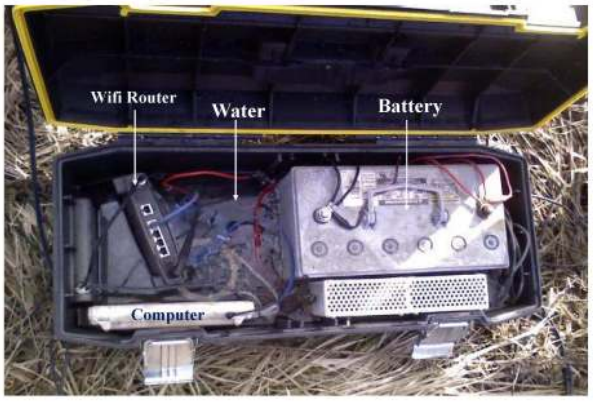


Figure 4: One node drowned in the flood by heavy rains.

We take the above 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. The tele-diagnostic system is trained to recognize failures from their recorded power traces. Common failures can be emulated in the lab prior to deployment for purposes of diagnostic subsystem training. As new failures are encountered after deployment, their traces are used to re-train the diagnostic subsystem to recognize them in the future. Observe that,

in the architecture described in Section 3, the diagnostic algorithm is run on the operator's machine, as opposed to in the field. Hence, retraining simply involves updating the classifier at the operator's desk using data received from the field. It does not entail a need to upload new software to remotely deployed nodes.

Some failure modes, such as system crashes, entail application failure. Others, such as radio failures, do not give information on application status. It is therefore desired that the diagnostic subsystem can tell, upon occurrence of such failures, which applications are still running (i.e., are able to save their sensed data to disk). We exploit the fact that sensor networks do not typically run a wide range of different applications concurrently. Remotely deployed networks often have very specific purposes. Hence, the application count is limited. This significantly simplifies the diagnostic task. Indeed, the techniques presented in this paper are not likely to scale to a large number of applications. However, in a sensor networks context, they may still be useful for the cases of dedicated deployments. For example, only two applications are running in our current deployment.

To test the accuracy of the diagnostic techniques, we therefore set, as a benchmark, the goal of distinguishing among the twelve failure states shown in Figure 5. These include router failures (radio device is out), antenna failures (radio device is on, but the antenna is damaged), operating system crashes, solar energy-depletion, short-circuits (presumably induced by water damage but emulated in our tests by shunting power inputs using a small resistor), and infinite loops involving a system reboot (since, unlike other infinite loops, these would interfere with both the application execution and operating system functions, causing the node to potentially become unresponsive). The above failures were chosen because they had been observed in the field. Moreover, in cases that do not necessarily entail application failure (namely radio and antenna failures), it is desired to tell which of the installed applications is running. The two installed applications in our system are acoustic monitoring and camera surveillance. This leads to the diagnostic tree shown in Figure 5.

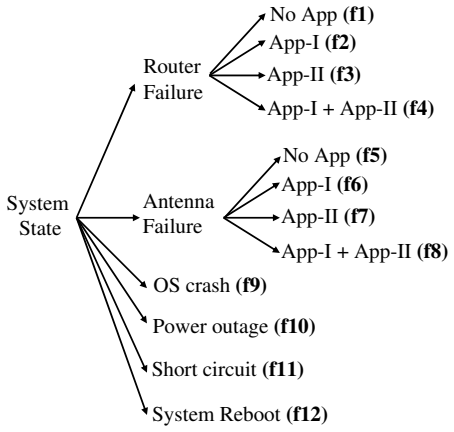


Figure 5: Possible failure states in our system. App-I is the application responsible for sensing sound using microphone. App-II is the application responsible for recording images.

When a failure occurs, diagnosing it within a few minutes is considered good enough. We thus use a 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.

4.2 Static Power Consumption Features

In the simplest case, we characterize the power consumption pattern for a particular failure state by the parameters of the probability distribution of power in the sampled power time-series observed for this state. These parameters are the mean, μ_k, and the standard deviation, σ_k, for the time series of state 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 $x_1, x_2, x_3, ..., x_N$. The mean, μ_k, and standard deviation, σ_k, over a training window of n samples in state k, are calculated as follows:

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

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

After training, each state is represented using a (μ_k, σ_k) pair and stored for later reference. At run-time, 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 test the accuracy of the above scheme, we collected 80,000 samples at the rate of 4.5 samples per second for each of the system states shown in Figure 5. 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. 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 6. Our experiments show that the improvement in classification accuracy with increased window size diminishes after a size of approximately 10 minutes. In the next section, we seek to remedy the above inaccuracies using more involved classifiers.

4.3 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

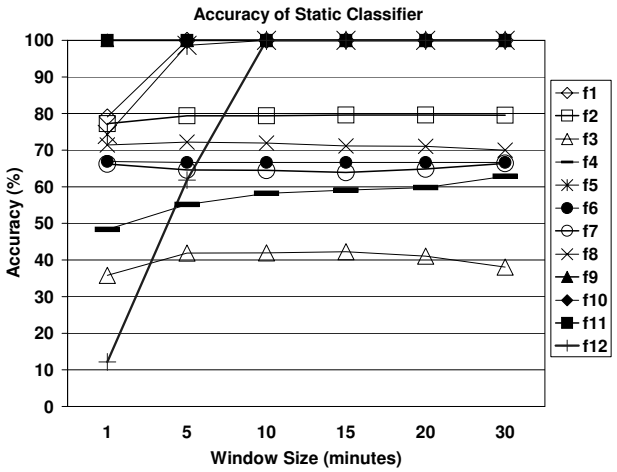


Figure 6: Effect of the window size on classification accuracy for static feature based classification(Data preprocessing used: Outlier filtering)

can be borrowed from the literature [19, 22, 26, 24]. We explore the use of Markov Models.

4.3.1 Modeling

A popular method for capturing the dynamics of complex time-series is the Hidden Markov Model (HMM). The model determines 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 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 an outlier filtering step. We collect data for each system state and subsequently calculate the mean(μ) and standard deviation(σ) for that system state, then discard any value that is outside the range of $[\mu - 5 * \sigma, \mu + 5 * \sigma]$ as an outlier. Next, we can perform an optional step where we may perform smoothing or normalization to input data based on system configuration.

Discretization: Power consumption produces continuous-valued time-series data, which are difficult 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 Watts can be represented as "a", 1-2 Watts as "b", and so on. These symbols represent measured power consumption levels, henceforth called *power states*. The entire power trace is therefore converted to a string of symbols.

Model construction: 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 total of 6 transitions (e.g., the first "a" is followed by the second "a", second "a" is followed by the third "a", 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 above are subsequently used for classifying system states during runtime diagnosis. When a node becomes unresponsive, we collect δ power samples and build a transition probability matrix for the collected samples. Next, we calculate the probability that the observed sequence of samples is indeed generated by the Markov Model for system state k, using the transition probability matrix generated during the training stage for system state k. The state which has the highest probability of generating the observed sequence is returned as the classification result.

To test the accuracy of the above family of classifiers, we used the same training and testing data set that is used for the static feature based classification. We used this approach to answer the following questions regarding the classifier design.

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

These questions are addressed below. In the following, for brevity, we refer to the Markov Model as an HMM (although technically the states in our HMM are not "hidden").

4.3.2 Effect of HMM Size

To see the effect of the number of HMM states on classifier accuracy, we varied the number of states as 5, 10, 15, 20, 30, 40, 50, 60 and 70 and tested the HMM with a window size of 30 minutes. The effect of the number of states on accuracy is given in Figure 7. For this experiment, we trained the HMM on raw data (after noise reduction). As we can see from Figure 7, 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 HMM 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 paper, we use 50 states for HMMs unless we specify otherwise.

4.3.3 Effect of Sampling Frequency

Since reducing the sampling interval increases energy consumption, we evaluate the effect of accuracy of the HMM classifier with various sampling intervals. We train the

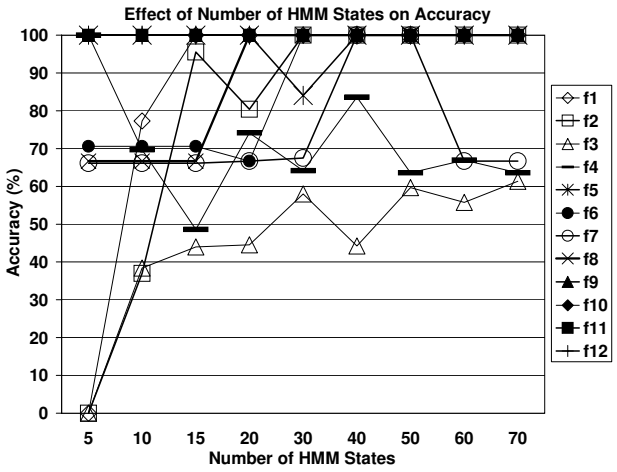


Figure 7: Effect of number of HMM states on classification accuracy(Window size=30 minutes, Data preprocessing used: Outlier filtering)

HMM classifier at the sampling interval of 222 ms, 444 ms, 888 ms, 1776 ms, 3552 ms, 7104 ms, 14208 ms, 28416 ms, and 56832 ms respectively. The lower sampling intervals were obtained from the same data by down-sampling the original time series (i.e., selecting one every N original samples for $N = 1, 2, 4, ..., 256$). We present the effect of the sampling interval on accuracy in Figure 8. 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.

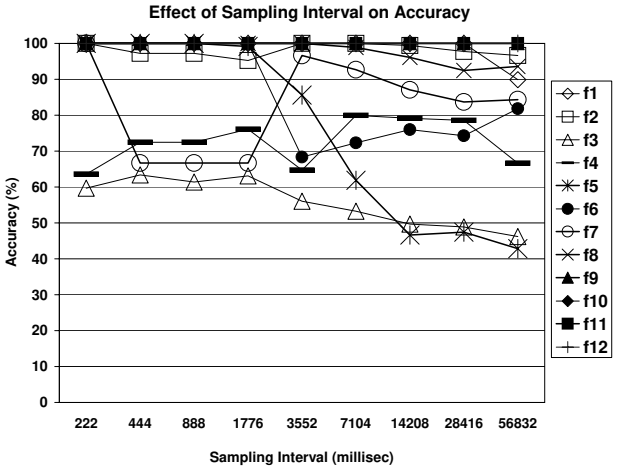


Figure 8: Effect of sampling rate on the classification accuracy of HMM (Window size=30 minutes, Number of states=50, Data preprocessing used: Outlier filtering)

4.3.4 Effect of Window Size

To test the effect of window size on accuracy, we trained the HMM 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 9. In general, increasing window size helps increase the overall accuracy. The amount of improvement varies between different failure states.

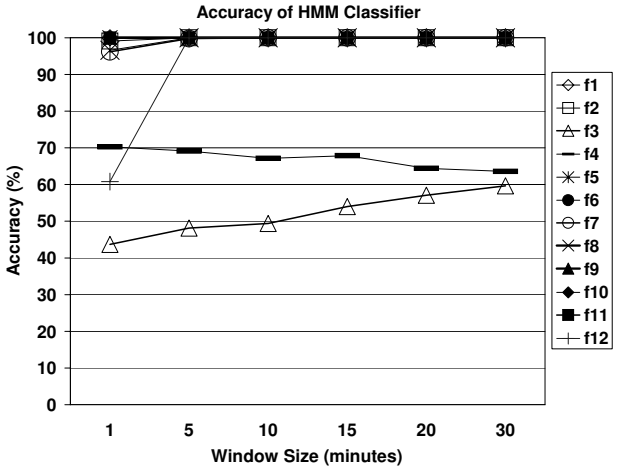


Figure 9: Effect of window size on the classification accuracy of HMM(Number of states=50, Data preprocessing used: Outlier filtering)

We also show the confusion matrix to determine the probability of misclassification and illustrate which states are most likely to get confused. Table 1 gives the confusion matrix for a window size of 30 minutes. A cell (i,j) in the confusion matrix represents the probability that of system state i (the row) will be classified as system state j (the column). The performance of the HMM 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 occassionally 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 infering which application is running.

	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

Table 1: Confusion matrix for classification using HMM (window size=30 minutes, Number of states=50, Normalization used: Outlier filtering)

4.3.5 *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 HMM 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 normalization, and (b) normalization based on relative difference. We describe each of these techniques below.

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

$x'_i = (x_i - \mu_k)/\sigma_k$

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. 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 data mining literature. In Figure 10 we present the impact of varying window size on accuracy of an HMM trained based on z-score data. It turns out that the accuracy of HMMs using z-score normalization is not encouraging and can not be used for diagnosis effectively. We omit the confusion matrix due to space limitations.

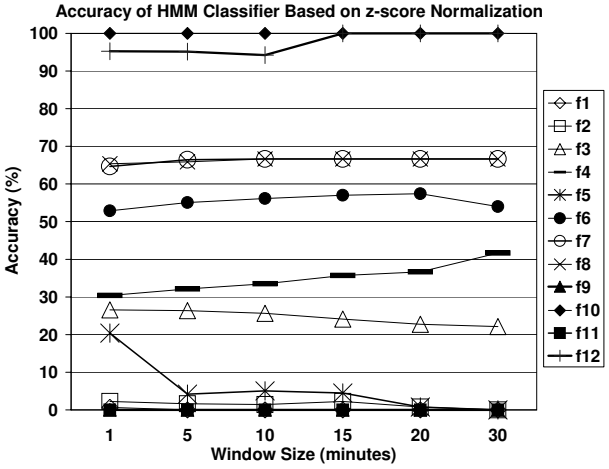


Figure 10: Effect of window size on the classification accuracy of HMM(Number of states=50, Data preprocessing used: Outlier filtering, z-score normalization)

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 formula:

$x'_i = x_i - x_{i-1}$

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 11 we present the impact of window size on the accuracy of the trained HMM. As we can see from Figure 11, the window size has a significant impact on HMM classifier accuracy. The accuracy is considerably less compared to HMM when no normalization is done.

The intuition behind such poor performance when normalization is used is that 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. We omit the confusion matrix due to space limitations.

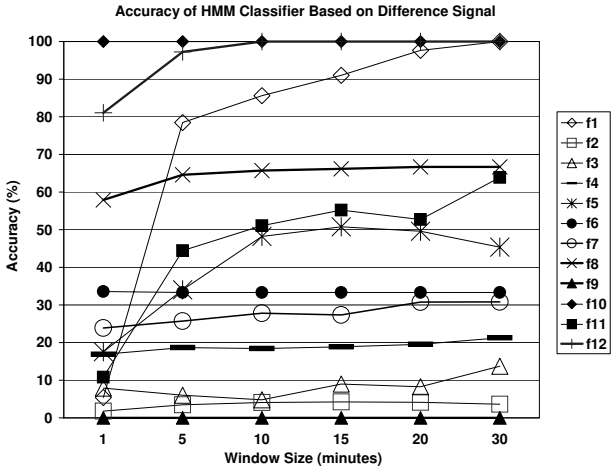


Figure 11: Effect of window size on the classification accuracy of HMM(Number of states=50, Data preprocessing used: Outlier filtering, difference between consecutive signal)

4.3.6 *Discretization by Clustering*

Discretization of the power signal is an important step towards computing the HMM. 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 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 HMM.

In this section, instead of using even ranges, we employ the hierarchical clustering technique [11] 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. Distance between two clusters is given by the average distance between points belonging to the different clusters. Table 2 shows the confusion matrix for

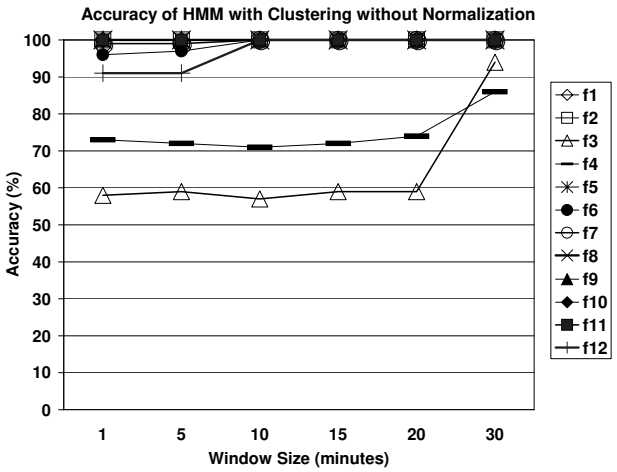


Figure 12: Effect of window size on the classification accuracy of HMM with clustering(Number of clusters=50, Data preprocessing used: Outlier filtering)

the HMM with the clustering scheme with 50 clusters. As the results show, the HMM classifiers with clustering perform better than earlier schemes. This result is encouraging since with the same number of states, the HMM classifier with clustering performs better. Figure 12 shows the detailed accuracy of each failure state of the system versus the window size.

	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

Table 2: Confusion matrix for HMM with clustering (window size=30 minutes, Number of clusters=50, Normalization used: Outlier filtering)

4.4 Discussion

To summarize, from our evaluation it is clear that the static feature-based classifiers exhibit poor performance. HMMs trained using original signal (after removing outliers) without any normalization schemes are better and give reasonable performance. HMM that uses clustering for discretization performs best. One way to improve accuracy is to train a separate HMM for the states that are misclassified. For example, in our case we trained one HMM for all the states and another HMM with 300 states only for state f3 and f4. The idea is to first use the HMM with 50 states to classify the sample and if it is classified as any other state except f3 and f4, we know that it is correct. The secondary HMM is used only when any state is classified as f3 or f4 by the 50 state HMM. Using this two stage scheme we obtained an overall classification accuracy of 98.3% for f3 and 86% for f4. Another point to note is that even if we have less than 100% classification accuracy for a particular state, as we are doing remote diagnosis, we can afford to collect data for more than 30 minutes and try to identify the cause of the problem across multiple 30 minute windows and narrow down the cause of the problem. In the future, we plan to explore the idea of ensemble classifier approaches widely used in machine learning for classification [1, 7] where classification accuracy is improved by combining a group of weaker ones, called subclassifiers.

4.5 On-site Evaluation

In the previous section, a near perfect classifier was developed that uses low-frequency power samples to identify several problems with sensor network nodes in our lab. Below, we describe the experience of using this classifier in an outdoor deployment.

Our experimental testbed bundles higher-end solar powered sensor node hardware with data storage and communication services. It comprises of 10 nodes, where each node is powered by one 12 Volt deep cycle battery (DEKA 8G31) with a capacity of 98 AH, charged by a set of two solar panels. Aiming at high-end data acquisition, we chose Asus EEE PCs for their powerful computing capabilities, reasonably large local storage and high power efficiency. In addition, each node is equipped with a Linksys WRT54GL router, which is configured to the ad-hoc mode to support high-bandwidth communication between nodes. To experiment with using power consumption measurements to diagnose node failures, we developed a compact remote power meter discussed in Section 3.2.

To evaluate our diagnostic system, we collected real data from the real deployment described in section 3. For testing purposes, we artificially induced problems such as operating system crashes, antenna and router failures. Our scheme was able to identify correctly which component failed. Since in a live system we have access to several 30 minute windows of meter data (each shifted a small interval with respect to the previous one), it is easy to take majority vote. Hence, while individual windows may have led to classification errors, taking majority vote over a sequence of windows compensated for these.

5. LIMITATIONS AND FUTURE WORK

In principle, the approach used in this paper 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 paper specifically for high-end sensing systems such as SolarStore [34] 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 Volt 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.

Another algorithmic limitation of the current system is its general lack of scalability. As the number of applications increases, the number of possible system states grows exponentially making them increasingly more difficult to classify. Rather than learning to tell the difference among an exponentially growing number of states, future incarnations of our algorithm will need to find characteristic energy features of each application or failure state that are *invariant* in that they do not change with the introduction of other concurrent applications or failures. Finding such invariant energy features is a non-trivial undertaking that would be of great interest to explore. A related topic is one of possibly adding "energy watermarks" to the execution of different applications in order to create such invariant features where they do not exist naturally.

In the current paper, the authors artificially restricted the system to one 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.

Finally, it is worth mentioning that we currently assume that all application failure signatures are known to the diagnostic algorithm in advance by virtue of prior training. Hence, once our algorithm is trained for a specific system with specific applications, it may not be used to troubleshoot other systems without re-training. Designing an adaptive algorithm that can learn new system states dynamically is a good direction for future work. Of particular interest is to design predictive algorithms that anticipate failure signatures of a modified system given previously recorded signatures and a description of the nature of modification. This will remove the need, for example, to retrain the system upon every software upgrade.

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 above. We should stress that the above 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, dedicated to a single or to a small number of applications.

6. CONCLUSIONS

This paper 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 was shown that accurate diagnostics were possible. 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. To the authors' knowledge, this paper is the first in exploring the use of power traces for diagnostic purposes in sensor networks. Future work will focus on algorithm scalability and robustness as described in the previous section, as well as on experiences with the deployed system.

7. ACKNOWLEDGMENTS

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. The views expressed are those of the authors only. We would also like to thank numerous anonymous reviewers and our shepherd Ákos Lédeczi for their valuable feedback.

8. REFERENCES

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

[2] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 3–11, CA,USA, 2004.

[3] P. Ballarini and A. Miller. Model checking medium access control for sensor networks. In *Proceedings of the 2nd International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'06)*, pages 255–262, Paphos, Cyprus, November 2006.

[4] Q. Cao, T. Abdelzaher, J. Stankovic, K. Whitehouse, and L. Luo. Declarative tracepoints: A programmable and application independent debugging system for wireless sensor networks. In *Proceedings of the 6th SenSys*, 2008. Raleigh, NC, USA.

[5] E. Ertin, A. Arora, R. Ramnath, M. Nesterenko, V. Naik, I. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, and H. Cao. Kansei: A testbed for sensing at scale. In *Proceedings of the 4th IPSN (SPOTS track)*, pages 399–406. ACM Press, 2006.

[6] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *Proceedings of the 3rd International Workshop on*

Cryptographic Hardware and Embedded Systems, pages 251–261. Springer-Verlag London, UK, 2001.

[7] G. Giacinto, F. Roli, and G. Fumera. Design of effective multiple classifier systems by clustering of classifiers. In *Proceedings of ICPR2000, 15th Int. Conference on Pattern Recognition*, pages 3–8, 2000.

[8] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In *Proceedings of the ATEC*, pages 24–24, Boston, MA, 2004.

[9] Y. Hanna, H. Rajan, and W. Zhang. Slede: A domain-specific verification framework for sensor network security protocol implementations. In *Proceedings of the 1st ACM Conference on Wireless Network Security (WiSec)*, Alexandria, VA, March-April 2008.

[10] G. Hart. Nonintrusive appliance load monitoring. *In Proceedings of the IEEE*, 80(12):1870–1891, Dec 1992.

[11] S. C. Johnson. Hierarchical clustering schemes. In *Psychometrika*, pages 241–254. Springer New York, 1967.

[12] M. M. H. Khan, T. Abdelzaher, and K. K. Gupta. Towards diagnostic simulation in sensor networks. In *Proceedings of the 4th DCOSS*, pages 252–265, 2008. Greece.

[13] M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han. Dustminer: Troubleshooting interactive complexity bugs in sensor networks. In *Proceedings of the 6th SenSys*, pages 99–112, 2008. Raleigh, NC, USA.

[14] M. M. H. Khan, L. Luo, C. Huang, and T. Abdelzaher. Snts: Sensor network troubleshooting suite. In *Proceedings of the 3rd DCOSS*, pages 142–157, 2007. Santa Fe, New Mexico, USA.

[15] M. G. Kuhn. Security limits for compromising emanations. In *Proceedings of CHES 2005, volume 3659 of LNCS*. Springer, 2005.

[16] M. LeMay and J. Tan. Acoustic surveillance of physically unmodified pcs. In *Proceedings of Security and Management*, pages 328–334, 2006.

[17] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st SenSys*, Los Angeles, California, USA, 2003.

[18] B. Li, C. Quan, S. Zhao, W. Tong, and P. Tao. 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*, pages 1–4, 2005.

[19] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. 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*, pages 2–11. ACM Press, 2003.

[20] K. Liu, M. Li, Y. Liu, M. Li, Z. Guo, and F. Hong. Pad: Passive diagnosis for wireless sensor networks. In *Proceedings of the 6th SenSys*, 2008. Raleigh, NC, USA.

[21] P. Olveczky and S. Thorvaldsen. Formal modeling and analysis of wireless sensor network algorithms in real-time maude. In *Proceedings of the IPDPS*, Rhodes Island, Greece, April 2006.

[22] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*, pages 370–377, 2002.

[23] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras. Atemu: A fine-grained sensor network simulator. In *Proceedings of the 1st SECON*, pages 145–152, Santa Clara, CA, October 2004.

[24] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *Proceedings of SIGMOD*, pages 13–25, Tucson, Arizona, United States, 1997. ACM, New York, USA.

[25] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd SenSys*, pages 255–267, 2005.

[26] D. E. Shasha and Y. Zhu. *High Performance Discovery in Time Series: Techniques and Case Studies*. Monographs in computer science. Springer, first edition, June 2004. ISBN-0387008578.

[27] F. Sultanem. Using appliance signatures for monitoring residential loads at meter panel level. *IEEE Transactions on Power Delivery*, 6(4):1380–1385, 1991.

[28] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceedings of the 2nd EWSN*, pages 121–132, Istanbul, Turkey, February 2005.

[29] P. Volgyesi, M. Maroti, S. Dora, E. Osses, and A. Ledeczi. Software composition and verification for sensor networks. *Science of Computer Programming*, 56(1-2):191–210, 2005.

[30] Y. Wen and R. Wolski. s^2db: a novel simulation-based debugger for sensor network applications. In *Proceedings of the 6th EMSOFT*, pages 102–111. ACM Press, October 2006.

[31] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. In *Proceedings of the 4th IPSN(SPOTS track)*, pages 483–488, April 2005.

[32] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler. Marionette: Using rpc for interactive development and debugging of wireless embedded networks. In *Proceedings of the 5th IPSN(SPOTS track)*, pages 416–423, Nashville, TN, April 2006.

[33] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. In *Proceedings of the 5th SenSys*, pages 189 – 203, 2007.

[34] Y. Yang, L. Wang, D. K. Noh, H. K. Le, and T. F. Abdelzaher. Solarstore: enhancing data reliability in solar-powered storage-centric sensor networks. In *Proceedings of the 7th MobiSys*, pages 333–346, New York, NY, USA, 2009. ACM.