Pulse

MIT Ambient Sensing Network

Katrine Tjoelsen (ktjolsen)

Sarah Vente (svente)

Daniel Ziegler (dmz)

6.033 Design Project Proposal

Recitation: Szolovits/Yang, TR1

March 20, 2015

1 Overview

MIT facilities seeks to build a campus-wide ambient-sensing system by deploying on the order of 100,000 sensors. These sensors communicate through Bluetooth Low Energy (BLE), and each sensor measures one of several different values (e.g. temperature, humidity, light intensity, etc.).

Facilities wants to collect all sensor data in the Facilities Central Server (FCS). Unfortunately, BLE has a short communication range (about 10-20 meters), and Facilities does not have the budget to install BLE access points all over campus to allow the sensors to communicate with the FCS directly. To solve this problem, we propose *Pulse*: a network that utilizes MIT community members' mobile phones as communication bridges between the sensors and the FCS.

The main modules of Pulse are the sensor nodes, the FCS, and the mobile device application (see **Figure 1**). The sensor nodes collect data and transmit it to the FCS via the mobile devices in a best-effort fashion, prioritizing important data and compressing data when space runs out. The FCS stores the data for quick retrieval and pushes configuration updates to the sensor nodes via the mobile devices. In addition to relaying messages, the mobile devices let users report issues to the FCS and query it for past data.

The design prioritizes simplicity over optimality, ensuring that data is transmitted and managed in a way that conserves sensor node battery, aims for reliable transmission, and allows for efficient querying, without introducing unnecessary complexity.

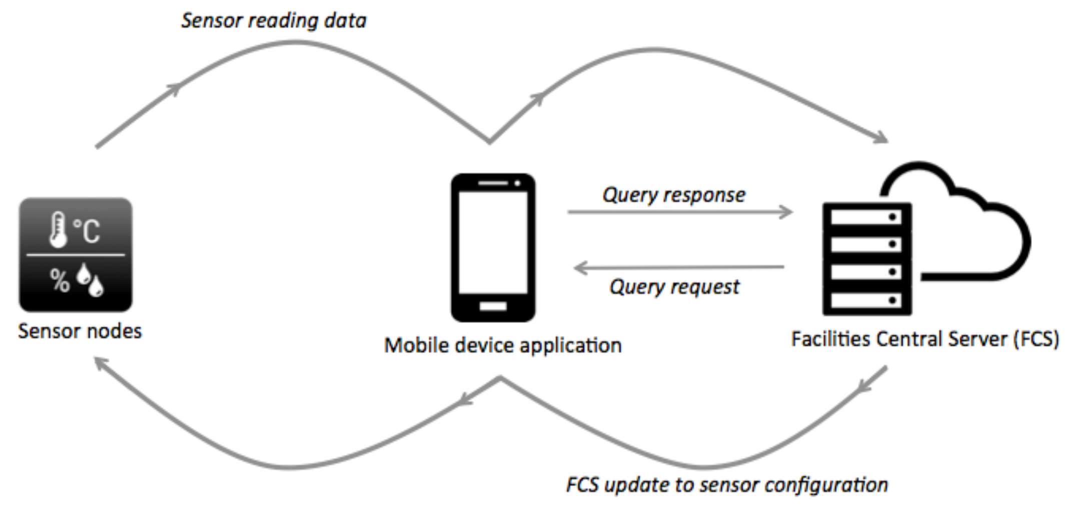
Sensor reading data
Query response
Query request
Sensor nodes
Mobile device application
Facilities Central Server (FCS)
FCS update to sensor configuration

Figure 1: The three main modules of Pulse are the sensor nodes, the mobile application, and the Facilities Central Server (FCS). The sensor nodes and FCS communicate through the mobile app, and the mobile app can query the FCS for archived data.

2 Design description

2.1 Sensor nodes

At configurable regular intervals, each sensor node takes a 16-bit sample with a 32-bit timestamp. The nodes transmit data to mobile devices using a strategy that prioritizes "anomalous" data, maximizes the probability that data makes it to the FCS, and conserves battery. Until the data is deleted, it is stored in flash memory and compressed when space runs out.

2.1.1 Transmission

Most of a node's energy is spent periodically broadcasting its presence to allow mobile devices to connect. To amortize the power spent establishing connections, a node suppresses the broadcast unless it has at an entire 4 KB[1] contiguous "chunk" of samples to send. Once a mobile device connects, the node continuously sends data using the link-layer protocol (described below) until either the connection fails or all the data has been transmitted.

Each node is configured with a range of values that are considered "normal", and chunks containing any anomalous samples are prioritized. Additionally, chunks are transmitted multiple times to increase the probability that they safely make it to the FCS. Thus, the chunks are kept in separate sets distinguished by the normality and the send count of the data: five[2] sets for anomalous data and three[3] sets for nominal data. First the unsent anomalous chunks are transmitted, newest first, then the once-sent anomalous chunks, and so forth. Once a chunk has been sent, it is moved to the next set, and after either five or three transmissions (depending on normality), it is deleted.

2.1.2 Link-layer protocol

Sensor nodes transmit data to the FCS via mobile devices and communicate with mobile devices over BLE. A sensor sends data to a mobile device by (1) broadcasting advertisement messages such that a mobile device can connect to it, and then (2) transmitting the data in 20-byte packets.

(1) Broadcasting advertisement messages. Nodes advertise themselves through BLE broadcasts once per second with a 128-bit region ID, 16-bit major ID, and 16-bit minor ID. All the sensor nodes advertise the same region ID. The 16-bit major ID and first 8 bits of

[1, 2, 3] This parameter is subject to revision and will be justified in the report.

the minor ID (24 bits total) is used to uniquely identify each sensor node. The usage of the remaining 8 bits of the minor ID is to be determined.

(2) Sending data in 20-byte packets. Every 20-byte packet will contain three readings (sample value and timestamp) of six bytes each and three bits that indicate whether or not a particular reading is anomalous. The message protocol is shown in **Figure 2**.

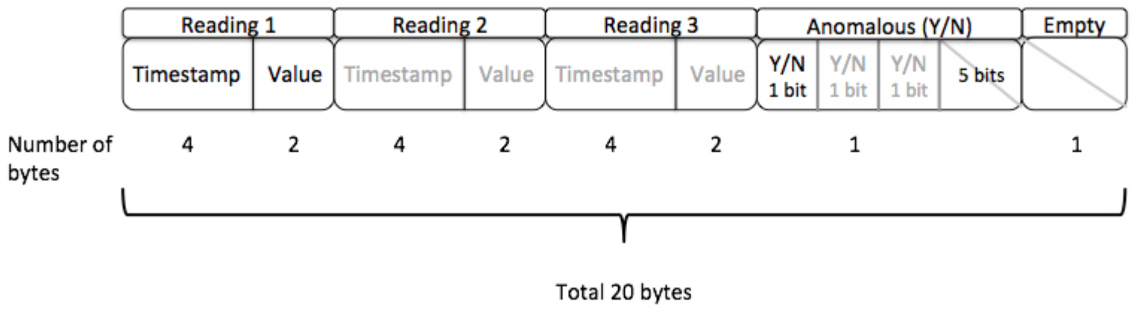

Figure 2: *Message protocol for sensor data transmitted to the mobile app.* Each 20-byte BLE packet contains three sensor readings (timestamp and value). For each reading, 1 bit communicates whether or not it is anomalous.

2.1.3 Compression

Some sensor nodes might not connect to mobile devices for a long time and run out of storage space. Rather than deleting data once this happens, the nodes instead "compress" a large sequence of low-priority data, averaging groups of 8 samples into a single sample. How exactly to choose the data and manage different levels of compression is left for future work.

2.1.4 Data storage

Each chunk is stored as its own file on the 8MB flash memory. In order to support efficiently finding chunks with the lowest or highest priority, for compression or transmission, the node keeps the files in a 5-level directory hierarchy (see **Figure 3**). At the root, there are two folders: one for anomalous data and one for normal data. Beneath these, there is one folder for every set, according to send count. The sets are deleted from at both ends and inserted into in random locations, and both types of access must be efficient. Thus, each set consists of three levels of directories according to the year, month, and day of the timestamp, with chunk files at the bottom. Empty directories are deleted. The number of entries in each directory is kept bounded because the nodes are likely to run a FAT-like file system with slow performance for large directories.

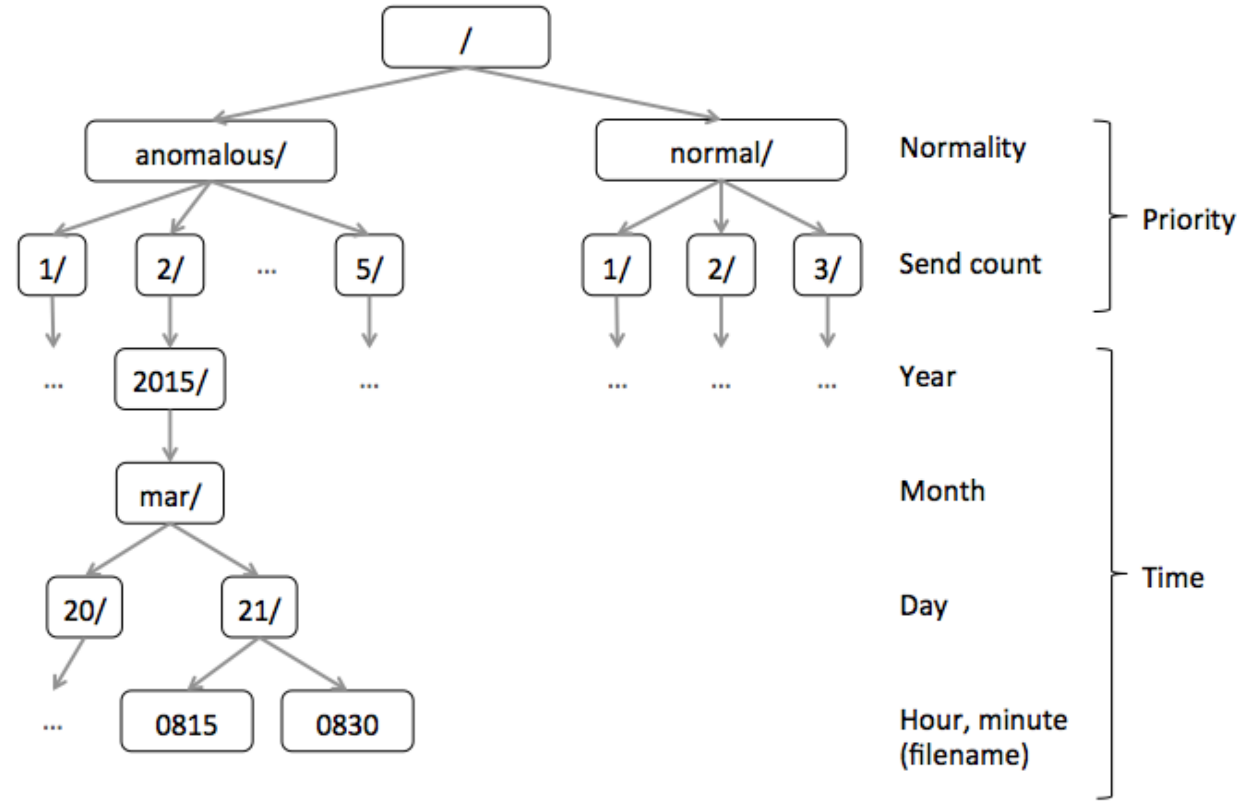

Figure 3: *File system structure for the sensor node*. The sensor data is separated based on normality, send count, and time. This allows for the prioritization of transmitting anomalous data and recent data, in a best-effort fashion.

2.1.5 Receiving updates from the FCS

The FCS sends update messages to the sensor nodes in a format described in detail in section **2.2.3**. This is the only type of message that the sensors receive. Upon receiving an update message, the sensor node extracts the update type (e.g. periodicity or anomaly threshold), the version number for the given update type, and the new value for the update parameter. If the version number is greater than the one that is stored, the parameter value is changed in the sensor node.

2.2 FCS software

The FCS is designed with two main purposes. First, the FCS must maintain a database of sensor readings and support queries to this database from the mobile app. Second, the FCS must be able to send messages through the mobile app, with the intended final destination being one or more sensors.

2.2.1 Database

The FCS maintains a SQL database consisting of two tables, shown in **Figure 4**. The first table stores the individual sensor information, and serves as a mapping from each sensor ID to its sensor type, its building, and its room. This table's primary key is the sensor ID, and it is also indexed by sensor type and location. The second table stores the sensor reading information. Each sensor reading entry contains: the sensor ID, the timestamp, the value of the reading, and whether or not the reading is considered anomalous. This table's primary key is the sensor ID + timestamp, and it is also indexed by both timestamp and sensor ID separately.

The FCS uses INSERT IGNORE to insert new information in the sensor reading table, which allows the database to ignore duplicate readings (sent from multiple mobile phones).

In order to maintain as much recorded data as possible, the FCS stores all data it inserts and does not delete nor compress data. The data storage will be supported by regularly adding hard drives to the system.

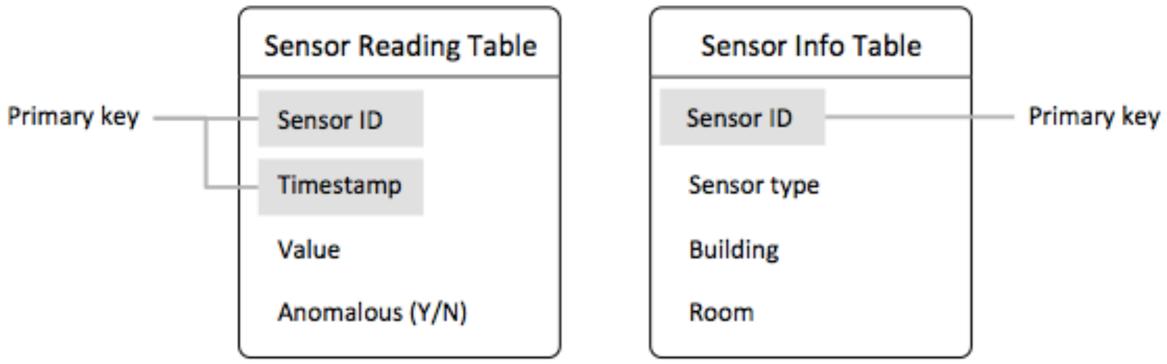

Figure 4: *Database tables in the FCS.* The sensor info table allows lookup of sensor IDs by type, building, and room number. The sensor reading table stores all sensor data and allows lookup by sensor ID, timestamp, and normality.

2.2.2 User query support

As described in section **2.3.4**, the mobile app sends queries to the FCS as JSON-parsed dictionaries specifying location, sensor type, time range, and the form of the return values. The two tables enable the FCS to support these queries efficiently. For example, if the user wants an aggregate value for the time series by sensor type and location, the system: (1) looks in the sensor info table to select all sensor IDs of the specified type and location, (2) looks up the selected sensor IDs in the sensor reading table to select the relevant sensor readings, and (3) computes the aggregate value.

2.2.3 FCS outgoing messages

The FCS sends two types of messages: responses to user queries, and directives to specified sensors.

When a mobile phone queries the FCS database, the FCS sends a response over HTTP. The exact format of this message is left for future work.

The sensor directives change the configuration of the sensors in some way. Examples of configuration changes include: changing the threshold value for what is considered an anomalous reading and changing the frequency at which the sensor takes readings. When an update is requested to specified sensors, the FCS pushes one message per sensor. Each message is pushed at random to a fraction[1] of the mobile devices, and includes the 3-byte sensor ID of the sensor to update and the update message. The update message itself is sent to the sensor as one 20-byte BLE packet, and this message protocol is illustrated in **Figure 5**.

The first two bytes of the update packet specify the update type (e.g. anomalous threshold update, periodicity update, etc.). The next four bytes specify the version number of the update. The version number is tracked separately for each sensor node and update type; for example, the third time we update the periodicity for a specific sensor, that update will have version number 3, which will not affect the version number for the sensor's threshold setting. The remaining 14 bytes are reserved for the update parameters, which vary by update type.

[1] This fraction is to be determined.

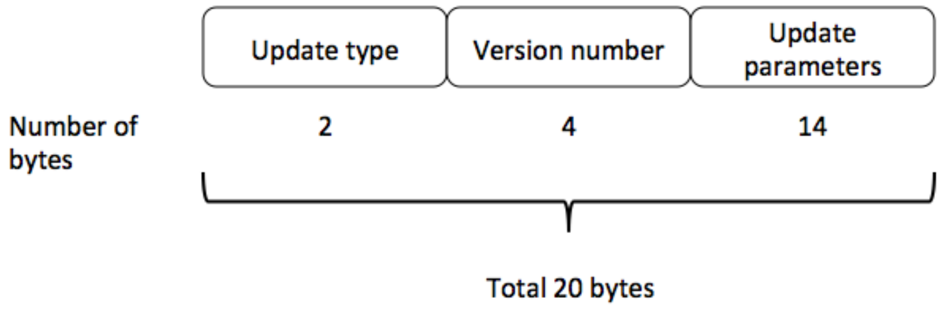

Figure 5: *Message protocol for sensor updates.* Each sensor update message is sent in a 20-byte packet to a sensor. These 20 bytes specify the update type, version number, and update parameters.

2.3 Mobile device application

The mobile device application has to (1) connect to and receive data from sensor nodes, (2) send sensor node data to the FCS, (3) send update messages from the FCS to the sensor nodes, (4) enable the user to query the FCS for information, and (5) report issues. Components (1)-(3) run as background tasks, while components (4) and (5) run in the foreground in order to allow user interaction.

2.3.1 Connecting to sensor nodes

As described in section **2.1.2**, sensors advertise their existence with a specified region ID that is the same for all sensors. The mobile device app listens for this region ID and identifies all sensor nodes within range that are broadcasting advertisements. The mobile device app connects to all sensor nodes that are broadcasting advertisements and whose sensor IDs are not in the list *prohibitedSensors*.

When the mobile device disconnects from a sensor during a connection, the app stores the ID of the given sensor in a list *prohibitedSensors*. When a mobile device successfully transmits stored data to the FCS, it empties the *prohibitedSensors* list, allowing it to reconnect to previously encountered sensors. This ensures that the sensor is able to send the same chunk of data to multiple different mobile devices, protecting the system from faulty devices.

2.3.2 Sending data from sensor nodes to FCS

The sensors send data to the FCS using best-effort delivery. The mobile device app parses the packets from the sensor and forwards the data to the FCS.

In a connection between a sensor node and the mobile device app, the mobile app receives four 20 byte packets. For each packet, the app does the following:

1. As outlined in **Figure 2**, each packet contains three timestamp-value pairs and three bits specifying whether each reading is anomalous. The app extracts the three timestamp-value pairs and for each pair it creates a dictionary that contains the sensor id, the time stamp for the reading, the value of the reading, and whether or not it is anomalous.
2. Finally, the mobile device pickles the dictionary that represents the sensor reading with JSON and sends it to the FCS server.

2.3.3 Updating sensor nodes with updates from the FCS

As described in section **2.2.3**, the FCS sends updates to the sensor nodes via a fraction of the mobile devices. The mobile device app stores a table that maps unique pairs of sensor IDs and update types to update messages it receives from the FCS; for example, one mobile app may store two different updates for the same sensor: an anomaly threshold update and a periodicity update, but it may not store two updates of the same type. When a mobile device connects to a sensor node whose ID is in the table, the mobile device sends the corresponding update message to the sensor, and removes the entry from the table.

If the mobile device app receives an update message from the FCS for a sensor ID and update type that already exists in the table, the app overwrites the old update message with the new update message.

2.3.4 Query the FCS for information

The mobile device application has a front-end component that enables the user to query the FCS for sensor data. The front-end application allows the user to select parameters for location, sensor type, time range, and whether to return an aggregate measure. The user may also query for anomalous readings. The query is constructed as a JSON-parsed dictionary that specifies the query parameters.

2.3.5 User issue reporting

To be able to respond directly to users' needs and incentivize people to install it, the app also allows users to report an issue in a particular location. The user enters a room number and a complaint from either a preset category (e.g. "the room is too cold") or a custom category (e.g. "there's a faint smell of sulfur"), and the mobile device uploads it. Then, the FCS can correlate the location and time with the data that it has received from nearby sensors and notify the appropriate Facilities employees. Since nodes do not transmit data immediately, it is possible that the FCS may not have the most recent data for a particular report. However, since nodes are not expected to alert about emergencies, this is an acceptable limitation.

3 Conclusion

The design of Pulse is simple while still meeting the requirements of the Facilities' use cases. The system supports archiving sensor readings in the FCS servers, detecting and prioritizing anomalous data, and allowing users both to query the FCS and to report issues to the FCS. These features are enabled by using mobile devices as communication bridges between sensor nodes and the FCS, and implementing best-effort communication mechanisms.

There are a few problems that remain to be solved. Specifically, the design will need to further specify how compression in sensor nodes works, the format of the FCS response for user queries, and how best to store user reported data in the FCS to allow facilities to act on it. Finally, it remains to estimate specific values of parameters (e.g. send count and size of file chunks) by making assumptions about failure probability and failure tolerance.