Fall-curve: A novel primitive for IoT Fault Detection and Isolation

Tusher Chakraborty, Akshay Uttama Nambi, Ranveer Chandra, Rahul Sharma, Manohar Swaminathan, Zerina Kapetanovic, Jonathan Appavoo
Microsoft Research
t-tucha,t-snaksh,ranveer,t-rasha,manohar.swaminathan,v-zekap,t-joappa@microsoft.com

ABSTRACT

The proliferation of Internet of Things (IoT) devices has led to the deployment of various types of sensors in the homes, offices, buildings, lawns, cities, and even in agricultural farms. Since IoT applications rely on the fidelity of data reported by the sensors, it is important to detect a faulty sensor and isolate the cause of the fault. Existing fault detection techniques demand sensor domain knowledge along with the contextual information and historical data from similar near-by sensors. However, detecting a sensor fault by analyzing just the sensor data is non-trivial since a faulty sensor reading could mimic non-faulty sensor data. This paper presents a novel primitive, which we call the Fall-curve – a sensor's voltage response when the power is turned off – that can be used to characterize sensor faults. The Fall-curve constitutes a unique signature independent of the phenomenon being monitored which can be used to identify the sensor and determine whether the sensor is correctly operating.

We have empirically evaluated the Fall-curve technique on a wide variety of analog and digital sensors. We have also been running this system live in a few agricultural farms, with over 20 IoT devices. We were able to detect and isolate faults with an accuracy over 99%, which would have otherwise been hard to detect only by observing measured sensor data.

CCS CONCEPTS

• Computer systems organization → Embedded and cyber-physical systems; **• Hardware → Fault tolerance**;

KEYWORDS

IoT system; Fault detection; Fault isolation; Sensor identification; Analog and Digital sensors, Reliability

ACM Reference Format:
Tusher Chakraborty, Akshay Uttama Nambi, Ranveer Chandra, Rahul Sharma, Manohar Swaminathan, Zerina Kapetanovic, Jonathan Appavoo. 2018. Fall-curve: A novel primitive for IoT Fault Detection and Isolation. In *The 16th ACM Conference on Embedded Networked Sensor Systems (SenSys '18), November 4–7, 2018, Shenzhen, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3274783.3274853

1 INTRODUCTION

Internet of Things (IoT) systems gather data from various sensors, which is processed to provide unique insights from the IoT deployment. These sensors are often battery powered, rely on low-cost components, and might be deployed in harsh environments. Given the likelihood of sensor failures, a key challenge in the design of IoT systems is ensuring the integrity, accuracy, and fidelity of data reported by the sensors.

To identify sensor failures, existing schemes typically use a data-centric, rule-based approach [27, 28, 34, 37, 44]. These schemes detect anomalies in the reported sensor data (see Table 1) for fault detection. However, such an approach has inherent limitations. First, faulty sensor data can mimic non-faulty data. For example, sensor data obtained when an "open" ADC/ground connection of a sensor mimics non-faulty sensor data. We observed this phenomenon in several instances in our real world deployment (see Section 2). Second, an anomalous sensor reading is often not enough to identify the root cause of the sensor failure. For example, an incorrect reading could be caused by a bad sensor, low battery, or an error with the microprocessor, among other factors. The capability to isolate the faulty component is especially important for IoT deployments where the field staff might have limited technical background, and it might be expensive for staff with deep technical expertise to reach remote areas, for example, in oil and gas, agriculture, mining, forestry, and other verticals.

In this paper, we address the above challenges using a key innovation, called the *Fall-curve*. When a sensor is powered off it continues to output analog signal for a short period of time, primarily due to the parasitic capacitance of the sensor board. These values decay in a characteristic "Fall-curve". Our key observation is that a sensor's Fall-curve constitutes a unique signature and is agnostic to the phenomena being monitored (See Figure 3). Hence, we can develop a Fall-curve based technique to identify the sensor connected to an IoT device and determine whether the sensor is correctly operating.

In a typical operation, the Fall-curves are sampled either periodically, or on-demand in response to anomalous data. If there is a suspected error in the reported readings, Fall-curve is used to identify the likely cause of the error. Using several measurements, we have shown that the above schemes, including the algorithms to match Fall-curve, consume extremely low-power, and can be run locally on the IoT device itself with less than 0.3% energy overhead. We have implemented our approach in a system that includes variety of analog and digital sensors. The system has been running live in a few agricultural farms, with over 20 IoT devices (having up to 4 sensors each) that are monitoring soil properties for more than 3 months. In this setup, we were able to identify sensors faults with an accuracy over 99.13%.

We note that the proposed Fall-curve based technique significantly improves the state-of-the-art in sensor fault detection and isolation in IoT systems. The Fall-curve provides a way to identify faults by shutting the power off to the sensor, and thus it is independent of the sensing environment. Hence, it is able to identify faulty sensors, even when the reported readings are similar to real-world data. Finally, the Fall-curve based technique is able to isolate the cause of the fault without requiring additional hardware, or spatiotemporal correlation across multiple physical sensors. It also works across a range of sensors and IoT environments.

Through this work, we make the following contributions:

- We introduce the concept of a Fall-curve as a way to characterize a sensor, including theoretical analysis and experimental evaluation over a range of sensors.
- We show that Fall-curves can identify faulty sensors, and show how they can be used for both analog and digital sensors, over a wide variety of sensors.
- We demonstrate the efficacy of the proposed technique for detecting faults in a real-world agricultural IoT deployment, where more than 20 IoT devices have been deployed for more than three months.

2 IoT DEVICE AND ITS FAULTS

A data fault refers to the data reported by an IoT device that is inconsistent with the measured phenomenon's true behavior. Prevalent research efforts have analyzed sensor data to detect faults from various IoT deployments. Table 1 outlines the most common sensor data faults observed in IoT deployments: short, spike, stuck-at, noise, and calibration [34].

One possible way to detect these faults is using a data-centric approach, that extracts features based on the characteristics of sensor data to define expected behavior for both faulty and non-faulty sensor data [27, 34]. Due to the diverse and unconstrained nature of IoT deployments, detecting faults requires significant domain knowledge, historical data, and contextual information, and eventually, often requiring human intervention to isolate the faults [27, 28, 34]. Table 1 (Columns 3 & 4) describes the prerequisites for detecting faults in an IoT device using the data-centric approaches.

This necessitates going beyond data-centric approaches for fault detection and isolation. Here, the hypothesis is that all the data faults encountered in an IoT system can be mapped to a set of hardware component failures in an IoT device [27, 34]. The core components of a typical IoT device are (i) microcontrollers, (ii) sensors, (iii) analog-to-digital converter (ADC), (iv) connectivity module, and (v) batteries. Based on the extensive empirical evaluation and state-of-the-art studies, we present a mapping between sensor data faults to probable system-centric faults as shown in Table 1 (Column 5). It can be seen that most sensor data faults can be mapped to a finite set of IoT system components[1] such as sensor, battery, microcontroller, and ADC.

[1]Here, we do not consider missing data faults which are mostly related to the communication part in a sensor network deployment.

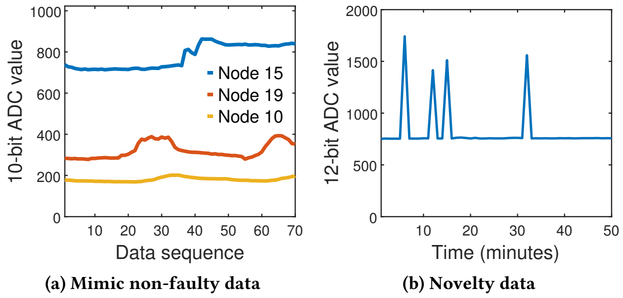


Figure 1: Challenges in accurately detecting faults.

Sensor data faults due to failure in a microcontroller, ADC, and/or battery can be determined by actively probing the system components. For example, faults in a microcontroller such as the swiss chess problem in SRAM, GPIO writing delay, interrupt failure, etc., can be determined using real-time software-based self-testing (SBST) technique [7]. SBST is a classical method for testing different modules by exploiting a predefined test program, wherein the microcontroller executes the test program. Similarly, low battery voltage, as well as battery fluctuation, is one of the prevalent causes of sensor data faults in an IoT deployment. Faults induced due to battery fluctuations can be determined by analyzing whether the response of a sensor to a battery voltage fluctuation is within a permissible range (typically described in the sensor datasheet).

While the above causes do occur, *sensor failures* are often the most common causes of sensor data faults in IoT deployments. Sensor failures can be transient or permanent due to malfunction of sensor hardware circuity, physical damage to the sensor, exposure to the harsh environment, etc. Detecting sensor data faults is challenging using a data-centric approach. Consider a sensor deployment in an agricultural farm to monitor the soil moisture value at different soil depths without having any prior data. In this case, the spatial correlation of sensor data is not effective given the heterogeneous nature of the field. Further, temporal correlation requires irrigation timing information and rainfall detection, which are loosely correlated across the field due to the diversity in moisture sensing depth, soil texture, and composition. Thus, it is impossible to model sensor data without significant domain and contextual information.

Figure 1(a) represents a scenario where faulty sensor data mimics non-faulty sensor data in our real-world deployment. In this figure, we depict soil moisture sensor values from three different IoT devices deployed in the farm. The devices are time synchronized and sensor data from all three sensor nodes are in the expected range. However, from manual field investigation, we found an open ADC wire connection fault in Node 19. Here, the open ADC was acting as an antenna which was picking a signal from the PCB traces and generating values which mimic real sensor data. A similar fault can also be observed in case of an open ground connection. Hence, it is crucial to accurately monitor the condition of the sensor.

Similarly, Figure 1(b) shows data from an LDR sensor deployed in a room. It can be seen that the sensor data exhibits periods where light values are significantly high. These regions correspond to the abrupt changes in room-lighting. While traditional data-centric models might raise a false alarm here, our investigation identified

Table 1: Fault categorization in IoT deployments using data-centric approaches and the corresponding system-centric causes.

Fault	Definition	Prerequisites for detection	Further investigation required?	Possible system-centric causes
Short	A data point that significantly deviates from the expected temporal or spatial trend of the data	Domain knowledge of sensor readings, spatiotemporal context, sensor redundancy, historical data	Yes	Battery fluctuation, transient sensor malfunction, loose wire connection, etc.
Spikes	A rate of change much greater than expected over a short or longer period of time which may or may not return to normal afterwards	Domain knowledge of sensor readings, spatiotemporal context, sensor redundancy, historical data, human intervention, correlation	Yes	Low battery, ADC failure, trigger failure, short-circuit, loose wire connection, control unit fault, etc.
Stuck-at	A series of data points having zero or almost zero variation for a period of time greater than expected	Domain knowledge of sensor readings, spatiotemporal context, sensor redundancy, historical data, spatial correlation	Yes	ADC failure, trigger failure, control unit fault, short-circuit, open connection, clipping, sensor malfunction etc.
Noise	Sensor data exhibiting an unexpectedly high amount of variation over temporal domain	Spatiotemporal context, prior knowledge, correlation, human intervention, sensor redundancy	Depends on context	Battery fluctuation, open connection, Trigger failure, sensor malfunction, etc.
Calibration	Sensor data may have offset or have a different gain from the ground truth values	Domain knowledge of sensor readings, sensor redundancy, spatial correlation	Yes	Sensor drift, ADC drift, battery offset, control unit fault, sensor malfunction, etc.

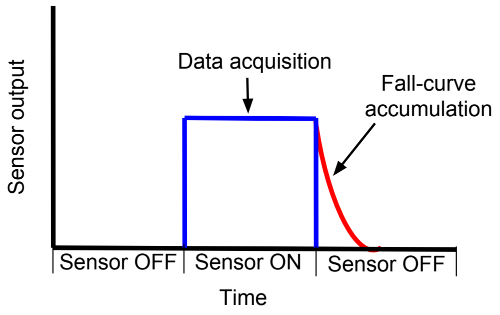


Figure 2: Overview of sensor Fall-curve accumulation.

these as valid data. Hence, fault detection techniques need to be robust to isolate novel data from faulty data.

We present a novel Fall-curve based technique to reliably detect and isolate sensor faults thus addressing the above challenges in Section 3. Further, in Section 5 we describe Fall-curve processing on the IoT device to determine the operation of the sensor (faulty or non-faulty) in real-time.

3 FALL-CURVE

As described in the previous section, data-centric approaches fail to reliably detect faults associated with the sensors. For example, an analog sensor having a hardware malfunction continues to generate a faulty analog signal and can mimic non-faulty data. Hence, it is challenging to distinguish between a faulty and non-faulty sensor data. In this section, we propose a technique to detect faults in analog and digital sensors with a novel primitive – Fall-curve.

It is common practice in IoT systems to power on sensors only for the period of data collection in order to save energy. Typically, when the sensor is turned off the output signal takes a certain amount of time before going down to zero, which is known as Fall-time [29]. This Fall-time is primarily due to parasitic capacitance in a sensor circuit that exists between the parts of an electronic component or PCB traces because of their proximity to each other.

During the powered ON state of a sensor, these parasitic capacitors charge and when the power is turned off, they start discharging over the circuit of the sensor. Consequently, the sensor's voltage response when the power is turned off goes down to zero following a curve, which we define as the "Fall-curve". The characteristic of the Fall-curve also depends on the electronic components and circuitry of a sensor. To measure the Fall-curve, we continue to sense the ADC reading for a short period of time after turning off the sensor. Figure 2 shows the overview of Fall-curve accumulation after turning off the sensor.

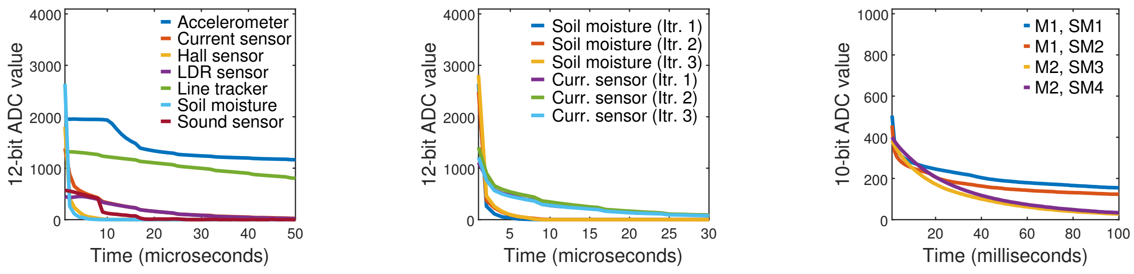
3.1 Fall-curve characteristics

To study the characteristics of Fall-curve across different sensors, we conducted several testbed experiments with over 20 different sensor types. These sensors are commonly used in diversified IoT applications, such as agriculture, robotics, air quality monitoring, smart home, etc. The set of sensors includes soil moisture (from different manufacturers), soil temperature, ambient temperature, accelerometer, current sensor, ambient light sensor (LDR), different gas sensors (MQ series), obstacle detector, heart rate sensor, sound sensor, vibration sensor, hall effect sensor, line tracker sensor, etc. Here, to accumulate a Fall-curve through ADC, we used two different microcontroller-based development boards: ATmega328P based Arduino Uno [3] and ARM Cortex M4 based Tiva TM4C123G [15]. Figure 3 shows the experimental results and depicts the characteristics of the Fall-curve for various sensors.

Characteristic-1: Uniqueness. Figure 3a shows Fall-curves for 7 non-faulty sensors. The x-axis indicates the time in microseconds and the y-axis shows the sensor value obtained from a 12-bit ADC. It can be seen that each sensor has its own unique Fall-curve due to the varying parasitic capacitance and circuitry in the sensor. Thus, each sensor has a unique Fall-curve signature that can be leveraged to identify the sensor attached to the microcontroller.

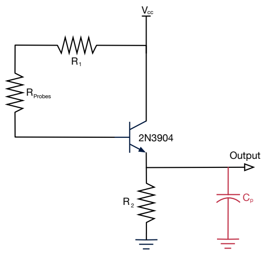
Characteristic-2: Independent of the environment or sensing phenomenon. As described in Figure 2, the Fall-curve is accumulated only after the sensor is turned off, and hence, the Fall-curve data collected should be agnostic to the phenomena being monitored. To evaluate this we analyzed the Fall-curve of a sensor at different sensor values. For example, we accumulated Fall-curve of a soil moisture sensor when it was measuring moisture content at 80%, 65%, and 55%, respectively. Figure 3b shows the Fall-curves of a soil moisture sensor for different soil moisture values. The Fall-curves of the same soil moisture sensor has similar characteristics, even though, the measured moisture contents were different in each iteration (Itr). Figure 3b also shows Fall-curves of a current sensor with different measured current values.

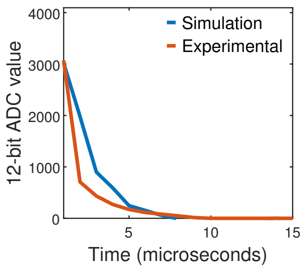
Characteristic-3: Manufacturer dependent. The circuitry associated with a sensor type varies from one manufacturer to another. This variation is due to the use of different resistor/capacitor values, dissimilar component models (part numbers), circuitry manufacturing variations, etc. Consequently, leading to a varying parasitic capacitance of a sensor circuit. Thus, the Fall-curves of the same



(a) Uniqueness **(b) Independent of environment** **(c) Manufacturer dependent**

Figure 3: Fall-curve characteristics across various sensors.





(a) Soil moisture sensor **(b) Fall-curve comparison**

Figure 4: Theoretical analysis of the Fall-curve.

sensor from different manufacturers should be distinguishable. To evaluate this, we performed an experiment involving two soil moisture sensors from each of two different manufacturers. Figure 3c shows the Fall-curves for various soil moisture sensors, where SM1, SM2 are from Manufacturer 1 (M1) and SM3, SM4 are from Manufacturer 2 (M2). It can be seen that Fall-curves of SM1/SM2 from M1 has a distinguishable characteristic compared to SM3/SM4 from M2. Thus, the same sensor from different manufacturers has a distinct Fall-curve. Furthermore, two sensors from the same manufacturer have similar Fall-curves (as seen in Figure 3c where Fall-curves of SM1 and SM2 resemble each other).

3.2 The underlying physics

In the previous section, we described the key characteristics of a Fall-curve based on experimental observations. In order to validate and generalize these observations, we use a simulator to characterize the sensor behavior. First, we derive the sensor circuitry using the sensor's datasheet (from the manufacturer) to determine the parasitic components. We then feed this circuitry to the SPICE simulator [21] to extensively study the Fall-curve characteristics.

To this end, we perform an extensive analysis of the Fall-curve for a resistive soil moisture sensor, *viz.,* SEN13322 [36]. Figure 4a shows the circuit of the SEN13322 sensor (from its datasheet) that consists of a transistor (2N3904), two resistors ($R1$, R_2), and two resistive probes represented as a single resistance across their outputs (R_{probes}). Given there exists no capacitance in the sensor circuitry, we would expect the Fall-curve to drop down immediately to zero. We can represent this by deriving the equation for the output voltage of the sensor. To do so, we must first analyze the base current (I_B) and emitter current (I_E) of the transistor, $I_B = \frac{V_{cc}-0.6}{(R_1+R_{probes})+(1+\beta)*R_1}$ and $I_E = I_B * (\beta + 1)$, where β is the transistors current gain and V_{cc} is the input voltage.

Using these two equations, we can now derive the equation for the output voltage,

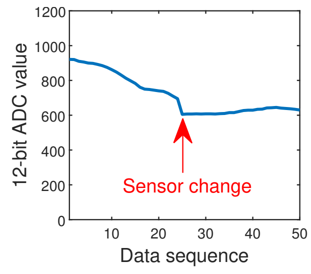
$$V_{out} = R_2 * \left(\frac{(V_{cc} - 0.6) * (1 + \beta)}{(R_1 + R_{probes}) + (1 + \beta) * R_1}\right) \qquad (1)$$

The soil moisture sensor can experience a variety of changes, for example, the resistance of R_{probe} changing due to the moisture content of the soil. Hence, to analyze this circuit extensively we use a SPICE simulator [21]. To ensure we have an accurate simulation, we use the same component values that are on the physical circuit, where $R_1 = 100\Omega$, $R_2 = 10k\Omega$, and a 2N3904 Bi-Polar NPN transistor are used. To set R_{probes}, we measured the resistance across the resistive probes when under wet conditions, which came out to be approximately 1500kΩ. The voltage supply was set to be a square wave at 5V.

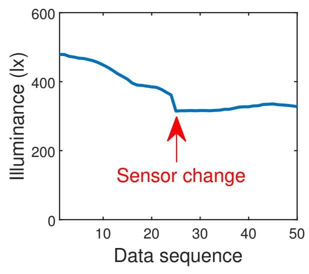
In our simulations, we observed that Fall-curve drops immediately down to zero, as expected. However, the experimental results showed a delay in the Fall-curve, implying that a small amount of capacitance is present at the output terminal of the sensor. In order to justify the experimental result for the Fall-curve, we analyze the parasitic capacitance of the circuit. For this sensor, the parasitic capacitance from the copper trace at the output terminal is approximately 30pF (found after extensive evaluation). Using the SPICE simulator, we included this parasitic capacitance at the output terminal, shown in red in Figure 4a, and re-evaluated the Fall-curve. When the parasitic capacitance is considered, the Fall-curve characteristics of the soil moisture sensor in simulation and experiments align quite well as seen in Figure 4b. This can be also represented by multiplying Eq. 1 by e^{-t/C_p}. Thus Fall-curves of any sensor can be modeled by analyzing the sensor circuity obtained from the datasheet.

4 SENSOR IDENTIFICATION AND FAULT DETECTION USING FALL-CURVE

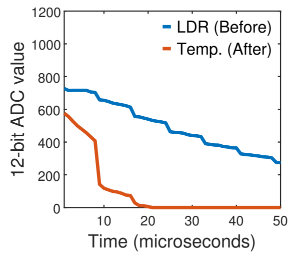
In Section 3.1 we discussed the three key characteristics of the Fall-curve *viz.,* uniqueness, independent of the environment, and manufacturer dependent. In this section, we first present how to identify a sensor connected to a microcontroller using the Fall-curve characteristics. We then describe how to reliably detect and isolate sensor faults in an IoT deployment.



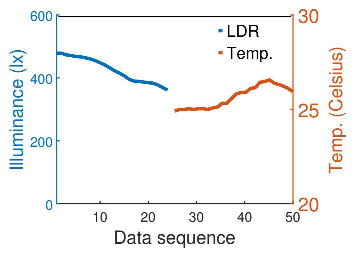
(a) Raw data of light sensor



(b) Human-interpretable light data



(c) Fall-curves



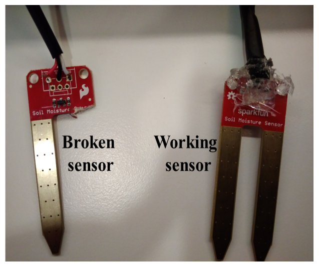
(d) Human interpretable data after applying correct processing formula

Figure 5: Sensor port mismatch and identification.

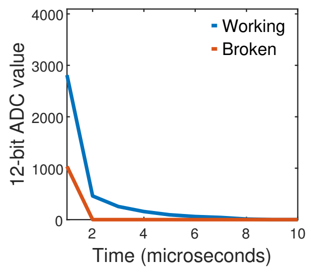
4.1 Sensor identification

A typical IoT device consists of multiple ports to which various sensors are connected. The measured raw sensor data at each port is then processed either locally or transmitted to the cloud for human-interpretability. The processing formula for this raw sensor data is dependent on the sensor type and the manufacturer. For example, two temperature sensors from different manufacturers can have different processing formulas. Hence, if one mistakenly connects a sensor to a wrong port (during sensor attachment or replacement), the IoT device still continues to get raw data from the sensor, however, incorrectly processed due to the application of wrong processing formula. This necessitates knowing the $< sensor, port >$ information, i.e., the port to which a particular sensor is connected, for correctly processing the raw sensor data.

Sensor identification problem can be solved by utilizing two key characteristics of the sensor Fall-curve – uniqueness and manufacturer dependent. These properties ensure each sensor type has a unique Fall-curve, which can be used to identify the sensor attached to a port. To show the effectiveness of sensor identification, we performed an experiment using an IoT device that is monitoring ambient light and temperature every 15 minutes in a building. Figure 5a shows the analog sensor value of a light sensor attached to an IoT device on Port 1. The x-axis indicates the data sequence where each point indicates 15 minutes time interval. Figure 5b shows the corresponding human-interpretable sensor data in illuminance (lx) after applying the sensor-specific processing formula (obtained from the datasheet). After some time period, we physically replaced the light sensor with a temperature sensor, specifically at data sequence 24 as shown in Figure 5a. In this scenario, one cannot determine if there is a fault in the sensor data as the data is well within the permissible range of the light sensor (see Figure 5b). In such cases, the proposed Fall-curve based technique can identify the sensor attached to the port as the Fall-curves obtained before and after sensor change should be significantly different. Figure 5c shows distinctive Fall-curves before (LDR light sensor) and after (temperature sensor) the sensor change. Thus, we can reliably detect sensor port change. Then, we can either raise an alert or automatically detect the new sensor connected to the port and locally adapt the processing formula. Figure 5d shows the human-interpretable sensor data after automatically applying the correct processing formulas upon detection of sensor change.

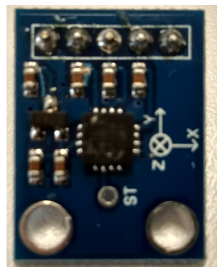


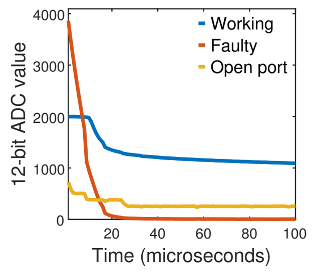
(a)



(b)

Figure 6: Manually injected faults: Soil moisture.


(a)



(b)

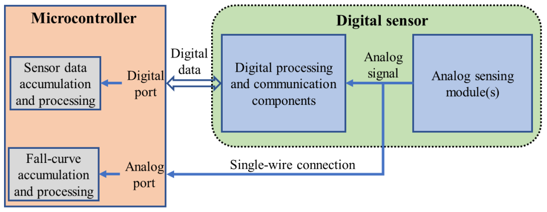
Figure 7: Manually injected faults: Accelerometer.

4.2 Sensor fault detection

There are two main types of sensors: analog and digital sensors. Analog sensors generate a continuous output signal which is typically proportional to the value being measured. They interface with a microcontroller through an ADC. Digital sensors produce a discrete digital output signal and interface with the microcontroller using a digital communication protocol, e.g., I2C, RS485, SPI, etc. In this section, we describe how we can leverage Fall-curve to accurately detect faults in analog and digital sensor.

4.2.1 Analog sensor fault detection. Based on the discussion so far, we can deduce that any permanent or transient hardware malfunction in a sensor results in a change of Fall-curve shape. We use this phenomenon to detect non-faulty and faulty sensors. To evaluate this, we conducted several experiments where we manually injected faults in the sensor by physically damaging sensors or exposing to heat, or passing current with high voltage, etc., which occurs commonly in an IoT deployment.

Figure 6a shows a broken (faulty) and working soil moisture sensor, and Figure 6b shows the corresponding Fall-curves. We can see that Fall-curve for a broken sensor is significantly different from that of a working sensor, which can be used to determine faulty sensors. Figure 7a shows a faulty accelerometer sensor where we manually applied a high voltage beyond its tolerance limit. In



Microcontroller
Sensor data accumulation and processing
Digital port
Fall-curve accumulation and processing
Analog port
Digital data
Digital sensor
Digital processing and communication components
Analog signal
Analog sensing module(s)
Single-wire connection

Figure 8: Fall-curve accumulation in digital sensor.

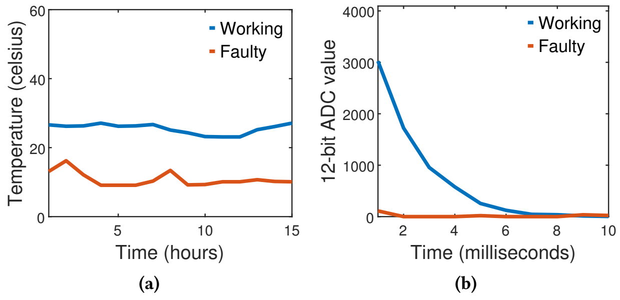
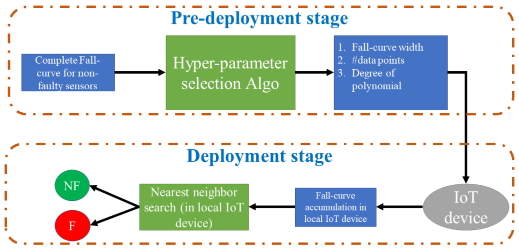


Figure 9: Non-faulty vs faulty digital DHT sensor.



Pre-deployment stage
Complete Fall-curve for non-faulty sensors
Hyper-parameter selection Algo
1. Fall-curve width
2. #data points
3. Degree of polynomial
Deployment stage
IoT device
Fall-curve accumulation in local IoT device
Nearest neighbor search (in local IoT device)
NF
F

Figure 10: Block diagram of Fall-curve processing pipeline.

Figure 7b, we see the contrast between the Fall-curves of a faulty and non-faulty accelerometer. Note that, in both of the scenarios, the damaged/faulty sensor was generating some sensor data in the valid data range.

Finally, we also evaluate a scenario where the port is open and the IoT device still reads some data (port acting as an antenna and picking some signal from the PCB traces). Figure 7b shows distinctive Fall-curves when the ADC port is open and when the port is connected to a working sensor. Thus, Fall-curve can identify faults in analog sensors without any additional hardware or contextual information about the IoT sensor deployment.

4.2.2 *Digital sensor fault detection.*

A digital sensor has two main components – digital block and an analog block. The digital block contains a digital data processing and communication unit, memory, and internal ADC. The analog block contains one or multiple analog sensing modules. The measurement of sensing phenomenon begins as an analog signal, which is then converted into a digital signal through ADC in the digital block and transmitted to the external microcontroller.

In a faulty digital sensor, any or both of the blocks can fail. If the digital block of the sensor is faulty, it can easily be detected as the sensor stops responding or transmits default out-of-service value. However, if the analog block is faulty, the digital block continues to process and transmit the faulty signal from the analog sensing module(s). While the Fall-curve based fault detection of the analog sensor is possible, the external microcontroller does not have access to the analog block of the digital sensor. To this end, we propose a single-wire connection between the analog sensing module and the ADC port of the external microcontroller.

Figure 8 depicts a block diagram of a microcontroller and digital sensor interfacing along with the proposed single-wire connection. Note that this single-wire connection *does not* have any influence on off-the-shelf digital sensor data communication. The single-wire connection is only utilized to accumulate Fall-curve when the sensor is turned off. Thus, we can detect faults in a digital sensor at both, (i) digital block: checking for response from the digital block and (ii) analog block: by accumulating the Fall-curve.

Figure 9a shows an instance of faulty and non-faulty DHT sensor [25] where the x-axis shows the time in hours and the y-axis shows the temperature values in degree Celsius. On one of the DHT sensor, we replaced the analog temperature module (thermistor) with a faulty thermistor. While the temperature readings from both the sensors look normal (within valid range), upon analyzing the Fall-curves (see Figure 9b), we can detect that one of the DHT sensors is faulty. This fault could not have been detected using any other techniques unless having more co-located sensors. In Section 6.5 we present some of the limitations of the proposed Fall-curve based technique to reliably detect faults in a digital sensor.

5 FALL-CURVE PROCESSING

A key challenge in using the proposed Fall-curve primitive is that one has to analyze and match the Fall-curves for each sampled sensor data. In this section, we first present an edge machine learning algorithm that can run on the microcontroller in real-time to (i) determine if the sensor is faulty or non-faulty and (ii) identify the sensor. We then describe various schemes that allow IoT designers to determine the granularity of fault probing in an IoT device.

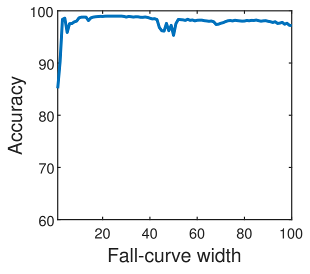
5.1 Edge algorithm to process Fall-curve

As mentioned earlier, the Fall-curve is collected for a short period of time when the sensor is turned off (see Figure 2). Our proposed edge algorithm pipeline has two stages as shown in Figure 10, (i) pre-deployment stage, wherein the Fall-curves for each sensor is collected and analyzed to extract key features and (ii) deployment stage, wherein the features from a new Fall-curve is matched with the previously extracted features to detect and isolate faults.

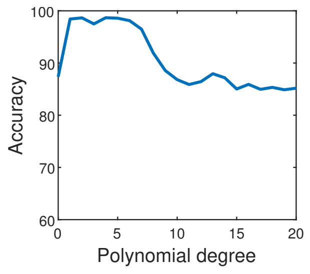
5.1.1 *Pre-deployment stage.*

In this stage, we first collect the Fall-curves for all non-faulty sensors used in an IoT deployment. We then find the best feature vectors that can be used to represent a Fall-curve. These feature vectors are further optimized to derive a feature dictionary for all sensors that can be loaded into the microcontroller towards sensor identification and fault detection. We now describe the steps involved:

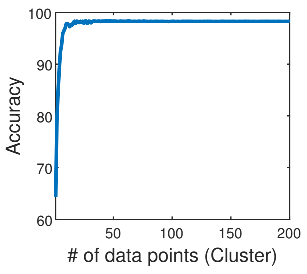
(1) When installing the sensors for the first time we record the Fall-curves of the *non-faulty sensors* and their corresponding sensor label.



(a) Fall-curve width



(b) Polynomial degree



(c) # of clusters

Figure 11: Effect of varying hyper-parameters on accuracy.

(2) We then fit a polynomial curve to each Fall-curve time-series and use the corresponding polynomial coefficients as the feature vector.
(3) We perform clustering on these polynomial features for each sensor to identify the unique features. This significantly reduces the search space and generates a smaller dictionary of polynomial features.
(4) Considering the resource and power constraints of the IoT devices, we optimize a set of hyper-parameters *viz.,* Fall-curve width[2], degree of polynomial, and number of clusters.
(5) The resulting feature dictionary along with the chosen hyper-parameters is then loaded onto the IoT devices for real-time Fall-curve analysis.

Note that, the aforementioned pre-deployment steps are performed with *only non-faulty sensors* before deploying in the field.

5.1.2 Deployment stage. In the deployment stage, we first extract the polynomial features of a new Fall-curve, then find its nearest neighbor from the feature dictionary obtained during the pre-deployment stage. If the nearest neighbor distance is within a certain threshold we classify the Fall-curve as working and assign the corresponding sensor label, else we classify the sensor as faulty and send the Fall-curve to the gateway/cloud for further processing. The nearest neighbor search is performed locally on the IoT devices using an efficient algorithm – ProtoNN [14].

5.1.3 Evaluation. We first present how to select optimal hyper-parameters and then present overall accuracy achieved in identifying the sensors and detecting sensor faults.

1. Selection of hyper-parameters: As described in Section 5.1.1, the proposed edge algorithm enables IoT designers to trade-off accuracy based on resource and power constraints on the IoT device. There are three hyper-parameters that can be fine-tuned based on the fault detection accuracy.

(i) Fall-curve width: Each Fall-curve has a different fall-time to reach zero (see Figure 3). Intuitively, the larger the Fall-curve width the higher is the time and energy required to process the Fall-curve samples. Figure 11a shows the trade-off in the accuracy of detecting a sensor for varying Fall-curve width. We can see that the accuracy saturates after Fall-curve width of just 10 samples, indicating, the polynomial curve fitting is able to capture the key features of the Fall-curve with just a few samples.

[2]A Fall-curve width of n implies the first n samples from the Fall-curve time-series.

Table 2: Performance of Fall-curve processing

Sensor	Accuracy	Precision	Recall
Faulty sensor	98.88	93.54	94.15
Accelerometer	99.49	96.92	97.50
Current sensor	100.00	100.00	100.00
Hall sensor	100.00	100.00	100.00
LDR sensor	99.82	99.48	98.50
Line tracker	99.34	99.79	92.90
Soil moisture (Gravity)	99.79	99.20	98.50
Soil moisture (Vegetronix)	99.57	95.48	100.00
Soil temperature	99.76	97.47	100.00
Sound sensor	100.00	100.0	100.00
Vibration sensor	100.00	100.0	100.00

(ii) Polynomial degree: In general, as we increase the polynomial degree, the accuracy also increases up to a certain extent. However, the higher the polynomial degree the higher is the resource requirement. Figure 11b shows the detection accuracy across different polynomial degrees. We can see that up to polynomial degree of 8, the accuracy is close to 98%, after which the accuracy drops. This accuracy drop indicates that the higher polynomial degrees are not able to capture any new information in the Fall-curve.

(iii) Number of clusters: A Fall-curve for the same sensor might have some variations resulting in different polynomial coefficients/ features. To eliminate this variation, we perform clustering across all polynomial features obtained for a particular sensor. Thus resulting in a set of optimal polynomial features for each sensor. A lower number of clusters reduces the time required to match the polynomial features. Figure 11c shows the accuracy across varying number of clusters. We can see that the accuracy saturates after the number of clusters is greater than 10.

Thus, we select Fall-curve width of 10 samples and polynomial degree of 4 with 10 clusters as our optimal values.

2. Accuracy: In order to evaluate the proposed edge algorithm, we use 10 different sensors with each having 1000 non-faulty and 2000 faulty Fall-curve instances collected from lab experiments and real-world deployments. We use the optimal hyper-parameters selected above to determine the accuracy of identifying the sensor. Note that, if the Fall-curve is not matched to one of the sensors, then we classify the Fall-curve as faulty.

Table 2 shows the Accuracy, Precision, and Recall values in percentage for 10 sensors. Accuracy is the ratio of correctly predicted instances to the total number of instances. We can see that among 20000 faulty Fall-curves across 10 sensors, the proposed algorithm is able to detect 19776 Fall-curves as faulty, resulting in an accuracy of 98.88% (shown in row 1 of Table 2). Furthermore, our algorithm achieves an overall accuracy of 99% in identifying the correct sensor.

5.2 Fault analysis schemes

We now present various schemes for Fall-curve based fault analysis. These schemes allow designers to trade off accuracy, granularity of fault analysis, and power consumption to determine the quality of data as required by their particular IoT application.

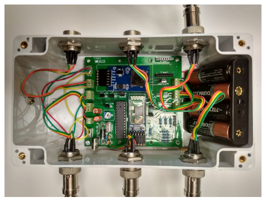
5.2.1 Sampling interval triggered fault analysis. The number of sensor readings taken by an IoT device is governed by the sampling interval. Generally, the sampling interval is set based on the application scenario and power requirements. Fall-curve based fault analysis can be triggered based on the sampling interval that is controlled with a rate parameter (r), which determines the granularity for fault analysis. If $r = 1$ fault analysis is triggered for every sensor reading, which is beneficial for IoT applications that require highest data fidelity, for example, when the phenomenon being monitored continuously varies. Similarly, $r = 100$ implies fault analysis performed on every 100^{th} sensor reading. A low rate parameter has a bearing on power consumption, however, supports higher accuracy. A high rate parameter comes up with a probability of missing faulty data points from being probed, which may be acceptable if the required data fidelity is not high. Upon detection of a fault, this scheme adapts the rate parameter to a lower value in order to aggressively probe for faults, otherwise, it maintains the pre-defined rate parameter.

5.2.2 Event triggered fault analysis. In this scheme, we first analyze the sensor data and look for an outlier event. Upon detecting an outlier event, Fall-curve based fault analysis is triggered for a definitive answer. Outlier detection could be a simple rule-based detection to complex Hidden Markov Models (HMM) or artificial neural networks. Note that, hitherto fault analysis scheme is only triggered based on an investigation on sensor data. As described in Section 2 (see Figure 1a), there are scenarios where faulty sensor data can mimic non-faulty sensor data. While this scheme is efficient in detecting faults when the sensor data has an outlier, it may miss probing faulty data points in the aforementioned scenarios.

5.2.3 Hybrid fault analysis. This scheme combines both sampling interval triggered and event triggered fault analysis schemes. Here, a Fall-curve based fault analysis is triggered on a particular interval defined by the rate parameter, or after an outlier event detection in sensor data. Hence, it decreases the chance of missing faulty data points from being probed compared to the two aforementioned schemes, and thus increases the accuracy. In addition, it is power-efficient compared to the sampling interval triggered fault analysis having a low valued rate parameter. Upon detection of a fault, this scheme also adapts the rate parameter to a lower value in order to aggressively probe for faults.

6 EVALUATION

We evaluated Fall-curve based technique on a large-scale IoT deployment over a period of three months. In this section, we first delineate our IoT node design and deployment setup. We then highlight the faults we encountered in our deployment, and how our system detected and isolated these faults leveraging Fall-curve.


(a) Internal components

(b) Deployed sensors

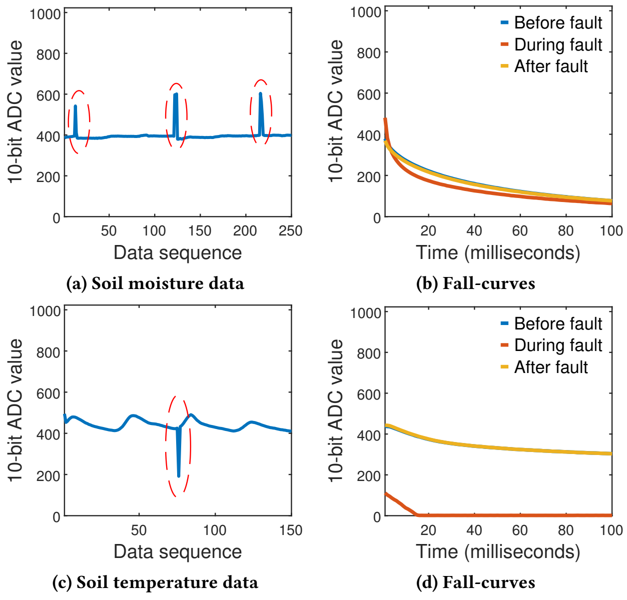
Figure 12: IoT sensor node deployed in agricultural farms.

6.1 IoT sensor node design and deployment

We have designed a custom-off-the-shelf IoT sensor node (Figure 12a) for monitoring the soil properties in agricultural farms. In this IoT sensor node, we have incorporated a low-power 8-bit microcontroller, ATmega328P, as the control unit. The operating voltage for the microcontroller is set to 3.3V at 16MHz frequency. It consumes ∼ 6.6mA current in the active state and ∼ 0.1µA in sleep mode. We have six three-wire (VCC, Signal, GND) ports for connecting external sensors to our IoT sensor node. Each of these ports can support both analog and digital (one-wire protocol based) sensors. Furthermore, each port is individually powered by the microcontroller through a Bipolar Junction Transistor (BJT) based trigger circuitry. Thus, it allows the microcontroller to turn ON a sensor only during data acquisition.

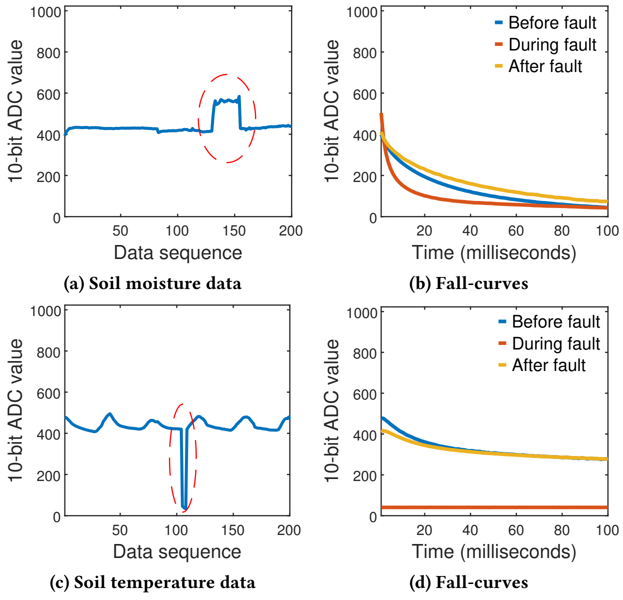
In our deployment, each IoT sensor node contains four low-power analog sensors. We have two soil temperature sensors from Vegetronix [40], each of them consumes ∼ 3mA current. One capacitive soil moisture sensor from Gravity [11] which consumes ∼ 5mA current. Finally, we have another soil moisture sensor from Vegetronix [41] having a current consumption of ∼ 13mA. In addition, each IoT node includes an SD card for data storage where we log sensor data, corresponding Fall-curves, detailed results of Fall-curve processing, and battery voltage. We utilize these logs for further analysis in offline. We have two onboard communication modules - LoRa (868MHz) and Bluetooth. LoRa module sends sensor data, results of corresponding Fall-curve processing, and battery voltage to a Raspberry Pi based SubEdge node. This SubEdge node uploads the data on to the cloud after some minor processing of sensor data. Note that, considering the low data-rate and payload size of LoRa, we do not send the raw Fall-curve data or detailed results of Fall-curve processing from the sensor node to the SubEdge. For each Fall-curve of a sensor, we send the sensor type label along with nearest neighbor distance computed by our edge algorithm for alerts/visualization. Finally, to power up the IoT node, we use a series of four 1.5V AA batteries.

We have deployed 20 such IoT sensor nodes in a few agricultural farms with diverse crops for data-driven precision agriculture. Sensors were deployed at different soil depths (15cm to 30cm) depending on the type of crops. Given small irrigation cycles and protracted change in soil temperature, a moderate data sampling rate of 2 hours was used for sensor data collection. Ground truth for fault detection was collected by deploying an additional sensor of the same type at exact location and depth. To further strengthen our ground truth, rigorous manual hardware probing was conducted on a regular basis. For example, we asked a field staff to check for



(a) Soil moisture data (b) Fall-curves (c) Soil temperature data (d) Fall-curves

Figure 13: Short fault and its corresponding Fall-curves.



(a) Soil moisture data (b) Fall-curves (c) Soil temperature data (d) Fall-curves

Figure 14: Spike fault and its corresponding Fall-curves.

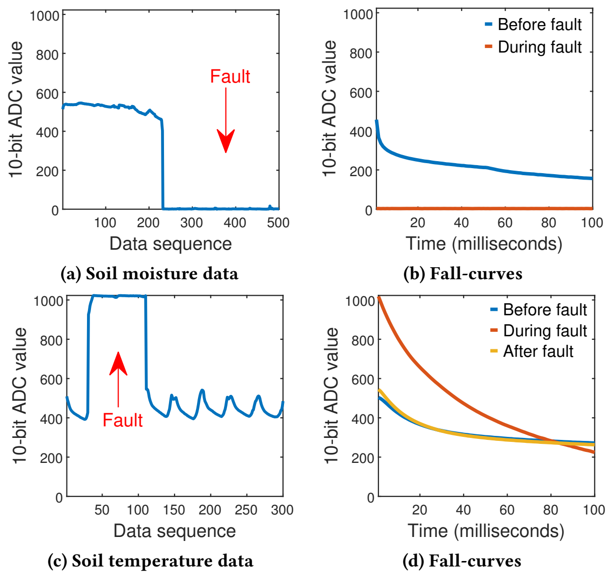
system faults, such as open wire connection, any physical damage, etc., twice a day.

6.2 Fault detection and isolation using Fall-curve

We now describe the faults that we encountered during our deployment and how our Fall-curve based technique detected and isolated these faults.

6.2.1 Short fault. Figure 13a, 13c depicts Short fault instances in soil moisture sensor (Gravity) and soil temperature sensor of node 3 from our deployment. The x-axis represents the data sequence where each point represents a 2 hour time interval. It can be seen that the soil moisture data in Figure 13a is generally stable as these are rain-fed farms with occasional Short fault. Similarly, in Figure 13c, we can see the diurnal pattern of soil temperature data. Figure 13b, 13d shows the Fall-curves accumulated from the sensors just before, during, and after the Short fault instance. In these instances, though all the system components were working fine, both the sensors had a transient malfunction. It can be seen that, during the Short fault, the corresponding Fall-curves are different and our system was able to detect these transient sensor malfunctions. The transient sensor malfunction could be due to the environmental factors, etc.

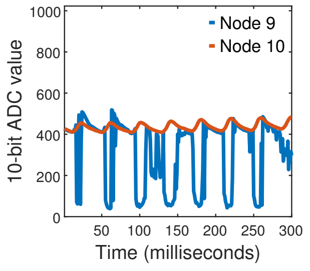
6.2.2 Spike fault. Figure 14a, 14c shows Spike faults encountered in our deployment. The highlighted region in Figure 14a shows that the soil moisture data is high for over 2 days (24 data sequences). Data-centric techniques cannot determine if the sensor data is faulty or not in this scenario without contextual information. By analyzing the Fall-curves, our system identified that the faults were due to transient malfunction of the sensor. Figure 14b and 14d shows the corresponding Fall-curves before, after, and during the Spike fault. The pattern of the Fall-curve corresponding to a faulty sensor data is different from the one corresponding to any non-faulty sensor data. Further, upon manual investigation of Spike fault in Figure 14a, it turned out that mud was found on the exposed soil moisture circuity, and after removal of the mud, the sensor started working again as shown in the Figure. 14a.



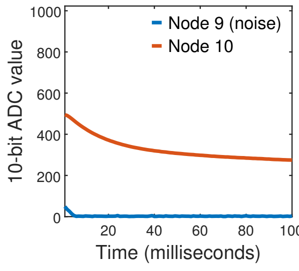
(a) Soil moisture data (b) Fall-curves (c) Soil temperature data (d) Fall-curves

Figure 15: Stuck-at fault and its corresponding Fall-curves.

6.2.3 Stuck-at fault. Figure 15a shows an instance of Stuck-at fault in soil moisture sensor. To determine if it is a sensor fault we analyzed the Fall-curve before, after, and during the Stuck-at fault. Figure 15b shows two significantly different Fall-curves indicating a sensor failure. To understand the reasoning of sensor failure, we analyzed the logs stored in the onboard SD card of the IoT node and identified that the Stuck-at fault was due to a battery voltage spike. This caused a *permanent* failure in soil moisture sensor. Figure 15b



(a) Noise fault



(b) Fall-curves

Figure 16: Noise fault and its corresponding Fall-curves.

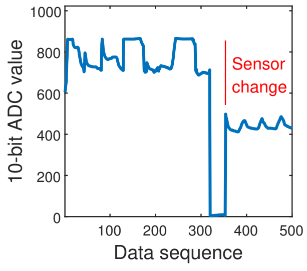
shows the corresponding Fall-curves of not-faulty (before battery spike) and faulty (after battery spike) soil moisture sensor.

Figure 15c shows another instance of Stuck-at fault, where soil temperature sensor permanently went bad. Our on-field investigation revealed that some parts of the sensor wire had clear damage possibly due to rats chewing up the wire. After replacing the sensor, soil temperature value went back to the normal state. Figure 15d shows that the Fall-curves for the initial working sensor (before fault) and the replaced sensor (after fault) are similar, whereas the Fall-curve in the faulty state of initial sensor is completely different.

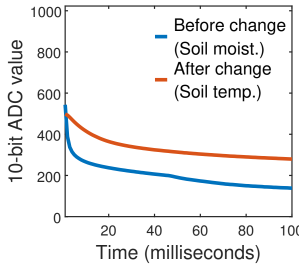
6.2.4 Noise. Figure 16a shows soil temperature data of two sensor nodes, where data from node 9 exhibits the Noise fault. Figure 16b shows the Fall-curves of the faulty sensor from node 9 and a non-faulty sensor from node 10 for comparison. It can be seen that the Fall-curves are completely different for these two sensors. During the on-field investigation, we found a loose connection on the ground pin of the soil temperature sensor from node 9, resulting in noisy sensor data.

6.2.5 Sensor port mismatch. In one of the sensor nodes, soil moisture sensor had permanently gone bad due to a stuck-at fault. However, during replacement, the soil temperature sensor was mistakenly added on the port of soil moisture sensor by the field staff. Figure 17a shows the sensor data, where data before sequence number 319 was from soil moisture sensor and data after 353 was from soil temperature sensor. Our Fall-curve based technique automatically identified this mismatch by analyzing the Fall-curves as shown in Figure 17b. It can be clearly seen that the Fall-curves before data sequence 319 and after 353 have two different signatures. Further, our system was able to match the Fall-curve to the soil temperature sensor and mark the data from this port as soil temperature by applying correct processing formula, rather than applying an incorrect processing formula.

6.2.6 Beyond sensor data faults. A key feature of our approach is to detect such faults that previously could not be diagnosed, such as faulty sensor data mimicking non-faulty sensor data and detection of unseen non-faulty data (novel data). In Section 2 (see Figure 1), we described these two scenarios where data-centric approaches fail. In Figure 18, we show how Fall-curve based technique can be used to detect these. We can see that, in Figure 18a, the Fall-curves for node 10 and 15 look identical, whereas, node 19 has a distinguishable Fall-curve indicating a fault. Upon manual inspection, it was found that node 19 had an open ADC wire connection. In another instance, Figure 1b depicts a few spikes generated by an LDR sensor deployed

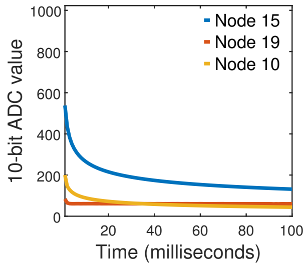


(a) Senor port mismatch

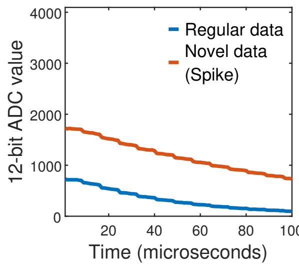


(b) Sensor identification

Figure 17: Sensor identification and port mismatch.



(a) Mimic Non-faulty data



(b) Novel data

Figure 18: Fall-Curves for detecting faulty sensor data mimicking as non-faulty and novelty.

in a room. These spikes are due to the abrupt changes in the lighting environment as opposed to faulty data. Figure 18b presents the Fall-curves for both regular data points and spikes. As we can see, both Fall-curves show the similar pattern. Consequently, our system isolated those spikes as novel data leveraging the Fall-curve. *Thus, with the help of Fall-curve, our system can detect and isolate faults in an IoT deployment.*

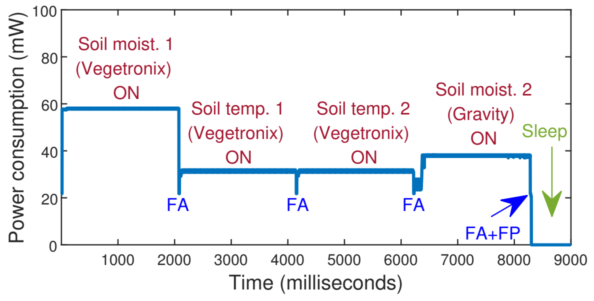


Figure 19: Power profile of an IoT node incorporating Fall-curve technique.

6.3 Energy consumption profiling

We now describe the energy consumption profile of the IoT sensor node (described in Section 6.1), which uses Fall-curve based technique to detect and isolate faults. Due to the power constraint, IoT nodes are generally duty cycled, where the node switches between (long) sleep state and (short) active state. In our setup, the sampling interval is set to 2 hours wherein the node is in the sleep state for 2 hours followed by a short active state of 8.25 seconds. In the active state, the IoT node triggers each individual sensor sequentially to the ON state, which includes sensor warm-up time (up to 2sec per sensor to stabilize) and sensor data collection. The energy consumption in the active state of 8.250 sec is 0.32J.

The Fall-curve based fault detection technique introduces two new activities in the sensor node active state, *viz.,* Fall-curve accumulation (FA) for each sensor and one-time Fall-curve processing (FP) (at the end of current active state) as shown in Figure 19. Note that, we collect Fall-curve after turning off a sensor as a complementary part of sensor data collection. The time taken to collect Fall-curve per sensor is around 10ms with 0.22mJ energy consumption. Thus, the total time taken to collect Fall-curve from 4 sensors in our deployment takes 40ms with 0.88mJ energy consumption. Furthermore, for local Fall-curve processing, we adopt the optimal values of hyper-parameters as discussed in Section 5.1.3, i.e., Fall-curve width of 10, 10 clusters for each sensor, and polynomial degree of 4. The total time taken for all the Fall-curve processing is $\sim$ 10ms with energy consumption of $\sim$ 0.1mJ. Thus, the total energy consumed in the active state with the proposed Fall-curve technique includes default active state energy (0.32J) + Fall-curve accumulation for all sensors (0.88mJ) + Fall-curve processing (0.1mJ) = 0.32098J. The resulting overhead due to Fall-curve accumulation and processing in the active state of the node is 0.3%, i.e., ((0.32098-0.32)/0.32)x100. Thus the proposed Fall-curve accumulation and processing algorithms consume very nominal energy overhead of 0.3% when each data sample is analyzed for fault detection.

6.4 Complete system evaluation

Now, we will show the performance of three Fall-curve processing schemes, described in Section 5.2, in order to tradeoff accuracy, granularity of fault analysis, and power consumption for determining the quality of data. This evaluation is performed on the dataset collected from our deployment described above. Based on the ground truth labels, 10.49% of total 29622 data samples were found faulty, which is comparable to the fault frequency in a typical IoT deployment [27, 34].

We utilize four different metrics, *viz.,* accuracy, precision, recall, and F1 score, where for each data point we compare the assigned labels (faulty/non-faulty) by the decision engine to the ground-truth labels. A false positive is defined as a non-faulty data point incorrectly labeled as faulty, and a true positive is defined as a faulty data point correctly identified. We then calculate the aforementioned performance metrics using the standard formula [30].

Table 3: Performance evaluation of various schemes.

Schemes	Parameter	Performance metrics				% of extra energy
		Accuracy	Precision	Recall	F1	
Baseline	-	66.9	90.29	45.37	60.39	NA
Sampling interval triggered	$r = 1$	99.13	93.44	98.58	95.94	0.3
	$r = 3$	99.19	97.39	94.88	96.12	0.13
	$r = 6$	98.82	98.21	90.41	94.15	0.1
	$r = 12$	98.25	98.64	84.52	91.04	0.07
	$r = 24$	97.17	99.26	73.74	84.62	0.06
Event triggered	$d = 10$	97.3	98.98	75.06	85.37	0.08
Hybrid	$r = 6$, $d = 10$	99.16	97.92	93.98	95.91	0.12

Table 3 presents the performance of various proposed schemes and a baseline scheme for fault detection. We first employ an existing technique, which uses time-series based SVM to detect faults in IoT sensor data [28] as our baseline scheme. Using this technique on our dataset, we found accuracy to be 66.9% with precision and recall values of 90.29% and 45.37% respectively. Since the dataset includes various types of faults (e.g., faulty data mimicking non-faulty data) beyond typical sensor faults the accuracy and recall values are low. Besides, this scheme cannot run on an embedded microcontroller with limited resources. Hence, we are not showing a number for the power consumption using the baseline scheme.

Turning to the ability of our Fall-curve based technique in detecting faults. In case of sampling interval triggered fault analysis scheme, we vary the rate parameter (r) starting from 1 to 24. We obtain an accuracy of 99.13% with 0.3% of extra energy consumption by probing at every data sampling event, i.e, $r = 1$. Both performance and energy consumption decreases with the increment of r. As we increase the value of r, the probability of missing a faulty data point from being probed increases, and thus the performance number decreases. For example, if we probe at every 6^{th} data sample, i.e., twice in a day in case our deployment, we obtain an accuracy of 98.82% with 0.1% of extra energy consumption. As we mentioned earlier, our system starts aggressively probing after detecting a faulty data point. Thus, the accuracy does not significantly degrade with $r = 6$ compared to $r = 1$. To show the effectiveness of aggressive probing after a fault detection, we performed an evaluation with and without aggressive probing. For $r = 3$, we obtain overall accuracy of 99.19% (0.13% energy overhead) and 92.76% (0.1% energy overhead) with and without aggressive probing. Similarly, for $r = 6$, accuracy is 98.82% (0.1% energy overhead) with aggressive probing and 91.13% (0.05% energy overhead) without aggressive probing. Hence, it is clear that aggressive probing results in higher accuracy, however, costing a little bit higher power consumption.

Next, as a part of our event triggered scheme, we incorporated a lightweight threshold-based anomaly detector exploiting the absolute distance between sensor data samples. Here, we tune the absolute distance parameter (d) which can vary depending on the application and context. As the soil moisture and soil temperature values do not change rapidly, we set $d = 10$ in our deployment. We obtain a fault detection accuracy of 97.3% with only 0.08% of extra energy consumption. However, the recall value significantly degrades compared to sampling interval triggered scheme, since the dataset contains a good number of faulty data samples mimicking non-faulty data (Figure 1a). Finally, with our hybrid scheme, which combines both sampling interval and event triggered schemes, we obtain an accuracy of 99.16% with 0.12% of extra energy consumption. This shows the ability of the Fall-curve based technique to accurately detect sensor faults in a real-world deployment with no hardware modifications.

6.5 Discussion

In this part, we discuss the limitations of the Fall-curve technique towards fault detection with possible solutions to overcome it.

(i) Hardware modification to detect fault in digital sensors: As described in Section 4.2.2, in order to detect faults in the analog block of a digital sensor, we proposed a single-wire connection between the analog sensing module and the ADC port of the external microcontroller. There are a few limitations to this approach, *viz.,* requires small hardware modification to attach the single-wire, in some sensors the analog part of the digital sensor may not be accessible without breaking the sensor, etc. Modern, industrial-grade digital sensors, e.g., T6713 CO_2 sensor [8], PMS7003 dust sensor [43], etc., have an internal microcontroller, which accumulates data from the analog sensing module of the sensor through the on-board ADC. The same ADC port could be used to accumulate the Fall-curve of the analog sensing module. However, this requires firmware modification by the manufacturer. Further, low-cost digital sensors, e.g., ADXL345 accelerometer [12], Adafruit PIR sensor [1], etc., do not have an internal microcontroller. In such cases, accumulating Fall-curve requires hardware modification, which may not be feasible considering the compact form-factor of sensors.

(ii) Sensors that require significant warm-up time: There are few sensors that require significant warm-up time before measuring the phenomenon. For example, air quality sensors such as MQ-135 [22] and carbon monoxide sensor MQ-7 [23] requires sensor pre-heating of over 24 and 48 hours respectively. In such cases, the proposed Fall-curve technique would not be compelling, as it requires to turn-off the sensor to accumulate the Fall-curve. However, newer gas sensors [2] have warm-up times in few minutes, which accommodates the Fall-curve based fault detection.

(iii) Fault detection when sensor value is low: If the raw values generated by a sensor are very low, then the resulting fall-time after turning off the sensor is negligible. This results in either a very small or no Fall-curve being accumulated. For example, a distance sensor is generating a low value as the object of interest is very close. One possible solution is using a higher sampling rate to accumulate the Fall-curve.

7 RELATED WORK

Over the past decade, researchers have shown a profound interest in detecting sensor data faults in IoT and sensor network deployments. These studies employ techniques such as simple heuristic-based approaches to complex machine learning and neural network models. Majority of the prior work on detecting faults rely on analyzing sensor data to look for fault patterns. They are specific to either the type of sensor, fault, or application context [34]. Furthermore, these techniques can be broadly classified into four categories: heuristic-based, estimation-based, temporal analysis-based, and machine learning-based methods.

Heuristic-based methods use domain knowledge of sensor data, spatiotemporal granularity, and contextual info to develop heuristics [16, 32, 33]. These methods are lightweight, however, their performance largely depends on the selection of parameters [27, 34]. Besides, scalability of these methods is questionable considering heterogeneous contexts and non-stationary environments.

Estimation-based methods model normal sensor behavior leveraging spatiotemporal correlation and different probabilistic models, such as Bayesian approach, Gaussian distribution, etc., [4, 5, 17, 19, 26, 34, 35, 38]. For example, Least-Squares Estimation (LLSE) method leverages spatial correlation of sensor data and computes covariance between sensor measurements to detect faults [34]. Tolle et al., [38] exploit spatiotemporal correlation in micro-climate on a redwood tree to reveal trends and gradients in the accumulated dataset. Estimation-based methods are more accurate when the environment is homogeneous.

Temporal analysis methods leverage temporal correlations in sensor data collected by the same sensor to estimate the parameters of a predefined model (e.g., ARIMA) for the current deployment [10, 24, 34, 39]. Sharma et al. [34], proposed a multiplicative seasonal ARIMA time series model for fault detection, where the parameter captures the periodic behavior in the sensor measurement time series. The downside of temporal analysis methods is that they are prone to false positives and are not feasible in long-term deployments having short faults.

Finally, machine learning-based methods (e.g., HMM, SVM classification, etc.) have also been used for fault diagnosis [13, 18, 34]. Although the machine learning approaches are more accurate than other approaches, they rely on the availability of historical data for a new deployment and use a static model for a non-stationary environment. Several complex approaches have been recently proposed to adapt to non-stationary environments in both local and distributed computing [6, 9, 20, 31, 37, 42, 44]. Although they address the problem with non-stationary environments, they still require historical data for initial training of the models. Moreover, these approaches are highly sensitive to the isolation of novelty and anomaly, which is still prone to lower accuracy considering a non-stationary environment. In addition, a scenario explained in Figure 1 can exacerbate the overall performance of the system for a long period.

In contrast to the above approaches, Fall-curve is a new way to detect faults, even when the faulty readings mimic actual data – a scenario that is not addressed by prior work. Our system also isolates faults within the sensors, a scenario that to the best of our knowledge, has not been looked at before. Finally, the proposed fault detection and isolation runs locally on the IoT device without any hardware modification and has an energy overhead of 0.3%.

8 SUMMARY & FUTURE WORK

The growth of sensors and IoT deployments has led to a need for automating the detection of sensor faults and the isolation of the cause of the error. Existing schemes rely purely on sensor data to detect anomalies, and as we show in this paper, they miss out on detecting several failure scenarios. We propose a new primitive, called Fall-curve, that can identify a faulty sensor using its voltage response when power to the sensor is cut off. We have evaluated this system in a real agricultural system, and shown that it is able to identify and isolate faults that could not be diagnosed before.

Moving forward, we are extending this work in two directions. First, we are developing a new "fidelity" parameter associated with sensor data. In existing IoT systems, the sensors only report the data to the cloud. Using Fall-curve, one can envision a future where the data is accompanied by a confidence metric, based on Fall-curves of the sensors. Second, we are investigating a technique to bring down the cost of IoT deployments. Existing commercial deployments use high-end sensors, which significantly increases the cost of a deployment. For example, in agriculture, commercial grade moisture sensors cost a few hundred dollars, even though the low-end sensors can be purchased for less than 10 dollars. Using Fall-curve, we are investigating if a few low-cost sensors can be used to replicate the fidelity and reliability of one expensive sensor. Our technique can identify when a low-cost sensor is reporting incorrect data, and trigger the operation of another sensor. Our initial experiments with some off-the-shelf sensors show very promising results.

REFERENCES

[1] Lady Ada. 2018. PIR Motion Sensor. (2018). Retrieved Sep 16, 2018 from https://goo.gl/gvF93e

[2] Alphasense. 2018. Alphasense sensors. (2018). Retrieved May 27, 2018 from https://goo.gl/ivXVes

[3] Arduino. 2018. Arduino Uno. (2018). Retrieved April 4, 2018 from https://goo.gl/pDxdRv

[4] Laura Balzano and Robert Nowak. 2007. Blind calibration of sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*. ACM, 79–88.

[5] Vladimir Bychkovskiy, Seapahn Megerian, Deborah Estrin, and Miodrag Potkonjak. 2003. A collaborative approach to in-place sensor calibration. In *Information Processing in Sensor Networks*. Springer, 301–316.

[6] Vassilis Chatzigiannakis and Symeon Papavassiliou. 2007. Diagnosing anomalies and identifying faulty nodes in sensor networks. *IEEE Sensors Journal* 7, 5 (2007), 637–645.

[7] Li Chen, Srivaths Ravi, Anand Raghunathan, and Sujit Dey. 2003. A scalable software-based self-test methodology for programmable processors. In *Proceedings of the 40th annual Design Automation Conference*. ACM, 548–553.

[8] Amphenol Corporation. 2014. Telaire T6713 Series CO_2 Module. (2014). Retrieved Sep 16, 2018 from https://goo.gl/2CGciZ

[9] Daniel-Ioan Curiac and Constantin Volosencu. 2012. Ensemble based sensing anomaly detection in wireless sensor networks. *Expert Systems with Applications* 39, 10 (2012), 9087–9096.

[10] Amol Deshpande, Carlos Guestrin, Samuel R Madden, Joseph M Hellerstein, and Wei Hong. 2004. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 588–599.

[11] DFRobot. 2012. Gravity: Analog Capacitive Soil Moisture Sensor- Corrosion Resistant. (2012). Retrieved April 4, 2018 from https://goo.gl/p5bhFK

[12] Bill Earl. 2018. ADXL345 Digital Accelerometer. (2018). Retrieved Sep 16, 2018 from https://goo.gl/96X6fs

[13] Eiman Elnahrawy and Badri Nath. 2003. Cleaning and querying noisy sensors. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*. ACM, 78–87.

[14] Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. 2017. ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 1331–1340.

[15] Texas Instruments. 2013. Tiva TM4C123G Launchpad. (April 2013). Retrieved April 4, 2018 from https://goo.gl/GX1itw

[16] Shawn R Jeffery, Gustavo Alonso, Michael J Franklin, Wei Hong, and Jennifer Widom. 2006. Declarative support for sensor data cleaning. In *International Conference on Pervasive Computing*. Springer, 83–100.

[17] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. 2006. Towards correcting input data errors probabilistically using integrity constraints. In *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*. ACM, 43–50.

[18] Farinaz Koushanfar, Miodrag Potkonjak, and Alberto Sangiovanni-Vincentelli. 2003. On-line fault detection of sensor measurements. In *Sensors, 2003. Proceedings of IEEE*, Vol. 2. IEEE, 974–979.

[19] Bhaskar Krishnamachari and Sitharama Iyengar. 2004. Distributed Bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. *IEEE Trans. Comput.* 53, 3 (2004), 241–250.

[20] Mohammad Ahamdi Livani and Mahdi Abadi. 2010. Distributed PCA-based anomaly detection in wireless sensor networks. In *Internet Technology and Secured Transactions (ICITST), 2010 International Conference for*. IEEE, 1–8.

[21] LTspice 2018. LTspice. (2018). Retrieved May 27, 2018 from http://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html

[22] MQ-135 2018. MQ-135 Carbon Monoxide Sensor Technical datasheet. (2018). https://goo.gl/PKkn1G.

[23] MQ-7 Gas Sensor 2018. MQ-7 Gas Sensor Technical datasheet. (2018). https://goo.gl/MfhBLJ.

[24] Shoubhik Mukhopadhyay, Debashis Panigrahi, and Sujit Dey. 2004. Model based error correction for wireless sensor networks. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*. IEEE, 575–584.

[25] Dejan Nedelkovski. 2016. DHT11 & DHT22 Sensors Temperature and Humidity Tutorial using Arduino. (2016). Retrieved April 4, 2018 from https://goo.gl/YvroSj

[26] Kevin Ni and Greg Pottie. 2007. Bayesian selection of non-faulty sensors. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*. IEEE, 616–620.

[27] Kevin Ni, Nithya Ramanathan, Mohamed Nabil Hajj Chehade, Laura Balzano, Sheela Nair, Sadaf Zahedi, Eddie Kohler, Greg Pottie, Mark Hansen, and Mani Srivastava. 2009. Sensor network data fault types. *ACM Transactions on Sensor Networks (TOSN)* 5, 3 (2009), 25.

[28] Colin O'Reilly, Alexander Gluhak, Muhammad Ali Imran, and Sutharshan Rajasegarar. 2014. Anomaly detection in wireless sensor networks in a non-stationary environment. *IEEE Communications Surveys & Tutorials* 16, 3 (2014), 1413–1432.

[29] Bob Orwiler. 1969. *Vertical Amplifier Circuits*. Tektronix, Inc., Beaverton, Oregon.

[30] Precision and recall 2018. Precision and recall. (2018). https://en.wikipedia.org/wiki/Precision_and_recall.

[31] Sutharshan Rajasegarar, Christopher Leckie, James C Bezdek, and Marimuthu Palaniswami. 2010. Centered hyperspherical and hyperellipsoidal one-class support vector machines for anomaly detection in sensor networks. *IEEE Transactions on Information Forensics and Security* 5, 3 (2010), 518–533.

[32] Nithya Ramanathan, Laura Balzano, Marci Burt, Deborah Estrin, Tom Harmon, Charlie Harvey, Jenny Jay, Eddie Kohler, Sarah Rothenberg, and Mani Srivastava. 2006. Rapid deployment with confidence: Calibration and fault detection in environmental sensor networks. (2006).

[33] Nithya Ramanathan, Tom Schoellhammer, Deborah Estrin, Mark Hansen, Tom Harmon, Eddie Kohler, and Mani Srivastava. 2006. The final frontier: Embedding networked sensors in the soil. (2006).

[34] Abhishek B Sharma, Leana Golubchik, and Ramesh Govindan. 2010. Sensor faults: Detection methods and prevalence in real-world datasets. *ACM Transactions on Sensor Networks (TOSN)* 6, 3 (2010), 23.

[35] Bo Sheng, Qun Li, Weizhen Mao, and Wen Jin. 2007. Outlier detection in sensor networks. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 219–228.

[36] SparkFun 2018. SparkFun Soil Moisture Sensor. (2018). Retrieved May 27, 2018 from https://www.sparkfun.com/products/13322

[37] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. 2006. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 187–198.

[38] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, et al. 2005. A macroscope in the redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*. ACM, 51–63.

[39] Daniela Tulone and Samuel Madden. 2006. PAQ: Time series forecasting for approximate query answering in sensor networks. In *European Workshop on Wireless Sensor Networks*. Springer, 21–37.

[40] Inc. Vegetronix. 2014. Soil Temperature Sensor Probes. (2014). Retrieved April 4, 2018 from https://goo.gl/iot1YW

[41] Inc. Vegetronix. 2014. VH400 Soil Moisture Sensor Probes. (2014). Retrieved April 4, 2018 from https://goo.gl/tDTh95

[42] Miao Xie, Jiankun Hu, Song Han, and Hsiao-Hwa Chen. 2013. Scalable hypergrid k-NN-based online anomaly detection in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems* 24, 8 (2013), 1661–1670.

[43] Zhou Yong. 2016. Digital universal particle concentration sensor. (2016). Retrieved Sep 16, 2018 from https://goo.gl/LvJJKz

[44] Yang Zhang, Nirvana Meratnia, and Paul JM Havinga. 2013. Distributed online outlier detection in wireless sensor networks using ellipsoidal support vector machine. *Ad hoc networks* 11, 3 (2013), 1062–1074.