coSense: The Collaborative Sensing Middleware for the Internet-of-Things

STEPHAN SCHMEIßER and GREGOR SCHIELE, University Duisburg-Essen

We present *coSense*—the collaborative, fault-tolerant, and adaptive sensing middleware for the Internet-of-Things (IoT). By actively harnessing the greatest asset of the IoT, the sheer number of devices, *coSense* is able to provide easy data acquisition with quality-of-service-based data cleaning by combining unsupervised learning and information fusion. It can also greatly improve sensor accuracy and fault tolerance to produce measurements specifically tailored for modern data-driven IoT empowered applications. In this article, we focus on the general concepts behind *coSense* and evaluate the accuracy gain based on a real-world dataset.

CCS Concepts: • **Computer systems organization** → **Sensor networks; Sensors and actuators; Redundancy**;

Additional Key Words and Phrases: IoT, sensor fusion, clustering

ACM Reference format:

Stephan Schmeißer and Gregor Schiele. 2020. *coSense: The Collaborative Sensing Middleware for the Internet-of-Things*. *ACM/IMS Trans. Data Sci.* 1, 4, Article 28 (November 2020), 21 pages. <https://doi.org/10.1145/3395233>

1 INTRODUCTION

The Internet-of-Things (IoT) is becoming a reality. Gartner estimates up to 25 billion IoT-enabled devices by 2021 [11]. Modern data science algorithms and machine learning can use the IoT to get huge amounts of different kinds of input data, mostly originating from sensors. However, using data from the IoT can be difficult, as sensors must be cheap. Additionally, they often have poor sensing quality and are prone to failure. To be of use for machine learning and data science algorithms, we need to improve the data quality of IoT devices. To this end, we are introducing *coSense*—the collaborative, fault-tolerant, and adaptive sensing middleware for the IoT. *coSense* combines unsupervised learning with the fault-tolerant interval (FTI) [23] function to achieve scalable, reliable, and adaptable information fusion to empower modern IoT-driven applications with varying quality-of-service (QoS) requirements. In this article, we focus on the general concepts behind *coSense*. We describe how the accuracy of measurements can be improved and adapted to application needs while easing data acquisition. To this end, we provide the following core contributions:

Authors' address: S. Schmeißer and G. Schiele, University Duisburg-Essen, Embedded Systems, Bismarckstr. 90, 47057, Duisburg, Northrhine Westphalia; emails: {stephan.schmeisser, gregor.schiele}@uni-due.de.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.

2577-3224/2020/11-ART28 \$15.00

<https://doi.org/10.1145/3395233>

- (1) We show how to use graph-based community detection (known from social network analysis) to create clusters of (quasi-)redundant sensor measurements and sensors. This produces more meaningful clusters without *a priori* knowledge about the number of clusters.
- (2) We propose a novel way to combine such clusters of quasi-redundant sensors into configurable, abstracted (virtual) sensors using the FTI function known from clock synchronization.
- (3) We design a flexible API for application developers that allows them to configure dynamically an abstracted sensor toward improved accuracy or fault tolerance as needed.
- (4) We provide an end-to-end real-world implementation of these components as an integrated middleware and evaluate its performance.

The rest of the article is structured as follows. In Section 2, we describe a scenario together with requirements highlighting the need for modern IoT applications. In Section 3, we discuss related systems. Section 4 provides an overview of the general concepts and architecture of the coSense middleware. Section 5 gives an in-depth description of the FTI function. Section 6 presents unsupervised clustering with a focus on community detection for our middleware. In Section 7, the developed coSense middleware is evaluated according to its requirements. Section 8 concludes the article and provides an outlook on future work.

2 SCENARIO AND SYSTEM MODEL

A possible scenario for an IoT-empowered application using machine learning is environmental control on an office floor: there are multiple office spaces, conference rooms, equipment rooms, break or recreational rooms, and some form of kitchen all with different requirements for environmental control. An application controlling it has to keep track of many different data sources, and needs to compensate for faulty inputs, monitor the system state, check if all measured parameters are within given thresholds, learn different reporting schemes to accommodate business hours, and much more. The combination of IoT and machine learning makes for an obvious choice: IoT offers easy deployment of devices into pre-existing facilities without worrying about access to power, as they can be battery powered. Being battery powered usually entails a lot of maintenance for changing batteries. IoT devices, however, do not have high computational needs, which reduces the energy consumption to extend battery lifetime and eases maintenance. Networking is achieved by wireless communication and Internet access is possible so as to have remote access to the data. The devices are very cost efficient and can be easily deployed in bulk. Due to their cost-efficient nature, sensory parts of IoT devices lack accuracy, precision, and reliability, as pointed out in Section 1. Machine learning offers flexible and scalable algorithms to deal with a wide variety of input data.

2.1 Requirements

For any application to be used in the preceding scenario, it must fulfill a set of general requirements:

- (1) *Compensate inaccuracy*: According to ISO 5725, measurement *accuracy* describes the closeness of agreement of a measured value and its *true* value [13]. The true value is the exact physical quantity. Off-the-shelf sensors in most cases have a documented degree of inaccuracy, mostly found in the datasheet. It expresses how the measured value reported by the sensors deviates from the true value. Most sensors only measure a rough estimate of the observed phenomenon. This can be seen as the maximal expected inaccuracy. To make measurements useful, it is desirable to minimize the inaccuracy as much as possible.
- (2) *Compensate faults*: If multiple sensors measure the same thing and one sensor is deviating from all other measurements by more than the expected maximal inaccuracy, we call it

faulty. It is desirable to identify faulty devices and compensate for the failure without necessarily sacrificing overall performance.

- (3) *Provide flexibility and adaptation*: Some decision making and learning needs more accurate measurements than others. It is desirable to be able to adapt the system and its measurements in such a way that it can provide information for a wide variety of algorithms.
- (4) *Manage devices*: IoT systems can easily scale up to large numbers of devices, making the management a tedious task. This is especially the case if devices have to be queried one by one to gather information. It is desirable to ease the data access and management of devices.

Requirements (1) and (2) aim at dealing with broken or inaccurate sensors, whereas (3) and (4) aim at the system being more flexible and useful. A common way of achieving both aspects is by not looking at single devices at a time but rather multiple ones. The surplus of input data can then be used to create more useful information. At the same time, it reduces complexity. Only a single more useful piece of information has to be processed instead of multiple useless pieces. The process of using the data from multiple sources at the same time is usually referred to as aggregation or fusion.

3 RELATED WORK

In this section, we discuss the fundamentals of information fusion and give examples and scenarios of the different kinds of fusion approaches. We also present and discuss work related to harnessing redundancy in the IoT as well as in wireless sensor networks (WSNs).

3.1 Information Fusion Fundamentals

Information fusion has been established as a cornerstone of the IoT. The IoT produces a high volume of simple sensor measurements. To just cope with the large amount of information, it is being fused. Additionally, information fusion offers a gain in quality, reliability, and so forth for applications [1]. Information fusion is used in various different fields with varying intentions under different names. A comprehensive overview can be found in the work of Nakamura et al. [18] and Alam et al. [1]. Common names in the literature include *information fusion*, *data fusion*, *multi-/sensor fusion*, *multisensor integration*, and *data aggregation*. Information fusion and data fusion are used as synonyms and describe the broadest definition: exploiting the synergy of information, independent of the source, so that the resulting information is of more value, being it quantitatively or qualitatively [18]. Multisensor fusion and sensor fusion refer to information fusion with sensory information as the sole data source. Multisensor integration is from the field of robotics/computer vision and industrial automation and refers to information fusion with sensory information and other external information (e.g., from databases), with the dedicated purpose of doing inference on the fused data [18]. Data aggregation is often used in the context of the WSN and usually refers to a specific technique/set of functions, like SUM or AVG, with the dedicated goal of reducing information to save energy when transmitting information [1, 12, 18]. A more comprehensive classification can be found in other works [1, 2, 8, 12, 18]. For the purpose of this article, we will stick to the term *information fusion*. An integral part of information fusion is the availability and excess of information to be fused. The most common form of excess information is *redundancy*.

3.2 Harnessing Redundancy

Redundancy, especially for measurements, has been an established fact for as long as there have been sensors. The most common form is hardware redundancy, which simply means there are multiple sensors. Oftentimes these sensors measure the same phenomenon. Due to placement and

other environmental factors, like rate of change limitations, those sensors can produce redundant measurements. In early cases, this was done as a backup solution in case of sensor failures. Actively harnessing redundancy has become of interest with the introduction of WSNs. Sensor nodes are (randomly) deployed in great quantities, and (hardware) redundancy is a given fact. The common idea behind harnessing it is performing some kind of voting algorithm on the excess information. A systematic study on redundancy in WSNs was conducted by Curiac et al. [6]. It specifically defined and split redundancy into *spatial* and *temporal* redundancy. Spatial redundancy can stem from (i) multiple sensors observing the same phenomenon at the same location or (ii) an analytical component where additional sensors are estimated based on a mathematical model. Temporal redundancy, however, describes redundancy in the time domain. The same measurement is performed by the same sensor at different points in time, creating a time series for increased reliability. Sometimes the term *information redundancy* is used in the literature. It can refer to spatial redundancy but should be understood as redundancy in the encoding of messages. This provides tolerance against transmission errors such as parity bits and Hamming code [6]. The authors provide an algorithm for harnessing spatial redundancy that calculates a redundancy factor based on the location. It takes into account how many redundant sensors are available for a specific location. It then performs an aggregating voting algorithm. The specific goal of said algorithm is the facilitation of (i) energy usage, by using the redundancy factor to determine sleep schedules; (ii) reliability, using the voting algorithm to reach a consensus; and (iii) security measures, since the probability of a malicious attacker controlling all available sensors in a specific location is small. Compared to our approach, it needs to know the specific location of every sensor. The voting algorithms only improve reliability by identifying and filtering out faults. Compensation of inaccuracy as well as flexibility combined with simplifying data access is not possible with the proposed algorithm.

A more in-depth analysis of the voting algorithm based on spatial redundancy is presented in the work of Sun et al. [26]. The authors developed *confidence-weighted voting* to increase resilience against sensor errors. Compared to simple majority voting and distance-weighted voting, they were able to increase the resilience by up to 49%. Their algorithm, however, requires the sensors to be placed with a specific k -cover placement strategy and is only able to increase resilience. Clouqueur et al. [5] take a closer look at the difference between different kinds of redundant data: raw values and decision data, and how it impacts fault detection. In what they call *value fusion*, the raw sensor data is being fused at a central point by calculating the average and prior dropping the m minimal and maximal values. Dropping those values acts as a filter for faulty sensors that are defined as outliers. A decision is then created based on the fused data. In *decision fusion*, each sensor creates its own decision, based solely on its own data, which is forwarded to a central point. At this central point, a majority vote is performed to form a consensus. Their evaluation shows that, in regard to fault detection, value fusion performs better when the nodes can be considered reliable. Decision fusion outperforms it when faults are introduced. Factors other than fault detection are not considered.

A more general approach to using redundant or at least correlated measurements is proposed in the work of Benaskeur [3]. The author argues that the common use of a Kalman filter is difficult in a distributed manner, especially when the sources of information are correlated. Instead he proposes the *largest ellipsoid algorithm* to overcome the limitations of a Kalman filter and of a covariance-based approach. The algorithm treats measurements as ellipsoids. They overlap when measurements are redundant. It calculates the largest overlap, which represents the correlation of measurements. The algorithm is discussed with target tracking in mind. The evaluation shows that the proposed algorithm outperforms the Kalman filter and other covariance-based approaches. However, it lacks flexibility, the possibility for compensating inaccuracy, and easy data access.

An approach for filtering outliers and increasing accuracy is presented in the work of Zhao et al. [28], in which the authors use an adapted weighted fusion algorithm combined with Dempster-Shafer theory on redundant data from a WSN to create a wildfire warning system. To this end, they create a separate cluster of sensor nodes based on the deployment region, each with a dedicated clusterhead. This clusterhead takes the (redundant) local estimations of each cluster member and fuses them using Dempster-Shafer theory to create a global estimate (i.e., is there a fire in the deployment region?). The Dempster-Shafer theory is able to filter outliers, which increases the estimation accuracy. There is, however, no direct control how the accuracy is influenced and how sensitive the outlier detection performs.

Another statistical strategy for outlier detection is proposed by Kumar et al. [14]. In their approach, a sensor measurement is seen as a statistical variable with a Gaussian distribution with a corresponding probability density function (pdf). By extending Bayes theorem, they are able to detect if two measurements are redundant so as to fuse their pdfs to increase accuracy in the variable's estimation. The prerequisite for fusing is the compatibility of the measurements—for instance, none of them is spurious, and they are redundant. The extension of Bayes theorem checks their compatibility and discards a measurement if it is not compatible. Control over the possible accuracy gain is not given, as well as a way of easing data access.

Neto et al. [4] present a system for reliably selecting sensors in the IoT era. The assumption is that in the era of IoT, many people will deploy off-the-shelf sensors, which inevitably leads to redundant, but not necessarily reliable, measurements. To use those sensors, it needs a strategy to select reliable and accurate sensors. To deal with accuracy, they propose to fuse redundant measurements with a majority vote. The majority is based on the pairwise distance between measurement estimations. The distance between measurement estimations, however, is only calculated if the sensors are considered reliable. The selection of reliable sensors is based on the Pearson correlation coefficient r (PCC). The PCC describes the deviation of a measurement from the ground truth. If the value of r is high enough, the sensor and its measurement are considered reliable. The evaluation was conducted on real-life, crowdsourced IoT weather data. It shows that the reliable sensor selection combined with a majority vote can produce suitable results. However, the problem is the need for ground truth to calculate r . In case of evaluation, the authors could fall back to national weather data as a ground truth. This need for ground truth limits the area in which it can be used.

A different strategy to ensure reliability for sensor measurements, although offline, is presented by D'Aniello et al. [7], in which the authors present a data imputation strategy based on association rule mining that actively harnesses redundancy. They introduce a semantic model to represent the quality of sensor measurements as annotations. Those annotations serve as a basis for the association rule mining for quality assurance and data imputation. The goal of the presented strategy is provide a framework that can deal with multiple or even competing quality requirements to improve the users' perceived quality. Sensor virtualization is used to express the different quality requirements. The evaluation shows that the proposed algorithm outperforms other common data imputation techniques, such as k -nn and singular value decomposition. The flexibility toward quality with semantic annotations caters toward our system, but the fact that it is designed for offline data imputation renders its use in an online system difficult.

The more common use case for harnessing redundancy in WSNs is presented by Xu et al. [27]. The idea is to find redundant routing paths rather than using redundant measurements. You can switch off multiple devices while still ensuring sufficient routing paths for energy conservation. In conjunction with approaches like the one presented by Gao et al. [10], which provides an analytical model to determine the degree of partial and full redundancy for optimal scheduling, it demonstrates the potential for harnessing redundancy even on lower layers like routing.

Fig. 1. coSense middleware overview.

Even though redundancy and its uses have been studied in WSNs and in the IoT, as has been presented, none of the approaches are comparable to our proposed coSense middleware. They do sometimes offer ways to increase accuracy but lack a way of controlling it, making it difficult to use them for a wide variety of algorithms. Only one of the presented systems was designed for managing devices, as it introduces a sensor virtualization, but that system is intended for offline use only. In general, none of the presented approaches fulfills all of our requirements.

4 COSENSE MIDDLEWARE

The goal of the coSense middleware is to fulfill the requirements mentioned in Section 2.1 by actively harnessing the plethora of IoT devices in a collaborative fashion. To do so, the coSense core is a centralized multistage fusion system: first it performs *unsupervised clustering* to group sensor nodes, and second it performs *information fusion* on each group of sensors individually. The general idea is outlined in our earlier work [22]. An overview of the system can be found in Figure 1. On the lowest layer, the middleware communicates with IoT systems, receiving measurements and sending control messages. On the next layer above, measurements and their corresponding devices are grouped together using unsupervised clustering (see Section 6). Grouping of the measurements is the data cleaning pre-process for information fusion: only similar measurements are grouped together so that outliers are filtered. Grouping also reduces the complexity of the information fusion, since the number of groupings is smaller than the total number of devices.

Individual groupings are then abstracted using the FTI information fusion function (see Section 5). Due to the FTI's ability to deal with any two-dimensional data, we designed the middleware to deal with the same data for sensors. FTI can be configured with two parameters n and f to either focus on higher accuracy or fault tolerance. Although users could select these parameters directly, our middleware provides a further simplification by providing them with different pre-defined QoS levels that are translated into individual values of n and f by a QoS mapping component (see Section 5.3). The desired QoS level can be selected for each group of sensors in our API, shown at the very top of the coSense middleware.

Beginning from the very bottom with a multitude of IoT devices, the complexity is decreasing and steadily moving toward the top of the middleware, with autonomous unsupervised grouping of devices, with further abstraction resulting in only a few abstracted sensing entities, easily controlled by simple QoS-level configurations.

4.1 Problem Analysis

To get a better understanding of coSense and how it fits the requirements from Section 2.1, a deeper understanding of the given requirements is necessary. Before the requirements can be discussed, we introduce some notation and definitions:

Given a set of sensors

$$S = \{s_1, \dots, s_n\}, \quad (1)$$

and a series of timestamps

$$T = \{t_1, \dots, t_p\}, \quad (2)$$

we can define a set of all measurements with timestamps as such:

$$X = \{(x_{s_i}, t_j) | x \in \mathbb{R}, s_i \in S, t_j \in T\}, \quad (3)$$

where x_{s_i} is the measurement of the i -th sensor at time t_j . Note that there can only be one measurement for one timestamp. For simplicity, every sensor provides a measurement for each timestamp. For notation purposes, we introduce the following shorthand:

$$(x_{s_i}, t_j) \hat{=} x_i^j. \quad (4)$$

(For readability, the timestamp can be omitted when, for instance, talking about two measurements from two different sensors. It is implied that the measurements happened at the same time, if not stated differently.)

Let σ be the deviation of the measurement from the true value as discussed in Section 2.1 (1). We can then define the range of a measurement to be the closed interval:

$$I_i^t = [x_i^t - \sigma, x_i^t + \sigma], \quad (5)$$

whereas $|I_i^t|$ describes the length of that interval (i.e., the maximal expected inaccuracy). Rather than viewed as a scalar value, the reported measurement from a sensor is viewed as a random pick from the interval dictated by the accuracy given in the datasheet. Under the assumption of the interval being correct and containing the true value, denoted as C ,

$$\frac{1}{|I_i^t|} = P(\tau | C) \quad (6)$$

gives you the probability of choosing the correct value τ when picking a random value from the interval without making any (wrong) assumptions about value distribution of the interval.

Compensate inaccuracy. Compensating inaccuracy can therefore be seen as an optimization over $|I_i^t|$ as described in (7): maximize the accuracy of sensor i at time t : minimize the interval and maximize the likelihood of picking out the true value from the interval.

$$\min |I_i^t| \Rightarrow \max P(\tau | C) \quad (7)$$

Compensate faults. Since faults are defined as outliers, a device can be viewed as faulty with respect to another device if

$$I_i^t \cap I_k^t = \emptyset, \quad (8)$$

which means that the intervals of measurements of devices i and k do not overlap and their values are farther apart than the maximal expected inaccuracy.

Manage devices. Managing devices entails reducing the complexity in terms of number of devices the application has to deal with. The goal is to abstract multiple devices as a singular one to reduce the complexity: finding all grouping of devices

$$\mathbb{C} \subset \mathcal{P}(S), \quad (9)$$

$$\forall c_i, c_j \in \mathbb{C} : c_i \cap c_j = \emptyset \text{ for } i \neq j, \quad (10)$$

where $\mathbb{C}$ is the set of all disjunct groupings and $\mathcal{P}(S)$ the powerset of all sensors with

$$\bigcup_{c \in \mathbb{C}} c = S \quad (11)$$

leading to

$$|\mathbb{C}| \leq |S|. \quad (12)$$

A grouping c is characterized by only containing devices with overlapping intervals. These groupings not only reduce the complexity but also aid in identifying faults to ease their compensation as pre-processing for the abstraction.

Provide flexibility and adaptation. Flexibility means being able to adapt to the applications's needs. In our case, we specifically adjust the accuracy of the reported measurements to support different machine learning algorithms. Those different algorithms can have different requirements for measurement quality. To provide such flexibility, we introduce a parameter f into Equation (7):

$$\max (P(\tau|C), f). \quad (13)$$

f allows us to adjust the degree of maximization of the probability (i.e., how small we make the interval from which we pick the value). It essentially is the degree of resilience against faults, which is inverse to the probability maximisation: the higher the resilience against faults, the lesser maximisation of the probability is possible. Furthermore, we introduce a QoS function mapping f to a pre-defined QoS level.

In the following sections, we take a closer look at each of the different stages of the coSense middleware and how they deal with problems imposed by the requirements. We first look at the information fusion process, as this has a direct influence on how we employ unsupervised clustering.

5 ABSTRACTION THROUGH THE FTI

The FTI function was introduced by Schmid and Schossmaier [23] and is an extension of the $\mathcal{M}$ function presented by Marzullo [17], as well as the *abstract sensor* concept from Marzullo [16]. The FTI function is primarily designed for clock synchronization, but due to its versatile nature in working with any two-dimensional data, it can be modified for sensor data.

5.1 $\mathcal{M}$ Function

The abstract sensor calculated by the $\mathcal{M}$ function is based on the idea that no sensor can measure a truly accurate value without any deviation. Any sensor measurement is treated as inaccurate and mapped to an interval whose borders are described by the sensors accuracy (i.e., the length of the interval is the degree of accuracy). The assumption is that the true value might not be represented by the measured value but is a part of the created interval [16] (i.e., the interval is believed to be correct). As discussed in Section 4.1, the probability to randomly pick the true value from the interval is defined by Equation (6). If you have multiple sensors measuring the same phenomenon, you can increase that probability: (i) create the intervals for each sensor, and (ii) combine those intervals into a singular one by taking the biggest intersection of all intervals. As the intervals are assumed to be correct, the biggest intersection must contain the true value. The resulting interval is smaller, which increases your probability [16]. In the literature, this is referred to as a fault-tolerant abstract sensor [16]. An abstract sensor is a software abstraction of physical sensors. The abstraction is created by performing sensor fusion on the measurements of multiple sensors. By fusing the measurements, you combine multiple sensors into a single, more powerful sensing entity. If the assumption that all intervals are correct is only partially fulfilled and faults can occur, the algorithm still works: let n be the number of all intervals and f be the number of faults you want to tolerate. The function $\mathcal{M}_n^f$ describing the abstract sensor dictates that the intersection of all intervals must contain at least $n - f$ intervals to still contain the true value. Tolerating up to f faults is bound by $f \leq \frac{n-1}{2}$, as $\mathcal{M}$ is a majority vote. If the abstract sensor tolerates the maximum number of faults (f) for the number of intervals (i.e., $n = 2f + 1$), then the length of the abstracted interval is at most the length of the smallest of the n individual intervals. The accuracy of that abstract sensor is that of the most accurate individual sensor (Equation (14)):

$$\mathcal{M}_{2f+1}^f \Rightarrow |\mathcal{M}_{2f+1}^f| \leq \min |I_{i \in n}|. \quad (14)$$

In this worst case, we do not gain any accuracy. But we also do not lose any either. In most cases, however, we can assume that less than the maximum number of faults will occur. In these cases, we will gain accuracy [16]. With the introduced fault tolerance and potential gain in accuracy, $\mathcal{M}$ is a suitable abstraction for process control being dependent on inaccurate and unreliable sensors.

5.2 $\mathcal{F}$ Function

Note that small changes in the interval borders can lead to great changes in the abstracted interval as shown in [23]. The reason is that the $\mathcal{M}$ function is not Lipschitz continuous. This can lead to unnecessary inaccuracies. To fulfill the Lipschitz condition, the FTI function $\mathcal{F}$ was developed in [23]. Rather than using the intersection of $n - f$ intervals, the FTI algorithm relies on the union of two intervals [23]:

$$[x, y] \cup [u, v] = [\min\{x, u\}, \max\{y, v\}]. \quad (15)$$

It is assumed that the intervals are compatible to each other—that is, they do not deviate drastically from each other. The FTI is then defined as follows:

$$\mathcal{F}_n^f = [(f + 1)\text{-max}\{x_1, \dots, x_n\}, (f + 1)\text{-min}\{y_1, \dots, y_n\}], \quad (16)$$

where $n \geq 1$ denotes the number of intervals of which at most $f < n$ can be faulty and $h\text{-max}\{\dots\}/h\text{-min}\{\dots\}$ give the h -largest/ h -smallest element of the respected set. This can be computed efficiently by sweeping over the intervals' left/right edges until you encounter the $(f + 1)$ -th one: for the left edge, you sweep from right to left (finding the max), and for the right edge from left to right (finding the min). This results in an overall complexity of $O(n \log n)$ [23]. Similar to $\mathcal{M}$, $\mathcal{F}$ can deal with a number of different faults each with its own maximum f . It can tolerate

Table 1. Maximum f for Each Different Type of Fault [23]

Fault Type	Max f
Omission	$n - 1$
Non-accurate	$\lfloor \frac{n-1}{2} \rfloor$
Unbounded	$\lfloor \frac{n-1}{3} \rfloor$

omission faults, in which intervals are missing due to lack of report; *non-accurate faults*, intervals whose values are not similar to any other (i.e., outliers); and *unbounded accuracy faults*, in which the reported interval is so big that it is basically useless. The number of tolerable faults is given in Table 1.

The fusion of intervals offers a low complexity but still powerful way of increasing the possible gain in the measurement's accuracy. In doing so, it provides a sophisticated way of detecting and dealing with faults. The parameters of f and n can be adapted to increase either the fault tolerance or the accuracy of the measurements. This provides the necessary flexibility to support a wide variety of applications. Additionally, the union of multiple intervals can be treated as an abstraction of multiple sensors into a singular entity (the abstracted sensor), easing the management from the application's perspective. It lacks, however, an easy way to filter out the compatible intervals as the solution presented in [23] does a simple but costly pairwise comparison. A better way is to provide compatible intervals to the FTI function. We do this with unsupervised clustering. Another issue is the management of parameters. Each abstracted sensor has its own set of parameters you need to keep track of. Rather than managing f and n individually, we provide a QoS-level abstraction. This maps an application's needs to the right parameters automatically.

5.3 QoS Mapping

The common use case for $\mathcal{F}$ in the IoT is to deal with non-accurate sensory data. The general bound for tolerable faults according to Table 1 is given by $\lfloor \frac{n-1}{2} \rfloor$. As $n \gg 0$ is usually the case and f is bounded by n , a possible QoS function with three levels can be defined as follows:

$$\text{QoS}(f, n) \rightarrow \{0, 1, 2\}, \quad (17)$$

$$\text{QoS}(f, n) = \begin{cases} 0, & f = \min f \\ 1, & f \in [\min f + 1, \max f - 1] \\ 2, & f = \max f \end{cases} \quad (18)$$

QoS level 0 denotes the most accurate measurement but is prone to faulty sensors. Level 2 is the least accurate but tolerates the highest number of faults. Level 1 represents a balance between them. The $\min f$ here describes the smallest f for which the FTI functions yields a valid result. Usually $\min f = 0$, but experiments have shown that there is the rare occasion where $\min f > 0$. The $\max f$ is defined according to Table 1. Such a QoS scheme is well known from communication protocols like TCP and MQTT. It provides the flexibility to change the measurements as need be, without the knowledge of the very specific individual abstracted sensors. This alleviates application design, as it does not force the application to keep track of the composition of every abstracted sensor.

6 UNSUPERVISED CLUSTERING

Clustering generally describes grouping of data into different groups. All data in the same group (cluster) is similar to each other and dissimilar to the data in a different cluster. Similarity is usually

a pre-defined property. This decision making makes it a classification problem. The most common example for clustering is the k -means algorithm. It divides a dataset into k groups. Each of those groups has a centroid. The centroid is the arithmetic mean of that group. It is calculated to have the minimal distance to each member of the cluster. For this algorithm, the specific property is the number of groups, k , which is pre-determined and given by the user. Classification is the act of assigning classes to data points within a dataset. Those classes are pre-defined as well as the properties a data point has to fulfill to be classified into that class. Therefore, classification is more constrained than clustering. Both clustering and classification can be unsupervised. The lack of supervision is expressed through a similarity function and its optimization without outside interference. In Equation (10), we define the set of cluster $\mathbb{C}$ to only contain a disjunct cluster. Each cluster c only contains devices with overlapping intervals—that is, they do not contain faulty devices with respect to the rest of that cluster (Equation (8)). We do this to facilitate the FTI and its adaptation.

6.1 Cluster Adaptation for the FTI

The nature of the FTI and its adaptation impose challenges on the clustering algorithm. To properly abstract a set of sensors with the FTI, a single cluster may only contain compatible intervals upon its creation (i.e., measurement intervals that overlap). This implies, however, that certain *a priori* knowledge cannot be assumed: the total number of clusters and the individual size of a cluster. Instead, you want to find the natural clustering in which each cluster adheres to our overlap constraint. Therefore, a certain family of algorithms, namely the ones requiring *a priori* knowledge, is not well suited for that task. The most common member of that family is the k -means clustering algorithm. As mentioned earlier: to cluster the input data, it requires the number of cluster, k , a priori. Even though there are techniques for determining a suitable k , they usually entail an overhead. You have to run multiple times and make a decision based on an abstract measure. However, that measure is specific to the algorithm. For k -means, it is the root mean square error of the distance to a cluster centroid. This does not necessarily reflect an overlap of measurement intervals. If, for instance, a single sensor does not overlap with any other sensor, it should not be abstracted using the FTI, as information will be lost. In case of a fixed number of clusters, that specific sensor has to be added to a cluster. It may or may not end up in a cluster that has other members, but that is solely dependent on the total number of possible clusters. If that number is not high enough, it will end up in a cluster with other sensors of which none share an overlap. Abstracting with the FTI on that specific cluster will result in a useless abstracted sensor, as no gain in accuracy or reliability can be obtained. However, it will perform worse and possibly report false measurements.

Similar to the number of clusters, the actual size of a cluster, upon its creation, cannot be determined *a priori*. The reasoning is the same: a pre-determined cluster size forces sensors to be clustered into a cluster, perhaps independent of an existing overlap. This will again result in a worse-performing abstracted sensor. The overarching property is the overlap, which can be further refined.

6.2 Quasi-Redundancy

Frolík et al. [9] coined the term *quasi-redundancy*. It denotes two sensors being quasi-redundant if they show a well-correlated behavior over a defined period of time. In their work, they used the statistical correlation coefficient of 0.85 and greater as a measure for quasi-redundancy. Correlation, however, can be interpreted differently. We introduce $\sim$ to express the quasi-redundant relationship and define quasi-redundancy for our purposes to be

$$A \sim B \Leftrightarrow \text{dist}(A, B) \leq \delta, \quad (19)$$

where A and B are scalar values, dist is a function that calculates a distance between two values, and $\delta > 0$. If $\delta = 0$, it would be the definition of true (linear) redundancy. In our case, quasi-redundancy describes the distance between measurements of two sensors as a means of correlation. It is, however, not limited to a simple distance function but can be defined to be a complex (mathematical) metric including, for instance, a spatial or temporal model. For our purposes, that means

$$x_i \sim x_j \Leftrightarrow \text{dist}(x_i, x_j) \leq \delta, \quad (20)$$

and we define two measurements to be quasi-redundant if their distance is below threshold δ .

6.2.1 Remotely Quasi-Redundant. With quasi-redundancy being reliant on a distance function, and assuming A and B are measurements, it is trivial to see the following:

$$A \sim B \Leftrightarrow B \sim A. \quad (21)$$

The problem arising from that definition is transitivity, as described in Equation (22):

$$A \sim B, B \sim C \Rightarrow A \sim C. \quad (22)$$

If $\text{dist}(A, C) \leq \delta$, the implication is true. However, if the distance $\text{dist}(A, C) > \delta$, A and C cannot be considered quasi-redundant and we call C to only be *remotely* quasi-redundant to A . The task of any clustering algorithm for the coSense middleware therefore is: find all groupings of quasi-redundant sensors while making sure that remotely quasi-redundant sensors are excluded from the group.

6.3 Community Detection

A possible approach to clustering for coSense is community detection in a graph. A community (cluster) in this case is a strongly connected component of the graph—that is, a subgraph where all vertices are interconnected to all other vertices in that component. Ideally, you end up with multiple communities that are entirely disconnected from each other. The size of the community is irrelevant, and communities with only one member are a valid result. To map our problem to a graph, we use the following method of construction: (i) a vertex represents a measurement, and (ii) an edge between two vertices represents fulfilling the quasi-redundancy relationship according to Equation (19). If we set $\delta \leq 2\sigma$ to be smaller than the maximal expected inaccuracy of the sensors, an edge between measurements is placed when their corresponding intervals overlap. Mapped to a graph, the remotely quasi-redundant sensor is represented by (i) a single vertex that is connected to a community by only few edges, (ii) a single edge that connects the otherwise disconnected communities. Instead of identifying the remotely quasi-redundant sensors based on their value, you are looking for specific edges in a graph expressing the relation in a more direct fashion.

6.3.1 Edge Betweenness. Community detection is a well-established problem in graph theory, but it only found reasonable application with the introduction of *edge betweenness* from Newman and Girvan [21]. Edge betweenness assigns scores to edges based on the connectivity. The most well-known approach uses the shortest-paths metric: the edge betweenness is based on the overall contribution of an edge to all shortest paths in the entire graph. That means that a single edge connecting multiple vertices (i.e., an edge connecting two communities) will get assigned a high score, since all shortest paths connecting the two communities must traverse this specific edge. It is similar for a single vertex that is connected to a community through a single edge, as all shortest paths to the singular vertex must traverse that one edge. With the help of edge betweenness, the *Girvan-Newman* algorithm was developed [21]. It is an iterative approach that calculates the edge betweenness for each edge and removes the edge with the highest score. After each edge

removal, the edge betweenness is recalculated because the number of edges has been reduced and therefore the shortest paths have changed. Since edges directly translate to our quasi-redundancy, the Girvan-Newman algorithm is quite promising. Additionally, edge betweenness is calculated on the entire graph yielding the global knowledge to distinctly identifying remote quasi-redundancy. The algorithm, however, has two fundamental problems: (i) determining the quality of a partitioning (clustering) (i.e., when to terminate the algorithm), and (ii) the iterative recalculation of edge betweenness leads to a high complexity of $O(n^3)$ making it computationally infeasible for even smaller graphs [21].

6.3.2 Modularity. To alleviate the computational load for community detection and provide a metric for determining the quality of a graph partition, Newman [19, 20] developed *modularity* (Q). Modularity is a scalar value defined as $Q \in [-1, 1]$. Given any partitioning, it describes the fraction of edges within communities minus the expected fraction of edges, if placed at random [19]. As such, Q indicates the quality of the given partitioning compared to a random graph of the same degree (i.e., the same number of vertices and edges). The modularity for a graph is calculated according to Equation (23), where e_{ii} is the fraction of edges in community i and a_i^2 is the fraction of all edges connected to community i , but placed at random. A high modularity, $Q \geq 0.3$, indicates dense connections within communities and sparse connections between communities.

$$Q = \sum_i (e_{ii} - a_i^2) \quad (23)$$

The idea behind modularity-based community detection is to optimize Q over all possible partitions and maximize it. It is easy to see that an optimization over all possible partitions is computationally infeasible, even for small graphs. To still use modularity for optimization, Newman suggests greedy algorithms that do not calculate the global optimum of Q for the entire graph but instead focus on the change of modularity, if you were to merge two communities [19]. In the literature, this change of modularity is known as ΔQ . One such example would be hierarchical agglomerative clustering. At the beginning, each vertex is its own community and you iteratively merge communities if $\Delta Q > 0$. A modularity gain of $\Delta Q \leq 0$ indicates that merging these communities will have no effect or even a negative effect on the overall modularity of the graph. Such a greedy algorithm runs on average in $O((m+n)n)$, where m is the number of edges and n the number of vertices. There are several optimizations to such greedy algorithms to further improve the complexity (e.g., [24, 25]).

Community detection presents a viable option for the unsupervised clustering as it has a deterministic optimization function at a feasible computation cost. Furthermore, the modeling of the graph allows for a high detection rate of the remote quasi-redundancy problem. This enables community detection to produce a clustering beneficial for the FTI function.

7 EVALUATION

We evaluate coSense with a sensor dataset provided by the Intel Berkeley Research Lab.¹ The dataset consists of about 2.3 million readings from 54 WSN motes, deployed over multiple weeks within their facilities and measuring various environmental phenomena like temperature, humidity, and light. For the sake of simplicity, we focus only on temperature readings, as those readings are easy to understand compared to the others. But coSense works with other types of readings as well. First, we need to define our two parameters δ and σ . The distance threshold δ is used for defining the quasi-redundancy relationship used in our clustering. The sensor inaccuracy σ is used for the FTI function. Since all sensors are the same, the same σ can be applied to all sensors

¹<http://db.csail.mit.edu/labdata/labdata.html>.

universally. We set σ to the inaccuracy reported in the data sheet of the sensor, $\sigma = 0.4$. To maximize the possible intersection of two intervals, we set $\delta = \frac{\sigma}{2}$ (i.e., $\delta = 0.2$). This has been shown to improve the quality of our clustering in earlier experiments. If we were using any of the other environmental features, we simply would have to define σ and δ according to the sensors: σ for the inaccuracy and δ as the distance threshold.

As our evaluation metric, we use a measure called *accuracy gain* γ . We define γ in more detail in the following. Note that for simplicity, our definition makes use of the fact that all sensors use the same σ . This can be extended to more complex cases if needed.

7.1 Evaluation Metric γ

Equation (6) defines the probability of picking out the true value, $P(\tau|C)$, at random for the general case for any given interval I_i^j . Since the FTI function yields an interval, $I_{\mathcal{F}_n}^j$, we can define the probability of picking out the true value at random from the said FTI to be

$$P^{\mathcal{F}}(\tau|C) = \frac{1}{|I_{\mathcal{F}_n}^j|}. \quad (24)$$

Since the deviation σ is equal for all sensors (in our example), $P'(\tau|C)$ describes the probability of picking out the true value from any interval created by a singular sensor. With that,

$$\frac{P^{\mathcal{F}}(\tau|C)}{P'(\tau|C)} - 1 = \gamma \quad (25)$$

describes the accuracy gain γ as the factor of how the probability to pick the true value for the abstracted sensor increases compared to the probability of a single sensor. This means that the accuracy gain always compares the abstracted sensor to a normal sensor. We have chosen this approach due to the lack of other comparable approaches, as is pointed out in Section 3.

Note that, depending on the value of γ , we can distinguish three distinct cases:

$$\gamma \begin{cases} < 0 & \text{The abstracted sensor is highly inaccurate or invalid (see Section 7.4).} \\ = 0 & \text{The abstracted sensor performs equal to a single sensor.} \\ > 0 & \text{The abstracted sensor outperforms a single sensor.} \end{cases} \quad (26)$$

For our experiments, we took an excerpt from the dataset spanning 24 hours. We divided the day into 5-minute-long time windows so that every mote has had the chance to contribute at least one measurement. Due to congestion, messages were lost frequently. We use these 5-minute windows to run the community detection followed by the FTI function, meaning that for each 5-minute window, the coSense middleware reevaluates and recreates all abstracted sensors. Before we evaluate coSense, however, we briefly discuss our community detection algorithm.

7.2 Community Detection Algorithm

As mentioned in Section 6.3, we can map our clustering problem to community detection. The advantage of community detection is the sole reliance on the data itself rather than needing *a priori* knowledge. Mapped to a graph, the relationship defined in Equation (19) is mapped to edges. By optimizing the modularity of that graph, we can detect communities (clusters) based on their relationship to each other. In essence, our implementation conforms to the greedy strategy suggested by Newman [19]: we iteratively merge two communities based on the modularity gain of that merge, starting with every vertex as its own community. Note that a vertex can only be in one community, as stated in Equation (10). Thus, similar to other modularity-based greedy algorithms, we need to decide into which community we should merge a vertex, if both merges would result

Fig. 2. Accuracy gain for clusters containing Mote 2 for the entire day for QoS level 0.

in an equal modularity gain. Since an optimization over all possible partitions is computational infeasible, it is not possible to judge which decision will yield the highest overall gain in a later step. Therefore, we merge with the smaller community as suggested in the literature to increase the overall modularity in previous evaluations [25]. This might still, however, result in a false-positive decision.

We use Apache Spark² for its powerful distributed graph processing framework GraphX³ for our implementation. GraphX offers a property graph abstraction with a variety of pre-built transformation for performance optimizations. Similar to Apache Spark, GraphX performs most of its calculations in a distributed manner, allowing for easy scalability as your system is growing. A similar algorithm for community detection implemented for Apache Spark is presented in the work of Ling et al. [15] but does not yield the actual communities with their members, which are essential for the coSense middleware to work. Other implementations (e.g., [25]) require specific hardware, which was not available to us or not available at the time of this writing [24].

7.3 Compensating Inaccuracy

In the following sections, we evaluate coSense and discuss how well it fulfills our initial requirements, namely (i) compensating for inaccuracy, (ii) compensating for faulty sensors, (iii) increasing flexibility, and (iv) reducing management complexity.

Our first requirement deals with the inaccuracy of sensors and how to compensate for it. It aims at increasing the accuracy by minimizing the abstracted interval to increase the chance of picking the true value at random (i.e., our accuracy gain γ). According to Equation (26), our approach is beneficial if $\gamma > 0$, with bigger values of γ denoting better results. Figure 2 shows the gain (γ) for

²<http://spark.apache.org/>.

³<http://spark.apache.org/graphx/>.

Table 2. Average Accuracy Gain Per Mote for QoS Level 0 for the Entire Excerpt

Mote	Average Gain	Mote	Average Gain	Mote	Average Gain
1	10.5	19	7.701	37	10.164
2	7.591	20	5.778	38	8.842
3	6.607	21	7.238	39	10.266
4	7.078	22	4.045	40	10.429
5	n/a	23	6.07	41	6.098
6	7.762	24	3.52	42	6.06
7	8.144	25	5.015	43	7.607
8	10.523	26	6.202	44	6.665
9	8.667	27	5.447	45	8.409
10	8.194	28	n/a	46	8.576
11	6.421	29	8.466	47	7.731
12	4.335	30	6.109	48	9.101
13	5.274	31	9.09	49	8.839
14	9.972	32	7.635	50	4.971
15	0.0	33	8.85	51	9.02
16	7.656	34	7.012	52	8.53
17	10.73	35	10.307	53	2.979
18	9.539	36	9.896	54	2.787

Note: Motes 5 and 28 are missing in the original dataset.

the clusters containing Mote 2 at QoS level 0 compared to using Mote 2 individually. On average, our approach increases the sensing accuracy over Mote 2 by $\gamma = 7.6$ (i.e., the probability of picking the true value is increased more than seven times). We can also see that the achieved accuracy gain fluctuates widely between 0 (no gain) and approximately 160. This fluctuation is due to the coSense middleware constantly adapting and reevaluating the abstracted sensor that includes Mote 2: it tries to find the best clustering and abstraction for every time window to optimize the achievable gain. Sometimes measurements vary a lot and the resulting cluster is small. Sometimes measurements are very similar and thus cannot be improved a lot. Our approach is able to adapt to such changing situations very dynamically.

Our results also differ between motes. Table 2 shows the average accuracy gain per mote for QoS level 0. Note that we excluded Mote 5 and 28, as our dataset did not include any measurements done by them. Another mote worth mentioning is Mote 15 with an average gain of $\gamma = 0.0$. Upon closer inspection, it turns out that Mote 15 was faulty and only contributed measurements to less than 10 time windows. In those windows, it was singled out as a singular cluster only containing itself, resulting in no possible gain. All other motes show a consistent positive gain between 2.8 (Mote 54) and 10.73 (Mote 17).

Gain for varying f and QoS levels. As discussed before, increasing f allows to tolerate an increasing number of faulty sensors but reduces the achieved accuracy. To show how different f influence the accuracy gain γ , Figure 3 plots γ for different f for randomly picked clusters of different sizes. For this experiment, f is only bound by the actual cluster size n (i.e., $0 \leq f < n$). The horizontal line depicts $\gamma = 0$. Anything lower means that we lost accuracy. The square ticks depict a cluster with $n = 5$, the circle ones a cluster with $n = 10$, and the triangle ones a cluster with $n = 15$. These sizes have been chosen since they occurred frequently in our experiments.

As expected, we can see γ decreasing steadily with larger f , with the largest gain for $f = 0$. Interestingly, for each cluster size, we eventually reach an f for which γ becomes negative. At this

Fig. 3. Exemplary *gain* per *f* for maximum *f* for three clusters of different sizes encoded as shapes (triangle: $n = 15$; circle: $n = 10$; square: $n = 5$). Horizontal line depicts the 0 gain mark.

Table 3. $\max f$ for Different Cluster Sizes and Gain, γ for $\max f$

n	$\max f \left(\mathcal{F}_n^f \right)$	$\gamma \left(\mathcal{F}_n^{\max f} \right)$
5	2	0
10	4	0.05
15	7	0

point, we start losing accuracy. Looking into the results in more detail, we assume that the largest f for which we did not yet get such a negative gain is given as follows:

$$\max f \left(\mathcal{F}_n^f \right) = \left\lfloor \frac{n-1}{2} \right\rfloor \Rightarrow \gamma \geq 0. \quad (27)$$

Looking further into these cases for the different cluster sizes, we can see in Table 3 that the gain for these maximum possible f is at least 0. This proves our assumption in Equation (27), at least for the exemplary cluster. Upon closer inspection of our results, we found that our assumption is true for all communities in our experiment. Of the 3,193 inspected communities, 1,139 have a positive gain and 2,054 have a gain of zero for $\max f$. These values for $\max f$ are used for QoS level 2. As the parameters for QoS level 1 lie in between those of levels 0 and 2, a positive gain can be expected. The consistent positive accuracy gain across all QoS levels demonstrates the ability of coSense to compensate inaccuracy.

7.4 Compensating Faults

We defined faulty devices to be outliers in a majority vote. To compensate for them, you have to identify that a fault is present. The FTI function can easily identify and compensate faulty devices.

Fig. 4. Gain and interval length per f . Circles represent the accuracy gain and strokes the interval length. Horizontal line depicts the 0 gain mark.

In our case, $\gamma < 0$ indicates a faulty device. In the general case, where the γ function is not an option, you can use a different method to identify faults: the case of $\gamma < 0$ is equivalent to $|I_{\mathcal{F}_n}^j| < 0$ (i.e., an invalid abstracted sensor). Once you have detected such an invalid sensor, simply increase f to compensate for it until the interval length becomes at least zero.

Negative γ for $f = 0$. With the way the FTI function works, the intuition for the gain γ is to be maximal for $f = 0$. This is consistent with our results shown so far and was true for most cases. However, there is a small set of abstracted sensors that show the behavior depicted in Figure 4, having a $\gamma < 0$ for $f = 0$. The reason for this becomes apparent when looking at the resulting interval length of the FTI function. They are also negative, represented by the strokes in Figure 4. A negative interval length indicates that not all sensors in that specific cluster are quasi-redundant (see Equation (19)) and violate the conditional transitivity described in Equation (22). This is also evident from the fact that there is a massive gain increase when increasing f as this filters out the sensors violating the relation. This behavior is present for about 4.5% of all groupings in the examined data. All of those instances share the same shape of a $\frac{1}{x^2}$ function pointing again toward a suboptimal clustering. The most likely reason is the decision problem encountered when two community merge possibilities with equivalent modularity gain are present, as described in Section 7.2. To correct that problem, it is possible to adjust f up to the point of a positive gain, which can also be done autonomously to not propagate false information.

Autonomous adaptation. The potential for autonomous adaption of f becomes apparent in Figure 5, where the accuracy gain for two distinct sensor nodes over time is plotted. Note that our results so far already include autonomous adaptation. For this experiment, we deactivated this explicitly. The compared sensor nodes are selected to be distinct over the entire shown time period (i.e., they are always in different clusters). The horizontal lines depict the average gain. The left part of the figure shows the gain without autonomous adaptation and the right part with it. The cluster containing Mote 1 experiences multiple fluctuations resulting in a negative gain and an average gain of $\gamma = -0.05$. The cluster containing Mote 22 does not experience such fluctuations and has an average gain of $\gamma = 0.7$. When we use the autonomous adaptation of f , the cluster containing Mote 1 can increase its average gain about six times to $\gamma = 3.94$.

If f cannot be increased because the limit of the tolerable faults is reached, the entire sensor has degraded and needs to be reconstructed—that is, a full reevaluation by the coSense middleware is necessary. The reclustering will find a more suitable grouping so a stable abstracted sensor can be constructed. Figure 3 combined with Table 3 suggests, however, that in a non-degraded

Fig. 5. Left: Accuracy gain for the clusters containing Motes 1 and 22 without dynamic adaptation. Right: Accuracy gain for the clusters containing Motes 1 and 22 with dynamic adaptation. Horizontal lines depict the mean.

state, increasing f to its maximum is a valid option. This is useful if there is currently a need for information and a full reevaluation by the coSense middleware is not possible. This can be the case due to timing constraints, for instance.

7.5 Increasing Adaptability and Flexibility

The adaptability is relying on the FTI's ability to change the parameter f . Theoretically, f can always be increased, as long as $f < n$, but once the mark of $\max f$ is passed, it yields less desirable results. To omit the problem of passing $\max f$, the QoS-level abstraction deals with the adaptation of f to alleviate an application from knowing each $\max f$ individually. At the same time, it does allow for enough flexibility to suit applications with varying needs. Figures 3, 4, and 5 all show adaptations of f demonstrating the flexibility of the coSense middleware.

7.6 Easing Device Management

Finally, the FTI constructs abstracted sensors based on the groupings provided by the unsupervised clustering. These abstracted sensors can be viewed as a single sensing entity: an application does not have to deal with its individual (physical) sensors. In the entire experiment, 3,193 clusters were created, compared to 14,976 requests to individual sensors. In each time window, clusters were created from up to 52 motes. The number of created clusters varied between $|\mathbb{C}| = [4, 24]$ with a single cluster size varying between $|c| = [1, 39]$. The overall average number of clusters is around 11 with an average cluster size of 4. In all combinations, the resulting number of abstracted sensors is at least halved compared to the number of physical devices, but usually less. This reduces the complexity for an application to deal with devices severely. Combined with the QoS approach, the application does not need to keep track of every possible sensor configuration. It can focus on retrieving data from a smaller set of sensors. This makes the coSense middleware a suitable abstraction for large-scale IoT systems.

8 CONCLUSION AND FUTURE WORK

In this article, we presented coSense—the collaborative, fault-tolerant, and adaptive sensing middleware for the IoT. It combines unsupervised clustering with sensor fusion to actively harness the greatest asset of the IoT—the sheer number of devices. In doing so, coSense is able to increase the accuracy of measured sensor data and cope with faulty sensors. An API based on abstracted sensors and application selectable QoS levels reduces complexity for application developers and allows to tailor the middleware to different application needs. Our evaluation shows that coSense is

able to provide an average accuracy gain of 2.8 to 10.7 for sensor devices in a real-world dataset. In addition, using a straightforward autonomous parameter adaptation approach, accuracy is never reduced.

This shows the great potential of our approach for applications that require high-quality IoT data, such as data analysis algorithms and machine learning. In future work, we plan to investigate how to extend our approach with a more sophisticated approach to initiate reevaluations of clusters. In addition, we plan to explore the potential of our approach to save energy for IoT devices by coordinating the sleep schedules of clustered devices, effectively simulating failing devices and using coSense to handle them accordingly.

REFERENCES

- [1] Furqan Alam, Rashid Mehmood, Iyad Katib, Nasser N. Albogami, and Aiiad Albeshri. 2017. Special section on trends and advances for ambient intelligence with Internet of Things (IOT) systems, data fusion and IoT for smart ubiquitous environments: A survey. *IEEE Access* 5 (2017), 9533–9554. DOI : <https://doi.org/10.1109/ACCESS.2017.2697839>
- [2] Archana Barde and Sweta Jain. 2018. A survey of multi-sensor data fusion in wireless sensor networks. In *Proceedings of the 3rd International Conference on Internet of Things and Connected Technologies (ICIoTCT '18)*. DOI : <https://doi.org/10.2139/ssrn.3167286>
- [3] Abder Rezak Benaskeur. 2002. Consistent fusion of correlated data sources. *IECON Proceedings (Industrial Electronics Conference)* 4 (2002), 2652–2656. DOI : <https://doi.org/10.1109/IECON.2002.1182812>
- [4] João B. Borges Neto, Thiago H. Silva, Renato Martins Assunção, Raquel A. F. Mini, and Antonio A. F. Loureiro. 2015. Sensing in the collaborative Internet of Things. *Sensors (Switzerland)* 15, 3 (2015), 6607–6632. DOI : <https://doi.org/10.3390/s150306607>
- [5] Thomas Clouqueur, Parameswaran Ramanathan, Kewal K. Saluja, and Kuang-Ching Wang. 2001. Value-fusion versus decision-fusion for fault-tolerance in collaborative target detection in sensor networks. In *Proceedings of the 4th International Conference on Information Fusion*.
- [6] Daniel Ioan Curiac, Constantin Volosencu, Dan Pescaru, Lucian Jurca, and Alexa Doboli. 2009. Redundancy and its applications in wireless sensor networks: A survey. *WSEAS Transactions on Computers* 8, 4 (2009), 705–714.
- [7] Giuseppe D'Aniello, Matteo Gaeta, and Tzung-Pei Hong. 2017. Effective quality-aware sensor data management. *IEEE Transactions on Emerging Topics in Computational Intelligence* 2, 1 (2017), 65–77. DOI : <https://doi.org/10.1109/tetci.2017.2782800>
- [8] Geetika Dhand and S. S. Tyagi. 2016. Data aggregation techniques in WSN:Survey. *Procedia Computer Science* 92 (2016), 378–384. DOI : <https://doi.org/10.1016/j.procs.2016.07.393>
- [9] Jeff Frolik, Mohamed Abdelrahman, and Parameswaran Kandasamy. 2001. A confidence-based approach to the self-validation, fusion and reconstruction of quasi-redundant sensor data. *IEEE Transactions on Instrumentation and Measurement* 50, 6 (2001), 1761–1769. DOI : <https://doi.org/10.1109/19.982977>
- [10] Yong Gao, Kui Wu, and Fulu Li. 2003. Analysis on the redundancy of wireless sensor networks. In *Proceedings of the 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '03)*. 108–114. DOI : <https://doi.org/10.1145/941365.941366>
- [11] Gartner. n.d. IoT Device Estimate. Retrieved September 10, 2020 from <https://www.gartner.com/en/newsroom/press-releases/2018-11-07-gartner-identifies-top-10-strategic-iot-technologies-and-trends>.
- [12] Paulo Jesus, Carlos Baquero, and Paulo Sergio Almeida. 2015. A survey of distributed data aggregation algorithms. *IEEE Communications Surveys & Tutorials* 17, 1 (2015), 381–404. DOI : <https://doi.org/10.1109/COMST.2014.2354398>
- [13] Joint Committee for Guides in Metrology (JCGM). 2008. *JCGM 200 :2008: International Vocabulary of Metrology—Basic and General Concepts and Associated Terms(VIM)*. International Organization for Standardization, Geneva, Switzerland. DOI : [https://doi.org/10.1016/0263-2241\(85\)90006-5](https://doi.org/10.1016/0263-2241(85)90006-5)
- [14] Manish Kumar, Devendra P. Garg, and Randy A. Zachery. 2006. A generalized approach for inconsistency detection in data fusion from multiple sensors. In *Proceedings of the 2006 American Control Conference*. 2078–2083. DOI : <https://doi.org/10.1109/acc.2006.1656526>
- [15] Xiao Ling, Jiahai Yang, Dan Wang, Jinfeng Chen, and Liyao Li. 2017. Fast community detection in large weighted networks using GraphX in the cloud. In *Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications, the 14th IEEE International Conference on Smart City and the 2nd IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS'16)*. 1–8. DOI : <https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0012>
- [16] Keith Marzullo. 1990. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems* 8, 4 (Nov. 1990), 284–304. DOI : <https://doi.org/10.1145/128733.128735>

- [17] Keith Ansel Marzullo. 1984. *Maintaining the Time in a Distributed System: An Example of a Loosely-Coupled Distributed Service (Synchronization, Fault-Tolerance, Debugging)*. Ph.D. Dissertation. Stanford University, Stanford, CA.
- [18] Eduardo F. Nakamura, Antonio A. F. Loureiro, and Alejandro C. Frery. 2007. Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Computing Surveys* 39, 3 (2007), 9–es. DOI : <https://doi.org/10.1145/1267070.1267073>
- [19] M. E. J. Newman. 2004. Fast algorithm for detecting community structure in networks. *Physical Review E: Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics* 69, 6 (2004), 5. DOI : <https://doi.org/10.1103/PhysRevE.69.066133> arxiv:cond-mat/0408263
- [20] M. E. J. Newman. 2006. Modularity and community structure in networks. *PNAS* 103, 23 (2006), 8577–8582.
- [21] M. E. J. Newman and M. Girvan. 2003. Finding and evaluating community structure in networks. *Physical Review E* 69 (2003), 026113. DOI : <https://doi.org/10.1103/PhysRevE.69.026113>
- [22] Stephan Schmeisser and Gregor Schiele. 2017. Adaptive aggregation of redundant sensor data in the Internet of Things. In *Proceedings of the 2016 International Conference on Computational Science and Computational Intelligence (CSCI'16)*. 1387–1388. DOI : <https://doi.org/10.1109/CSCI.2016.0262>
- [23] Ulrich Schmid and Klaus Schossmaier. 2001. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing* 14, 2 (2001), 101–111. DOI : <https://doi.org/10.1007/PL00008927>
- [24] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. 2013. Fast algorithm for modularity-based graph clustering. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*. 1170–1176. DOI : <https://doi.org/10.1542/peds.2005-2834>
- [25] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. 2015. SCAN++: Efficient algorithm for finding clusters, hubs and outliers on large-scale graphs. *Proceedings of the VLDB Endowment* 8, 11 (2015), 1178–1189. <http://www.vldb.org/pvldb/vol8/p1178-shiokawa.pdf>.
- [26] Tony Sun, Ling-Jyh Chen, Chih-Chieh Han, and Mario Gerla. 2005. Reliable sensor networks for planet exploration. In *Proceedings of the 2005 IEEE Conference on Networking, Sensing, and Control*. IEEE, Los Alamitos, CA, 816–821. DOI : <https://doi.org/10.1109/ICNSC.2005.1461295>
- [27] Ya Xu, John Heidemann, and Deborah Estrin. 2001. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01)*. ACM, New York, NY, 70–84. DOI : <https://doi.org/10.1145/381677.381685>
- [28] Juanjuan Zhao, Yongxing Liu, Yongqiang Cheng, Yan Qiang, and Xiaolong Zhang. 2014. Multisensor data fusion for wildfire warning. In *Proceedings of the 2014 10th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN'14)*. 46–53. DOI : <https://doi.org/10.1109/MSN.2014.13>

Received August 2019; revised March 2020; accepted April 2020