1 Introduction

Around campus, you'll find many different mechanical or electronic devices performing small tasks:

- Thermostats monitor room temperatures and adjust them as needed.
- Motion detectors detect whether people are using a room, and turn lights off when a room is not in use (and back on when it is).
- Video cameras capture video of traffic in classrooms or other common areas.

There are a variety of situations in which it would be useful for MIT Facilities staff to be able to monitor these devices:

- To collect data for long-term projects. For example, if Facilities can use a camera feed to figure out how many people use a particular space over some period of days or months, they can better allocate space and plan for new buildings.
- To intelligently adjust the temperature in rooms depending on whether they're in use, which can lead to significant cost savings for the Institute.
- To detect mechanical failures so that they can send someone to fix the problem in a timely manner, rather than waiting until it gets reported by an employee.

MIT is looking to move some of these devices towards "smart" functionality: the smart devices will be able to transmit data back to a server that Facilities owns. Facilities will be able to use that data to improve the campus environment.

Your primary job in this project will be to design a system that supports these smart devices, and thus enables MIT Facilities to monitor them, collect data, and respond to failures or events. Your system will be comprised of a network of nodes that communicate with the above smart devices, and transmit data back to a centralized server.

In designing this system, you will find that there are many constraints. The smart devices, the communication network, and the capabilities of the server all place constraints on the amount of data that can be transmitted and processed at once.

2 Existing Infrastructure

2.1 Smart Devices

Although there are many electronic and mechanical devices around MIT's main campus, Facilities is only interested in upgrading three types of devices at this time: thermostats, motion detectors, and video cameras. Your system will need to transmit some type of data from each smart device back to a centralized server at Facilities, known as the FCS. Details on the communication between the smart devices and the FCS are in §2.2.

Each smart device has a small Bluetooth radio built in. This radio allows the smart device to transmit data up to 30 feet at up to 2Mbit/sec using the Bluetooth Low Energy (BLE) Communications

Command	Action
`get_temp()`	Returns the current temperature.
`set_temp(t)`	Sets the desired temperature for this thermostat to `t`.
`update(binary)`	Initiates a software update.

Table 1: Thermostat Software Abstractions

Protocol (see §2.2.5). Your design will specify what data is transmitted by each smart device, and when.

The sections below detail specific functionality provided by each type of smart device. In addition, assume that all smart devices are able to retrieve a 32-bit timestamp reflecting the current time.

2.1.1 Thermostats

Thermostats monitor room temperatures and adjust temperatures as needed. There are roughly 15,000 thermostats on MIT's main campus: on average, there is one per room, though some rooms have more than one thermostat in them.

Each thermostat has a single desired temperature, stored as 32-bit floating-point value. The thermostat—which is connected to the larger HVAC system for the room—turns heating or cooling devices on or off as needed to achieve that desired temperature. Thus, thermostats have both a passive and active component: they passively monitor temperature in the room, and actively set the temperature by turning various components on or off. For the purposes of this project, you do not need to worry about the thermostat's connection to the HVAC system, or how the thermostat itself is powered (via battery or otherwise).

Facilities uses temperature data from thermostats to detect problems: If the temperature is consistently rising (or falling), there may be a problem with the thermostat itself or the HVAC system as a whole. On the FCS, the temperature reading for each thermostat should be no more than five minutes old.

Thermostats can receive commands from Facilities that explicitly set their desired temperature, but do *not* have an interface for people in the space to set the temperature.

To facilitate the above actions, the thermostat software provides the function calls described in Table 1.

2.1.2 Motion Detectors

Motion detectors use infrared radiation to detect whether people are using a room. Similar to the thermostats, there are 15,000 motion detectors on MIT's main campus: roughly one per room, though some rooms will have multiple motion detectors.

Each motion detector is directly connected to some component of the lighting system in a room, and can automate the process of turning lights on and off depending on whether a room is in use. Each motion detector stores the last time that motion was detected in the room (as a 32-bit

Command	Action
`get_time()`	Returns the last time that motion was detected in the room.
`turn_lights_off()`	Turns the lights off
`turn_lights_on()`	Turns the lights on
`get_light_status()`	Returns a bit representing the status of the lights (1=on)
`update(binary)`	Initiates a software update.

Table 2: Motion Detector Software Abstractions

timestamp). For the purposes of this project, you do not need to worry about the motion detector's connection to the lighting system, or how the motion detector itself is powered.

Facilities would like to know how long it has been since the room was last in use. The FCS uses this data to set the temperature in rooms: if a room has not been in use for awhile, Facilities will send a command to the relevant thermostat(s) to lower the temperature (see §2.3).

To facilitate the above actions, the motion-detector software provides the function calls described in Table 2.

2.1.3 Video Cameras

Video camera capture video of common areas on campus. These cameras capture five frames per second; each frame is 28 kbytes. Unlike thermostats and motion detectors, most classrooms do *not* have video cameras in them. The cameras are largely in hallways and common areas, and there are about 1,000 on MIT's main campus. There is an average of 100 feet between any two cameras, though the variance is high because of the layout of various buildings (in particular, it's certainly possible that two or more cameras could be quite close together).

Facilities uses the data from the video-camera feeds for a variety of things:

- To collect long-term data on the number of people who use different areas of campus at different times. Facilities uses this data for a variety of projects (e.g., optimizing pedestrian flow in various places). The algorithms that Facilities uses to do people-counting run on the FCS (see §2.3 for more details).
- To capture potentially-illegal activity on certain areas of campus. After a crime has occurred, Facilities wants to be able to review the footage; to do this, they need to store one week worth of data from each of the cameras.

In both cases, Facilities requires your system to provide them with at least one frame per second; however, having access to additional frames can offer improvements (e.g., more accurate counting).

Under normal operation, video frames from the cameras should be available at the FCS within five hours. However, when in **crisis mode**, Facilities will need to receive data from certain cameras in real-time. See §3.2 for more details.

The cameras themselves are more powerful than thermostats or motion detectors. They are capable of buffering 4 GB of frame data. By default, they will store all five frames per second that the camera records (this works out to about eight hours of video). However, the cameras can be set

Command	Action
`start_filter(fps)`	Puts the camera in smart-filtering mode. It will store `fps` frames per second, instead of five frames per second.
`stop_filter()`	Stops smart filtering. The camera goes back to storing five frames per second.
`get_frame(frame_id)`	Returns the frame with the specified ID. If this frame is not in the buffer, returns `NULL`.
`get_latest_frame()`	Gets the latest frame from the buffer. This command returns the frame as well as an associated 32-bit frame ID.
`is_equal(frame_id_1, frame_id_2)`	Returns True if the two frames depict effectively the same image. Returns False otherwise, including if one (or both) frames are not in the buffer.
`delete_frame(frame_id)`	Deletes the frame with the specified ID. If this frame is not in the buffer, does nothing.
`update(binary)`	Initiates a software update.

Table 3: Camera Software Abstractions

to automatically filter before buffering, and capture fewer than five frames per second, thereby storing frames from more than eight hours ago.

Each frame has an ID stored as a 32-bit integer. The first frame ID is 0, and increases with each frame. When the last frame ID is reached ($ID = 2^{32} - 1$), the frame IDs roll over and start again at zero.

Cameras are capable of determining whether any two frames depict essentially the same image (this might happen at night, in common areas that are not in use), and deleting specific frames from their buffer.

You do not need to worry about how the cameras are powered.

To facilitate the above actions, the camera software provides the function calls described in Table 3.

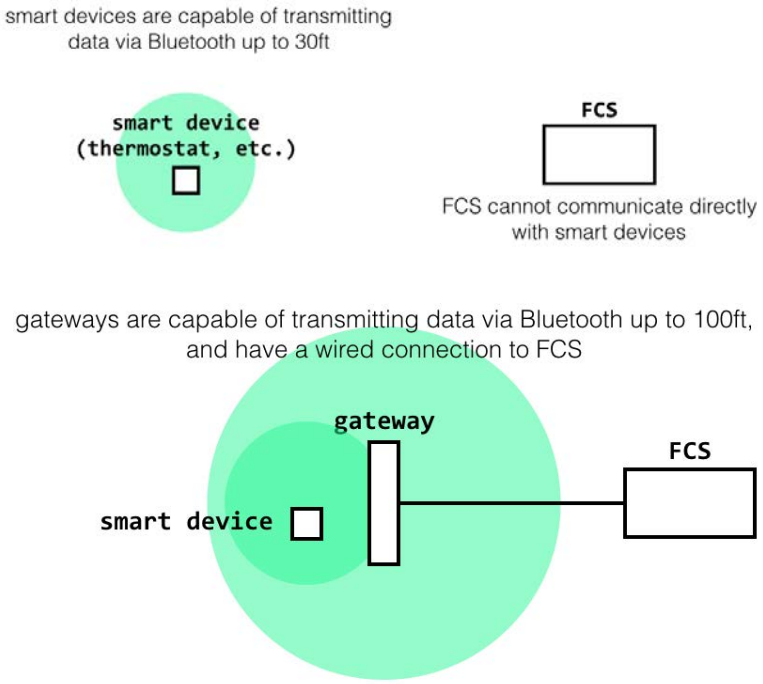
2.2 Building the Network

2.2.1 Smart Devices and Gateways

In addition to having a small transmission range, smart devices themselves cannot connect directly to the Internet, as shown below.

Thus, to get data from a smart device to the FCS, you'll need some additional components.

Gateways are devices that *can* connect to the Internet, and so can communicate directly with the FCS. Gateways can also communicate with the smart devices using the Bluetooth Low Energy (BLE) Communications Protocol (§2.2.5); in this way, gateways act as a bridge between components that speak Bluetooth and the rest of the Internet. A smart device in range of a gateway could communicate with the FCS through that gateway, as shown below.



In addition to acting as a bridge between Bluetooth components and the rest of the Internet, gateways can send transmit data via Bluetooth with a range of 100 feet and have 32 GB of persistent storage. If a gateway has to transmit through a wall or floor, its range will decrease by about ten feet for each impediment (e.g., if the signal needs to pass through a single wall, it will have a transmission range of 90 feet).

Because they do so much, gateways cost a lot: five hundred dollars per gateway. Gateways have an average lifetime of one to five years. When they fail, the replacement process is somewhat involved; it takes an hour to swap the old gateway for a new, working one, and to configure the new one to work correctly.

Your design will specify where to place these gateways such that data can get from each smart device to the FCS. One possible solution is to place at least one gateway within range of every single smart device; smart devices would send data to their nearest gateway, which would then send data to FCS (as shown in the previous figure).

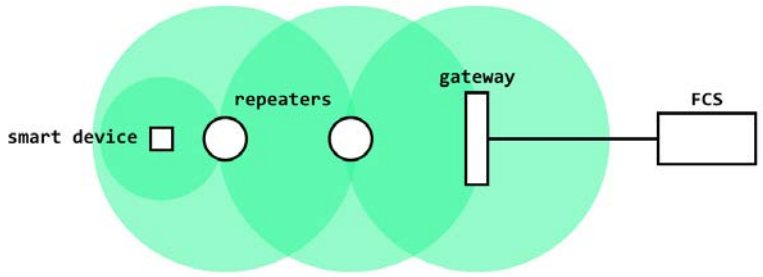
This design comes with a high monetary cost, among other things.

2.2.2 Repeaters

As an alternative to placing a gateway within range of every smart device, Facilities is allowing you utilize **repeaters**. The repeaters have Bluetooth radios, and so can communicate with smart devices as well as gateways. Repeaters allow you to effectively extend the range of smart devices, as shown below.

There are two different types of repeaters available to you:

- **BLE repeaters** are ultra-low-powered devices. They have no persistent storage and 4 MB of RAM. These repeaters can transmit data within a range of 30 feet; if their range is inter-



repeaters can effectively extend the range of a smart device
(or devices)

rupted by a wall or a floor, that range will decrease to 20 feet. BLE repeaters are extremely cheap (ten dollars per repeater) and draw very little power from their built-in battery. For the purposes of this project, you can assume that the batteries never need to be replaced.

- **BLE+ repeaters** also communicate using only the BLE Communications Protocol. However, they are slightly more powerful than BLE repeaters: they have 64 MB of persistent storage, and have a transmission range of 100 feet; as such, they also cost a bit more (fifty dollars per repeater), and draw more power.. Like gateways, if BLE+ repeaters transmit through a wall or floor, their range will decrease by about ten feet for each impediment.

  BLE+ repeaters have a battery life of around six months to one year. When their battery runs out, someone at Facilities needs to replace the battery. The battery replacement is fast: once a Facilities staff member is there, it only takes five minutes to replace the battery.

2.2.3 Node Discovery

Before any communication can happen, components of your system will need to "discover" each other.

Smart devices advertise their existence periodically using beacons, which broadcast their 48-bit identifier. Your system should specify how this ID is constructed. When Facilities installs a smart device, however, there is a small probability that they will set the ID incorrectly. Your system should be able to recover in this case. You can assume that, even if an ID is set incorrectly, all IDs will still be unique (unless you have designed an ID scheme that intentionally duplicates IDs).

To ensure a sufficient battery life, the smart devices are configured to broadcast these beacons once per second. A BLE repeater, BLE+ repeater, or gateway can discover any smart device within range by listening for these beacons.

On the other end of the network, the FCS has an IP address that is fixed and known to all of the gateways in the system. Similarly, each gateway has a unique IP address that is fixed and known to the FCS.

If your design requires smart devices to communicate with gateways via additional repeaters, you will need to specify a node-discovery process. Like the smart devices, you can assume that BLE repeaters and BLE+ repeaters have fixed, 48-bit identifier; your system should specify how those are set as well. This identifier is known to the repeater itself, but not to any other part of the system unless your design specifies otherwise. The repeaters and gateways are capable of

broadcasting information in the same way that the smart devices are, i.e., you could set them up to send out beacons. But you are also free to choose other ways to handle this process.

2.2.4 Routing

In order for your system to work, it will need to be able to route data from the smart devices to the FCS **and** from the FCS back to various smart devices. Your system design should specify how data gets to the correct place. Note, for instance, that Facilities does **not**, by default, have a way to route data directly to any smart device. In some cases, Facilities will also need to know that its data was received by the appropriate smart devices.

It's possible that some data will traverse multiple hops in your network (e.g., smart device → repeater#1 → repeater#2 → gateway). Assume that each hop will introduce a latency of 100ms. If your topology requires any component to disconnect from one component and reconnect to another mid-transmission, assume that the same latency will apply. (There may also be additional affects, depending on your system design.)

With what we've given you so far, it would be possible to set up an arbitrarily complicated "mesh network", where nodes attempt to connect to many other nodes, and where nodes can rapidly change the nodes to which they're connected. This will quickly bring up a few challenges:

- Routing in a mesh network is a complex problem in and of itself.
- Data will accumulate as it travels through the mesh. For instance, if two smart devices are each transmitting .5Mbit/sec of data to the same repeater, that repeater now has 1Mbit/sec of data to transmit somewhere else. This will likely lead to all sorts of "max flow" problems.
- The more hops a piece of data has to travel, the more latency it will incur.

It is **not** our intent to have you focus on complex mesh-networking problems in this project. We have not trained you to handle them; moreover, the complexity of mesh network will be a very difficult thing to justify for this project. There are a variety of good designs for this system that do not require such complexity.

That said, we are not limiting your routing algorithm or your network topology. If absolutely you cannot resist the pull of a large mesh network, go for it. But be prepared to justify the complexity to your audience. It would be wise to remember that good design is iterative: start with a simple design that works, and build up from there.

2.2.5 BLE (Wireless) Communications Protocol

Any data that goes between smart devices and repeaters, smart devices and gateways, or between one repeater and another, follows the BLE communications protocol.[1] For one component to send to another, the receiving component must first initiate a connection to the sender. Once a connection is established, the sender will be able to send packets to the receiver. The maximum packet size is 20 Bytes of data, plus a header; the maximum header size is 64 bits. How much data the sender can send depends on how many other connections are running through the receiver.

[1] The protocol that we've given you for this project is based on an actual BLE protocol, but is not exactly the same.

At most 8 connections can be active at a time on a BLE repeater, at most 16 connections can be active at a time on a BLE+ repeater, and at most 64 connections can be active at a time on a gateway. Beacons from smart devices do not count towards this connection limit.[2] If there are n connections active on a repeater (or gateway), each connection will get at most $1/n^{th}$ of the total bandwidth that that repeater (or gateway) is capable of. BLE repeaters are capable of 2Mbit/sec; BLE+ repeaters are capable of 4Mbit/sec; and gateways are capable of 16Mbit/sec.

The BLE Communications Protocol does **not**, by default, guarantee perfect reliability. The network drops about .0001% of all packets.

2.2.6 Wired Communications Protocol

The gateways can communicate with the FCS over wires, via a standard TCP protocol. For the purposes of this project, you can assume that TCP provides **perfect** reliability. Each gateway can communicate with the FCS at a rate of up to 1GB/sec, with data packets up to size 1500 bytes. The minimum round-trip-time between the gateways and the FCS is negligible, but that latency may increase depending on your system design (e.g., if queues grow).

In both the wired and wireless cases, it is your job to specify the format of the messages that traverse the network.

2.3 The FCS

Facilities has a single centralized machine—the FCS—to store its data, with 100 TB of storage. You can assume that the FCS has the specs of a standard modern server. The IP address of the FCS is fixed and known by all gateways in your system.

Any packets sent to the FCS will arrive in a buffer that is treated as a FIFO queue to be processed by the server's processing unit. Facilities uses the FCS to store data, as well as to perform some computations on the data; the processing unit—which you will design—takes care of sending data to storage as well as to code that performs those computations.

The processing unit is comprised of a pool of threads that run concurrently. Your system will specify some of the details of these threads.

2.3.1 The Main Thread

The primary thread of the processor is responsible for monitoring the queue of incoming packets and performing the following actions:

- Storing all video frames it receives—unless there is a compelling reason for it to discard some—along with a timestamp and the number of people in the frame. The system call `process_frame(frame)` will return that number.
- Deciding whether to adjust a thermostat in response to motion (or lack-of-motion) in a room. If a location has not been in use for the past two hours, the relevant thermostat(s) should be

[2] In terms of how this would work in practice: the beacons would be sent on a different frequency than the connection-oriented protocol.

set to ten degrees cooler than their current desired temperatures. If a location is newly in use, the relevant thermostat(s) should be set back to their desired temperatures.

This thread is **not** responsible for calling the code that looks for historical anomalies in temperature readings; that code is run as a separate background process. However, to work well, the anomaly-detection code needs to be able to access the last two weeks of temperature readings for a particular room. It should be able to retrieve a time-ordered list of these readings.

Your system should describe how the main thread of the processing unit completes these actions. You are welcome to have the main thread spawn child threads, via a system call such as `fork()`.

2.3.2 Crisis and Update Threads

Additional threads in the processing unit's thread pool deal with crisis mode and software updates.

When a Facilities staff member enables crisis mode on a particular room, a thread waiting on that event will be resumed, and call the function `enable_crisis(camera_ID)` for each camera in the room. When a Facilities staff member disables crisis mode on a room, a thread waiting on that event will be resumed, and call the function `disable_crisis(camera_ID)` for each camera in the room. Your system should specify the details of those two functions.

Similarly, when a Facilities staff member requests to perform a software update on all devices of type `device_type`, a thread waiting on that event will be resumed, and call the function `update(device_type, software_binary)`. Your system should specify the details of this function. This function should not return until it can guarantee that the update has been received by all smart devices.

2.3.3 Data Storage

You can assume that Facilities has already stored a mapping of each smart device's ID to a location at MIT. Locations are given as a building number and room (even hallways have "room" numbers). If you would like to extend this mapping, to store additional or di erent location information, you may; just describe how that process is done.

How the rest of the data is stored on the FCS is largely up to you. In some cases, you may store data as a series of files; in those cases, you should specify the filesystem structure. You do not need to specify details of the file format unless you're doing something creative (e.g., it's fine to say you store something "as a .txt file" rather than giving us the specifications of the .txt file format).

In some other cases, you may find a database more appropriate. In those cases, you can assume that you have access to a modern DBMS, but you should detail your database schema and relevant interactions with the database.

Regardless of how you store your data, you will need to support the goals that have been outlined in this project (e.g., that Facilities needs access to one week of historical data from cameras, that they need two weeks of data from thermostats, etc.).

3 Requirements

3.1 Components of Your Design

Using the infrastructure specified in the previous section, your design should specify the implementation and functionality of the following components:

- The network topology. You have, at your disposal, repeaters as well as gateways. Your design should describe a basic scheme for connecting these devices. Is every smart device in range of a gateway? Of multiple gateways? Is every repeater? Are some repeaters *not* in range of gateways? On average, how many smart devices is each repeater responsible for? Etc. Note that you also have the ability to create a naming scheme for the smart device (and repeater) IDs.
- The communication protocol. At a minimum, you should specify message formats, the circumstances under which messages are sent (periodically? in reaction to some event?), and any network mechanisms that you need to add to meet the requirements of the system. In particular, the network between the smart devices and the gateways is unreliable; you should specify any mechanisms you use to provide reliability, if you feel that they are needed.

  As part of this specification, you should detail how the smart device retrieves the data that it sends (what function does it use to retrieve it?), whether it performs any computations before sending, and what it does in reaction to any data that it may receive.
- The processing unit on the FCS. You should describe the implementation of this software in detail. Does the main thread spawn any child threads? What do they do? Do any threads need to hold locks on any piece of data? If so, how/when do they acquire those locks? Etc.
- The data storage mechanism on the FCS. How is data stored? Is it possible for your system to collect duplicate pieces of data? If so, how is that handled? Does your system guarantee the reception of *every* piece of data?
- What happens during certain failures. For example, both BLE+ repeaters and gateways can fail. What does your system do in those cases? Is it easy for Facilities to know that a component has failed?

The existing infrastructure already imposes some requirements on your system: you cannot exceed the storage or processing power of any component, nor the maximum speed of the networks involved.

You must also meet Facilities' requirements for data storage, as well as allow Facilities to send commands and push software updates to smart devices.

Moreover, your system must work at scale. The intended scale of the system is the main MIT campus. However, if your system performs well, other larger universities may be interested in adapting it.

3.2 Use-cases

As you design your system, you should consider its performance under the following use cases (as well as under normal operation). You will be required to address these use cases in your final report.

- **Inconsistent Readings:** Two motion detectors in the same room give inconsistent readings: one reports that the room is in-use, the other does not. What does your system do in this case?
- **Crisis Mode:** In general, data from the video cameras does not have to arrive at the FCS immediately. However, during a campus crisis, Facilities will need data from certain cameras immediately (within five seconds is acceptable, but the faster the better).

  The software running on the FCS that supports this operation allows Facilities to specify a room number on campus; video feeds from the cameras in that location should be sent to Facilities immediately. As the crisis occurs, it's possible that Facilities will update their location-of-interest (for example, if they are tracking an intruder down a hallway).

  A crisis situation does not obviate the need to eventually store data from all cameras. Even though some data may need to be prioritized and delivered immediately, data from all cameras should still be available to Facilities within five hours.
- **Server Maintenance:** Suppose the Facilities server is taken down for maintenance. When it comes back online, will your system still be able to deliver all relevant data? When it is offline, can your system replicate any of its functionality? In particular, can your system still intelligently set the temperature in rooms?
- **Software update:** Facilities has to push an important security update out to all video cameras. How quickly can your system make this happen? How can you guarantee that every video camera gets the update?

In addition to those specific scenarios, you should also think about what will happen to your system over time. Currently, Facilities is only upgrading devices in the main campus buildings. Suppose MIT wants to adapt your system to the rest of campus, or to new buildings that the Institute purchases. Could your system scale to meet this need? What about if a different university adopted your system? Similarly, suppose Facilities decided to upgrade a particular smart device, or support a new type of smart device. Could your system handle that?

3.3 Trade-offs

As you design, you will encounter many places where you have to trade off one aspect of your design for another. We've highlighted some below, although these are not the only tradeoffs.

- **Cost vs. complexity:** Facilities has not restricted your budget for this project; however, they would prefer to spend less money. How much does your design cost? Could you get similar functionality for less money?
- **Data timeliness vs. congestion:** In some scenarios, you may not be able to send all of the data you're collecting to the FCS at once, either because of the limit on the maximum

number of connections per repeater, or because the wireless network cannot handle that much traffic.

- **Reliability vs. Performance:** BLE repeaters effectively never fail, since their batteries never need replacing. BLE+ repeaters can fail occasionally, but are more powerful. Gateways can also fail, and are extremely powerful. How do you balance these issues?
- **Reliability vs. Overhead:** The wireless network is unreliable, dropping .0001% of all packets. Should you run an end-to-end reliability protocol, such as TCP, across this network? Should you create your own reliability mechanisms, which might be lighter-weight than TCP? Are there any times when you don't need perfect reliability?

6.033 Computer System Engineering
Spring 2018