IntelliNet: A Campus-Wide Internet of Things

Anelise Newman, Christopher Wang, Eric Luu, Giancarlo Delfin
{apnewman, czw, ericluu, gdolphin}@mit.edu
Adam Belay - Friday 2pm

May 8, 2018

Contents

1 Introduction

The MIT campus contains many devices such as thermostats, cameras, and motion detectors that monitor and control the state of MIT's buildings. Currently, these devices lack persistent storage and the ability to intelligently respond to real-time events. MIT Facilities wants to modernize this system by creating a network of "smart" devices connected to a central server (the FCS) that is capable of aggregating data, issuing commands, and responding to campus crises. This network will allow Facilities to automatically adjust the temperature of vacant rooms, collect crowd data for space allocation projects, detect the failure of individual devices, and perform other intelligent tasks.

To meet these goals, our proposed system, IntelliNet, collects video, temperature, and motion data generated by the smart devices through a network of Bluetooth nodes and Internet-compatible gateways. It then stores this data for later analysis and dispatches real-time responses to events on campus. IntelliNet provides an elegant and extensible framework for processing smart device data across a complicated campus environment, while achieving our design goals of simplicity, scalability, performance, and fault tolerance.

2 System Overview

IntelliNet is composed of four main components. Our network is organized into "blocks" that can be arranged to match the layout of the underlying buildings, leading to a simple and extensible topology. Our hierarchical networking protocol balances simplicity and fault-tolerance by using a constrained, local version of link-state routing. The FCS's data processing module intelligently requests data from smart devices to balance network load and achieves high performance via modularity and parallelization. Finally, the FCS's data storage module organizes data for later analysis and stores backups in the case of failure.

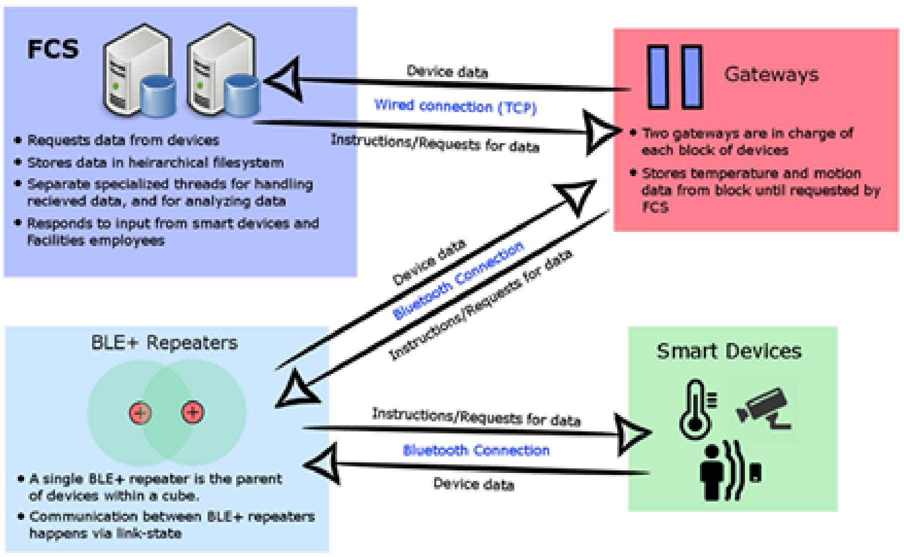
FCS
• Requests data from devices
• Stores data in heirarchical filesystem
• Separate specialized threads for handling recieved data, and for analyzing data
• Responds to input from smart devices and Facilities employees
Device data
Wired connection (TCP)
Instructions/Requests for data
Gateways
• Two gateways are in charge of each block of devices
• Stores temperature and motion data from block until requested by FCS
Device data
Bluetooth Connection
Instructions/Requests for data
BLE+ Repeaters
• A single BLE+ repeater is the parent of devices within a cube.
• Communication between BLE+ repeaters happens via link-state
Instructions/Requests for data
Bluetooth Connection
Device data
Smart Devices

Figure 1: *IntelliNet system diagram. Requests for data are sent from the FCS to one of two parent gateways per block, where they travel through the grid of BLE+ repeaters to get to the device. Device data follows a similar path to the FCS, which analyses and aggregates the data and responds with device-specific instructions.*

The sections that follow will specify our proposed design and how it achieves our main design goals of simplicity, scalability, performance, and fault tolerance.

3 System Design

3.1 Network Topology

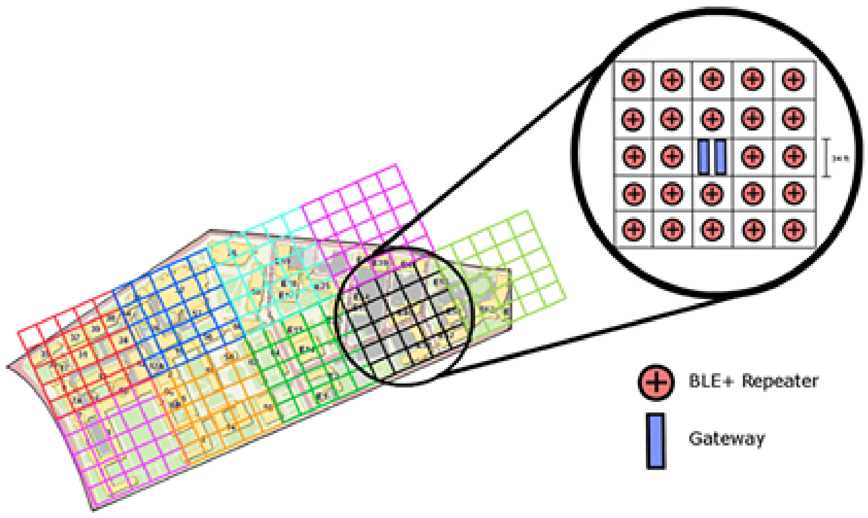

Figure 2: *An example of a flat block consisting of 25 cubes arranged in a* $5 \times 5 \times 1$ *grid. Note that there is also a BLE+ repeater in the center square, which has been omitted for clarity.*

We propose a modular, campus-agnostic network topology that is both simple and extensible. It divides the campus into hierarchical domains that provide the basis for our routing protocol.

IntelliNet's topology is composed of blocks, which are in turn subdivided into smaller units called cubes. A cube is a $34' \times 34' \times 34'$ unit with a single BLE+ repeater at its center. We choose a side length of 34 feet so that a smart device placed anywhere in the cube is within 30 feet of the repeater, the maximum range for a smart device (in the absence of walls). Each device is assigned to a single "parent" BLE+ repeater.

A block is made up of 20-30 cubes arranged contiguously so that each cube shares at least one face with another cube in the same block. Blocks can vary in their dimensions in order to fit the architecture of the campus. For instance, a large but short building complex, like the hallways around the infinite corridor, might use a 25-cube block with dimensions $5 \times 5 \times 1$, whereas Stata, which has 9 stories, might make use of a $3 \times 3 \times 3$ block. Each block contains two gateways, which are the "parent" gateways for that block.

Walls limit the range of smart devices and may put them out of range of their parent BLE+ repeater. To resolve this problem, we also place small chains of BLE repeaters, where required, to connect a device and its parent.

Building our network out of generic, composable elements contributes to simplicity and scalability. Our design makes no assumptions about MIT's architecture, allowing one to reason about standardized components like blocks and cubes instead of the particularities of each building. Thus, our system can easily accommodate future buildings or the needs of other institutions.

3.2 Network Protocols

Our hierarchical network protocol balances complexity and performance to meet Facilities' needs while handling device failures.

3.2.1 Naming

Each device has a unique 20-bit ID. In practice, this is implemented by setting the first 28 bits of the device's ID to 0. This length lets us fit both the source and destination IDs into a TCP header. 20-bit IDs are set randomly, with the provision that no ID collisions occur. Note that our system does not require IDs to be set perfectly, only that they be the same on the device and the FCS. Thus, our scheme tolerates ID typos.

3.2.2 Hierarchical Routing

The path a packet takes between a smart device and the FCS is illustrated in Figure 1. Packets from a device to the FCS are routed first to a device's parent BLE+ repeater and then to one of the two gateways in its block.

If a device is in range of its parent BLE+ repeater, it sends its packets directly; otherwise, its packets pass through a chain of BLE repeaters first. This protocol is simple but may generate some extra congestion. However, since repeaters have a very limited range (30 feet) and are only used in cases where range is already limited, we do not think this will cause significant slowdown in the network as a whole.

The BLE+ repeaters and gateways of a block form a mesh network. Packets are routed through this mesh via a link-state routing protocol until they reach one of the two parent gateways, which then forward them to the FCS. The distance metric we use for choosing paths in the network is number of hops between source and destination, and we choose which gateway to route through by selecting the one with the shorter distance. The procedure for sending packets from the FCS to a device is symmetric.

The system balances simplicity and fault tolerance. Although link-state routing adds complexity, it is only used to route between the repeaters and two gateways within a given block. In this limited scope, the flood overhead caused by link-state routing will be a constant factor, and will not scale with the size of the overall system. Importantly, this flexible routing scheme means that BLE+ and gateway failures can be detected and routed around (see our evaluation for more details).

3.2.3 Packet format

Each Bluetooth packet has a header of 64 bits. The fields in the header indicate the source and destination of the packet, the 20-bit sequence number, the direction (inbound/outbound from the FCS) and type (ACK, data, etc.).

Note that the packets sent between repeaters are Bluetooth packets, while the packets sent between gateways and the FCS are internet packets. Gateways repackage packets at the Bluetooth-internet boundary. To simplify our transport layer protocols, we enforce that all our internet packets be at most 20 bytes (the max size of a Bluetooth packet), so only their headers have to be changed.

Source ID (20 bits)	Destination ID (20 bits)	Sequence Number (20 bits)	Type (3 bits)	Direction (1 bit)
Body (160 bits)				

Table 1: Message layout for Bluetooth packets

3.2.4 Network Setup

Routing protocols in our network are bootstrapped in a hierarchical way using information managed by the FCS. Gateways have the FCS's IP address upon startup. When gateways come online, they send the FCS a message requesting routing information for their block (i.e. the IDs of the BLE+ repeaters and devices). The gateway stores the IDs of its BLE+ children in its routing table and then forwards this routing information to the BLE+ children in its range. In turn, repeaters store information about their BLE+ siblings as well as their device children. They pass this information to other BLE+ repeaters until all repeaters in the block have received the routing information. The FCS also stores the gateways' addresses in its routing table.

When a repeater or smart device is set up, Facilities enters the device's coordinates into the FCS. The FCS will calculate an appropriate block, cube, and parents according to the location of the device. The FCS will alert the parents and device of their relationship and devices will route accordingly.

3.2.5 Beacons and Announcements

Devices continually beacon their presence to other devices in range. BLE+ repeaters also send advertisements containing data about their neighbors every second to all other BLE+ repeaters in their link-state block.

In addition to providing a backbone for our routing system, these beacons/announcements can be used to detect device failure. If a BLE+ repeater stops receiving beacons from its child for more than 15 seconds, or if the distance to another node in the link-state mesh becomes infinity, this indicates that the device has failed. The BLE+ repeater sends a failure message to the FCS with the ID of the missing device. If the device is later rediscovered, the FCS sends a corresponding "reconnection" message.

3.2.6 Transport Protocols (TCP)

Our transport layer implements a simplified version of TCP over Bluetooth and Internet to ensure that messages are delivered reliably in spite of unreliable hardware. The data reliability achieved using reliable transport justifies the modest increase in congestion resulting from TCP (discussed further in the evaluation section).

3.2.7 Temporary Data Storage

Data accumulates in our network until explicitly requested by the FCS. Temperature and motion data are stored on the two gateways in each block. As soon as smart devices generate data, they push it to one of their parent gateways for temporary storage. Video data is buffered on the camera itself. Gateways and cameras wait for an FCS request to upload their saved data.

This push-pull model of data collection allows the FCS to intelligently load balance and manage congestion. It also lets us save data even in in the case of an FCS shutdown. This makes our system more performant and fault-tolerant.

3.2.8 Fault tolerance

Our design is resilient to gateway failure because our link-state protocol can route around a failed gateway (see our evaluation section for a full analysis). Regarding BLE+ repeater failure, our design balances resiliency and simplicity. If a BLE+ repeater fails, connection will be lost to its immediate children, but other data will be sent to the FCS as normal because link-state routing finds alternate paths from live BLE+ repeaters to the gateway. Since BLE+ repeaters take only 5 minutes to repair, we feel that the simplicity gained in naming, routing, and topology from having one parent per device justifies losing connection to children for this short window.

3.3 Data processing

As depicted in our system diagram (Figure 1), the FCS has three main jobs for data collection: requesting data from smart devices, processing incoming data, and responding to input from smart devices and Facilities employees. The FCS's processes are designed to be as modular and fault-tolerant as possible.

3.3.1 Data Requests

We use a push-pull model of data collection, which increases performance by allowing the FCS to intelligently coordinate data flow through the network.

3.3.2 Temperature and motion data.

The FCS requests temperature and motion data from gateways every two minutes, well within the 5-minute interval requested by Facilities. Requests to the various gateways are staggered throughout the 2-minute window to balance the load on the FCS. When a gateway receives a request from the FCS, it sends all the data in its backlog

3.3.3 Camera data

.

By default, the FCS will collect camera data from one camera per block at a time and will cycle through the cameras in a block. This prevents excessive camera data from overwhelming the network. In normal mode, the FCS requests all data in the camera's backlog at a rate of 1 frame per second. We think the low frame rate is acceptable for non-crisis film. The vast majority of this video will likely be used for traffic estimation on campus as opposed to viewing by a person, and it is unlikely that the number of people in a frame will change drastically within a second. We therefore sample at a lower rate to decrease congestion in the network and storage requirements on the FCS.

In crisis mode, the FCS requests real-time data from the crisis cameras in each block. Non-crisis cameras continue to store old frames in their backlogs. Given that the non-crisis cameras can buffer up to 40 hours of data at one frame per second, delaying collection from them should not cause data loss in most circumstances (see Section 5.1.1). Since we expect that crisis video will be watched by a human, we want to provide the highest frame rate the network can handle, which is 4 frames/second if there is one crisis camera in the block. We discuss this more in our evaluation section.

The FCS's ability to stagger the data coming from different devices will make the traffic less bursty, reducing congestion in the network and leading to a more performant system with lower latency. Furthermore, if the FCS goes offline for maintenance or because of a failure, it can resume operation normally with minimal data loss by continuing to request backlogged data stored at gateways and cameras.

3.3.4 Data Ingestion

All data enters the FCS via the same FIFO queue. Thus, it is necessary to split up the incoming data by type and pass it to the appropriate handler.

The FCS's main thread spawns child threads that pop data from the input queue and distribute it to several specialized thread-safe message-passing queues, each with their own workers. We designate special queues for temperature data, motion data, video data from non-crisis cameras, video data from crisis cameras, and update confirmations. Each queue's handlers take care of calling the appropriate methods to store the data for subsequent analysis.

This design is robust to failure because even if one thread fails, having multiple workers allows data ingestion to continue until the master can restart the thread. It is scalable because the amount of workers can be increased to handle throughput, and if multiple processors are required, the modularity of having specialized queues means that work can be split between processors. It is also simple because each worker only has to worry about performing a specific task on a specific type of data instead of coordinating with a host of other processes.

3.3.5 Data Processing

Certain information needs to be handled by the FCS in real-time: changing thermostat temperatures in response to motion data, responding to crises or update requests, and keeping track of failures. Special threads are assigned to each task.

Threads for data analysis are separated from threads for data ingestion: they read from data storage as opposed to directly from the input FIFO queue. Thus, a failure of the analysis module will not disrupt data ingestion. Also, this division of labor means that these two tasks can be run on different servers if Facilities decides to increase the FCS's processing power in the future, making this design scalable.

3.3.6 Thermostat updates

Thermostat temperatures must be adjusted based on motion data for a room. Every five minutes, a thread is awakened that reads the last timestamp from each of the motion detectors in a room to determine if temperature adjustments are needed and sends temperature-update commands to appropriate thermostats.

3.3.7 Responding to a crisis

Handlers that read from the crisis video data queue wait on the event `start_crisis_processing` to begin operation. The event `enable_crisis(camera_ID)` has two effects: `camera_ID` is added to a global set storing the IDs of cameras in crisis, and the event `start_crisis_processing` is emitted. `disable_crisis(camera ID)` results in `camera_ID` being removed from the set of cameras in crisis. Once all cameras have been removed from this set and the crisis queue has been cleared out, crisis queue handlers will resume waiting. Having separate handlers for crisis frames will reduce latency when dealing with this data because it will not have to queue behind other non-crisis camera data, ensuring that Facilities will have the information they need to resolve the crisis in a timely manner.

3.3.8 Responding to an update

On the event `update(device_type, software_binary)`, a thread awakens that sends the binary to devices of the correct type. This is done in a hierarchical way, where the update is first distributed to the gateways, which distribute it to the BLE+ repeaters, which distribute it to their child devices, etc. The thread returns once all devices have responded with an update confirmation message (processed via the update queue) or a timeout has expired, in which case the devices who did not respond are reported to Facilities as having failed.

3.4 Checking for malfunctions

As discussed earlier, devices detect if their neighbors are inaccessible and send failure messages to the FCS. The FCS maintains a set of failing devices and adds or subtracts from that set as it receives failure or reconnection messages, respectively. This list of IDs is continuously available to Facilities, making it easy to locate and repair failing devices.

4 Data Storage

The FCS's data storage unit stores data in the filesystem and makes use of backups to protect data.

4.1 Data Organization

The FCS will store its data in a hierarchical directory structure. The top level has three directories: `Temperature`, `Motion`, and `Video`. Each device category is separated out by room number and then device ID. Data received from smart devices is stored in `.txt` files corresponding to the date of generation. For instance, all thermostat readings for a single day will appear in their own `.txt` file. Each motion detector directory also contains a one-line file containing the last timestamp that it detected motion, allowing Facilities to easily determine if the temperature in the room should be changed. This simple directory structure allows Facilities to access any piece of data by traversing three directories.

The FCS writes to data storage by obtaining the lock for the file corresponding to the given device and appending to that file. In the case of motion data, it also overwrites the latest-timestamp file. Note that we are not required to store historical motion data, but we decided to provide it to Facilities to facilitate debugging of devices.

4.2 Data Backups

The FCS has 100 TB of storage to store a week's worth of camera footage, two weeks' worth of thermostat data, and various motion detector timestamps. Its hard drive will be divided into two partitions: one with 90 TB for normal data and one with 10 TB for backups. The 90 TB partition will store the camera, thermostat, and motion detector data. The 10 TB partition will store camera footage compressed using the HEVC standard, which reduces size by a factor of up to 1000, along with duplicates of thermostat and motion detector readings.

4.3 Deleting Data

To free space on the hard drive, every day at midnight, the FCS will go through each device directory and delete stale data. This can be accomplished quickly by deleting one file in each device directory corresponding to the oldest date.

5 Evaluation

5.1 Quantitative Evaluation

In this section, we will show that our system meets Facilities' technical requirements and functions subject to the limitations of our devices.

5.1.1 Topology

First, we estimate how many blocks will be required to cover MIT's campus. Recall that each block is composed of 20 to 30 cubes of dimension $34' \times 34' \times 34'$. Assuming that a typical story of a building is around 10 feet high, we conservatively estimate that a cube spans two stories. Thus, a single cube contains about 34×34 square feet per story times 2 stories = 2312 ft^2 of floor space. If the average block contains 25 cubes, then a block covers $57,800$ ft^2.

MIT's academic area is 7.9 million ft^2 [2], so our system requires about $7.9 \times 10^6/57,800 = 137$ blocks to cover MIT's buildings. Thus, our system requires $137 \times 2 = 274$ gateways and $137 \times 25 = 3,425$ BLE+ repeaters.

Walls limit the range of BLE+ repeaters. Devices within 20 feet of their BLE+ parent will be in range, even if separated by a wall. Smart devices farther away may require a BLE repeater to boost their signal. The volume inside a cube that falls more than 20 feet away from the BLE+ repeater has volume $5,794$ ft^3 [1]. The volume of the cube is $39,304$ ft^3. Thus, around $5,794/39,304 \approx 15\%$ of devices will require BLE intermediaries. For 31,000 total devices, this will be $0.15 \times 31,000 \approx 4,650$ BLE repeaters.

We can now calculate the cost of our system. We plan to spend $\$500 \times 274 = \$137,000$ on gateways, $\$100 \times 3425 = \$342,500$ on BLE+ repeaters, and $4650 \times \$10 = \$46,500$ on BLE repeaters. This brings the total price to $\$526,000$. We feel this price is reasonable given the performance and fault-tolerance of our system. Table 5.1.1 compares our system to a couple of naive approaches and shows that it is significantly less expensive while still meeting Facilities' requirements.

	One device per gateway	**Chain of BLEs (3 repeaters per device)**	**Intellinet**
Gateways	31,000	274	274
BLE+	0	0	3425
BLE	0	93,000	4650
Total Cost($)	15,500,000	1,067,000	526,000

Table 2: Our system is cost-effective relative to other naive options. The first alternative we consider is placing one gateway next to each device, which is perhaps the simplest and most performant option, but is 30 times more expensive than IntelliNet. The second is using the same gateway placement as our system, but connecting each device to a gateway with its own chain of the BLE repeaters.

There are 15,000 thermostats, 15,000 motion detectors, and 1,000 cameras around campus. After dividing the number of smart devices by the number of BLE+ repeaters, we see that every BLE+ repeater is expected to be a parent of around $(15,000 + 15,000 + 1,000)/3425 = 9$ smart devices. We also expect $1000/137 = 8$ cameras per block. These figures are important in later analysis.

5.1.2 Network

There are four types of traffic on the network: beacons (which do not create congestion), data, ACKs, and advertisements. We will show that even with all this traffic, data arrives at the FCS on time.

5.1.3 Advertisements

In our topology, BLE+ repeaters have at most 6 neighbor repeaters. They exchange advertisements with neighbors once every 30 seconds. The size of a link-state advertisement is:

$$544 \text{ bits } = 64 \text{ bit header } + 6 \text{ entries for neighbors } \times (48 \text{ bit neighbor ID } + 32 \text{ bit distance })).$$

If each repeater advertises to the (at most) 29 other repeaters in its block, this will introduce $30 \times$ sources $\times$ 29 recipients $\times$ 544 bits $=$ 0.5 Mbit of overhead every 30 seconds. Assuming that advertisement overhead is distributed evenly across repeaters, this is 0.016 Mbit of overhead per repeater every 30 seconds. A BLE+ repeater can process .016 MBits in .004 seconds, which means it spends only about 0.01% of its time processing advertisements. Thus, the congestion from link-state advertisements is negligible.

5.1.4 Temperature and Motion Data

When repeaters are not exchanging packets, they disconnect from each other. This ensures that bandwidth is not wasted on connections that are not transmitting data. A thermostat packet contains 128 bits: 64 bits for the header, 32 bits for the temperature data, and 32 bits for the timestamp. Consider the path of repeaters that a thermostat packet takes from the device to the FCS. Even if all the repeaters along this path have a full set of 16 connections (leaving $4/16 =$ 0.25 Mbit/s bandwidth), it will take only $5x10^{-4}$ s for a repeater to transmit this data. If each hop adds 100ms latency, then the packet will reach the FCS in at most

$$(30 \text{ repeaters } + 1 \text{ gateway }) \times 100 \text{ gateways } = 3.1 \text{ seconds}$$

Combined with the 1-minute interval that the device could wait to be buffered at the gateway, this easily meets Facilities' requirement for data freshness.

A packet of motion detector data contains 96 bits: 64 bits for the header and 32 bits for the timestamp. Thus, it takes even less time to send.

5.1.5 Camera Data

We expect to see approximately 8 cameras per block. As shown in our previous section, thermostat, motion detector, and advertisement data can pass through a repeater very quickly. Thus, we make the simplifying assumption that camera data is the only traffic on the network.

Recall that the FCS requests data from the cameras in rounds. This means that there is only one camera's worth of data traveling through the mesh at a time. If a BLE+ repeater has 2 connections open (one to the camera, one to the destination repeater/gateway), then the repeater has 4 Mbit/s $/2 = 2$ Mbit/s of bandwidth available for camera data.

If a camera films 1 frame per second, and the FCS polls the camera every T seconds, then a camera will accumulate $T \times 0.224$ Mbit of data: $T \text{ s } \times 1 \text{ frames/second } * 0.224 \text{ Mbit/frame } = T \times 0.224 \text{ Mbit }$ of data Then for $T = 60$s, a single camera's data can be processed in

$$T \times 0.224 \text{ Mbit } \times (1 \text{ s } /2 \text{ Mbit }) = 6.72 \text{ seconds}$$

Thus, the FCS can process a minute's worth of camera data from a block of 8 cameras in

$$6.72 \text{ s} \times 8 = 53.76 \text{ seconds}$$

The FCS will finish processing the block before the next round of requests begins. This also allows each camera to send an extra 1.56 Mbits of video data from its backlog every round.

5.1.6 ACKs and Retransmissions

Assuming devices are distributed uniformly, there are

$$31,000 \text{ devices } /137 \text{ blocks } = 226 \text{ devices per block}$$

Consequently, there are at least 226 ACKS that must circulate through the block per second, assuming that each thermostat and motion detector pushes once per second. In addition, in one second a single camera will dump:

$$12,500 \text{ packets } = 2 \text{ Mbit } /160 \text{ bits per packet body.}$$

This will result in 12,500 ACKs. Each ACK only requires the information in the header, so it only adds 64 bits of bandwidth. Thus per second, the added congestion is:

$$((12,500 + 226) \times 64 \text{ bits }) = 0.81 \text{ Mbit}$$

Assuming that this congestion is spread evenly throughout the network, it will not place a burden on any single repeater. We also estimate congestion resulting from re-transmission. If there are a maximum of 30 repeaters that a packet must pass through in a mesh, then then we can union-bound the probability of dropping any given packet as:

$$30 \text{ hops } \times 0.0001\% = 0.0030\%.$$

Assuming that retransmissions only occur once per packet, the expected additional congestion will be: $0.000030 \times ((1,400 \text{ packets/s } + 15 \text{ seconds } \times 226 \text{ packets/second }) \times 224 \text{ bits }) \approx$ 36 bits/second, which is again minimal.

5.1.7 FCS

The FCS has no problem receiving data from the network. We consider camera data for simplicity since it is by far the largest flow of data in the network. There are 137 blocks, and the FCS pulls 1 minute's worth of camera data from 8 cameras every minute.

$$1000 \text{ cameras } \times 0.224 \text{ Mbits/frame/camera } \times 1 \text{ fps } = 224 \text{ Mbits/s } = .028 \text{ GB}$$

Which is well below the FCS's 1G/s intake limit.

5.1.8 Data Storage

The amount of data stored on the FCS at any moment will never exceed 100TB. The data on the 90 TB partition is calculated as follows:

1. **Video Data:** MIT campus has 1000 cameras, each of which can record at 1 to 5 frames per second. Each frame is 28 KB, and Facilities wishes to store a week's worth of video data. To account for higher resolution video in crisis mode, this calculation assumes that every camera records at 4 frames per second (in reality, most video stored will be at 1 frame per second). Since the FCS deletes data once per day, the FCS holds 8 days worth of data in the worst case. Thus, the total storage required by all 1000 cameras in a week is as follows:

$$\begin{aligned}&(28 \text{ KB/frame}) \times (10^{-9} \text{ TB/KB }) \times (4 \text{ frames/sec }) \times (3600 \text{ sec/hour })\\ &\times (8 \text{ days }) \times (24 \text{ hours/day }) \times (1000)\\ =&77.4144 \text{ TB}.\end{aligned}$$

2. **Thermostat and Motion Detector Readings:** MIT campus has 15,000 thermostats and 15,000 motion detectors. A single reading from each device is 8 bytes: a 32-bit float and a 32-bit timestamp. Facilities wants to store two weeks' worth of readings in 5 minute intervals, but in our design, devices sample state every minute. Thus, the total data used by a device in a week is:

$$(8 \text{ bytes/reading }) \times (1 \text{ reading/minute }) \times (10080 \text{ minutes/week }) = 80640 \text{ bytes/week}$$

Thus, over the course of 2 weeks and 1 day, all 30,000 devices will use 5,184,000,000 bytes, or $5.184 * 10^{-3}$ TB.

Altogether, the 90 TB partition uses less than its maximum capacity to store all of the data required by Facilities.

The 10 TB backup partition stores copies of the thermostat and motion detector readings, along with compressed versions of video data using the HEVC standard, which reduces size by a factor of up to 1000. Altogether, this uses $0.0774144 \text{ TB } + (5.184 \times 10^{-3} \text{ TB }) = 0.0825984$ TB of data.

5.1.9 Extent of Scalability

The biggest limiting factor on our system is the density of cameras. In our evaluation, we assume 8 cameras per block. If there are more than 8 cameras in the block, then a chain of repeaters must be added to connect any additional cameras directly to the gateway. If we were to implement this system on a more camera-dense campus, we could simply choose a smaller cube size so that each cube contained fewer devices.

Extending our system to cover more area, such as a new building, is simple: we simply allocate additional blocks to cover the desired area. The FCS's bandwidth could easily handle $10\times$ the data received from MIT's campus, and our ID format accommodates over a million devices.

FCS processing power is another limiting factor in our system. However, since our FCS processes are very modular, we can scale by splitting workers across multiple machines. For instance, we could connect two processors to a common database and have one take care of ingesting data from the input queue and the other run analysis jobs, like sending temperature updates.

5.1.10 Ease of Use

Installing a new device is fairly simple. The facilities employee can place the device wherever is convenient, then input the device's coordinates and room number into the FCS to receive a random id to program into the device. No manual configuration is required to ensure proper routing.

The FCS also quickly reports broken devices to Facilities, eliminating the need for manual debugging and fulfilling one of their design requirements.

5.2 Exceptional Use Cases

We analyze how well our system can operate under particular use cases that may come up.

5.2.1 New Devices

When a new device is first entered into the FCS, the FCS alerts the assigned parent repeater that it has a new child so it will begin forwarding the data. As calculated above, a pessimistic estimate for the network's latency is just over 3 seconds, so once the device's info is entered into the FCS, it should be ready to broadcast data almost immediately.

5.2.2 Crisis Mode

In crisis mode, the frame rate that our system can live-stream depends on the number of cameras in crisis mode per block. If only one camera is in crisis mode and produces 5 frames per second, then it will generate: $0.224 \text{ Mbits/frame } \times 5 \text{ fps } = 1.12 \text{ Mbits/second}$ If all repeaters in the block only have two connections open (one to receive camera data, and one to send camera data), a repeater will have 2 Mbit/s bandwidth available per connection. Thus, the data can be live streamed.

Similar calculations (omitted for brevity) show that we can stream data from 2 cameras at 2 frames per second and data from 3 cameras at around 1 frame per second.

During a crisis, data from non-crisis cameras will accumulate in camera storage. Once the crisis ends, all cameras will begin to send their backlogged data, sending the oldest frames first. To meet Facilities's requirement on the staleness of camera data, backlogged data cannot be more than 5 hours old.

Recall that it takes roughly 53 seconds for a block to clear 60 seconds of data for 8 cameras in a block, leaving 7 seconds to send backlogged data. Since the backlog of camera data drains at roughly 10% of the rate of real-time data, it takes about 2 days for the camera data from a 5-hour crisis to drain completely. Thus, our system can handle a 5-hour crisis every 2 days.

5.2.3 Software Updates

We assume for ease of analysis that updates are not attempted soon after a crisis or an FCS failure– that is, there is no camera data backlogged in the network. We use the extra "slack" bandwidth in our network that is used for sending backlogged camera frames to send out the update. If we assume conservatively that we send the update to one device in a block at a time, then we have around 7 seconds of 2 Mbit/s BLE+ bandwidth that won't be taken up by camera data. If our update is 1MB large and we are sending it to the expected 110 thermostats in a block, then the update's propagation takes:

$$\begin{aligned}&(110 \text{ thermostats } \times 1 \text{ MB/thermostat })/(7 \text{ s } \times 2 \text{ Mbit/s/ minute })\\&= 880 \text{ Mbit } /14 \text{ Mbit/minute } \approx 63 \text{ minutes}\end{aligned}$$

We believe this is a reasonable amount of time to update 15,000 devices.

5.2.4 Failure Cases

As discussed in the section on beacons, device failures are detected by the network within 15 seconds and indicated to the FCS. Assuming a pessimistic 3-second latency for the failure message to reach the FCS, this means that FCS will know about a failure in under 20 seconds. The FCS can provide facilities with the exact location of the failing smart device to facilitate quick repairs.

If a BLE+ repeater fails, we lose connection with its child devices. Given the quick rate of failure detection and the quick repair time for these devices (5 minutes), we would not expect a parent repeater to be out of service for more than half an hour. Any cameras within the cube of the failed BLE+ repeater would be able to store video data in their backlogs just as they would during a crisis. In summary, we would expect to lose about 30 minutes of temperature/motion data for devices in that block and no camera data. We trade off this small loss of data for the simplicity gain of having one repeater parent per device

Let's examine a gateway failure. When a gateway fails, we will lose at most 2 minutes of temperature and motion sensor data buffered there, which still gives us time to meet Facilities' 5-minute deadline. Recall that there are 2 gateways per block. If the FCS receives the message that one gateway failed, it will route its messages through the remaining gateway. Within the network, the link-state routing protocol ensures that the BLE+ repeaters route packets towards the accessible gateway.

A single gateway can buffer all the temperature and motion data flowing through a block. On average, we expect 110 thermostats and 110 motion detectors per block. Each thermostat contributes 64 bits of data every minute to the gateway. Each motion detector sends a 32-bit timestamp every minute to the gateway. In total, we expect the total data generated by all smart devices in the block to be:

$$\begin{aligned}&(110 \text{ thermostats }) \times (64 \text{ bits / minute })\\&+ (110 \text{ motion detectors }) \times (32 \text{ bits / minute })\\&=10560 \text{ bits/minute.}\end{aligned}$$

After 2 minutes, the FCS accumulates (2 minutes)*(10560 bits/minute) =2.64 KB of data stored, which is significantly less than the maximum allowed 32 GB of storage for a gateway.

A single gateway can also broadcast data from all 8 cameras. Every minute,

$$(60 \text{ sec }) \times (0.224 \text{ Mbit/s/ camera }) \times (8 \text{ cameras }) = 107.52 \text{ Mbits}$$

of video data is pumped into the block by the cameras. The single gateway, working with two connections, can clear this data in (107.52 Mbits)×(sec /8 Mbits) = 13.44 seconds, which means that the gateway can process camera data as fast as it is generated.

We can also handle FCS failures. In this case, temperature and motion data will have to be stored in the gateways for as long as possible. If temperature and motion data is evenly buffered between two gateways, we can store data for

$$\begin{aligned}&(32 \text{ GB } + 32 \text{ GB }) \times (8 \times 10^9 \text{ bits / GB })\\&\times (1 \text{ minute } /15840 \text{ bits }) \times (1 \text{ hr } /60 \text{ minutes })\\&= 538,720 \text{ hours.}\end{aligned}$$

Video data is buffered on cameras themselves at 1 frame per second. Thus we can store video data on the backlog for at most

$$(1 \text{ sec } /1 \text{ frame }) \times (1 \text{ frame } /28 \text{ kB }) \times (4 \text{ GB }) = 39.68 \text{ hours.}$$

In the case of a 10 hour FCS downtime, each camera would accumulate

$$(28\text{ kB / frame }) \times (1\text{ frame / sec }) \times (10\text{ hours }) \times (8\text{ bits / byte })$$
$$= 8064\text{ Mbits.}$$

We would expect to have all this video data back at the FCS in

$$(8064\text{ Mbits }) \times (1\text{ min }/1.56\text{ Mbits }) = 86.15$$

hours, or a little over three and a half days. Thus, we can safely recover from a 10 hour FCS failure, though the video data would be considered stale by Facilities.

5.3 Security

To ensure that only Facilities staff members can securely log into the system, we implement a password system that stores hashes of members' passwords concatenated with salts, which prevents adversaries from breaching the system by constructing a lookup table. This design assumes that FCS employees are themselves trustworthy or that proper management oversight is in place to prevent abuse.

Even though smart devices are unable to encrypt data, the setup of our system ensures that adversaries cannot easily stalk individuals on MIT campus. Thermostat and motion detector data offer no useful information about individual behavior. Video data is more sensitive because it can be used to identify individuals. However, our cameras are all located in semi-public areas such as hallways, so privacy expectations are lower. Also, since the FCS pulls video data in bursts and not in real-time, it isn't possible for an adversary to follow someone across campus. Thus, our system provides a reasonable degree of protection for people on campus.

6 Conclusion

Our proposed system fulfills the three main goals of MIT Facilities: collecting data for future project planning, reducing temperatures of inactive rooms to save energy, and automatically detecting infrastructure failures to facilitate faster repairs. Our system ensures that commands and requests for data are communicated reliably, even when individual machines fail. Our system also has mechanisms to prevent data loss in the case of FCS failure. Designed to be simple, performant, fault tolerant, and scalable, our system will provide the infrastructure needed to improve the quality of campus life.

7 Individual Contributions

We worked very well together as a team and divided up work equitably. Each person was in charge of designing and writing the description for one main component of the system (Giancarlo: topology, Christopher: network, Anelise: FCS, Eric: data storage). We then worked together to refine our design, edit our report, and finalize our calculations.

8 Acknowledgments

We would like to thank our recitation leader Adam Belay and TA Zach Miranda, for their helpful technical feedback on our preliminary design. We would also like to thank our Jared Berezin for his guidance and comments on the first draft of our report. Finally, we thank Katrina LaCurts and the 6.033 staff for their help and instruction throughout the semester.

References

[1] White, Andrew
http://thewhitelab.org/Blog/2013/04/cube-sphere-intersection-volume.html

[2] MIT Facilities http://web.mit.edu/facilities/maps/build-info.html