Base-IC Home Automation

Rusty Griggs, Preston Baker, Brennon Loveless
University of Utah

Abstract— **Home automation is a growing industry with hundreds of products on the market. Because these products are sold by different vendors, they are not made to communicate with each other creating difficulties for people who are becoming smart home owners. This project integrates a multi-faceted home automation system. By creating a base station that communicates with several different peripherals we completed three tasks: eliminating the need for wiring throughout the average household, increase consumer usability and customization, and reduce clutter.**

I. INTRODUCTION

It is the year 2017. Houses are still dumb, but cars can drive themselves? Imagine a world where the front door unlocks to greet the occupant upon arrival. The thermostat adjusts itself to the occupants' presence and preferences. These are the first world problems that have been solved. An affordable automated house has been created. The goal was to provide a very modular home automation system that can work for as cheap as a single base station, one peripheral, and downloading an app to one's phone. Now it is possible to add many peripherals and build the smart house from movies like 2001 Space Odyssey.

Home automation has existed for decades, as early as 1985 [1], though traditionally a home automation system was for the very wealthy and had to be incorporated into the home during construction. It required hard wiring every device with a centralized computer. This process of running wires manually throughout the home is why new home construction was required rather than simply retrofitting a home. As wireless networks and sensors have progressed, it is no longer necessary to hard wire every device together. This is what has been demonstrated in this project. No communication wires were used between the base station or any peripheral. The home automation system described in this paper communicates wirelessly using a distributed/meshed network as shown in fig. [2].

The current state of the art in home automation involves many different systems communicating to the user through many different input sources. For example, a light switch and a thermostat require specific inputs and varied outputs. But if the light switch and thermostat are controlled by a smart device, they would likely have their own management interface. In this example, the user would have to open one app to turn off their light, then open another app to turn down the thermostat. A true home automation system, where many peripherals come together to form an ad-hoc system, is not an accident. Each peripheral works through a single management interface and is designed and engineered to work together [3].

As computer engineers, designing a system that connects people to their home is an intriguing idea. The Internet of Things is the future and its potential should be studied and understood. Many things in homes already connect to the Internet and can be controlled wirelessly, such as a TV or a refrigerator. The problem is that all of the controls are disconnected and disjointed. The controls for items and appliances around the home have been centralized and incorporated into a unified home automation application. The authors have used the skills that have been developed over the course of their university education to build a system that seamlessly connects the occupant to the home. It has been proven that this type of project is easily modularized by developing a common protocol between the base station and the peripherals and have used the modularization of the project to develop and work on each peripheral individually. Only in the final months of development were all of the sections of the project brought together to form a unified home automation system.

The original idea and plan was to simply build a small automatic door. An automatic door shouldn't be so hard for a home since it seems to be ubiquitous at supermarkets and other public places. All that would be needed is some kind of door opening apparatus that is connected to an app on one's phone. Sensors would then be incorporated in the door so the door wouldn't swing open or closed if anything was in it's path. The thought of overcoming those obstacles and working together to achieve this goal is a strong motivation.

II. NETWORK CONFIGURATION

The first hurdle was deciding on the network configuration. A few modules were tested, specifically a TI CC1310 that used a hub-and-spoke style network configuration, and a Zigbee module implementing a meshing network configuration.

The first network considered was the hub-and-spoke style network using TI CC1310's. In this network configuration every device communicates directly with the base station which in turn relays the message to the relevant peripheral. As shown in Fig. 1.

The hub-and-spoke configuration has the advantage of being a simpler configuration when covering a small area. Each peripheral communicates directly with the base station and the base station communicates directly with the peripheral that it needs to send a request to. It also presents some limitations, namely the distance that can be covered. This configuration is most prevalent in WiFi networks where every

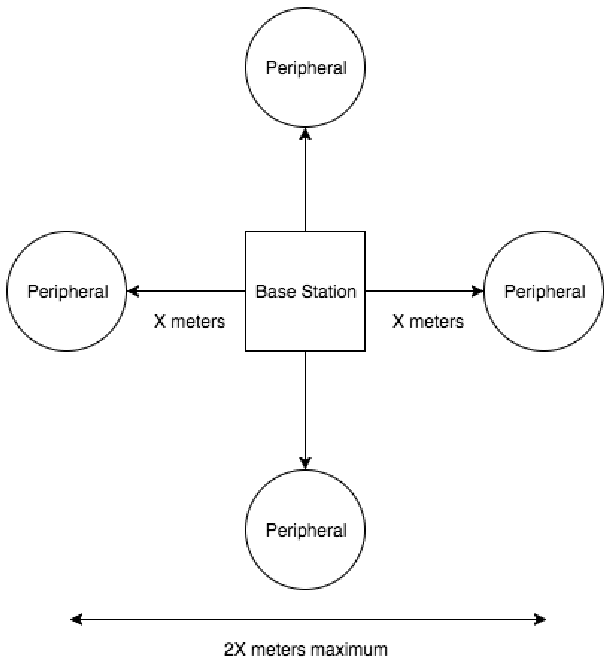

Fig. 1. Hub-and-spoke Network Configuration

device has to be in close proximity to the router. WiFi networks get around this limitation by allowing a single network to be represented by a number of routers with overlapping coverage. While this works for WiFi networks the complexity of a home automation network would increase dramatically by needing to have the base stations in contact with each other to coordinate tasks across multiple base stations.

Another trade-off for the hub-and-spoke modules is that to increase the range the transmission speed has to be drastically reduced. This is fine for simple messages, but when transmitting something like an IR hex code over the air this speed reduction can, perceivably, slow the network and degrade the user experience.

The next network configuration discussed is a mesh style network configuration shown in Fig. 2. In this configuration the peripherals would be able to communicate with each other, and each module would be able to relay the message to the correct location (I.E. to the base station).

The mesh configuration has the benefit of being able to cover much larger distances while maintaining high transmission speeds by relaying the messages through adjacent peripherals. The Zigbee modules used in the project implement Dijkstra's algorithm for finding the shortest path from peripheral to peripheral. This means that if an object were placed in the middle of two modules as demonstrated in

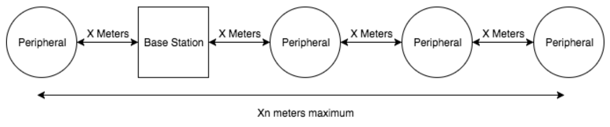

Fig. 2. Mesh Network Configuration

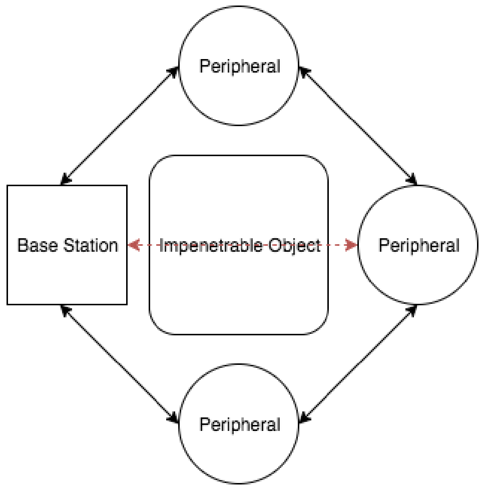

Fig. 3. Meshing Path Around Rather Than Through

Fig. 3 the network would naturally create a path around that object (given that peripherals exist around the object) rather than trying to blast a signal through the object to reach the other peripheral. Since lighting is so ubiquitous in home automation this makes for the perfect backbone to get around the entire network, through tough obstacles, and cover great distances. The routing logic is already built into the Zigbee module firmware and can simply be used, rather than having to be implemented.

While the hub-and-spoke network configuration is entirely possible, it was decided to go with a mesh network to reduce the complexity of the network as well as covering larger distances at a higher rate.

III. PROTOCOL

The next step was to design a protocol that peripherals would use to communicate to the base station. The major concern was how to support different types of inputs and outputs that a peripheral may use. For example, the 9 button keypad that was used during the demonstration of the project consisted of 9 toggle inputs both in hardware and in software. The curtain consisted of a single range output that controlled a stepper motor. The door consisted of 2 toggle inputs to control the deadlock and the opening/closing motors individually. The TV remote had a hex output that could output an IR signal used to control the TV. Last, the Android app that was used as part of the demonstration has a dynamic interface, shown in Fig. 4, that reads every peripheral from the network and displays the correct inputs to the user depending on the peripheral they wish to control. For example, the user could control the door as shown in Fig. 5, or the user could control the position of the curtain as shown in Fig. 6.

The protocol defined the way the peripherals were developed in addition to the hardware used for the peripherals. The protocol is hardware independent but was only implemented on the Zigbee modules for the purpose of the demonstration. The protocol, specifically the recipes used in the protocol

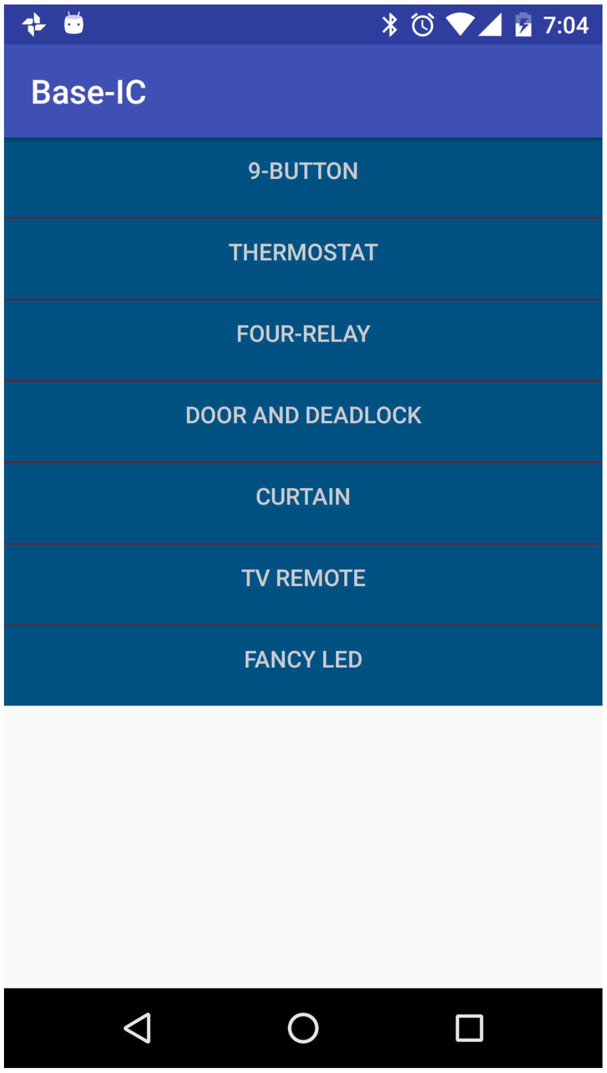

Fig. 4. Android app showing a list of all available peripherals.

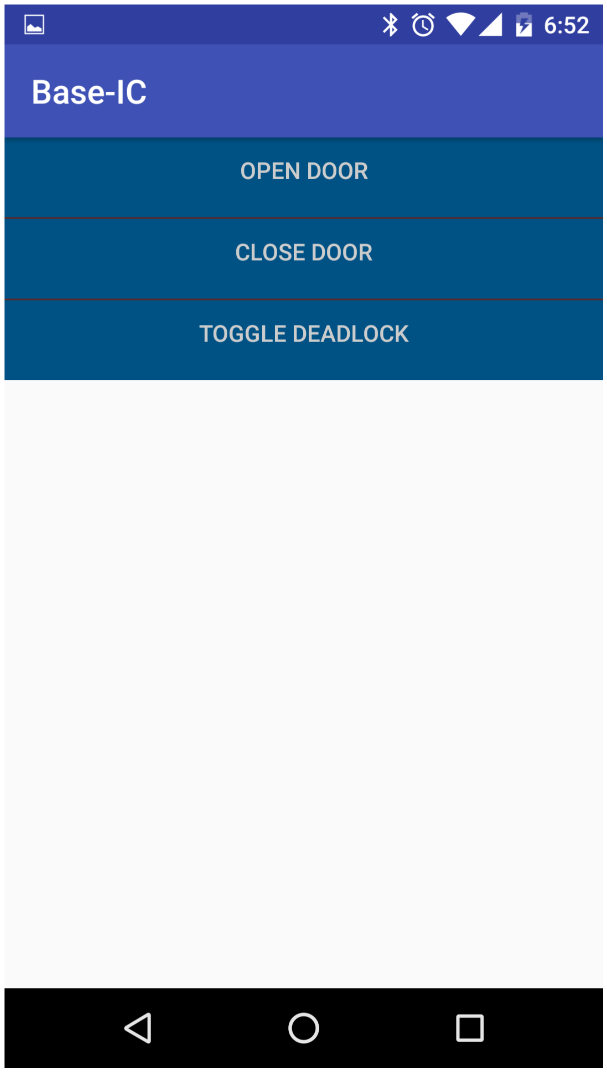

Fig. 5. Android app allowing the user to open/close the door and control the deadlock.

allow for the same protocol to be routed out to any device through the use of queues on the base station. For example, a signal could be received on the Zigbee device on the base station but then routed out via WiFi/ZWave/Bluetooth on the base station, making the protocol hardware independent.

Peripherals send out a packet over Zigbee containing the structure of the protocol. This packet is always routed to the base station since the base station is the only place that the recipes exist in the system. The base station also knows the addresses of the receiving peripheral or peripherals since the recipes are not restricted to a single input and a single output. Rather, they can have a single input and multiple outputs.

The peripherals communicate with the base station using a simple protocol that begins with a command number, then any command data separated by tabs that is necessary for that command to execute. The command is ended with a newline.

The commands and their explanations are as follows:

First, a peripheral registers itself with the base station. It will provide its name and all the input services and output services it provides which will automatically be enumerated as input service one, input service two, ... , input service n and the same for the output services. The first digit will always be the command number and the rest of the command will be dynamic depending on the command.

A. *Register a peripheral (command 0)*

0\t**PERIPHERAL NAME**\t**NUMBER OF INPUT SERVICES**\t**INPUT SERVICE TYPE**\t**INPUT SERVICE TYPE**\t**NUMBER OF OUTPUT SERVICES**\t**OUTPUT SERVICE TYPE**\t**OUTPUT SERVICE TYPE**\n

For example: **0**\t**Lamp Switch**\t**3**\t**1**\t**2**\t**3**\t**1**\t**1**\n: Register a peripheral with name “Lamp Switch” that provides three input services a hex service as input service one, a range service as input service two, and a toggle service as input service three, as well as one hex output service as output service one.

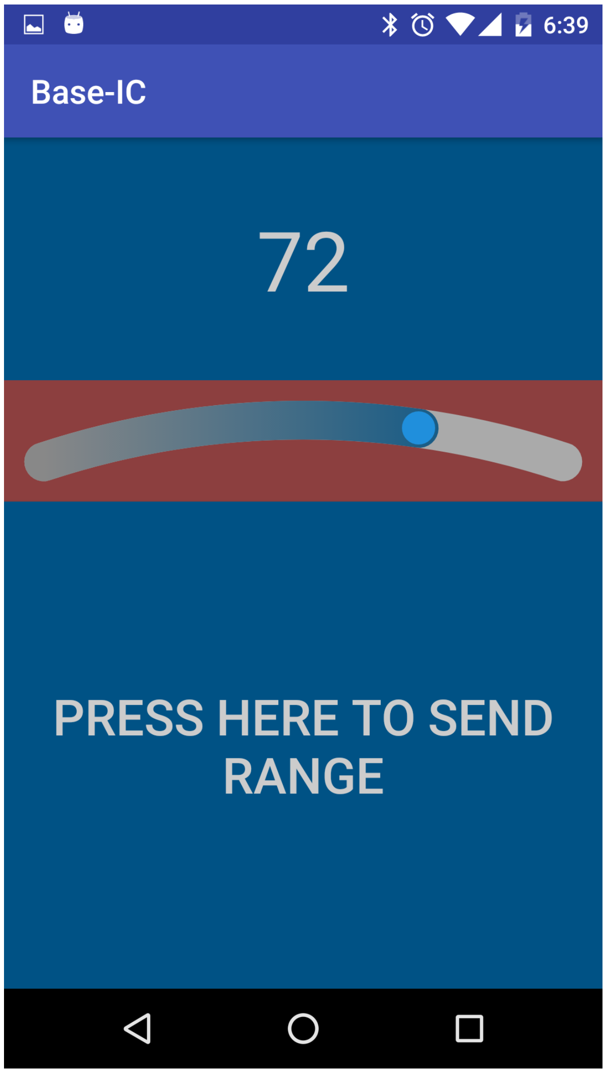

Fig. 6. Android app allowing the user to control the position of the curtain.

B. Send hex value (command 1)

1\tSERVICE NUMBER\tCOMMAND VALUE\n

For example: **1\t1\tFF0000\n**: Send hex FF0000 to service one.

C. Send a range value (command 2)

2\t**SERVICE NUMBER**\t**RANGE VALUE**\n

For example: **2\t2\t50\n**: Send 50% range to service two.

D. Toggle a service (command 3)

3\t**SERVICE NUMBER**\t**SERVICE VALUE**\n

For example: **3\t3\t01\n**: Toggle service three. Toggles also allow a value to be sent so a peripheral can be directed to a specific state if necessary; *i.e.* a door can toggle open/close, but it is also desired that the door is directed to close even if it is already closed. On this "toggle close" command, the door would do nothing if the door was already closed.

E. Read a value from a service (command 4)

4\t**SERVICE TYPE**\t**SERVICE NUMBER**\n

For example **4\t1\t1\n**: Read service value of type hex from service one.

The protocol supports modules registering themselves with the base station. As long as the peripheral and the base station are listening on the same channel the base station will allow the new module to register itself with the base station. After a module registers itself with the base station, the module provides a name to the base station along with whatever input and output services that peripheral provides. For example, a module will report its name is "Light Switch" and that it provides a toggle output service and a toggle input service. In most cases peripherals would provide an input service that is directly connected to it's own output service to allow the phone app to dynamically build the interface for that peripheral. This configuration allows the Android app to control the light switch without having another peripheral that is specifically built for controlling the light switch peripheral.

With only a few commands the protocol is able to support a multitude of peripherals. The peripherals that inspired the current protocol, along with the services they would provide, are as follows:

- Button Pad / Hex
- Light Switch / Toggle
- Power Outlet / Toggle
- Curtain / Range 0 - 100
- Door Open and Close / Toggle
- Door Lock / Toggle
- IR Emitter for TV control / Hex
- Light Dimmers / Range 0 - 100

In order to support the advanced interactions between peripherals that has been discussed the protocol will interact with recipes that exist on the base station. These recipes will take in commands from input services commands based on the input that was received. These recipes will follow the familiar pattern called If This Then That (IFTTT) [4].

Some examples of work flows are as follows:

- If Light L1 is turned on, then Send IR Code to TV1
- If temperature in house is greater than 90 degrees, then set window position to 25%
- If door is left ajar for 30 minutes, then close door

IV. RESULTS

The Base Station, shown in Fig. 8, is the center piece to the entire project. The Base Station is the simplest peripheral as far as the hardware goes, containing only a Raspberry PI and a Zigbee module, but it housed the protocol and recipes for the project. This is by design and allowed the peripherals to be very simple. The peripherals are responsible for relaying any input they received to the base station and responding to any messages received for themselves. The Base Station then took that information compared it with the recipes that are stored locally and sent out the appropriate responses to the network to control the rest of the devices. The Base Station also has a web interface that allows the user to create, edit, and delete recipes.

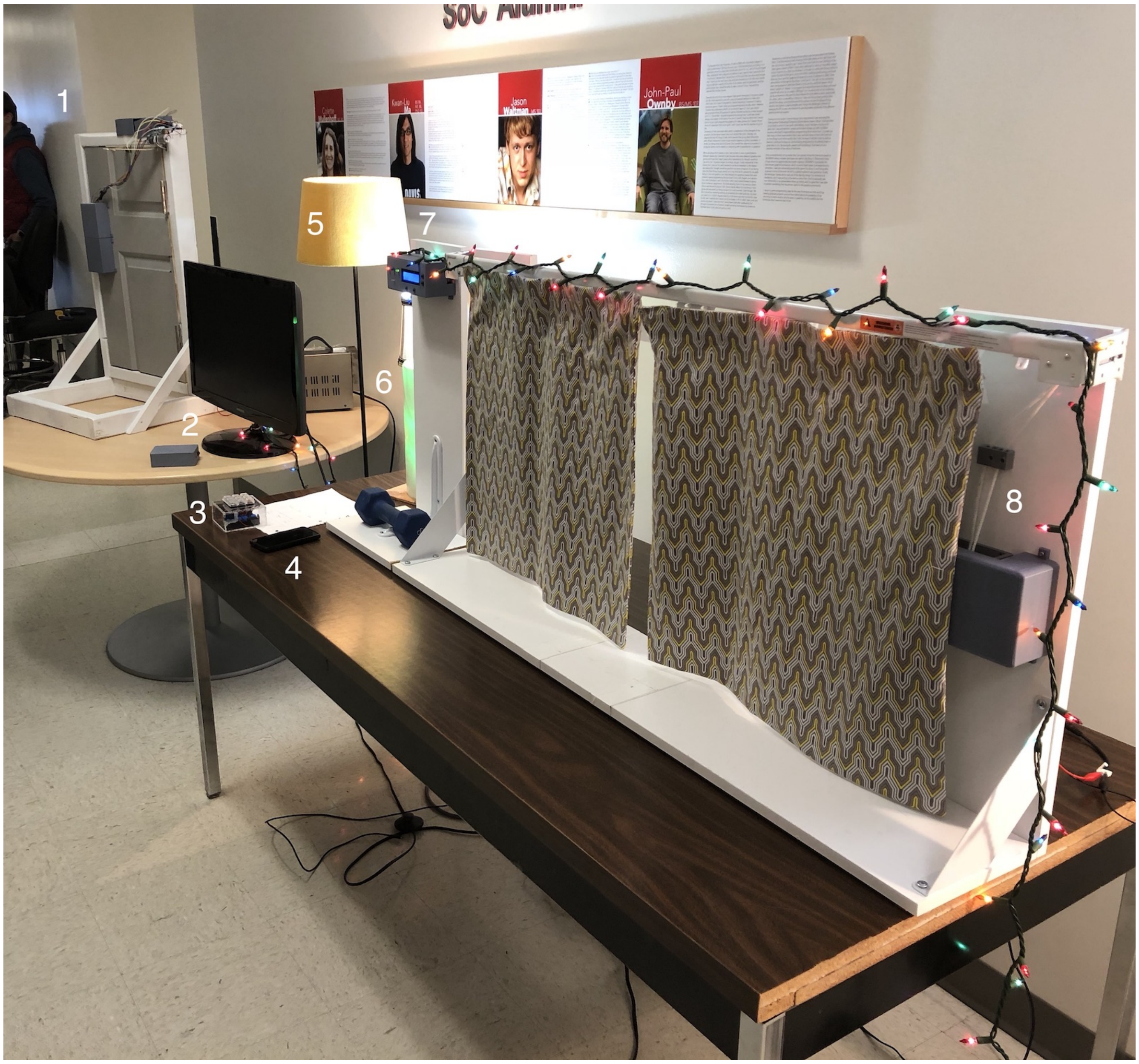

Fig. 7. Final Project

The Door Peripheral, Fig. 7 (1) and Fig. 9, proved to be a mechanically intensive project. Opening and closing the door turned out to be complicated and we had to try different motors to find one that would successfully open and close the door. Luckily, we made the door small enough that it could be powered by a very low geared DC motor. The door was also fitted with a motor to lock and unlock the deadbolt. The door was initially powered with batteries, but the power required to open and close the door meant that the batteries didn't last very long. We ended up powering the door using a bench top power supply.

The TV Remote, Fig. 7 (2), is an IR emitter with codes hard coded into the Arduino and controlled via presets. This works but could have been much better if we could send the codes from the Base Station to the peripheral, rather than relying on the peripheral to store the codes. The codes are huge and take up a significant portion of memory. We also came across issues trying to transmit the codes from the Base Station to the peripheral (although this is the way it should have been done), in addition to this we needed to have a delay where we could send a code from the Base Station to the peripheral, and after a number of seconds send another code. For example, sending the power command then waiting a number of seconds, then sending the channel commands. Currently this is not possible.

The 9-Button Keypad, Fig. 7 (3), is a very simple peripheral. The Arduino has nine buttons attached directly to it. Whenever the Arduino senses that one of the buttons has been depressed it sends the relevant signal to the Base Station. The Base Station then matches which button was pressed with a recipe and sends out the appropriate signals to the network.

The Android App, Fig. 4 Fig 5 Fig. 6 and Fig. 7 (4), worked by making an API call to the Base Station. This API call returned the entire state of the network including all the inputs that are available on the network, which was used to generate the app interface. The app then displayed the peripherals that existed on the network, this allowed the app to impersonate any peripheral on the network and send commands to the Base Station for the impersonated peripheral.

The Four Relay, Fig. 7 (5) and Fig. 10, is used to control four wall power outlets. Each outlet is wired individually to a relay which is wired inline to break the circuit to the outlet when the relay is disengaged. The four relays are connected to an Arduino and Zigbee module. The Base Station sends control signals to the Arduino to turn on or off each outlet.

The Fancy LED, Fig. 7 (6) and Fig. 11, consists of a one meter strip of RGB LED's that are individually addressable. This allows for animations as well as single colors from the Fancy LED. The Arduino again became an issue here and was running out of RAM while attempting to receive communications from the Base Station and playing the animation. This could have been alleviated by using a more powerful micro-controller that can run a real time operating system, such as FreeRTOS. This would allow the peripheral to schedule tasks like playing the animate and checking for messages on the Zigbee network making them appear to be running at the same time.

The Thermostat, Fig. 7 (7) and Fig. 12, while not connected to an actual heating and cooling system, simulated this by displaying a red LED when the system would be heating, displaying a blue LED when the system would be cooling, and a green LED when the system would enable the fan. The Thermostat worked by sensing the ambient temperature and working to make that temperature match the temperature that the thermostat was set to. The Base Station would send a preset number to the thermostat which would configure the thermostat to be in one of three modes: heating, cooling, and off.

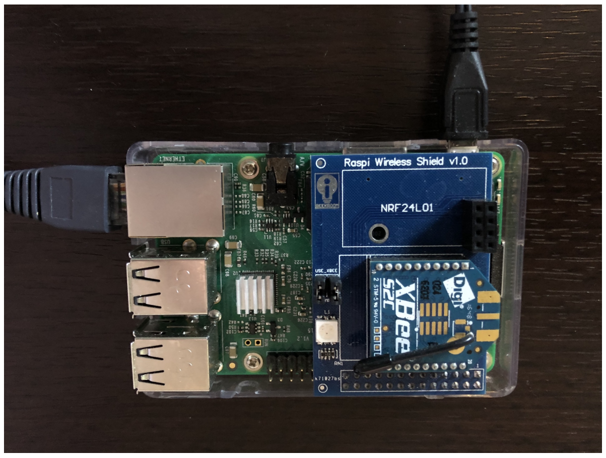

Fig. 8. Base Station

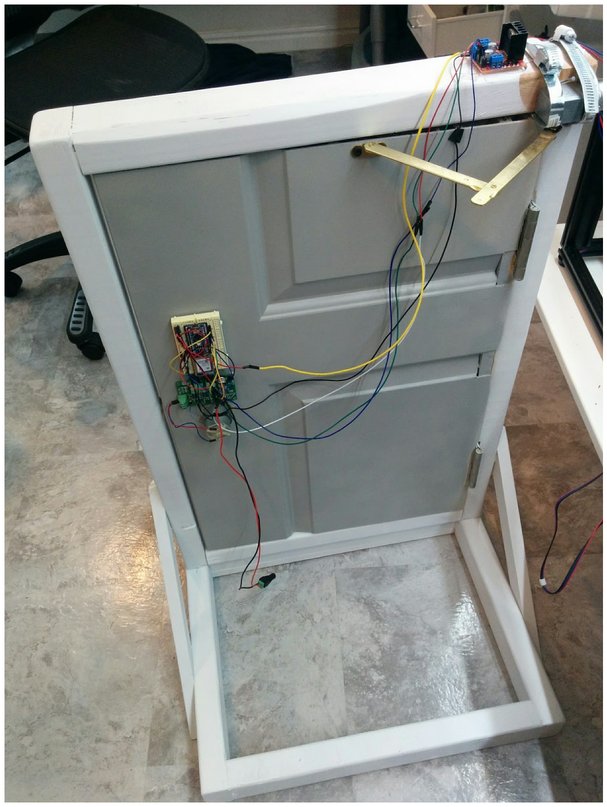
Fig. 9. Door

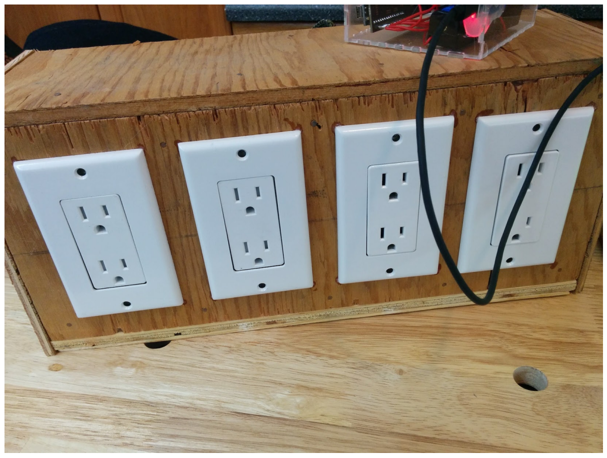
Fig. 10. Four Relay

The Curtain, Fig. 7 (8), was controlled by a range which allowed the curtain to be placed at any position between 0% and 100% open. This peripheral can only be controlled by the Android app since there is no potentiometer input to the system. Initially we thought that the curtain was going to have two stepper motors one to control the opening and one to control the closing of the curtain, but with a little engineering we were able to 3D print a device that would

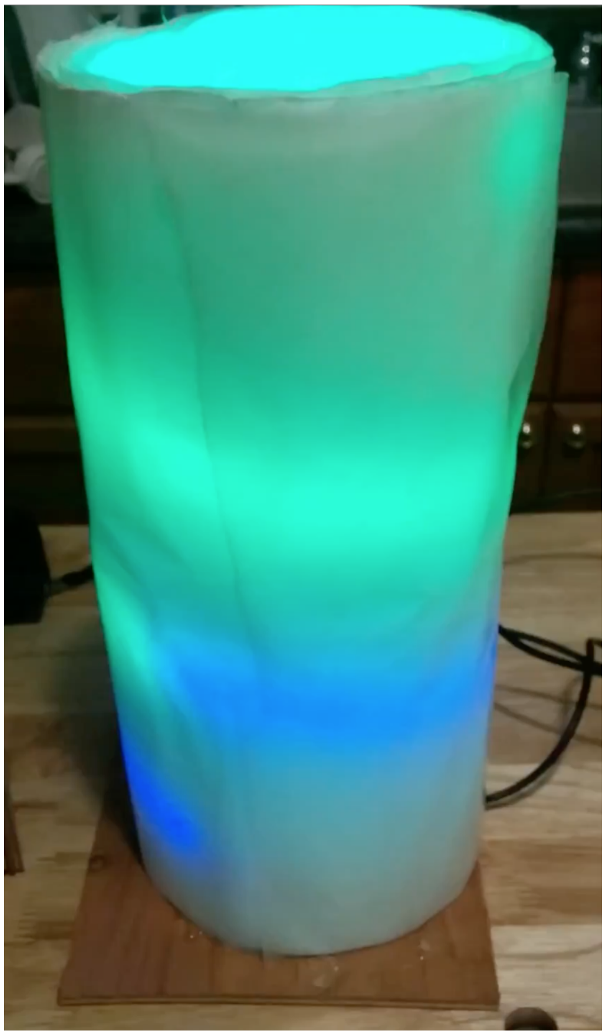

Fig. 11. Fancy LED

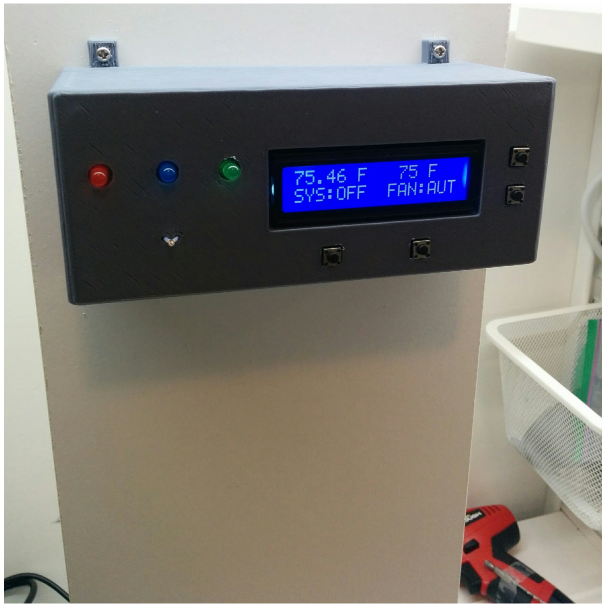

Fig. 12. Thermostat

allow for the curtain to be opened and closed with a single peripheral.

A demonstration of the final project can be found on YouTube [5].

V. FUTURE

This project was built to be a proof of concept that an open home automation system is possible. After doing research on newer technologies (I.E. Bluetooth Low Energy Mesh) it seems that the protocol should evolve in that direction rather than using Zigbee. Bluetooth Low Energy (BLE) has a security built directly into the core of the specification while the Zigbee implementation used in this project had no security at all.

While security is also built into the Zigbee specification the process has a few flaws. First and foremost mobile devices such as a cell phone do not have Zigbee built in. Meaning that in order to configure a Zigbee device, or provide a unique encryption key (similar to a WiFi password) means that the Zigbee device would have to join the Zigbee network unencrypted initially in order to receive the encryption key which would be sent over an unencrypted RF connection. Also, since the idea behind this home automation system is to be as open as possible it is not possible to hide any default encryption key in the source.

Alternatively, using a BLE Mesh would allow the user to install the device, connect to it with a smart phone, and configure the device to join the BLE mesh network using a custom password. BLE has methods for upgrading a pairing to an encrypted connection so the user can securely send the custom password to the new device. This process would be as follows:

- The user would install the device. In the example of a controlled power outlet the user would insert the Base-IC device in between the wall and the utility that is going to be configured.
- The Base-IC device would then determine that is cannot connect to a mesh network, either because the mesh network it was connected to before is now unavailable, or because it has never been connected to a mesh network before.
- The Base-IC device would put itself into a configuration mode where the user can use a phone app to choose from a list of available mesh networks and enter the password securely into the device.
- The Base-IC device would store the mesh network and the mesh network password, so when the device is restarted it will automatically re-connect to the last mesh network.
- The Base-IC device would then disable the configuration interface and connect to the mesh network where any further configuration can occur through the home automation app.

The beauty about this is that the device only needs a single BLE capable RF radio, and the app that is used to control the home automation network is the same app that is used to configure the new Base-IC devices.

References

[1] [Online]. Available: https://www.youtube.com/watch?v=0BHIknNa6Eg

[2] C. Badica, M. Brezovan, and A. Badica, "An overview of smart home environments: Architectures, technologies and applications."

[3] W. K. Edwards and R. E. Grinter, "At home with ubiquitous computing: Seven challenges."

[4] L. W. Braun, "Life hacking with ifttt: 'if this then that' links your applications to help streamline your life." *School Library Journal*, vol. 2, p. 15, 2013. [Online]. Available: http://link.galegroup.com.ezproxy.lib.utah.edu/apps/doc/A342467280/AONE

[5] P. B. Rusty Griggs, Brennon Loveless, "Demo day university of utah," 2017. [Online]. Available: $https : //youtu.be/59mXbl6_lT8$