# Being SMART About Failures: Assessing Repairs in Activity Detection

Krasimira Kapitanova, Enamul Hoque, John A. Stankovic,
Sang H. Son, and Kamin Whitehouse
*University of Virginia, Charlottesville, VA*
{*krasi, eh6p, stankovic, son, whitehouse*}*@virginia.edu*

Daniele Alessandrelli
*Scuola Superiore Sant'Anna*
*Pisa, Italy*
*d.alessandrelli@sssup.it*

***Abstract*—One of the main challenges for activity recognition systems is that there are numerous sensors which are likely to move or exhibit failures due to hardware degradation, inaccurate readings, and environmental changes. In this paper, we propose a Simultaneous Multi-classifier Activity Recognition Technique (SMART) that uses application-level semantics to detect sensor node failures and improve the detection accuracy under those failures. Once a node failure is detected, instead of immediately dispatching maintenance, SMART evaluates the severity of the failure by using data replay analysis. Maintenance is dispatched only if the severity analysis indicates that the node failure would have caused an application-level failure in the past and the system could not have recovered from it by updating the classifier ensemble it is using. Evaluation of SMART on a set of activities from two public datasets shows that SMART decreases the number of maintenance dispatches by 45% on average and almost triples the mean time to failure of the application. SMART identifies all application-level failures at run time and improves the activity detection accuracy under node failures by more than 70%.**

***Keywords*-wireless sensor networks; activity detection; machine learning; failure analysis**

## I. Introduction

Activity detection using wireless sensor networks (WSN) is becoming increasingly valuable for a wide range of applications such as healthcare, education, and improving the efficiency of heating, ventilation, and air conditioning (HVAC) systems. A major challenge with activity detection is that there are numerous sensors which are likely to move or exhibit either fail-stop or transient failures due to hardware degradation, battery depletion, and environmental changes. These problems are becoming even more important as sensors built into objects become ubiquitous. As the numbers of sensors increase, even a large mean time to failure (MTTF) per sensor can translate to a small MTTF for applications using multiple sensors.

Even when extensive pre-deployment analysis is performed, system designers do not have a full understanding of how to best detect activities in a particular home with particular residents because of all possible patterns the different activities of interest could exhibit. Therefore, in order to improve the activity detection accuracy, these systems are often deployed with some level of redundancy. As a result of this redundancy, node failures might have different effect on the application performance. In this paper, we describe a Simultaneous Multi-classifier Activity Recognition Technique (SMART) that takes advantage of the node redundancy in a system and minimizes the necessary number of repairs and maintenance dispatches. SMART uses multiple classifier instances that are trained preemptively for the occurrence of node failures by *holding one sensor out* and employs application-level semantics to detect sensor node failures. SMART provides two key features:

**1. Failure assessment.** When a node failure is detected, we assess its severity by using data replay analysis to determine what the impact of the failure would have been on the activity detection accuracy of the application in the past. This relies on the basic notion that the past is similar to the present. A lot of work has been done to show that learning over a time period is useful for predicting future patterns [1]. Our system attempts to recover from a node failure by replacing the existing classifier with one that does not depend on the failed sensor node(s). If the detection accuracy would have remained above an application specified threshold when the new classifier is used, no repair is necessary in the present. If, however, the detection accuracy would have deteriorated significantly, physical maintenance is necessary in the present. Since not all node failures are repaired as soon as they occur, our approach *a) decreases the number of maintenance dispatches*, and *b) decreases the number of sensor node repairs* by only repairing the minimal number of sensors such that the detection accuracy of the application is restored back to a satisfactory level.

**2. Simultaneous use of multiple classifiers.** SMART simultaneously uses multiple classifier classes and classifier instances that are preemptively trained for failures. This allows it to: *a) Detect node failures at run time:* The basic approach is to constantly monitor for fluctuations in the relative number of activities reported by the different classifiers and use this information to infer the occurrence of sensor failures. *b) Improve the application accuracy in the presence of failures:* At run time, based on the characteristics of the detected activities and the state of the system, SMART dynamically chooses the most accurate classifier instance. This allows it to maintain high detection accuracy even in the presence of failures.

The key difference between SMART and state of the art health monitoring systems, such as Memento [2], Sym-

pathy [3], or trust-based approaches [4], is that it uses a top-down failure detection technique. Instead of detecting and reporting low-level hardware failures, our approach relies on application semantics to determine the severity of application-level failures, i.e. failures that affect the high-level application behavior of the system. In addition, bottom-up approaches that rely on analysis of raw data, like the ones mentioned above, are not easily applicable to binary, event-triggered sensors, such as motion sensors or reed switches.

In our evaluation we use publicly available data sets from three homes [5], [6]. The node redundancy level in these datasets is very low since they were not designed to be fault tolerant. We believe that SMART will be even more powerful when applied to activity detection systems that are designed with a sufficient level or redundancy in order to provide fault tolerance. Despite the limitations of the datasets we use, our evaluation shows that SMART 1) reduces the number of necessary repairs by 70% and the number of maintenance dispatches by 45% on average, 2) increases the MTTF of the application 2.6 times, 3) can be used to identify 100% of the application-level failures and 60% of all node failures, and 4) improves the activity detection accuracy under the presence of failures by more than 70%.

## II. Related work

Many applications have been developed to help achieve sensor network hardware verification. Health-monitoring systems, such as Memento [2], MANNA [7], and LiveNet [8], employ sniffers or specific embedded code to monitor the status of a system. Sympathy [3] was developed to help with detecting routing problems. However, these applications are focused on examining low-level system parameters. Therefore, they cannot be used as a substitute for our approach. Instead, if available, SMART can take advantage of such techniques and use them to verify that fail-stop sensor failures.

The run time assurance methodology (RTA) is similar to our work in the sense that it targets failures visible at the application level. However, instead of monitoring the performance of activity detection classifiers, RTA uses techniques such as end-to-end application-level tests [9] or statistical checking of probabilistic properties [10], [11], to validate at run time that the application can function correctly despite any changes that might have occurred to the underlying system or the environment. Our approach goes one step further than RTA in that instead of dispatching maintenance when a severe node failure occurs, it attempts to adapt to the failure by switching to a classifier that is preemptively trained for this failure.

A number of applications have used outlier detection to identify faulty sensors. Systems like Suelo [12] learn to identify faults with the help of human expert assistance. This approach is not suitable for home deployments since 1) people living in the monitored homes are usually not experts on activity detection, and 2) people's recollections about the time of occurrence, duration, and order of activities are often inaccurate and incomplete. Another approach based on outlier detection is the reputation-based framework proposed by Ganeriwal et al. [4]. However, building and maintaining a community of trust for activity applications is inherently hard because of the wide variety of sensors involved and the diverse sensor firing patterns each activity can be characterized with.

The idea of using multiple classifiers and dynamically selecting the most accurate one has been explored in pattern recognition and neural networks. Partridge and Yates proposed a number of techniques that exploit heuristic rules for choosing classifier groups [13]. Additional techniques were proposed by Roli and Giacinto [14], [15]. These techniques could be incorporated into SMART to further increase the accuracy of activity detection under failures. This, however, is not the focus of this paper and we leave it for future work.

A lot of work has been done in the area of using sensor nodes for activity detection, such as [5], [16], [17]. The majority of projects have focused on achieving high activity detection accuracy. Different approaches have been applied to this problem, such as designing new classification algorithms, adding more sensors, and increasing the granularity of the activities of interest. Unlike previous work, which does not consider the effect of node failures, our paper studies the accuracy of activity detection algorithms in the presence of sensor node failures.

## III. Approach

The key insight behind SMART is that the severity of node failures should be considered when deciding whether maintenance is necessary and, if so, which nodes should be repaired. Consider the scenario in Figure 1. The activity we are interested in is *cooking*. There are two sensors in the kitchen but only the star-shaped one is located close to the appliances and the sink. If this sensor fails, the detection accuracy for *cooking* will be impacted significantly. Therefore, this failure is severe and maintenance will be necessary if it occurs. On the other hand, if the oval sensor fails, this might not affect the accuracy of detecting *cooking*. In that case, the node does not need to be repaired immediately. Also, if the node redundancy in the kitchen was higher, the failure of the star-shaped node might not have been considered severe since it might have barely affected the detection accuracy.

The core of SMART can be divided in two components: offline and run time (Figure 2). Preliminary failure severity analysis of the effect of sensor failures on the classifiers' performance is performed offline. This analysis can also be rerun in the future if the system behavior evolves over a longer period of time. Also offline, our approach trains classifier instances for all possible combinations of node failures by holding those nodes out of the training set.

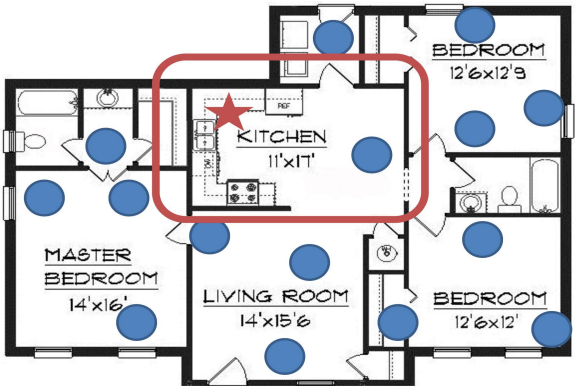


Figure 1. The severity of a sensor node failure is very strongly correlated with the level of redundancy in the system. Therefore, because of the low node redundancy in the kitchen, if the star-shaped node fails, the accuracy of detecting *cooking* activities would be severely impacted.

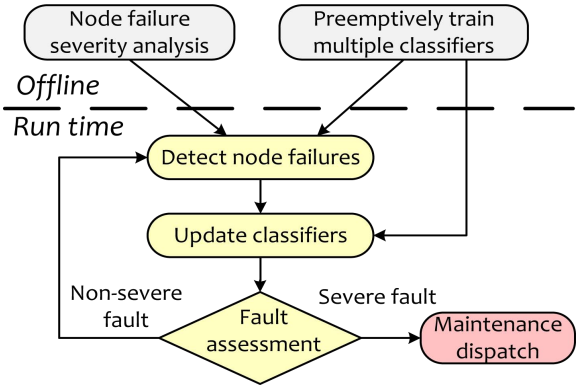


Figure 2. SMART has two components: offline and run time. Preliminary offline analysis of node failure severity is performed and multiple classifiers are trained. The results of this analysis together with the classifiers are used at run time to detect node failures and determine if maintenance is needed.

The results from the failure severity analysis together with the classifiers are used at run time to detect node failures. SMART chooses which of the preemptively trained classifiers to include in the current classifier ensemble based on the failures that have been detected so far. When a new node failure is detected, the system adapts to that failure by updating the classifier ensemble to contain classifiers that are trained for the failure. Since all classifier instances are trained beforehand, the overhead of updating the classifier ensemble at run time is negligible. If the new classifier ensemble can help maintain the detection accuracy of the application above the application specified severity threshold $TH_S$, the failure is considered non-severe. We define $TH_S$ as a fraction of the original accuracy of the system when no failures are present. If a failure is not severe, no maintenance is necessary since the application can still meet its high-level requirements. However, if the node failure is severe and the detection accuracy falls below $TH_S$, one or more sensors need to be repaired. Instead of repairing all failed nodes, SMART determines the minimal number of necessary node repairs that will restore the detection accuracy of the application above $TH_S$. In the rest of this section we describe in more detail: 1) the node failure severity analysis, which allows us to decrease the number of repairs and maintenance dispatches, and 2) the training of multiple classifiers, which allows us to detect failures at run time and maintain higher detection accuracy under failures.

We use the following notation:

**1.** $C\,(Training\ set,\ Testing\ set)$, where, for a classifier instance of class *C*, *Training set* and *Testing set* are the sets of sensors used for training and testing, respectively.

**2.** $C_{Accuracy}$: accuracy of classifier instance C;

**3.** $C_{NDA}$: number of detected activities by classifier instance C;

**4.** $S$: the set of all nodes in the system.

### A. *Node failure severity analysis*

When a failure is detected at run time, we use historical data and apply data replay analysis to evaluate what effect this failure would have had on the application had it occurred in the past. Assume that node $s$ fails, where $s \in S$. We use historical data to estimate $C_{NDA}(S, S)$ that would have been reported by the classifier before the failure. After that, we remove all readings produced by the failed node $s$ from both the training and the testing set and calculate $C_{NDA}(S - s, S - s)$. The difference $|C_{NDA}(S, S) - C_{NDA}(S - s, S - s)|$ determines the effect of the failure on the application. As mentioned earlier, a failure is considered severe if it decreases the detection accuracy below the severity threshold $TH_S$. Some applications might consider failures to be severe if they change the number of detected activities by more than 30% of the original number of detected activities. However, for safety-critical applications, a failure could be severe even if it changes the number of detections by 5%. Instead of using a global severity threshold, an application can also specify per-activity severity thresholds. In this way, if some activities are considered more critical than others, they can be assigned stricter severity thresholds.

*1) Computing the minimal number of repairs:* Maintenance is dispatched every time the detection accuracy of the system falls below the specified threshold $TH_S$. Although repairing all nodes that have failed would be ideal, it might not always be feasible. First, it might take too long and, therefore, cost too much, to repair all nodes in a house with hundreds of nodes where tens of those nodes have failed. Also, if some of the nodes themselves are expensive, it might be more cost-effective to repair just a subset. For those and other similar situations, our approach computes the minimal number of necessary repairs that need to be performed when maintenance is dispatched.

If $S$ is the set of all nodes in the system, the number of possible failure traces is $2^{|S|} - 1$. We use a dynamic programming algorithm to compute the minimal number of

repairs for each of them. At each step we consider failures involving exactly $k$ sensors, starting from $k = 1$ (only one sensor failing) and ending with $k = |S|$ (all sensors failing). The number of $k$-sensor failures is the number of $k$-combinations of all sensors. Each failure $F_{(k,i)}$ is identified by its set of $k$ faulty sensors, and $i$, where $i$ ranges from 1 to $\binom{|S|}{k}$. For each $k$-sensor failure we compute the classifier accuracy $C_{Accuracy}(S, S-k)$. If the accuracy is above the severity threshold $TH_S$, we consider the failure as insignificant to the classifier and no repair is necessary. Otherwise, at least one of the $k$ sensors needs to be repaired, thus going back to a $k-1$-sensor failure situation. In order to minimize the total number of repairs, we always choose to repair the sensor that leads to the $k-1$-sensor failure requiring the minimum number of repairs. The algorithm is formally represented by the following iterative formula:

$$r(F_{0,1}) = 0 \tag{1a}$$

$$r(F_{k,i}) = \begin{cases} 0 & \text{if } C_{Accuracy} \geq TH_S \\ 1 + min\{r(F_{k-1,j}) \\ \quad |F_{k-1,j} \subset F_{k,i}\} & \text{if } C_{Accuracy} < TH_S \end{cases} \tag{1b}$$

### B. Training multiple classifiers

Performing a complete failure severity analysis requires consideration of every possible combination of faulty sensors. The analysis of all $2^{|S|}-1$ failure sequence traces could pose scalability issues for large networks with hundreds of sensors. However, not all of the sequence traces need to be considered simultaneously at run time. Empirical analysis we performed showed that it is enough to initially consider only the $|S|$ single-node failures. Therefore, in addition to the classifier instance $C = (S, S)$ trained on all nodes, we also analyze $|S|$ more classifier instances, where $\forall s \in S$ a classifier instance is preemptively trained for the failure of node $s$, i.e. its training set contains $S - s$ sensors. Figure 3 shows how the training sets for the additional $|S|$ classifier instances are identified. To identify when failures occur, we monitor the relative detection rates of the original classifier instance and that of the other $|S|$ instances.

*1) Failure detection:* To detect sensor failures at run time, SMART relies on monitoring changes in the relative behavior of the classifiers. The accuracy of a classifier changes when one of the nodes it has been trained on fails. Therefore, if we notice a change in the accuracy of a classifier at run time, we can infer that one (or more) of the nodes used for training has failed. To detect changes in the accuracy, we need to have ground truth for comparison. Unfortunately, this is not available at run time. Instead, we compare the behavior of the original classifier instance trained on all available sensors to that of the additional classifier instances trained by holding one of the sensors out.

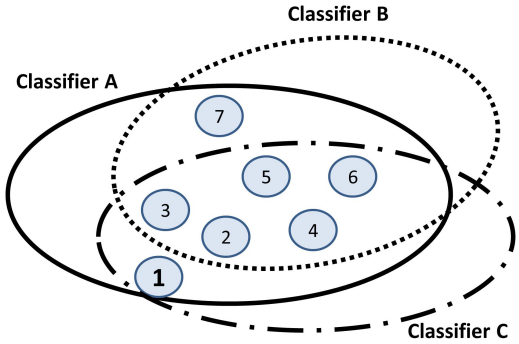


Figure 3. Preemptively training classifiers for the occurrence of failures by *holding one sensor out* from the training set allows us to build failure-aware systems. Classifier A is trained on all nodes but classifiers B and C are trained by holding node 1 and node 7, respectively.

At run time we monitor for changes in the relative number of activities detected by each of the classifiers.

When sensor $s$ fails, we expect that the ratio of reported activities by the original classifier instance trained on all $S$ sensors and that trained without sensor $s$ will change from $R_{old}$ to $R_{new}$, where

$$R_{old} = \frac{C_{NDA}(S,S)}{C_{NDA}(S-s,S)} \qquad R_{new} = \frac{C_{NDA}(S,S-s)}{C_{NDA}(S-s,S-s)}$$

When $|R_{old} - R_{new}| > TH_R$, where $TH_R$ is a predefined threshold, we can infer that node $s$ must have failed.

Unlike health-monitoring, SMART considers more than just fail-stop failures. We also address transient failures that could be caused by node displacement, temporary changes in the environment and connectivity, or faulty values due to power depletion. In the case of a transient failure, when node $s$ recovers from the failure, the ratio between the number of activities detected by the two classifiers should return back to the original $R_{old}$. If the system detects that the original detection ratio has been restored, it can go back to using the classifier trained on all nodes including node $s$.

An important thing to consider here is that small fluctuations in the ratios might occur even without failures. In addition, non-severe failures might cause very small changes to the behavior of the classifiers. Therefore, the threshold $TH_R$, which helps distinguish between sensor failures and normal fluctuations while also allowing the detection of the majority of node failures, should be carefully chosen. Section IV discusses how choosing different values for $TH_R$ affects the ability of our approach to detect sensor failures.

Another issue is that SMART might have difficulty detecting node failures that cause very small difference between $R_{old}$ and $R_{new}$. In general, this should not be a problem, since such non-severe failures will have very little effect on the application-level behavior. However, for applications where it is important that all node failures are detected, health-monitoring approaches can be used to help identify some of the node failures that SMART cannot detect.

*2) Maintaining detection accuracy under failures:* We want a system that is able to adapt itself and maintain

high activity detection accuracy even in the presence of node failures. We approach this challenge by simultaneously running more than one classifier instance. Some of the classifier instances are preemptively trained for the occurrence of failures using the *hold one sensor out* approach. At run time, based on the relative behavior of the classifiers, the system dynamically chooses the best classifier instance from the current classifier ensemble.

In addition to training multiple classifier instances, SMART also simultaneously uses different classifier classes, such as Naive Bayesian (NB), Hidden Markov Model (HMM), Hidden Semi-Markov Model (HSMM), or decision trees. SMART takes advantage of the fact that classifiers perform differently based on the current state of the system. One classifier class might be more accurate than another in situation A, and less accurate in situation B. This can occur for a number of reasons, such as the nature of the detected activity, the number and location of failures, and the network topology. At run time SMART dynamically chooses the best classifier instance among all classes and all instances that are simultaneously running in the system.

## IV. Evaluation

We use two publicly available activity detection datasets for our experiments. We study two of the houses from the first dataset [5]: House A and House B. The data from House A was collected in the course of 24 days from 14 sensors. House B contained 27 sensors and the data was collected over 13 days. Each day is divided into fixed-length timeslots and the activity performed in each timeslot is classified based on the sensor firings within that particular timeslot. The subjects living in the houses annotated a number of different activities that include *leave house*, *toileting*, *sleeping*, *prepare breakfast*, *prepare dinner*, and *showering*. We use this annotation as the ground truth. The dataset also contains NB and HMM classifiers for activity detection. Later in [18], an HSMM classifier was used with the same dataset. In our experiments we use these classifiers for the evaluation on activities from House A and House B.

The second dataset is from the PlaceLab project at MIT [6]. The home they monitor is deployed with over 900 sensors, including wired reed switches, current and water flow inputs, object and person motion detectors, and RFID tags. This dataset contains 104 hours of annotated data with examples of 43 typical house activities. The authors use two of the classifiers from the WEKA software package [19]: NB and C4.5 decision trees. We also use the WEKA classifiers in all experiments involving the PlaceLab dataset.

Although we run experiments with only four classifier classes, namely NB, HMM, HSMM, and decision trees, we are confident that other classifier classes can easily be incorporated into our approach.

The detection of simple activities, such as *leave house* and *toileting* relies on a single sensor. When that sensor fails, the corresponding activity can no longer be detected. Such a failure is considered to be severe and maintenance should always be dispatched. In our evaluation we focus on *complex* activities whose detection relies on multiple sensors. This gives us the opportunity to encounter node failures with different levels of severity. There are many complex activities that one might want to monitor, such as cooking, cleaning, studying, exercising, getting dressed, and unpacking groceries. In our experiments we evaluate the detection of the following five activities:

House A: *preparing breakfast* and *preparing dinner*
House B: *preparing breakfast* and *preparing dinner*
PlaceLab: *leisure*

Our choice of activities was constrained by the limitations of the datasets: most of the activities were simple and the fault tolerance level was low. Preparing dinner and breakfast were the only complex activities in House A. We study the same activities in House B for comparison. The PlaceLab dataset lacks annotations for many of the cooking activities and they coult not detect those very well. Therefore, we used the leisure activity which relies on 6 sensors: television, remote control, computer keyboard, and kitchen, office, and living room PIR.

The severity threshold for all of the experiments is set to $TH_S = 0.3$. We chose this particular value based on empirical analysis we performed with smaller thresholds, such as 0.1 and 0.2. When such small thresholds are used, due to the low level of redundancy in the datasets, most of the node failures are classified as severe. Therefore, we chose a threshold that allowed us to have a more diverse failure profile. In the rest of this section, a failure is considered to be severe if it decreases the original accuracy of the application by more than 30%.

The datasets do not contain failure information. All node failures in the experiments were introduced by removing the sensor data for the "failed" node from the testing set. We use *F-score* as accuracy measurement for all classifiers:

$$\text{F-score} = \frac{2 * Precision * Recall}{Precision + Recall} \quad (2)$$

*Precision* is the number of correct activity detections divided by the number of all returned activity detections. *Recall* is the number of correct activity detections divided by the number of all activity detections that should have been returned.

### A. Node failure severity

*1) Number of repairs and maintenance dispatches:* With the baseline repair technique, every time a sensor node fails a repair is necessary and maintenance has to be dispatched. Figure 4 shows the effect of our approach on the number of necessary repairs and maintenance dispatches. For this experiment we only look at the behavior of different classifier classes in isolation, i.e. SMART has one classifier class to

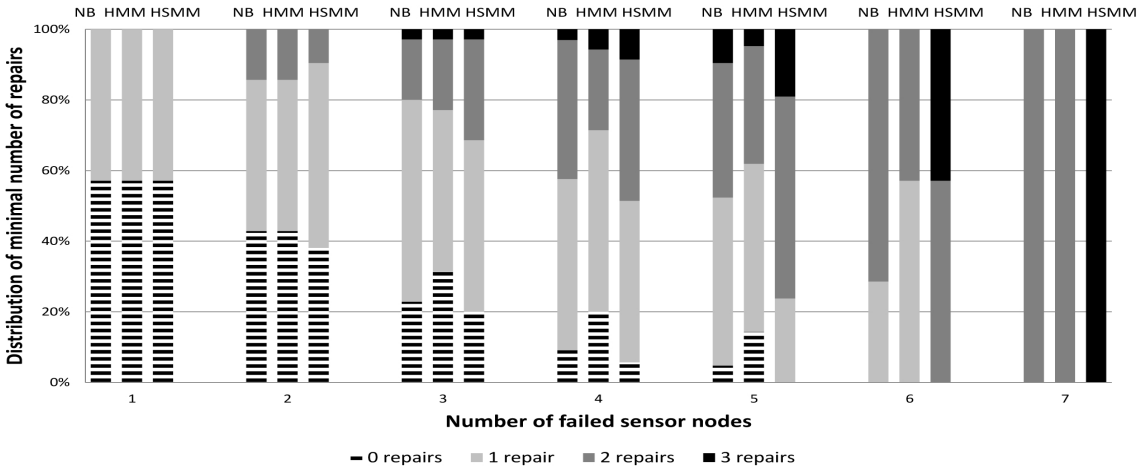


Figure 4. Distribution for minimal number of repairs for activity *prepare breakfast*. Compared to a baseline where a repair is necessary every time a node failure occurs, our approach repairs only the severe failures and thus decreases the number of repairs by 70% on average, and the number of maintenance dispatches by 45%. We evaluate three classifier classes - NB, HMM, and HSMM.

work with. This allows us to see the differences between the classifier classes. The activity shown in the figure is *prepare breakfast* from House A. The figure contains the repair distribution over all possible combinations of failures of the sensor nodes used to detect this activity.

When one of the nodes fails we need to perform a repair in only 43% of the cases. This happens because only 3 of the 7 nodes that participate in the detection of activity *prepare breakfast* have high influence on its detection accuracy. Therefore, only 3 of the single-node failures are severe. The number of necessary repairs gradually increases when more nodes fail, but always remains smaller than the number of node repairs required by the baseline approach. For example, when all 7 sensors fail, SMART determines that repairing only 2 nodes (3 in the case of HSMM) can increase the activity detection accuracy above the specified severity threshold. Calculating the average number of necessary repairs for all five activities over all classifier classes shows that our approach decreases the number of repairs by 70%.

Figure 4 also shows the decrease in the number of maintenance dispatches. A maintenance dispatch is avoided whenever no repairs are necessary. When a NB classifier is used, the number of dispatches is decreased by 57% with one sensor failure, 43% with two failed sensors, and so on. The results are similar for the HMM and HSMM classifiers.

There might be a significant difference in the number of repairs based on the classifier class being used. For example, for the *prepare breakfast* activity, introducing failures causes faster accuracy degradation for HSMM than for NB or HMM. This difference in the performance of the different classifiers becomes more prominent when $TH_S$ increases.

*2) Mean time to failure:* To evaluate our impact on the MTTF of the application we compare our approach where we use an ensemble of classifier instances some of which

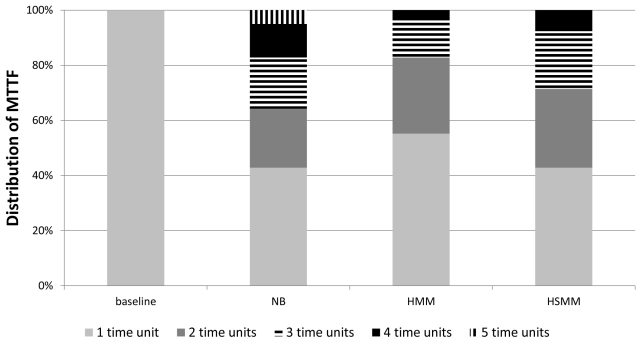


Figure 5. Mean time to failure distribution for House A *preparing breakfast*. We assume that a new sensor failure occurs after each time unit. The average MTTF for NB, HMM, and HSMM are 2.2, 2.1, and 1.9 time units respectively.

are preemptively trained for failures to the baseline where there is just one classifier instance trained on all nodes. We consider all possible sequence traces of sensor node failures. For each of those traces, we evaluate at what point in time, i.e. after how many node failures, a repair should be performed. Unlike the baseline, our approach determines that the application has failed not when the first node fails, but when the first severe node failure occurs.

Figure 5 shows the results for the MTTF for the *prepare breakfast* activity in House A. We introduce a new node failure after each time unit. With the baseline approach, since every node failure is also considered to be an application failure, the MTTF of the system is always 1 time unit. The results on Figure 5 show that when SMART uses a NB classifier, the system can sometimes survive 5 failures before it needs maintenance. This happens when all 4 non-severe node failures occur first. In that case, the application's accuracy will remain above the severity threshold until the fifth failure, which has to be severe.

| Activity | MTTF per classifier | | | | |
|---|---|---|---|---|---|
| House A breakfast | baseline 1 | NB 2.2 | HMM 2.1 | HSMM 1.9 | SMART 2.2 |
| House A dinner | baseline 1 | NB 5 | HMM 2.3 | HSMM 4.1 | SMART 5 |
| House B breakfast | baseline 1 | NB 4.1 | HMM 2.3 | HSMM 3.4 | SMART 4.1 |
| House B dinner | baseline 1 | NB 2.9 | HMM 1.9 | HSMM 1.7 | SMART 2.9 |
| PlaceLab leisure | baseline 1 | NB 2.5 | C 4.5 tree 2.5 | | SMART 2.5 |

Table I
AVERAGE MTTF FOR ALL FIVE ACTIVITIES. OUR APPROACH INCREASES THE MTTF 2.6 TIMES ON AVERAGE. THE MTTF IMPROVEMENT ACHIEVED WITH EACH CLASSIFIER VARIES WITH THE NATURE OF THE ACTIVITY. SMART CHOOSES THE MOST SUITABLE CLASSIFIER AT ANY TIME AND THUS ACHIEVES THE HIGHEST MTTF.

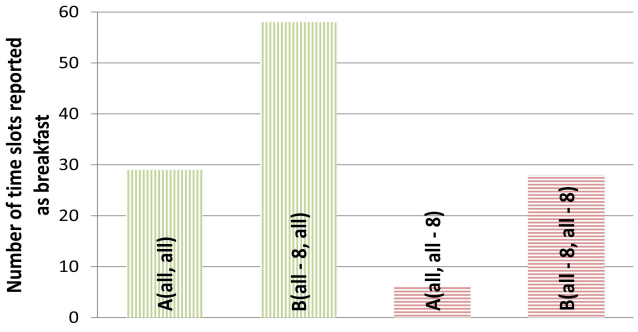


Figure 6. Number of time slots classified as *prepare breakfast* House A before and after the failure of node 8 for two NB classifiers: A and B. Classifier A is trained on all nodes while classifier B is trained by holding out node 8 from the training set. A significant change in the ratio of reported activity instances by classifier A and classifier B could indicate that this particular node has failed.

Table 1 shows the average MTTF values for all five activities. The MTTF achieved by different classifiers varies based on the activity. For example, for activity *preparing breakfast* from House A, using HMM results in much higher MTTF than when HSMM is used. However, HSMM considerably outperforms HMM for the *preparing dinner* activity again from House A. Since our SMART approach dynamically chooses the best classifier depending on the current state of the system, it maintains the highest application MTTF.

### B. Using multiple classifiers

*1) Identifying sensor node failures:* SMART detects the occurrence of sensor failures at run time by simultaneously running multiple classifier instances and monitoring changes in their relative output. Figure 6 shows how SMART identifies that the fridge node (node ID 8) in House A has failed. We use the data for the *prepare breakfast* activity and compare the performance of two NB classifiers, A and B, before and after the failure occurs. Classifier A is trained on all nodes in the system assuming no failure. Classifier B is preemptively trained for the failure of node 8 by excluding it from the training set. The first and second bars in Figure 6 show the number of time slots identified as *prepare breakfast* by classifiers A and B, respectively, before the failure. The third and fourth bars show the behavior of the two classifiers after node 8 fails. The significant difference in the ratio of the first and second bar (0.50) and the third and fourth bar (0.21) can be used to indicate that node 8 has failed.

Table 2 shows $R_{old}$ and $R_{new}$ for all nodes that affect the detection of the *prepare breakfast* activity in House A and House B. The calculated change between the two ratios shows that SMART is very successful at identifying severe failures. The nodes that have the highest influence on the detection accuracy of activity *prepare breakfast* for House A are nodes 8, 9, and 23. Similarly for House B, we can easily identify that node 3 is the most critical sensor. The difference between $R_{old}$ and $R_{new}$ is not as noticeable when the sensors are not critical for the activity. As mentioned earlier, health monitoring can be used together with our approach to help identify low-severity failures.

The results in Table 2 also reveal that failure severity is deployment-specific even when the same activity is being analyzed. There is only one sensor in the overlap between the nodes that are critical for detecting *prepare breakfast* in House A and House B. This shows that the failure severity analysis of two deployments could have very different results even if the same activities are being detected.

Choosing an appropriate $TH_R$ value that is low enough to detect node failures but also high enough to accommodate the natural fluctuations of the system is an important issue. Figure 7 shows the effect of using different $TH_R$ values on the number of node failures that our approach is capable of identifying. All activities show a similar trend, where with smaller thresholds almost all failures can be detected. As $TH_R$ increases, the number of node failures that we can detect decreases steadily. When $TH_R$ reaches 30% the only failures that our approach can detect are the severe ones. Therefore, based on this empirical analysis a suitable value for $TH_R$ for the activities we examine would be close to 15%. It will allow us to 1) detect all severe failures; 2) detect additional non-severe failures; 3) avoid interpreting natural system fluctuations as node failures. Additional experiments have shown that the effect of $TH_R$ on the failure detection accuracy of HMM and HSMM is very similar to that for NB. Since the relative behavior of the classifiers may vary based on the classifiers that are used and the nature of the detected activities, choosing an appropriate general value for $TH_R$ may not always be possible. However, the results from the offline node failure severity analysis can help in determining suitable general or activity-specific value(s) for $TH_R$.

*2) Maintaining high classification accuracy under failure:* Figure 8 shows the relative accuracy in the presence of failures achieved by NB, HMM, and HSMM classifiers that were preemptively trained for failures. The results are normalized to the accuracy of a baseline NB classifier trained on all nodes when there are no failures in the system. Since SMART picks the classifier providing the highest detection

| House A : prepare breakfast | | | | | House B : prepare breakfast | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **ID** | **Location** | $R_{old}$ | $R_{new}$ | **Change** | **ID** | **Location** | $R_{old}$ | $R_{new}$ | **Change** |
| 1 | microwave | 0.83 | 0.83 | 0% | 3 | fridge | 1.45 | 0.63 | 57% |
| 7 | cups cabinet | 0.83 | 1.00 | 21% | 9 | plates cabinet | 0.99 | 1.00 | 1% |
| 8 | fridge | 0.50 | 0.21 | 51% | 14 | cutlery drawer | 0.96 | 0.99 | 3% |
| 9 | plates cabinet | 0.73 | 0.30 | 59% | 15 | stove lid | 1.16 | 0.92 | 21% |
| 17 | freezer | 0.73 | 0.94 | 29% | 26 | toaster | 1.19 | 0.93 | 22% |
| 18 | pans cabinet | 0.94 | 1.19 | 27% | 27 | microwave | 1.11 | 0.97 | 13% |
| 23 | groceries cabinet | 0.71 | 0.35 | 51% | 28 | motion sensor | 1.16 | 1.27 | 9% |

Table II
CHANGE BETWEEN $R_{old}$ AND $R_{new}$ FOR THE *prepare breakfast* ACTIVITY IN HOUSE A AND HOUSE B WHEN SINGLE NODE FAILURES ARE INTRODUCED IN THE SYSTEM. IF A NODE HAS A SIGNIFICANT INFLUENCE ON THE DETECTION ACCURACY OF AN EVENT, ITS FAILURE CAUSES A NOTABLE CHANGE IN THE RATIOS OF DETECTED ACTIVITIES BY THE TWO DIFFERENTLY TRAINED CLASSIFIERS.

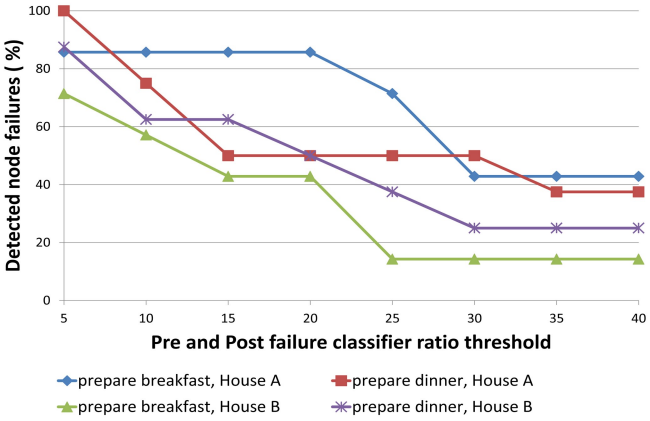


Figure 7. Number of failures that our approach identifies using different values for $THR$ for a NB classifier. The increase of $THR$ limits the number of failures we can identify.

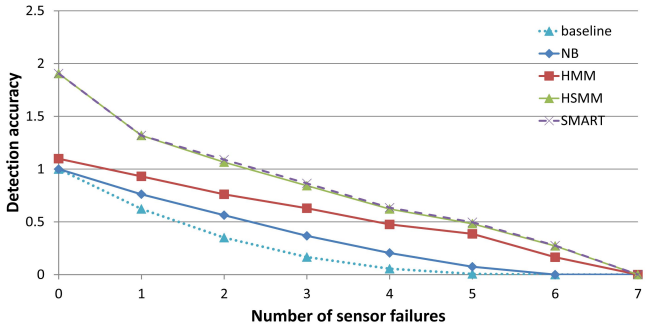


Figure 8. Detection accuracy for *preparing breakfast* in House A for NB, HMM, HSMM, and SMART when the classifiers are trained for the occurrence of failures. The results are normalized to the accuracy of a baseline NB classifier trained on all nodes.

accuracy, the SMART curve overlaps with that for HSMM.

The next experiment evaluates the accuracy improvement achieved by the classifiers trained for failures over the classifiers that were trained on all nodes. For example, we compare the accuracy of NB that is preemptively trained by holding nodes out to that of NB trained on all nodes in the system. Figure 9 shows the results for NB, HMM, HSMM, and SMART for the *prepare breakfast* activity in House A.

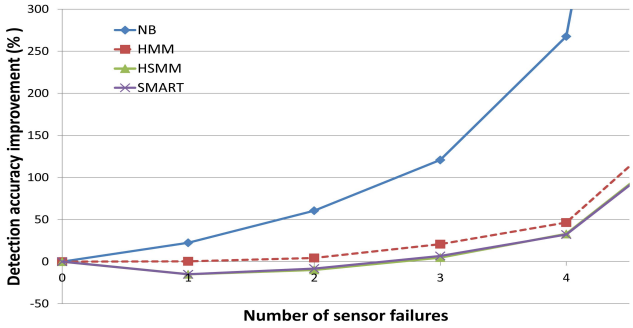


Figure 9. Using classifiers that are preemptively trained for failures instead of ones trained on all nodes in the system improves the detection accuracy for *preparing breakfast* in House A. The curve for SMART overlaps with that for HSMM since HSMM provides the highest detection accuracy.

The highest gain is achieved with the NB classifiers. For example, when 3 nodes fail, using a NB classifier that is trained for these failures instead of a NB classifier trained on all nodes improves the detection accuracy by more than 100%. The curve for SMART again overlaps with that for HSMM because of HSMM's accuracy. When more than four node failures have occurred, the accuracy improvement of our approach grows exponentially since the accuracy of the original classifiers trained on all nodes becomes close to 0.

Table 3 shows the average accuracy improvement for all five activities when failure-trained classifiers are used. The level of improvement for a classifier differs based on the activity. For example, for *preparing breakfast* in House A, NB shows the highest improvement while for the same activity in House B SMART achieves almost double the improvement of NB. Interestingly, using the failure-trained classifiers leads to very small accuracy improvement for the *leisure* activity. However, even if our approach does not always increase the detection accuracy under failures, it can still be used to detect node failures and decrease the MTTF of the application and the number of necessary repairs.

## V. CONCLUSIONS AND DISCUSSION

We presented a general repair assessment approach for failures in activity detection applications. Even though the

| Activity | Accuracy improvement | | | |
|---|---|---|---|---|
| House A breakfast | NB 307% | HMM 50% | HSMM 33% | SMART 33% |
| House A dinner | NB 239% | HMM 13% | HSMM 15% | SMART 262% |
| House B breakfast | NB 12% | HMM 20% | HSMM 34% | SMART 9% |
| House B dinner | NB 65% | HMM 79% | HSMM 58% | SMART 112% |
| PlaceLab leisure | NB 1% | C 4.5 decision tree 0% | | SMART 0% |

Table III
WE EVALUATE HOW USING CLASSIFIERS TRAINED FOR FAILURES INSTEAD OF ONES TRAINED ON ALL NODES AFFECTS THE DETECTION ACCURACY OF THE SYSTEM. THE AVERAGE ACCURACY IMPROVEMENT UNDER THE PRESENCE OF FAILURES FOR ALL FIVE ACTIVITIES WE EVALUATE IS 70.6%.

datasets used in our evaluation do not directly address fault tolerance or redundancy, our approach still achieves significant gains. SMART decreases the number of maintenance dispatches by 45% and almost triples the MTTF of the application on average. It also maintains sufficient activity detection accuracy in the presence of failures by dynamically updating the classifiers at run time so they can adapt to the failures that occur. SMART detects all severe sensor failures that affect the application-level behavior and over 60% of all sensor failures at run time. Further, SMART improves the activity detection accuracy under the presence of failures by more than 70% on average.

In a deployment where there is absolutely no node redundancy in the system and all of the activities of interests are simple, i.e. for each activity there is just one designated sensor whose readings can be used to infer the occurrence of that activity, SMART performs just as well as the best classifier in its classifier ensemble. However, in such a scenario all node failures are severe and maintenance is needed after each failure. Therefore, SMART will not improve the MTTF of the application or the number of maintenance dispatches, but it can be used to detect fail-stop or transient node failures.

As long-lived sensor network applications become more common in real homes, the need for fault tolerance and ground truth validation will lead to increased node redundancy. We expect that applying SMART to moderately and highly redundant systems will result in much higher gains. In the future, we plan to evaluate SMART on a deployment designed with better fault tolerance features. We would also like to integrate our approach with the design of an activity recognition system and evaluate how providing appropriate node redundancy at design time can further aid fault tolerance, reduce dispatches, and improve accuracy.

## REFERENCES


[1] T. M. Mitchell, "Machine learning and data mining," *Communications of the ACM*, vol. 42, 1999.

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

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

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

[5] T. van Kasteren, A. Noulas, G. Englebienne, and B. Kröse, "Accurate activity recognition in a home setting," in *UbiComp*, 2008.

[6] 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.

[7] L. Ruiz, J. Nogueira, and A. Loureiro, "MANNA: A management architecture for wireless sensor networks," in *IEEE Communications Magazine*, February 2003.

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

[9] Y. Wu, K. Kapitanova, J. Li, J. A. Stankovic, S. H. Son, and K. Whitehouse, "Run time assurance of application-level requirements in wireless sensor networks," in *IPSN*, 2010.

[10] U. Sammapun, I. Lee, O. Sokolsky, and J. Regehr, "Statistical runtime checking of probabilistic properties," in *RV*, 2007.

[11] U. Sammapun, I. Lee, and O. Sokolsky, "Rt-mac: runtime monitoring and checking of quantitative and probabilistic properties," in *RTCSA*, 2005.

[12] N. Ramanathan, T. Schoellhammer, E. Kohler, K. Whitehouse, T. Harmon, and D. Estrin, "Suelo: human-assisted sensing for exploratory soil monitoring studies," in *SenSys*, 2009.

[13] D. Partridge and W. Yates, "Engineering multiversion neural-net systems," *Neural Computation*, vol. 8, 1996.

[14] F. Roli, G. Giacinto, and G. Vernazza, "Methods for designing multiple classifier systems," in *MCS*, 2001.

[15] G. Giacinto and F. Roli, "Dynamic classifier selection," in *MCS*, 2000.

[16] E. M. Tapia, S. Intille, and K. Larson, "Activity recognition in the home using simple and ubiquitous sensors," in *Pervasive Computing*, ser. Lecture Notes in Computer Science, 2004.

[17] M. Philipose, K. Fishkin, M. Perkowitz, D. Patterson, D. Fox, H. Kautz, and D. Hahnel, "Inferring activities from interactions with objects," *IEEE Pervasive Computing*, vol. 3, 2004.

[18] T. van Kasteren, G. Englebienne, and B. Kröse, "Activity recognition using semi-Markov models on real world smart home datasets," *J. Ambient Intell. Smart Environ.*, 2010.

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," in *SIGKDD Explorations*, vol. 11, 2009.