SmartSense: A System for Ambient Sensing at MIT

Sami Alsheikh, Feras Saad, Nick Uhlenhuth

{*alsheikh, fsaad, nicku*}*@mit.edu*

Outline

Collecting and monitoring data about environmental conditions, such as temperature, humidity, radiation, and water pressure, is of key interest to the Facilities Department at the Massachusetts Institute of Technology. This task is now achievable due to the recent boom of inexpensive, low-power sensors that are capable of basic computation and communication. This paper proposes a design for a campus-wide ambient sensing system at MIT that is simple, reliable, and scalable.

The basic design is composed of three modules: *sensors* will be deployed at various locations around campus, and relay their readings to a *facilities central server* (FCS) via *smartphone* intermediaries. This is achieved by designing an API for sensor-smartphone and smartphone-server communication protocols. A SQL-like data model exists on the FCS, which permits highly flexible querying of the sensor data.

1. System From High Level

After facilities install sensors across campus, each sensor collects readings of a particular type. Transferring sensor data to the centralized server is achieved via crowdsourcing; members of the MIT community carry smartphones which are registered to interact with sensors via BLE[1]. Figure 1 shows the flow of information between system modules.

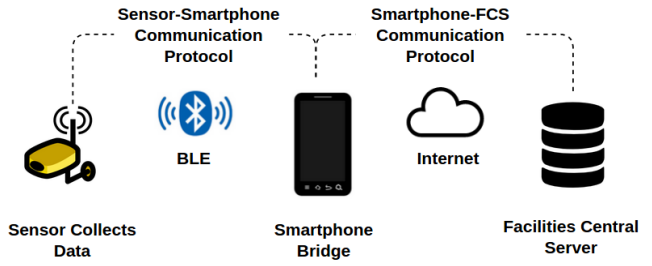


Figure 1: Overview of System Modules and Communication

The next sections of this paper will describe the sensor, smartphone, and central server modules, as well as the communication protocol API between each module.

2. Sensor

The most important questions regarding sensors are determining how they should be named, how they should manage sensor readings in memory, and how to make decisions about transmiting data to smartphones. A main design consideration is maximizing sensor battery life.

2.1. Naming

Each sensor is uniquely identified by a 148-bit sensor id (sid) which is included its advertisement messages. The purpose of the sid is to encode the exact location of the sensor on campus, and the type of sensor (light, temperature, etc). The Department of Facilities maintains highly detailed floor plans of every MIT building[2]. For example, E14-790C means room 790C (on the seventh floor) of building E14. It is assumed every sensor is placed in an addressed location.

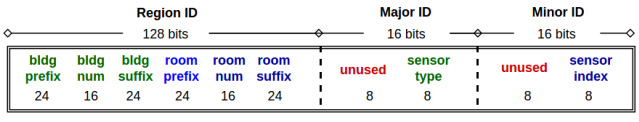


Figure 2: Naming Scheme

Figure 2 shows how every sensor in the system can be uniquely named and identified. The Region ID encodes the exact location, where prefixes and suffixes are 3 bytes that represent 3 ASCII characters. The Major ID holds the sensor type as an 8 bit number, and the sensor index is used to differentiate (up to 256) sensors of the same type and location.

[1] Bluetooth Low Energy is a wireless personal area network technology

[2] Accessible with MIT certificates at `https://floorplans.mit.edu`

2.2. File Management

Because sensors record one reading per second, it is important to create a simple yet powerful data storage model. Sensors store readings in files that are created every five minutes upon initialization of the sensor. More specifically, at every five minute timestamp, one file is created for *normal* data and another is created for *anomaly* data. These two files are named using the format (*timestamp_boolean*), where *timestamp* is the real world time when the file was created, and boolean indicates the type of readings stored in the file (0 for normal, 1 for anomalous).

When the sensor records a non-anomalous (anomalous) reading, it writes the reading into the most recently created *normal* (*anomaly*) file. Because new storage files are created every 5 minutes, if a file is empty after a 5 minute period, it is deleted to save space without deleting data. For example, suppose the normal and anomaly files are created at 15:00:00. If there are no anomalies between 15:00:00 and 15:05:00, then the anomaly file is empty and will be deleted.

The vast majority of files will only be deleted when the sensor receives an `ACK` from a mobile device. This `ACK` specifies the name of the file that can be cleared from the sensors storage.

Each sensor reading is 6 bytes (4 bytes timestamp and 2 bytes reading value), which means a sensor can record readings at 1 Hz for approximately 16 days before the 8 Megabyte capacity is exhausted. In the rare case that storage is full before connecting to a smartphone, the oldest *normal* file will be deleted; *anomaly* files are not to be deleted to make space for new data.

2.3. Transmission Behavior

A sensor can only transmit data files when connected to a smartphone via BLE. For this reason, connecting to a mobile device is prioritized. The sensor connects with the first consenting mobile device that is in range. Because the initial connection is expensive (1 mA) relative to maintaining a connection (25 μA), the sensor stays connected with the current smartphone until it is out of range or the user disconnects. This minimizes the number of new connections that have to be initiated.

When a smartphone first connects to a sensor, all non-active files are transmitted to the mobile device. More specifically, the sensor transmits all files (*normal* and *anomalous*) that have timestamps strictly less than the timestamp of the current file. While the mobile device is still connected to the sensor, only non-active files are sent (at a rate of two files per five minutes).

3. Smartphone Bridge

The smartphone application will allow the user to view current connections and send queries to the FCS. When acting as a background application, the application serves as the link between the sensor and FCS. When acting as a foreground application, the application allows the user to query the FCS and see which sensors it is connected to.

3.1. Background Application

The background application is always running as long as the mobile device is on. Once a sensor connection has been established, a smartphone will receive sensor readings from each non-active file on a connected sensor. The application will allow a mobile device to connect to up to ten sensors, though we do not imagine this cap will be reached regularly. The phone transmits received sensor data to the FCS, waits for an acknowledgement that the data has been received, and notifies the sensor to delete the transmitted files.

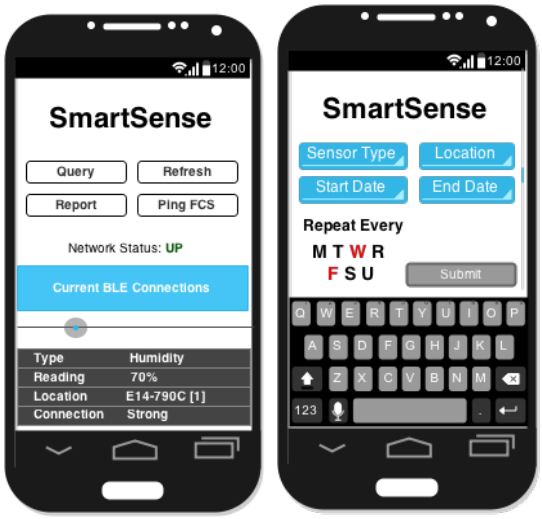
The file transmission and acknowledgement will not always work so smoothly, so we introduce a fault-tolerance scheme. In the event a smartphone dies or loses connection at any point after receiving sensor data, the application will send the data to the FCS as soon as the phone is able to. If the mobile device is now unable to notify the sensor to delete the transmitted files, the smartphone notifies the FCS which sensor and files have not yet been acknowledged.

When a phone connects with the FCS for the first time in the day, the FCS will send a list of sensors and messages to relay to those sensors. The mobile application will parse this message and map all sensors that have pending messages to the messages meant for that sensor in a hash table, abstractly in the form $(\langle sid_1, m_1\rangle, \ldots, \langle sid_k, m_k\rangle)$. Once a connection is made with a sensor, the application will check its table to see if there are any messages to send to that sensor. If one of these messages is sent to the sensor, the application notifies the FCS.

3.2. Foreground Application

When the user opens the application, he will see any current BLE connections to sensors and a button offering query functionality, as shown in Figure 3a. There will also be a link to a form to report a problem. The application will derive sensor information from the sid. The location is directly determined from the floor plan mapping in the Region ID, the index can be read from the Minor ID, and the sensor type can be extracted from the Major ID.

The user can choose to query the FCS for sensor data if he logs in and waives his anonymity. This is so that the FCS can evaluate the user's permissions. Note that the user only reveals his identify if he would like to send a



(a) Landing Page (b) Query Page

Figure 3: Wireframe schematic for foreground smartphone application.

query. The querying UI shown in Figure 3b allows the user to submit a request for information to the FCS. A SQL generator will be used to produce the appropriate syntax to send to the FCS for execution.

4. Facilities Central Server

The FCS has three functions; store sensor readings, issue configuration commands to the sensors, and reply to queries from smartphones for data statistics.

4.1. Data Model

A single reading from any sensor in the system arrives to the FCS as a 4-tuple (s, v, a, t) where s is the 148-bit sid, v is the 16-bit reading, a is a boolean variable for anamolous reading, and t is the UNIX timestamp at which the reading was originally recorded (a and t are easily extracted from the `filename`). A simple SQL database will store incoming readings with the schema as shown in Table 1.

Duplicate data is avoided by enforcing the `UNIQUE` constraint on the pair of columns (timestamp, sid). If multiple smartphones send readings where these two fields match an existing record in the table, then a duplicate has been found and the readings are ignored.

Almost any kind of query can be performed using SQL commands; below are some examples that demonstrate the power of the database schema.

Find the average humidity on the second floor of building 10 during December 2014

Column	Data Type
timestamp	UNIX_TIMESTAMP
sid	BINARY(8)
sensor_type	BINARY(8)
sensor_reading	BINARY(16)
anomaly	BOOL
bldg_pre	CHAR(3)
bldg_no	INTEGER(2)
bldg_suf	CHAR(3)
rm_pre	CHAR(3)
rm_no	INTEGER(2)
rm_suf	CHAR(3)
sensor_index	INTEGER(2)

Table 1: FCS SQL Database Schema

```
SELECT AVG(sensor_reading) FROM fcs
WHERE
   (timestamp BETWEEN <2014:12:01:00:00:00>
       AND <2014:12:01:00:00:00>)
   AND (sensor_type  = <humidity>)
   AND (bldg_no = <10>)
   AND (rm_no BETWEEN <200> AND <300>)
```

Find maximum temperature on West Campus, between midnight and 6 am during the first week of August 2014

```
SELECT MAX(sensor_reading) FROM fcs
WHERE
   (timestamp BETWEEN <2014:08:01:00:00:00>
       AND <2014:08:07:00:00:00>)
   AND (DATEPART(hour,timestamp) >= <0>
       AND DATEPART(hour,timestamp) <= <6>)
   AND (sensor_type  = <temperature>)
   AND (bldg_pre LIKE <%W%>)
```

4.2. Communicating with Sensors via Smartphone Bridge

There are two situations in which the FCS must communicate with a sensor. First, the FCS can *reconfigure* a sensor, such as changing its threshold value, or reading frequency. Second, the FCS knows about sensors which never received an `ACK` for their transmitted files, as discussed in Section 3.1. To remedy this situation, the FCS sends `MISSED_ACKS` messages to smartphones upon initiation of the TCP session at the start of the day. The protocol for these messages is clarified in Section 5.2.

4.3. Identifying Malfunctioning Sensors

There are two ways that the FCS can identify malfunctioning sensors. If a particular sid reports a very high number of anomalies under regular conditions, this indicates the sensor may be faulty. The second way is by cross-validating readings; if the *name* of two sensors s_1

and s_2 differ only by the `sensor_index`, then we know these sensors are in the same location and of the same type. If readings differ significantly then something might be wrong with one of the sensors, or a sensor may have been moved. A simple front-end to the database can facilitate both these tasks.

5. Communication Protocols

This section outlines the communication protocol between the FCS and smartphones (over internet), and between smartphones and sensors (over BLE).

5.1. Smartphone to FCS

When the mobile device has a file to send to the FCS, it will transmit the sid of the sensor sending the data, the filename, and the data itself. Each data entry has a timestamp t and value d.

```
DATA <sid> <filename> <t_0 d_0 ... t_n d_n>
```

As described in Section 3.1, a smartphone may lose BLE connection with a sensor before it notifies the sensor that the FCS has received its file. In this case, the smartphone notifies the FCS about this situation.

```
MISSED <sid> <filename>
```

The FCS uses the smartphone as a bridge to send `MISSED_ACKS`, `THRESHOLD`, and `FREQUENCY` messages to a sensor. If a smartphone manages to relay one of these messages to its associated sensor, it notifies this success to the FCS.

```
SERVICED <sid> <original_command>
```

The only user-triggered message is a data lookup. The mobile application processes a user query for information and produces the corresponding SQL query. It sends a message to the FCS with the user Kerberos and SQL query.

```
LOOKUP <kerberos> <sql_query>
```

5.2. FCS to Smartphone

Once the FCS has received a `DATA` message from the mobile device, it sends it an acknowledgement of receipt.

```
ACK <sid> <filename>
```

The FCS will send out a message to all mobile users once a TCP connection is initiated for a given day. These messages include `MISSED_ACKS`, `THRESHOLD`, and `FREQUENCY`.

The mobile application will parse the aggregated `MISSED_ACKS` message and create entries $(\langle sid_1, m_1\rangle, \ldots, \langle sid_k, m_k\rangle)$ in a hash table, which maps a sensor to messages meant for that sensor.

```
MISSED_ACKS <sid_0 f_0 ... sid_n f_n>
```

The `THRESHOLD` and `FREQUENCY` messages are both desired configurations created by an administrator on the FCS, and are meant to be relayed to the sensor. These are batched messages, which reduces the network overhead of sending individual messages for each sensor.

The `THRESHOLD` message gives the lower and upper bounds on non-anomalous data.

```
THRESHOLD <sid_0 lower_0 upper_0 ... sid_
n lower_n upper_n>
```

The `FREQUENCY` message gives the desired time interval in seconds between sensor readings.

```
FREQUENCY <sid_0 interval_0 ... sid_n interval_
n>
```

5.3. Smartphone to Sensor

All messages sent from the mobile device to the sensor are those forwarded from the FCS. Notice that all methods are nearly identical. The major difference is that sensor IDs are removed and information is sent to a given sensor one at a time.

```
ACK <filename>
```

```
THRESHOLD <lower upper>
```

```
FREQUENCY <interval>
```

5.4. Sensor to Smartphone

Sensor readings are the only messages sensors will send to the smartphone.

```
DATA <filename t0 d0 ... tn dn>
```

6. Conclusion

We have presented SmartSense, a system used to collect and monitor data from sensors around the MIT campus. Our design prioritizes simplicity by modularizing the system and providing an intuitive communication interface. Fault tolerance and reliability are addressed by designing a scheme were sensors store and transmit a data file until receiving confirmation that the file has arrived at the central server.

To incentivize members of the MIT community is by ensuring participation is easy, such as incorporating the smartphone module into the MIT Android and iPhone apps (with user permission). Moreover, we plan to make sensor readings publicly available, which appeals to the many people at MIT who are passionate about big data analysis.

Future work will involve a full power analysis to argue that sensors can last an average of three years, and will incorporate feedback about the overall design from our collaborators.