MegaSense

A Smart Device System for MIT Facilities

By: Uma Roy[1], Justin Lim[2], Philip Sun[3]

Monday, May 7, 2018

[1] umaroy@mit.edu, (Rec) Howard Shrobe @ 12
[2] justinl@mit.edu, (Rec) Howard Shrobe @ 1
[3] sunphil@mit.edu, (Rec) Howard Shrobe @ 12

Table of Contents

Section 1: Introduction

MIT has recently decided to upgrade motion detectors, temperature sensors and video cameras across campus to “smart” devices that are able to receive data from and transmit data to a facilities server (known as the FCS). Facilities is 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 energy savings. Currently the devices on campus have minimal interaction with facilities. We design a new system called MegaSense to allow facilities to effectively interact with the new smart devices and serve their needs.

Each smart device is equipped with bluetooth, but the limited range of bluetooth necessitates a network of communication nodes to forward bluetooth traffic to the facilities server. The nodes in the network must be carefully placed to ensure full coverage of the MIT campus. Other considerations in node placement include network bandwidth and latency constraints. Video cameras require relatively high bandwidth, so the network must be able to handle this traffic adequately. Latency constraints are primarily dictated by a mode for viewing potentially dangerous situations (called *crisis mode*), where a live camera stream is necessary. The communication network has to ensure data from video cameras is able to reach the server quickly in this scenario.

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, for budgeting concerns. Other goals include include low latency for live video camera viewing in *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 meet bandwidth and coverage constraints while minimizing video camera latency and expensive components. Our facilities server design allows for long-term data storage and for facilities to easily retrieve and analyze data.

Section 2: System Overview

MegaSense allows for smart devices to send relevant data to the FCS and receive commands and updates from the FCS. MegaSense’s communication network combines gateways and repeaters in a tree-like structure, as shown in Figure 1. The FCS has several processes running that process the packets it receives and store the relevant information in a convenient format for facilities analysis.

Figure 1: *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. The FCS can send information and updates to the smart devices by sending packets down the tree of communication nodes.*

2.1: Communication Network

In building our network, we prioritize simplicity of design, low setup and maintenance costs and low latency transmission, in addition to meeting coverage and bandwidth constraints.

Modules and Topology: The communication network is comprised of the following modules: gateways, BLE repeaters (low maintenance and *low cost*), and smart devices (thermostats, motion detectors and video cameras). As seen in Figure 1, our communication network is built as a tree structure. The FCS is connected to multiple gateways, which in turn are connected to chains of repeaters that are connected to sensors in classrooms. The IDs of the network nodes encode their location in the tree hierarchy (Section 3.2). We provide an algorithm to ensure the tree topology fully covers the MIT campus in Section 3.1, and analyze the expected bandwidth usage in Section 4.4 to show MegaSense satisfies facilities constraints.

Messages and Routing: Because the tree structure ensures that there is a unique path between the FCS and each smart device, and the component IDs locate them within the tree hierarchy, routing from the FCS to any smart device becomes *very simple*. The tree structure has sufficient bandwidth to support live streaming of all cameras at any time, *simplifying* the implementation of crisis mode. Additionally, we ensure the tree structure is fairly shallow, resulting in *low latency* for crisis mode. Because our topology has no redundancies to keep the design *simple and low-cost*, we introduce a system of ack messages for video cameras that prevents data loss in the event of failures or FCS maintenance.

2.2: FCS

The FCS has 2 primary functions: parse and store the data received from the smart devices and send updates and commands to the smart devices. The smart devices transmit relevant data to the FCS in packets sent through the network. The FCS receives packets from different types of sensors on different ports, and has different processes (Section 3.5) running on each relevant port that are specific to the type of received data. Information is stored in a format that is easily retrievable and amenable to analysis by facilities workers (a file system format for video data, and a database for temperature and motion data).

Section 3: System Design

3.1 Network Topology

The network topology allows sensors to communicate with gateways using intermediate repeaters. We provide generalizable rules for our topology to ensure full coverage on any campus. For each two adjacent classrooms, we place one BLE repeater so that the motion detectors and thermostats in both classrooms 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, well within the size of a classroom), forming a chain of repeaters to the gateway. We place the gateways in "central" locations (for instance in the middle of a long corridor), generally 1-2 per floor (depending on the layout and size of the floors). We also ensure that each chain is responsible for at most one video camera; if this is not true, we add another chain of repeaters connecting the video camera to the gateway, which is done to ensure the video camera has sufficient bandwidth available. Figure 2 demonstrates how a typical hallway layout would look.

Figure 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.*

We impose the condition that any chain of repeaters from a gateway will be at most 7 deep (i.e. any smart device is at most 6 hops away from a gateway). In general, since repeaters have a 30 ft radius, a chain of 6 repeaters can span around 180 ft., and placing the gateway in the center gives a 360 ft. diameter, which we believe is sufficient for an average-case hallway. If a corridor is too long for this condition to hold or a building has a weird shape, we simply add another gateway on the same floor to ensure this condition holds true, although this should happen rarely.

We analyze the bandwidth usage of such a network in Section 4.3 and show our network is able to handle video cameras continuously transmitting a frame per second. The shallowness of our tree ensures that live viewing in crisis mode meets latency requirements (Section 4.4). The tree-like structure of our network topology makes routing easy. Continuous live transmission of video camera data is low-latency and means simple handling of crisis mode.

3.2 Network Discovery

Crucial to our network discovery and routing is component IDs, which we describe below.

Component IDs: Each component of the network (FCS, gateways, repeaters, smart devices) 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 shown in Figure 3.

A device's ID determines the path from the FCS to the given device. For a given ID *x*, *x* with its last non-zero chunk removed is the ID of *x*'s parent. By repeatedly removing the last non-zero chunk from a given ID, we know the IDs of all devices along the path from a given device to the FCS. This makes it very simple to route messages from the FCS to any device and vice versa. In this scheme, the FCS has an address of all zeros.

For example, if the FCS wanted to send a message to the device with ID equal to that shown in Figure 3, it would forward the message to the gateway with ID equal to C_1: `0b0101001010001`. The gateway would then forward it to the repeater C_1C_2, which would forward it to $C_1C_2C_3$, and so on until it reached $C_1C_2C_3C_4C_5C_6$. This device would know not to forward the message any further because $C_7 = C_8 = 0$.

Figure 3: *An example 48-bit component ID showing how it is split into eight chunks. Chunk C_1 is 13 bits while all other chunks are 5 bits. There is one gateway (specified by C1) and four repeaters (C2 through C5) in between the FCS and the device specified by this ID.*

On startup: The FCS has knowledge of the entire network topology and all component IDs, and stores this in persistent memory. All other nodes in the network only know their own ID upon startup, and they can compute their parent ID by removing the last non-zero chunk from their own ID. The child always initiates the connection with the parent after computing its parent ID and attempts to send a message to this ID. The parent, upon receiving this message, acknowledges it, thus establishing the connection. Upon connecting with the parent, each device will query the FCS asking what the IDs of its children are. This is essential for detecting and fixing incorrectly set IDs as described in Section 3.4.1.

3.3 Routing and Communication Protocol

Once the network discovery process has occurred and the nodes know their parent and children in the tree, they communicate using packets with 64-bit headers and 20 bytes of data.

3.3.1 Routing

Packet headers: The first 2 bits of the header designate the origin of the data: FCS, gateway, repeater, or smart sensor. Thus any node can determine which direction (towards parent or towards child) to route a packet. If the FCS is sending the data, the next 48 bits contain the ID of the destination; otherwise, they contain the sender ID. This leaves 14 bits in the header for optional metadata that is packet-specific.

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, the node sends the packet to its parent, with gateways transmitting to the relevant port. 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.

3.3.2 Data Transmission From Sensors

Because BLE is unreliable, we send more information than strictly necessary to ensure timely delivery of information (with high probability). We do not to use TCP (or other reliable transport) because it introduces extra latency and complication. In doing so, we make a tradeoff in favor of simplicity and better latency in sacrifice of extreme reliability, which we feel is unneeded. For each sensor we specify a procedure for packet creation.

Motion + Temperature Sensor: Every minute, the motion detector runs `get_time()` and the temperature sensor calls `get_temp()`. The packets sent by these sensors include the header, the current timestamp, and the sensor reading. Because packets can be lost and these packets consume very little bandwidth, we send data more frequently that strictly necessary: every minute instead of every five. 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.

In a separate process, the motion detector runs `get_time()` every second, and checks whether the returned timestamp is within 20 minutes of of the current time. If this is true, it runs `turn_lights_on()`, otherwise it runs `turn_lights_off()`. Thus the motion detector self-regulates the lights in the room (without communicating with the FCS).

Video Camera: Every second the camera runs `get_latest_frame()` and gets a 28 kilobyte frame. This frame is broken up into a predetermined grid of ~2400 rectangular chunks of 12 bytes each. For each chunk, the camera sends a packet with the chunk number (12 bits) in the header metadata. The packet data contains the timestamp (4 bytes), frame number (4 bytes) and bytes of the chunk (12 bytes). In this manner, a single frame takes 2400 packets to send fully. Note that this system transmits video camera data continuously to the server and requires no modification for crisis mode, since the FCS already receives live footage from all cameras. Bandwidth analysis in Section 4.3 shows that our network can handle this traffic. Furthermore, the latency of transmission is quite low (around 0.7 seconds, Section 4.4), which is important for crisis mode. Thus this design achieves simplicity, by always being prepared for crisis mode, while also having low latency!

Ack System: We implement an *ack system* for acknowledgement of video camera frames to ensure no important information gets lost during component failures. When a gateway receives a series of packets corresponding to a frame, the gateway examines the frame chunk numbers and waits until it receives all packets for a frame before forwarding the frame to the FCS. Once this happens, it sends an acknowledgement packet to the video camera containing the frame id. Upon receiving an acknowledgement message, the video camera runs `delete_frame(frame_id)`. If there are any old frames that have not been acknowledged that are less than 8 hours old, the camera transmits the most recent unacknowledged frame along with its current frame (thus using double its typical bandwidth). Our network can handle this extra bandwidth consumption (worst-case scenario, Section 4.3). Every hour, the camera deletes all frames more than 8 hours old. This acknowledgement system is useful for dealing with gateway failures or FCS maintenance (Section 3.4).

3.4 Detecting and Dealing with Failures

3.4.1 Network failure

To detect failures in the communication network, we use a **system heartbeat**. The FCS sends out packets every 5 minutes to all gateways, with the header set to a special 64-bit integer (known by all components) that signifies heartbeat. Every gateaway upon receiving this packet sends back an ack to the FCS. For every gateway the FCS doesn't receive an ack from, it alerts facilities staff who then fix the gateway. The gateways propagate this heartbeat message to their children, which recursively propagate the message to all nodes (but do not respond to the FCS). Thus, every 5 minutes, all nodes know whether they can communicate with the FCS. In the event of a gateway failure, the video acknowledgement system in Section 3.3.2 ensures no valuable video data is lost (more detailed evaluation in Section 4.5).

Self-regulation mode: If a temperature or motion sensor does not receive a heartbeat packet for 10 minutes, they'll go into self-regulation mode by signalling to their parent repeater that self-regulation mode is necessary. The parent repeater will function as normal, but also store the motion data it receives and then perform similarly to the FCS by raising or lowering temperature according to motion detected in the past 2 hours.

3.4.2 Incorrectly set smart sensor IDs

MegaSense uses device IDs to describe network topology and routing. A smart sensor with an incorrectly set ID--which we will call a *bad sensor*--can't send or receive information from the FCS. MegaSense includes a software-based mechanism that can differentiate incorrectly set IDs from hardware errors, and resolves most of these errors without any human intervention.

For any device in the MegaSense network, the prefix of the device's ID is the device's parent's ID. Upon startup, bad sensor will send to an invalid parent ID and its message will be lost, thus never establishing a connection with its parent.The bad sensor will know something is wrong because it will never receive any heartbeats from the FCS. Note that this is distinguishable from the event when a parent device encounters hardware failure, because a bad sensor will never receive any heartbeats, while a functioning sensor with a broken parent will have received some heartbeat messages before the parent failed.

The bad sensor will detect something is wrong and broadcast an ORPHAN message, which contains the bad sensor's current (incorrect) ID and the sensor type. All repeaters in range of the bad sensor will receive the message and infer whether the orphan belongs to them. This is possible because each device *d* is told at startup the IDs and types of its children (Section 3.2). By comparing this against the IDs and types of the devices that have actually communicated with *d*, *d* can infer whether any of its children are missing. If *d* receives an ORPHAN message and knows its child c_i is missing, *d* responds to the ORPHAN message with *d's* ID. The bad sensor now knows its parent is *d*. In addition, *d* stores a mapping between the bad sensor's (incorrect) ID and c_i's (original) ID. When the FCS sends a message destined for c_i's ID, *d* will re-write the message to contain the bad sensor's ID, and pass it onto the bad sensor. When the bad sensor sends a message to the FCS, *d* will rewrite the incorrect ID to c_i's ID. With this NAT-style procedure, communication from the bad sensor to the FCS and vice versa are preserved.

There are a few corner cases that this bad sensor procedure cannot fully resolve, such as two sensors of the same type in the range of the same repeaters both having incorrectly set IDs. In this case, the repeater cannot distinguish between the bad sensors. However, the FCS will detect a lack of communication from these bad sensors and they will be dealt with as hardware failures (by alerting facilities staff). Assuming that the probability of a sensor ID being set incorrectly is 5%, and each repeater is connected to all three sensor types (the worst case scenario), there is a $1 - (1-0.05^2)^3 \approx 0.7\%$ chance of this occurring in a given repeater--a rare situation.

3.4.3 Dealing with faulty smart devices

If a motion sensor consistently returns no motion or a temperature sensor returns strange temperature readings, facilities can run anomaly detection scripts on the stored data in the FCS that will alert them to broken devices.

3.5 FCS

3.5.1 FCS Data Storage

The FCS will use a relational database to store temperature and motion data, and a file hierarchy for camera data. A standard relational database such as PostgreSQL will suffice for storing the temperature and motion data. The temperature will go in one table, while the motion will go in a separate table. Each of these tables will have four columns: an auto-incrementing, unique ID column (to help facilitate with joins and other queries), a timestamp for when the sensor reading was sent, the 48-bit ID of the sensor that sent the reading, and the value of the reading itself.

We chose to use a database for storing temperature and motion data because this data can be 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.

In contrast, relational databases don't handle video data well; they excel with numeric data and strings, not the binary information in video frames. As a result, we store videos directly in the filesystem, using folders to organize them. In a root folder named `videos`, there is one folder for each camera, each named after the camera's device ID. Within each of those folders, there is a H.264-encoded .mp4 file for every hour of camera footage. This way, for a given camera, an administrator can easily view the footage across the past week. By splitting the footage per hour, each video is kept fairly small, allowing for easy export of selective time ranges of video.

3.5.2 FCS Processing Unit

MegaSense runs five processes on the FCS, which are detailed below. Each process is run as a daemon with `upstart` which uses `fork()` to start these processes upon FCS startup.

Temperature and Motion Data: This 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 query the database for the last time we saw motion in the room. If there are multiple motion sensors in a room giving different readings, we assume there has been motion. If we detect 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 (by sending a packet to the relevant temperature sensor).

Video Frames: The second process records incoming video frames. For each camera, the FCS allocates a circular buffer of 201.6MB, enough to store two hours of uncompressed video footage. Whenever a video frame comes in, the FCS appends it to the proper buffer. Since each video frame takes ~2400 packets, we have to piece together the packets to make a frame using information in the packet headers. If the buffer is more than half full, this process will spawn a thread that will compress the first half of the buffer and write it to disk in the location described in Section 3.5.1. This child thread will clear the first half of the buffer as it compresses the frames.

Deletion: The third process deletes old data by running periodic commands to delete database entries over a year old and video files with timestamps over a week old. This more than meets the facilities constraints for data storage.

Crisis Thread: 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. An authenticated web portal allows facilities staff to inspect the buffer (and therefore view the live footage) within the FCS video-listening process of any desired camera.

Updates: Unlike the other processes, the update process doesn't constantly run on the FCS. The FCS administrator executes the update process through the command line when firmware needs updating. This process propagates a firmware update through the entire network by nodes recursively passing the update to their children. The FCS is in charge of packetizing the update so smart devices can assemble the entire binary.

When a device receives a packet corresponding to a firmware update (as indicated in the packet header), it first sends the update to all of its children. It then waits for all children to send an UPDATE FINISHED message before attempting to update itself. After all children of a device finish updating, the device checks if its device type matches the type in the firmware update. If not, it sends an UPDATE FINISHED message to its parent to signal it has no update work left to do. Otherwise, the device verifies it has all packets necessary to perform the update--some may have been lost due to BLE unreliability. The device re-requests all missing packets from the FCS, and then applies the new firmware by calling `update(binary)` with the newly received binary (after assembling it from the packets). Upon finish, the device sends an UPDATE FINISHED message to its parent.

3.5.3 FCS Maintenance

Before the FCS undergoes an update, it sends out a message to the gateways to hold all the packets they've received. Because the gateways have ample storage (32 GB) and each gateway only has 4 video cameras (that dominate bandwidth usage), they will be able to store such packets for the period of time that the FCS is down (estimated around 30 minutes). Upon FCS restart, the gateway will transmit all the packets it has received in the interim. Transmission is very fast through this wired network, so there won't be much delay for the current data. Furthermore, since temperature sensors and motion detectors won't receive heartbeat packets from the FCS, they'll go into self-regulation mode, as described in Section 3.4.2.

Section 4: Evaluation

4.0 Use Cases

Inconsistent Readings: If 2 motion detectors give inconsistent readings, we assume the room has people in it. Since one of the motion detectors will turn on the lights, and we explicitly deal with this when setting temperature, both lights and temperature will be set as if there is someone in the room.

Crisis Mode: Because our system continuously transmits live video, we do not need to do anything special for crisis mode. As we show in Section 4.4, the FCS receives video camera frames within 1 second, well within the facilities constraint of 5 seconds in crisis mode. Thus our system meets its design goal of low-latency video transmission and simplicity in dealing with different modes of operation.

Server Maintenance: The protocol specified in Section 3.5.3 ensures that when the FCS is down that the gateways store all of the information they receive and send it after the update ends. Thus no important information is lost when the FCS is updating. The self-regulation mode in Section 3.4.2 ensures that the temperature can still be appropriately set in rooms when the FCS is down or if there is a gateway failure.

Software update: The protocol specified in Section 3.5.2 ensures that all smart devices receive all necessary packets for an update and can ask the FCS for missing packets. Suppose the software update is of size M, and that it takes time f(M) for each component to apply it. At each depth, the components can perform the updates simultaneously, so it takes 7*f(M) time for the update to be applied everywhere. For a software update of reasonable size, say 1MB, the update can be sent to the sensors within a few seconds, and if it takes 1 minute for the update to be applied, then the entire process takes around 7 minutes.

4.1 System Startup

When a network node comes online, it needs to calculate its parent ID from its own ID, and then connect to its parent (Section 3.2). In the typical case where the IDs are correct, this will take 0.2 seconds due to BLE latency. In the worst-case of incorrect IDs, the process in 3.4.1 should take at most 1.4 seconds, since we must wait for the FCS to send the IDs of the node's children for resolving the ORPHAN messages (network RTT is 1.4 seconds, Section 4.4). Given that all sensors have been installed and powered, the network is ready to transmit information within 2 seconds (conservative estimate). Our simple ID-based routing protocol allows for this quick startup time, as opposed to a more complex, dynamic routing protocol which might take some time to initially discover paths and converge.

4.2 Layout and Cost

Because our network topology does not have a specific shape or layout, our design can handle buildings of arbitrary topologies. This is better than designs that use a “tiling” network topology, which would be inefficient for buildings that do not match the tiles perfectly.

At MIT, there are 15000 classrooms, and each repeater is responsible for 2 classrooms, so our system uses 7500 BLE repeaters in total. Each gateway is responsible for 50 classrooms (since there are 4 chains of 7 repeaters, so around 28 total repeaters each responsible for 2 classrooms per gateway), so our system uses 300 gateways. Repeaters never need to be replaced, while gateways are replaced once every three years (on average).

Component	Number	Cost per component	Maintenance cost per component per year
Gateways	300	500	166
BLE repeaters	7500	10	0

The initial cost of the system is $225,000, and the maintenance cost per year is $49,800. Other designs with more redundancies might use more gateways and repeaters, so they incur a greater initial and maintenance cost than our system. Because we use the cheap BLE repeaters that require no maintenance and are low cost, and try to minimize number of gateways, we prioritize low cost in our system over extreme reliability.

4.3 Bandwidth Usage

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 BLE repeater has a bandwidth of 2Mbits/sec. Each video camera sends 28KB per second. In our design, we ensure that each chain of repeaters to a gateway serves at most one video camera, which uses a bandwidth of 28*8 = 224KBits/sec. Each smart device sends at most one packet of 20 bytes per second, and each chain serves at most 25 smart devices, using a bandwidth of 20*8*25=4KBits/sec. So each chain will serve at most 228KBits/sec, assuming all this data is passing through the same connection. Our design ensures that each repeater has at most 3 active connections, so each connection will have 2MBits/sec / 3 = 667KBits/sec of bandwidth, which is more than what we require. Gateways have a bandwidth of 16MBits/sec, and each gateway has around 4 active connections, so each chain has a bandwidth of 4MBits/sec, which again is more than what we require.

There are two cases that would require our system to use more bandwidth than we normally do. One of them arises when a gateway fails. In this case, the video cameras starts storing frames and sends old frames back to the FCS when the gateway recovers. This is done at a rate of 1 (old) frame per second, so it introduces an additional 224KBits/sec. In this case, a chain would have to accommodate a maximum of 452KBits/sec, which is still handled by our 667KBits/sec of bandwidth allowed by the repeaters. The other case occurs during software updates. In this case, none of the sensors are sending data anyway, and a software update of, say, 1MB can be sent to the components in less than two seconds. Therefore our system has sufficient bandwidth for all our use cases at all times. Our system has low branching factor (maximum of 3 at the repeater level), so it can use a larger bandwidth compared to a system with many branches. However a drawback of this design is that it might be hard to scale significantly since the branching factor must be low to accommodate bandwidth needs. To scale further, we would need to add more gateways, which might be very expensive. In our situation, we trade-off simplicity for high cost of scalability (since the cost of scalability is reasonable for full coverage of MIT's campus).

4.4 Latency and Performance Analysis

Our network topology is a tree of gateways, repeaters and sensors that has depth at most 7. Since each hop introduces a latency of 100ms, there is a latency of at most 0.7s when sending data from the sensors to the FCS and the RTT is 1.4s. The time required for data collected by a smart device to be transmitted to the FCS is at most 0.7s + (0.5MBits)/ (2MBits per sec) = 0.95s, since we send at most 0.5MBits of data at once (Section 4.3). This is sufficient for all of our needs, including sending video frames in crisis mode. The maximum depth of 7 is a good tradeoff between latency and bandwidth constraints. In our design, because routing is fixed, there is no overhead for route discovery, as in a system with dynamic routing.

4.5 Reliability in the face of failures

Response to component failures: Our system design uses gateways, BLE repeaters and sensors. Since the BLE repeaters and sensors never fail, the only possible point of failures are at the gateways. The system heartbeat (Section 3.4.1) ensures that failed gateways are detected within 5 minutes and notifies facilities to replace and configure the gateway, which takes 1 hour. In the worst case, a gateway failure will be resolved in 2 hours (if we assume a facilities member is always on call, which is typical for MIT), and at best within an hour, which we believe to be reasonable. Any single gateway is responsible for at most 3 cameras and 50 classrooms, so during those 2 hours, we will be unable to send or receive data from these sensors. After the gateway has recovered, the data from the motion sensors and thermostats will be lost, which is a tradeoff for the simplicity of MegaSense. However, according to the protocol described in 3.3.2 the video cameras will have stored the frames that were captured while the gateway was down since they won't have received acknowledgements for those frames, and these frames will be sent to the FCS and not be lost.

Although the reliability would be improved by having redundancies (for gateways), we believe this improvement is not worth the additional cost. Each gateway has a lifespan of one to five years, so on average they fail every 3 years, so the downtime is only 2 hours every 3 years per gateway. Furthermore adding another gateway might complicate the routing protocol (since we have to redirect traffic to the new gateway in the event of failure). Thus we make a tradeoff favoring simplicity and lower cost for slightly lesser reliability (since we believe motion and temperature data loss isn't that significant).

Packet losses: The BLE communication protocol isn't perfectly reliable; it drops about 0.0001% of all packets. For a single packet containing data from a motion sensor or thermostat, the probability it is dropped is extremely low; even if dropped, the loss of a single temperature or motion reading is insignificant and we send redundant information in order to meet facilities requirements with high probability. For a video frame, 2400 packets must be sent. Since the frames are compressed, the loss of a single packet would corrupt the entire frame. The probability that none of the 2400 packets are lost is $(1\text{-}0.0001\%)^{2400}$ = 99.8%, so there is only a 0.2% chance we lose a frame. In this case, the gateway will not acknowledge the frame (Section 3.3.2), and the frame will be retransmitted. Thus frames are very rarely dropped, and when they are, the video acknowledgement protocol recovers almost all dropped frames with low bandwidth overhead.

4.6 FCS Storage and Facilities Usage

Video camera footage takes up the bulk of storage in the FCS.Temperature and motion data is negligible when compared to the video footage. There are 1000 cameras in total, and we only store one week's footage. Even without compression, this takes 1000 * 28 * (60 * 60 * 24 * 7) KB = 16TB, well within the FCS' 100TB of storage. If needed, the FCS has the capacity to store $(100*10^9)$ / (1000) / (28) / (60 * 60 * 24) = 41 days of footage from all of the cameras.

Facilities can easily access all of these data. Since the temperature and motion data is stored in a relational database, Facilities can send in various queries to get the information about the classrooms. For instance, to access all data from a particular thermostat, they can get all records matching the ID of that thermostat. To determine which rooms are currently in use, they can ask for all records representing motion data from the last 5 minutes. We store video footage in chunks of one hour each for each camera, labeled by timestamp, so that Facilities can easily access footage from a particular time. The videos are organized into a simple file-system, organized by the camera ID and then by time.

4.7 Scalability

MegaSense is generally very scalable. When MIT adds a new building to its campus, we simply add more gateways and use the same network topology as before. The main limiting factor is the number of gateway IDs in our current naming scheme. We have 13 bits of addresses available for gateways, so we can have at most 2^{13}-1=8191 gateways. Our design for MIT's campus uses 300 gateways, so our system is scalable to 25x the size of MIT. However we note that since the branching factor for repeaters is low (due to bandwidth concerns), scalability is potentially higher cost than other designs since we have to add more gateways. We believe this tradeoff is worth it, as for a campus of MIT's size, our design has relatively manageable cost.

In response to the addition of more types and higher densities of sensors, we can attach them to existing repeaters/gateways in the network. Currently, gateways have four and repeaters have up to seven children, and chunks C_2-C_8 in our naming scheme accomodate for up to 2^5-1=31 children, so this isn't a concern.

4.8 Security Issues

To make sure that only facilities staff members can log onto the FCS, we use a password based authentication scheme. Each approved facilities member has an account on the facilities server with their MIT ID as their username and a self-set password upon account creation. For additional security, we can also require facilities members to use 2-factor authentication to login to the server. The FCS also likely has a several server administrators that might require remote access (through

SSH). A dedicated admin adds new SSH public keys to the list of authorized users as necessary, with the responsibility to only allow legitimate uses to request SSH access.

Because of their low computational capability, smart devices are not capable of encrypting data. This poses a major problem since an attacker sniffing network traffic can read any of the data coming from the smart devices. As a result, sensitive video camera data is available for the attacker's viewing, which poses a severe privacy risk for people in video camera footage. Additionally, because the FCS does not validate the data that is receives and it is unencrypted, a malicious adversary could spoof fake video camera data and send it to the FCS. Thus an attacker could 'loop' a video camera to send unchanging images, while in reality, the attacker could be carrying out a robbery (or other malicious actions), that would not be visible to facilities. This is potentially a large security flaw.

Section 5: 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. The topology of our network ensures that there is sufficient bandwidth for video cameras to transmit live and with low-latency (due to the small height of the tree). Although there is no redundancies in place, the system of acks ensures that the probability of significant data loss from video cameras is low. The FCS processes the incoming data and stores it to meet facilities needs. The FCS also is able to distribute updates and issue commands to the smart devices. An system heartbeat ensures that failures are promptly detected and self-regulation mode ensures temperatures are appropriately set even when the FCS is undergoing maintenance. Overall, our system allows facilities to effectively interface with the smart devices.

Section 6: Author Contributions

The authors jointly brainstormed and came up with the overall system design. Special attention should be given to Philip Sun for detailing the FCS, Uma Roy for formulating how to deal with failures and Justin Lim for evaluation of the design. The authors jointly wrote the report, with Justin primarily writing the evaluation section and Uma and Phil contributing equally to the other aspects of the report.

Section 7: Acknowledgements

We'd like to thank Lynda Tang, Howard Shrobe and Juergen Schoenstein, as well as Katrina LaCurts and the entire 6.033 staff for running this extremely well-organized and informative class.