

# **EE202C**

# **Networked Embedded Systems Design**

**Hospital Security & Contamination  
Prevention**

**Design Document**

**Salma Elmalaki**

**Raymond Andrade**

**Anthony Nguyen**

**Pranjal Rastogi**

**Version 2**

**12-10-15**

## **Revision History**

<b>Version</b>	<b>Date</b>	<b>Author</b>	<b>Comments</b>
Version No	Date Completed	Author(s)	Comments on Versions
1	10-10-15	Salma/Raymond/Ant hony/Pranjal	First draft and ideas for the project
2	12-10-15	Salma/Raymond/Ant hony/Pranjal	Final design document for the project

## **Table of Contents**

<a href="#">1 Abstract</a>	
<a href="#">2 Team</a>	
<a href="#">2.1 Salma</a>	
<a href="#">2.1.1 Main focus</a>	
<a href="#">2.1.2 Detailed contribution</a>	
<a href="#">2.2 Raymond</a>	
<a href="#">2.2.1 Main focus</a>	
<a href="#">2.2.2 Detailed contribution</a>	
<a href="#">2.3 Anthony</a>	
<a href="#">2.3.1 Main focus</a>	
<a href="#">2.3.2 Detailed contribution</a>	
<a href="#">2.4 Pranjal</a>	
<a href="#">2.4.1 Main focus</a>	
<a href="#">2.4.2 Detailed contribution</a>	
<a href="#">3 Technical Approach</a>	
<a href="#">3.1 Localization system</a>	
<a href="#">3.2 Secure communication system</a>	
<a href="#">4 System Design</a>	
<a href="#">4.1 Localization System</a>	
<a href="#">4.1.1 IR data emission</a>	
<a href="#">4.1.2 IR data reception</a>	
<a href="#">4.2 Secure Communication System</a>	
<a href="#">4.2.1 Threat model and assumptions</a>	
<a href="#">4.2.2 System guarantees</a>	
<a href="#">4.2.3 Goals</a>	
<a href="#">4.2.4 Services</a>	
<a href="#">5 Implementation Schedule</a>	
<a href="#">6 Implementation Description</a>	
<a href="#">6.1 Localization</a>	
<a href="#">6.1.1 Infrared Transmitter Implementation</a>	
<a href="#">6.1.1.1 Transmitter Hardware setup</a>	
<a href="#">6.1.1.2 Algorithm and Details of Implementation</a>	
<a href="#">6.1.2 Infrared Receiver Implementation</a>	
<a href="#">6.1.2.1 Receiver Hardware setup</a>	
<a href="#">6.1.2.2 Algorithm</a>	
<a href="#">6.1.2.3 Notes on Algorithm Implementation</a>	
<a href="#">Sampling</a>	
<a href="#">Consolidating the raw data[ ] to populate the values[ ]</a>	
<a href="#">Inferring the Bits and the Location from the values[ ]</a>	
<a href="#">6.2 Secure communication system</a>	
<a href="#">6.2.1 Design architecture</a>	
<a href="#">6.2.2 Architecture model</a>	
<a href="#">6.2.2.1 multiple clients - single server</a>	
<a href="#">6.2.2.2 ready/read message - monitor flag</a>	
<a href="#">6.2.2.3 server shutdown</a>	
<a href="#">6.2.2.4 secure communication protocol</a>	
<a href="#">6.2.3 Communication protocols</a>	
<a href="#">6.2.3.1 The main communication</a>	
<a href="#">6.2.3.2 The control communication</a>	
<a href="#">6.2.3.3 The services communication</a>	
<a href="#">7 System Source Code</a>	
<a href="#">7.1 Localization System</a>	
<a href="#">7.2 Secure communication system</a>	
<a href="#">7.2.1 Server</a>	
<a href="#">7.2.2 Client with random location for unit testing</a>	
<a href="#">7.2.3 Client integrated with IR receiver</a>	
<a href="#">7.2.4 Services</a>	
<a href="#">7.2.5 GUI</a>	
<a href="#">8 Test and Evaluation System and Testing Results</a>	
<a href="#">8.1 Hardware Testing</a>	
<a href="#">8.1.1 IR Emitters (LEDs)</a>	
<a href="#">8.1.1.1 Procedure</a>	
<a href="#">8.1.1.2 Results</a>	
<a href="#">8.1.2 IR Receiver (Diodes)</a>	

## Design Document

- [8.1.2.1 Procedure](#)
    - [8.1.2.2 Results](#)
  - [8.1.3 MOS Power Driver Kits](#)
    - [8.1.3.1 Procedure](#)
    - [8.1.3.2 Results](#)
  - [8.1.5 Demo Hardware Assembly](#)
    - [8.1.5.1 Procedure](#)
- [8.2 Software Testing - IR Transmit/Receive](#)
  - [8.2.1 IR Emitters \(LEDs\)](#)
  - [8.2.2 IR Receiver \(Diodes\)](#)
- [8.3 Software Testing - GUI](#)
- [8.4 Software Testing - Server](#)
  - [8.4.1 Server-Tag](#)
  - [8.4.2 Server-GUI](#)
- [8.5 System integration](#)
- [8.6 Full System Operation](#)
- [9 Test System Source Code Repository](#)
- [10 Analysis and Further Work](#)
  - [10.1 Localization](#)
    - [10.1.1 Receiver Line of Sight](#)
    - [10.1.2 Enhancing Responsiveness](#)
    - [10.1.3 Enhancing Cost Effectiveness](#)
    - [10.1.4 Considerations for Physical Installation](#)
    - [10.1.5 Other Known Issues](#)
    - [10.1.6 Intel Edison SparkFun Breakout Blocks](#)
  - [10.2 Secure communication system](#)
    - [10.2.1 Communication delay](#)
    - [10.2.2 Number of clients](#)
- [11 Shared Systems](#)
- [12 References](#)
  - [12.1 Hardware References](#)
    - [12.1.1 IR System Components](#)
    - [12.1.2 Intel Edison](#)
  - [12.2 Software References](#)

# 1 Abstract

Hospital contamination is a real life problem. Ensuring that the hospital is a safe and healthy environment is not an easy task as it seems. Occupants in the hospital can be patients, visitors and physicians and all of these occupants are susceptible to contamination. Problems happen when a patient is diagnosed with a contamination disease. The need to track down all the encounters with this patient is essential. In our project, we aim at building a prototype system that can tackle the problem of locating the contaminated patient and all the encounters who can be possible carriers. The identity and the location of the hospital occupants should be handle in a secure way in order to protect the privacy of the people involved. Our prototype system manifests the Infrared sensors to achieve localization using tags worn by all the occupants in the hospital. These tags localize themselves and send their information in a secure way to the main hospital server. The main server is a trusted entity that keep track of all the identities and the tracks of all the tags. We provision that these kind of systems will be scalable and applicable in real life to help hospitals protect against contamination spread.

## 2 Team

### 2.1 Salma

#### 2.1.1 Main focus

Communication protocols and the security of transmission including the implementation of server/clients and the algorithms running on the server to detect the possible contamination carrier as well back-end services.

#### 2.1.2 Detailed contribution

1. Set the threat model and the system guarantees of the system.
2. Designed and implemented the back-end asynchronous dispatch server along with the client code to handle the communication protocol in a secure way.
3. Designed a communication protocol to maintain the integrity of the information by acknowledgment scheme between the server and the clients.
4. Designed and implemented the server services including:
  - a. Tracking map for all the tags
  - b. Protected zone handling
  - c. Encounters history for contamination
5. Designed and implemented three layers of communication between the server and the clients:
  - a. Main protocol to communicate the location between the clients and the server
  - b. Control protocol for updating/controlling the communication scheme between anchors and tags
  - c. Services protocol to handle the server services and communicating the actions needed to the tags (ex. alerting a violating tag in a certain location)
6. Worked on integrating the server code and the client code with the IR receiver code.
7. Worked on integrating the server code with the GUI.
8. Added support of the services in the GUI.
9. Maintained a special data structure on the server that can hold the time series of the locations for each individual tag in a way that makes the query fast. This database is

updated at runtime in parallel thread in order not to harm the responsive of the server.

## **2.2 Raymond**

### **2.2.1 Main focus**

Infrared communication algorithms, hardware design, and GUI development.

### **2.2.2 Detailed contribution**

1. Created GUI and was responsible for debugging and scaling of the GUI.
  - a. Created the algorithms necessary for multi-threaded tracking of tags.
  - b. Designed and refined the algorithms used to display tags on the GUI.
  - c. Created the menu bars which allowed for simple integration of features into the GUI.
2. Wrote the original IR emitter/receiver algorithms.
  - a. Collaborated with group members to expand those algorithms to what was implemented during the final project.
3. Designed the MOSFET driver circuits and was responsible for debugging issues involving either the emitter or receiver circuitry.
  - a. Experimented with different driving currents to establish the appropriate design for the project.
  - b. Discovered and resolved hardware issues that were causing the IR communication to become unreliable.
  - c. Wrote a C code testbed for the MOSFET drivers, emitters, and receivers.
    - i. This allowed for the observation and confirmation, via Oscilloscope, of two critical system aspects.
      - The first being the inability of the IR receivers to operate correctly when too close to the emitters(this distance depends on emitter output power).
      - The second was the interference between two transmitted data packets corrupting the received signal. Making understood the importance of the design tradeoff between the emitter output power, and the spacing between the emitters.
4. Worked on the integration of all aspects of the project.
  - a. Collaborated with group members to get the server to receive a tag's location in real-time.
  - b. Collaborated with group members to get the GUI to display the values received by the server in real-time.

## **2.3 Anthony**

### **2.3.1 Main focus**

Working on the IR Transmitter Receiver Code. MCU development to increase system performance. Preventing Replay attacks using Variable Length Preamble. SparkFun Blocks to improve system portability.

### 2.3.2 Detailed contribution

1. Exploration of database services appropriate to store and maintain the private location data that would be collected by
  - a. This contribution was not relevant to our demo because we did not develop any features that require tracking over a long period of time. This could be discussed further under Future Work.
2. Extensive collaboration with other group members developing the IR emitter/receiver algorithms.
  - a. Collaborated with group members to expand those algorithms to what was implemented during the final project.
  - b. Individually added components to allow the system to dynamically change the IR message transmitted between Emitter and Receiver, preventing replay attacks.
3. Development of System Scalability optimizations
  - a. Developed an automated method to add new Tags and Anchor Nodes into our system so that the server is aware of them.
    - i. Key to this was learning how to configure programs on the Edison to run upon boot time to allow for non-interactive configuration
4. SparkFun Development
  - a. Performed investigations into using the GPIO pins on the SparkFun breakout boards. Learned the Linux convention to set up, enable and disable the pull-up and pull-down resistors on the Edison GPIO pins. Configure Multiplexers on the SparkFun blocks to enable PWM output.
5. MCU Development
  - a. Developed code based on tutorials provided by Chris that would allow for greatly shortened IR message length in the hundred-microsecond range
  - b. Code was developed as a side feature but we did not complete integration with the main system.
6. Worked on the integration of all aspects of the project.
  - a. Collaborated with group members assemble the hardware portion of the demo so that we could transmit the values to GUI on the server in real-time.

## 2.4 Pranjal

### 2.4.1 Main focus

Infrared Communication Protocol Design and Software Development, System Integration and Testing, Software Service Implementation

### 2.4.2 Detailed contribution

1. Designed and developed the Infrared Communication Protocol
  - Initially proposed Infrared Communication as the suitable localization technology to achieve our project mission
  - Came up with the appropriate preamble and bits patterns for our protocol, in consultation with other team members
  - Designed and Implemented the Infrared Receiver Algorithm which is being used for project currently
  - Helped in debugging bugs in the transmit code
2. Worked on enhancing the responsiveness and robustness of the smart tags
  - Designed and implemented scheme for improving robustness of the IR communication towards outliers.
  - Designed and implemented scheme for having a more resilient smart tag and overcome

## Design Document

- the line of sight problem
- Implemented the scaling factor for bringing down the transmission duration, thereby making the smart tag more responsive
- Rigorously tested the IR communication to ensure robust operation
- 3. Worked on hardware assembly for the final demo
  - Helped in setting up the SparkFun boards for driving the infrared Leds
  - Did experimentation in collaboration with Raymond to figure out the correct spacing between the Leds for the purpose of our demo
  - Debugged issues related to assembly and system integration
- 4. Collaborated with team members in integration of GUI and the smart tag
- 5. Developed security scheme for averting jamming attack on the smart tags
- 6. Collaborated with team members to identify security threats in Infrared Communication and helped in testing the security aspects
- 7. Designed and developed software for implementing softwall service for our project. This service enables the doctor to track spread of contamination from an infected person to others. This service is the key utility to aid the doctors in averting hospital acquired infections



### 3 Technical Approach

Hospital security needs improvement. Anyone who walks into a hospital can move freely throughout the entire hospital with being validated or tracked. With the emergence of IoT platforms there is a convenient way to solve this problem and several more. Being able to deploy tags that can communicate securely with a server that can track its location throughout the hospital will not only allow the hospital to monitor the activities of those who walk into the hospital, it can also be used to ensure the safety and security of the people who visit the hospital. This IoT system can keep certain people out of areas they should not be and to ensure swift action to quarantine anyone who could have been in contact with a contagious individual.

The system is divided into two main components - **the localization system** and the **secure communication system**.

#### **3.1 Localization system**

This system consists of four nodes, which drive four infrared emitters that correspond to a particular section of a room. Each emitter will be sending a location data packet that includes its node's ID along with its emitter ID. These two pieces of information combined will establish the unique location of the emitter. Meanwhile, a person who has a tag will be walking around a room or floor. The tag is equipped with IR receiver that is able to receive the emitted data packet and will save the received location into a file. A central server will then securely and wirelessly read the location from that file, and will utilize that form of communication to both the nodes and tags to execute a secure communication protocol.

The limitation to this approach is the emission of the IR data packets due to the restricted area that is covered by the IR emitters. Although they provide a highly localized solution, they will require a system design that will have a sensitive tradeoff between their emitted power and distribution.

#### **3.2 Secure communication system**

The second part of this project is the secure communication. The system uses the information that is retrieved from the localization system, namely the whereabouts of the visitors and workers of the hospital. It keeps track of the path of everyone and is able to know when an employee or visitor walked into an area that they are not cleared to enter. If a patient who walks into the hospital, is given a tag, and is later deemed to be contagious, the system is then used to flag the path of that patient, and determine who had crossed paths with that patient, which ultimately leads to a precise way of quarantining exposed people.

The expected limitation to handle communication securely is the timing delay and the responsiveness of the system. There is a tradeoff between the throughput of communicating the data directly between the entities in the system, and to handle them securely.

## 4 System Design

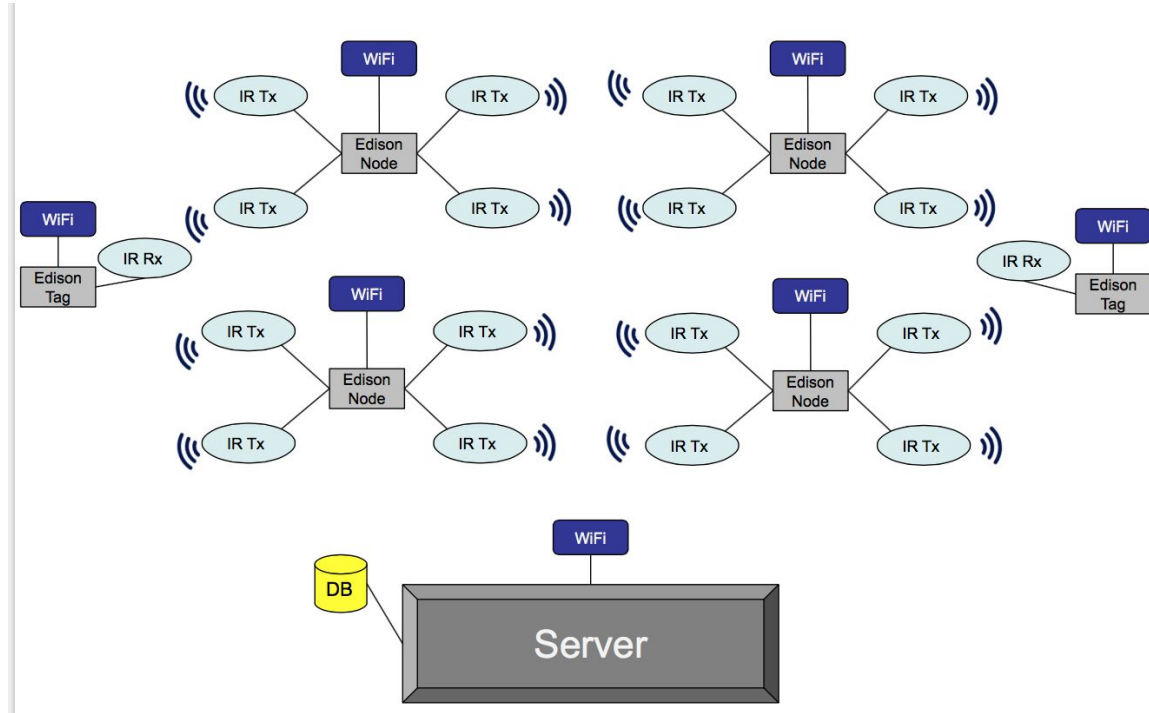


Figure 1

The project system design depends on a continuously running server that receives location updates from tags and act accordingly. The tags determines their location using IR based localization scheme. All the system components are interacting with each other through WiFi. The system design view is shown in Figure 1 in which Edison boards act as nodes as well as tags. A server maintains a database for the location of tags in order to facilitate the query for any information needed later.

### 4.1 Localization System

To achieve localization, our system executes effective emission and reception schemes of infrared location packets.

#### 4.1.1 IR data emission

Our system design initially called for every node to emit a data packet **one at time** in round robin fashion. After experimentation, it was found that this method yielded a large delay in tracking a tag's transition from one location to another. After this observation, our emission scheme was altered to have every emitter sending its location **all the time**, thus greatly improving upon the previously observed transition delays.

The data packet itself consisted of **three** parts:

1. **Preamble** - Used to identify the **beginning of a message**
2. **Edison(Node) ID** - Used to identify a **sub location within a room or floor**

3. **Emitter number** - Used to identify a **specific location within a sub location**

#### 4.1.2 IR data reception

The method of receiving the emitted packets consists of **sampling** the output of the receiver, and the implementation of a **FSM to extract a location** from the collected data.

The sampling has **two** stages:

1. Sample **raw** data of 1's and 0's, and store those into an array
2. Sum the **consecutive** 1's or 0's, and store their sums into an array

The FSM is then executed as follows:

1. Takes in the array of consecutive summations
2. Looks for the preamble
3. Looks for the edison/emitter ID's, the upper and lower nibble of the location respectively

For added security the emitter and receiver are securely told by the server what the length of the preamble should be. This length is generated at random, and was implemented to prevent a replay and jamming attack.

### 4.2 Secure Communication System

To ensure the security of communication between the server and the tags to maintain the privacy of the location information of the tags, we set the threat model and the system guarantees as well as the system goals. As shown in Figure 2 we want to maintain a secure communication between the server and all the tags.

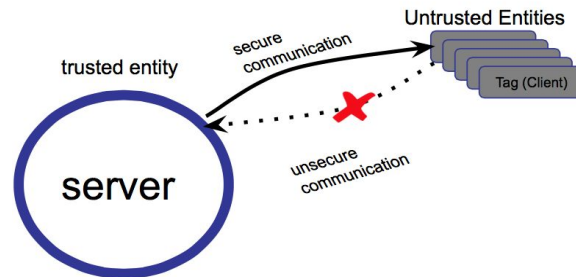


Figure 2

#### 4.2.1 Threat model and assumptions

1. We assume that the server is a trusted entity
2. Server has root access to all the Tags available
3. Server has root access to all the Anchors available

#### 4.2.2 System guarantees

1. Server is the only entity that can keep track of the information and the contamination information
2. Tags have no access to the server

### 4.2.3 Goals

1. **Accuracy.** The data received should be accurate. This means a tag should not identify itself in a wrong location.
2. **Privacy.** The location of the tag should not be known by any entity other than the server to maintain the privacy of the track information for any person in the hospital
3. **Integrity.** The communication protocols should maintain the integrity of the data sent and received to and from the server. The system should ensure that no loss of information takes place.

### 4.2.4 Services

1. **Contamination encounters (Softwall detection):** The server has the information of the contaminated tag and its location. The server should be able to track down all the tags that encountered the contaminated person in order for the hospital to react accordingly.
2. **Secure access control:** A protection zone service is very useful in a hospital settings. If a tag enters a room that it doesn't have privilege to enter, the server should react. This can be done by comparing the tag IDs within this room across the IDs that have privileged access to it. The information of the privileged tag IDs for each room is kept on the server. Afterwards, an alert is sent to the violating tag to inform the owner of the tag that this area is restricted.
3. **Tracking:** Tracking all the tags in the hospital is essential. One direct application for this tracking is to track patients who should have a daily schedule in the hospital. For example, going to specific room every day and then go to another and so on. This will help the hospital be sure that all the patients are following the right schedule during the day.

According to these requirements from the secure communication system, three layers of communication are necessary to achieve the goals as well as the guarantees:

1. **Main protocol.** This layer is responsible for communicating the location to the server from all the tags asynchronously. Hence, whenever the location of a tag changes, the server should be notified and the location of the tag is updated in the database maintained by the server.
2. **Control protocol.** This layer is responsible for controlling the communication pattern between the anchor nodes and the tags. As previously mentioned, the anchors hold IR transmitter and the tags hold IR receiver, the communication pattern between the two is determined by the server. The server should be able to access the nodes and the tags and update the IR communication acknowledgment pattern.
3. **Service protocol.** This layer is responsible for managing the services maintained by the server. The server should be able to communicate to the tags to alert it in case of contamination or a protected zone breaching.

## 5 Implementation Schedule

This is a critical step. For this section please include dates for:

1. Design completion date - 10/16/15
  - Should have final system architecture in place
2. Test plan completion date - 10/21/15
  - Should have tested sensors and understand how to properly distribute them for localization system
  - Should have established secure communication between nodes and server
3. Design review date of Midterm in Week 6 **to include demonstrations of system operation**
  - Must have the localization aspects and ssh communication aspects finished at least. This will allow our group to show the ability for our system to implement security and contamination prevention.
  - This should be finished by: 10/28/15
4. Testing phase complete - 11/11/15
  - Should have our server able to track tags in real time and able to implement contamination prevention
5. Implementation completion date - 11/25/15
  - Should have full systems complete and ready to scale security features

## 6 Implementation Description

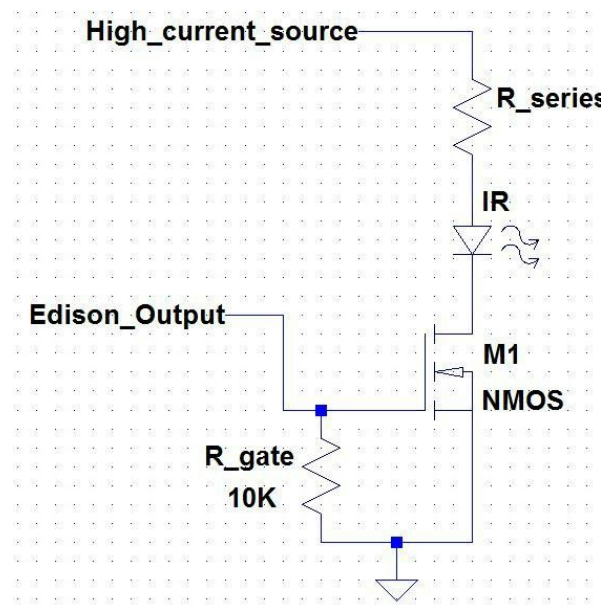
### 6.1 Localization

Ours is a localization scheme which makes use of fixed anchor nodes which deploy infrared communication with the smart tags. Once a smart tag receives a location information from an anchor node, it sends that to the server. We will focus on the details of the infrared communication in this section.

#### 6.1.1 Infrared Transmitter Implementation

##### 6.1.1.1 Transmitter Hardware setup

The circuit that needs to be setup on the transmitter side is shown below -



The current through the IR LED was adjusted by choosing a suitable resistor, such that the current was less than its rated current. Also the ground of the LED circuit should be connected to the ground of the Edison for the circuit to work. The output of the driver pwm pin of the Edison was connected to the gate of the MOSFET.

##### 6.1.1.2 Algorithm and Details of Implementation

The algorithms at the transmitter end is simple and entails transmitting the preamble and the bit pattern corresponding to the edison ID and the LED ID. The preamble comprises of transmitting a HIGH followed by a LOW and this pattern repeated for a number of times. The number of times the pattern is repeated is controlled by setting the variable preamble\_length. By default the preamble length is set to 5. Please note the definition of HIGH and LOW as stated below -

HIGH - Infrared light modulated at 38 KHz

LOW - No Infrared light at all

The bits are transmitted after the preamble is transmitted. In our IR protocol, the IR signal corresponding to the Bits is as indicated below -

Bit 1 = 20 ms HIGH followed 5 ms LOW

Bit 0 = 5 ms LOW followed by 20 ms HIGH

The durations indicated above correspond to scaling factor of 1. So for scaling factor of 1 and preamble length of 5, total duration of one message will be:

$5 \times 20 \text{ ms (for the preamble)} + 25 \times 4 \text{ (for the bits)} = 200 \text{ ms}$

If we were to set the scaling factor to 5, the duration of transmission with preamble length of 5 will become =  $2000 \text{ ms} / 5 = 400 \text{ ms}$

We have tested scaling factor of 5 and it is working fine. While going to scaling factors more than 7, we ran into reliability issues. So for further reduction in transmission duration, we suggest using onboard MCU of Edison.

### 6.1.2 Infrared Receiver Implementation

#### 6.1.2.1 Receiver Hardware setup

The TSOP infrared receiver was plugged on to the arduino breakout board for Edison with the help of wires. This served as a prototype for the smart tag.

#### 6.1.2.2 Algorithm

1. Sample the signal output from the Infrared receiver. Store the samples (0's or 1's) in an array `raw_data[ ]`
2. Count the number of consecutive 1's and 0's in the array `raw_data[ ]` and store the counts in `values[ ]` array
3. Detect the position where the preamble ends in the `values[ ]` ( The preamble is known to be - 20 1's followed by 20 0's - this pattern repeated 6 times ).
4. Read the 8 values from the `values[ ]` array, after the preamble and infer the transmitted bits from these values. A 10 followed by a 40 corresponds to a bit 0 and a 40 followed by a 10 corresponds to a bit 1
5. From the 4 bits inferred, the first 2 correspond to the Edison ID and the remaining 2 correspond to the LED ID

#### 6.1.2.3 Notes on Algorithm Implementation

### Sampling

The code snippet where the sampling is done is shown below -

```
while(samples_remaining > 0 ) {  
    raw_data[i] = mraa_gpio_read(gpio);           // This is where value of gpio pin is read  
  
    for (j = 0 ; j < SAMPLING_RATE ; j++);        // This for loop implements the sampling delay  
        i++;  
    samples_remaining--;  
}
```

We have used IDLE for loops to implement the sampling interval. The constant `SAMPLING_RATE` is defined at the beginning of the program and another constant `SCALING_FACTOR` is used to alter all the delays in the program by same factor.

### Consolidating the `raw_data[ ]` to populate the `values[ ]`

The `raw_data[ ]` array after sampling has 1's and 0's like shown below -

111111111111111111000000000000111111111111110000000000000000 .....

We can count the number of consecutive 1's and 0s and use them to populate the `values[ ]`. The `values` array will look like the following at the end of this operation -

16,13,16,17, 21, .....

### Inferring the Bits and the Location from the values[ ]

We know that the preamble is supposed to 20ms HIGH followed by 20ms LOW and this pattern shall be repeated number of times which is decided by the user input or the variable preamble\_length ( which is set to 5 by default ). So a Preamble should ideally be represented as 20,20,20,20,20,20,20,20,20,20 ( in case the preamble length is 5). But due to variabilities in pwm pin response time and other sources of errors, what we end up receiving is something like 16,17,19,22,23,21,22,19,20,22. We should infer the above pattern as Preamble. So we need to set thresholds for detecting a 20 in preamble ; let's say we decide that any value in values[ ] array which is between 15 and 25 could be considered as a 20. With this scheme in place, we can infer a preamble out of the values [ ] array.

Once a preamble has been inferred, we consider the next 8 values in the values array. The correspondence between the bits and values is as follows -

1 - 40,10 ( in raw\_data[ ] array it will be 40 1's followed by 10 0's )

0 - 10,40 ( in raw\_data[ ] array it will be 10 1's followed by 40 0's )

While examining the 8 values after the preamble, if any value is found to be in nonconformance with the above scheme, we will discard the entire message.

For example, consider that the following 8 values are present in the values [ ] after a preamble has been detected - 38,13, 37,12, 8, 35, 12, 39. In this, if we consider 2 values at a time we can infer the following 4 bits - 1 1 0 0. Note that any value between 35 and 45 will be considered to be 40 and any value between 5 and 15 will be considered to be 10. Once we get the 4 bits, we can correctly the location from them.

## **6.2 Secure communication system**

### **6.2.1 Design architecture**

The implementation of the secure communication system manifests a flavor of the 'Producer-Consumer' design pattern. As shown in Figure 3, the clients, which are the tags in our scenario, act as a producer and the server acts as a consumer. The producer produces new location data and write it in a local file which will be the shared resource in the producer-consumer pattern. The consumer then reads (consumes) the data in the file. To maintain the monitor flag on the shared resource, a 'ready' message is sent from the producer to the consumer whenever a new data is produced and then a 'read' message is echoed back from the consumer to the producer whenever the consumer reads (consumes) the new data.

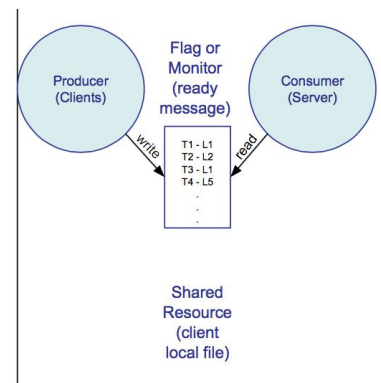


Figure 3

### **6.2.2 Architecture model**

#### **6.2.2.1 multiple clients - single server**

The first model that the design architecture should support is managing multiple clients by one server. These clients represent the tags carried by the people in the hospital. Hence, it is crucial that that the server is able to handle communication and data coming from multiple clients. In order to achieve this, the server is designed to be asynchronous dispatch server. The server asynchronously read the new data from each client whenever a new data is 'ready' in a separate thread. Whenever a server receives a 'ready' message from any client, it dispatches a new thread to handle this



communication and reads the new data. The threads are dispatched asynchronously and are not interfered with each other.

### 6.2.2.2 ready/read message - monitor flag

The ready/read message are very important to make sure that the consumer receives the data whenever the producer generates a new data. To implement this, a TCP/IP socket packet is used with a 'ready' message in it and sent to the server. When the server receives the 'ready' message, it dispatches a new thread and read the information from the client. After the server finishes reading the new data, it echos back a 'read' message in the same socket connection. When the client receives the 'read' message, it empties this data from its local buffer which is the local file maintained at the client. This way we ensure no loss of information.

### 6.2.2.3 server shutdown

The system should continue working even if the server is shutdown. This is handled by not emptying the data unless the 'read' message is received by the client. This means that if the server is powered off then the client will append every new locations in its local file (buffer) and when the server comes on again, the server will read all the buffer at once. This ensures that no data is lost even if the server is powered off.

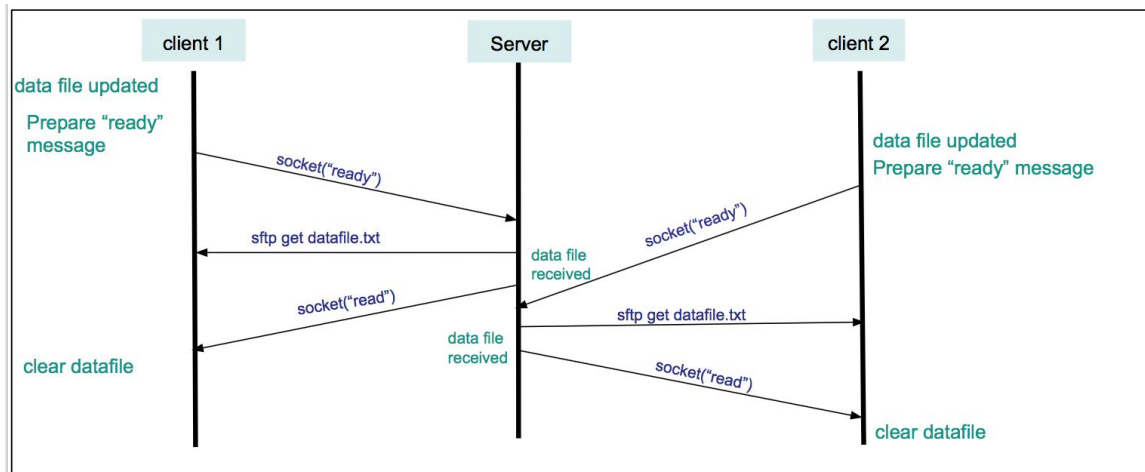
### 6.2.2.4 secure communication protocol

The localization is done by transmitting the IR message from anchors to the IR receiver on the tags. This message has certain field to mark the beginning of the message so that the IR receiver knows how to interpret the location. In order to add a level of security, the server continuously updates this field in the transmitter (anchors) and the receiver (tags) so that it becomes hard for an attacker to learn the transmission protocol. More details on how this field is changing the communicated message between the transmitter and the receiver are explained in the implementation of the localization part of the project.

## 6.2.3 Communication protocols

The server handles three layers of communication namely - the main communication, the control communication and the service communication.

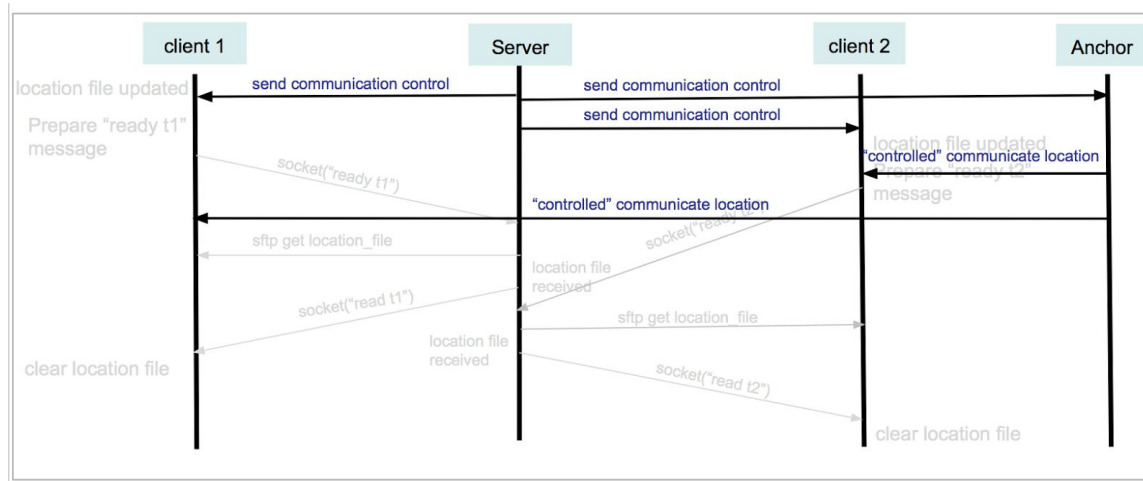
### 6.2.3.1 The main communication



The main communication is responsible for delivering the location of the tag to the server. As shown in the above figure, the server can handle multiple clients but for the sake of clarity we show only the communication with two clients. The client prepares the location and appends it in a datafile then opens a TCP/IP socket and sends a 'ready' message to the server. When the server accepts the connection and reads the 'ready' message, it uses secure file transfer protocol 'sftp' to get the data file from the client. We use the sftp protocol to make sure that the transfer of data is over secure channel also we make the sftp is from the server side using 'get' command in sftp protocol. This ensures that only the server maintains the password of the clients and not the other way around as explained in our system guarantees.

After the server 'gets' the data file it echoes back a 'read' message in the same opened TCP/IP socket to the client. Once the client receives the 'read' message it clears the data file.

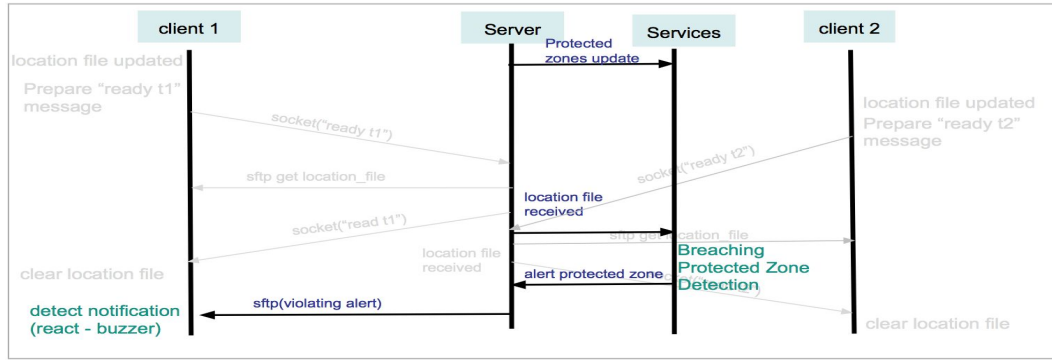
### 6.2.3.2 The control communication



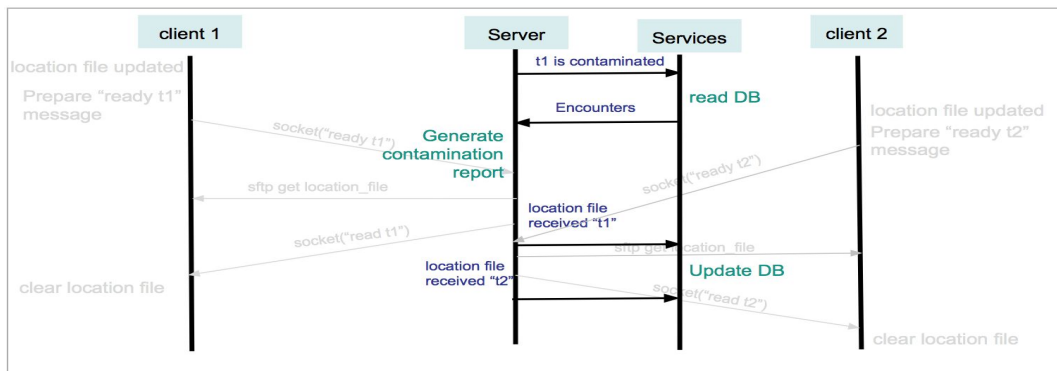
The control communication is responsible for changing the message pattern sent from the anchor to the client. As shown in the above figure, the server sends this information to the clients and the anchors at the same time and once the anchors and the clients have the same message pattern then the client can interpret the message into location.

### 6.2.3.3 The services communication

This layer of communication is responsible for handling the back end services on the server and communicating the actions taken by the server to the tags. The services supported in our system are the protected zone alert, the contamination, and the tracking.

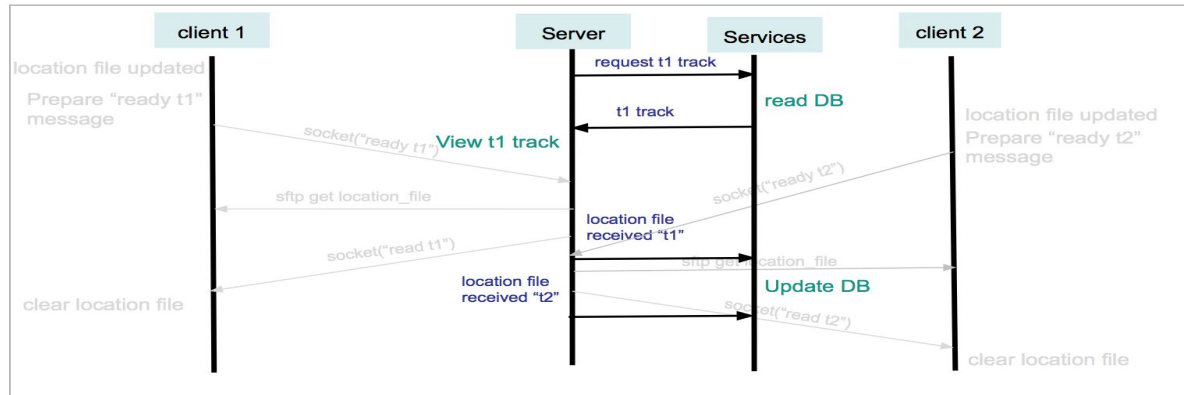


1. The protected zone alert is an alert sent by the server to the clients if the server detected that a certain tag breached a zone that it is not allowed to enter. This is done by having a separate thread running called 'Services' that continuously gets updates about the current location of each tag as well as gets the information of tag/allowed zone information from the server and accordingly sends to the server thread if breaching of a protected zone is detected. The server then reacts by sending a violating alert to the tag. When the tag is notified that it has breached the zone, it notifies its user. In the current system, we implemented this notification as a buzzer sound. However, the notification can be less intrusive by having the client send a text message to the tag holder to know that s/he is not allowed in in this location. The protocol is shown in the figure above.



2. The second service supported by the server is the contamination hazard. If a patient is declared as by a doctor to be contaminated then the services thread reads the database and sends all the encounters with this contaminated tag to the server. The server then generate a contamination report with all the encounters and the timestamp at which the encounter happens. This is like having a softwall around a contaminated patient and detecting all softwall violations. The protocol is shown in the figure above.

## Design Document



3. The third service that is supported by the server is the tracking of a tag in the hospital. The server requests the track of a certain tag from the services thread. The services thread then reads the database and generate a timestamped track of that tag and sends this information to the server to view it. The protocol is shown in the figure above.

## 7 System Source Code

### 7.1 Localization System

<https://github.com/resolutedreamer/IR-Transmit-Receive>

Please see the README.MD file for more information.

### 7.2 Secure communication system

Tutorial on how the communication code is implemented can be found in the Shared System section of this report.

The code available at the following links:

#### 7.2.1 Server

<https://drive.google.com/open?id=0Bw-Zik7FqIUTTIwUE1hRUpsUG8>

#### 7.2.2 Client with random location for unit testing

<https://drive.google.com/open?id=0Bw-Zik7FqIUTTIwUE1hRUpsUG8>

#### 7.2.3 Client integrated with IR receiver

<https://drive.google.com/open?id=0Bw-Zik7FqIUTLW4zU09vUjE2NzQ>

#### 7.2.4 Services

<https://drive.google.com/open?id=0Bw-Zik7FqIUTNGlpWIBDSmc4azQ>

<https://drive.google.com/drive/folders/0B0kC2pgiScfEaHJOX2ZVUnNneDQ>

#### 7.2.5 GUI

<https://drive.google.com/open?id=0Bw-Zik7FqIUTNmRIId0tWNDY5aVk>

## 8 Test and Evaluation System and Testing Results

Testing our system adequately, both in hardware and in software, was the most critical portion of our project. Unit testing for each part of the project was carried independently before the final testing after the integration.

### 8.1 Hardware Testing

We began the design of our hardware system with it as simple as possible, and then we eventually expanded it little by little.

#### 8.1.1 IR Emitters (LEDs)

##### 8.1.1.1 Procedure

A simple resistive divider circuit with a 9V battery and 330 ohm resistor was used to test our LEDs at each stage of our project where we had problems with emission. We would always test our LEDs independently of the Intel Edison to isolate any issues. The IR light emitted is not visible by the naked eye, however, we were able to use a cell phone camera in order to detect the presence of IR transmission or lack thereof.

##### 8.1.1.2 Results

During our initial test when we received the emitters, they all worked, as expected. However, as we experimented with different means of powering the LEDs, we overdrive the current on some of them, which resulted in the LEDs being burnt out. Therefore we had to continuously apply the testing procedure at every stage of our development to identify faulty LEDs and replace them.

#### 8.1.2 IR Receiver (Diodes)

##### 8.1.2.1 Procedure

The functionality of the IR Receiver was more difficult to test than the Emitters. This is because of the 38 KHz filter that is on the receiver, so in order to test if the receiver is functioning correctly, an emitter that is emitting at 38 KHz is required. Eventually, we realized that the best way to test the receiver was to point a television remote at it, because that would provide the signal required.

Under normal operation, the 3 terminals of the receiver should be operating normally. The terminal 3 of the receiver,  $V_s$ , should be connected to a supply voltage between 2.5 - 5 V, and when measured with a digital multimeter (DMM) it should read the voltage connected to. Likewise for the terminal 2 of the receiver, ground, we would connect it to a common ground and expected it to measure 0 V. Then, the terminal 3, output terminal, would be expected to be equal to  $V_s$  under the condition of no IR received, and equal to ground if IR is received.

##### 8.1.2.2 Results

During our initial test when we received the emitters, they all worked, as expected. We did not do anything unusual to the receivers, however later during testing we realized that one of the receivers was severely malfunctioning when we tested the output pin under the no IR condition and the voltage was 0 instead of 1 when it was completely covered to verify there were no IR signals pointed at it. No matter what voltage we used as the supply voltage, including AA batteries, 9V battery in series with resistor, or 5V Edison output, the output pin was simply stuck at 0 V. We ended concluding that the receiver was indeed faulty.

### 8.1.3 MOS Power Driver Kits

#### 8.1.3.1 Procedure

We used the MOS kits in order to power the IR Emitters with more current than the Intel Edison devices could supply by themselves. These arrived unassembled so they required additional work of soldering before they would become usable. Testing would commence afterwards. These kits are 5 terminal devices, with 3 input terminals and 2 output terminals. For the two output terminals, we connect our IR LED to them, and the + output terminal is one of interest for viewing on the oscilloscope. For the three input terminals, + is Vdd and connected to our same 9V in series with 330 Ohm, as in the test circuit for the single IR LED. Ground is connected to common ground, and the pin labeled C, control, is connected to the Edison controlled by software, but for our test circuit, is simply connected to the same 9V battery through a separate resistor.

#### 8.1.3.2 Results

After we soldered all of our MOS Power Kits, we tested them to verify our soldering job and we ended up needing to re-solder 3 or 4 of them. After that, we did not need to test these again until we began final hardware assembly on the cardboard pieces.

### 8.1.4 Demo Hardware Assembly

#### 8.1.4.1 Procedure

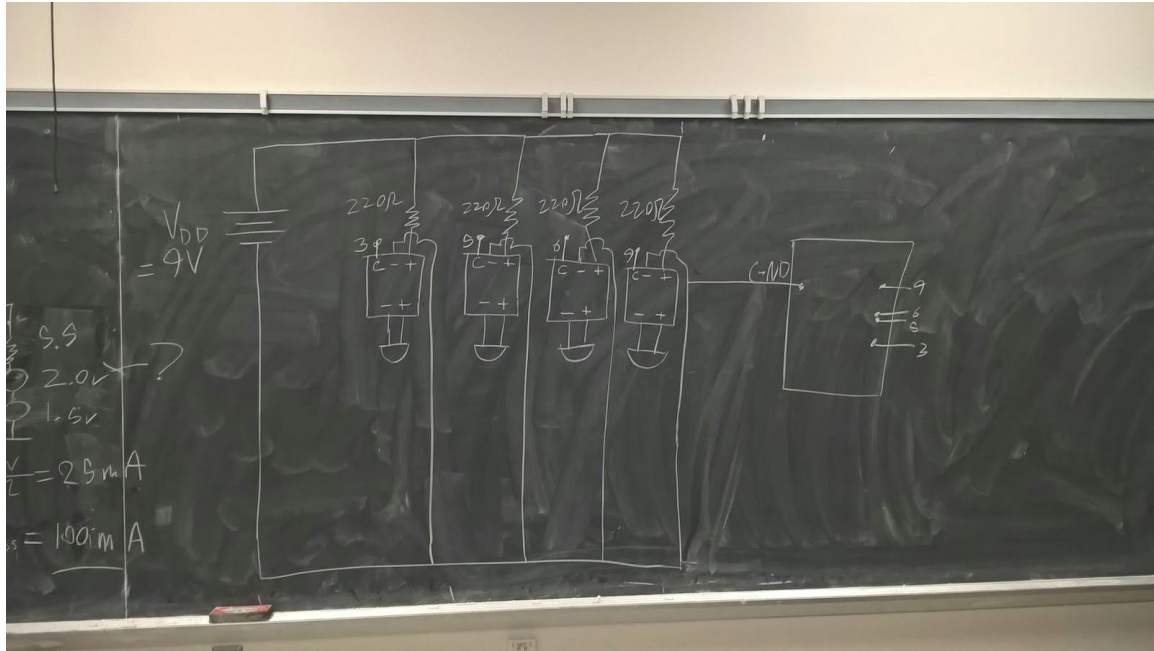
After we had all the individual components tested, we had to decide what form our demo would take, since we were not actually going to attach transmitter nodes onto the ceiling of a room. We ended up using a large piece of cardboard that functioned well:



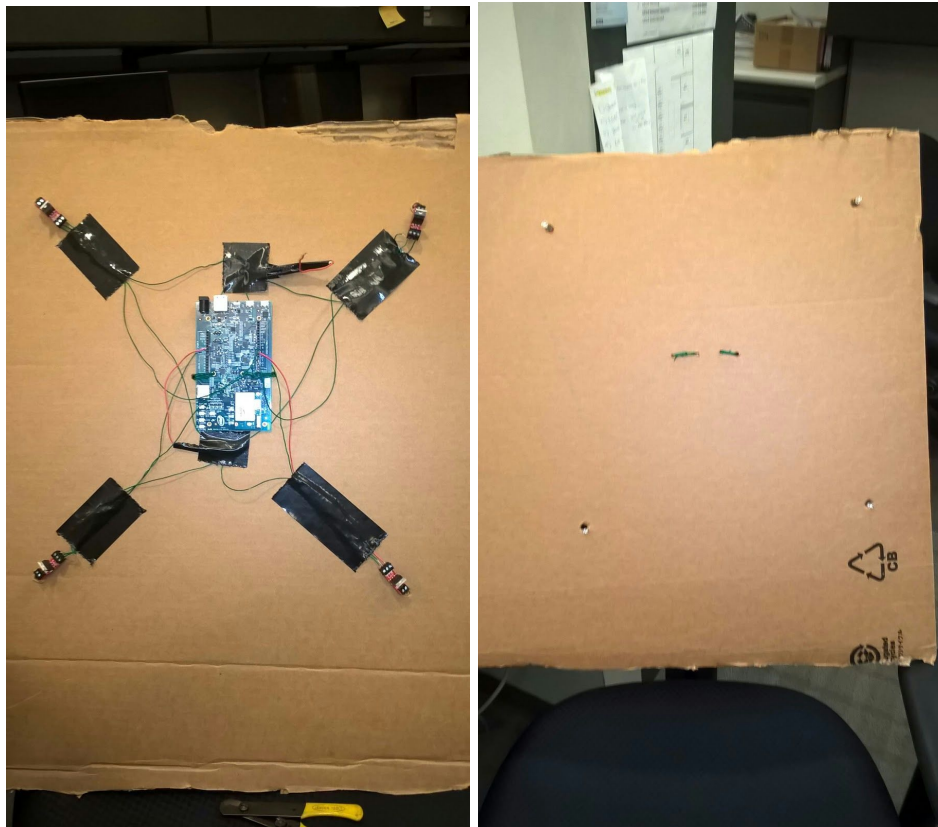
This shows the circuit used to connect the various components together:



## Design Document



Assembled, this is the circuit:



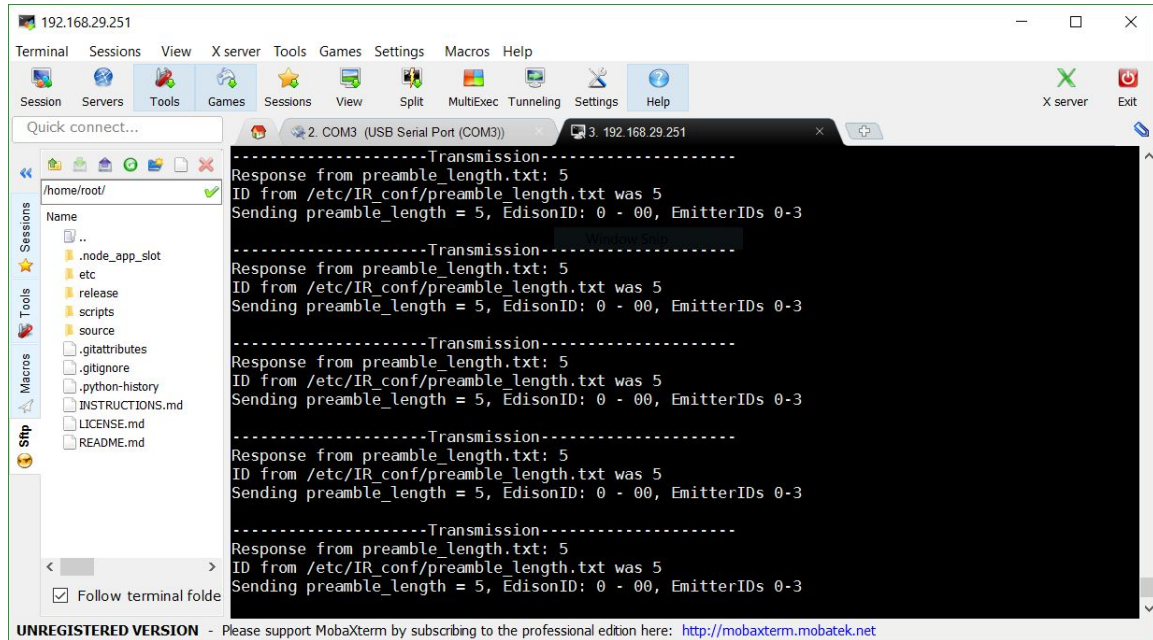


## 8.2 Software Testing - IR Transmit/Receive

### 8.2.1 IR Emitters (LEDs)

To test our emitter code we set up an oscilloscope to capture two signals, the emitted signal and the signal at the output of the receiver module. Since we knew the theoretical relationship between the emitted signal and the output of the receiver, capturing these two signal and checking their actual relationship against the theoretical one would confirm proper functionality.

This is a screenshot showing the debugging output of our IR Transmitter:



The screenshot shows a MobaXterm window with a terminal session connected to 192.168.29.251. The terminal displays the following output:

```
-----Transmission-----
Response from preamble_length.txt: 5
ID from /etc/IR_conf/preamble_length.txt was 5
Sending preamble_length = 5, EdisonID: 0 - 00, EmitterIDs 0-3

-----Transmission-----
Response from preamble_length.txt: 5
ID from /etc/IR_conf/preamble_length.txt was 5
Sending preamble_length = 5, EdisonID: 0 - 00, EmitterIDs 0-3

-----Transmission-----
Response from preamble_length.txt: 5
ID from /etc/IR_conf/preamble_length.txt was 5
Sending preamble_length = 5, EdisonID: 0 - 00, EmitterIDs 0-3

-----Transmission-----
Response from preamble_length.txt: 5
ID from /etc/IR_conf/preamble_length.txt was 5
Sending preamble_length = 5, EdisonID: 0 - 00, EmitterIDs 0-3

-----Transmission-----
Response from preamble_length.txt: 5
ID from /etc/IR_conf/preamble_length.txt was 5
Sending preamble_length = 5, EdisonID: 0 - 00, EmitterIDs 0-3
```

The MobaXterm interface includes a menu bar (Terminal, Sessions, View, X server, Tools, Games, Settings, Macros, Help), a toolbar, and a sidebar with a file explorer showing the directory structure of the remote host.

### 8.2.2 IR Receiver (Diodes)

After testing the emitter and confirming that the output of the receiver was correct, we then could connect the output to the edison and test the receiver code. By carefully using cout statements, we were able to test and confirm the proper functionality of the receiving code.

This is a screenshot showing our debugging output for the IR Receiver, in a case where the receiver is NOT able to find the location:

## Design Document

[illegible]

192.168.29.251

Terminal Sessions View X server Tools Games Settings Macros Help

Session Servers Tools Games Sessions View Split MultiExec Tunneling Settings Help

Quick connect...

2 COM3 (USB Serial Port (COM3)) 3 192.168.29.251

/home/root/

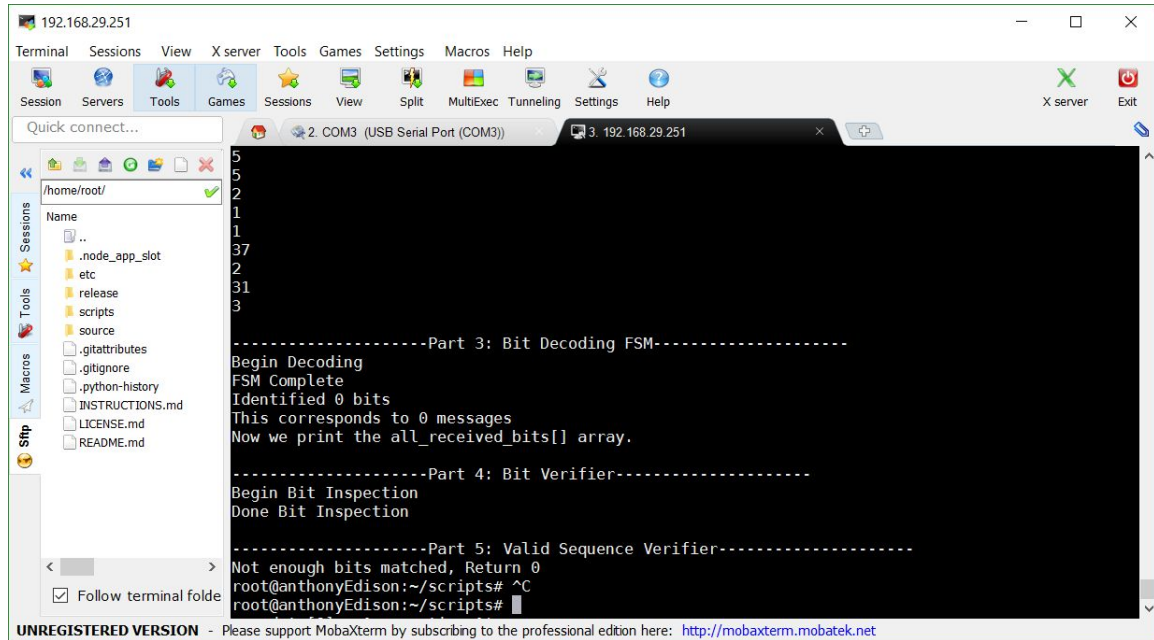
Name

- ..
- .node\_app\_slot
- etc
- release
- scripts
- source
- .gitattributes
- .gitignore
- .python-history
- INSTRUCTIONS.md
- LICENSE.md
- README.md

Follow terminal folder

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>

## Design Document



### 8.3 Software Testing - GUI

To properly test the GUI's tracking of a tag in real-time we generated an array of locations and created a thread that assigned the location of each tag. This allowed the accurate observation of how the GUI would behave when assigned real locations in real-time by each tag.

### 8.4 Software Testing - Server

Since the server had to be tested for interactions with both the tags and the GUI, there were two different simulations created to test both of those functionalities.

#### 8.4.1 Server-Tag

To test that the server and the tags were exchanging information correctly, a random generator was set up to generate random locations and store them into a file. Then the server was observed accessing those randomly generated locations, getting them from the tags and then save it in file. An example for a location file saved on the server side is as follows:

```
2015-12-04 18:07:38.408972 3
2015-12-04 18:13:29.326277 4
2015-12-04 18:28:00.428124 3
2015-12-04 18:28:19.924100 4
2015-12-04 18:28:23.785473 3
2015-12-04 18:28:28.808786 4
2015-12-04 18:28:38.010745 3
2015-12-04 18:28:41.261606 4
2015-12-04 18:28:45.536142 1
2015-12-04 18:31:13.326692 3
2015-12-04 18:31:20.539429 2
2015-12-04 18:31:25.700774 3
```

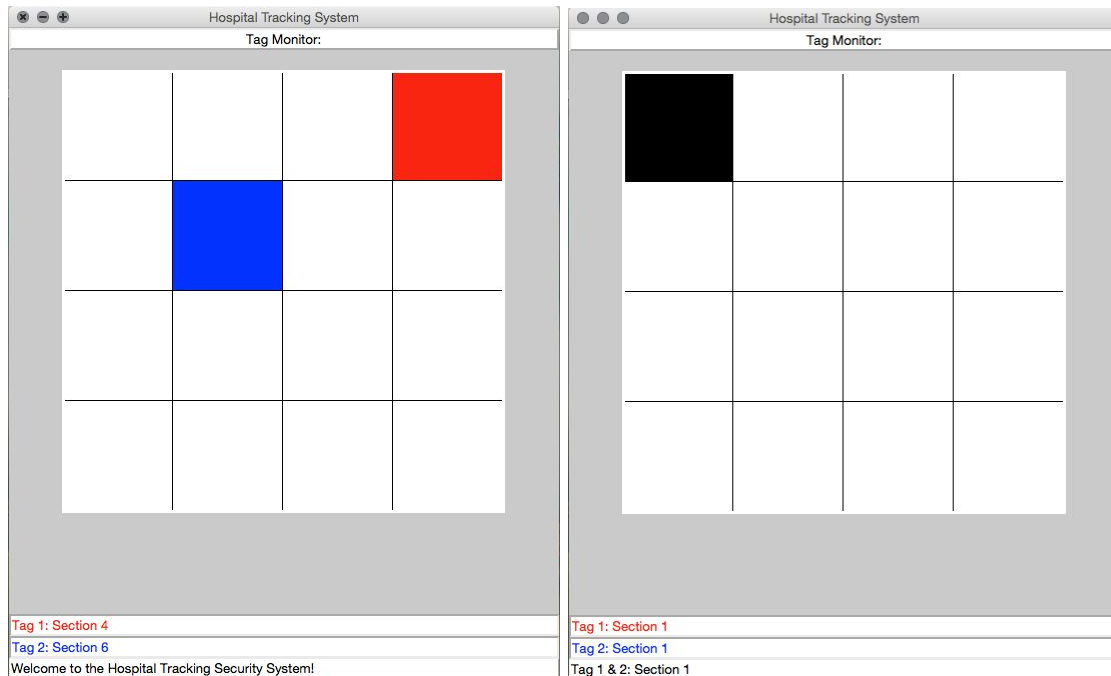
The first column represents the timestamp and the second column represents the location of the tag at this timestamp.

## 8.4.2 Server-GUI

To test the server interaction with the GUI, the server tag test bed described above was extended so that the randomly generated locations would be passed from the server to GUI, via a file, and then displayed on the GUI.

## 8.5 System integration

Integrating the system was a systematic process and was executed in two steps. The first step was to link to localization system to the server with real locations. Then the second step was an extension to that flow and had the GUI displaying the received localization data.



**Figure 8.5.1-** Multiple tags can be tracked (left). When more than one tag occupies a section, it turns black (right).

When our system is fully integrated, the GUI displays the current positions of the tags in our system. Our GUI is also capable of detecting when two or more tags share the same location as shown in figure 8.5.1. With the same location values that are used to display a tag's location, the GUI can display the last 5 locations of any tag, along with displaying any collision between any two tags. It does this by utilizing the timestamp that accompanies a tag's location, and these function can be executed by clicking on the proper tabs on the GUI. The encounters and the tracking information are shown in Figure 8.5.2. The generated report for the encounters looks as follows:

```
Encounter between t1 and t3 2015-12-04 18:28:28 4
Encounter between t1 and t3 2015-12-04 18:31:20 2
Encounter between t1 and t3 2015-12-04 18:28:41 4
Encounter between t1 and t3 2015-12-04 18:28:45 1
```

These lines explain the encounters that happened between tag1 and tag 3 at which timestamp at which specific location.

Design Document

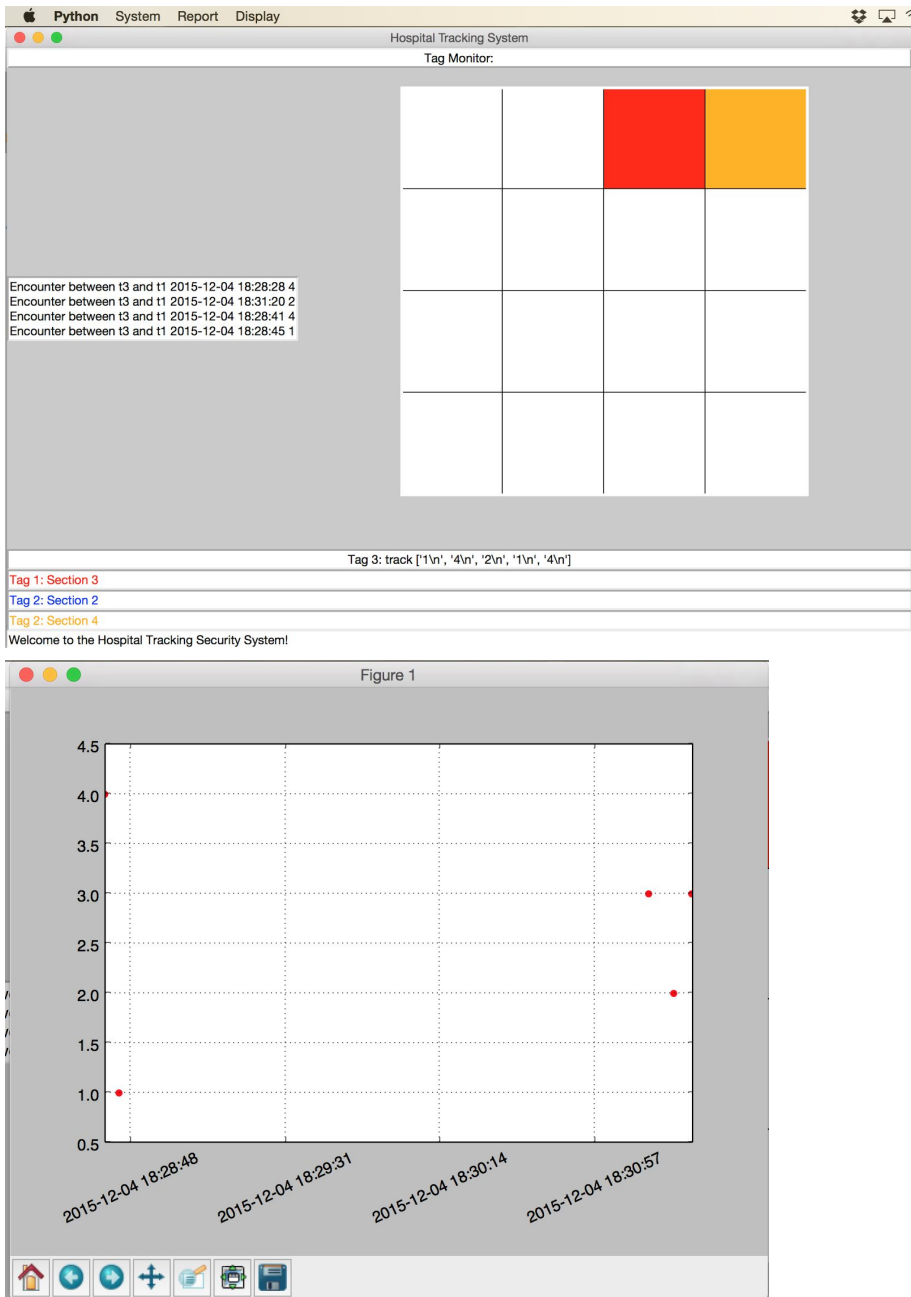


Figure 8.5.2 Encounters and Tracking information

## 8.6 Full System Operation

Step #	Step	Test
1	Server.py generates a random number between 0 and 9 to be used as the preamble length for current transmission.	print statements should confirm generation, no errors thrown
2	Server uses SFTP to write the random number to preamble_length.txt on all the anchor nodes and all the clients.	print statements should confirm file write, no errors thrown
3	Anchor reads updated preamble length from file and adjusts the number of preambles sent.	print statements should confirm preambles being transmitted, oscilloscope can confirm new preambles.
4	ir_receive program on the tag reads the preamble_length.txt file and uses that value to interpret messages being received. program returns correct value to Client.py	C print statements should confirm preamble length being searched for matches value in preamble_length.txt. Number of preambles viewed on the oscilloscope should match the amount being sent on transmitter.
5	Client.py writes the detected location into current_location.csv	check value in current_location.csv to verify it matches with expected value written from Python
6	Server pulls location from current_location.csv	check value picked up by server matches value on the tag.
7	Server uses location from current_location.csv and updates location displayed on the GUI	Verify location displayed on the GUI matches the location in the text file
8	Server takes adequate actions depends on the current location of the tag and the service required	Protection zone service is verified using the sound of a buzzer when the the tag enters a protected zone

## 9 Test System Source Code Repository

Client with random location for unit testing:

<https://drive.google.com/open?id=0Bw-Zik7FqIUtlwUE1hRUpsUG8>

## 10 Analysis and Further Work

### 10.1 Localization

In this project we successfully demonstrated Infrared transmission and reception by interfacing the IR LED and the receiver with the Intel Edison a. This serves as an enabling technology for localizing smart tags within a hospital environment. The LEDs used for the the project purpose were low power ones, but the same working principle also applies to higher power IR LEDs which can be deployed on the ceiling of a hospital.

At the time of the midterm we were transmitting a message ( including preamble and bits) in 200 ms ; but by the end of the quarter we brought down the transmission time to 40 ms by using scaled delays. This would bring about a good improvement on the responsivity of the smart tags. Also we improved the robustness and resilience of the smart tag towards variabilities, noise and outliers. Our research on IR LEDs show that even higher power IR lights have focussed beams and therefore our method of localization can prove to be a low cost and efficient way of localization in a hospital indoor environment.

#### 10.1.1 Receiver Line of Sight

Line of sight problem of IR communication can be overcome by having wider ( but non-overlapping) cones of Infrared light for each of the anchor nodes.

#### 10.1.2 Enhancing Responsiveness

Using the MCU tutorials provided by Chris, we completed code to deploy the onboard MCU of Intel Edison to further bring down the time needed to transmit a message. However, this code was not used in the version presented at the final exam because we did not have time to implement the more interesting features of our transmission system on the MCU, such as server controlled variable length preamble, variable edison ID, and scaling factor.

With the current version of the system, messages take 44ms to transmit, however times on the order of ~10 microseconds are possible through use of the MCU. This would help us to improve the resilience of our system. If all it takes for the transmission of location from emitter to the smart tag is < 1ms, the probability that a location of a person is sensed by the smart tag and communicated to the server increases.

#### 10.1.3 Enhancing Cost Effectiveness

The second thing that can help to improve the cost efficiency of our system is to use the standard GPIO pins of the Edison to drive the emitter LEDs. This should be achievable because 38 Khz is not a very high frequency to be generated using software, so even using non-pwm pins it should be possible to transmit messages. If we do this, we can further reduce the cost of our system.

#### 10.1.4 Considerations for Physical Installation

When installing the system on the ceiling, care should be taken to ensure that the location of the anchor nodes is both secure and obscured, to prevent tampering with the anchor nodes by unauthorized personnel. The Edison anchor node pins provide a signal, which could then be amplified and sent through long wires to the LED endpoints in each hospital room. Care should be



## Design Document

taken for the system so that the hospital can “set it and forget it,” allowing for it to be a one time installation when the hospital is built. Software updates can be delivered to the Edison via WiFi.

It is still possible that routine repair and maintenance will be required. The server can intelligently determine what areas there may be failed IR LEDs by analyzing the data and looking for zones where no location is ever detected by a tag. During maintenance hours this can then be verified by sending service personnel with a known tag to investigate these trouble spots.

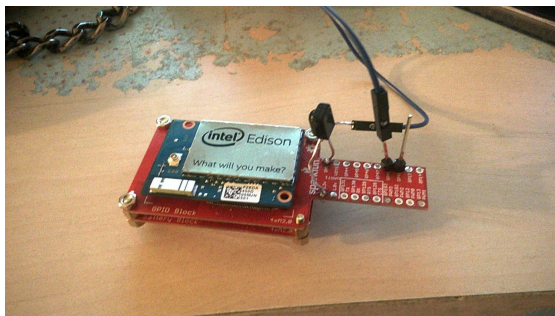
### 10.1.5 Other Known Issues

One other issue that was brought up is the question about what if a person conceals a smart tag? We have considered how to detect and resolve this issue and the solution that we have come up with is this:

Detection: The server will be able to detect if a smart tag goes offline because it will not receive an updated location from the tag for a certain amount of time.

Action: Send an alert message to the tag, such as by a buzzer or a pre-recorded voice message to alert the user and those around the user that the tag is not functioning properly. The alert should be sufficiently loud as to attract the attention of security if the hospital is completely quiet.

### 10.1.6 Intel Edison SparkFun Breakout Blocks



We were not able to make use of the SparkFun blocks for our project.

Assembly of the blocks was not an issue, the blocks all use the same type of connector. As you can see in the image below, the use of the SparkFun block allows considerable space saving over the full Arduino Breakout board. Instead, the issues we faced were software issues.





Originally we had ordered the Battery block and the GPIO block. We faced two significant problems using these blocks. The purchase of a Base block helped to mitigate the first issue we were having, but we were not able to resolve the second issue.

### 1. Unable to connect to Edisons

- a. Problem Details: Our working procedure to access the Edisons on the Arduino breakout board had been to plug in the power, plug in the serial cable, log in via serial to connect to the wifi, and then ssh over the wifi to access the Edison for development. However, without the base block, we were not able to follow this procedure to log into the Edison on the SparkFun blocks. We would connect to the wifi on the Arduino breakout board, shut off the Edison, then transfer it over to the GPIO/Power block unit, and then turn it on, attempt to SSH into it, and typically we would receive no response at the IP address we had been using before. We concluded (correctly) that the IP address was changing when it was turned on with the SparkFun blocks, so that we would need to find the new IP to access the Edisons. Since we did not have the Base Block, we could not find the new IP.
- b. Solution 1: Ordering the Base block is the simplest solution to this issue. Wired access to the Edison is thus available and then the current IP can be accessed.
- c. Solution 2: IP Auto Reporting. We developed this solution by learning how to schedule programs to start automatically every time Linux boots up. I followed the guide linked to at (<http://stephaniemoyerman.com/?p=41>) with great success. I strongly recommend that this guide is made available to future classes using the Edison, because it took me a lot of searching to find the correct method that would work on this device. I found many other guides that would work on more full Linux distributions, but this was the only one that worked for the Edison. Once we knew how to schedule the program on startup, we created a script to send us an notification about the current IP address as soon as the Edison was connected to the internet.

### 2. Unable to interface with GPIO

- a. Problem Intro: This issue was the dealbreaker that we were not able to solve, and we do not know why. The SparkFun GPIO pins were not working the same way as the Arduino breakout board pins. The original intent for the SparkFun boards were to use them as a more portable receiver unit, which could be act as the smart tag worn by a person. Thus, we wanted to be able to add two devices to the GPIO blocks - the buzzer to make an alarm for the restricted zone, and the IR receiver diode itself so it would be able to receive a location.
  - i. Problem Details: For the Buzzer, we needed PWM capability on the GPIO block. The block reveals the 4 pins for the 4 PWM outputs capable of the Edison, however attempting to run any program that used the PWM capability would always lead to "Segmentation Fault" errors.
  - ii. Problem Details: For the IR Receiver, we took receiver devices that were known to work and well tested on the Arduino breakout board and then

attached them to the SparkFun GPIO blocks. Connecting terminals 3 and 2 to  $V_s = 3.3\text{ V}$  and  $\text{GND} = 0\text{ V}$  gave a correct measurement of 3.3V (high) across the terminal 1 (Out) for the IR receiver when the receiver was NOT connected to a GPIO pin. However, when trying to connect the terminal to any of the GPIO pins the voltage on the terminal would read 0.37V, neither high nor low. We made sure that the pins were configured as input devices.

- iii. Problem Result: We were not able to diagnose why these errors were occurring. For PWM, we assume the issue is not properly configuring the Edison pins for PWM but we were not able to locate suitable instructions. For the IR receiver, we have no idea what might be wrong.

## **10.2 Secure communication system**

### **10.2.1 Communication delay**

The communication delay between the server and the client for successfully receiving the location and informing the client that the location was received took around 500 ms. This time delay is expected. Since our heuristics assume that the tag can not move from one location to another in less than 250 ms, our system is expected not to lose information. However, since the way the localization scheme works which continuously reading the IR transmitter, we experienced timing excess timing delay because the server has to get the location continuously. To solve this issue, the client code only establishes a connection with a 'ready' message to the server when a new location is detected and not repeated location value. This decreases the timing delay and made the server gets the information in a smooth way.

### **10.2.2 Number of clients**

The server is tested on three clients and is successfully able to monitor and receive the location from them at the same time. However, we have not experimented what the maximum of clients that the server can connect to is. We needed more hardware to act as multiple clients in order to experiment that. However, our expectation is that the number of clients will be bounded by the number of parallel threads that the processor of the machine which hosts the server can maintain at the same time. Keeping this in mind, the current implementation of the server, makes sure that the thread that holds the socket communication channel is closed every time the server finishes receiving a new information. In this case, the threads are continuously dispatched and then killed throughout the communication between the server and the clients. However, in the extreme rare case all the tags will have new location information at the same time and the server is required to dispatch parallel threads for all of them at the same time and that's when the number of threads will be bounded by the processor of the server machine.

## **11 Shared Systems**

A tutorial on how to implement the server with multiple clients can be found on this link:

<https://drive.google.com/file/d/0Bw-Zik7FqIUTUDNRM2Y0NkFUbGM/view?usp=sharing>

A tutorial on how to implement the Infrared Communication using Intel Edison can be found at -

<https://drive.google.com/open?id=0B0kC2pgiScfENDJkSIVCUFFhbnc>

## 12 References

### 12.1 Hardware References

#### 12.1.1 IR System Components

Page Title	URL	Unit Cost	Quantity	Cost
IR Receiver Diode - TSOP38238	<a href="https://www.sparkfun.com/products/10266">https://www.sparkfun.com/products/10266</a>	\$1.95	16	\$31.20
LED - Infrared 950nm	<a href="https://www.sparkfun.com/products/9349">https://www.sparkfun.com/products/9349</a>	\$0.95	16	\$15.20
SparkFun MOSFET Power Control Kit	<a href="https://www.sparkfun.com/products/12959">https://www.sparkfun.com/products/12959</a>	\$3.95	16	\$63.20
Hook-up Wire - Red (22 AWG)	<a href="https://www.sparkfun.com/products/8023">https://www.sparkfun.com/products/8023</a>	\$2.50	2	\$5.00
Total Cost				\$114.60

#### 12.1.2 Intel Edison

Page Title	URL	Unit Cost	Quantity	Cost
Intel Edison Arduino Breakout Kit	<a href="http://www.amazon.com/Intel-Edison-Arduino-Antenna-EDI2ARDUIN-AL-K/dp/B00PTVSU7U/">http://www.amazon.com/Intel-Edison-Arduino-Antenna-EDI2ARDUIN-AL-K/dp/B00PTVSU7U/</a>	\$87.00	8 (4 purchased)	\$696.00
SparkFun Block for Intel® Edison - GPIO	<a href="https://www.sparkfun.com/products/13038">https://www.sparkfun.com/products/13038</a>	\$14.95	4 (2 purchased)	\$59.80
SparkFun Block for Intel® Edison - Battery	<a href="https://www.sparkfun.com/products/13037">https://www.sparkfun.com/products/13037</a>	\$24.95	4 (2 purchased)	\$99.80

## Design Document

SparkFun Block for Intel® Edison - Base	<a href="https://www.sparkfun.com/products/13045">https://www.sparkfun.com/products/13045</a>	\$32.95	4 (1 purchased)	\$131.80
Total				\$987.40

More in-depth information about using the GPIO pins on the Intel Edison:

<http://www.emutexlabs.com/project/215-intel-edison-gpio-pin-multiplexing-guide>

Documentation for MRAA library, we consulted both C and Python sections:

<https://github.com/intel-iot-devkit/mraa>

Configuring programs to launch on boot up:

<http://stephaniemoyerman.com/?p=41>

SparkFun Tutorial on Infrared Communication can be found at:

<https://learn.sparkfun.com/tutorials/ir-communication>

## **12.2 Software References**

The Python Standard Library. Asyncore example basic echo server. <https://docs.python.org/2/library/asyncore.html>. Accessed: 2015-12.

Python Module of the Week. Asynchronous i/o handler. <https://pymotw.com/2/asyncore/>. Accessed: 2015-12.