MegaSense

Uma Roy, Justin Lim, Philip Sun

Introduction

MIT has recently decided to upgrade motion detectors, temperature sensors and video cameras across campus to “smart” devices that are able to receive and transmit data from a facilities server (known as the FCS). They are interested in monitoring and collecting data from these devices for many reasons, including accessing camera data for safety considerations and long-term projects, and leveraging temperature and motion sensor data for cost and energy savings. Currently the devices on campus have minimal interaction with facilities. We design a system from scratch, called MegaSense, to allow facilities to effectively interact with the new smart devices and serve their needs.

While each smart device is equipped with bluetooth, the range of this bluetooth is small, necessitating a network of communication nodes (of gateways and repeaters) that allow for the smart devices to receive and transmit data from the facilities server. In designing MegaSense, we primarily aimed for simplicity, so that it can be easily maintained and generalized to other campuses. We also prioritize keeping system setup and maintenance costs low, as partial motivation for the installation of smart devices is to reduce energy and heating costs for the Institute. Other goals include include low latency for live video camera viewing for dangerous situations (known as *crisis mode*), where seconds can matter for safety. The nodes of our network form a tree-like structure, and node placement in this topology is optimized to remain simple while minimizing expensive components and latency from video cameras. Our facilities server design allows for long-term data storage and for facilities to easily retrieve and analyze data.

System Overview

Our system combines gateways and repeaters in a tree-like structure to allow for the sensors and cameras to communicate information to the FCS (facilities server) by sending packets through the network, which the FCS subsequently processes and stores.

Diagram 1: Overall System Layout

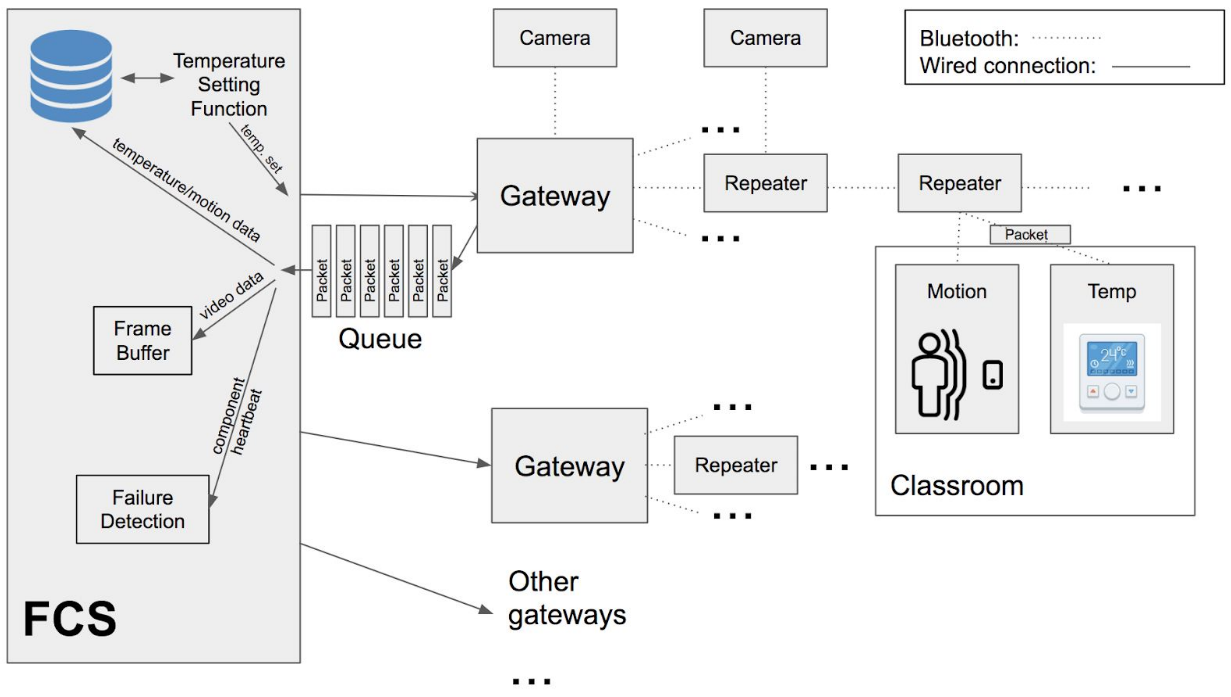

Diagram 1 description: *The FCS is connected to many gateways (wired connection), which are connected to chains of repeaters by bluetooth communication. These repeaters communicate with the motion and temperature sensors in each classroom, as well as video cameras. Packets carrying information from the sensors and cameras travel upward through this tree to the FCS, which processes the data and stores/analyzes it appropriately.*

Network Topology

The network topology allows for sensors to communicate with gateways using intermediate repeaters. As shown in Diagram 1, the FCS is connected to many gateways, which are connected to chains of repeaters that connect to sensors and cameras. Each node in the network has a single parent node that it transmits data to when sending data to the FCS. The exact layout of the repeaters and gateways in our network will depend on the particular layout of the classrooms on a given campus, but we provide generalizable rules for our network topology for any campus. For each two adjacent classrooms, we place one BLE repeater so that the motion detector and thermostat in each classroom are connected to the same repeater. The repeater for a pair of classrooms will be in range of the repeater for the next pair of classrooms (within a 30 foot radius, which is well within the size of a classroom). We place the gateways in "central" locations, for instance in the middle of a long corridor, near video cameras. This gateway will be responsible for the repeaters in that corridor. If a video camera in the hallway is far away from the gateway, then a repeater is placed close by and is connected to the chain of other repeaters to form a path to the gateway.

There are 15000 classrooms, so with 50 classrooms per gateway and 2 classrooms per repeater, our design will use approximately 300 gateways and 7500 repeaters. On average, each gateway will be responsible for 50 classrooms and 4 cameras (since there are 1000 cameras total). The gateway is in the middle of the corridor, so it will take at most 6 or 7 hops between repeaters to reach the outermost sensor in our network (Diagram 2). This introduces a latency of at most 0.7 seconds in transmitting data from a video camera to the FCS (if done in real-time), which minimizes latency while also keeping the number of gateways low, to keep costs low. The tree-like structure of our network topology makes routing very simple, and the placement of nodes strikes a balance between minimizing latency of data transmission and minimizing number of expensive gateway components.

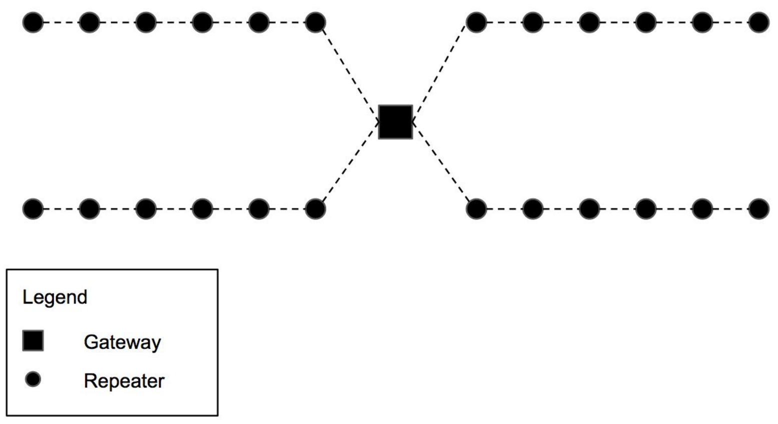

Diagram 2: *Layout of gateways and repeaters in a hallway. The gateway is in a central location and the repeaters are on either sides of the corridor.*

Network Discovery

The FCS has knowledge of the entire tree structure and stores it in its persistent memory. However the other nodes in the network discover their parents and children in the tree structure using component IDs.

Component IDs: Each component of the network has a 48-bit ID. We use a hierarchical naming system similar to those of IP addresses. Each ID has one chunk of 13 bits, followed by 7 chunks of 5 bits, as described in Diagram 3.

Diagram 3: Component IDs

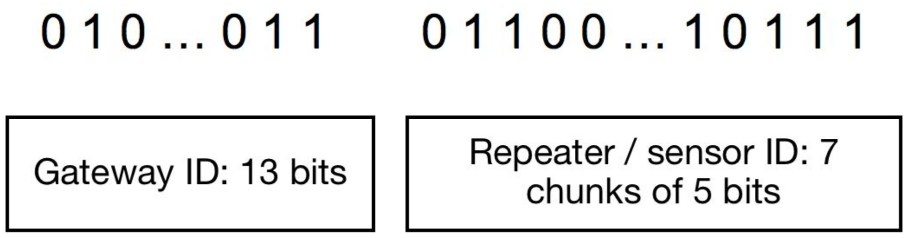

Diagram 3 description: An example of how a 48-bit component ID would look like. The ID describes a path from the FCS to the component. In this case, to send a message to the component, the FCS sends it to the gateway identified by the first 13 bits. It then sends it to the child with ID 01100 and so on. If a chunk is filled with 0 bits, then the address has ended.

On startup: Prior to startup, each device will already have knowledge of the structure of the tree around it, specifically, its parent and children. We are assuming this can be hardcoded into the device binary, as the number of parents and children are small. Upon startup, each device broadcasts its ID to its surroundings and simultaneously receives the IDs of the devices around it. Once a parent device receives a message containing the ID of its child, it establishes a connection to that device and other messages can be passed between them.

Routing and Communication Protocol

Once the network discovery process has occurred and the smart devices, repeaters and gateways know their parents and children in the tree structure, these devices must all communicate with each other to transmit data and route data to and from the FCS and the sensors/cameras.The data is transferred through the network in packets with 64 bit headers and space for 20 bytes of data (the maximum allowed by bluetooth).

Routing

Packet headers: The first bit of the header designated whether or not the FCS is sending the data. If the FCS is sending the data, the next 48 bits are the ID of the destination. If not, then the next 48 bits are the ID of the sensor who sent the data. Thus a node can determine from the header whether the packet is meant for the FCS or for a smart device and who sent the packet.

Routing Procedure: When a node receives a packet, it determines from the header whether the destination is the FCS or a smart device. If the destination is the FCS, since each node knows its unique parent from node discovery, the node sends the packet to its parent. If the gateway is transmitting to the FCS, it transmits the packet to the correct port depending on whether it originated from a temperature or motion sensor vs. a camera. If the destination is a smart device, the node can easily compute the unique child to send the packet to because of the hierarchical ID scheme.

Data Transmission From Sensors

For each sensor we specify a procedure for creation of packets, including specifying packet data and additional header meta-data, to fulfill the constraints of the system.

Motion + Temperature Sensor: For each classroom, the FCS needs to know the last time motion was detected in that room and an at-most 5 minute old temperature reading. Every minute, the motion detector runs `get_time()` to get a 32-bit timestamp of the last time motion was detected in the room and sends this timestamp as well as the current timestamp as packet data. The temperature sensor runs `get_temp()`, which returns a 32-bit float of the current temperature, and sends this float, along with the current timestamp, in a packet. Because packets can be lost and these packets consume very little bandwidth, we send data more frequently that strictly necessary. When the temperature sensor receives a packet from the FCS to set its temperature, it will run `set_temp(body)`where body denotes the 32-bit float in the body of the packet.

Video Camera: Every second the camera runs `get_latest_frame()` to get the latest frame in the buffer to transmit to the FCS. Each frame the video camera captures is 28 kilobytes. Each frame is broken up into a predetermined grid of ~2400 rectangular chunks of 12 bytes each, and for each chunk, the camera sends a packet with header meta-data containing the chunk number (12 bits), with the packet data containing the timestamp (4 bytes), frame number (4 bytes) and and bytes of the chunk (12 bytes). In this manner, a single frame takes 2400 packets to send fully. Since each gateway is responsible for around 4 video cameras and can push 16 MBit/sec (2000 kilobytes/sec), the bandwidth constraints at the gateway level are not a concern. Each repeater has a bandwidth of 2 Mbit/sec, and any repeater with a video camera only has 1 other connection to an upstream repeater, so the effective bandwidth for the video camera is 1 Mbit/sec. Because the depth of our tree is at most 6 or 7, and gateways are placed close to cameras, so no chain of repeaters has more than 2 cameras transmitting data to the gateway, our system is able to handle the bandwidth of videos continuously transmitting. This makes handling of crisis mode simple and also the latency in crisis mode very low.

We decide not to use TCP in our communication protocol because it introduces too much extra latency for our data. In doing so, we make a tradeoff in favor of simplicity and better latency in sacrifice of extreme reliability, which we feel is unneeded in this scenario.

Detecting Failures

Our system can detect failures of communication nodes or devices with a system heartbeat, but does not have any redundancies in place.

System heartbeat: The FCS sends out packets periodically (every half hour) with the destination ID in the header coded to signify a 'heartbeat' packet to all the gateways. Every device upon receiving this packet sends 5 packets back to the FCS with the data containing their software version number and forwards the packet to their children. For any node the FCS did not receive any acknowledgement packets from, it signals that the device is broken (note the false positive rate is low since the change of losing all 5 packets is quite low).

No redundancies: In case of failure, our system does not have redundancies, and we simply lose all the data during that period of time--a tradeoff we make in the favor of simplicity and reduced costs (as adding extra components costs money). In our system, only gateways can fail and have sufficiently low fail-rates, meaning not much data will be lost.

FCS

The FCS must have processes to analyze and store the data it receives from the sensors. It also must have processes devoted to detecting failures and updating the sensors, as well as deleting old data on the server. Since the FCS has ample compute power and storage (100 TB), we do not worry about space or performance constraints, only that the FCS serves the facilities needs well.

FCS Data Storage

The FCS will use a relational database to store temperature and motion data, and a file hierarchy for camera data. We chose to use a database for storing temperature and motion data because we foresee this data being used for analysis that would be cumbersome to implement without SQL. For example, to detect thermometer malfunctions, we may want to find all thermostats whose reported average temperature in the past hour was above 100 or below 40 degrees Fahrenheit. If we used a flat file storage system instead of SQL, this kind of query would be very difficult to implement; however, with a database it's trivial.

In contrast, relational databases don't handle video data well, so we choose to store videos directly in the filesystem, using folders to organize them. In a root folder named `videos`, there is one folder for each camera, with the folder name equal to the camera's device id. There is a file of compressed video for each hour of footage, allowing easy viewing of video from the past week by facilities workers.

FCS Processing Unit

We will implement four threads that will run in a process on the FCS: one for recording temperature/motion data and setting temperature, one for recording video data, one for deleting old data, and one for detecting failures of network components. The process, when first started, uses `fork()` to start these processes, which then operate effectively independently.

Temperature and Motion Data: The first process listens to incoming temperature and motion data on a network port and writes the data to the database. A database write is done per incoming packet of information. This is simple because temperature and motion data are both small enough to fit in a single packet, so no piecing of packets is needed. This process also sets room temperatures; when temperature data arrives, we also query the database for the past time we saw motion in the room. If we saw motion recently and the temperature is low, we raise it; if we haven't seen motion in two hours but the temperature is high, we lower it.

Video Frames: The second process records incoming video frames. For each camera, the FCS allocates a circular buffer of 201.6MB. This is enough to store two hours of uncompressed video footage. Whenever a video frame comes in, the FCS appends it to the proper circular buffer. Since each video frame takes ~1400 packets, we have to piece together the packets to make a frame using information in the packet headers. Using the timestamp and chunk number in the header, the process computes where into the buffer to index. If the circular buffer is more than half full, this process will spawn a thread that will compress the first half of the buffer, one hour's worth of video, and write it to disk in the location described in the previous section. This child thread will clear the first half of the buffer as it compresses the frames.

The total memory usage of all the camera buffers will be in the high hundreds of gigabytes, which is reasonable for a modern server. The total cost of such memory is negligible compared to the cost of the gateways and other networking equipment.

Deletion: The third process deletes old data by running periodic commands to delete SQL database information over a year old and video files with timestamps over a week old.

Crisis Thread: Note there is no need for an explicit crisis thread in our system, because our system can display real-time camera footage of any camera at all times as all data from video cameras is constantly transmitted. The second process will be paired with a web front-end that allows facilities staff to inspect the buffer (and therefore view the live footage) of any desired camera.

Updates: Our system currently does not handle system updates, which remains to be solved in future iterations.

Conclusion

MegaSense utilizes a simple tree-like layout of nodes in its communication network for simple routing of information from smart devices to the FCS using packets. By having sufficient number of gateways and repeaters in our network, there is sufficient bandwidth for video cameras to transmit live and with low-latency. Because our system prioritizes low cost and simplicity, there is no redundancy to deal with failure, which is a point of improvement for future iterations. The

FCS processes the incoming data and stores it to meet facilities needs. Future iterations would improve reliability, and include more in depth cost analysis and ability for the FCS to distribute software updates, amongst other additions.