Resense: Transparent Record and Replay of Sensor Data in the Internet of Things

Dimitrios Giouroukis Julius Hülsmann Janis von Bleichert Morgan Geldenhuys
Tim Stullich Felipe Oliveira Gutierrez Jonas Traub Kaustubh Beedkar Volker Markl
Technische Universität Berlin & DFKI
firstname.lastname@tu-berlin.de

ABSTRACT

As the scientific interest in the Internet of Things (IoT) continues to grow, emulating IoT infrastructure involving a large number of heterogeneous sensors plays a crucial role. Existing research on emulating sensors is often tailored to specific hardware and/or software, which makes it difficult to reproduce and extend. In this paper we show how to emulate different kinds of sensors in a unified way that makes the downstream application agnostic as to whether the sensor data is acquired from real sensors or is read from memory using emulated sensors. We propose the Resense framework that allows for replaying sensor data using emulated sensors and provides an easy-to-use software for setting up and executing IoT experiments involving a large number of heterogeneous sensors. We demonstrate various aspects of Resense in the context of a sports analytics application using real-world sensor data and a set of Raspberry Pis.

1 INTRODUCTION

The growth of the Internet of Things (IoT) has led to many disruptive technologies and applications such as smart homes, autonomous vehicle fleets, health and well being, personal and home security, and natural disaster management. In such applications, the IoT data (i.e., sensor data generated by devices connected to the Internet) may get very large and may involve billions of sensors [6]. Therefore, efficient means to automatically collect, store, and analyze massive amounts of IoT data plays an important role in our modern information-based society.

IoT research connects two communities: The database community deals with sensor data acquisition [3, 12, 14] and distributed query processing [8, 15]. The infrastructure and networking community deals with network connections, application and resource management across sensor nodes, and cloud computing [4, 9, 10].

For database researchers, it is important to develop and test data management solutions on IoT testbeds which include large numbers of sensor nodes and provide real networking and data processing conditions. However, it is hard to conduct repeatable experiments on such testbeds for two main reasons: *(i)* One needs to fine tune and test different versions of algorithms (A/B or split testing), but sensors data varies over time which leads unequal test conditions. *(ii)* Applications need to be tested on rare events which leads to extremely long test durations. For example, consider a sports analytics application that predicts injuries in real time. Player injuries are rare and highly important, but one cannot repeat them in the real world. Being able to replay sensor data (e.g., data recorded from sensors on players' body) on real test beds solves the issues stated above and enables database researchers to run repeatable experiments.

© 2019 Copyright held by the owner/author(s). Published in Proceedings of the 22nd International Conference on Extending Database Technology (EDBT), March 26-29, 2019, ISBN 978-3-89318-081-3 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

In this paper we present Resense, a framework which transparently emulates sensors and provides an efficient way to replay and record sensor data. Resense emulates sensors at the operating system level such that it is transparent for applications whether they acquire values from real sensors or from memory using emulated sensors. One can install Resense on testbeds without changing any application and gains the flexibility to switch between live sensor data and replayed sensor data easily.

Resense further provides mechanisms to orchestrate IoT experiments involving a large number of heterogeneous sensors. Our framework allows users to easily configure many different physical and/or emulated sensors as well as the data to be replayed. We provide an intuitive user interface for loading experiment configurations, for deploying experiment data to a large number of sensor nodes, and for starting/stopping experiments on all (or some) sensor nodes. The automatic deployment of the required data, the centralized configuration of sensor nodes, and the centralized control and monitoring of experiments reduce the administrative overhead when running IoT experiments and developing IoT applications.

Our demonstration highlights how easy it can be to setup and run experiments on a real testbed. We show various aspects of Resense using a set of Raspberry Pis as sensor nodes and some physical sensors. In particular, we demonstrate the record and replay functionality where attendees can attach physical sensors to a Raspberry Pi and use the graphical user interface to inspect the (physical) sensor readings, record them, and replay the recorded data. We also demonstrate setting up and executing a full scale IoT experiment based on real-world data from the DEBS 2013 Grand Challenge dataset [13]. The data consists of about 15 000 position events per second which were recorded at a football match in which sensors were embedded on the shoes of the players as well as the ball. Attendees will be able to use the Resense UI to deploy, control, and monitor the experiment and configure the sensor nodes.

In summary, this paper makes the following contributions:

(1) We show how to transparently emulate sensors in order to record and replay sensor data.
(2) We provide an easy to use software for setting up and executing IoT experiments involving large numbers of heterogeneous sensors.
(3) We demonstrate our software by recording and replaying real world sensor data in the context of sports analytics on a set of Raspberry Pis.

Resense is available as open source project[1] and runs on any GNU/Linux system independent of the underlying hardware (e.g., x86-based servers and ARM single-board computers).

The rest of the paper is organized as follows: in Section 2, we give an overview of our approach for transparently emulating sensors and orchestrating IoT experiments. Section 3 gives a

[1] https://github.com/TU-Berlin-DIMA/resense

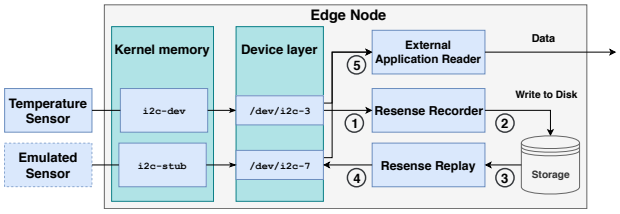

Figure 1: Recording and replaying data with Resense.

description of our demonstration. We discuss related work in Section 4 and conclusion in Section 5.

2 EMULATING SENSORS

Implementing the emulation of sensors while being capable of replaying scientific datasets is hard since the number of sensors that can be involved in a single experiment can vary. On top of this, the heterogeneity of the sensors makes it hard to include every possible sensor architecture in a single emulator. In this section we discuss the decisions behind the architecture of Resense and show how we overcome these obstacles.

2.1 Emulation Abstraction & Replaying Data

The first major contribution of this paper is to emulate a sensor transparently. Transparent emulation allows applications that request data from sensors to treat the emulated sensor as if it was a physical one. The applications cannot distinguish between physical and emulated sensors and expect the same behavior from them. The GNU/Linux kernel provides modules with a stable API and implementations suited for such a task. For example, it provides kernel drivers for commonly used sensor protocols like I2C, SPI, and UART among others. These kernel modules are fully featured drivers that emulate physical devices. We employ their capabilities in Resense in order to provide emulated sensors that are indistinguishable from physical sensors from the point of view of an application.

In more detail, the kernel modules allow the addition and removal of emulated sensors using file descriptors at the device layer of the host operating system. These file descriptors have a consistent naming scheme and are allocated in memory by the kernel module in order to be subsequently perceived as physical components by other parts of the system. Since different file descriptors provide the same API to external applications, it allows Resense to emulate different sensor types regardless of their underlying protocol (and communication bus) for which the host operating system includes its respective kernel module.

Figure 1 shows an end-to-end example from the perspective of a single edge node. A physical temperature sensor is attached to the edge node through an I2C bus. In addition, there is an emulated sensor provided by Resense. The `i2c-dev` kernel module[2] makes all I2C sensors accessible from userspace using character device files. Each device gets assigned a number, starting from 0. The device files follow the pattern of `/dev/i2c-{0,1,2,...}`. In our example, `i2c-dev` has allocated `/dev/i2c-3` for the temperature sensor. The `i2c-stub` kernel module[3], which is also depicted in Figure 1, is responsible for creating character device files for *emulated* devices. The emulated device files follow the same pattern as the ones created from `i2c-dev` and are indistinguishable from the point of view of a userspace application. In our example, the emulated sensor is allocated to `/dev/i2c-7`.

[2]https://www.kernel.org/doc/Documentation/i2c/dev-interface
[3]https://www.kernel.org/doc/Documentation/i2c/i2c-stub

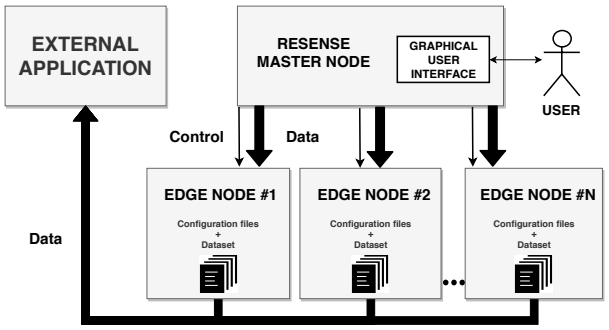

Figure 2: Resense architecture.

Three applications operate on top of physical and emulated sensors in our example: *(i)* The Resense Recorder, *(ii)* Resense Replay, and *(iii)* a reader of an external application. The applications act as follows: The `Resense Recorder` reads sensor values ① and stores them for later use ②. This allows for capturing events in order to replay them later on. The `Resense Replay` module can read the sensor values stored by the recorder ③ and replay them through the emulated sensor ④. We consider this process a *replay* of an experiment. The external application reader can access all sensors (the physical temperature sensor and the emulated sensor) in the same way ⑤.

The `Resense Recorder` and the `Resense Replay` module act as a bridge between the physical sensors of an edge node and the emulated sensors. Any application which consumes sensor data can consume values from both, physical and emulated sensors, since both types of sensors are exposed through device files. Our current approach replays data at the same rate at which they were captured. In future, we plan to support more fine grained control over the rate of recording and replaying. Moreover, our future work also includes time synchronization mechanisms across edge nodes in order to provide reliable and accurate replaying of sensor data. As a remark, the maximum number of physical and emulated sensors per edge node is limited by the host operating system as well as the node's hardware.

2.2 Experiment Orchestration

In order to administrate a huge number of edge nodes and sensors, Resense employs a Master/Slave approach. A visualization of the interaction between master and slave nodes can be seen in Figure 2. Users can control Resense through the graphical user interface of the master node. From there, users can deploy sensor data to sensor nodes, replay and record data, and monitor the progress of experiments. For external applications it is transparent whether they read data which originates from physical or emulated sensors.

Resense stores experiment setups pertaining to applications in configuration files in order to reload and rerun experiments as needed. An experiment consists of sensor data, edge nodes, emulated sensors, and physical sensors. Listing 1 shows an example configuration file. We first define the name of the experiment in Line 1. Starting from Line 2, we provide the list of edge nodes associated with the experiment. Each edge node has a unique id (Line 4), an IP address or domain name (Line 5), and the user name and password for remote access (Lines 6 and 7). Each edge node may host many sensors listed starting from Line 8. Each sensor has a unique id, an output type, and a sensor address. The sensor id refers to a sensor contained in the data file associated with the experiment. In our example, we connect two sensors from the DEBS 2013 grand challenge.

```
{
  "experiment": "debs-2013",
  "nodes": [
    {
      "edgeId": "edge-1",
      "host": "192.168.1.3",
      "user": "pi",
      "password": "pi",
      "sensors": [
        {"sensorId": "ball", "type": ["GPS"], "address": 3},
        {"sensorId": "referee_left", "type": ["GPS"], "address": 4}
      ]
    },{...}
  ]
}
```

Listing 1: An excerpt of sample configuration file.

In order to run an experiment with Resense, users first create a configuration file through the graphical user interface or load an existing experiment configuration. Resense automatically splits datasets associated with the experiment such that each result file contains the data of one individual sensor declared in the configuration file. The master node deploys the sensor data and the node configuration to each edge node as needed. Each edge node now creates the file descriptors for the emulated sensors according to the configuration provided by the master node. Once the deployment and setup are complete, the user can start the experiment from the dashboard. Edge nodes will then replay sensor data as described in Section 2.1 and shown in Figure 1. Users can also pause, resume, stop, and restart experiments through the dashboard. During execution, Resense monitors the progress of the experiment and displays the status of sensors and edge nodes as well as the currently replayed data.

3 DEMONSTRATION

We demonstrate Resense on a Raspberry Pi testbed as shown in Figure 3. In our setup, the laptop machine acts as the master node, five Raspberry Pis as edge nodes, and some of the edge nodes are equipped to read data from physical sensors (a GPS, an accelerometer, and couple of ultrasonic sensors).

In our demonstration, attendees can record sensor data, replay the recorded data, and set up experiments with data from real-world data sets. Thereby, users can combine physical and emulated sensors installed onto our demonstration platform. Through the Resense dashboard, users can monitor edge nodes and sensors. We further provide an example application which is independent of Resense, reads data from emulated and physical sensors, and streams the live-data to a visualization dashboard.

3.1 Real-World Testdata

As an example, we use the datasets of the DEBS 2013 Grand Challenge [13], which was recorded at a football match and contains the speed, acceleration, and position of the players, the referees, and the balls used during the match. We chose this dataset because it provides sensor values recorded at a high data rate. The tracking frequency is 200Hz for players and of 2000Hz for the ball. Players are tracked with two sensors located on their shoes. Goal keepers are equipped with two additional sensors located at their hands. Overall the dataset provides values from 37 sensors with a total data rate of 15000 position events per second.

3.2 Demonstration Platform

The Raspberry Pi single board computer is our platform of choice for this demo. Its cost effectiveness, its plethora of available connection protocols as well as the fact that it is popular for sensor-based research were the most important factors that lead to our decision. We use Raspberry Pi 3 Model B for our demonstration and the latest version of the operating system that is available, specifically version `2018-10-09` as of January 21, 2019.

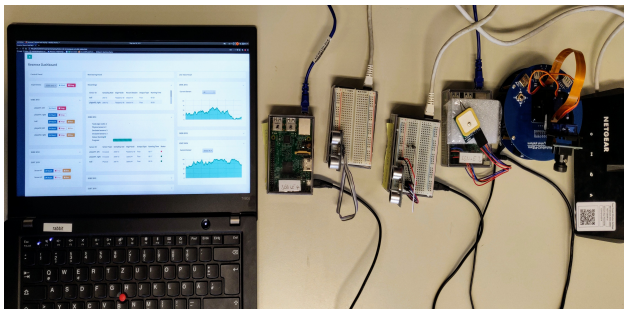

Figure 3: Resense demonstration setup.

For our demo, we choose to focus on the I2C protocol and the `i2c-stub` kernel module to emulate sensors from file descriptors. The protocol supports serial, 8-bit oriented, synchronous, bidirectional data transfer. Synchronization of devices on the bus uses a Master/Slave architecture. Moreover, I2C supports multi-master configurations with collision detection. We chose I2C over the other protocols that are available on a Raspberry Pi board because of the simplicity of the protocol and the maturity of the code of the kernel modules associated with it.

It is worthy of note that Resense is not tied to this demonstration platform. Resense runs on any GNU/Linux based operating system and is agnostic to the hardware architecture. This allows for running experiments with Resense on a large variety of hardware architectures and helps to emulate sensors that lack driver support for multiple architectures.

3.3 Demonstration Scenario

Figure 4 shows a screen shot of Resense's UI. The dashboard has three panels: a control panel on the left, a monitoring panel on the center and a live-view panel on the right.

The control panel allows either for selecting an existing experiment or creating a new one to be run. For each running experiment, the panel shows the list of sensors involved and the available controls. The controls depend on whether the sensor is physical or emulated. Readings from physical sensors can be stopped, resumed, or can be recorded (for replaying later). For emulated sensors, their readings can either be stopped or resumed. By default, all sensors are activated upon starting an experiment. The monitoring panel displays statuses of current recordings and experiments that are running. In particular, the box for recordings shows details about sensors and data that is being recorded. The box for each experiment shows progress, number and type of sensors, number of edge notes, and details about the sensors and the data being read. Finally, the live-view panel shows time series data for sensor readings for currently running experiments and allows to select specific sensor(s) from which data should be rendered.

During the demonstration, the attendees will run an experiment by selecting an existing configuration file (e.g., based on the DEBS 2013 dataset) or by creating a new one. Attendees can interactively control individual sensors and view readings in the live-view panel. Attendees will also be able to create an experiment involving available sensors, record sensor data, and replay recorded data.

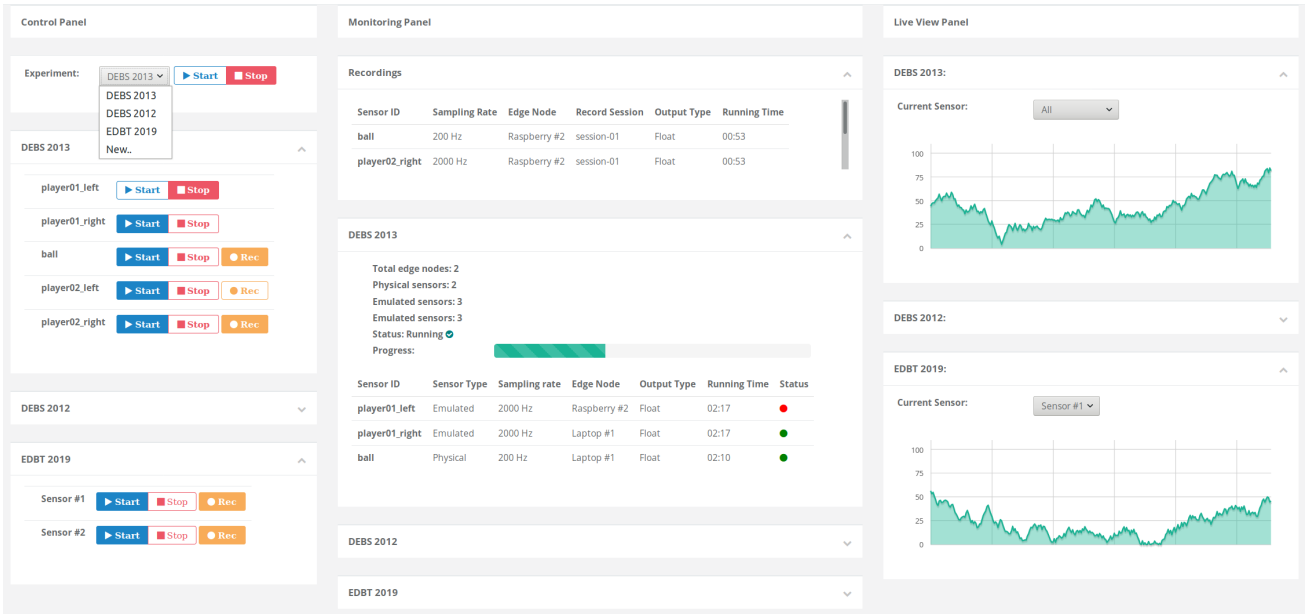

Figure 4: Resense dashboard

4 RELATED WORK

In the existing literature, Chernysov et al. [1] give an overview on the topic of emulating and simulating IoT infrastructure. According to their work, IoT simulators are categorized into three different sets: *(i)* perceptual emulation, emulates all of the layers of an IoT infrastructure, *(ii)* network emulation, focuses on network properties and *(iii)* application level emulators, emulate workloads only on the application layer. To the best of our knowledge, our work cannot be constrained to only these three types because our implementation is capable of ingesting sensor data at the kernel level of the host operating system. Thus, it is fully transparent to external software that can itself be already in one of these aforementioned categories.

In contrast to the work on IoT simulators [4, 5], our implementation differentiates between the edge nodes and the sensors and focuses on emulating the sensors and reading from sensors. This allows for more granularity while replaying sensor data and can give a more detailed view of the experiment. While other solutions [7] require a fixed full-stack emulation, our work provides a mixed approach since it is able to integrate emulated as well as physical sensors. This makes it easier for testing new physical sensor architectures with existing ones.

Emulating sensors for conducting experiments have also been studied using FPGAs [2, 11]. Usually, FPGAs are first designed using a hardware description language (like VHDL/Verilog) and later implemented into the actual hardware. Our solution reduces the time and the cost involved in prototyping an experiment as it does not depend on any specialized hardware.

5 CONCLUSIONS

In this paper we showed how to emulate sensors to replay recorded sensor data independent of the underlying hardware or software. We presented the Resense framework that allows transparent record and replay of sensor data and provides an easy to use software for setting up and executing IoT experiments. We demonstrated various features of Resense using real world sensor data, a set of Raspberry Pis, and some physical sensors.

Acknowledgments: This work was supported by the German Ministry for Education and Research as Berlin Big Data Center (01IS14013A) and the European Union's Horizon 2020, under the Marie Skłodowska-Curie grant agreement No 765452.

REFERENCES

[1] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally. 2018. Internet of Things (IoT): Research, Simulators, and Testbeds. *IEEE Internet of Things Journal* 5, 3 (2018), 1637–1647.
[2] Antonio De La Piedra, An Braeken, and Abdellah Touhafi. 2012. Sensor systems based on FPGAs and their applications: A survey. *Sensors* 12, 9 (2012), 12235–12264.
[3] Amol Deshpande, Carlos Guestrin, Samuel R Madden, Joseph M Hellerstein, and Wei Hong. 2004. Model-driven data acquisition in sensor networks. In *VLDB*. 588–599.
[4] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. 2017. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience* 47, 9 (2017), 1275–1296.
[5] Son N Han, Gyu Myoung Lee, Noel Crespi, Nguyen Van Luong, Kyoungwoo Heo, Mihaela Brut, and Patrick Gatellier. 2015. Dpwsim: A devices profile for web services (DPWS) simulator. *IEEE Internet of Things Journal* 2, 3 (2015), 221–229.
[6] Mark Hung. 2017. Leading the IoT, Gartner Insights on How to Lead in a Connected World. *Gartner Research* (2017), 1–29.
[7] Vilen Looga, Zhonghong Ou, Yang Deng, and Antti Yla-Jaaski. 2012. Mammoth: A massive-scale emulation platform for internet of things. In *CCIS*. 1235–1239.
[8] Samuel R Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. 2005. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst* 30, 1 (2005), 122–173.
[9] Alli Mäkinen, Jaime Jiménez, and Roberto Morabito. 2017. ELIoT: Design of an emulated IoT platform. In *PIMRC*. 1–7.
[10] Mirjana Maksimović, Vladimir Vujović, Nikola Davidović, Vladimir Milošević, and Branko Perišić. 2014. Raspberry Pi as Internet of things hardware: performances and constraints. *Design Issues* 3 (2014), 8.
[11] Eric Monmasson and Marcian N Cirstea. 2007. FPGA design methodology for industrial control systems—A review. *IEEE Trans. on Industrial Electronics* 54, 4 (2007), 1824–1842.
[12] Conor Muldoon, Niki Trigoni, and Greg MP O'Hare. 2011. Combining sensor selection with routing and scheduling in wireless sensor networks. In *VLDB*.
[13] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. 2013. The DEBS 2013 grand challenge. In *DEBS*. 289–294.
[14] Jonas Traub, Sebastian Breß, Tilmann Rabl, Asterios Katsifodimos, and Volker Markl. 2017. Optimized on-demand data streaming from sensor nodes. In *SoCC*. 586–597.
[15] Yong Yao and Johannes Gehrke. 2002. The cougar approach to in-network query processing in sensor networks. *ACM Sigmod Record* 31, 3 (2002), 9–18.