6-13-2019

Doorbell for the Hearing Impaired

Shannen Edwin

Dominic Magdaluyo

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 8, 2019

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Shannen Edwin
Dominic Magdaluyo

ENTITLED

Doorbell for the Hearing Impaired

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE & ENGINEERING

B. Dezfouli
Thesis Advisor

Department Chair

Doorbell for the Hearing Impaired

by

Shannen Edwin
Dominic Magdaluyo

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science & Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 13, 2019

Doorbell for the Hearing Impaired

Shannen Edwin
Dominic Magdaluyo

Department of Computer Engineering
Santa Clara University
June 13, 2019

ABSTRACT

Doorbell options for hearing impaired individuals is seriously limited. Affordable solutions are not scalable while other solutions are expensive. With this in mind, we designed a scalable and affordable system that will be beneficial to hearing impaired individuals in a small aspect of their life. Our solution takes advantage of affordable IoT devices and software to build a proof of concept. Due to the scope of the project, we only designed a proof of concept, in the hope that a company can design a viable product that will not only benefit hearing impaired individuals but bring a powerful IoT system to the homes of others.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problem and Motivation

Doorbells and knocking are often useless for those that are audibly impaired. Someone with hearing impairments needs to depend on other residents to answer the door for them or purchase expensive equipment to send them phone notifications. Depending on the neighborhood, the ability to know theres a visitor at the door is essential for quality of life including community involvement and package delivery.

The inspiration for this system came from one of our homes. Since the doorbell stopped functioning within certain parts of the home, it was easy to see the impact not knowing someone is at the door can have on a resident due to the inability to hear a doorbell. This also created a burden on others within the house due to repeated doorbell presses and searching for person the visitor intends to speak with.

1.2 Solution

In this section, we look at current solutions that exist and provide insight into our solution.

1.2.1 Current Solutions

Current solutions are limited in supporting these individuals. Louder doorbells are useless to those that are completely deaf and can be bothersome for housemates that are not hearing impaired. Some solutions like Physens Doorbell Kits are viable and affordable solutions but they are not scalable. Physen and other manufacturers make closed systems that offer limited options. There are also doorbells such as the Rings doorbell which can send a notification to a phone; however, this solution can be expensive. Rings security camera doorbell costs $100. That can be well above the financial capabilities of many people. Especially among deaf individuals, we find that unemployment is high at 47% [4] and many in this population are senior citizens; therefore, finding an affordable solution is very important. There are also scalable options such as the IFTTT platform, which can connect various IoT devices that are readily available on their platform. Unfortunately, this system is closed. Companies must be registered to their system and developed

for it. The IFTTT platform also drops and adds devices to the system without warning. The devices available to help hearing impaired individuals in the IFTTT platform can be expensive such as connecting the Ring doorbell to Philips Hue lights which can add up to at least $200.

1.2.2 Contribution

Our solution is a proof of concept for a scalable and affordable system. Our doorbell is low power and connects to a modular system. A gateway consisting of a Raspberry Pi Zero W and an XBee will be able to connect any device a user desires by using Node-RED.

Chapter 2

Requirements

2.1 Description

In this chapter, we list the functional and nonfunctional requirements prioritizing them from top to bottom. In section 2.2, we list functional requirements by critical, recommended, and suggested and these describe what the system will do. In section 2.3, we list non-functional requirements in the same way and these describe how the system will perform. Lastly, in section 2.3, we discuss our design constraint and how it affects the scope of this project.

2.2 Functional Requirements

- Critical
  - Doorbell will activate light system
- Recommended
  - Doorbell will activate vibration system
  - Doorbell is integratabtle with other light/vibration devices available
  - The System will be low power
- Suggested
  - Doorbell will not communicate over WiFi

2.3 Non-Functional Requirements

- Critical
  - The system will be cheap, ideally $50 or less for a full house integration
- Recommended

  - The system will be able to connect to various types of devices to allow preferred devices per user

- Suggested
  - The system has an appealing aesthetic

2.4 Constraints

The constraint of our design is that we are not building from the ground up. This makes achieving our goal of an affordable system very difficult as we will be using already built parts. As such is the case, we will have to design this system as a proof of concept.

Chapter 3

Use Case Diagram

3.1 Description

The use case diagram (Figure 3.1) shows the potential functions of our system for two actors: the Resident and the Visitor. Our use cases are listed in the following section by which actor is allowed the action and what preconditions, postconditions, and/or exceptions are there for the specific use case in our system.

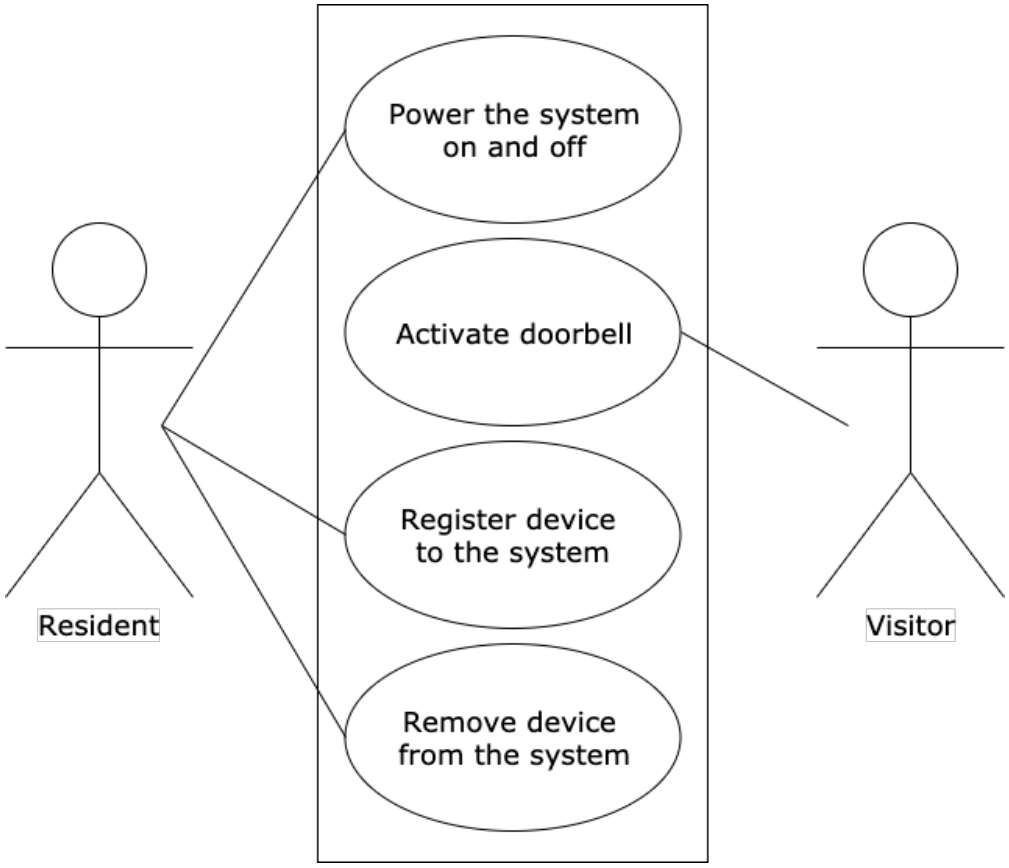

Figure 3.1: Use Case Diagram

3.2 Functionality

3.2.1 Power System On and Off

Actor: Resident

Goal: Setting up system and personalizing it

Postcondition: System should safely shut off or turn on with previous settings

3.2.2 Activate Doorbell

Actor: Visitor

Goal : Alert the Resident, that someone is at the door

Precondition: System is successfully activated with a device paired to it

Postcondition: Alert the Resident through their connected device(s)

Exceptions: Nothing happens if System is not active or device is disconnected

3.2.3 Register Element to the System

Actor: Resident

Goal: Connect a device so Resident may be alerted of Visitor at door

Precondition: System is powered on successfully

Postcondition: Device is registered successfully to system and will be activated once prompted

Exception: Unsuccessful device registration and prompt user of error in pairing

3.2.4 Remove Element from the System

Actor: Resident

Goal: Unregister device from the system when no longer using it

Precondition: System should be on

Postcondition: Device successfully unregistered

Chapter 4

Activity Diagrams

4.1 Description

The activity diagrams show the potential paths of actions a user may take with the system. For the Resident actor (Figure 4.1), upon turning on a system, the user may register or remove devices. The system will be then available and a user may receive a response when the doorbell is pressed. From any of these actions a user may take, they may also power the system off. For the Visitor actor (Figure 4.2), when the system is on, the user will be able to activate the doorbell by a button and have one of the connected devices respond to the Resident actor.

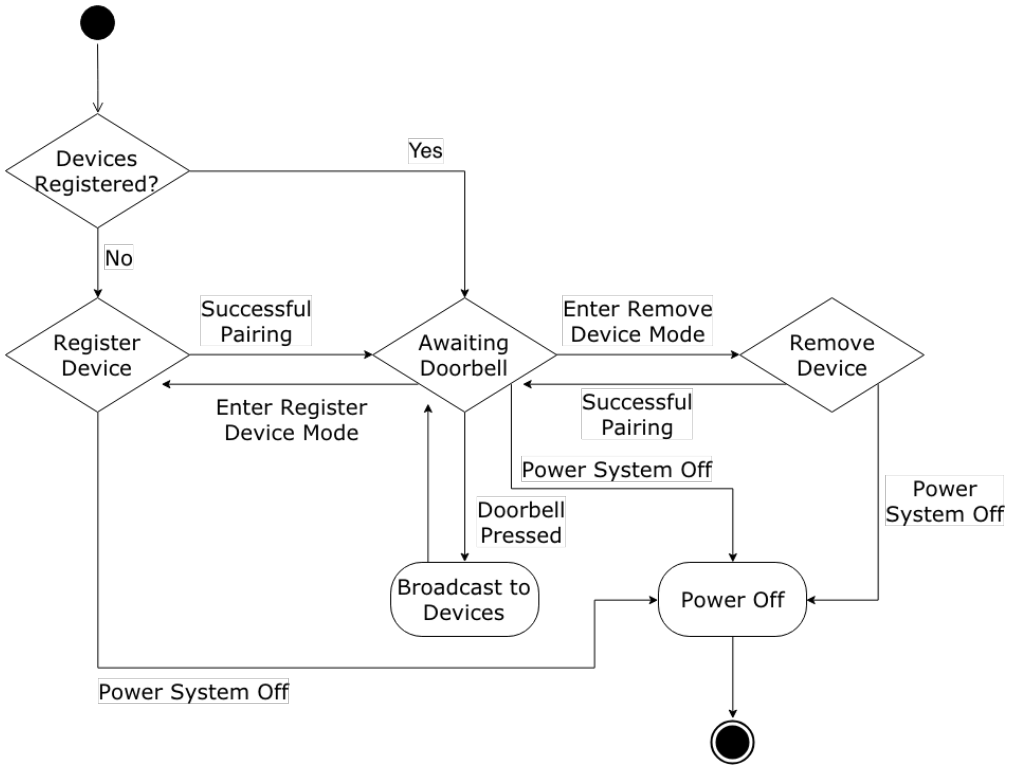

Figure 4.1: Resident Activity Diagram

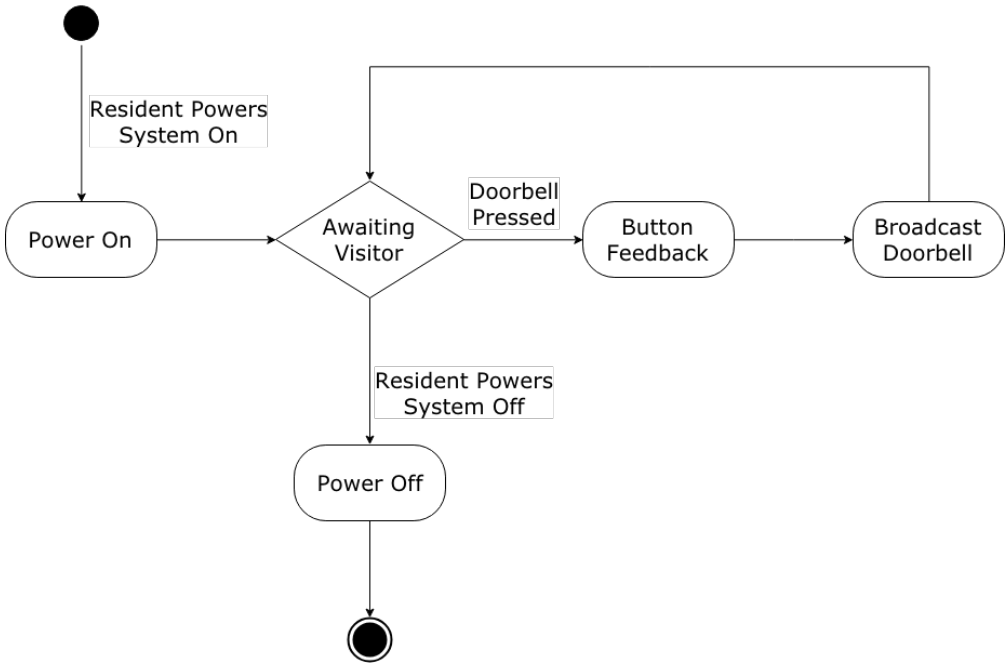

Figure 4.2: Visitor Activity Diagram

Chapter 5

Architecture Diagram and Conceptual Model

5.1 Architecture

The system relies on an event-based architecture. As shown in the architecture diagram (Figure 5.1), the arrows show the flow of data between the different components of the system. The doorbell button triggers an event that goes to the Raspberry Pi Zero which acts as an announcer in this architecture. This event tells the announcer to broadcast the appropriate doorbell event to all types of devices currently registered in the system. The doorbell offers feedback to the Visitor in the form of a pulsating light to confirm that the button was pressed and the event was broadcast to the Resident's devices. The arrow going to the Raspberry Pi Zero from the Home Network is to confirm that the devices received the broadcast event.

The architecture is event-driven. Each of the parts are in a waiting state. Once a trigger is sent, the state will change to its respective action and then repeat to a waiting state. The entire system is triggered solely by the Doorbell. It, being in a low-power waiting state, will wake up on a doorbell press and alert the Raspberry Pi Zero. The Raspberry Pi Zero will then have to process the package, alerting the respective devices through the Wi-Fi router. The Philips Hue Lights and other devices in the system will leave their waiting states once alerted through the Wi-Fi router and begin to notify the resident. All states are reverted back to waiting once the devices have finished notifying the resident. The advantage of having all devices to maintain a waiting state is that we can more quickly notify the user. The disadvantage is that the system will require more power. Fortunately, the only part of our system that would require that is the doorbell which would generally be outside, unable to plug into an outlet. The Raspberry Pi Zero can be plugged into an outlet. The best option would be to connect it next to the router and use an Ethernet cable to connect to the Wi-Fi router. The Philips Hue lights and other devices will be per-user requirement. Certain devices may operate on battery but that is not a concern of this project, but rather a concern for the resident.

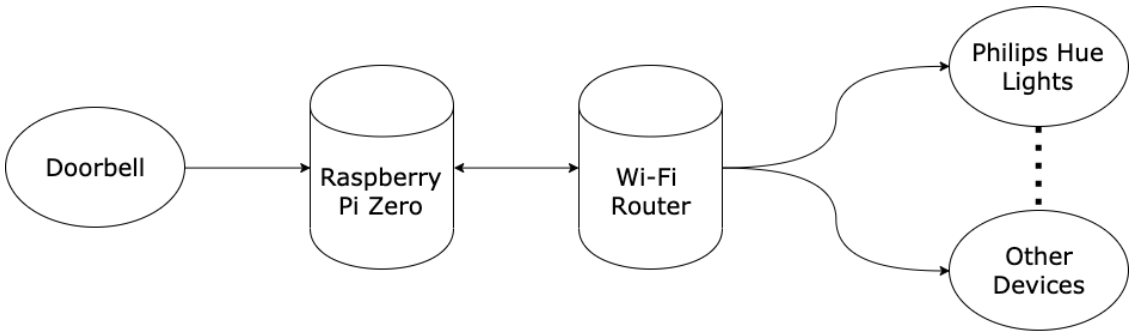

Figure 5.1: Architecture Diagram

5.2 Conceptual Model

The system uses the Raspberry Pi Zero as the main gateway. A router will be necessary in communicating to other devices in the household. The button of the doorbell will be attached to the an Arduino and communicate with the Raspberry Pi Zero. Since Arduino and Raspberry Pi do not come with 802.15.4 capabilities, an XBee was used to communicate between the devices. Furthermore, to abstract user experience, a smartphone application would be necessary to control the system. With simple actions, the resident should be able to perform all Use-Case actions described in Chapter 3. The system also uses a variety of communication protocols, so the appropriate hardware modules will be necessary.

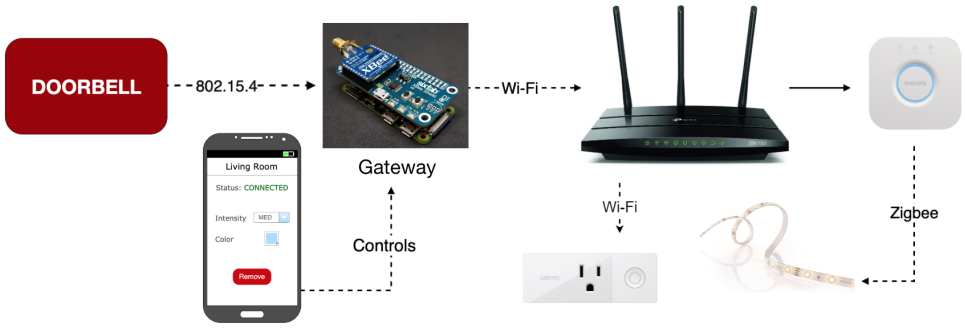

Figure 5.2: Conceptual Model

Chapter 6

Technology Used and Design Rationale

6.1 Description

In this chapter, we discuss the necessary devices and technology needed for this system and our reasoning for their choice in our design.

6.2 Communication Protocols

In the IoT world, there are a few communication protocols that we could use. In the following subsections, we go over the benefits and disadvantages of using each protocol.

6.2.1 802.11 (Wi-Fi)

802.11 is a common protocol with 802.11 access points present in many places today. It has high bandwidth at 11 Mbps and is supported by many devices. It is powerful and has moderate range between 30 and 100 meters. Unfortunately, this is a high power protocol which is not ideal for components in our system that will run on battery such as the doorbell and devices that may be used to notify the user. [3]. This is a huge disadvantage but it would be extremely easy to implement. Considering a Raspberry Pi, almost all the models come with 802.11 configurations built in.

In our design, we decided to use this protocol in only broadcasting to devices connected to our system that will relay notifications to the user. Most likely these devices will have consistent power and will already be connected to the users WiFi network. Assuming a resident has a Wi-Fi router in his or her house, all notifications could be relayed through that. If this isn't the case, the Raspberry Pi can be configured to run as a Wi-Fi router using Hostapd. This would further increase the possible outreach to users as some may not be able to afford Wi-Fi routers.

6.2.2 802.15.1 (Bluetooth)

802.15.1 is another common protocol. Many devices come with Bluetooth to provide low range, moderate bandwidth connections. The bandwidth is moderate at 800 kbps while also providing moderate power consumption. The range

is low at about 5 to 30 meters which is not useful if components of our system are placed far away from our central gateway. While a huge disadvantage, Bluetooth is available in many devices today such as smartphones.

In our design, we decided to use this protocol in broadcasting to devices connected to our system like 802.11 and for connecting a smartphone application to the gateway. The smartphone application would allow an average user to control the system as necessary for themselves. Although, a user will only be able to control the system if they are close to the gateway. With Bluetooth's moderate bandwidth, changes can be applied quickly.

6.2.3 802.15.4 (Zigbee)

802.15.4 is more commonly used in IoT applications. It is useful for its low power consumption and far range (about 100 meters). It has low bandwidth but that is not a problem for most IoT applications that may only require simple relays of information between devices. Most development in 802.15.4 uses Zigbee, a application layer protocol, that provides security and optimization for home automation. Zigbee is effective in making sure packets are not dropped and associating devices with each other. Bare 802.15.4 broadcasts data to a channel and doesn't worry about the end device receiving the data.

In our design, we decided to only go with 802.15.4. Development with Zigbee is covered under the Zigbee Alliance which would require us applying into the alliance. Considering the scope of this project, this seemed like an unnecessary delay. Furthermore, we dont need the benefits of Zigbee to show a proof of concept. We found that 802.15.4 would be best in broadcasting between the doorbell and the gateway. This provides good range and allows the doorbell to consume less power than other communication protocols. In this way, we can increase the lifetime of the doorbell and allow the gateway to be placed farther from the doorbell if necessary. Although, an actual implementation should use Zigbee in order to provide security and to make sure that all messages are received by the gateway.

6.3 Hardware

Our system consists of edge devices and a gateway. The edge devices are the doorbell and the devices relaying notifications to the user. The gateway is made up of a Raspberry Pi Zero W and an XBee. The gateway will receive a notification from the doorbell and will notify the other devices through a home network.

6.3.1 Gateway

The gateway consists of a Raspberry Pi Zero W and an XBee. XBee is a programmable module from DigiKey. The S1 variant can use either 802.15.4 or Zigbee depending on the firmware flashed. Furthermore, it uses serial Tx and Rx to communicate so it can be added easily to any module that has a serial port. The Raspberry Pi Zero W was favorable to us as it would make our development easier being a familiar Unix-based system and having a vast open-source community. Furthermore, it is low cost and the W variant comes with Bluetooth and WiFi. Using these protocols, we

can communicate to various devices connected to a home network through an access point. It is also a small device at 2.6 x 1.2 x 0.6 inches making it an unobtrusive device in the home. It has plenty more power than we require but its open source environment would allow for better development for us and in the future. It would also allow a user with programming knowledge to configure the system to their wants. There are 26 GPIO pins available. A possible configuration would be to set the GPIO pins to also notify users from its location by the router.

6.3.2 Doorbell

The doorbell was originally intended to be Texas Instruments SensorTag C2650. The SensorTag has the option of either using Bluetooth or 802.15.4, which can be configured by flashing different firmware. SensorTag comes with many sensors which we thought would be great for us to experiment with such as the motion sensor. It was also advertised to last for two years using a lithium coin battery. This is a huge advantage as we dont want users to be changing the battery often for the device. While more expensive than we would like ($30), the SensorTag c2650 could be a viable option with a greater amount of actions to benefit the resident.

We eventually decided it was best to move away from SensorTag due to issues covered in a later chapter. Our solution became to use two XBees to communicate with each other. XBee is a powerful, encrypted module that can operate with any 3.3V pin of a small computer. We knew that two XBees would work so long as they had the same parameters flashed. We verified this, using Digikey's XCTU software which allows us to flash parameters and debug the input. To accomplish this, we decided to use Arduino as the main processor for the doorbell. We used Arduino Uno for prototyping but Arduino Nano would be the best option in order to maintain a low power solution. Using an Arduino shield, interfacing with the XBee was simple using an XBee library from DigiKey. (Our design is implemented with the S1 variant. Different variants do not communicate with another variant, so it is important that both are the same.)

The XBee has many parameters that can be flashed per the 802.15.4 specifications. Fortunately, an XBee out of the box will work with another XBee out of the box with no configuration. This could lead to security issues though. As this was not an important aspect in our project of building a proof of concept, we did not concern ourselves with this. We did change the broadcast address so it transmits to many channels. With the receiving XBee scanning those channels, we can improve reliability with sending packets. We may see some packets dropped if there is interference on a specific channel. Furthermore, we can improve the power consumption of the XBee by configuring the device to wake up on a trigger on any of the pins.

For the doorbell, we also wanted a visitor to have feedback that the doorbell was pressed. A common occurrence in pressing a doorbell is hearing the doorbell chime, informing the visitor that the doorbell is working and has sent a notification to the resident. We decided that an LED pulse would accomplish this same effect. Using an Illuminated Momentary Pushbutton from Adafruit, we were able to simply accomplish this in our doorbell. Alternatively, a resident

could configure the doorbell to be attached with a speaker if they so desired. Arduino has many configurable speakers that can be attached to their devices.

6.3.3 Conneced Devices

A Philips Hue LED strip was used to test the implementation. Although expensive, Philips Hue has well-designed devices with an open API for Raspberry Pi, known as PiHue, that makes it easy to integrate into the system. Since this is just a proof of concept, showing the system performance was a larger concern.

6.4 Software Development

Software consisted of C/C++, Python and Node-RED. The software was used in different ways in the system and is described in more detail in the following subsections.

6.4.1 Doorbell Software

The doorbell software is written in Arduino code which is based off of C/C++. The Arduino board can communicate to the XBee using the already available serial library. Since Arduino runs code continuously in a loop, the code waits for a trigger from the doorbell press. On press, the LED pulses to notify the visitor that the button is broadcasting. The XBee will send a broadcast. Any XBee with the same parameters in range will receive the packet.

6.4.2 Gateway Software

The gateways software comprised mostly of Node-RED. Created by IBM for IoT applications and written in Javascript, Node-RED uses flow programming to trigger nodes. Each node is a block that runs a process when triggered and provides an output that can trigger another node or end the flow. Using a start-up-trigger node, the application will immediately start the flow when the Node-RED server starts. The flow will then wait to receive serial output from the XBee through the Pis Rx pin. When the node returns, the flow will split and notify all of the connected devices in the house each with their own respective nodes. To avoid unnecessary spawning of node processes and causing undefined behavior from doorbell spam, the flow waits for all nodes of notification devices to return. With the wait-paths node, once all nodes return, the flow is repeated. For even better configuration, the individual nodes can be modified. Each is a .json object and has a few inputs that a script performs on.

Node-RED is also an open source software. This is beneficial because an open source community has already developed many nodes and may continue to develop more. For example, there are already a few Philips Hue nodes. More open API edge devices may have available nodes. And if there are none, a user with minimal programming knowledge could create their own node and make it available to other user and our system.

Unfortunately, there is some delay with Node-RED starting. The Raspberry Pi Zero is a weak computer. If the SD card is not fast or there are many processes, it will be slow. From booting up to the flow starting, we found that it took about 4 minutes before we could start using the system. This isn't favorable performance but ideally, it is only performed once. In the case of restarting the flow, that would take about a minute. Again, this is not favorable performance but it is a disadvantage of using a lower cost module. A user could decide to go with a more powerful Raspberry Pi and SD card if they so desire and the performance will greatly increase.

6.4.3 Broadcasting Software

With Raspberry Pis versatility as a Unix-based system, any programming language can be used to activate the connected devices. In our testing, the APIs were available in Python. With Node-RED, any script can be run using an exec node. Furthermore, since Node-RED is open source, anyone can create nodes for use by creating a .json, .js, and .html file. Creating nodes are simple. With some technical knowledge, a growing network of nodes could be created for this application and may already be available for many devices.

Using Python or any scripting language also has a benefit in our system. We can easily modify the scripts when the resident desires to do so. Using the smartphone application, a user can modify settings of notification devices. By using a scripting language like python, we don't need to compile the code with every configuration. Furthermore, it would be unnecessary to restart the flow for Node-RED.

Chapter 7

Test Plan

7.1 Description

The following subsections are the testing procedures we went through to ensure success of the system.

7.2 Unit Testing

Unit testing is performed to verify that a small piece of code is performing as desired. Before using Node-RED to build the flow, we tested each of the potential functions of our system. Some examples include, making sure communication between the doorbell and gateway works as intended, writing small scripts to test communication to devices in a home network, and making sure each device functions as intended.

7.3 Functional Testing

Functional testing is to check the performance of multiple units as one functional unit. This was mostly just in our Node-RED application. We had to make sure that each unit could be integrated into a node and be triggered and output properly. This proved to have some difficulties due to unfamiliarity with Nod-RED's behavior.

7.4 Component Testing

Component testing is to check multiple functional units working together. We made sure that the doorbell will send a trigger as expected. Furthermore, we made sure that a flow could function by itself and not need any user actions to set up. The flow of Node-RED should return back to its waiting state after devices have notified the user.

7.5 System Testing

System Testing tests all the components together. In this step, all the communications between the components (doorbell, gateway, and connected devices) came together. We had to make sure that each step of the activity diagrams

could be performed and all of our use cases were implemented. The system had to also perform for long periods of time. We left if on for a few days and verified that the system performed appropriately.

Chapter 8

Results

8.1 Description

In this section, we go over the performance of the system's communication reliability and the cost of the system.

8.2 Communication Reliability

An area of concern with the wireless communication of the system involved interference between the 802.15.4 network and 802.11 Wi-Fi. Since both protocols use the same 2.4 GHz frequency, it's possible for packets to be lost due to channel interference. Figure 8.1 shows the frequency coverage of each Wi-Fi and 802.15.4 channel. In order to maximize the reliability of our system, we use 802.15.4 channel 26 so that there is no overlap with any common Wi-Fi network. Most Wi-Fi devices go as high as channel 11, which would not interfere with any communication happening on 802.15.4 channel 26. Although dropped packets were rarely encountered in the early stages of testing, switching to channel 26 completely eliminated all packet loss in our test environment.

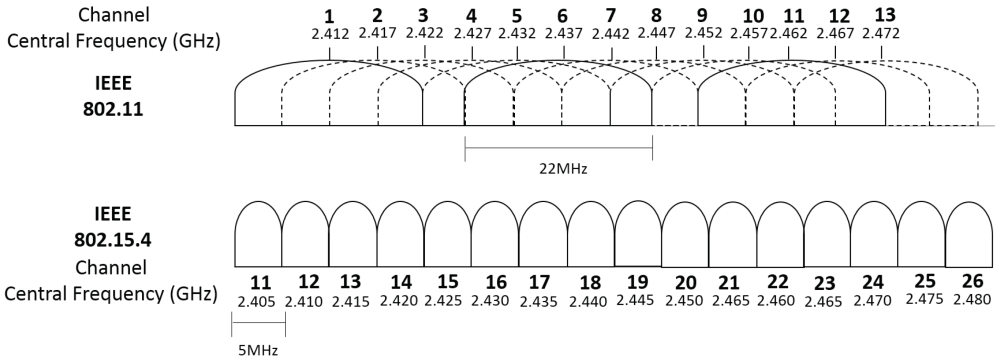

Figure 8.1: IEEE 802.11 and 802.15.4 Channel Comparison [2]

8.3 Cost

One main goal of this project was to minimize the cost of an open, scalable system. In considering the costs of solely our system (the doorbell and gateway), we find the cost to be reasonable. A Raspberry Pi Zero W costs $15. If a user were to decide not to use the W variant and connect the Raspberry Pi Zero directly to the router through Ethernet, it would cost $5 - $10. In our implementation we did use Arduino Uno but an implementation could use the smaller and low power alternative, Arduino Nano which costs $5 - $10. Unfortunately, the expensive module in this design was the XBee module. The cost is around $20 and we need two. The final cost is about $50 - $75. Of course, the resident would also have to purchase an IoT device that can send notifications. A possible affordable option would be a smart plug, which can effectively make any device a smart device. Cheapest options are around $10. At the cheapest and most expensive options, the cost is at least below $100. By some metrics, that may still be too expensive for individuals. Hopefully, as IoT starts to grow, we start to see small computers like Raspberry Pi and Arduino that implement 802.15.4 directly into the board.

Chapter 9

Development Timeline

9.1 Description

In this chapter, we show (Table 9.1) the development timeline of the system. Each light blue header marks a academic quarter and the work is divided up by the weeks in a quarter. We struggled to keep up with these deadlines due to obstacles encountered as described in a future chapter.

Task Name	Duration	Start	ETA	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Quarter Break
Fall Quarter														
Problem Statement	14 days	18.09.22	18.10.05											
Funds Proposal	14 days	18.10.06	18.10.19											
Design Report (Rough Draft)	28 days	18.10.20	18.11.16											
Design Report (Final)	5 days	18.11.26	18.11.30											
Minor Development (Base System)	50 days	18.11.17	19.01.06											
Winter Quarter														
Initial Operating System	67 days	19.01.07	19.03.15											
Design Review	18 days	19.01.07	19.01.25											
Design Report (Revised)	7 days	19.01.26	19.02.01											
Testing	21 days	19.02.23	19.03.15											
Spring Quarter														
Recommended Requirements	28 days	19.04.01	19.04.28											
Demo System	11 days	19.04.29	19.05.09											
Testing	18 days	19.04.15	19.05.02											
Design Presentation	18 days	19.04.22	19.05.09											
Project Report	28 days	19.05.10	19.06.07											
Final Implementation	28 days	19.05.10	19.06.07											

Table 9.1: Development Timeline.

Chapter 10

Future Work

10.1 Current State

Our project is a working proof of concept for a successful affordable option for hearing impaired individuals. It is not a perfect system and components have more than is needed driving up costs; however, the benefit is that we use easily available devices that others can use to build upon the product. Since Node-RED is open source, members of the community can easily add nodes that benefit from our system.

10.2 Next Steps

Since the architecture is in place, a company could build a viable product. The main missing part to our system is a completely functioning smartphone application like in Figure 10.1. The purpose of the smartphone application is to abstract user control of the system. The functionality is all possible but scripts have to be made to modify the necessary parts of the Node-RED application.

Another benefit would be to make a Raspberry Pi image that works directly as is. This would make it easier for users to download software for the Raspberry Pi. Our current implementation in Node-RED is already set to work on start up. More abstraction would allow this system to not only be useful for those that are familiar with the Raspberry Pi ecosystem but for the average user.

Figure 10.1: Mobile App Design

Chapter 11

Lessons Learned

We learned a lot about the IoT world. Before this project, we werent even aware of 802.15.4 which became the main communication protocol in our system. We also learned to make radical changes to the system where necessary when the original plan does not work. Finally, we learned the different advantages and disadvantages to how IoT devices are currently implemented.

Chapter 12

Obstacles Encountered

12.1 Description

In this section, we discuss the obstacles associated with this project that led to delays in development and different solutions to be created from our original design.

12.2 SensorTag

The main obstacles we encountered within our design involved the SensorTag device from Texas Instruments. In trying to use the Zigbee firmware, we were unable to communicate between the SensorTag and XBee. Using SensorTags SmartRF Packet Sniffer 2 software, we were able to configure the SensorTag as a sniffer module. We found the packets to be very different from one another. In the SensorTag packet (Figure 12.1), it is very simple with a length value, sequence number, and data following the start frame delimiter. The XBee packet (Figure 12.2) looks largely different. The packet is shown in hex as some values are not human-readable. It starts with a sequence number (0x1d), has various hex values, and then data (0x61 - 0x65). Even more peculiar, we find there to be a second sequence number (0x08). The data between the sequence numbers is unknown to us. With a few parameter changes to the XBee we are able to modify the packet a little bit but we were unable to have it coincide with the SensorTags packet which was not modifiable.

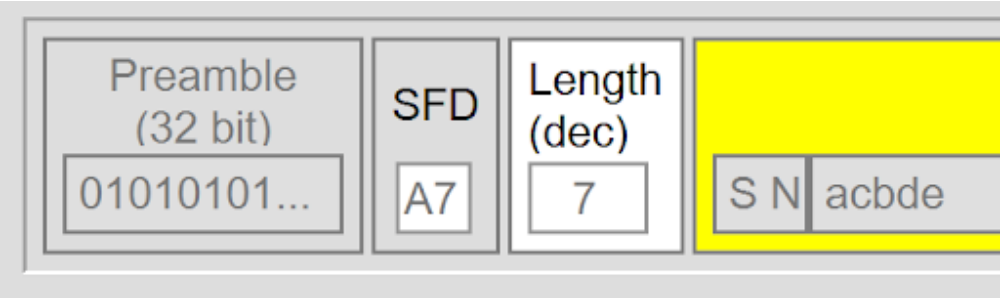

Figure 12.1: SensorTag 802.15.4 Packet

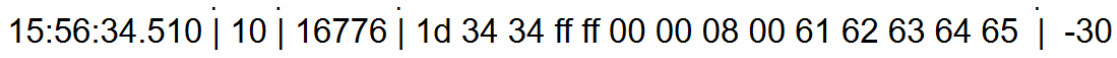

Figure 12.2: XBee 802.15.4 Packet

We decided that it might be possible to use Bluetooth instead. The SensorTag could connect directly to the Raspberry Pi Zero W and reduce a cost of our system by not using the XBee. Unfortunately, we found SensorTag to be a difficult device to program. The IDE had a steep learning curve and documentation was difficult. Changes in documentation were not always reflected elsewhere causing confusion.

12.3 XBee

We encountered some confusion with XBee similarly to the SensorTag. There are two main variants to XBee that we were looking at: the S1 and S2C. It seemed like the S2C was an updated module but instead it was designed for mesh network implementations, which was unnecessary for our design. It was also thought that the XBee S1 could communicate with the S2C variant. That was not the case. The reasoning was unclear but it appears that some of the hardware configuration is different at the Mac layer; thus, the two could not interpret the data of the other. This cost us many delays as we had to wait for more modules to ship to us in these cases and when once module was defective. In buying 5 XBees, one was defective. Hopefully, this is not indicative of the reliability of these modules.

Chapter 13

Ethics

The purpose of this project was to address the ethical issue of accessibility when it comes to people with hearing impairments and doorbells. This is best examined with the Markkula Center for Applied Ethics Justice and Fairness approach [1]. The relevant factors within the Justice and Fairness approach in this case include effort, contribution, and need. In terms of effort, we recognize that those with hearing impairments need to place more effort in order to be alerted about visitors without this system. This also creates a burden on other residents, so this system can lift the burden from all residents once properly set up in regards to visitors approaching. When thinking about contributions, we observed that this population has less opportunities to contribute to their community due to the inability to hear a doorbell. This system puts hearing impaired residents in a position of choice when responding to visitors. Since this is a proof of concept, we addressed the immediate needs of the hearing impaired community in regards to a doorbell. We look forward to users getting hands on with the system in order to better address their needs and concerns to better add and adjust features in the future.

Chapter 14

Conclusion

The main goal of the project was to improve the lives of the hearing impaired individuals. In most systems, scalability and affordability do not coincide often. Affordability was extremely important to us so that we can reach a wider audience. By using affordable, powerful devices and software we were able to achieve this. Considering the time for this project, building a solution from the ground up was out of the picture. Therefore, we hope this can be an inspiration for an affordable IoT system that the hearing impaired community can take advantage of, while also being an affordable IoT home system for any user, bringing IoT home applications to more users.

Bibliography

[1] Ethical Decision Making, 2015. accessed May 31, 2019 https://www.scu.edu/ethics/ethics-resources/ethical-decision-making/.

[2] Antnio Marcos Alberti, Marlia Martins Bontempo, Jos Rodrigo Dos Santos, Jr. Sodr, Arismar Cerqueira, and Rodrigo Da Rosa Righi. NovaGenesis Applied to Information-Centric, Service-Defined, Trustable IoT/WSAN Control Plane and Spectrum Management: 2018. *Sensors*, 2018. accessed May 29, 2019 https://doi.org/10.3390/s18093160.

[3] Diffen.com. Bluetooth vs. Wi-Fi, 2018. accessed May 31, 2019 https://www.diffen.com/difference/Bluetooth$_{v}s_{W}ifi$.

[4] Carrie Lou Garberoglio, Stephanie Cawthon, and Mark Bond. Deaf People and Employment in the United States: 2016. *Washington, DC: U.S. Department of Education, Office of Special Education Programs, National Deaf Center on Postsecondary Outcomes*, 2016. accessed May 29, 2019 www.nationaldeafcenter.org/sites/default/files/Deaf%20Employment%20Report$_{f}inal.pdf$.

Appendix A

Source Code

All code for this project is available to view at github.com/Magdaluyo/SeniorDesign

A.1 Philips Hue Code

```
#!/usr/bin/python
import time
from phue import Bridge

b = Bridge('192.168.0.123')

#b.connect()

b.set_light(1, 'on',True)

for x in range(10):
        b.set_light(1, 'bri', 127, transitiontime=1)

        time.sleep(0.5)

        b.set_light(1, 'bri', 63, transitiontime=1)

        time.sleep(0.5)

b.set_light(1, 'on', False)
```

A.2 XBee Code

```
import serial
import time

ser = serial.Serial("/dev/ttyUSB0", baudrate=9600)

while True:
        data = ser.readline()
        print(data)
```

A.3 Doorbell Code

```
/*************************************************************
Set up a software serial port to pass data between an XBee Shield
and the serial monitor.

Hardware Hookup:
  The XBee Shield makes all of the connections you'll need
  between Arduino and XBee. If you have the shield make
  sure the SWITCH IS IN THE "DLINE" POSITION. That will connect
  the XBee's DOUT and DIN pins to Arduino pins 2 and 3.

*************************************************************/
// based on code from https://learn.sparkfun.com/tutorials/xbee-shield-hookup-guide/all
#include <SoftwareSerial.h>

//For Atmega328P's
// XBee's DOUT (TX) is connected to pin 2 (Arduino's Software RX)
// XBee's DIN (RX) is connected to pin 3 (Arduino's Software TX)
SoftwareSerial XBee(2, 3); // RX, TX

//Setup LED button
int LED = 9;
int button = 8;

void setup()
{
  // Set up both ports at 9600 baud. This value is most important
  // for the XBee. Make sure the baud rate matches the config
  // setting of your XBee.
  XBee.begin(9600);
  Serial.begin(9600);
  pinMode(button, INPUT_PULLUP); // button
  pinMode(LED, OUTPUT); // LED
  digitalWrite(LED, HIGH);
}

void loop()
{
  int val = digitalRead(button);
  if (val == LOW) { // if button is pressed, send packet and flash button light
    //while(XBee.available()) {}
    XBee.write("DOORBELL\n");
    while (val == LOW) {
      digitalWrite(LED, LOW);
      val = digitalRead(button);
    }
    digitalWrite(LED, HIGH);
    pulsate_led(LED);
  }
}

void pulsate_led(int ledPin) {
  // based on code from https://www.arduino.cc/en/tutorial/fading
  for(int i = 0; i < 30; i++) {
    for (int intensity = 0 ; intensity <= 175; intensity += 5) {
```

```
      analogWrite(ledPin, intensity);
      delay(10);
    }
    for (int intensity = 175; intensity >= 0; intensity -= 5) {
      analogWrite(ledPin, intensity);
      delay(10);
    }
  }
  digitalWrite(ledPin, HIGH);
}
```

A.4 Node-Red Code

The code in this section is the implementation of the Node-Red flowchart in Figure A.1

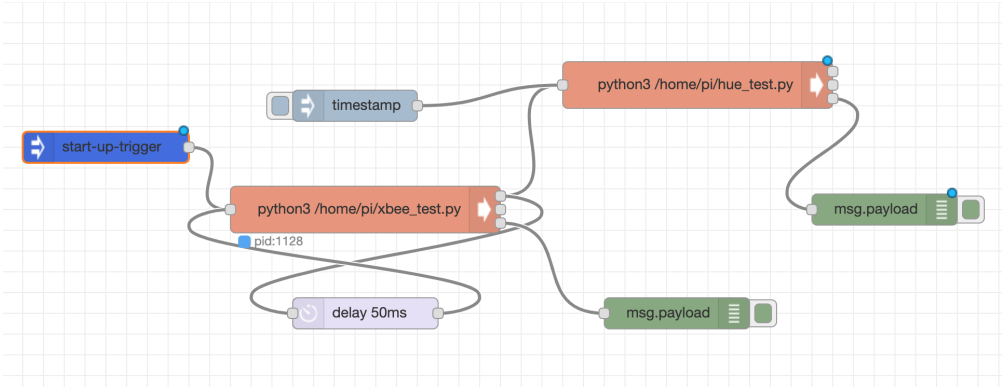

Figure A.1: Node-Red Flowchart

```
[
    {
        "id": "f5bd703e.670cf",
        "type": "tab",
        "label": "Flow 2",
        "disabled": false,
        "info": ""
    },
    {
        "id": "77e6b858.296d78",
        "type": "start-up-trigger",
        "z": "f5bd703e.670cf",
        "x": 120,
        "y": 160,
        "wires": [
            [
                "d38a81cd.93dc4"
            ]
        ]
    },
    {
        "id": "d38a81cd.93dc4",
        "type": "exec",
        "z": "f5bd703e.670cf",
```

```
    "command": "python3 /home/pi/xbee_test.py",
    "addpay": true,
    "append": "",
    "useSpawn": "false",
    "timer": "",
    "oldrc": false,
    "name": "",
    "x": 370,
    "y": 220,
    "wires": [
        [
            "8f3679ef.1c53d8",
            "70a54f5c.fed12"
        ],
        [],
        [
            "ca33c4dc.f52ab8"
        ]
    ]
},
{
    "id": "ca33c4dc.f52ab8",
    "type": "debug",
    "z": "f5bd703e.670cf",
    "name": "",
    "active": true,
    "tosidebar": true,
    "console": false,
    "tostatus": false,
    "complete": "false",
    "x": 670,
    "y": 320,
    "wires": []
},
{
    "id": "8f3679ef.1c53d8",
    "type": "exec",
    "z": "f5bd703e.670cf",
    "command": "python3 /home/pi/hue_test.py",
    "addpay": true,
    "append": "",
    "useSpawn": "false",
    "timer": "",
    "oldrc": false,
    "name": "",
    "x": 690,
    "y": 100,
    "wires": [
        [],
        [],
        [
            "740086bb.85aed8"
        ]
    ]
```

```
},
{
    "id": "740086bb.85aed8",
    "type": "debug",
    "z": "f5bd703e.670cf",
    "name": "",
    "active": true,
    "tosidebar": true,
    "console": false,
    "tostatus": false,
    "complete": "false",
    "x": 870,
    "y": 220,
    "wires": []
},
{
    "id": "b9ab7ca2.f8883",
    "type": "inject",
    "z": "f5bd703e.670cf",
    "name": "",
    "topic": "",
    "payload": "",
    "payloadType": "date",
    "repeat": "",
    "crontab": "",
    "once": false,
    "onceDelay": 0.1,
    "x": 360,
    "y": 120,
    "wires": [
        [
            "8f3679ef.1c53d8"
        ]
    ]
},
{
    "id": "70a54f5c.fed12",
    "type": "delay",
    "z": "f5bd703e.670cf",
    "name": "",
    "pauseType": "delay",
    "timeout": "50",
    "timeoutUnits": "milliseconds",
    "rate": "1",
    "nbRateUnits": "1",
    "rateUnits": "second",
    "randomFirst": "1",
    "randomLast": "5",
    "randomUnits": "seconds",
    "drop": false,
    "x": 370,
    "y": 320,
    "wires": [
        [
```

```
                "d38a81cd.93dc4"
            ]
        ]
    }
]
```