FreeSeats

Authors: Nathan Ang, Jonathan Cheng, William Foy
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of reflecting work/study-space availability in real-time, giving students valuable insight into where they choose to settle down for the day. This eliminates the disappointment people often face after traveling to a cafe or library, only to discover there are no seats available. The product is able to detect how many available and taken seats there are in a given study area and reflect that information on a publicly accessible web interface in real-time.**

***Index Terms*—Circuits, IoT, MERN, Occupancy, Raspberry Pi, Smart Spaces, Web Application, Zigbee**

1 INTRODUCTION

Each day, students at Carnegie Mellon University find themselves disappointed after arriving at a study space that is at maximum capacity. This is a problem FreeSeats solves, allowing users to view available seats at various spaces near them. This will save students' time, effort, and frustration as it allows them to know space occupancy ahead of time. FreeSeats utilizes a distributed network of sensors attached to each chair for occupancy detection, micro-controllers for data collection and processing, and a web software stack for user interfacing.

Competing technologies include a previous Capstone solution, which attempted to use Computer Vision to detect occupancy [1]. Our solution improves upon this, as physical sensors have the capacity for much higher detection speed and accuracy. The previous Capstone solution has 1 minute detection latency and aimed for 70% accuracy. FreeSeats detects occupancy at variable intervals, up to 15s, derived from the minimum time it takes for someone to find a seat starting outside a study space. Output of taken and available chairs must have an accuracy of at least 75%, ensuring user satisfaction and improving on a previous, parallel Capstone solution. Finally, FreeSeats is energy efficient and has a battery life of over a month . Our goal is to not only make FreeSeats an optimal and viable solution to the problem students face each day at study spaces, but also extendable to any space such as theatres, stores, restaurants, etc.

2 DESIGN REQUIREMENTS

The project's major design requirements can be broken down into two categories: speed and reliability. Since this is a real-time system, low system latency (ie. speed) is a priority. Our group conducted preliminary testing in Sorrells library, bench-marking how much time it takes to go from the library entrance to sitting down at a vacant seat. The results averaged to approximately 15 seconds. Thus, new updates must arrive within 15s intervals for users to consume. Apart from data freshness, we also have to consider user experience. Previous academic studies we have encountered during our research conclude that the web application should have a process latency of under 2 seconds to minimize the loss of users. Any longer would be an intolerable wait time, and the user may stop using the product.

To test whether our system meets these speed requirements, timing the time from sitting on the chair to seeing the web application update is the core test. This tests end-to-end latency between all components of our system. As for the web application itself, we can use the application's software itself to embed timers in code and take down the performance data.

The second category of requirements relates to reliability. Firstly, we want zero down-time of the system during normal operation. In other words, under a regular working day, the system should be up and running for all 24hrs. While it may be reasonable to synchronize the downtime of the system to the closing time of the work space itself, we want FreeSeats to be as portable as possible, and thus require that our design supports 24/7 operation. In addition, our sensor-enabled chairs shouldn't be tied down to power outlets, and thus must be battery powered. Since the chair sensor is battery-powered, there must be some downtime whilst routine battery changes are happening. After looking at portable high-capacity batteries ($2000mAh$) and our requirement for 2 months of battery life, we can estimate the average current draw we need to meet our specs.

$$2 \text{ months} = 60 \text{ days} = 1440 \text{ Hours} \tag{1}$$

$$\frac{2000mAh}{1440 \text{ Hours}} = 1.38mA \tag{2}$$

Thus we needed to aim for an average current draw of less than 2mA to reach our requirements.

Another aspect of reliability is accuracy. We conducted a small set of user polling, and found that people would tolerate about 1 faulty update every minute ("faulty" being defined as $< 100\%$ of chairs are accurate). Since we have 4 updates a minute, three of those four updates must be completely accurate. Thus, we require $\frac{45s/15s}{60s/15s} = 75\%$ accuracy on the work space level. Translating that to the individual chair sensor accuracy, lets assume Sorrells has 50 chairs: $(75\%)^{-50} = 0.994 = 99.4\%$. This means each chair must be accurate 99.4% of the time.

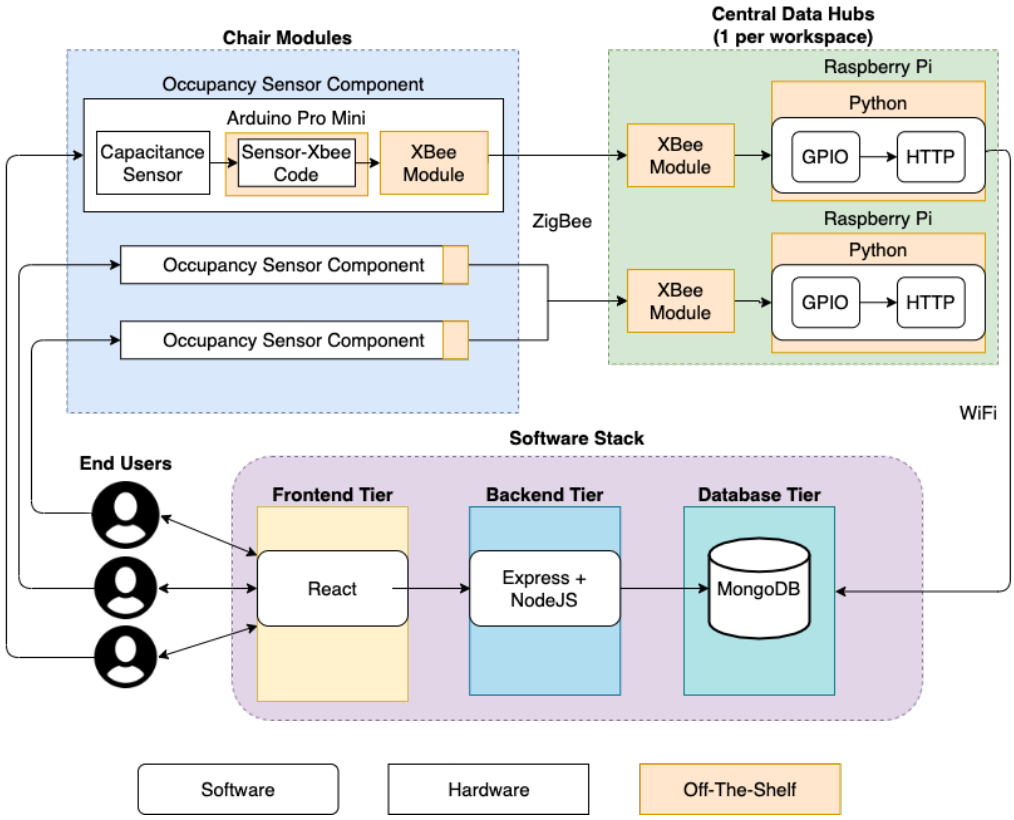

Figure 1: Block diagram depicting the data flow of the FreeSeats system.

3 ARCHITECTURE OVERVIEW

When we were designing the system architecture, our priorities were scalability, maintainability, and modularity. We wanted a system that could comfortably support our intended use case of multiple campus workspaces, each with 50+ of chairs. In addition, each component of the system should be a black box to other components, drastically increasing maintainability of the code base and hardware. We also wanted to design our solution to work with a plethora of chair types such as not to limit the product to just one type of chair. Finally, architectural components should represent modular, independent pieces of the FreeSeats puzzle, split along logical divides of responsibilities.

The FreeSeats architecture, thus, has been divided into three main components: Chair Modules, Central Data Hubs, and the Software Stack. Each component has a distinct and critical responsibility, and as per our design principles listed earlier, each one is a black box to the others.

The Chair Module's purpose is to detect human presence on the chair level and report it to the Data Hub. The Data Hub then aggregates all this incoming data from multiple chairs in the workspace and intermittently pushes updates to the backend web API. Finally, the Software Stack powers the web application, which pulls from the backend and presents the data in a user-accessible manner with a mobile-friendly frontend.

Note that in this Architecture Overview, we will only be going over component designs and purposes, not implementation. For implementation details, please refer to the System Description.

3.1 Chair Module

The Chair Modules (Figure 1, top left grouping) are a collection of IoT devices connected together to detect human presence in a specific chair. Every individual chair module should be discreet enough to be tolerable, if not unnoticeable, to the seat occupant. As such, the chair modules will either be hidden on the underside of the chair itself, or nestled under the chair cushion.

As stated above, the purpose of the Chair Module is to determine whether a given chair is currently occupied or not. This necessitates a sensory sub-component, some circuitry by which to actually determine occupancy. The sensor will most likely be the component nearest the individual, so this component must be small and discreet.

Next, in order to work with the sensor findings, the Chair Module needs some sort of compute power. This should be a low-power, lightweight microcontroller that can be powered by battery for up to 2 months, according to our design requirements. In addition to being low-powered, the compute must be network-enabled with a chosen system-wide protocol. This is needed to communicate the sensor readings with the Central Data Hub of the local workspace. In order to consume as little power as possible, the Chair

Module must only send updates to the Data Hub when changes in state occur.

3.2 Central Data Hub

The Central Data Hub (Figure 1, top right grouping) is a single unit of significant compute power, responsible for listening to all broadcasted Chair Module sensor readings and comparing the data against the previously saved state of the workspace. In addition, the Data Hub will push regular updates to the backend API, updating the database that the web application pulls from. The workspace range will be defined by the range of the Zigbee networking protocol, specified in the System Description. Unlike the Chair Module, this will not be in direct contact with any users, so there is no need for it to be hidden.

Since the Data Hub's purpose is to aggregate and send mission-critical data, the system cannot tolerate any unexpected Data Hub downtime. Thus, the Data Hub must be powered via a constant and reliable source of power, such as a wall socket. Since there is only one Data Hub per workspace, the consumption of the wall socket (something that is usually a scarce resource) should not be a problem.

The Data Hub will only be receiving changes in chair status, and as such they have to maintain an internal map of all chair states in the workspace so the updates can still paint a full picture. This full state will then be sent via HTTP to the web application backend.

In order to reduce network traffic, the Data Hub will be pushing fresh state only intermittently, as opposed to whenever a new chair update is received. These intervals will be 15s long, in accordance with our design requirement.

3.3 Software Stack

The Software Stack (Figure 1, bottom grouping) is a collection of different web software components, namely the Database, Backend, and Frontend tiers.

Starting in the middle with the Backend tier, we can see that this is the component that ties everything together. The Backend is responsible for receiving fresh workspace state from Data Hubs, and populating the Database tier with the fresh data. In addition, it will fetch relevant data from the Database, process it appropriately, and feed it to the Frontend when requested. This is the coordinator for the web application's data flow, as well as the interface for our hardware components. This is a component that we will be writing ourselves.

As mentioned briefly, the Database tier will store data for the Backend. It will support standard CRUD (Create Read Update Destroy) operations on data, and have persistent storage for long-term functionality. Workspace states will be pushed and stored on this tier, and fetched whenever the Backend sends a request. While we will not be writing our own custom database, we will be writing the interfacing code between the Backend and our cloud-hosted Database solution.

The Frontend tier's sole responsibility is to take the data from the Backend and display it on a mobile-friendly web application user interface. The Frontend is only a consumer of data, and does not support any data feedback to the Backend and/or Database. All visual components of the Frontend will be compatible with both desktop and mobile users, as specified in our design requirements.

The full Software Stack will be tightly integrated and highly responsive, in order to satisfy the two second latency threshold laid out in the design requirements.

4 DESIGN TRADE STUDIES

Different designs had to be considered in order to pick the best solution to meet our user requirements. Research and studying the trade-offs between different designs led to the selection of a design best-fit in order to implement a working, proof-of-concept system.

4.1 Chair Sensor

One of the requirements for a satisfactory product is to have an accurate system where 75% of the time, all chairs in a space report accurate readings on the web app. In order to develop a solution with high accuracy, different occupancy sensors were considered for the chair module.

- Small load sensors on the feet of a chair were considered since implementation seemed easy and the sensors would be very reliable. Reasons to not use these included the fact that a bridge board and amplifier board would be needed to connect the sensors to the microcontroller. Also, lots of wiring would be needed to connect the sensors on the feet of the chair to the rest of the chair module underneath the seat.

- A commercially available seat sensor used in car seats was also considered. This sensor goes in a cushion that a user sits on and the sensor automatically detects occupancy. This sensor is an easy solution, but it involves changing the nature of the chair by adding a cushion, and the sensor itself in addition to a cushion didn't make sense from a cost perspective when developing a scalable solution.

- Our final consideration and chosen design involves pieces of aluminum foil in conjunction with resistors and the Arduino library CapSense. This solution should be inexpensive, reliable, and hopefully highly accurate. The aluminum foil acts as a capacitive sensor such that when human presence comes within inches of the foil, an electrical disturbance is created which the Arduino can pick up.

We believe that the capacitive sensor will be the best combination of having to add minimal components to a chair while being discrete and also highly accurate.

4.2 Chair Microcontroller

Another requirement for this project is to allow a battery life of up to 2 months. To accomplish this, we need to limit the power consumption as much as possible. This lead to us considering different options for the microcontroller that will run off of battery power.

- We first considered the NodeMCU which is a variant of the ESP8266. The pros for this microcontroller include its price of only $5 and it's also completely compatible with Arduino code and libraries. It also has a built-in WiFi module but our chair sensors did not need this capability. The NodeMCU turned out to consume more power than other alternatives, so we went in another direction.
- We also highly considered using the Arduino Nano. It's highly used in practice so there is a lot of documentation online along with compatibility with the CapSense library we aim to use. It has a small form factor and requires at least a 5V unregulated voltage source. It consumes less power than the NodeMCU and initially seemed like the best option.
- Our final consideration was the Arduino Pro Mini. In comparison to the Nano, the Pro Mini only needs a 3.3V unregulated voltage source. It doesn't have a USB port, which cuts on power consumption but also makes it a bit more difficult to upload code. The Pro Mini is a bit smaller than the Nano as well but also has the same ATMega328p processor and all needed Arduino libraries are still compatible. The Pro Mini consumes less power than the Nano and NodeMCU according to our research.

The table below compares the power consumption of the three microcontrollers.

Microcontroller	Power Consumption
NodeMCU	80mA
Arduino Nano	22.1mA
Arduino Pro Mini	5.1mA

We believe the Arduino Pro Mini is best suited for our solution since it consumes the least amount of power and is compatible with our capacitive sensor and XBee.

4.3 Communication Protocol

There needs to be some form of communication between the chair modules and the local data hubs. This communication needs to be power efficient in order to meet our battery life requirements, and it also has to be scalable to however many chairs are in a workspace with a single datahub.

- An initial consideration was using WiFi and having the chair modules use NodeMCUs which have WiFi capabilities. This would be a simple solution since each chair module could stream their data directly over WiFi to our database. We chose not to use WiFi since WiFi is a very power hungry protocol and not very power efficient. Research shows that using WiFi consumes 5-10 times the amount of power compared to a Zigbee-enabled device. Also, we would need to rely on the CMU WiFi network, which is a potential risk.
- Bluetooth was another consideration. Bluetooth uses about half as much power as Zigbee, although they are both much lower than WiFi. Bluetooth is very power efficient but unfortunately is limited to around 10 nodes that a single node can connect to. Thus it would be impossible for the data hub to use bluetooth to communicate with all chair modules. One consideration was to add intermediary bluetooth hubs at different tables for a few chairs, but this added another entire layer of complexity and cost.
- Our final consideration was Zigbee. Zigbee has a power efficiency very comparable to B‘luetooth and thus met our needs for that. Zigbee also supports mesh networking and nodes can connect to nearly 10,000 other nodes, so it is very scalable. Zigbee does require an external module to connect to the microcontroller in order to send and receive Zigbee messages. Zigbee modules such as XBees have indoor ranges of close to 250 ft.

The table below compares the power consumption of the three communication protocols when transmitting data [2].

Communication Protocol	Power Consumption
WiFi	251mA
Bluetooth	39mA
Zigbee	52mA

We chose to use Zigbee to communicate between our chair modules and data hubs because we believe it meets both of our requirements for battery life and scalability. The mesh networking capability will also allow use to extend the range of our system past just the range that the data hub module has since chair modules can relay information from other chair modules.

5 SYSTEM DESCRIPTION

Our overall system is comprised of three subsystems: a chair module, data hub, and web application. The chair module is responsible for collecting occupancy data, the data hub collates data from a single workspace, and the web app takes the data and shows it to the user of our product.

5.1 Chair Module

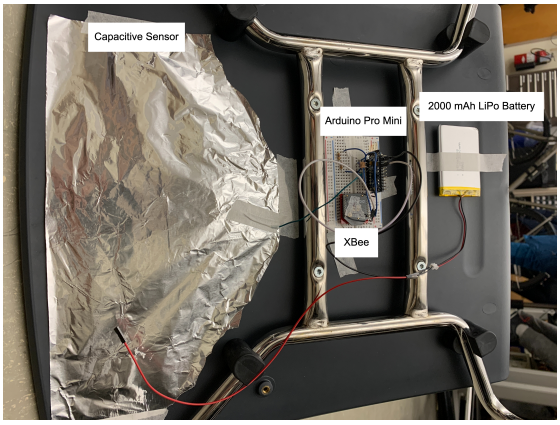

Figure 2: Image showing the components of the chair module underneath a chair.

As mentioned earlier, the chair module is responsible for determining whether a chair or not is currently occupied and relaying that information to the data hub so that the data can be populated in the web app.

The core component of the Chair Module is the Arduino Pro Mini, an inexpensive and small, yet powerful microcontroller. The Arduino Pro Mini was chosen due to its power consumption as well as compatibility with the CapSense Arduino library and the XBee module.

The first component of the chair module and most vital is the capacitive sensor. With a large piece of aluminum foil on the bottom of a plastic chair, we can detect human presence when someone sits down. The aluminum foil is connected to the Pro Mini with a $20M\Omega$ resistor, and the Capacitive Sensing Library is used to detect when someone sits down or stands up. We did not need to write the algorithms to detect changes in capacitance as we were able to utilize the CapSense library [3].

In order to send data to the Central Data Hub, the chair module needs to use the Zigbee networking protocol. To save power, only changes in occupancy are actually sent from the Chair Module to the Central Data Hub. Since mesh networking is a feature of Zigbee, any messages a Chair Module receives from other Chair Modules will also be forwarded on. To enable Zigbee, an XBee board must be interfaced with the Pro Mini. The Tx and Rx pins of the Mini can connect to the data lines of the XBee. To program the XBee we used the XCTU program which handles all configuration.

The last component of the Chair Module is the battery. Since there will be Chair Modules on every chair in a workspace, it's impractical to wire them all up to wall outlets. Thus it becomes necessary to run everything off batteries, and have those batteries last as long as possible. In order to meet our battery life requirement of over a month, we have chosen to use a 2000mAh LiPo battery with a voltage of 3.7V. We chose this battery since its compact, has a large capacity, and runs at 3.7V which is the minimal voltage we need to power the Arduino Pro Mini and XBee, thus not wasting any extra voltage. This capacity is also high enough to reach our battery life requirements as shown in (2).

To save power we programmed the Pro Mini and XBee to go into sleep mode whenever they are not being used for computation or communication in order to get the current draw as low as possible. The Pro Mini's processor goes to sleep between the intervals that it reads the capacitance of the capacitive sensor. Whenever we aren't checking if there is a change in occupancy, to save power we go to sleep. Likewise whenever we aren't sending information with the XBee we put the XBee to sleep to save more power as well.

5.2 Central Data Hub

As described in the Architecture Overview, the Central Data Hub is in charge of aggregating the chair module data in the workspace, and periodically pushing these updates to the backend API of our Web Application.

At the core of the Central Data Hub is the Raspberry Pi, a full single-chip computer, complete with an operating system and Wifi capabilities. We selected the Raspberry Pi because of its rich functionality and powerful processing, especially when compared with other common IoT options such as the ESP series or Arduinos. This is the main compute component of the hardware-facing side of the project, so it has to be able to comfortably handle many network inputs and outputs, as well as execute computations quickly enough to meet our strict timing requirements (see Design Requirements section). In addition, the Raspberry Pi is Wifi-enabled by default, which is crucial for communicating fresh data to the backend API endpoint.

This brings us to the Zigbee networking protocol. As described previously, the Zigbee protocol allows for potentially hundreds of nodes to communicate in a low-power manner. To enable the Raspberry Pi with Zigbee functionality, we also have to use the XBee module and program it to be a Zigbee receiver. This module will be electrically attached to the GPIO pins of the Raspberry Pi, and will receive network data from other XBee modules (attached to the Chair Modules) to make available to the Raspberry Pi code.

Finally, we have the software portion of the Central Data Hub. We plan to have a Python script that polls the XBee module using the Python Requests library (designed for these types of use cases). With every update that the XBee receives, the Raspberry Pi will update a virtual workspace state that keeps track of how many chairs are available and taken. This workspace state is then what is pushed up to the backend API using the Python Requests library (and sending simple POST requests with the appropriate arguments).

In terms of power, the Central Data Hubs in the FreeSeats system have to be plugged into a wall outlet. This is because, as stated in the Design Requirements, the Data Hubs convey mission-critical information from the chairs to the web-app. A power outage on the Data Hub's

part will cause the data flow to be broken, and the whole system will be crippled. As such, the FreeSeats system cannot tolerate the Data Hubs ever losing power, and thus we must go with the most reliable option available.

5.3 Web Software Stack

The Web Software Stack is a classic MERN stack, chosen for the use-case focus of creating a functional, scalable MVP as fast as possible. This means MongoDB for the data base, Express.js and Node.js for the backend, and React.js for the frontend.

The database schema, as seen in Figure 3, will be setup in a one-to-many data model. Each central data hub will have its own id, and a map of seat id to seat. Each seat will have its own id, and status of occupied or not. The map data structure enables for seat updates in O(1) time complexity in contrast to the O(n) time complexity of a naive list approach, with n being the number of seats. Central data hubs send data of all their respective chairs to an API endpoint in the same schema. On the database, chairs are updated as necessary, and the data will be posted to the front-end for users to see.

For interacting with the database, we implemented an API in JavaScript to perform CRUD operations. This allows for creating, deleting, updating, and getting hub and seat data,

On the front-end in Figure 4, each study space is divided into regions, for more specific occupancy data.

For hosting the website as well as the server for the API, we used Heroku. This resolves all DNS and CNAME configurations appropriately as well as load balancing for free.

6 TEST & VALIDATION

Since the nature of FreeSeats is such that many configurations of chairs and data hubs can exist in the same ecosystem, we planned to test every single combination of 3 chairs and 2 data hubs that are possible (ie. 1 to 1, 2 to 1, 2 to 2, 3 to 2). This should cover all significant cases that our system will encounter. The motivation behind this is that if we can robustly test cases including multiple chairs in a workspace, then we can say with better confidence that a full-scale system will function as designed.

Testing the FreeSeats system is broken down into three main categories of key performance indicators. The first is faithfulness to design, or in other words the Integration Correctness category. This set of measurements essentially indicate whether or not the FreeSeats system does what it set out to do, and with what degree of correctness does it perform. The term "correctness" here is used to indicate logical correctness, not adherence to the design requirements (which is a broader standard).

The second major category is latency. Timing benchmarks were be taken not only on the system-wide level, but also on the component level. The most important timing benchmark is the time from user activity to reflection of that activity in the web-app frontend. In other words, that is the time from a person sitting down to the app showing that person occupying that chair. However, in order to ensure that each component is performing as specified, we need to make sure that latencies on the component level are well-balanced and within expected range.

Lastly, the final category of testing pertains to power usage and performance. Since a large portion of our system is battery-powered embedded devices, we need to make sure that these will last long enough for a real use case, such as the installation of the chair modules in Sorrells. Battery life requirements can be found in the Design Requirements.

6.1 Results for Integration Correctness

Evaluating the correctness of the entire integrated system is one of the easier tests to run. By simply bringing in many volunteer users and having them sit in the chairs for any desired amount of time, we can note down the exact user behavior, and then reference the web application frontend to see what is actually being picked up by the system. Of course, this assumes that the testers have a standard understanding of the system's ideal output, which we can say with high confidence is the case.

The second, and perhaps more challenging, set of tests consist of multiple volunteers sitting in and getting out of many seats at once. This simulates a high-traffic workspace, which is something that we, as designers, often came back to as an optimal use case for the FreeSeats system. The data hub should collect data from its assigned workspace chairs, and propagate them through to the web application frontend, where the human tester can check the result against the real life context.

We ran the following trials five times each: idle chair registration, empty to sitting, sitting to empty, sitting to empty to sitting. In addition, we performed this set of tests on two of the three chairs (the third was not fabricated at the time of these tests). All correctness tests passed without issue.

In addition, we ran tests involving multiple chairs (chair A and B), each one three times: idle chair registration, sitting A, sitting B, sitting A + B, sitting to empty A, sitting to empty B, sitting to empty to sitting A, sitting to empty to sitting B. For each of these tests, we changed the state of one of the chairs while the other chair's state was randomly chosen (we either left it empty or occupied based on a random number generator). All multi-chair correctness tests also passed without issue, and we confirmed that the state of each chair was indeed independent of one another.

6.2 Results for End-to-end Latency

To evaluate if our system meets latency requirements, we performed multiple tests where we time how long it takes data to travel from one component to another. On a component level, we tested how long it takes from the time as user sits on a chair to when the message is sent

server > models > JS freeseats-model.js > ...

```
1   const mongoose = require('mongoose')
2   const Schema = mongoose.Schema
3
4   const Seat = new Schema({
5           _id: String,
6           occupied: Boolean,
7       }
8   )
9
10  const Hub = new Schema({
11          _id: String,
12          seats: {
13              type: Map,
14              of: Seat
15          }
16      }
17  )
18
19  module.exports = mongoose.model('seat', Seat)
20  module.exports = mongoose.model('hub', Hub)
21
```

Figure 3: Example schema of a single hub data object of the FreeSeats system.

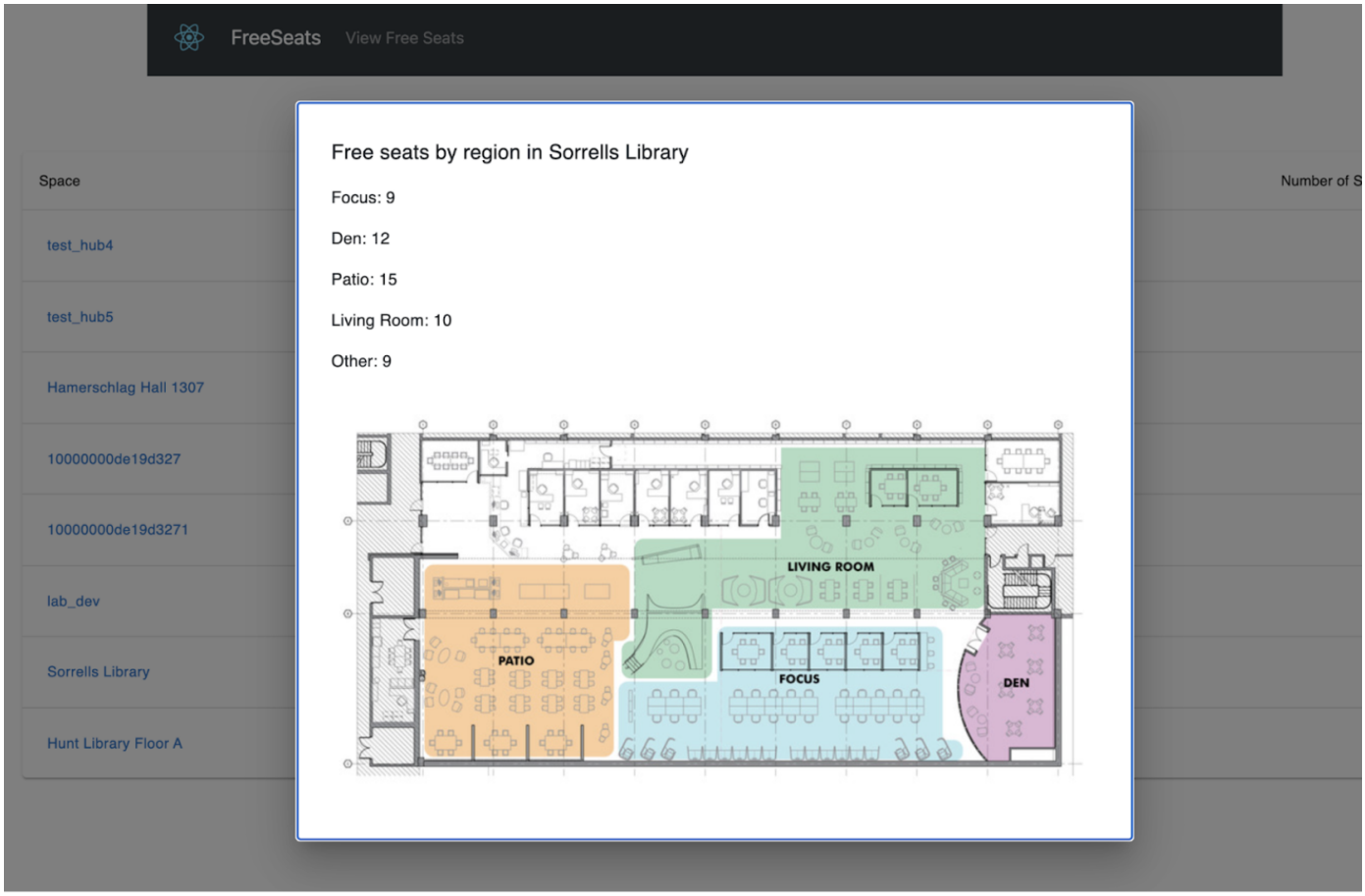

Figure 4: Front-end of FreeSeats

from the XBee in the Chair Module. This ensures that the Chair Module is processing data fast enough. We also timed how long it takes from when a Central Data Hub receives a Zigbee message from a Chair Module to when it transmits a WiFi message to the cloud-based web app. This ensures that the RPi processing is fast enough. Finally we performed testing in the software stack to ensure the database is populated quickly, the frontend and backend can communicate quickly, and that the frontend UI is quick and responsive.

The other latency tests we performed are end-to-end. This means we calculate how long it takes from the moment someone sits down or stands up from a chair to when that data is shown to a user in the web app. By meeting this requirement we ensure our entire solution meets all user needs in regards to latency.

Our first latency tests (chair to datahub) examined how long it takes for a change in chair state to register with the datahub. With a 2 second scan performed by the chair sensor every 5 seconds, we have the following results (averaged over 5 trials):
Initialization Time: 17.76s
Empty to Sitting: 7.23s
Sitting to Empty: 9.83s
The initialization time is much higher because the chair sensor, when powered on, requires a period of time to measure the idle capacitance and calibrate its baseline threshold off of those first few seconds. This feature was necessary to ensure that future users will not need to manually calibrate each and every sensor. Apart from that, the occupied and empty state changes fall well within the 15s update threshold, which meets our use case.

Additionally, we realized that we can arbitrarily set the update timers to be faster or slower, only at the cost of power usage. For instance, we reduced the polling cycle for the sensors to 2s for the final demonstration of FreeSeats, as we didn't want the viewers to wait such a long time for an update. As such, we are confident that we can easily meet timing requirements that are even more strict than the ones we set out initially.

6.3 Results for Power + Performance

The final metric to test is our power consumption and how long our batteries will last in the Chair Modules. We first used a multimeter to test the current draw and voltage levels coming from the batteries, during different phases of operation, such as when data is sent or when we are in sleep. We needed to ensure not too much current is being drawn, and adjust our code or components if necessary. The following table shows the average current draw under different conditions of sleep. These were measured under an interval of 4s for how often the Arduino wakes up and checks the capacitance and triggering a change in occupancy every 10s to have the XBee send data.

Condition	Average Current
No sleep	25.2mA
Arduino sleep, XBee no sleep	14.3mA
Arduino and XBee sleep	2.3mA

In addition, we investigated the tradeoff between power consumption and update latency. As mentioned in the previous subsection regarding latency, our system operates on programmer-defined update timers, meaning that changing the latency is as easy as typing in a new number. However, we have to keep in mind that a lower latency will result in higher power consumption. Our test results are shown in Figure 8 (in the Appendix).

7 PROJECT MANAGEMENT

7.1 Schedule

Refer to schedule attached at the end of document. We were on schedule for the duration of our project to produce and functioning MVP and demo by the due date of December 12th.

7.2 Team Member Responsibilities

Team member responsibilities is distributed as such: Foy leads the chair sensor portion, Cheng leads the data collection hub portion, and Ang leads the software stack portion. We have all agreed to helping each other on each part as necessary.

7.3 Budget

Refer to Bill Of Materials on Page 11. We have been careful to use cost-efficient materials for our use-case to stay within our budget. Each chair averages out to a cost of 30 dollars within the MVP stage. When we move to production, we can cycle to our own custom IC chips and make each chair much more cost-efficient.

7.4 Hosting Credit Usage

We used to Heroku to host our website. We chose to host in the cloud rather than directly on the RPis in the Central Data Hubs since there will be various hubs and we want one reliable system to serve the web app. Heroku provides free hosting services that we are using for our MVP.

7.5 Risk Management

We accounted for design, schedule, and resource (budget and personnel) setbacks that we encountered through the semester (e.g., fallback designs, risk reduction measures).

The first was shipping. We ordered parts and components are reasonable prices, and some took a long time to ship, due to the fact that some were shipped from overseas. For this reason, we planned to order parts at a reasonable

time, in mid October, so we had buffer time for shipping delays. Delays of up to a few weeks were not overly detrimental to our progress, as we were able to continue on portions of the project without some parts in possession.

Another was implementation setbacks. When we conducted research for the design portions, we had very little friction when it came to setbacks. The main risk we faced was in our sensing. We used a capacitance sensor using aluminum foil for a low cost, high accuracy approach. We recognized the risk brought up during advisor meetings and set slack time in our schedule to test this sensor implementation. This was sufficient risk management duration for implementation setbacks, and the sensor ended up working very well and was robust.

8 ETHICAL ISSUES

There are certain ethical issues that users may have concerns about. However, we hope to alleviate concerns by being transparent about the implementation of FreeSeats.

One concern that may arise is location tracking of users. However, we will not associate occupancy to any specific user or track user location - we will simply determine if a given chair is available or not. Another is privacy of weight. As users sit on the chair, they may be concerned that their weight is measured. However, we do not measure weight and simply calculate occupancy on at a binary granularity.

Our web stack will follow all appropriate and legal web protocols for receiving permission to collect cookies and tracking data if necessary.

9 RELATED WORK

A previous capstone solution, called Smart Library, attempts to solve the same problem as FreeSeats, which is occupancy detection and broadcasting [1]. Their approach was to use cameras and machine learning to detect occupancy in a space. Their metrics were: Latency of 1 minute, accuracy of 76%, and battery life of 72 hours. Our approach with physical chair sensors will improve on all aspects (latency by 45 seconds, battery life of 2 months, and accuracy of at least 1%, as 76% is reasonable according to users of the application).

There is a commercial application named Density. It uses radar emmisions to calculate density of a room in order to then process occupancy in a space. Their metrics and benchmarks are proprietary and not disclosed [4]. Other applications use IR waves, thermal detection, or Wifi-device detection to determine occupancy, but fall short due to accuracy and privacy concerns [5].

10 SUMMARY

10.1 Future Work

We plan on continuing working on this project after the semester ends. Our goal is for this project to actually be implemented and help everyone save time and effort when going to places. After the semester ends, we will turn the MVP into a more polished solution and extend it to larger study spaces. Eventually, we see FreeSeats in any application that faces a capacity issue.

10.2 Lessons Learned

We have learned many valuable lessons throughout the Design portion of the project. The first, is that design takes much longer than we had expected. We believe this could only be learned first-hand through experience, which is what we did. The design process is very creative and iterative in nature, which is why we spent half the semester devoted to it.

We also learned that our focus as solution-makers should be on the customer problem. We should not choose a technology simply because we want to, but it is a must that it is driven by the fact that it would be the best solution for the problem at hand, or at least has high potential to be. This was a significant part of our design process and a key lesson learned.

Glossary of Acronyms

- RPi – Raspberry Pi

11 References

[1] Raguram, Arjun, et al. "Team B4: Smart Library." Team B4 Smart Library, 20 Dec. 2020, http://course.ece.cmu.edu/ ece500/projects/f20-teamb4/.

[2] Comparative Study of Communication Interfaces for Sensors and Actuators in the Cloud of Internet of Things - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Power-Consumption-Test-for-Bluetooth-ZigBee-and-Wi-Fi$_t$$bl1_3$18054538

[3] Stoffregen, P. (2020, August 20). CapacitiveSensor Library. computer software. Retrieved from https://github.com/PaulStoffregen/CapacitiveSensor.

[4] "People Counting Sensors amp; Innovative Software for People Counting." People Counting Sensors; Innovative Software for People Counting, https://www.density.io/people-counting-solutions/technology-sensors-software.

[5] Farah, Andrew. "7 Technologies That Count People." Medium, Density, 9 Oct. 2020, https://medium.com/density-inc/7-technologies-that-count-people-buildings-offices-742785d2030f.

12 Appendix

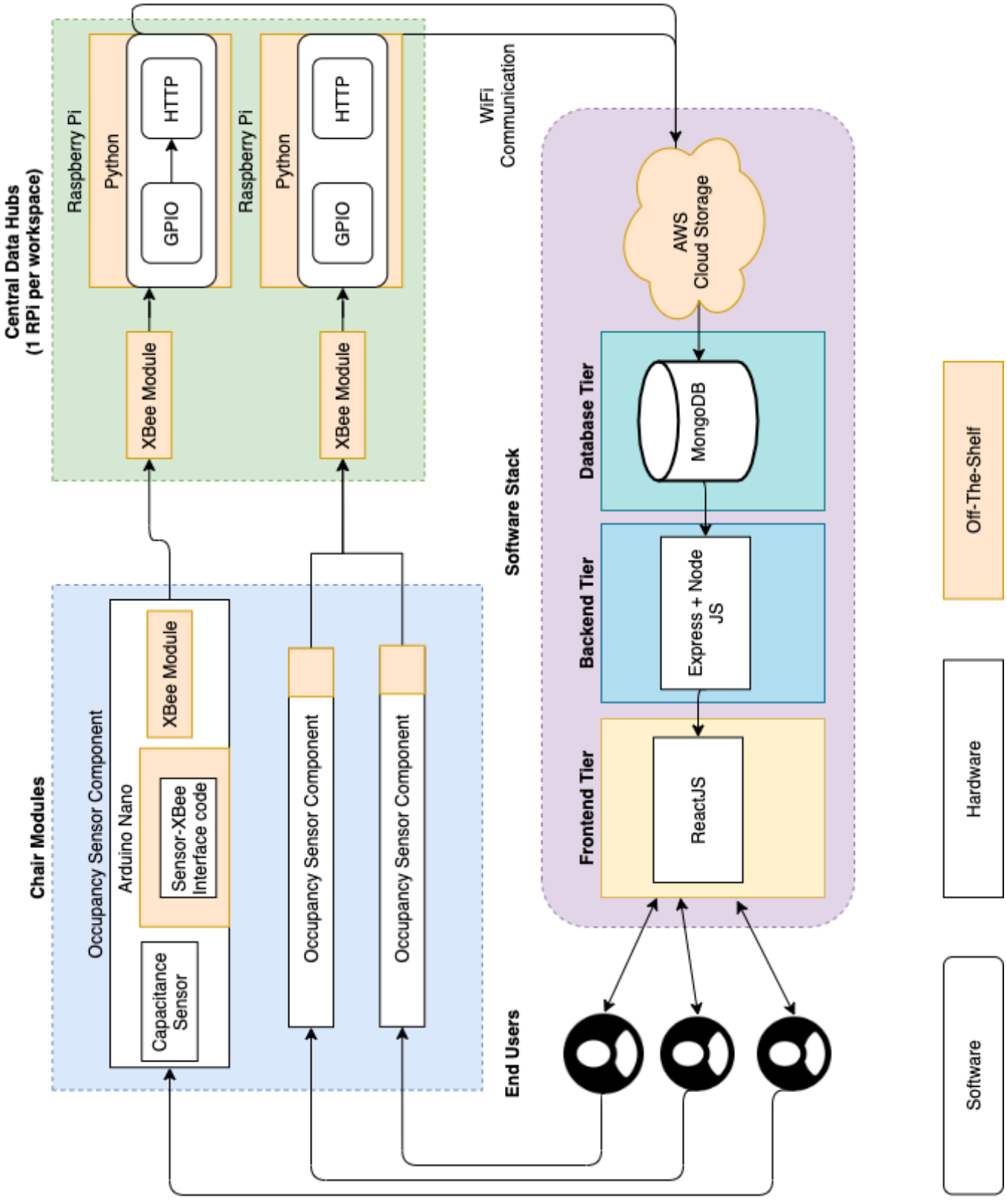

Figure 5: A full-page version of the same system block diagram as depicted earlier.

Bill Of Materials

Item	Model	Manufacturer	Price	Quantity	Total
XBee Module	V3, PCB Antenna	DigiKey	$23.26	5	$116
Raspberry Pi	V4, Starter Kit	Raspberry Pi	-	2	-
Arduino Pro Mini	3.3V/8MHz	Sparkfun	$9.95	3	$29.85
LiPo Battery	2000mAh, 3.7V	Sparkfun	$12.95	3	$38.85
Battery Charger	LiPo USB-C Charger	Sparkfun	$10.50	1	$10.50
Capacitive Sensor	Aluminum Foil	Amazon	$3.79	1	$3.79

Figure 6: Bill of Materials

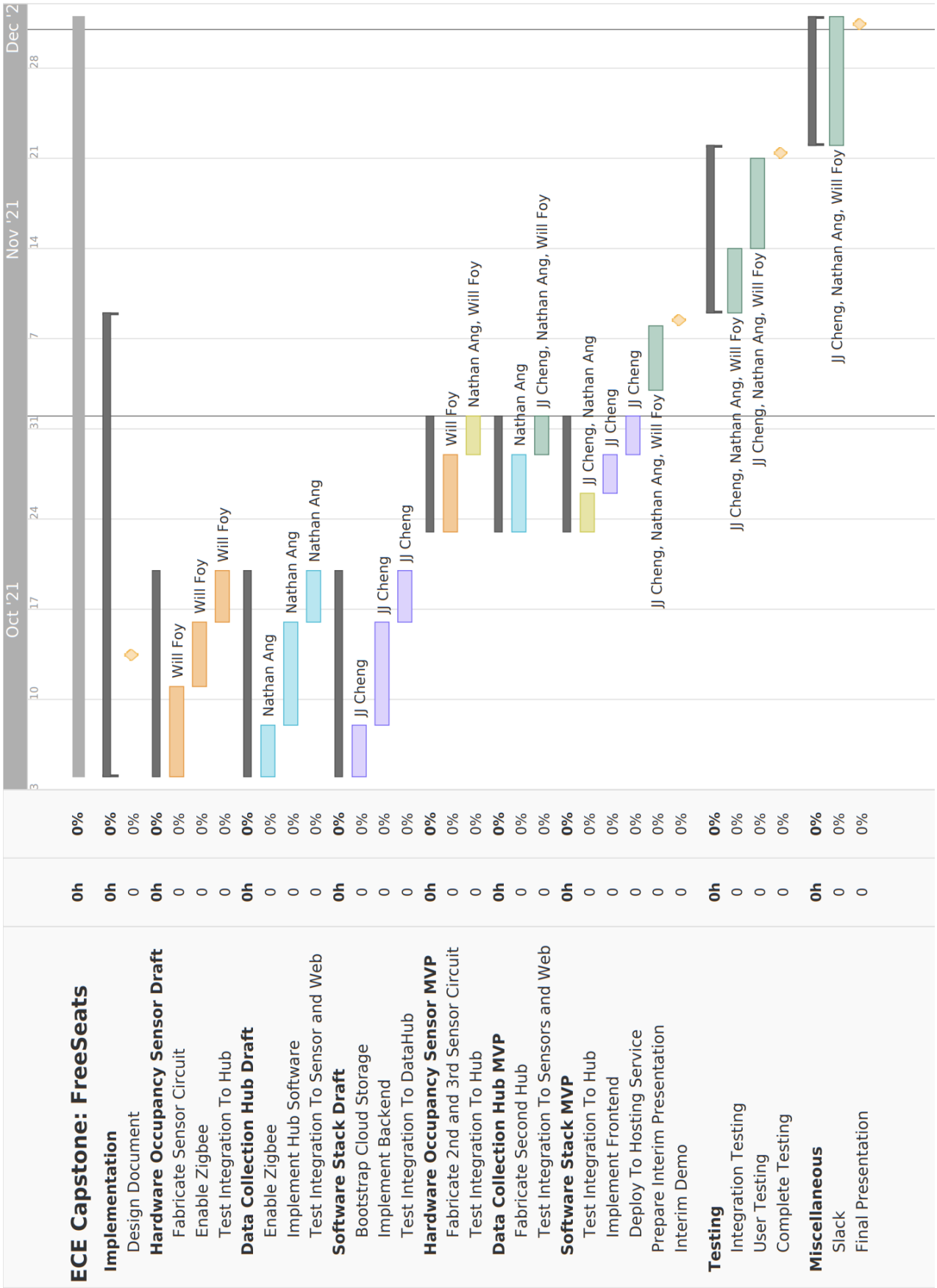

Figure 7: Milestone and Schedule Chart

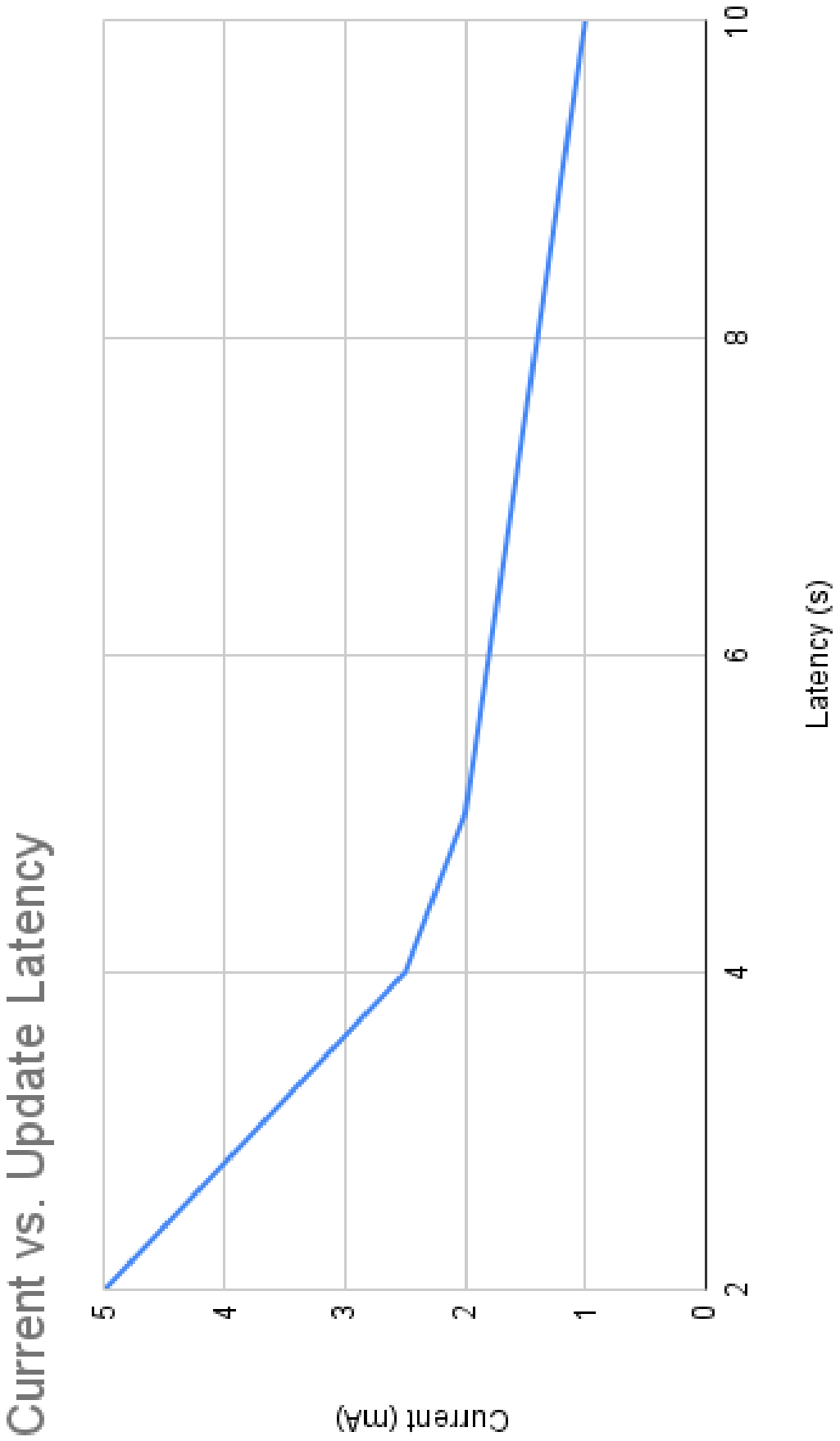

Figure 8: Paredo curve illustrating the tradeoff between power consumption and system update latency, on the hardware level.