FailureSense: Detecting Sensor Failure using Electrical Appliances in the Home

Sirajum Munir
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
munir@cs.virginia.edu

John A. Stankovic
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
stankovic@cs.virginia.edu

***Abstract*—With the proliferation of inexpensive sensors, sensors are increasingly being used in smart homes. Recent experience on long term sensor deployment in residential homes [1] has identified the potential risk of various types of sensor failure. Motivated by real examples, we develop new schemes to detect not only *fail-stop* failure, but *obstructed-view* and *moved-location* failures that are not the traditional fault detection foci. Our proposed solution, *FailureSense*, uses a novel idea of using electrical appliances to detect sensor failure at home. We learn the regular patterns of sensor firing with respect to appliance activation events and report a failure when we observe a significant deviation from the regularity. By using data from three real home deployments of over 71 days and 2818 recorded turn on and off events of 19 monitored appliances, we observe that FailureSense can detect *obstructed-view*, *moved-location*, and *fail-stop* failures with 82.84%, 90.53%, and 86.87% precision, respectively, with an average of 88.81% recall.**

***Index Terms*—Sensor failure detection, sensor reliability.**

I. Introduction

With the proliferation of inexpensive, wireless, battery powered, and do-it-yourself sensors, sensors are increasingly being used in smart homes for various applications, including home health care, energy management, and security. Recent experience [1] on large scale and long term sensor deployment in residential homes has identified that although the failure rate of a single sensor is low, when we consider hundreds of simultaneously deployed sensors over long periods, that leads to weekly or even daily sensor failure. Table I shows some sensor failure statistics from some deployments in [1]. The deployments had a wide range of sensors ranging from 47 to 217 deployed for 25 to 44 weeks. From Table I we see that the average number of sensor failures per day is at least one for each home and for one home it is more than 30. Sensors just don't die, they experience failure in a variety of ways. For example, sensors installed on furniture are moved or covered and produce invalid data. Sensors get dislodged not only due to regular usage, but due to guests, cleaning services and other non-residents. Detecting these types of failures is not the traditional fault detection foci, but will be increasingly important as some predict that smart homes will have hundreds of sensors deployed in the near future.

Motivated by real examples, we develop new schemes to detect not only *fail-stop* failure, but *obstructed-view* and *moved-location* failures that are very difficult to detect even if sensors are made highly reliable and are not addressed well enough to date. *Fail-stop* failure happens when a sensor fails and stops reporting. It may happen due to a hardware failure or power outage. *Obstructed-view* failure happens when a sensor's field of view is compromized. A motion sensor experiences *obstructed-view* failure when it is blocked because some furniture is placed in front of it, or a magnetic sensor may be obstructed because an appliance with a motor is placed near it. A sensor experiences *moved-location* failure if it is moved to a different location. It may happen if the sensor gets dislodged and someone puts it in a different place. It may also happen if it is mounted on furniture and the furniture is moved to a different place.

State of the art techniques [2] [3] that detect *fail-stop* failure cannot detect *moved-location* failure as the sensor is still responding, but the reported values are incorrect. SMART [4] and RFSN [5] detect *non-fail-stop* failure, where a sensor doesn't die completely, but reports incorrect values. But SMART requires non-trivial effort in training. Also, it is not scalable in detecting multiple failures if multiple sensors fail within a short period of time. But as shown in Table I, the potential risk of multiple sensor failures within a short period of time can be high as we observed more than 30 sensors to fail per day on average in one of the homes. Our solution can detect multiple sensor failures even if they fail simultaneously and it requires less training effort than SMART. Another state of the art solution, RFSN requires sensor redundancy to detect sensor failure. Deploying redundant sensors in a home not only increases the cost of deployment, it creates unnecessary channel contention and hurts the aesthetics of the rooms. Our solution doesn't require sensor redundancy, although it is permitted.

We propose a novel technique, called FailureSense that detects not only *fail-stop* failure, but *obstructed-view* and *moved-location* failures by monitoring the usage of electrical appliances in the home. The idea behind the proposed solution is, a significant portion of our activities involve handling electrical appliances. When

House	Deployment Duration (days)	# Sensors Deployed	Sensor Downtime (#days*#sensors)	Avg Sensor Failure per Day
House G	308	217	11346	37
House H	273	67	786	3
House I	224	157	5789	26
House J	175	47	432	3

TABLE I. SOME STATISTICS OF SENSOR FAILURE FROM FOUR REAL-HOME DEPLOYMENTS. COMPUTED FROM [1].

someone turns on an electrical appliance, e.g., light, fan, microwave, stove, dishwasher, washing machine etc., he has to be physically present to turn it on. We realize that this assumption may not hold for all the electrical appliances, e.g., someone can turn on the TV by a remote, but still it holds for a significant portion of electrical appliances in the home and a careful selection of the appliances suffices. Thus, whenever someone turns on the selected appliances, there will be a positive correlation between the appliance activation event and the firing of the motion sensor that is covering that area. Since the movements of the residents are constrained by the floorplan of the house, with sufficient training data, we learn *all the regular intervals* between sensor firing and appliance activation and report a failure when we see a significant deviation from the *regular* behavior.

There are several advantages in using the power infrastructure to monitor electrical appliance usage to detect a sensor failure. First, the power infrastructure does not fail as often as sensors do. Correlation based techniques rely on sensor redundancy and can't detect a failure when the redundant sensors also fail. Relying on the power infrastructure will significantly reduce this dependency. Secondly, some electrical appliances are turned on in a periodic fashion, e.g., we turn on lights in our rooms almost every night. So, we can compute the failure detection latency of sensors based on electrical appliance usage of the residents. Thirdly, with much emphasis on energy saving and smart energy management, power meters may provide real-time appliance specific energy usage in the near future. We already have several COTS devices, e.g., TED [6], and eMonitor [7] in the market and as these technologies become more popular, our solution can get a free ride and detect sensor failure almost free of cost.

This work has three major research contributions. First, to the best of our knowledge, we are the first to show that activation events of the electrical appliances in the home can be useful for detecting not only *fail-stop* failure, but importantly *obstructed-view* and *moved-location* failures that are common in smart homes and barely addressed in real deployments or in the literature to date. Second, our solution requires minor training effort, it is scalable in detecting multiple sensor failures even if they fail simultaneously, and it doesn't require sensor redundancy, thus saves cost of redundant sensor deployment, avoids unnecessary channel contention, and improves the aesthetics of the rooms at homes. Third, by using data from three real home deployments of over 71 days and 2818 recorded turn on and off events of 19 monitored appliances, we observe that our solution can detect *obstructed-view*, *moved-location* and *fail-stop* failures with 82.84%, 90.53%, and 86.87% precision, respectively, with an average of 88.81% recall.

II. RELATED WORK

Several techniques have been proposed to address *fail-stop* failure. In Memento [2], nodes in the network cooperatively monitor each other to implement a distributed *fail-stop* failure detector by using other protocol's beacons as heart-beat messages. LiveNet [8] uses passive sniffers co-deployed in the network to reconstruct network topology and disambiguate failures observed at application level. In Sympathy [3], each live node generates two types of data: it's own periodic traffic and Sympathy generated traffic. Code running at the sink detects a failure when a node generates less own traffic than expected. These techniques are mainly suitable for low-level debugging and detecting *fail-stop* failure, but not designed to detect *non-fail-stop* failure, e.g., *obstructed-view* and *moved-location* failures in an event-driven system.

A sensor may experience *non-fail-stop* failure in various ways and a taxonomy of common sensor data faults is defined in [9]. SMART [4] detects *non-fail-stop* failure by analyzing the relative behavior of multiple classifier instances trained to recognize the same set of activities based on different subset of sensors. SMART requires non-trivial effort in training and it is not scalable in detecting multiple sensor failure if they fail within a short time frame. RFSN [5] detects *non-fail-stop* failure by exploiting the correlation between neighboring sensors. But it requires sensor redundancy to build meaningful correlation and it doesn't work if the neighboring sensors are compromised or failed. FIND [10] detects *non-fail-stop* failure under the assumption that sensor readings reflect the relative distance from the nodes to the event. It works where the measured signals attenuates with distance, e.g., a system that captures acoustic signals. It also requires redundant sensors to compare relative distances. Compared to these solutions, our solution detects *obstructed-view*, *moved-location*, and *fail-stop* failures without redundancy with much less training effort.

Detection and classification of electrical appliance activation is not our main focus. Several techniques are available for this purpose that use single-point sensing [11] [12] [13], distributed sensing [14], or a combination of these two [15]. We prefer to use single-point sensing for our work, because it doesn't rely on the deployed sensors for which we are trying to assess failure.

III. APPROACH

Our approach for detecting sensor failure has two major steps: first, based on an empirical study we learn and model the *regular* behavior of a sensor with respect to electrical appliance activation at home. Second, we monitor sensor behavior continuously and report a failure when we see a significant deviation from the regularity. In this section, we describe our assumptions, data collection procedure, empirical study to model *regular* behavior of

House	# Days	# Motion Sensors	# Electrical Appliances	# Turn on/off Events
A	15	15	10	660
B	35	8	8	1921
C	21	5	1	237

TABLE II. SUMMARY OF DATA COLLECTION FROM THREE REAL-HOME DEPLOYMENTS.

sensor firing with respect to appliance usage, and how we use this model to detect a sensor failure.

A. Assumptions

We assume that sensors do not fail during the training period. We also assume that sensors and electrical appliances are stationary, i.e., they are not moved from their original position in the training period. This assumption does not hold for all the electrical appliances. But as long as it holds for some appliances in each room, like light switches, our solution will work.

B. Deployment and data collection

To learn how motion sensors fire with respect to turn on/off events of electrical appliances, we use data from two publicly available datasets [1] (Houses A and B) and collect data from one real-home deployment (House C). Overall we use 15 days of data from House A, 35 days of data from House B, and collect 21 days of data from House C. The number of motion sensors, the number of electrical appliances monitored, and the number of turn on and turn off events of the monitored appliances in each house is shown in Table II. Houses A and C are 4-person homes whereas House B is a 3-person home. The floorplan along with the positions of the motion sensors of House A is shown in Figure 1. The deployment at House C is mainly targeted to detect *obstructed-view* failure. Since we do not have any control over the deployment at Houses A and B, and to detect *obstructed-view* failure, we need to obstruct some sensors' view, we deploy a few sensors at House C and obstruct their views.

X10 motion sensors are used in all homes since they are inexpensive. We collect the timestamp and sensor ID of each sensor firing. Several electrical appliances are available in homes. But we use only lights in all homes since almost every room has at least one light and light switches are usually stationary. Since energy disaggregation is not our focus, ZWave Light Switches, Zwave Plugs, and X10 contact sensors are deployed to figure out when lights are turned on/off. We collect the timestamp and appliance ID of each of the turn on and turn off events of these electrical appliances.

C. Empirical Study and Sensor-Appliance Behavior Model

Based on a total of 71 days of collected data from three multi-person homes having 2818 turn on and off events of 19 monitored appliances, we try to understand the sensor firing pattern with respect to the usage of electrical appliances. We monitor the interval between

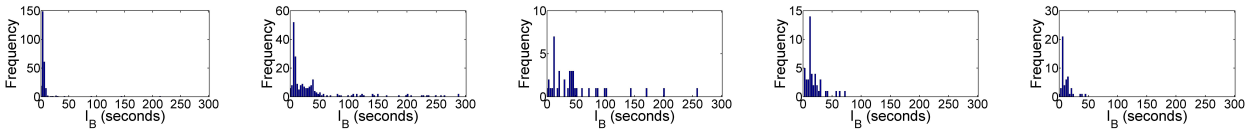
Fig. 1. Floorplan and positions of motion sensors in House A

appliance turn on/off and sensor firing events, and observe regularity in intervals for some sensor-appliance pairs.

As we mentioned before, for most of the electrical appliances, when they are turned on, someone has to be physically present to turn them on. So, if there is a motion sensor nearby, it should fire *before* the turn on event and *after* the turn on event. If the appliance is turned on the same way all the time, it would take the same amount of time for the sensor to fire before and after the turn on event. However, in reality, people may not access the electrical appliances the same way all the time. Since the movements of the residents are constrained by the floorplan of the house, there are only a few ways to reach to the electrical appliances. If we can collect enough training data, we are able to capture different possible ways the residents can reach the electrical appliances. For some electrical appliances we should observe the similar behavior for the turn off events too.

To characterize the sensor firing pattern with respect to appliance usage, we define 3 parameters: I_A, I_B, and $window$. Let I_A $(Interval\ After)$ be the smallest interval between a turn on/off event of an appliance and a sensor firing where the sensor firing happens *after* the turn on/off event. Similarly, let I_B $(Interval\ Before)$ be the smallest interval between a turn on/off event of an appliance and a sensor firing where the sensor firing happens *before* the turn on/off event. Note that I_A and I_B are different for different sensor-appliance pairs. We use a $window$ of 5 minutes and don't consider any sensor firing before/after 5 minutes of the turn on/off events of the appliances. We keep it fixed for all the homes and in all the experiments. Our subsequent analysis shows that I_A and I_B are useful *features* to detect *fail-stop* failure, *obstructed-view* failure, and *moved-location* failure.

As an example to see how these parameters reveal sensor firing patterns with respect to appliance usage, let's consider some sensors and electrical appliances from House A. The floorplan and the position of all the motion sensors of House A are shown in Figure 1. We choose the bathroom light and the mudroom light as appliances. We discretize I_B values into 3 seconds bins and plot the frequency distribution of I_B in Figure 2. Note that only the D1 sensor is in the bathroom, and other sensors are in different rooms when looking at Figures 2(a), 2(b), and 2(c). If D1 suffers from *fail-stop* failure, *obstructed-view*



(a) D1 (b) E1 (c) H2 (d) C4 (e) E2

Fig. 2. Frequency distribution of I_B of sensors (a) D1, (b) E1, and (c) H2 with respect to the bathroom light and of sensors (d) C4, and (e) E2 with respect to the mudroom light of House A.

failure or it is moved to some other room, the frequency distribution of I_B of D1 with respect to the bathroom light is going to change. I_A also shows a similar distribution. The bathroom light clearly helps in detecting the failure of D1 sensor. It may also help in detecting failure of other sensors, e.g., E1. For other sensors, we may need to consider other appliances.

Capturing regularity in intervals can be challenging since a sensor-appliance pair can have multiple regular intervals. To capture all the regular intervals of a sensor-appliance pair, based on the empirical study, we use a **Gaussian Mixture Model (GMM)** to represent the distribution of I_A and I_B. A GMM has q components, where each component captures one regular interval.

The probability density function of a GMM is a weighted sum of the q component Gaussian densities as shown in the following equation:

$$p(\boldsymbol{x}|\lambda)=\sum_{i=1}^{q} w_i g(\boldsymbol{x}|\boldsymbol{\mu_i},\boldsymbol{\Sigma_i}), \quad (1)$$

where $\boldsymbol{x}$ is a D variate observation, w_i are the mixture weights and $g(\boldsymbol{x}|\boldsymbol{\mu_i},\boldsymbol{\Sigma_i})$ are the Gaussian densities of the components, where $i = 1, 2, 3, ..., q$. Each component density follows a Gaussian distribution with mean vector $\boldsymbol{\mu_i}$ and covariance matrix $\boldsymbol{\Sigma_i}$ as shown in the following equation:

$$g(\boldsymbol{x}|\boldsymbol{\mu_i},\boldsymbol{\Sigma_i})=\frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma_i}|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_i})'(\boldsymbol{\Sigma_i})^{-1}(\boldsymbol{x}-\boldsymbol{\mu_i})} \quad (2)$$

Mixture weights satisfy the constraint that $\sum_{i=1}^{q} w_i = 1$. The complete GMM is parameterized by the mean vectors, covariance matrices, and mixture weights from all the component densities. These parameters of the GMM are collectively represented by the following notation:

$$\lambda=\{w_i,\boldsymbol{\mu_i},\boldsymbol{\Sigma_i}\}i=1,2,3,...,q. \quad (3)$$

In our case, we have separate Gaussian Mixture Models for I_A and I_B and our $\boldsymbol{x}$ is a single variate observation of I_A or I_B. The parameters of the mixture model (λ) are estimated using the **Expectation Maximization (EM)** algorithm from the training data. When choosing λ, we try using 1-4 components and choose the one that minimizes the Akaike Information Criterion (AIC), which is a measure of relative goodness of fit of a statistical model. Note that it is not necessary for all the sensor-appliance pairs to follow their I_A and I_B according to GMM. We just need enough appliances so that each sensor is covered by at least one appliance, which we find an easy requirement for all the three houses based on the empirical study.

D. Appliance Selection

For each appliance, we compute the probability of each sensor firing within the *window* of appliance activation events. For each sensor, we choose top k appliances with which the sensor firing is most probable. We *associate* these appliances with these sensors. It means that the failure detection of these sensors only depends on these appliances. The higher the value of k, the lower are the failure detection latency and precision of failure detection. The value of k can be chosen based on application requirement. We use k = 2 in the evaluation since the instrumented appliances were limited in our data collection.

E. Online Failure Detection

After modeling the *regular* behavior of sensor firing during the offline training phase, we monitor the sensor-appliance behavior in terms of I_A, I_B online and report a failure when we observe a deviation from the distribution according to some thresholds (described below). More specifically, at runtime, we detect sensor failure using the following steps:

1) Appliance Monitoring: For the selected appliances, we monitor when the appliances are turned on/off.

2) Sensor Monitoring: When a selected appliance is turned on, we monitor the firing of the *associated* sensors within the duration of $window$.

3) Probability Computation: Based on appliance usage and sensor firing, we compute i_A and i_B, which are the observed values of I_A and I_B, respectively. We also compute p_A and p_B, the probabilities of observing these i_A and i_B using equations (4) and (5) as follows:

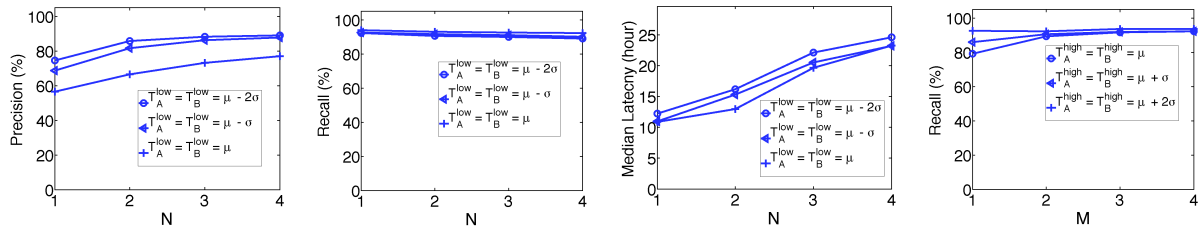
The probability of observing $I_A = i_A$ is,

$$p_A = p(I_A = i_A|i_A \leq window) * p(i_A \leq window) \quad (4)$$

The reason for having these two separate terms is because we only consider I_A and I_B values within $window$ at the time of modeling GMM. We compute $p(i_A \leq window)$ directly from the training data, and $p(I_A = i_A \mid i_A \leq window)$ is computed from the GMM using equation (1). Similarly, the probability of observing $I_B = i_B$ is,

$$p_B = p(I_B = i_B|i_B \leq window) * p(i_B \leq window) \quad (5)$$

4) Failure Decision: We decide the state of the sensor from the values of p_A and p_B. If $p_A \leq T_A^{low}$ *and* $p_B \leq T_B^{low}$ for N times in a row, we report a failure of the associated sensor, where T_A^{low}, T_B^{low}, and N are thresholds. If $p_A \geq T_A^{high}$ *or* $p_B \geq T_B^{high}$ for M times in a row, we report that the sensor is working fine, where T_A^{high}, T_B^{high}, and M are thresholds. Also, to avoid flooding of reports, after sending one status report, we suppress all the subsequent reports related to that sensor for the next 6 hours.



(a) (b) (c) (d)

Fig. 3. Effect of N, T_A^{low}, T_B^{low} on (a) precision, (b) recall, (c) median latency of failure detection and effect of M, T_A^{high}, T_B^{high} on (d) recall in detecting three types of failure in three houses.

F. Threshold Selection

The performance of FailureSense largely depends on the selection of the thresholds. Overall, we use 6 thresholds: T_A^{low}, T_A^{high}, T_B^{low}, T_B^{high}, N, and M at the time of deciding sensor failure status. These thresholds are specific to sensor-appliance behavior and a static threshold may not work for all possible sensor-appliance pairs in all homes. So, we compute these thresholds by taking into account specific sensor-appliance behavior from the training data. Note that the computation is fairly automatic and it doesn't require any user involvement other than turning on/off appliances while they perform their activities of daily living. The goal is to choose threshold values that maximize precision and recall while minimizing failure detection latency.

1) Selection of T_A^{low}, T_B^{low}, and N: We report a failure of the associated sensor if we observe $p_A \leq T_A^{low}$ *and* $p_B \leq T_B^{low}$ for N times in a row. So, as N increases, the latency to detect a failure also increases. As N increases, we report fewer number of failures which reduces the recall of detecting a failure, but increases the precision. As T_A^{low} or T_B^{low} increases, the frequency of reporting a failure increases, which in turn increases the recall, but decreases the latency and precision.

We compute mean (μ) and standard deviation (σ) of the p_A, p_B values. We change N from 1 to 4 and change T_A^{low}, T_B^{low} using μ and σ of the p_A, p_B values, respectively, while keeping the other thresholds unchanged and show the impact on average precision, recall, and median latency in Figures 3(a), 3(b), and 3(c), respectively in detecting the three types of failure in three houses (experimental setup along with how precision and recall are computed is specified in the next section). We see that as N increases, precision and latency of failure detection increase, but recall decreases. Also, as T_A^{low} and T_B^{low} increase, recall increases, but precision and latency decrease, as expected according to our analysis. We choose N = 3 and select T_A^{low}, T_B^{low} to (μ - $2 * \sigma$) of the p_A, p_B values, respectively, which provides high precision at a little loss of recall with a reasonable median latency. Other threshold values can be chosen if that satisfies application requirement.

2) Selection of T_A^{high}, T_B^{high}, and M: We report the associated sensor is working fine when $p_A \geq T_A^{high}$ *or* $p_B \geq T_B^{high}$ for M times in a row. As these three thresholds T_A^{high}, T_B^{high}, and M are not used in reporting a failure, they don't have any impact on precision and latency of failure detection. But they do have impact on recall. As M increases, we deliver fewer number of reports saying that the sensor is working fine, which increases recall of failure detection. As T_A^{high} or T_B^{high} increases, sensors pass the thresholds fewer number of times, which again increases recall.

We compute mean (μ) and standard deviation (σ) of the p_A, p_B values. We change M from 1 to 4 and change T_A^{high}, T_B^{high} using μ and σ of the p_A, p_B values, respectively, while keeping the other thresholds unchanged and show the impact on average recall in detecting the three types of failure in three houses in Figure 3(d) (experimental setup is described in the next section). This figure shows that increasing M increases recall. Increasing T_A^{high}, T_B^{high} also have a similar impact, as expected according to our analysis. We choose M = 2 and select T_A^{high}, T_B^{high} to μ of the p_A, p_B values, respectively for the evaluation. Setting T_A^{high}, T_B^{high} to (μ + $2 * \sigma$) of the p_A, p_B values slightly increases the recall. However, choosing such high threshold values drastically reduces the number of reports saying that the sensor is working fine, which may not be appropriate for some sensors.

This analysis is useful in choosing thresholds to meet application requirements. Note that we do not compute separate thresholds for detecting different types of failure. We compute a single set of thresholds using the above technique and that works in three houses we tested in detecting all the three types of failure. At the time of reporting a sensor failure, we do not specify the type of sensor failure.

IV. Evaluation

The evaluation is based on data collected from three real homes. The data collection procedure is described in Section III-B. We use 15 days of data from House A, 35 days of data from House B, and collect 21 days of data from House C. The performance of FailureSense is evaluated through a *post-facto* analysis using these datasets, instead of deploying the FailureSense system in another home. Hence, the implementation of FailureSense is a Matlab program that analyzes the collected sensor data and reports a sensor failure using the strategy specified in Section III. It also means that FailureSense is agnostic to the purpose of sensor deployment and the applications that are using the sensor data.

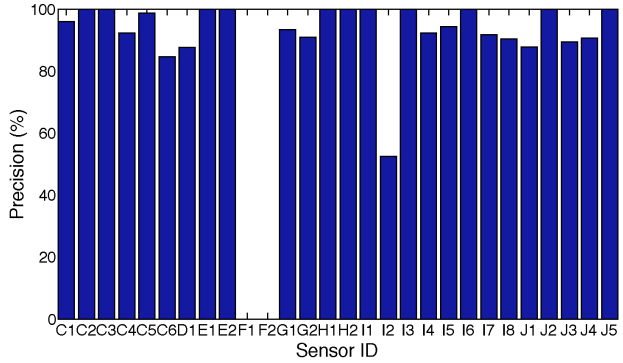


Fig. 4. Precision of detecting *fail-stop* failure at House A (sensors C1 - H2), House B (sensors I1 - I8), and House C (sensors J1 - J5).

We evaluate the performance of our algorithm in terms of precision (% of failure reports where the sensor is actually failed) and recall (% of failed sensor reported). The way we compute these metrics are, when FailureSense reports a sensor status, we compare with the ground truth state of the sensor and determine whether this report is a true positive (TP, i.e., we report a failure when it has failed), false positive (FP, i.e., we report a failure when it has not failed), true negative (TN, i.e., we report a sensor has not failed when it has not failed), or false negative (FN) (i.e., we report a sensor has not failed when it has failed). Then we compute precision = TP/(TP + FP) and recall = TP/(TP + FN). We compare our solution with the state of the art solutions in terms of sensor redundancy, training effort, and scalability in detecting multiple sensor failure at the same time. The experimental setup is different for different types of failure.

A. Performance Results

1) Fail-stop failure: We simulate the behavior of *fail-stop* failure by discarding all the readings of the failed sensor after it fails. We evaluate it on all the sensors of all the three houses containing 28 sensors. We perform 3 fold cross validation for the evaluation. More specifically, for Houses A and C, we train on 10 days of data and test on 5 days of data. For House B, we train on 24 days of data and test on 11 days of data. The reason for using only 15 days of data from house C is because, the sensors' views were obstructed at the 16th day to evaluate *obstructed-view* failure (c.f. Section IV-A2). The sensor failure day is chosen to be the first day of the testing period when computing true positive and false negative, and the last day of the testing period when computing false positive and true negative. Whether it is the first or the last day, we randomly choose 100 timestamps within that day, fail the sensor at these timestamps, and show the average results.

The precision of detecting *fail-stop* failure is shown in Figure 4. The average precision is 86.87%. The recall is 100% for all the sensors except sensors F1 and F2, for which it is 0. The average recall is 92.86%. We see that our solution detects failure of all the sensors except F1 and F2 of House A. The reason why the solution fails to detect the failure of sensors F1 and F2 is because F1 and F2 are in the bedroom of House A and the dataset of House A doesn't contain turn on/off events of any electrical appliances in the bedroom. That's why sensors F1 and F2 are not associated with any electrical appliances and we can not detect the failures of F1 and F2. Also, the precision is really low for sensor I2 of House B. I2 is deployed in one bedroom of House B and this is the only room where instead of using light switches, a lamp is instrumented with a Zwave plug. We are not sure if this is the reason or if this is due to the behavior of the residents, but there were no regularity in intervals for I2 with respect to the lamp. That's why although our solution can detect when the sensor fails, it suffers from a low precision. However, we see that the solution performs well for most of the sensors in three houses.

2) Obstructed-view failure: We evaluate the performance of detecting *obstructed-view* failure by simulating the sensor obstruction in Houses A and B, and by physically obstructing the views of sensors in House C. It is different from the *fail-stop* failure detection in that the failure takes place for a transient period. In House B, we train our solution with 15 days of data and test on the next 20 days of data. Within these 20 days, for each sensor, we randomly choose a 10 day period for which we assume that the sensor view is obstructed and we discard the sensor readings of these 10 days. Since the obstruction can happen at any time, we randomly choose this 10 day period 100 times for each sensor and show the average results. Similarly, for House A, we train our solution with 10 days of data and test on the next 5 days of data. Within these 5 days, for each sensor, we randomly choose a 3 day period of obstruction when we discard the sensor readings. We perform it 100 times and show the average results. At House C, after 15 days of data collection, we obstruct the views of the 5 motion sensors. This is a controlled experiment and we consider arbitrary positions for placing sensors in the home since X10 motion sensors are *do-it-yourself* sensors that end users just buy and place the sensors in their homes without any professional expertise. The way we obstruct them is shown in Table III. We train with the first 10 days of data, evaluate false positive and true negative on the next 5 days of data, we fail the sensors on the 16th day of data collection at 12:00 PM, and evaluate true positive and false negative on the next 5 days of data.

Sensor ID	House ID	How sensor view is obstructed
J1	C	It is mounted at the wall of the living room. A couch is moved that blocks its view.
J2	C	It is mounted at the wall of the dining room. Some moving boxes are placed in front of it.
J3	C	It is mounted at the wall of the kitchen. A cabinet is opened that blocks its view.
J4	C	It is mounted at one cabinet of the kitchen. Its view is blocked when the cabinet is opened.
J5	C	It is mounted at the refrigerator. We assume that it gets dislodged and someone puts it at the top of the refrigerator.

TABLE III. EXPERIMENTAL SETUP FOR *obstructed-view* FAILURE DETECTION.

The precision of detecting *obstructed-view* failure is shown in Figure 5. The average precision is 82.84%. The recall is 100% for all the sensors except F1, F2, I2, J3,

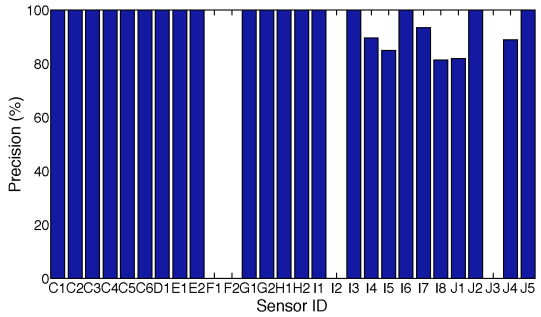


Fig. 5. Precision of detecting *obstructed-view* failure at House A (sensors C1 - H2), House B (sensors I1 - I8), and House C (sensors J1 - J5).

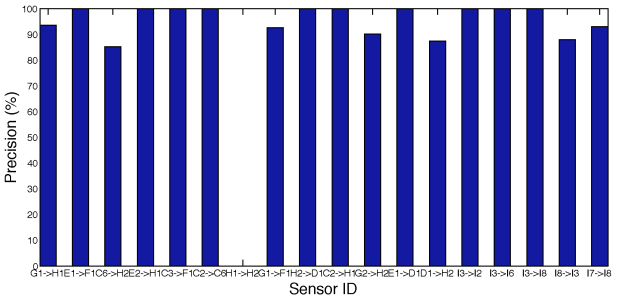


Fig. 6. Precision of detecting *moved-location* failure at House A (sensors C1 - H2) and House B (sensors I1 - I8).

C2, and D1. For sensors F1, F2, I2, and J3, the recall is 0. For sensors C2 and D1, the recalls are 80% and 85.71%, respectively. The average recall is 84.49%. We see that we can not detect the failure of sensors F1, F2, I2, and J3. The reasons for F1, F2, and I2 are, as described in Section IV-A1, F1 and F2 are deployed in the bedroom of House A and the dataset does not contain any turn on/off events of any electrical appliances of that room, and sensor I2 is deployed in one bedroom of House B and there were no regularity in intervals for I2 with respect to the lamp instrumented with a Zwave plug in that room. The reason for not being able to detect the failure of sensor J3 is, although the view of J3 is obstructed and it can not see any motion, it used to fire every time a light switch, which is close to J3, is turned on/off. It may be because the light switch is instrumented with a contact sensor and turning the light switch on/off causes a change in the magnetic field of the contact sensor and causes a small light in front of the contact sensor to blink. We are not sure exactly what caused J3 to fire every time the light switch is turned on/off. But it shows a potential caveat of applying the solution. Appliance selection can be tricky and if the appliance is close, it may cause disturbances.

3) Moved-location failure: We simulate the behavior of *moved-location* failure as in [4] by replacing the failed sensor's data with the data produced by the sensor at its new position. We evaluate it in Houses A and B since the deployment in these houses spans the whole house. We do not use House C in this evaluation as it has a small scale deployment mainly targeted to evaluate *obstructed-view* failure detection. We perform 3 fold cross validation for the evaluation, as described in *fail-stop* failure detection.

The precision of detecting *moved-location* failure is shown in Figure 6. The average precision is 90.53%. The recall is 95% or more in all these cases, except in E2 $\rightarrow$ H1, C2 $\rightarrow$ C6, H1 $\rightarrow$ H2, E1 $\rightarrow$ D1, and I8 $\rightarrow$ I3, where the recalls are 91.83%, 55.30%, 0, 85.71%, and 83.71% , respectively. The average recall is 89.09%. We select a pair of sensors and move the first one to the position of the second one, e.g., in the first case of Figure 6, sensor G1 is moved to H1's location. The positions of sensors along with the floorplan of House A are shown in Figure 1. The dataset doesn't offer us the floorplan of House B. However, from the description, we know that all the sensors of House B in Figure 6 are in different rooms. Our solution works well if sensors are moved from one room to another room. However, if a sensor is moved slightly within the same area, it doesn't cause a significant deviation from the distribution of I_A and I_B and the solution doesn't perform well. This is why when C2 is moved to C6 and H1 is moved to H2, the recalls are low. Note that small sensor displacement may not make any difference in application semantics. On these cases, this type of minor movement is not a movement failure and is not detected as such.

B. Comparison with state of the art

State of the art solutions that detect failure relevant to this work either use correlation based technique (RFSN [5]) or use a classifier based approach (SMART [4]). We compare our solution with both types of strategies.

1) Comparison with RFSN: Reputation-based Framework for Sensor Networks (RFSN) [5] detects misbehaviors sensors by allowing each sensor to maintain reputation of the neighboring ones. It doesn't work when there is no sensor redundancy as each sensor needs to monitor the correlation with the neighboring sensors to detect sensor failure. Even if there is sensor redundancy, if the neighboring sensors are compromised or failed, then building of the reputation metrics may fall into jeopardy. To detect the failure of a sensor, our solution doesn't rely on any neighboring sensors and it works in presence of no redundant sensors.

We evaluate the performance of RFSN and FailureSense when multiple sensors experience *fail-stop* failure at House A and show it in Figure 7. There are 15 motion sensors deployed at House A. We vary the number of failed sensors from 1 to 15. For i number of sensor failures, we randomly select i sensors, run the experiment 100 times and show the average % of sensor failure detected by both algorithms in Figure 7. As shown in Figure 4, FailureSense fails to detect the failure of F1 and F2 sensors, whereas RFSN fails to detect a failure when all the redundant sensors also fail. It shows that FailureSense maintains an average of 86.69% sensor failure detection across all *fail-stop* sensor failures where RFSN performance degrades when more sensors fail.

2) Comparison with SMART: SMART[4] detects *non-fail-stop* failure by analyzing the relative behavior of multiple classifier instances trained to recognize the same

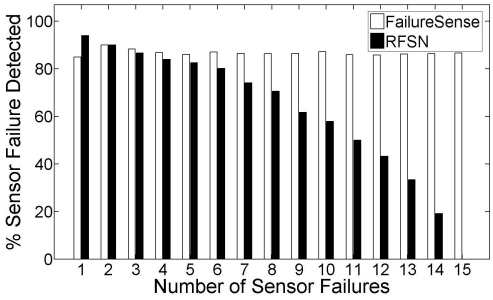


Fig. 7. Percentage of sensor failure detected by FailureSense and RFSN when multiple sensor experiences *fail-stop* failure at House A.

set of activities based on different subset of sensors. It requires *activity labeling* for each house to understand how each sensor relate to each activity, which is a non-trivial amount of work for the training. We do not need activity labeling and our solution reduces the training effort significantly (c.f. Section V-A). Also, SMART requires training of the classifier instances for all possible combinations of sensor failure as because different combinations of sensor failure affect classifier instances differently. If there are n sensors in a house, their approach requires $2^n - 1$ combinations of node failure analysis, which is not scalable. The authors admitted this limitation and evaluated their solution assuming single node failure. Our solution overcomes this limitation as we do not need different training to detect different combinations of sensor failure. We could not make a direct comparison with SMART due to the unavailability of datasets. Because, the dataset we have doesn't have activity labeled and the dataset that SMART uses doesn't have the turn on/off events of electric appliances. But admittedly SMART is not scalable in detecting multiple sensor failure at the same time and the scalability of our solution is shown above.

V. Discussion

In this section, we discuss several aspects of our solution including the training effort, latency of failure detection, availability of smart energy meters, dependency on human behavior, and generalization of the solution to detect failure of other types of sensors.

A. Training Effort

We need training effort to accomplish two tasks:

1) Appliance Activation Detection: We need to know which appliances are turned on/off from the power infrastructure. The training effort largely depends on the selection of appliance activation detection technique. For example, Electrisense [12] can automatically detect and classify the use of electronic appliances at home from a single point of sensing at the power wire using EMI signature. It requires turning on/off five to six times per appliance and some processing time to capture its EMI signature, which is very little.

2) Learning Regular Appliance-Sensor Behavior: To learn regular appliance-sensor behavior and build our model, we need to collect 30-40 turn on/off events per appliance. At this phase, the end-user just performs activities of daily living and our system captures timestamps of appliance turn on/off events and sensor firings. The duration of this phase may vary depending on appliance selection and personal preference of using the appliances. As an example, we observe that the average and standard deviation of the frequency of turn on and turn off events of bathroom light and mudroom light in House A are 16.20 ± 3.80 and 3.13 ± 3.80 per day. Based on the empirical study on three houses, we conclude that it may take about 10-15 days of data to build accurate models.

B. Failure Detection Latency

The failure detection latency depends on appliance selection, the frequency of appliance usage, and the selection of threshold values. As an example, for the experimental setup in Section IV-A, the average median latency of detecting *fail-stop*, *obstructed-view*, and *moved-location* failures are 20.03 hours, 17.69 hours, and 28.52 hours, respectively. The average median latency for detecting these three types of failure is 22.08 hours, which is reasonable for many applications, including remote health monitoring and energy management systems. However, for emergency health care and security services, this latency may be inadequate. Usually people purchase more expensive solutions for these applications.

C. Smart Energy Meter Requirement

Although our solution doesn't require sensor redundancy, it requires a smart energy meter that can detect appliance activation events. This requirement may look like another type of redundancy as such meters are not widespread yet. However, based on a recent study [16], the worldwide installed smart electricity meters will grow at a compound annual growth rate of 26.6 percent between 2010 and 2016 to reach 602.7 million. As people are getting more and more conscious about energy usage and monitoring, smart energy meters may become an indispensable part of future smart homes.

D. Results with Actual Smart Energy Meter

Our results assume that smart energy meters can detect appliance activation events with 100% accuracy. In actual practice, although the state of the art techniques are not 100% accurate, their accuracy is considerably high, e.g., 93.82% of ElectriSense [12] and 90% of [13]. Our solution also assumes that smart meters are more reliable than the deployed sensors, which we found to be true based on deployment experience with the eMonitor [7] energy management and X10 motion and contact sensors.

E. Dependency on Human Behavior

The performance of our solution relies on the behavior of the residents. If someone turns on all the lights and never turns them off, then we will not be able to detect sensor failures. However, based on data collected from three real homes, we see that people actually turn on and turn off lights almost everyday and thus it is reasonable to

assume such behavior. However, there may be exceptional cases. For example, if there is a party, then appliances may be used in a different way. To handle such exceptional cases, we plan to use Exception Flagging [17], where our solution just ignores the data collected during such period. If the behavior of the residents change after the training period, then our performance may degrade. Putting humans into the loop will allow the system to use human feedback to distinguish between behavior change and sensor failure. We consider it our future work.

F. Generalization to other types of sensors

We believe that our solution is generalizable to various types of sensors:

1) Light Sensor: When a light switch is turned on, the nearby light sensor should see a sudden increase of light intensity within a regular period. If the light sensor is failed, or covered, or moved to some other room, it will not be able to see that change. We can use this property to decide if the light sensor has failed.

2) Acoustic Sensor: When someone turns on TV, radio, washing machine, coffee maker, electric shaver, food processor, hair dryer, or a microwave oven, the nearby acoustic sensor should see an increase of sound intensity or even a sound pattern within a regular period. We can use this property to assess the failure state of the neighboring acoustic sensors.

3) Water Fixture Sensor: Sensors that are attached to various water fixtures, e.g., contact sensor attached to a faucet in the bathroom/kitchen and sensor attached to a bathroom flush fire within a short time when water starts to flow through the water fixture. We can use this property for assessing the reliability of the water fixture sensors as we can detect which water fixture is drawing water by using smart water meter and motion sensors as in [18].

However, the solution may not be adequate detecting failure of some other types of sensors, e.g., temperature, humidity, and radio activity. But, a lot of home health care and energy management systems use motion sensors, acoustic sensors, and water fixture sensors to monitor activities of daily living [19] [20] and occupancy patterns [21] of the residents. Such systems will greatly benefit from our solution.

VI. Conclusions

We develop and evaluate a novel failure detection scheme for sensor network deployments found in smart homes. The solution addresses not only *fail-stop* failures, but importantly, *obstructed-view* and *moved-location* failures; new types of failures common in smart homes that are very difficult to detect even if sensors are made highly reliable and barely addressed to date in the literature or in real deployments. Our solution is the first one to detect these failures without requiring sensor redundancy and with minimal training effort. Although our solution has its own limitations, it is applicable to most of the common types of sensors found in real deployments thereby significantly improving the state of art and it begins to address complicated and realistic sensor failures with novel insights.

VII. Acknowledgement

This paper was supported, in part, by NSF Grants CNS-1319302 and CNS-1239483, and a gift from Parc, Palo Alto.

References

[1] T. W. Hnat, V. Srinivasan, J. Lu, T. I. Sookoor, R. Dawson, J. Stankovic, and K. Whitehouse, "The hitchhiker's guide to successful residential sensing deployments," in *SenSys*, 2011.

[2] S. Rost and H. Balakrishnan, "Memento: A health monitoring system for wireless sensor networks," in *SECON*, 2006.

[3] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger," in *SenSys*, 2005.

[4] K. Kapitanova, E. Hoque, J. A. Stankovic, S. H. Son, and K. Whitehouse., "Being SMART about failures: Assessing repairs in smart homes," in *UbiComp*, 2012.

[5] S. Ganeriwal, L. K. Balzano, and M. B. Srivastava, "Reputation-based framework for high integrity sensor networks," *ACM Trans. Sen. Netw.*, 2008.

[6] TED. http://www.theenergydetective.com.

[7] eMonitor. http://www.powerhousedynamics.com.

[8] B.-R. Chen, G. Peterson, G. Mainland, and M. Welsh, "Livenet: Using passive monitoring to reconstruct sensor network dynamics," in *DCOSS*, 2008.

[9] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava, "Sensor network data fault types," in *ACM Trans. Sen. Netw.*, May 2009.

[10] S. Guo, Z. Zhong, and T. He, "Find: faulty node detection for wireless sensor networks," in *SenSys*, 2009.

[11] G. Hart, "Nonintrusive appliance load monitoring," in *Proceedings of the IEEE*, December 1992.

[12] S. Gupta, M. S. Reynolds, and S. N. Patel, "Electrisense: single-point sensing using EMI for electrical event detection and classification in the home," in *UbiComp*, 2010.

[13] S. N. Patel, T. Robertson, J. A. Kientz, M. S. Reynolds, and G. D. Abowd, "At the flick of a switch: detecting and classifying unique electrical events on the residential power line," in *UbiComp*, 2007.

[14] Y. Kim, T. Schmid, Z. M. Charbiwala, and M. B. Srivastava, "Viridiscope: design and implementation of a fine grained power monitoring system for homes," in *UbiComp*, 2009.

[15] X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler, "Design and implementation of a high-fidelity AC metering network," in *IPSN*, 2009.

[16] IntelligentUtility report. http://www.intelligentutility.com/magazine/article/253959/6027-million-installed-smart-meters-globally-2016.

[17] R. Yang and M. W. Newman, "Learning from a learning thermostat: Lessons for intelligent systems for the home," in *UbiComp*, 2013.

[18] V. Srinivasan, J. Stankovic, and K. Whitehouse, "Watersense: Water flow disaggregation using motion sensors." in *BuildSys*, 2011.

[19] B. Logan, J. Healey, M. Philipose, E. M. Tapia, and S. Intille, "A long-term evaluation of sensing modalities for activity recognition," in *UbiComp*, 2007.

[20] D. Cook and M. Schmitter-Edgecombe, "Assessing the quality of activities in a smart environment," *Methods of Information in Medicine*, 2009.

[21] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse, "The smart thermostat: using occupancy sensors to save energy in homes," in *SenSys*, 2010.