SensorClone: A Framework for Harnessing Smart Devices with Virtual Sensors

Huber Flores
University of Helsinki
huber.flores@helsinki.fi

Pan Hui
HKUST
University of Helsinki
pan.hui@helsinki.fi

Sasu Tarkoma
University of Helsinki
sasu.tarkoma@helsinki.fi

Yong Li
Tsinghua University
liyong07@tsinghua.edu.cn

Theodoros Anagnostopoulos
Athens University of Applied Sciences
thanags@di.uoa.gr

Vassilis Kostakos
The University of Melbourne
vassilis.kostakos@unimelb.edu.au

Chu Luo
The University of Melbourne
chu.luo@unimelb.edu.au

Xiang Su
University of Oulu
xiang.su@oulu.fi

ABSTRACT

IoT services hosted by low-power devices rely on the cloud infrastructure to propagate their ubiquitous presence over the Internet. A critical challenge for IoT systems is to ensure continuous provisioning of IoT services by overcoming network breakdowns, hardware failures, and energy constraints. To overcome these issues, we propose a cloud-based framework namely *SensorClone*, which relies on *virtual devices* to improve IoT resilience. A virtual device is the digital counterpart of a physical device that has learned to emulate its operations from sample data collected from the physical one. SensorClone exploits the collected data of low-power devices to create virtual devices in the cloud. SensorClone then can opportunistically migrate virtual devices from the cloud into other devices, potentially underutilized, with higher capabilities and closer to the edge of the network, e.g., smart devices. Through a real deployment of our SensorClone in the wild, we identify that virtual devices can be used for two purposes, 1) to reduce the energy consumption of physical devices by duty cycling their service provisioning between the physical device and the virtual representation hosted in the cloud, and 2) to scale IoT services at the edge of the network by harnessing temporal periods of underutilization of smart devices. To evaluate our framework, we present a use case of a *virtual sensor* created from an IoT service of temperature. From our results, we verify that it is possible to achieve unlimited availability up to 90% and substantial power efficiency under acceptable levels of quality of service. Our work makes contributions towards improving IoT scalability and resilience by using virtual devices.

MMSys'18, June 12–15, 2018, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5192-8/18/06…$15.00
https://doi.org/10.1145/3204949.3204952

CCS CONCEPTS

• **Computer systems organization** → **Client-server architectures**; **Embedded systems**; *Redundancy*; • **Networks** → Network reliability;

KEYWORDS

Cloud computing, Internet of Things, Virtual Sensor, Edge Computing, Resilience, Opportunistic Migration

ACM Reference Format:
Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Theodoros Anagnostopoulos, Vassilis Kostakos, Chu Luo, and Xiang Su. 2018. SensorClone: A Framework for Harnessing Smart Devices with Virtual Sensors. In *MMSys'18: 9th ACM Multimedia Systems Conference, June 12–15, 2018, Amsterdam, Netherlands.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3204949.3204952

1 INTRODUCTION

The vision of *Internet of Things (IoT)* [2] is gradually becoming a reality[1]. Currently, a large number of low-power devices (e.g., sensors and actuators) can be connected by heterogeneous networks with ease. This is possible due to the advances in cloud computing technologies, which provide the *"as-a-service"* platforms to easily develop and deploy applications on the fly. For 2020, the installed base of IoT devices is forecast to grow to almost 31 billion worldwide[2]. Thus, to improve IoT resilience, non-centralized systems and architectures need to be investigated.

As shown in Figure 1, a typical IoT environment consists of large-scale deployments of low-power devices, e.g., Arduino[3], Raspberry pi[4], that lack scalable resources (e.g., battery, memory, communication, and CPU) to provision its services continuously to others (1). One way to overcome this scarcity is to use the cloud to propagate devices' presence in a ubiquitous manner, such that a device's services can be consumed anytime and anywhere through the cloud by other devices or IoT services. Unfortunately, this approach is

[1] https://www.technologyreview.com/business-report/the-internet-of-things/
[2] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/
[3] https://www.arduino.cc/
[4] https://www.raspberrypi.org/

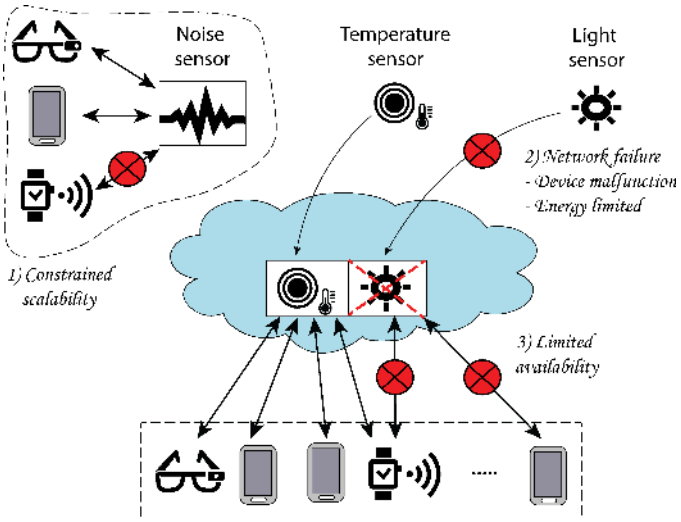

Figure 1: Internet of disconnected-things.

vulnerable to network failures, communication latency, device malfunction and increased energy consumption (2). As a result, a device is likely to become intermittently available, and at times unreachable by other devices (3).

Overcoming intermittent availability is a critical challenge in an IoT environment, because IoT applications are created spontaneously through interdependent relations between IoT services. Moreover, intermittent availability caused by drastic changes in communication (increased latency) harms the energy and performance of devices. Thus, a transparent mechanism is needed to ensure that IoT does not become an *Internet of disconnected-Things*.

Since devices need to keep continuous connectivity to cloud to propagate or access IoT services, oscillating communication latency remains a major concern for improving the battery life and performance of low-power devices. Several work has investigated how to overcome the provisioning issues of IoT services by reducing data transferred, fixing communication protocols and optimizing computational operations (e.g., duty cycle) of devices [1, 12, 16, 18]. More recent solutions have attempt to create virtual representations of devices in the cloud by collecting and analyzing data from physical devices (e.g., *Digital Twin*[5] of IBM). Virtual devices are used mainly for troubleshooting applications in time and detecting device's malfunction in different simulated environments. In this paper, we argue that virtual devices can be used further to improve IoT resilience by helping physical devices to extend their continuous operations and ubiquitous presence.

On the other hand, smart devices, e.g., smart fridges, smart TVs, etc., are ubiquitous in our daily lives and have computational capabilities that are comparable with cloud servers [7]. In addition, smart devices are at the edge of the network. Thus, other devices can easily access services hosted in smart devices without a computational overhead. Smart devices are also underutilized during the day, and thus their resources can be harnessed without disrupting the owner, e.g., idle time when the device is in the pocket, at the office, at the bus, or during the night. These opportunistic times can be exploited to migrate virtual devices from the cloud into smart devices, such that the physical counterpart of a virtual device can save energy, and help others to obtain service provisioning in a low latency environment. Thus, in this paper, we propose a cloud-based framework namely *SensorClone*, which can be used to create virtual devices that can be migrated opportunistically into smart devices.

Specifically, we investigate how the data collected from any IoT device with sensing capabilities can be processed in the cloud to build a virtual device that matches the behavior of the physical one. By using our approach, we identify two key opportunities. First, the virtual device can be migrated into smart devices from the cloud, e.g., end smartphones, to provision the same service that other devices access in the cloud. Second, a physical device can reduce its own power consumption as it can duty cycle more efficiently its service provisioning between its real and virtual representation. In other words, the physical device can turn into idle mode while the virtual is operating, and then, the physical device just gets into operational mode to re-calibrate the accuracy of the service provisioning of the virtual device. SensorClone enables the virtual device to re-calibrate based on software-defined networking (SDN) policies, which allows defining calibration based on different levels of Quality of Service (QoS).

Additionally, since the service provisioning of a virtual device can be subject to errors if the virtual device is not re-calibrated periodically with its physical counterpart, SensorClone implements a mechanism to monitor QoS degradation, such that the virtual device can include its current level of service quality when providing its service to other devices. In this manner, other devices can be aware about using or not the virtual service based on its own needs. To evaluate our framework, we present a use case of a virtual device created from an IoT service that provisions temperature information. Thus, here thereafter we refer a virtual device as a virtual sensor. Our results indicate that harnessing smart devices opportunistically is a feasible approach for improving IoT scalability and resilience.

Summary of Contributions

The contributions of the paper are summarized as follows:

- We develop, design and deploy SensorClone in a realistic setup. We demonstrate the feasibility of migrating virtual sensors from the cloud into smart devices to opportunistically harness their resources in underutilized periods.
- We quantify the theoretical and practical amounts of energy that can be saved when duty cycling sensor service provisioning by swapping between virtual and real sensors. Our results show that power efficiency can be increased up to 90%.
- Unlike other work, we present the first adaptive system that considers the trade off between power efficiency and service accuracy when using virtual devices. While in theory virtual sensors can induce major gains in energy, we find that those gains are not matched in practice as they are highly dependent on hardware specifications and operations modes of devices, e.g., Arduino.

[5]https://www.ibm.com/blogs/internet-of-things/iot-digital-twin-enablers/

The rest of the paper is organized as follows. In Section 2, we present the related work. We then highlight in Section 3, the challenges and technical problems of virtualizing the sensing behavior of a device in the cloud. We present our SensorClone framework in Section 4. We evaluate the benefits that can be obtained by leveraging SensorClone by providing a use case based on temperature sensing using a real testbed in Section 5. In the light of the results, we present a discussion in Section 6. Lastly, we conclude the paper in Section 7.

2 RELATED WORK

Previous work on improving the continuous provisioning of IoT services focuses on fixing communication deficiencies in the protocols and adapting the size of the communication channel by tuning the quality of the service of IoT applications [1, 7, 12, 18, 28]. Middleware solutions have been also proposed for the remote management of sensor data [14, 29]. Other solutions to overcome this challenge fake the presence of the service when a device is unavailable by using its last known state of service provisioning [27]. One such example is the "device shadows" feature of Amazon's AWS IoT. While this approach introduces fault tolerance in the system, it lacks QoS policies, which are critical for applications such as healthcare [26], transport systems [5], and urban mobility [4, 30]. While these approaches provide partial solutions to the problem, they remain agnostic regarding energy consumption and device malfunction. In terms of cooperation between devices, collaborative solutions for offloading sensing to other devices also have been investigated [8, 13, 20, 25]

The modeling and prediction of sensor data to improve the resilience of systems has also been explored [11, 21], but without opportunistic migration and quantification of power efficiency for IoT devices, which is the focus of our work. The virtualization of sensors has been explored from a scalability point of view without introducing energy efficiency policies [21]. Moreover, IBM has proposed the term of *Digital twin*, which consists of creating a virtual representation of the device in the cloud by collecting and analyzing data from the physical device. The digital twin is then used for troubleshooting the device based on different scenarios. In our work, we argue that virtual devices can be used to improve IoT resilience. We envision a virtual device that can be use to extend the battery life of a physical device counterpart by improving its duty cycling. In addition, we envision that a virtual device can exploit opportunistic times of other physical end devices, e.g., smartphones, such that a virtual device can be migrated from the cloud to the end smart devices temporally for providing an alternative service closer to users (low latency).

3 THE CASE OF VIRTUAL SENSORS: CHALLENGES AND TECHNICAL ISSUES

Certainly, a virtual sensor can be created by sampling data about its operations [21]. However, many challenges arise when constructing a virtual sensor in the cloud and then migrating its behavior into other devices. To achieve this, we must be able to: 1) model its behavior with minimal effort, 2) detect opportunities to migrate it without disturbing users, and 3) control the migration into the devices. We next present these relevant challenges.

3.1 Virtualizing and Difussing Behavior

Data Modeling —Instead of relying on the device itself to construct the sensor model for the virtual sensor, we envision the cloud as a platform that can construct the sensor models from opportunistic collected data [21]. This will reduce the effort of constructing and migrating sensors as it is centralized in the cloud. Thus, the key challenge is to model the data transparently in the cloud for creating a virtual sensor, which can learn to emulate the behavior from a physical counterpart sensor by producing similar outcomes [17]. Certainly, learning from the data is challenging as very diverse types of information can be produced by a device. However, the principal goal of the learning process is to identify patterns in the data that can be used to reduce the sensing process of the real device [22]. In other words, if the device readings can be predicted with high confidence, the device does not actually have to make those readings, since they can be considered redundant.

For example, Figure 2 depicts 16 exemplar data patterns commonly found in micro-mechanical artifacts and software-based sensors that are embedded in a device. While the sensor frequency is important, these examples show that exploitable patterns can be found regardless. Our vision is that by collecting enough data from a given device, it is possible to build a model that emulates its behavior *intermittently*. Naturally, since the behavior of each sensor depends on contextual factors [15], a model needs to be tailored for each specific sensor in a particular context.

Opportunistic migration — The detection of opportunities to migrate virtual sensors from the cloud depends on the type of device. Base stations, hot spots and other telecommunication infrastructures are the main target to migrate functionality for longer periods as the sensor execution can be piggybacked as part of other's application execution. Smart devices in fixed locations, e.g., smart television, smart refrigerators, etc., can be used for migration of virtual sensors when devices change to an idle mode. Personal devices, e.g., mobile devices, smart watches, personal computers, etc., can be used for migration in periods where the device is not being used by the mobile user, charging times or even when the battery is completely charged. In addition, opportunities to migrate functionality also depend on the stability of devices to provision services in a specific location, which correlates with users' mobility patterns [10]. For instance, a mobile device that is located in an office for two hours is better candidate than a mobile device detected for a few minutes in a bus ride. In [8], we identify different types of stability for devices detected in the wild (Figure 3). Stability is modeled as the duration and frequency in which a device is encountered. We classify the levels of stability into low, medium and high (Figure 3a). The level of stability is a parameter that needs to be considered when migrating a virtual sensor temporally into smart devices. We identify that devices with medium and high stability levels are strong candidates to migrate virtual sensor functionality in the wild (Figure 3b models the stability of infrastructure encountered by 1 user in a one month experiment).

3.2 Controlling Operations

Device management — While the exploitation of data patterns in the sensor data of the physical device allows the cloud to build a virtual sensor, the understanding of the physical device environment

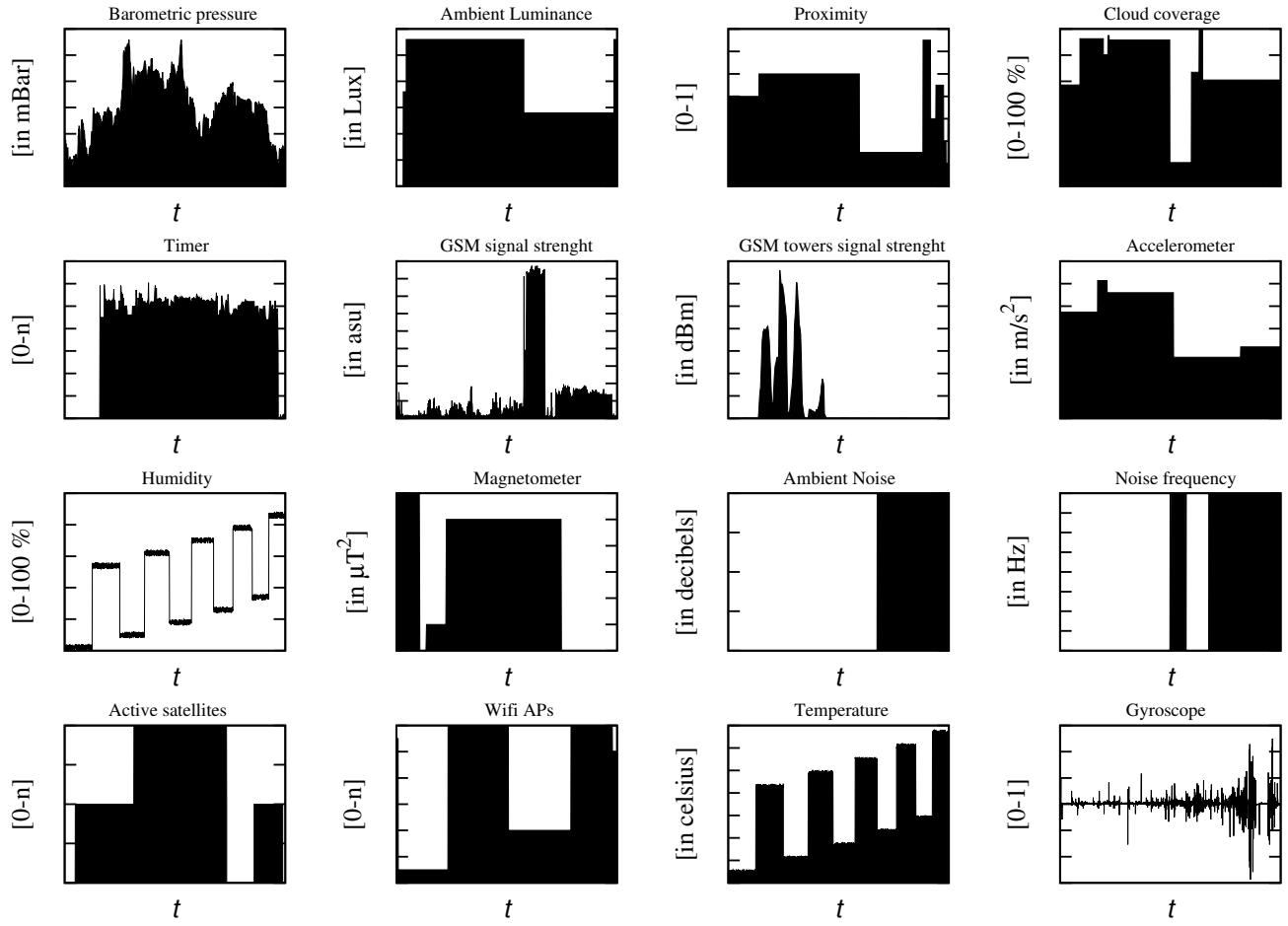

Figure 2: Samples of data patterns that can are produced by different sensors.

can also benefit the process of its virtualization. Table 1 summarizes how the deployment settings of a device can be used to influence its operational behavior from the cloud. Specifically, we classify deployment settings into three broad categories: standalone (e.g. automated sprinklers), scaled (air pollution in a city), and heterogeneous (temperature and humidity in a building). The differences between these categories are related to the number of data sources, the homogeneity of the data, the control policy, and data modeling we expect to use.

In a standalone deployment, a single sensor is used to collected data. In this case, a prediction model can be created by learning from this homogeneous dataset. In turn, the control policy of the device is based on operational scheduling. This means that the device is scheduled to change between operation modes based on whether the virtual sensor is able to predict the values of the real sensor with acceptable QoS. For instance, let us consider a sensor that controls an automatic sprinkler in a plantation of mushrooms that need to be kept under a certain humidity. The device senses the level of humidity in the ground at fixed intervals. To reduce the amount of readings and save energy, a virtual sensor predicts the sensor readings while the real device is in sleep mode. The real device is scheduled to be active a few times just to validate the predicted values provided by the virtual sensor. This validation is done in the cloud and consists in synchronizing the behavior of virtual sensor with its real counterpart.

In a scaled deployment, data from multiple identical sensors is collected. In this case, a predictive model is created by learning from that homogeneous data per each sensor. The model can attempt to exploit correlations in the values of multiple devices, and therefore the cloud can use a single model to predict the data of multiple devices. Moreover, to validate the prediction, we can coordinate the real sensors to obtain actual readings from a minimal subset of devices, such that the rest of the devices remain asleep. For instance, let us consider hundreds of sensors located across a city to measure air pollution/air quality. We can expect that sensors that are close to each other have similar measurements. Thus, a virtual sensor in the cloud can produce data to emulate the presence of multiple devices. Furthermore, we can coordinate the real devices to become active in a round-robin fashion in order to validate the model predictions given by the virtual sensor. This means that the gains in energy for each device will multiply, since the burden of sensing is now shared and balanced between multiple devices.

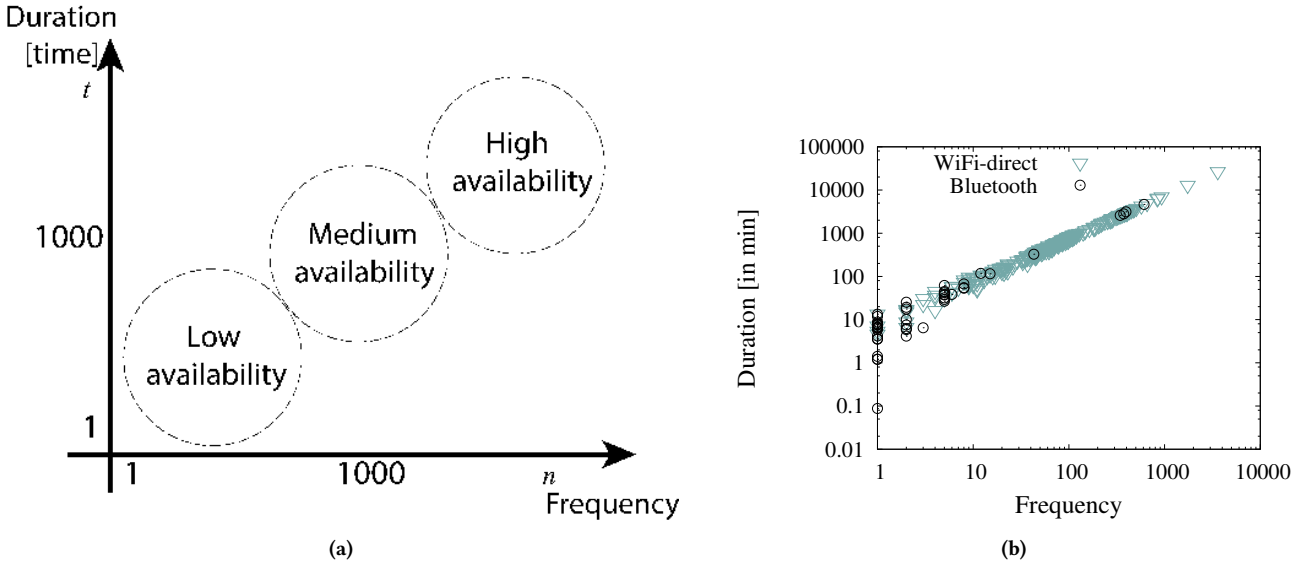

Figure 3: Opportunistic migration of virtual sensors in proximal infrastructure [8]. (a) Levels of stability of encountered infrastructure, (b) Quantification of encountered infrastructure.

Lastly, in a heterogeneous deployment, data from multiple types of devices is collected. A virtual sensor can be created by learning from the patterns of each sensor. Additionally, it is possible to identify causal relationships involving multiple types of sensors. For instance, let us consider multiple temperature and humidity sensors located in a building. The readings of a temperature sensor can be used to predict the values of nearby humidity sensors and temperature sensors. The rationale of this is that while the temperature describes how much heat is in the air, the humidity describes how much water vapor is in the air. Thus, when air temperature changes, humidity relative to that temperature also changes. As a result, both types of sensors can be coordinated and scheduled to improve energy efficiency. A virtual sensor can capture this relation and predict dual values for it.

Runtime environment — The migration of a virtual sensor into a smart devices, e.g., smartphone, smart home appliances, that does not perform that kind of sensing but it has service provisioning capabilities, it requires that the target device is equipped with the same runtime environment for executing the sensor model. For instance, a virtual sensor built in R[6] is easily executed in the cloud, but for executing the same model in a smartphone, it requires that the device is equipped with R for Android[7]. Otherwise, the virtual sensor cannot be executed.

Since the virtual sensor also can replace the service provisioning of the real sensor, approaches for managing the device from a remote location need to be in place. Approaches for remote managing, include, computation offloading [7], push notifications, REST-based requests, and agents [23], among others.

[6]https://www.r-project.org/
[7]http://www.r-ohjelmointi.org/?p=1434

4 OPPORTUNISTIC MIGRATION OF VIRTUAL SENSORS

By capturing the behavior of a physical device into a virtual sensor, we envision a system that can migrate the presence of a sensor into other devices opportunistically [24]. *An opportunistic migration occurs when there is an opportunity to rely on resources to perform a task without inducing any counterproductive effect on those resources*[8]. However, in the case that there are not opportunistic smart devices for migration, then SensorClone can be used instead to duty cycle the operations of the physical device between the physical device itself and the virtual sensor. By doing this, the physical device avoids using remote network communication to propagate its presence over Internet, and as a result, the physical device improves its energy consumption. The overall system of SensorClone is shown in Figure 4 and consists of the following components.

Analyzer — It builds the virtual sensor that learns from the collected data of the physical device. As proof of concept, we consider a simple model (sliding window prediction [9]) for the virtual sensor to demonstrate the potential of the approach. Naturally, more sophisticated models improve the accuracy of the sensor prediction [17]. Other techniques such as compressive sensing [3] or deep learning [19] can be used instead. Our model is built based on historical observations of the past behavior of a sensor. This allows the model to infer the future values of a sensor in order to emulate behavior. Let us define the model. Assuming that the values v of the sensors are notated by a binary class attribute $c = \{0, 1\}$. Each recorded pair of values of the train dataset is then formed the following tuple $t =< v, c >$. In the training phase, the model is built by incorporating all the past tuples observed by the sensor. For each consecutive sequence of c binary class attribute the v values are

[8]If there is an effect, then the effect is considered as marginal or it is compensated to achieve a balance in the system

Table 1: Categorization for data exploitation and modeling in IoT environments.

Operational environment	Features				Use cases
Deployment settings	**Data source**	**Data type**	**Control policy**	**Data modeling**	**Application examples**
Standalone	Single (1)	Homogeneous	Scheduling	Learning	Automatic sprinkler
Scaled	Many (2..*)	Homogeneous	Coordination and Scheduling	Learning	Air pollution in a city
Heterogeneous	Many (2..*)	Heterogeneous	Coordination and Scheduling	Discovery and Learning	Temperature and humidity in a building

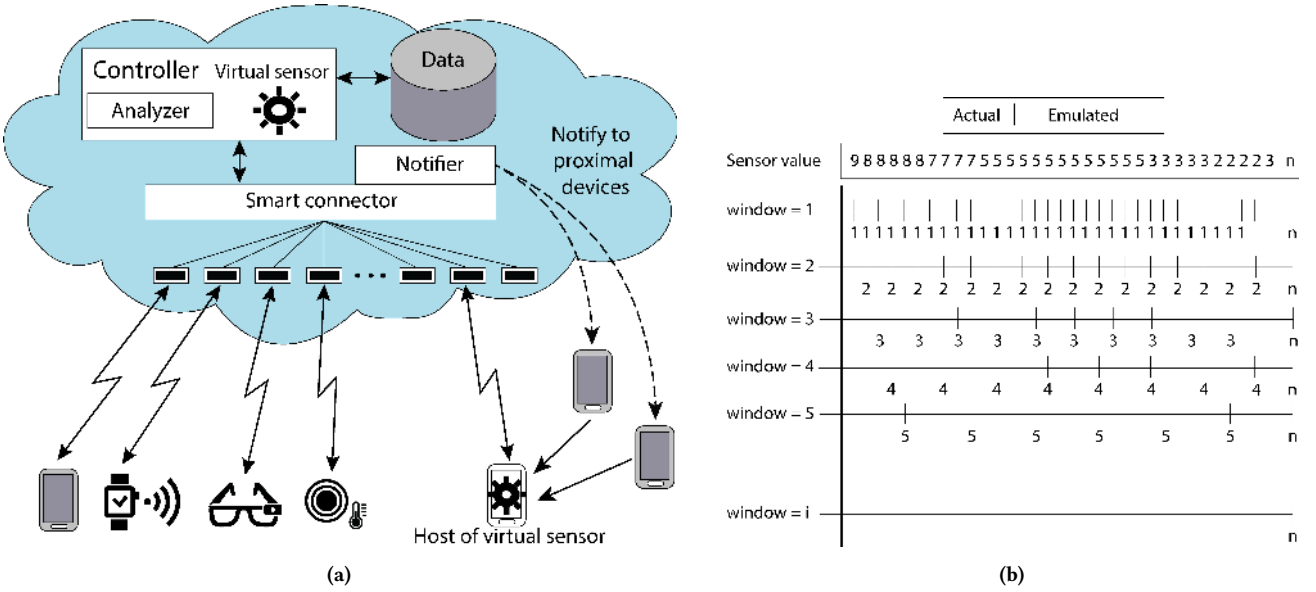

Figure 4: (a) Overview of SensorClone, (b) Sliding window model of the virtual sensor.

aggregated in order to form a final tuple of $T =< \sum_i v, c >$, where i is the number of the observed values for a certain c. Specifically, i is called a window by means of v value continuity within a certain c. The model is composed by a number of subsequent tuples T.

In the test phase, the model is fed with a new test dataset of certain tuples $t' = < v', c' >$. The model is initialized with the first tuple t' and searches the nearest match, i.e., nearest distance, between the actual value v' and the value of $\sum_i v$ of the tuple T for a certain c. When the match is achieved, the model predicts that the sensor will have the predicted value $\sum_i v$ for the next window size i records. During these records the sensor becomes idle thus being power efficient. When the window size is reached the sensor becomes active and it reads the next actual value v'. The model uses a sliding window to change the length of records to predict as shown in Figure 4b. The process is repeated until the end of the test dataset.

The metrics used to evaluate the model during the test phase are the prediction accuracy p and the power consumption e. The number of correctly classified instances defines prediction accuracy and it is computed by the test dataset during the test phase by assessing the nearest distance between the actual value v' and the predicted value $\sum_i v$ for certain c out of the number of all instances in the test dataset. Specifically, prediction accuracy p is framed by a threshold θ and a relaxation parameter $r \in [0, 1]$ i.e., $p \leq \theta + r \cdot \theta$. The threshold θ is computed by the train dataset during the train phase as the average distance between the actual value v and the predicted value $\sum_i v$. Prediction accuracy p is then standardized in order to fall in the range $p \in [0, 100]$.

Power efficiency e parameter is computed as the difference from unity of the number of times that the certain sensor becomes active during the test phase out of the number of all instances in the test phase. Subsequently, power efficiency e is standardized in order to fall in the range $e \in [0, 100]$. Lastly, the prediction accuracy p of the model can be configured dynamically during runtime in order to control the power efficiency of the device. Naturally, the higher the prediction accuracy p, the lower is the power efficiency e thus the amount of energy that can be saved from the device.

SDN controller — It allows a centralized authority to define the policies for emulating a device in the cloud. A policy defines the conditions in which the behavior of a device can be emulated. For instance, a policy to preserve power can define the minimum level of accuracy that a virtual sensor must guarantee. By adjusting the length of the prediction window, the system can reduce the power consumption of the device: while the virtual sensor is predicting behavior, the physical sensor can remain asleep and can avoid sensing data transmission. Other policies can cover network availability,

transmission costs, and can also consider sensor data. For instance, sensors in a smart home can be emulated using a virtual sensor if no inhabitants are detected inside the house. Our motivation to use SDN is to control the threshold between power efficiency and accuracy of virtual sensors dynamically. In addition, SDN policies are used for changing migration candidates (smart devices) on the fly.

Smart connector — It provides an interface to connect a device to the cloud, such that the device can transfer its sensing data using protocols such as XMPP, HTTP, and CoAP. The connector propagates the presence of the device to other connected devices to achieve inter-operation among devices. The main task of the connector is deciding when a virtual sensor is migrated to other devices. There are many considerations for deciding *when to migrate*. For instance, the availability of the infrastructure, e.g. base stations, laptop, etc, the compatibility of the target with the virtual sensor, the stability of the device in a location, the leasing of a device from a user, and so on. Since we focus on the issues of development and deployment of the system, we simply consider that a candidate device is chosen based on its stability in a location as presented in [6].

Once a device is chosen for migration, the connector sends a notification to the device, such that the device can retrieve the virtual sensor and execute it. The connector also informs other devices in the same location about the alternative proximal service via push notification. Lastly, if devices for migration are not found, then the connector uses the virtual sensor to manage the duty cycle of the service provisioning of physical devices whose behavior fits the virtual sensor.

5 CASE STUDY: EVALUATION AND RESULTS

To demonstrate the feasibility of our SensorClone, we present an experimental case study. We consider an IoT service that provisions temperature information that is hosted in the cloud. The service relies on readings of an actual temperature sensor artifact deployed in the field. We emphasize that the main goals of the case study are to show that it is possible 1) to create a virtual sensor and 2) to control the operational behavior of the physical device from the cloud. When fulfilling these two requirements, the migration of the virtual sensor becomes possible.

5.1 SensorClone setup

Our complete experimental setup is depicted in Figure 5. We use an Arduino micro-controller[9] as a sensing device. We configure the device to transmit temperature data to SensorClone in JSON format. We rely on *Arduino HTTPClient* and *WiFlyHQ* libraries[10] for implementing the HTTP functionality. The device is controlled through the *watchdog timer* provided by the JeeLib library[11], which controls the change between operations modes, such that once the virtual sensor is active, the physical sensor can change to idle. As notification mechanisms, we rely on GCM[12] (Google Cloud Messaging) and XMPP frameworks. GCM is a proprietary mechanism

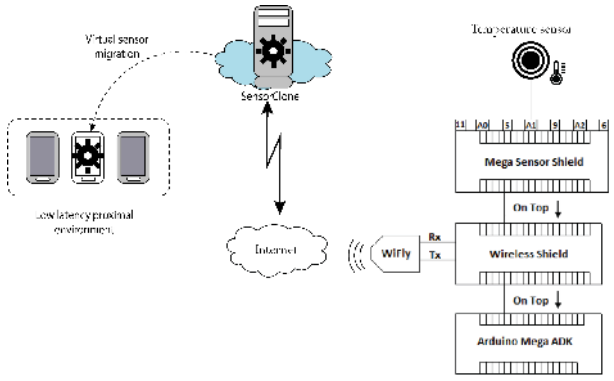

Figure 5: Experimental setup composition.

for Android, and XMPP[13] is an open mechanism that can used in any device independently of the provider.

SensorClone is developed using Java. It controls the Arduino through REST requests. We develop our own SDN component as the policies to control the trade-off (accuracy vs power efficiency) and sensor migration are not based on network traffic but quality of service provisioning and stability of devices. The *SDN controller* implements a power consumption policy in its *Analyzer* component, which relies on a stochastic model to analyze the sensor data (explained in detail in section 4). The virtual sensor is built based on historical observations of the past behavior of the sensor. Therefore, prior to predict data, the virtual sensor is fed up with a temperature dataset collected from the physical device. In the prediction process, when an input is passed to the Analyzer, this one uses the virtual sensor to find the closest match in the dataset by calculating the distance of the input against each record in the dataset, i.e., nearest distance. Based on the length of the window defined by the authority of the device, the model predicts n sequential values after the input. During the sensor prediction, the physical device is changed to idle mode by SensorClone. Thus, the device improves power efficiency. The virtual sensor wakes up the device at the end of each window in order to get a real value that can be used to synchronize the virtual sensor with its physical counterpart.

5.2 Testbed: setup and methodology

The aim of the experiment is to determine the benefits and trade-offs when creating a virtual sensor in the cloud. The key insight of the experiment is that the cloud can toggle the physical device (micro-controller) between active and inactive modes to reduce power consumption. In active mode, the micro-controller transmits sensor data to the cloud to propagate its presence. When another device requests the temperature service, the cloud service responds to the request with the actual values sensed by the micro-controller. Alternatively, in inactive mode the micro-controller is in an idle state and does not transmit data. Thus, when another device requests the temperature service, the cloud responds the request with the values predicted by the virtual sensor.

[9] http://www.arduino.cc
[10] http://www.arduino.cc/en/Tutorial/HttpClient
[11] http://jeelabs.net/pub/docs/jeelib/
[12] https://developers.google.com/cloud-messaging/
[13] https://xmpp.org/uses/instant-messaging.html

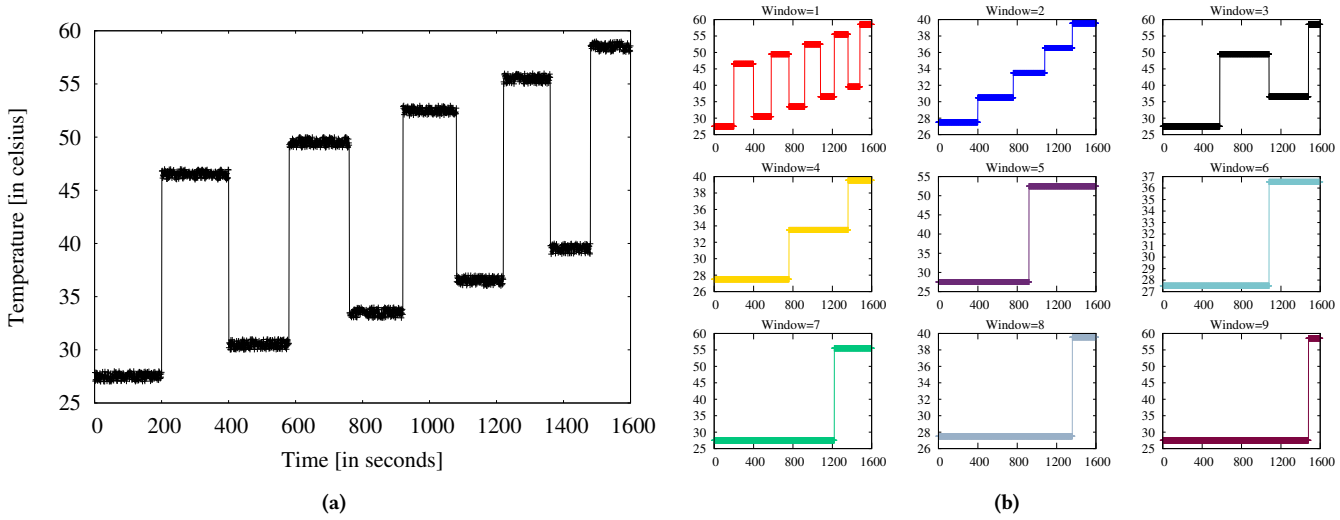

Figure 6: (a) Real data collected from a temperature sensor, (b) Virtual temperature data produced by SensorClone.

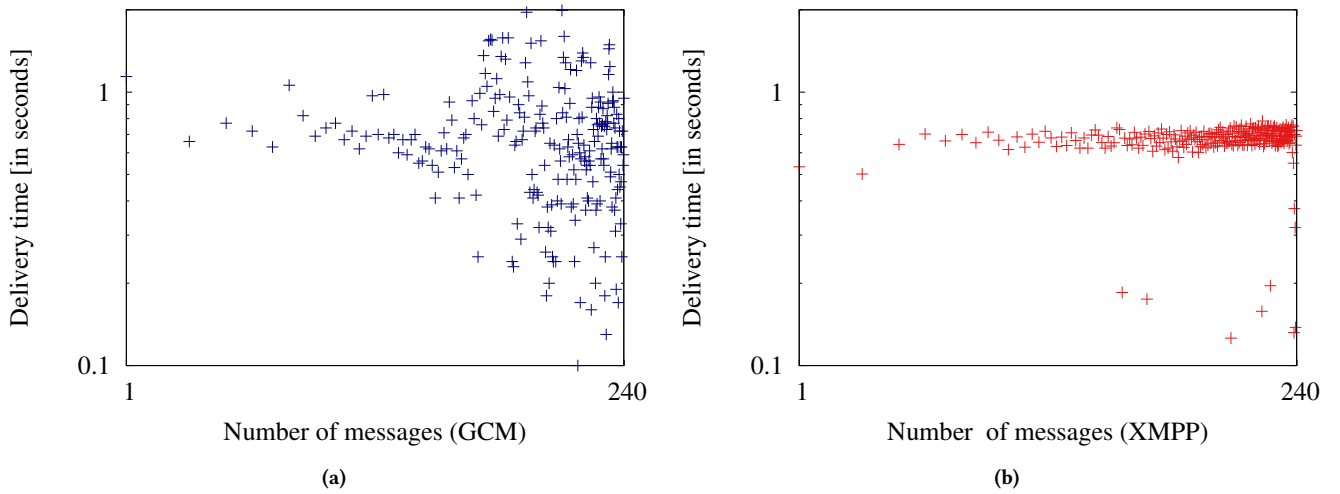

Figure 7: (a) Performance of GCM, (b) Performance of XMPP

In our setup, we expose the temperature sensor to high and low temperature induced by a nearby heater. Since our intention is to demonstrate how sensor data can be modeled to predict behavior from any system, we simulated a cooling system. From this setup, we took ≈1600 readings with an inter-arrival rate of ≈1 min.

We also measure the performance of two notification mechanism. Notifications are important when informing devices about available end devices that host a virtual sensor. The aim of the experiments is to determine the latency between a provider submitting a request and the target device receiving the notification (responsiveness). Messages are fixed to a size of 254 bytes, which is the lowest common denominator of the allowed message sizes of the considered approaches. Messages are formatted with similar characteristics so that they can ensure a fair comparison that is not affected by transportation factors such as data size, among others.

Messages are sent every second for 15 seconds in sequence, which is followed by a 30 minute sleep time, and then another set of 15 messages, repeating the procedure for 8 hours (240 messages in total). The frequency of the messages is set in this way in order to mitigate the possibility of being detected as a potential attacker to the cloud vendor, e.g. Denial of Service, and to refresh the notification service from a single requester and possible undelivered data. Moreover, the duration of the experiments guarantee having an overview of the service under different mobile loads, which may arise during different hours of the day.

SensorClone is deployed on Amazon EC2 in the Ireland region using a t2.large server. To measure the energy consumed by the micro-controller we rely on the Mobile Device Power Monitor[14].

[14]https://www.msoon.com/LabEquipment/PowerMonitor/

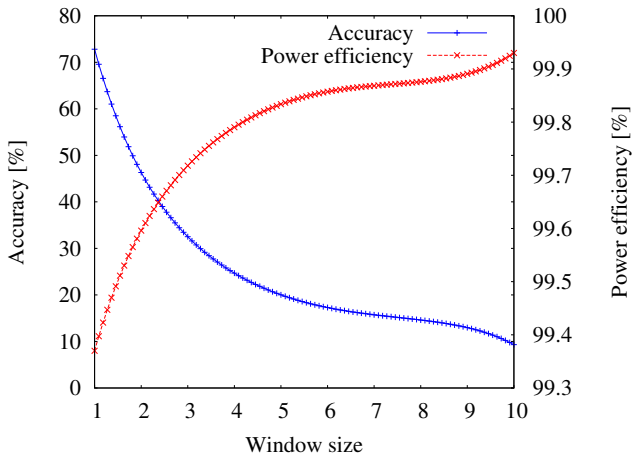

Figure 8: Conceptual improvements in power efficiency that can be achieved through SensorClone depending on accuracy level.

5.3 Results

Initially, we validate the poor scalability of the micro-controller and identify the demand to propagate its presence via the cloud. Therefore, we stressed the device with concurrent requests. We found that in average the device is able to handle ≈4 users simultaneously.

Next, we proceed to evaluate SensorClone. Figure 6a presents the actual data collected by the sensor. By learning from this historical data, we create a virtual sensor from an IoT service that provisions temperature information. Figure 6b shows the values predicted by the virtual sensor. From Figure 6b, we observe that different windows length can be used to schedule the operation modes when the micro-controller needs to transmit data and not. By default, the device is always in idle mode and the cloud wakes up the device to receive a real value. This value is used to synchronize with the actual data of the micro-controller and predict the sequence of the next values. We can observe (as expected) that the accuracy of the predicted values increase as the window length decreases. In contrast, as the window length gets longer, the device reduces its power consumption (by avoiding transmission) but the accuracy level of the emulated service drops. This trade-off is visualized in Figure 8.

Notification performance results are presented in Figure 7a and Figure 7b. Summary statistics are also shown in Table 2. According to the results, GCM provides poor delivery rates for notifications, with an average of ≈0.75 sec, median of ≈0.66 and standard deviation (SD) of ≈0.69. From the GCM delivery rate diagram, it can be observed that the QoS starts to decrease as the number of messages increase across time. Consequently, messages tend to arrive without a specific order. Some of the reasons that can cause that behavior include: the utilization of multiple servers, where each server handles its own individual queue; the unequal distribution of messages among the active servers sending notification; and high utilization of the notification system. Android is one of the most popular platforms for developers. Thus, the notification service is expected to handle the heavy load of messages. In contrast, XMPP

Mechanism	**Average delivery** (mean) [s]	**Median delivery** (median) [s]	Delivery variability (SD)
GCM	0.75	0.66	0.69
XMPP	0.6	0.75	0.10

Table 2: Summary statistics of message delivery time for both mechanism.

mechanism shows to provide better reliability for delivering messages, with an average delivery time of ≈0.6 sec, median of ≈0.75 and SD of ≈0.10. However, we need to mention that GCM handles a worldwide load of devices sending notifications, while XMPP is a private ad-hoc deployment.

However, the practical power efficiency is slightly different from the conceptual efficiency due to hardware considerations. While we assume that the idle mode requires less energy, this is not always the case. For instance, the idle mode of Arduino consumes almost the same energy as the active mode. In our experiments, the measured energy that the device consumes when transmitting data to the cloud (≈2.12mA) is comparable to that consumed in idle mode (≈1.93mA). Thus, to determine the actual gains in power consumption, we compare the actual energy consumption in our use case presented with the energy consumption when using a virtual sensor with a window of length 1 (as shown in Figure 6b). We use a window of length 1 as it provides similar levels of quality of service when compared with its real counterpart. From our use case, we measure about $3450mA$ for the normal service provisioning of the device, and $3098mA$ for the service provisioning using a virtual sensor. This is translated into an improvement in energy saving of ≈10% for the micro-controller.

6 DISCUSSION

Based on the results of our experiments, we present in this section a discussion about the benefits and drawbacks of virtual sensors.

Virtual sensor for power efficiency: While the conceptual results indicate that it is possible to increase the power efficiency of the device up to 90%, in practice, the gains of energy depend on the hardware's ability to enter to a low-power idle mode. In our case study, we need to consider that Arduino is an economic hardware that does not really support a low-power idle mode. Consequently, while there are significant gains in energy, those are less when compared with the conceptual model. However, more sophisticated devices that optimize their operation modes can benefit from our approach to save significant amounts of energy.

Virtual sensor modeling: Intuitively, based on our deployment categorization, the sensor data collected in standalone deployments requires more time and data to bootstrap the system, because a single device needs to provide all the possible cases of behavior. In contrast, the sensor data collected in scaled deployments can bootstrap the system faster as multiple devices act as data sources. Finally, the sensor data collected in heterogeneous deployments requires a priori analysis before can be used to create virtual sensors. We cannot immediately bootstrap emulation of a device until we are able to identify linked relations. Once that is achieved, we can

expect substantial gains since we can substantially increase the window length of each individual device.

While our case study successfully demonstrated the feasibility of creating a virtual sensor from a physical device and control operation modes of the physical device, these findings open a wide spectrum of questions towards the optimal coordination and scheduling of the sensors. We consider addressing these issues in our future work as more detail analysis is required.

Virtual sensor awareness: Our work shows that it is possible to create a virtual sensor which emulates the behavior of a real sensor. This virtual sensor can be migrated into other devices, e.g., base stations, to provision sensing services closer to users. The benefits of the approach are avoiding communication to remote resources that can harm the energy and performance of IoT devices. However, certain user's considerations must be taken also into account when relying on virtual sensors, for instance, should a user be notified when its mobile device is relying on virtual sensor data rather than the real one? at what extent is acceptable for a user to rely on virtual sensor data? We leave these considerations for future work, where we plan to conduct a user study that quantifies the perception of users towards different levels of service accuracy.

Virtual sensor for IoT: IoT resilience is difficult to achieve as it is influenced by many factors. While several approaches have been proposed in industry (e.g., device shadows of AWS IoT, IBM Gryphon, and Digital Twin among others) and academy sectors (e.g., sensor prediction approaches and proxy sensors), it is still unknown whether it is possible to achieve continuous service provisioning for IoT services in the wild. The utilization of virtual sensors is a promising solution that compliments existing systems and architectures. Naturally, the benefits of virtual sensors will increase on time as the models to create virtual sensors that capture behavior of physical devices get more accurate. In this context, we emphasize that our main contribution to advance the art is that SensorClone takes a step further when compared with other solutions as it leads to an intelligent and adaptive virtualization of a physical device behavior that can be migrated, such that it can be embedded in other environments via host devices.

7 CONCLUSIONS

In this article, we propose SensorClone, a cloud-based framework that extends the energy of IoT devices by harnessing underutilized smart devices, e.g., smartphones, smart home appliances. SensorClone builds in the cloud a virtual sensor of a physical device by collecting its behavioral data. SensorClone optimizes the energy of a physical device by duty cycling its service provisioning between its real and virtual representation. In addition, SensorClone exploits opportunistic and underutilized times of other devices at the edge of the network to migrate a virtual sensor which then can be used by others to obtain service provisioning in a low latency network. We build a prototype of SensorClone and present a case study based on a standalone IoT service that provisions temperature information to demonstrate the feasibility and potential of our approach. Lastly, we provide the source code of our case study as open source in GitHub[15].

[15]https://github.com/huberflores/Energy-AwareOffloading-IoT

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their insightful comments.

REFERENCES

[1] Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Shobha Venkataraman, and He Yan. 2014. Prometheus: toward quality-of-experience estimation for mobile apps from passive network measurements. In *Proceedings of the 15th ACM Workshop on Mobile Computing Systems and Applications (HotMobile 2014)*. Santa Barbara, California, US.

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The internet of things: A survey. *Computer networks* 54, 15 (2010), 2787–2805.

[3] Richard G Baraniuk. 2007. Compressive sensing [lecture notes]. *IEEE signal processing magazine* 24, 4 (2007), 118–121.

[4] Yin Chen, Takuro Yonezawa, Kazunori Takashio, Yutaro Kyono, Jin Nakazawa, and Hideyuki Tokuda. 2015. A public vehicle-based urban sensing system. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2015): Adjunct*. Osaka, Japan.

[5] Zipei Fan, Xuan Song, Ryosuke Shibasaki, Tao Li, and Hodaka Kaneda. 2016. CityCoupling: bridging intercity human mobility. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2016)*. Heidelberg, Germany.

[6] Huber Flores, Denzil Ferreira, Chu Luo, Vassilis Kostakos, Pan Hui, Rajesh Sharma, Sasu Tarkoma, and Yong Li. 2016. Social-aware device-to-device communication: a contribution for edge and fog computing?. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2016): Adjunct*. Heidelberg, Germany.

[7] Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Satish Srirama, and Rajkumar Buyya. 2015. Mobile Code Offloading: From Concept to Practice and Beyond. *IEEE Communications Magazine* 4 (2015).

[8] Huber Flores, Rajesh Sharma, Denzil Ferreira, Vassilis Kostakos, Jukka Manner, Sasu Tarkoma, Pan Hui, and Yong Li. 2017. Social-aware hybrid mobile offloading. *Pervasive and Mobile Computing* 36 (2017), 25–43.

[9] Ray J Frank, Neil Davey, and Stephen P Hunt. 2001. Time series prediction and neural networks. *Journal of intelligent and robotic systems* 31, 1-3 (2001), 91–103.

[10] Raghu Ganti, Mudhakar Srivatsa, Anand Ranganathan, and Jiawei Han. 2013. Inferring human mobility patterns from taxicab location traces. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2013)*. Zurich, Switzerland.

[11] Marisol García-Valls, Javier Ampuero-Calleja, and Luis Lino Ferreira. 2017. Integration of Data Distribution Service and Raspberry Pi. In *Proceedings of the International Conference on Green, Pervasive, and Cloud Computing (GPC 2017)*. Cetara, Amalfi Coast, Italy.

[12] Bo Han, Pan Hui, VS Anil Kumar, Madhav V Marathe, Jianhua Shao, and Aravind Srinivasan. 2012. Mobile data offloading through opportunistic communications and social participation. *IEEE Transactions on Mobile Computing* 11, 5 (2012), 821–834.

[13] Samuli Hemminki, Kai Zhao, Aaron Yi Ding, Martti Rannanjärvi, Sasu Tarkoma, and Petteri Nurmi. 2013. Cosense: A collaborative sensing platform for mobile devices. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys 2013)*. Rome, Italy.

[14] Christopher K Hess, Manuel Román, and Roy H Campbell. 2002. Building applications for ubiquitous computing environments. In *International Conference on Pervasive Computing (Pervasive 2002)*. Zurich, Switzerland.

[15] Geoff Hulten, Laurie Spencer, and Pedro Domingos. [n. d.]. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2000)*. San Francisco, Ca, USA.

[16] Michael O Jewell, Enrico Costanza, and Jacob Kittley-Davies. 2015. Connecting the things to the internet: an evaluation of four configuration strategies for wi-fi devices with minimal user interfaces. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2015)*. Osaka, Japan.

[17] Hongbo Jiang, Shudong Jin, and Chonggang Wang. 2011. Prediction or not? An energy-efficient framework for clustering-based data collection in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems* 22, 6 (2011), 1064–1071.

[18] Karthik Kumar and Yung-Hsiang Lu. 2010. Cloud computing for mobile users: Can offloading computation save energy? *Computer* 43, 4 (2010), 51–56.

[19] Nicholas D Lane and Petko Georgiev. 2015. Can deep learning revolutionize mobile sensing?. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile 2015)*. Santa Fe, New Mexico.

[20] Youngki Lee, Younghyun Ju, Chulhong Min, Seungwoo Kang, Inseok Hwang, and Junehwa Song. 2012. Comon: Cooperative ambience monitoring platform with continuity and benefit awareness. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys 2012)*. Low

Wood Bay, Lake District, United Kingdom.
[21] Sanjay Madria, Vimal Kumar, and Rashmi Dalvi. 2014. Sensor cloud: A cloud of virtual sensors. *IEEE software* 31, 2 (2014), 70–77.
[22] Adam J Oliner, Anand P Iyer, Ion Stoica, Eemil Lagerspetz, and Sasu Tarkoma. 2013. Carat: Collaborative energy diagnosis for mobile devices. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys 2013)*. Rome, Italy.
[23] Shumao Ou, Kun Yang, Antonio Liotta, and Liang Hu. 2007. Performance analysis of offloading systems in mobile wireless environments. In *Proceedings of the IEEE International Conference on Communications (ICC 2007)*. Glasgow, Scotland, UK.
[24] Luciana Pelusi, Andrea Passarella, and Marco Conti. 2006. Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *IEEE Communications Magazine* 44, 11 (2006).
[25] Kiran K Rachuri, Christos Efstratiou, Ilias Leontiadis, Cecilia Mascolo, and Peter J Rentfrow. 2014. Smartphone sensing offloading for efficiently supporting social sensing applications. *Pervasive and Mobile Computing* 10 (2014), 3–21.
[26] Suman Sankar Bhunia. 2015. Adopting internet of things for provisioning healthcare. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2015): Adjunct*. Osaka, Japan.
[27] Mahadev Satyanarayanan. 2001. Pervasive computing: Vision and challenges. *IEEE Personal Communications* 8, 4 (2001), 10–17.
[28] Mojca Volk, Janez Sterle, Urban Sedlar, and Andrej Kos. 2010. An approach to modeling and control of QoE in next generation networks. *IEEE Communications Magazine* 48, 8 (2010), 126–135.
[29] Royu Want, Trevor Pering, Gunner Danneels, Muthu Kumar, Murali Sundar, and John Light. 2002. The personal server: Changing the way we think about ubiquitous computing. In *International Conference on Ubiquitous Computing (Pervasive 2002)*. Zurich, Switzerland.
[30] Fengli Xu, Pengyu Zhang, and Yong Li. 2016. Context-aware real-time population estimation for metropolis. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2016)*. Heidelberg, Germany.