

University of Missouri - Columbia

# Remote Light Control

Department of Computer Engineering

ECE 4220 - Section A

Embedded Computing - SS2015

*Friday 09:00*

Benjamin Temple

May 14, 2015

# Contents

Abstract	5
Introduction	5
Background	5
Implementation	6
Experiment and Results	10
Conclusion	12
Appendix A: Server Code	14
Appendix B: Arduino Code	19
Appendix C: Android Code	29
Appendix D: Readme	41
References	43

# List of Figures

1	Basic System Block Diagram . . . . .	7
2	Android Application Block Diagram . . . . .	8
3	Server Block Diagram . . . . .	9
4	Arduino Block Diagram . . . . .	10

# List of Tables

1	Arduino Connection Table . . . . .	11
---	------------------------------------	----

## Abstract

My remote light control system is a system which focuses on intelligent and rapid control of a home's lights. Not only should a home's lights be controllable by a quick and intuitive light switch, the user should also be able to seamlessly control the lights with a mobile application. This application should be able to rapidly and reliably control the lights yet not override the physical wall switch.

## Introduction

The focus of this project was not on the actual lighting circuitry but instead on the transmission of the commands to control the lights. These commands needed to be sent as rapidly and reliably as possible. Another requirement for this project is the intuitive nature of the control. Not only should a smartphone app be able to control the lights, but the physical wall switch overrides the software commands and is simple and easy to use. This wall switch is absolutely necessary since the majority of the time a user will still use a wall switch to control the lights over the mobile application. A user will far more often push a button whereas the other form of control actually requires the user to open the phone, then the app, and finally issue the desired command. The phone app is useful for when the user is sitting down, already occupied, or already has their phone out, but many times this will not be the case.

For my project, the rapid and reliable transmission of commands is achieved through an Android app which simultaneously sends commands by both WiFi in the form of a UDP packet and over bluetooth. These commands are overridden by the physical rocker switch since if there is a user present controlling the lights physically, that should certainly override any software commands.

## Background

My solution addresses several problems with current solutions on the market currently. Many home automation systems currently on the market like X10 are extremely clunky and cumbersome to use. X10 is plagued with a lack of intuitive controls and requires the user to use either an unintuitive push button switch for dimming and controlling lights or a physical remote control. X10 doesn't support any form of mobile app and is slow and difficult to use. Other home automation systems such as the Z-Wave system which do allow the use of a mobile app with the proper modules installed requires the user to purchase special light bulbs and switches. These light-bulbs and switches are also unreliable as has been pointed out by multiple reviews from users complaining that the system only works some of the time. This is largely related to the fact that the system solely relies upon wifi to receive commands from the user's android app. There are also complaints of the wall switches not working

reliably as well.

As a whole, the current market lacks a product that is both easy to use, rapid, and extremely reliable.

## Implementation

My proposed method is to use multiple forms of transmission to ensure that the command is not only sent reliably but also as rapidly as possible.

For the mobile application I send both a Bluetooth and a UDP packet simultaneously. These packets contain device information, to specify the device to be controlled, a unique ID to prevent the same command from being processed multiple times, and then the actual value that the light should change to. Once either of these packets are received by the Arduino it saves the command into a linked-list of the last 100 commands. If a repeat command is received, such as the most commonly slower UDP packet, it is ignored if a command with a matching ID has already been processed.

The UDP packet is received initially by a Raspberry Pi which acts as a server forwarding network packets on to the Arduino via SPI communication.

The hardware control is two simple push buttons. These push buttons simulate a paddle switch which would be installed into a wall. This rocker switch would merely be setup in an identical fashion to the test case, directly to the Arduino, allowing the user to immediately control the lights without needing to use the Android app. I chose the following implementation of interpreting commands as the most intuitive:

A single tap on the top button longer than 50ms and shorter than 250ms will result in the light fully turning on.

A single tap on the bottom button longer than 50ms and shorter than 250ms will result in the light fully turning off.

A press on the top button longer than 250ms will begin brightening the light from the 250ms mark until the light is fully on. The light increases at a rate of  $1/255$  of the lights total brightness every 7ms.

A press on the bottom button longer than 250ms will begin dimming the light from the 250ms mark until the light is fully off. The light decreases at a rate of  $1/255$  of the lights total brightness every 7ms.

One key aspect of the hardware control is that it always will override any software commands received by the Arduino since a software command has a lower priority than a physical action by a user. By setting up the hardware control for the light in the manner mentioned above, the user is able to easily and rapidly control the state of the light in a very simple easy to understand manner. Any user who has never used the light switch before should be able to understand how it functions at least on a binary level (on or off) within a matter of seconds. A rocker switch accomplishes this objective.

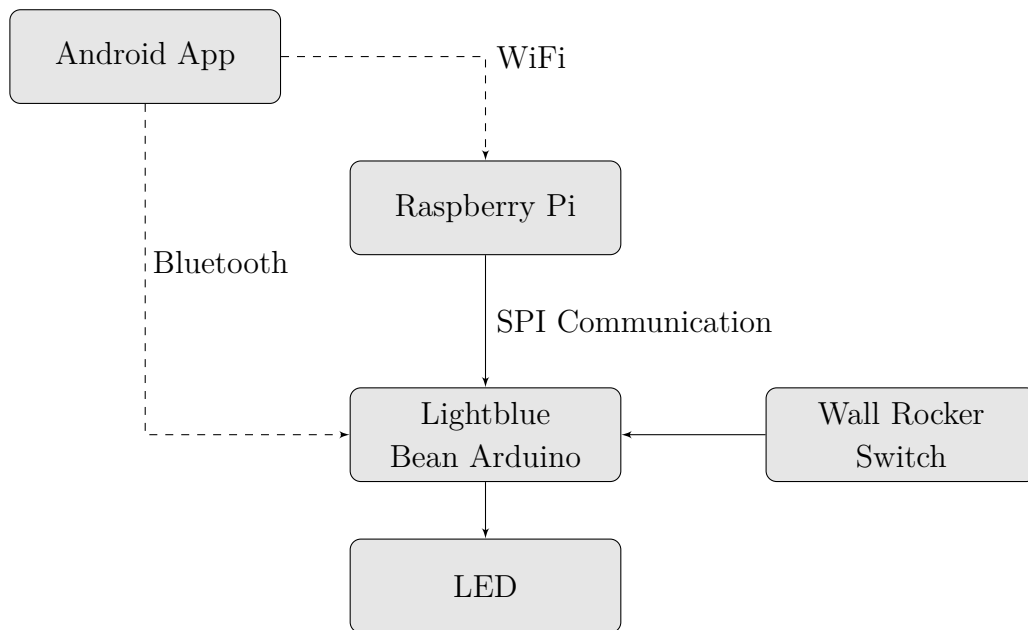


Figure 1: Basic System Block Diagram

Figure 1 shows a basic block diagram of how my system operates on a high level.

There were three applications that I had to implement. The first of these systems was the Android app which sends commands to the Raspberry Pi via UDP and Arduino via Bluetooth.

When the app is first loaded, it begins to attempt to connect to the Arduino via Bluetooth. A connection must be established before it can send packets via Bluetooth.

When the user performs some action with the Android app, the app first generates a unique ID. With this unique ID and the light brightness information which is a value from 0-255 in an asynchronous task it generates a UDP packet from the bytes of the command and sends it as via UDP broadcast to the Raspberry Pi which is listening on the corresponding port. It also sends an identical packet via Bluetooth to any connected Arduino's.

Figure 2 shows how the Android app functions on a more detailed level.

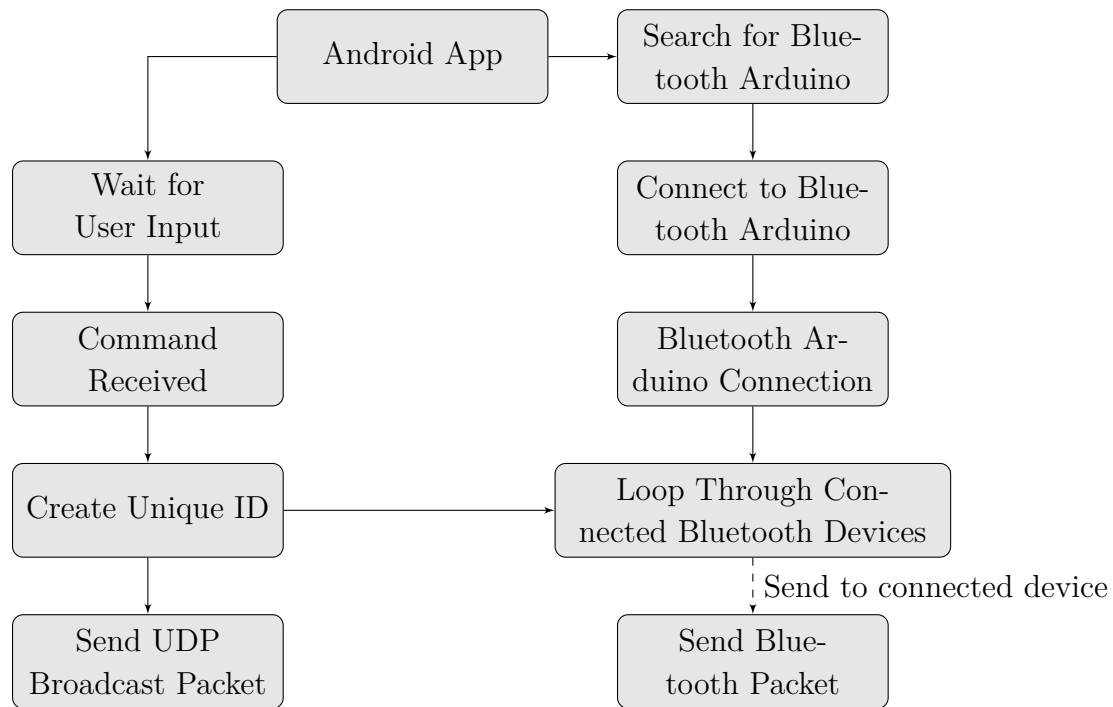


Figure 2: Android Application Block Diagram

The server application which runs on the Raspberry Pi was much simpler implementation.

It first initializes itself by reading its ip address so it knows the broadcast address to use, and it also initializes the SPI communication specifying the clock speed and pin to use for Slave Select. The clock speed I chose for SPI communication is 614400 which is a multiple of two. This did not necessarily have to be a multiple of two but I chose it for simplicity sake. With SPI communication, the clock signals synchronize automatically so the master (Raspberry Pi) can choose any clock speed, and the slave will automatically sync to this speed.

The server application then infinitely loops reading UDP packets from the specified or default port (5000). When a packet is received, it forwards the packet to the Raspberry Pi via SPI.

To communicate via SPI the Raspberry Pi must first bring a slave select line low to inform the Arduino that it is beginning to send a message. The Raspberry Pi then sends the message, and finalizes the communication by bringing the Slave Select high again.

Figure 3 shows a detailed block diagram of how the server functions.



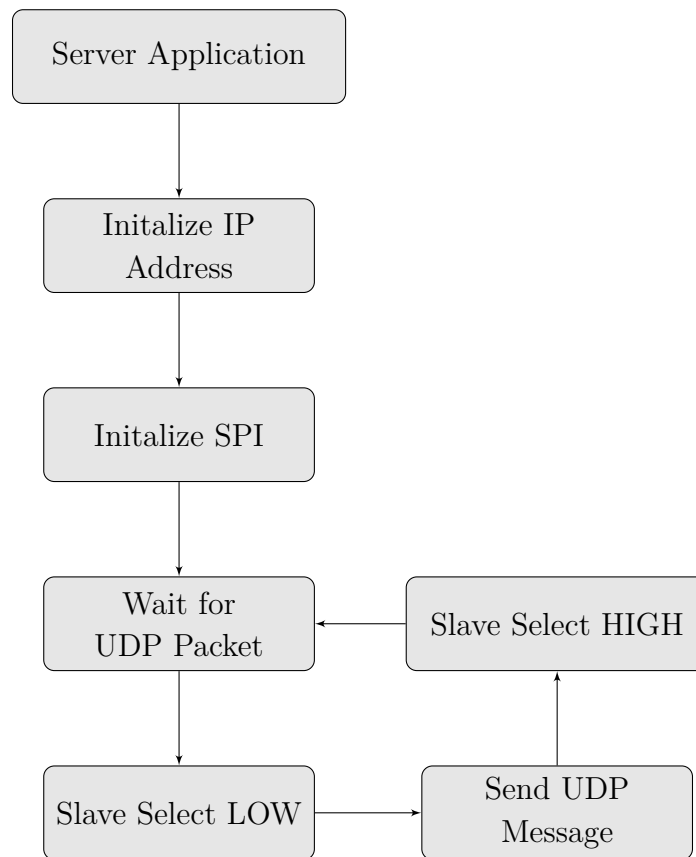


Figure 3: Server Block Diagram

Finally the Arduino application must receive all the commands and also simultaneously check for button presses. Once a button is pressed, a button direction key is set which prevents any software requests to control the light from having any effect. It then performs the corresponding action requested by the physical button.

If no key press is present, and a packet is received, the Arduino first parses the message into separate parts. Once parsed, the message is passed to a handler which checks to make sure that an identical command hasn't already been processed. If the command has already been processed, it is discarded. If the command has not already been processed, the Arduino saves the command to a linked-list of commands with a maximum size of 100. It then processes the command and returns to the main loop.

The main loop, when not handling commands from either the hardware switches or the software, is constantly checking the variable set by the software requests which informs the main loop if it needs to dim or brighten the light. If the server finds that it does need to change the light, it incrementally brightens or dims it based on a time factor of  $1/255$  ever 4ms. This increment is slightly faster than that of the hardware switches since a user can select the light's brightness instantly with a slider versus the hardware buttons where the user needs to be able to stop the light dimming or brightening at the desired brightness.

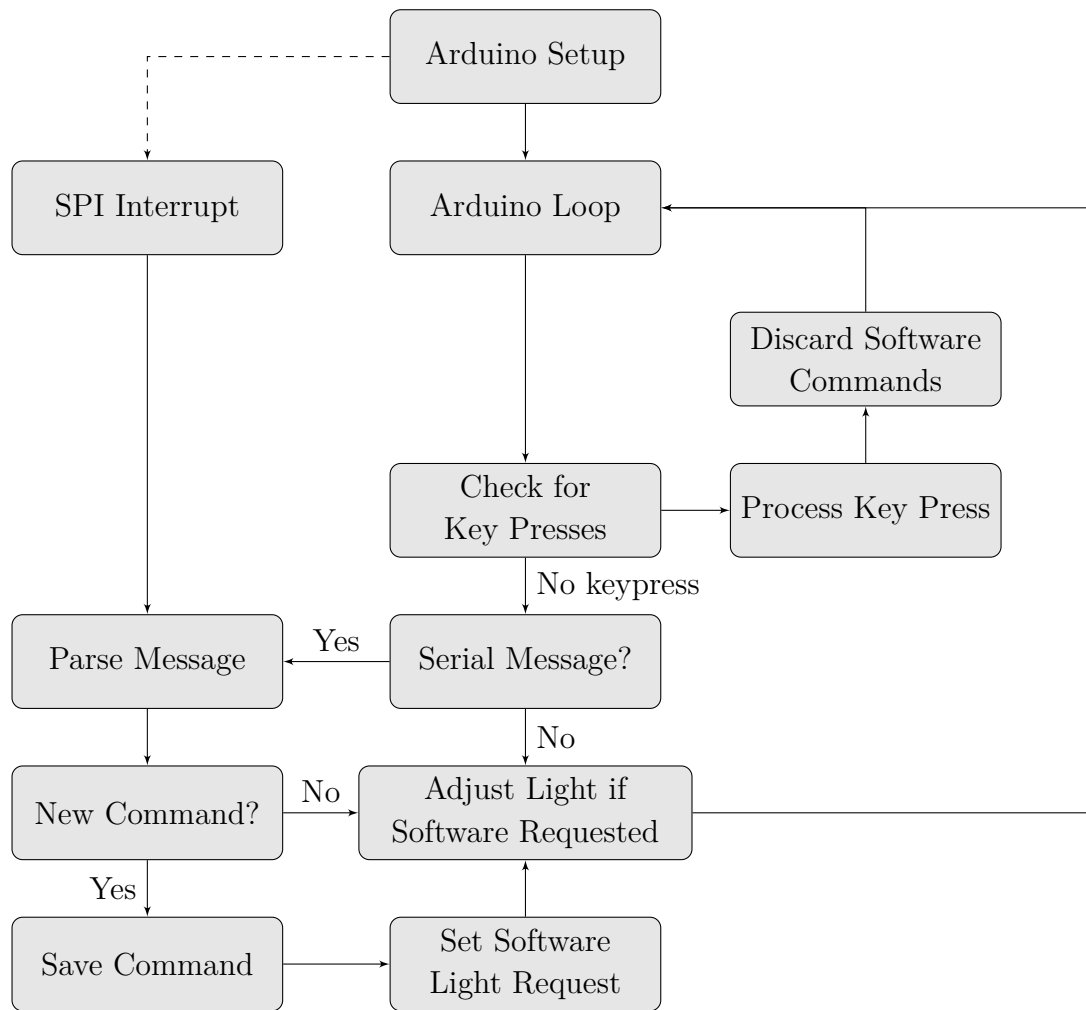


Figure 4: Arduino Block Diagram

Figure 4 shows in detail how the Arduino code functions.

Table 1 shows in detail how to connect the light blue bean with the other circuit components.

## Experiment and Results

To test my application I had to go through many stages of development.

Initially I created a socket server on the Raspberry Pi. Then I created my basic Android app and sent a packet to the Raspberry Pi.

The next step was to connect my Android app to the Arduino via bluetooth. This took a bit more work but wasn't too difficult.

Then I needed to actually transmit a packet to the Arduino from the Raspberry Pi. It

Lightblue Bean Pin	External Device
Pin A0	Blue LED (Bluetooth Indicator), 220 $\Omega$ Resistor in series
Pin A1	Red LED (UDP Indicator), 220 $\Omega$ Resistor in series
Pin 0	Up Push Button, 10k $\Omega$ Resistor
Pin 1	Down Push Button, 10k $\Omega$ Resistor
Pin 2	Raspberry Pi GPIO Pin 4 (SS)
Pin 3	Raspberry Pi MOSI Pin
Pin 4	Raspberry Pi MISO Pin
Pin 5	Raspberry Pi SCK Pin

Table 1: Arduino Connection Table

was at this point that I realized that the Lightblue Bean Arduino doesn't actually have a serial chip. I then did some research and chose SPI as my mode of communication since it is essentially the only other form of communication that my Lightblue Bean has besides bluetooth.

I encountered many issues when working with SPI. At first I wasn't using the Slave Select pin, so my Arduino wasn't listening for a message properly, but because then the Slave Select pin was always low it did receive some messages, but not the whole message. Next I realized that my Raspberry Pi had too high of a clock rate for the two devices to communicate with properly. Since there were multiple simultaneous issues it was quite difficult to troubleshoot. I was able to use an extremely useful tutorial by a member of the Arduino forum named Nick Gammon. [1] With the information he provided I was able to send messages via SPI between the Raspberry Pi and the Lightblue Bean.

From this point I setup the hardware buttons on a completely separate Arduino sketch. It wasn't too difficult to have the hardware buttons functioning as I wanted, dimming and lighting the led as specified above in my documentation.

I then returned to the Android app and SPI / Bluetooth sketch, and I modified them to deliver and interpret commands respectively.

When parsing the commands I encountered a serious issue that was quite difficult to trouble shoot. My linked-list of past commands didn't seem to be working and I couldn't figure out why my past commands weren't being saved. I originally assumed my linked-list code was the issue, but couldn't find any problems with it. Eventually I realized that the command malloc() doesn't actually return a chunk of memory that is guaranteed to live, so when I was calling malloc() the old commands were being lost from my linked-list. Once I figured this out, I was able to use the command "new" to allocate memory which solved all of the issues that I was experiencing with my linked-list of past commands.

After solving the linked-list issue, I was able to use the now parsed commands to request a change in brightness based on the command received. I implemented code in the main loop to check for a requested software change in the brightness of the LED which then adjusted

it accordingly with each loop pass.

Finally I combined my two sketches to include both the hardware key press code and the software command code, making sure that the hardware key presses cleared any requests by the software for light changes.

I then performed these tests to confirm that my programs were functioning as desired:

Only using UDP Packets - Controlling light brightness with Bluetooth disabled.

Only using Bluetooth Packets - Controlling light brightness with UDP Server disabled.

Using both Bluetooth and UDP Packets - Controlling light brightness with both services. For this method I was able to see that most packets were received and processed via bluetooth first. Occasionally a packet would arrive via UDP first, but since the network interface is WiFi it's unsurprising that this transmission is slightly slower than the Bluetooth transmission.

Finally I tested all of these scenarios while pressing the hardware buttons. As desired the hardware key presses universally overrode any software requests that I made with the app.

## Conclusion

As planned the entire remote light control application functioned as desired. The commands were received rapidly and extremely reliable from the Android app, and the hardware buttons effectively override any software commands.

Despite the success of this project there are many improvements that can be made.

In order to ensure that all commands are received by the server, there are two things that can be done. The first is for the server to respond after a lull of one or two seconds with a confirmation of the last command received. This would allow the mobile application to ensure that the last command sent was received. The second and probably more reliable method would be to include a third (or replace the second) form of interface and create a TCP connection with the server guaranteeing that any packet sent would arrive at its destination.

Another feature that needs to be added is the current status of the system. This would be a packet sent by the Android app requesting a packet back with the current light statuses. This would allow the user to see the actual status of the system at all times. The current system doesn't show system changes in the app when the hardware buttons are pressed or when a different client mobile app changes the system.

The app could also use some improvements on forming the Bluetooth connection. Instead of maintaining a constant connection, the app needs to have a way of sharing the bluetooth connection because currently only one device can be connected to each Lightblue Bean at a time. This could mean working with the Lightblue Bean to allow multiple connections or it could also be implemented by having the Android app disconnect after an idle period has passed.

Another feature to add to the app is the ability to pair with a Lightblue Bean and give it a unique ID that can be seen in the app. This would allow the user to easily name different lights / light switches. The ability to do this would also require a complete app user interface design since the app needs to be simple and easy to use but robust enough to handle many different lights. Currently the app only supports one device at a time.

Finally the eventual goal is to develop a mesh network with the Lightblue bean chips. This will take an extensive amount of coding, but would result in a network that should be able to very rapidly propagate any command, whether or not WiFi is present in the system or not.

There are some limitations to the project in its current state, but the hardware and software proof of concept worked as desired. It is both an intuitive and robust system that transmits commands rapidly and still allows the user to easily control the system without the need for a smartphone app.

## Appendix A: Server Code

Main Server Code:

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <string.h>
4  #include <time.h>
5  #include <stdlib.h>
6  #include <time.h>
7  #include <inttypes.h>
8
9  #include <sys/types.h>
10 #include <sys/socket.h>
11 #include <sys/ioctl.h>
12 #include <netinet/in.h>
13 #include <net/if.h>
14 #include <arpa/inet.h>
15
16 #include <wiringPi.h>
17 #include <wiringPiSPI.h>
18
19 //Max message size
20 #define MSG_SIZE 40
21
22 int currentVote = 0;
23 int currentIp = 0;
24 int currentMasterIp = 0;
25 char* address;
26
27 void sendSPIMessage(char[MSG_SIZE]);
28
29 //Initialization to get current address
30 void getAddress(){
31     int sckt;
32     struct ifreq ifr;
33
34     //Create a socket so we can pull our IP information out
35     sckt = socket(AF_INET, SOCK_DGRAM, 0);
36     if (sckt < 0){
37         printf("Error opening socket");
38         exit(0);
39     }
40
41     ifr.ifr_addr.sa_family = AF_INET;
42
43     //Using the default interface eth0
44     strncpy(ifr.ifr_name, "wlan0", IFNAMSIZ-1);
45
```

```
46     ioctl(sckt, SIOCGIFADDR, &ifr);
47
48     //Copy the current address into a character array.
49     address = inet_ntoa(((struct sockaddr_in *)&ifr.ifr_addr)->sin_addr);
50
51     //Split up address by periods so that we can save the last set of numbers as our current ip
        integer for comparison purposes.
52     char ipBuffer[20];
53     bzero(ipBuffer, 20);
54     strcpy(ipBuffer, address);
55
56     //printf("ipBuffer: %s\n", ipBuffer);
57
58     char** ip = NULL;
59     char* block = strtok (ipBuffer, ".");
60     int n_periods = 0, i;
61
62     while (block) {
63         ip = realloc (ip, sizeof (char*) * ++n_periods);
64
65         ip[n_periods-1] = block;
66
67         block = strtok (NULL, ".");
68     }
69
70     ip = realloc (ip, sizeof (char*) * (n_periods+1));
71     ip[n_periods] = 0;
72
73     //Check to make sure valid input
74     if (n_periods != 4){
75         printf("Invalid ip address");
76     }
77     else {
78         currentIp = atoi(ip[3]);
79     }
80
81     printf("%s\n", address);
82
83     close(sckt);
84
85     // -----
86 }
87
88 int main(int argc, char *argv[]){
89     int port = 0;
90
91     // Use arg 1 as current port if available, otherwise default to 5000
92     if (argc < 2){
93         port = 5000;
```

```
94     }
95     else {
96         port = atoi(argv[1]);
97     }
98
99     srand(time(NULL));
100
101     //Initialization to get current address
102     getAddress();
103
104     //Wiring Pi Setup
105     wiringPiSetup();
106     wiringPiSPISetup(1, 614400);
107     pinMode(7, OUTPUT);
108     digitalWrite(7, 1);
109
110     //Setup sockets for sending and receiving messages
111     int serverLength;
112     int boolVal = 1;
113     socklen_t fromlen;
114     struct sockaddr_in server;
115     struct sockaddr_in anybody;
116
117     serverLength = sizeof(server);
118     bzero(&server, serverLength);
119     server.sin_family = AF_INET;
120     server.sin_addr.s_addr = INADDR_ANY;
121     server.sin_port = htons(port);
122
123     int sckt = socket(AF_INET, SOCK_DGRAM, 0);
124     if (sckt < 0){
125         printf("Error opening socket\n");
126         exit(0);
127     }
128
129     if (bind(sckt, (struct sockaddr*)&server, serverLength) < 0){
130         printf("Error binding to socket\n");
131         exit(0);
132     }
133
134     if (setsockopt(sckt, SOL_SOCKET, SO_BROADCAST, &boolVal, sizeof(boolVal)) < 0){
135         printf("error setting socket options\n");
136         exit(-1);
137     }
138
139     anybody.sin_family = AF_INET;        // symbol constant for Internet domain
140     anybody.sin_port = htons(port);      // port field
141     anybody.sin_addr.s_addr = inet_addr("10.3.52.255"); // broadcast address
142
```



```
143     int inputLength = sizeof(struct sockaddr_in);
144     int n;
145     int value;
146     char buffer[MSG_SIZE];
147
148     while (1)
149     {
150         // Clean buffer with bzero since not all messages are the same length.
151         bzero(buffer, MSG_SIZE);
152
153         // receive message from socket
154         n = recvfrom(sckt, buffer, MSG_SIZE, 0, (struct sockaddr *)&server, &inputLength);
155         if (n < 0)
156             error("recvfrom");
157
158         //Log received message
159         printf("Received a message. It says: %s\n", buffer);
160
161         //Indiscriminately pass message on to Arduino
162         sendSPIMessage(buffer);
163     }
164
165     return 0;
166 }
167
168 //Function to send message to arduino via SPI
169 void sendSPIMessage(char message[MSG_SIZE]){
170     uint8_t a[] = {'0'};
171     uint8_t b[] = {'0'};
172
173     //First two garbage bits. For some reason the Arduino SPI consumes these before it starts
174     //receiving the message
175     wiringPiSPIDataRW(1,a,1);
176     wiringPiSPIDataRW(1,b,1);
177
178     //Save input into buffer
179     char input[10];
180     snprintf(input,sizeof(input),"%s",message);
181
182     //Write low to SS to specify to the Arduino an SPI message for it is arriving
183     digitalWrite(7,0);
184
185     //Send message via SPI
186     wiringPiSPIDataRW(1,input,strlen(input));
187
188     //Write high to SS to specify to the Arduino the spi message has finished sending
189     digitalWrite(7,1);
190
191     printf("Sent data: %s\n",input);
```

191 }

Main Server Script:

```
1 #!/bin/bash
2 gpio load spi
3 ./udpServer &
```

Makefile:

```
1 file=udpServer.c
2 output=udpServer
3 compiler=gcc
4
5 all: ${OBSJ}
6     ${compiler} -lwiringPi ${file} -o ${output}
7     ${compiler}  udpServer.c -o udpServer
8
9 %.o: %.c
10     ${compiler} -c -o $@ $<
11
12 clean:
13     rm ${output} ${OBSJ}
```

## Appendix B: Arduino Code

```
1 #include <SPI.h>
2
3 /*
4     Network / bluetooth communication setup
5 */
6 const boolean LIGHT_DEBUG = false;
7 const int bufferLength = 100;
8 const int MAX_COMMANDS = 100;
9
10 char spiBuffer[bufferLength];
11 char serialBuffer[bufferLength];
12 volatile byte spiPosition;
13 volatile boolean process_it = false;
14
15 int redLedStatus = 0;
16 int greenLedStatus = 0;
17 int blueLedStatus = 0;
18 int externalA0LedStatus=0;
19 int externalA1LedStatus=0;
20
21 long softwareCommandStart = 0;
22 int softwareGoalValue = -1;
23 long softwareDimmingSpeedDivisor = 3;
24
25 //Command typedef, currently device is unused
26 typedef struct commandStruct{
27     struct commandStruct* nextCommand = NULL;
28     int id;
29     int device;
30     int value;
31 };
32
33 struct commandStruct* headCommand = NULL;
34
35 /*
36     Physical Buttons setup
37 */
38 const int button1Pin = 0;
39 const int button2Pin = 1;
40
41 int dimmingDirection = 0;
42
43 int ledBrightness = 0;
44 int currentBrightness = 0;
45 int brightnessChange = 0;
46
47 int button1State;
```

```
48 int button2State;
49 int lastButton1State = LOW;
50 int lastButton2State = LOW;
51
52 long lastPressTime = 0;
53
54 long debounceDelay = 50;
55 long dimmingDelay = 250;
56 long dimmingSpeedDivisor = 7;
57
58 /*
59     Function declarations
60 */
61 void setLedBrightness(int);
62 void toggleRedLed();
63 void toggleGreenLed();
64 void toggleBlueLed();
65 void toggleAOLed();
66 void toggleA1Led();
67 void handleMessage(char[], bool);
68 void handleCommand(int, int, int, bool);
69 bool isPastCommand(int);
70 void saveCommand(struct commandStruct*);
71 void performAction(int, int, bool);
72
73
74 // Falling edge Slave Select interrupt, start of transaction, no command yet.
75 void ss_falling (){
76
77 }
78
79 void setup (void){
80     /*
81         Network setup
82     */
83     Serial.begin (57600);
84
85     //Enable bean advertising
86     Bean.enableAdvertising(true);
87
88     pinMode(A0, OUTPUT);
89     pinMode(A1, OUTPUT);
90
91     // have to send on master in, *slave out*
92     pinMode(MISO, OUTPUT);
93
94     // turn on SPI in slave mode
95     SPCR |= _BV(SPE);
96
```

```
97 // turn on interrupts
98 SPCR |= _BV(SPIE);
99
100 // get ready for an interrupt
101 spiPosition = 0;
102 process_it = false;
103
104 // interrupt for SS falling edge
105 attachInterrupt (0, ss_falling, FALLING);
106
107 /*
108    Physical Buttons Setup
109 */
110 pinMode(button1Pin, INPUT);
111 pinMode(button2Pin, INPUT);
112
113 //Initialize led brightness
114 Bean.setLedRed(ledBrightness);
115 }
116
117
118 // SPI interrupt routine
119 ISR (SPI_STC_vect){
120     // grab byte from SPI Data Register
121     byte c = SPDR;
122
123     // add to buffer if room
124     if (spiPosition < sizeof spiBuffer)
125     {
126         spiBuffer[spiPosition++] = c;
127
128         // newline character is termination character
129         if (c == '\n')
130             process_it = true;
131     }
132 }
133
134
135 // main loop - wait for flag set in interrupt routine
136 void loop (void){
137     //Bluetooth message
138     if(Serial.available() > 0){
139         /*
140         if (LIGHT_DEBUG){
141             toggleA1Led();
142         }
143         */
144
145         Serial.readBytesUntil('\n',serialBuffer,bufferLength);
```

```
146     //Handle bluetooth message
147     handleMessage(serialBuffer, false);
148     memset(serialBuffer, 0, sizeof(serialBuffer));
149 }
150
151 //SPI Message
152 if (process_it){
153     spiBuffer[spiPosition] = 0;
154     //Handle SPI message
155     handleMessage(spiBuffer, true);
156
157     memset(spiBuffer, 0, sizeof(spiBuffer));
158     spiPosition = 0;
159     process_it = false;
160 }
161
162 //Read button states
163 int button1Reading = digitalRead(button1Pin);
164 int button2Reading = digitalRead(button2Pin);
165
166 //Assuming button press is high and not already going the other direction
167 if (button1Reading == HIGH && (dimmingDirection == 0 || dimmingDirection == 1)){
168     //Record button press if it changes from low to high
169     if (button1Reading != lastButton1State) {
170         lastPressTime = millis();
171     }
172
173     long delay = millis() - lastPressTime;
174     if(delay > debounceDelay){
175         dimmingDirection=1;
176     }
177     if(delay > dimmingDelay){
178         //calculate brightness adjustment based on current brightness and direction
179         long timeAfter = delay - dimmingDelay;
180         brightnessChange = (int) (timeAfter / dimmingSpeedDivisor);
181         currentBrightness = min(255,ledBrightness + brightnessChange);
182
183         setLedBrightness(currentBrightness);
184     }
185 }
186 else if (button2Reading == HIGH && (dimmingDirection == 0 || dimmingDirection == -1)){
187     //Record button press if it changes from low to high
188     if (button2Reading != lastButton2State) {
189         lastPressTime = millis();
190     }
191
192     long delay = millis() - lastPressTime;
193     if(delay > debounceDelay){
194         dimmingDirection=-1;
```

```
195     }
196     if(delay > dimmingDelay){
197         //calculate brightness adjustment based on current brightness and direction
198         long timeAfter = delay - dimmingDelay;
199         brightnessChange = (int) -(timeAfter / dimmingSpeedDivisor);
200         currentBrightness = max(0,ledBrightness + brightnessChange);
201
202         setLedBrightness(currentBrightness);
203     }
204 }
205 else {
206     //Finished dimming, brightening, or with button press
207     ledBrightness = min(255,ledBrightness+brightnessChange);
208     //If a button state changed, we must perform an action
209     if (button1Reading != lastButton1State || button2Reading != lastButton2State){
210         long delay = millis() - lastPressTime;
211         //If shorter than dimming delay, turn on or off light respectively based on dimming
            direction
212         if (delay > debounceDelay && delay < dimmingDelay){
213             if (dimmingDirection == 1){
214                 ledBrightness = 255;
215             }
216             else if (dimmingDirection == -1){
217                 ledBrightness=0;
218             }
219             setLedBrightness(ledBrightness);
220             brightnessChange = 0;
221             dimmingDirection = 0;
222         }
223         //Else reset values and record currentBrightness as ledBrightness since the dimming is
            complete
224         else if(delay > debounceDelay){
225             ledBrightness = currentBrightness;
226             brightnessChange = 0;
227             dimmingDirection = 0;
228         }
229     }
230 }
231 }
232
233 //Save current state of buttons for next loop
234 lastButton1State = button1Reading;
235 lastButton2State = button2Reading;
236
237 //Software dimming as long as hardware isn't currently dimming
238 if (dimmingDirection == 0 && softwareGoalValue != -1){
239     //Software goal set and no hardware request present. Dim.
240     long currentTime = millis();
241     long timeAfter = currentTime - softwareCommandStart;
```

```
242     int softwareBrightnessChange = (int) (timeAfter / softwareDimmingSpeedDivisor);
243     if (ledBrightness == softwareGoalValue){
244         softwareGoalValue = -1;
245     }
246     else if (softwareBrightnessChange >= 1){
247         //Move software command start time forward
248         softwareCommandStart=currentTime;
249         //Increasing brightness
250         if (ledBrightness < softwareGoalValue){
251             ledBrightness = min(255,ledBrightness+softwareBrightnessChange);
252             if (ledBrightness >= softwareGoalValue){
253                 softwareGoalValue = -1;
254             }
255             setLedBrightness(ledBrightness);
256         }
257         //Decreasing brightness
258         else if (ledBrightness > softwareGoalValue){
259             ledBrightness = max(0,ledBrightness-softwareBrightnessChange);
260             if (ledBrightness <= softwareGoalValue){
261                 softwareGoalValue = -1;
262             }
263             setLedBrightness(ledBrightness);
264         }
265     }
266 }
267 //Hardware currently dimming, ignore.
268 else{
269     //Discard all software goals that have been set once a hardware press has occurred.
270     softwareGoalValue = -1;
271 }
272 }
273
274 void setLedBrightness(int brightness){
275     Bean.setLedRed(brightness);
276     return;
277 }
278
279 //Toggle red led
280 void toggleRedLed(){
281     if (redLedStatus == 0){
282         Bean.setLedRed(255);
283         redLedStatus = 1;
284     }
285     else{
286         Bean.setLedRed(0);
287         redLedStatus = 0;
288     }
289     return;
290 }
```



```
291
292 //Toggle green led
293 void toggleGreenLed(){
294     if (greenLedStatus == 0){
295         Bean.setLedGreen(255);
296         greenLedStatus = 1;
297     }
298     else{
299         Bean.setLedGreen(0);
300         greenLedStatus = 0;
301     }
302     return;
303 }
304
305 //Toggle blue led
306 void toggleBlueLed(){
307     if (blueLedStatus == 0){
308         Bean.setLedBlue(255);
309         blueLedStatus = 1;
310     }
311     else{
312         Bean.setLedBlue(0);
313         blueLedStatus = 0;
314     }
315     return;
316 }
317
318 //Toggle led hooked up to line A0
319 void toggleAOLed(){
320     if (externalAOLedStatus == 0){
321         analogWrite(A0,255);
322         externalAOLedStatus = 1;
323     }
324     else{
325         analogWrite(A0,0);
326         externalAOLedStatus = 0;
327     }
328     return;
329 }
330
331 //Toggle led hooked up to line A1
332 void toggleA1Led(){
333     if (externalAOLedStatus == 0){
334         analogWrite(A1,255);
335         externalAOLedStatus = 1;
336     }
337     else{
338         analogWrite(A1,0);
339         externalAOLedStatus = 0;
```

```

340     }
341     return;
342 }
343
344 void handleMessage(char buffer[100], bool fromSPI){
345     //Split command received based on spaces
346     char** splitArray = NULL;
347     char* delimiter = strtok(buffer, " ");
348     int nSpaces = 0;
349     int i=0;
350
351     while (delimiter) {
352         splitArray = (char**) realloc (splitArray, sizeof (char*) * ++nSpaces);
353
354         if (splitArray == NULL)
355             return;
356
357         splitArray[nSpaces-1] = delimiter;
358
359         delimiter = strtok (NULL, " ");
360     }
361
362     splitArray = (char**) realloc (splitArray, sizeof (char*) * (nSpaces+1));
363     splitArray[nSpaces] = "\n";
364
365     //If the number of objects in the command was 2, valid command.
366     if (nSpaces == 2){
367         int id = atoi(splitArray[0]);
368         int device = 0;
369         int value = atoi(splitArray[1]);
370         //Valid, handle command
371         handleCommand(id, device, value, fromSPI);
372     }
373
374     /*
375     if (LIGHT_DEBUG){
376         if (nSpaces == 2){
377             toggleRedLed();
378         }
379     }
380     */
381
382     //Free array after used
383     free(splitArray);
384     return;
385 }
386
387 void handleCommand(int id, int device, int value, bool fromSPI){
388     //Check to make sure command hasn't already been received

```

```
389     if (!isPastCommand(id)){
390         //Saving command with struct
391         struct commandStruct* command = new struct commandStruct;
392         command->id = id;
393         command->device = device;
394         command->value = value;
395
396         //Since new command, save in linked list of past commands.
397         saveCommand(command);
398
399         //Perform the action specified by the command
400         performAction(device, value, fromSPI);
401     }
402
403     return;
404 }
405
406 //Function to loop through linked list and check to see if the past command has been received
407 //before based on id
408 //Returns true if it is a past command, and false if it is a new command
409 bool isPastCommand(int id){
410     struct commandStruct* currentCommand = headCommand;
411     if (currentCommand == NULL){
412         return false;
413     }
414     else {
415         while (currentCommand != NULL){
416             //current command not null, bump to next command or return true if matching id
417             if (id == currentCommand->id){
418                 return true;
419             }
420             else {
421                 currentCommand = currentCommand->nextCommand;
422             }
423         }
424         //Command not found, return false
425         return false;
426     }
427 }
428 //Function to save command to linked list of commands
429 void saveCommand(struct commandStruct* command){
430     struct commandStruct* currentCommand = headCommand;
431     int numberOfCommands = 0;
432
433     //Save as head if no head
434     if (currentCommand == NULL){
435         headCommand = command;
436     }
```

```
437     else {
438         numberOfCommands++;
439         //Bump to end of linked list
440         while (currentCommand->nextCommand != NULL){
441             numberOfCommands++;
442             currentCommand = currentCommand->nextCommand;
443         }
444         //Set nextCommand to the new command
445         currentCommand->nextCommand = command;
446
447         //Maintain a linked list with a maximum size of commands
448         if (numberOfCommands > MAX_COMMANDS){
449             if (LIGHT_DEBUG){
450                 toggleGreenLed();
451             }
452             struct commandStruct* commandToFree = headCommand;
453             headCommand = headCommand->nextCommand;
454             //Free old head since our linked list has reached our maximum size.
455             free(commandToFree);
456         }
457     }
458
459     return;
460 }
461
462 //Function for performing an action on a device with a given requested brightness level.
463 void performAction(int device, int value, bool fromSPI){
464     //Toggle correct LED to indicate what interface the command was received with
465     if (fromSPI){
466         toggleA0Led();
467     }
468     else {
469         toggleA1Led();
470     }
471
472     if (LIGHT_DEBUG){
473         toggleRedLed();
474     }
475
476     //Software command, save current time
477     softwareCommandStart = millis();
478     //Set softwareGoalValue, keep within 255 and 0
479     softwareGoalValue = value;
480     softwareGoalValue = min(255, softwareGoalValue);
481     softwareGoalValue = max(0, softwareGoalValue);
482     return;
483 }
```

## Appendix C: Android Code

build.gradle:

```
1 apply plugin: 'com.android.application'
2
3 android {
4     compileSdkVersion 21
5     buildToolsVersion "21.1.2"
6
7     defaultConfig {
8         applicationId "net.quarkworks.bluetoothlightcontrol"
9         minSdkVersion 19
10        targetSdkVersion 21
11        versionCode 1
12        versionName "1.0"
13    }
14    buildTypes {
15        release {
16            minifyEnabled false
17            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18        }
19    }
20 }
21
22 dependencies {
23     compile fileTree(dir: 'libs', include: ['*.jar'])
24     compile 'com.android.support:appcompat-v7:21.0.3'
25     compile 'nl.littlerobots.bean:beanlib:0.9.2'
26 }
```

AndroidManifest.xml:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="net.quarkworks.bluetoothlightcontrol" >
4
5     <uses-permission android:name="android.permission.INTERNET" />
6     <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
7
8     <application
9         android:allowBackup="true"
10        android:icon="@drawable/ic_launcher"
11        android:label="@string/app_name"
12        android:theme="@style/AppTheme" >
13         <activity
14             android:name=".ParentActivity"
15             android:label="@string/app_name" >
```

```
16         <intent-filter>
17             <action android:name="android.intent.action.MAIN" />
18
19             <category android:name="android.intent.category.LAUNCHER" />
20         </intent-filter>
21     </activity>
22 </application>
23
24 </manifest>
```

ParentActivity.java:

```
1 package net.quarkworks.bluetoothlightcontrol;
2
3 import android.support.v7.app.ActionBarActivity;
4 import android.support.v7.app.ActionBar;
5 import android.support.v4.app.Fragment;
6 import android.os.Bundle;
7 import android.view.LayoutInflater;
8 import android.view.Menu;
9 import android.view.MenuItem;
10 import android.view.View;
11 import android.view.ViewGroup;
12 import android.os.Build;
13
14 import nl.littlerobots.bean.Bean;
15 import nl.littlerobots.bean.BeanDiscoveryListener;
16 import nl.littlerobots.bean.BeanManager;
17
18
19 public class ParentActivity extends ActionBarActivity {
20
21     public BeanDiscoveryListener listener;
22
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.parent_activity);
27
28         final ParentFragment parentFragment = new ParentFragment();
29
30         if (savedInstanceState == null) {
31             getSupportFragmentManager().beginTransaction()
32                 .add(R.id.container, parentFragment)
33                 .commit();
34         }
35
36         // create a listener
37         listener = new BeanDiscoveryListener() {
```

```
38         @Override
39         public void onBeanDiscovered(Beans bean) {
40             parentFragment.onBeanDiscovered(bean);
41         }
42
43         @Override
44         public void onDiscoveryComplete() {
45             parentFragment.onDiscoveryComplete();
46         }
47     };
48
49     //Start searching for bluetooth devices immediately
50     BeanManager.getInstance().startDiscovery(listener);
51
52
53 }
54
55
56
57 @Override
58 public boolean onCreateOptionsMenu(Menu menu) {
59     // Inflate the menu; this adds items to the action bar if it is present.
60     getMenuInflater().inflate(R.menu.menu_parent, menu);
61     return true;
62 }
63
64 @Override
65 public boolean onOptionsItemSelected(MenuItem item) {
66     // Handle action bar item clicks here. The action bar will
67     // automatically handle clicks on the Home/Up button, so long
68     // as you specify a parent activity in AndroidManifest.xml.
69     int id = item.getItemId();
70
71     //noinspection SimplifiableIfStatement
72     if (id == R.id.action_settings) {
73         return true;
74     }
75
76     return super.onOptionsItemSelected(item);
77 }
78
79
80 }
```

ParentFragment.java:

```
1 package net.quarkworks.bluetoothlightcontrol;
2
3 import android.os.Bundle;
```

```
4
5 import android.support.v4.app.Fragment;
6 import android.util.Log;
7 import android.view.LayoutInflater;
8 import android.view.View;
9 import android.view.ViewGroup;
10 import android.widget.Button;
11 import android.widget.SeekBar;
12
13 import java.io.UnsupportedEncodingException;
14 import java.text.SimpleDateFormat;
15 import java.util.ArrayList;
16 import java.util.Date;
17
18 import nl.littlerobots.bean.Bean;
19 import nl.littlerobots.bean.BeanListener;
20 import nl.littlerobots.bean.BeanManager;
21
22 /**
23  * Created by benjamin on 5/3/15.
24  */
25 public class ParentFragment extends Fragment {
26     private static final String TAG = ParentFragment.class.getSimpleName();
27
28     /*
29      Constants
30      */
31     //Port to listen on
32     public int COM_PORT = 5000;
33     //Threshold for seekbar to send a packet (send every 10 increments in this case)
34     public int SEEKBAR_THRESHOLD = 10;
35
36     /*
37      References
38      */
39     private Button toggleButton;
40     private Button testButton;
41     private SeekBar seekBar;
42     private ArrayList<Bean> discoveredBeans = new ArrayList<Bean>();
43     private ArrayList<Bean> connectedBeans = new ArrayList<Bean>();
44     private Button disconnectButton;
45     private Button connectButton;
46
47     /*
48      Data
49      */
50
51     private int messageId = 0;
52     private int lastSeekBarProgress = 0;
```



```
53
54
55     public ParentFragment() {
56     }
57
58
59     @Override
60     public void onStart() {
61         super.onStart();
62     }
63
64     @Override
65     public View onCreateView(LayoutInflater inflater, ViewGroup container,
66                             Bundle savedInstanceState) {
67         View view = inflater.inflate(R.layout.parent_fragment, container, false);
68
69
70         //Grab views
71         toggleButton = (Button) view.findViewById(R.id.toggle_button);
72         disconnectButton = (Button) view.findViewById(R.id.disconnect_button);
73         connectButton = (Button) view.findViewById(R.id.connect_button);
74         seekBar = (SeekBar) view.findViewById(R.id.brightness_seekbar);
75
76         //Set seekbar increments to 1 with a max value of 255
77         seekBar.setKeyProgressIncrement(1);
78         seekBar.setMax(255);
79
80         seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
81             @Override
82             public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
83                 //Assuming user moved seekbar, and greater or less than threshold, send command
84                 if (progress - lastSeekBarProgress >= SEEKBAR_THRESHOLD || progress -
85                     lastSeekBarProgress <= -SEEKBAR_THRESHOLD && fromUser){
86                     lastSeekBarProgress = progress;
87                     sendBrightnessCommand(progress);
88                 }
89             }
90
91             @Override
92             public void onStartTrackingTouch(SeekBar seekBar) {
93                 //On press track starting position for use with threshold
94                 lastSeekBarProgress = seekBar.getProgress();
95             }
96
97             @Override
98             public void onStopTrackingTouch(SeekBar seekBar) {
99                 //Send brightness command on release
100                 Log.d(TAG, "Seekbar Progress: " + seekBar.getProgress());
101                 sendBrightnessCommand(seekBar.getProgress());
102             }
103         });
104     }
105 }
```

```
101     }
102     });
103
104     //Toggle light on click listener
105     toggleButton.setOnClickListener(new View.OnClickListener() {
106         @Override
107         public void onClick(View v) {
108             //Move progress bar based on current position to the opposite side of current
109             //position
110             if (seekBar.getProgress() > 127){
111                 seekBar.setProgress(0);
112             }
113             else {
114                 seekBar.setProgress(255);
115             }
116
117             sendBrightnessCommand(seekBar.getProgress());
118         }
119     });
120
121     //Disconnect from bluetooth button listener
122     disconnectButton.setOnClickListener(new View.OnClickListener() {
123         @Override
124         public void onClick(View v) {
125             for (final Bean bean:connectedBeans){
126                 bean.disconnect();
127                 connectedBeans.remove(bean);
128             }
129         }
130     });
131
132     //Connect to bluetooth button listener
133     connectButton.setOnClickListener(new View.OnClickListener() {
134         @Override
135         public void onClick(View v) {
136             BeanManager.getInstance().startDiscovery(((ParentActivity) getActivity()).listener)
137             ;
138         }
139     });
140
141
142     return view;
143 }
144
145
146 //Send a command to the server with a unique id. Also included in this is the brightness level
147 public void sendBrightnessCommand(int brightnessLevel){
```

```
148     String message = messageId%999+1 + " " + brightnessLevel + "\n";
149     messageId++;
150
151     Log.d(TAG, "Sending Message: " + message);
152     //Create a new broadcast object
153     UDPBroadcast broadcast = new UDPBroadcast(getActivity(), COM_PORT);
154     //Send the message via UDP
155     broadcast.sendMessage(message);
156
157     //Log.d(TAG, "Sent message: " + message);
158
159     Log.d(TAG, "number of connected beans: " + connectedBeans.size());
160     for (final Bean bean:connectedBeans){
161         //For all beans, if connected send a bluetooth message to the bean.
162         if (bean.isConnected()) {
163             Log.d(TAG, "Sending BLE Message: " + message);
164             byte[] messageBytes = new byte[0];
165             try {
166                 messageBytes = message.getBytes("UTF-8");
167             } catch (UnsupportedEncodingException e) {
168                 e.printStackTrace();
169             }
170             bean.sendSerialMessage(messageBytes);
171         }
172     }
173 }
174
175 //On bean discovered listener. Log statements describe each function
176 public void onBeanDiscovered(final Bean bean){
177     Log.d(TAG, "Bean discovered: " + bean.getDevice());
178     discoveredBeans.add(bean);
179     BeanListener beanListener = new BeanListener() {
180         @Override
181         public void onConnected() {
182             connectedBeans.add(bean);
183             Log.d(TAG, "Successfully connected to device: " + bean.getDevice());
184             //bean.disconnect();
185         }
186
187         @Override
188         public void onConnectionFailed() {
189             Log.d(TAG, "Connection failed to device: " + bean.getDevice());
190         }
191
192         @Override
193         public void onDisconnected() {
194             Log.d(TAG, "Disconnected from device: " + bean.getDevice());
195             //connectedBeans.remove(connectedBeans.indexOf(bean));
196         }
197     }
198 }
```

```

197
198         @Override
199         public void onSerialMessageReceived(byte[] bytes) {
200             String string = "";
201             try {
202                 string = new String(bytes, "UTF-8");
203             } catch (UnsupportedEncodingException e) {
204                 e.printStackTrace();
205             }
206
207             Log.d(TAG, "Serial message received: " + string);
208         }
209
210         @Override
211         public void onScratchValueChanged(int i, byte[] bytes) {
212             Log.d(TAG, "Scratch value changed, i: " + i + ", bytes: " + bytes.toString());
213         }
214     };
215     bean.connect(getActivity(), beanListener);
216 }
217
218 public void onDiscoveryComplete(){
219
220     Log.d(TAG, "onDiscoveryComplete");
221 }
222
223 //Unique id, unused currently
224 public int getUniqueInt(){
225     SimpleDateFormat df = new SimpleDateFormat("DDDdhmmss");
226     Date date = new Date();
227
228     String dateString = "1" + df.format(date);
229     int uniqueInt = Integer.parseInt(dateString);
230     Log.d(TAG, "UniqueInt: " + uniqueInt);
231
232     return uniqueInt;
233 }
234 }

```

UDPBroadcast.java:

```

1 package net.quarkworks.bluetoothlightcontrol;
2
3 import java.io.IOException;
4 import java.io.UnsupportedEncodingException;
5 import java.net.DatagramPacket;
6 import java.net.DatagramSocket;
7 import java.net.InetAddress;
8 import java.net.SocketException;

```

```
9  import java.net.UnknownHostException;
10
11  import android.content.Context;
12  import android.net.DhcpInfo;
13  import android.net.wifi.WifiManager;
14  import android.os.AsyncTask;
15  import android.util.Log;
16
17  /**
18   * Created by benjamin on 5/3/15.
19   * Class for sending a broadcast udp packet
20   */
21  public class UDPBroadcast extends AsyncTask<byte[], byte[], Void> {
22      private static final String TAG = UDPBroadcast.class.getSimpleName();
23
24      /*
25       * References
26       */
27      private Context context;
28      private DatagramSocket socket;
29
30      /*
31       * Data
32       */
33      private int port;
34
35      UDPBroadcast(Context context, int port) {
36          this.context=context;
37          this.port=port;
38      }
39
40      //Send message function
41      public void sendMessage(String text){
42          byte[] dataArray = text.getBytes();
43          this.execute(dataArray);
44      }
45
46      protected Void doInBackground(byte[] ... dataArray) {
47          try {
48              socket = new DatagramSocket();
49              Log.d(TAG, "doInBackground: Socket created, maximum Buffer size:" + socket.
                    getSendBufferSize() + " bytes");
50          }
51          catch (SocketException e) {
52              e.printStackTrace();
53              Log.d(TAG, "broken socket");
54          }
55
56          byte[] data = dataArray[0];
```

```
57     Log.d(TAG, "Data size: " + data.length);
58
59     try {
60         // Fragment the data into small chunks and send them sequentially
61         int num_packets = (int) Math.ceil(data.length / 1024.0);
62         Log.d(TAG, "Sending " + num_packets + " packets");
63         byte[] tmp = new byte[1024];
64
65         for (int i = 0; i < num_packets; i++) {
66             // The last packet might be smaller
67             if (i == num_packets - 1)
68                 System.arraycopy(data, i * 1024, tmp, 0, (data.length - i * 1024) % 1024);
69             else
70                 System.arraycopy(data, i * 1024, tmp, 0, 1024);
71
72             //Send packet via with broadcast address
73             DatagramPacket packet = new DatagramPacket(tmp, tmp.length, getBroadcastAddress(
74                 context), port);
75             socket.send(packet);
76             Log.d(TAG, "Sent packet: " + i);
77         }
78         return null;
79     } catch (UnknownHostException e) {
80         Log.d(TAG, "Unknown Host");
81         e.printStackTrace();
82     }
83     catch (IOException e) {
84         Log.d(TAG, "IO Exception");
85         e.printStackTrace();
86     }
87     finally {
88         try {
89             socket.close();
90         } catch (Exception e) {
91             e.printStackTrace();
92         }
93         Log.d(TAG, "doInBackground: Finished");
94     }
95     return null;
96 }
97
98 protected void onProgressUpdate(byte[]... values) {
99     byte[] data = values[0];
100     String str;
101     try {
102         str = new String(data, "UTF8");
103         Log.d(TAG, str);
104         // appendToOutput(str);
```

```

105     } catch (UnsupportedEncodingException e) {
106         e.printStackTrace();
107     }
108 }
109
110 @Override
111 protected void onCancelled() {
112     Log.d(TAG, "Cancelled.");
113 }
114
115 protected void onPostExecute(Void v) {
116     Log.d(TAG, "onPostExecute");
117 }
118
119 //Get broadcast address (xxx.xxx.xxx.255)
120 InetAddress getBroadcastAddress(Context context) throws IOException {
121     WifiManager wifi = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
122     DhcpInfo dhcp = wifi.getDhcpInfo();
123
124     if (dhcp == null){
125         throw new IOException("Not connected to wifi");
126     }
127
128     int broadcast = (dhcp.ipAddress & dhcp.netmask) | ~dhcp.netmask;
129     byte[] quads = new byte[4];
130     for (int k = 0; k < 4; k++)
131         quads[k] = (byte) ((broadcast >> k * 8) & 0xFF);
132     return InetAddress.getByAddress(quads);
133 }
134
135 }

```

parent\_fragment.xml:

```

1 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
3     android:layout_height="match_parent" android:paddingLeft="@dimen/activity_horizontal_margin"
4     android:paddingRight="@dimen/activity_horizontal_margin"
5     android:paddingTop="@dimen/activity_vertical_margin"
6     android:paddingBottom="@dimen/activity_vertical_margin"
7     tools:context=".ParentActivity$PlaceholderFragment">
8
9     <Button
10         android:id="@+id/toggle_button"
11         android:layout_width="wrap_content"
12         android:layout_height="wrap_content"
13         android:layout_alignParentTop="true"
14         android:layout_centerHorizontal="true"
15         android:text="Toggle Light State"

```

```
16     />
17
18     <SeekBar
19         android:id="@+id/brightness_seekbar"
20         android:layout_width="wrap_content"
21         android:layout_height="wrap_content"
22         android:layout_alignParentStart="true"
23         android:layout_alignParentEnd="true"
24
25         android:layout_below="@+id/toggle_button"
26     />
27
28     <Button
29         android:id="@+id/disconnect_button"
30         android:layout_width="wrap_content"
31         android:layout_height="wrap_content"
32         android:layout_below="@+id/brightness_seekbar"
33         android:layout_centerHorizontal="true"
34
35         android:text="Disconnect"
36     />
37
38
39     <Button
40         android:id="@+id/connect_button"
41         android:layout_width="wrap_content"
42         android:layout_height="wrap_content"
43         android:layout_below="@+id/disconnect_button"
44         android:layout_centerHorizontal="true"
45
46         android:text="Connect"
47     />
48
49
50 </RelativeLayout>
```



## Appendix D: Readme

Readme:

```
1 README for Remote Light Control
2 Benjamin Temple
3 ECE 4220
4 SS2015 - University of Missouri
5
6 This code is divided into three parts, the Anrdoid app, the Arduino code, and the Raspberry PI
  server code.
7
8 Assuming all wiring is followed as shown in my documentation in my final report, this should
  function properly.
9
10
11 Android App:
12
13 In order to run the Android app, import the project into Android Studio 1.0.
14 It should compile and run without issues.
15
16 Arduino code:
17
18 The arduino code is specifically designed to be run on a Lightblue Bean designed by punchthrough.
19 The exact Arduino can be found here:
20
21 https://punchthrough.com/bean/
22
23 In conjunction with this found on punch through design's website they have a special loader which
  allows the Arduino code to be installed via bluetooth.
24 They have setup instructions which can be found here on how to setup the loader with the Arduino
  compiler:
25
26 https://punchthrough.com/bean/getting-started-osx/
27 https://punchthrough.com/bean/getting-started-windows/
28
29 Once the Lightblue Bean is setup and functioning with a demo project, merely load my remote.control
  .arduinoCode.ino file into the Arduino and the arduino code should function properly.
30
31 Server Code:
32
33 To use the server code, first a raspberry pi (arm processor) needs to be setup and running on the
  same network as the network the Android phone is connected to.
34 Once this is accomplished, the wiring pi library needs to be installed.
35 This can be found here:
36
37 http://wiringpi.com/download-and-install/
38
```

```
39 After wiringPi is installed, in my ./remote.control.serverCode/ directory compile the server code
    by running make:
40
41 cd ./remote.control.serverCode/
42 make
43
44 Next we need to load the SPI drivers, this can be accomplished with wiring pi's gpio utility:
45
46 gpio load spi
47
48 Finally run the server script as root:
49
50 sudo ./udpServer
51
52 A script can also be made to run this automatically on startup. Merely add the following line to "/
    etc/rc.local" somewhere before the exit.
53
54 sh /[path to remote.control.serverCode]/runServer.sh
```

# References

- [1] N. Gammon, *Gammon Forum : Electronics : Microprocessors : SPI - Serial Peripheral Interface - for Arduino*, Gammon.com.au, 2015. [Online]. Available: <http://www.gammon.com.au/forum/?id=10892>. [Accessed: 14- May- 2015].