# Not Exactly the Internet of Things for Outdoor Lighting
# Spring Term 2016 Report
# Final Stage

Oregon State University CS Senior Capstone Group 22
Malcolm Diller (dillerm@oregonstate.edu)
Sean Rettig (rettigs@oregonstate.edu)
Evan Steele (steelee@oregonstate.edu)
Client: Victor Hsu (hsuv@onid.orst.edu)

**Abstract**

"Not Exactly the Internet of Things for Outdoor Lighting" (or simply "PiLight") is a self-contained home lighting automation system designed to be powerful and highly customizable, yet inexpensive. Standard wallwart plug timers are cheap, but their usefulness is limited. High-tech Internet-connected automation systems are feature-rich, but are often very expensive, are not easily customizable, and can be rendered useless if the external service is ever shut down. PiLight solves this issue by not requiring any external dependencies to function normally. Furthermore, PiLight is built from commodity hardware and open source software, so the system is not only inexpensive, but also easy to extend and personalize, for the technically inclined. PiLight includes a sophisticated rule and scheduling system and has the potential to integrate with devices other than lights, such as sprinkler systems, garage door openers, and more.

Our associated GitHub repo is available at https://github.com/capstone22-Pilight/cs-senior-capstone

CONTENTS

# I. Introduction

"Not Exactly the Internet of Things for Outdoor Lighting" (or simply "PiLight") is a self-contained home lighting automation system designed to be powerful and highly customizable, yet inexpensive. Originally commissioned by Victor Hsu at Oregon State University for simple outdoor lighting automation, the project has expanded to include a sophisticated rule and scheduling system and has the potential to integrate with devices other than lights, such as sprinkler systems, garage door openers, and more.

While similar products do exist, PiLight attempts to fill a niche between "overly simplistic" and "walled garden". Standard wallwart plug timers are cheap, but their usefulness is limited; if you want your lights to turn on at sunset, for example, you must manually adjust the timer throughout the year. If you only want your lights on for the weekends, you're out of luck. High-tech Internet-connected automation systems are feature-rich, but are often very expensive and not easily customizable. Even worse, should the external Internet services running the show break or shut down, the entire system can be rendered useless. PiLight solves these issues by being self-contained; no external dependencies are required for the system to function normally. Furthermore, PiLight is built from commodity hardware and open source software, so the system is not only inexpensive, but also easy to extend and personalize, for the technically inclined.

PiLight consists of a wireless network of tiny "client" computers that each control up to 4 sets of lights and are controlled by a central "server" computer, which automatically sends out commands to the clients when it's time to turn on or off. The central node runs a control program that can be easily accessed via a touch screen, a web browser, or a mobile device, where the user can locally or remotely control each light individually. The control interface allows users to easily set "rules" for what their lights do and when, depending on the time of day, day of the week, sun position, and potentially even triggers such as weather conditions or calendar dates.

A Raspberry Pi with a small touchscreen and Wi-Fi card comprises the central control unit; it runs its own web server to allow control of lights, as well as a wireless network that allows both users and client nodes to connect. The client nodes are each composed of an ESP8266 Wi-Fi module and a relay, which automatically connect to the Pi's wireless network when the "Add Devices" button is pressed on the web UI. In a commercial implementation of PiLight, the client nodes could be integrated into a standard power strip/surge protector, so users can simply plug their devices in and start using the system immediately without having to order and wire up the components individually.

The PiLight team is composed of Malcolm Diller, Sean Rettig, and Evan Steele, who are computer science students at Oregon State University. Evan Steele's primary role was creating the Yocto Linux builds for the Raspberry Pi and writing the ESP8266 firmware, while Malcolm and Sean focused primarily on the web application, a Flask app with a Bootstrap frontend. In particular, Sean developed most of the rule system and advanced settings for lights, while Malcolm and Evan developed the front page UI, using Javascript to dynamically update the statuses and names of lights. Throughout development, we have met with our client, Victor, to help clarify project requirements, drive hardware/software design decisions, and provide general guidance to ensure the project's success.

# II. Original Requirements Document

## A. Team

*1) Team Name:* Cupcake Warriors

*2) Team Members:*


| | |
|---|---|
| Malcolm Diller | dillerm@oregonstate.edu |
| Sean Rettig | rettigs@oregonstate.edu |
| Evan Steele | steelee@oregonstate.edu |


*3) Client:*


Victor Hsu
Oregon State University
Phone: 541-737-4398
Email: hsuv@onid.orst.edu


## B. Introduction

*1) Problem Statement:* Outdoor lighting seems like a simple problem to solve, but the solutions on the market today are less than ideal. The standard transformer/timer combos available in the big box stores are rudimentary and clunky at best (constantly needing to be adjusted for the changing sunset and sunrise times), but are reasonably priced. The new-generation smart apps for home automation are flexible and fancier, but are quite spendy and lock you into a specific protocol. So why not use an open platform running on commodity hardware? Easy to use, reasonably priced, and highly customizable–that is our goal.

*2) Project Description:* Our system will consist of a wireless network of tiny "client" computers that each control up to 4 sets of lights and are controlled by a central "server" computer, which will automatically send out commands to the clients when it's time to turn on or off. The central node will run a control program that can be easily accessed via a touch screen, a web browser, or a mobile device, where the user can locally or remotely control each light individually. Want your lights to turn on at sunset and then dim gradually as the sun rises? Simple. Want your lights to flash when you're throwing a party? Just press a button. The control interface will allow users to easily set "rules" for what their lights do and when, depending on the time of day, the sun/moon position, and potentially even triggers such as weather conditions or calendar dates. This system will be easily extensible to potentially control other devices as well, such as garage doors, sound systems, and more.

*3) Design:* Specifically, we plan to use a Raspberry Pi running Linux as the central server computer. The Raspberry Pi will run a web server program that will allow nearby devices (such as laptops, phones, or tablets) to control it over a wireless network, or if connected to the home's internet connection, from practically anywhere in the world. The Pi will also have a small touchscreen connected to it with the website open in a browser, so the user has a dedicated interface for the device. The Pi will connect wirelessly to small wifi-enabled microcontrollers placed throughout the home, each of which are connected to up to 4 relays to control 4 sets of lights.

### *C. Requirements*

*1) Critical Requirements:* The following requirements are critical to the basic functionality of the system and must be functional prior to the expo in spring term.

1) Device control
   a) Server is loaded with Linux and is able to boot up to a graphical interface that can be viewed and controlled by the device's touchscreen.
   b) Server is able to start the server control program automatically when powered on.
   c) Clients are loaded with the client control program, which runs automatically when the device is powered on.
   d) Client control program is able to automatically discover relays/lights that are connected to it.
   e) Client control program can control the relays to power the lights on/off individually.
   f) Lights can be toggled over a user-specified period of time, e.g. a light can gradually turn on and grow brighter over the course of 30 seconds.
   g) Server is able to act as a wireless access point that each client can connect to.
   h) Clients can be paired with a server by putting the server into "pairing mode" (by pressing a button either on the user interface or on the Pi itself) that will last for a short time (just a few seconds). The server will then wirelessly broadcast UDP packets that nearby unpaired clients will see and use to connect to the server. This system will allow multiple servers to be used in close proximity with separate sets of clients.
   i) Clients are able to communicate to the server which lights are available for control.
   j) Server is able to send light toggle instructions wirelessly to clients.
   k) Clients are able to read light toggle instructions wirelessly from the server.
   l) Clients are able to perform light toggle instructions received from the server.
2) User interface
   a) Main control program on server serves a web site with a control interface for the user.
   b) The web interface is accessible via the system's built-in touchscreen.
   c) The web interface is accessible to other devices like laptops and phones via a local wireless network.
   d) The web interface displays a list of connected clients.
   e) The web interface allows users to give each client a "nickname" for easy identification.
   f) The web interface displays a list of connected lights.
   g) The web interface allows users to give each light a "nickname" for easy identification.
   h) The web interface contains buttons for each light that can be used to toggle lights individually.
   i) The web interface contains sliders for each light that can be used to change the intensity of lights individually.
   j) The web interface contains the ability to put lights into "groups" that can be controlled together all at once.
   k) The web interface displays a list of light groups.
   l) The web interface allows users to give each light group a "nickname" for easy identification.
   m) Groups can also be nested into other groups to create hierarchies of lights. For example, there can be two groups called "Front Porch" and "Back Porch" that control the front and back porch lights, respectively. Both groups can then be inside another group called "House". If the "House" group is toggled on, then both the front and back porch lights will be toggled on. If just the front porch lights are then toggled off, the back lights will stay on.

n) The web interface provides the user with options to toggle lights/groups based on rules, as described in the below "Rules" section.

3) Rules
   a) Lights can be toggled manually, overriding any rules (as described in the previous section). The manual setting will stay in effect until another rule is triggered. For example, if your lights are set to turn off every morning at 6am, but you manually turn them on at 7am, they will stay on all day until the next morning at 6am.
   b) Rules can be toggled manually to temporarily disable them. For example, if you have your lights set to turn on every night at 8pm, but are going on vacation for a week and don't need them, you can turn that rule off and it will stop triggering until you turn it on again, keeping the lights off until after you get back. This way, the rule doesn't need to be deleted and completely recreated from scratch.
   c) Lights can be toggled based on time of day (e.g. "turn on at 8pm and turn off at 6am").
   d) Lights can be toggled based on day of week (e.g. "turn on during Wednesdays").
   e) Lights can be toggled based on day of month (e.g. "turn on every 1st of the month").
   f) Lights can be toggled based on day of year (e.g. "turn on every January 1st").
   g) Lights can be toggled based on specific dates (e.g. "turn on from Dec. 24th at 2am to Dec 27th at 8pm").
   h) Lights can be toggled based on sunrise/sunset times (using sunset/sunrise times either preloaded onto the Pi or retrieved from an Internet service, and using a geographic location either entered by the user or determined through an Internet service)
   i) Multiple rules can be applied to a single light/group using an AND/OR system to combine the rules (e.g. "turn on between 8pm and 6am AND turn on on Wednesday" will turn the lights on between 8pm and 6am on Wednesdays, and will be off the rest of the time, whereas using an OR instead of an AND would cause the lights to be on every day from 8pm to 6am and also on all day during Wednesday).
   j) Lights can be toggled based on the toggle state of its parent group (e.g. by default, a light will turn on if its parent group turns on, but it can also be set to turn off if the parent group turns on).
   k) Rules can be set to toggle early/late by a constant or random period of time. For example, lights can be configured to turn on exactly 30 minutes after sunset, or perhaps randomly between 6pm and 6:30pm.
   l) Lights within a group can be set to individually toggle early/late by a random period of time so that they toggle in a staggered fashion. For example, if there are 5 lights in a group and are set to toggle off at 8pm, you can have the first light randomly toggle a few seconds before 8pm, then the next a few seconds later, etc. in a random order and with randomized times.
   m) Lights/groups can be set to gradually dim/brighten over a set period of time. For example, lights can turn on and gradually brighten from sunset to 30 minutes after sunset, after which they are at full brightness.

*2) Stretch Goals:* The following requirements are optional; they are not critical to the basic functionality of the system and may be implemented only if time permits.

resume

1) Device control
   a) System also controls devices other than lights
      i) Garage door openers (e.g. users can program their garage door to automatically close at night).
      ii) Music players (e.g. users can set music to automatically play in the evenings from an attached music player or from the Internet).
      iii) Holiday decorations (e.g. users can set snow globes or animatronic deer to automatically turn on at night)
      iv) Sprinkler systems (e.g. users can set their sprinklers to turn on from 4am to 5am).
2) User interface
   a) The web interface is accessible over the Internet (i.e. the user can control the system from anywhere in the world with an Internet connection, not just at their home)
      i) The web interface is accessible only to the homeowner or other authorized individuals (to prevent the general public from being able to view/control the user's lights). This could be implemented by asking the user to provide a password when logging into the system for the first time from a particular device. The device could then stay logged in via a cookie until the user logs out.
      ii) The web interface is hosted on a remote server for high availability and the lack of need for the user to perform port forwarding on their router. If the web server is running on the user's existing home network, the site won't be available unless the user manually opens their router's interface and creates a port forwarding rule, and also won't be available in the case that the user's router/modem loses power or connection to the Internet. If the site was hosted externally, the user would still be able to view how the system is currently set up and make changes that will apply once the user's power and/or Internet connection are restored.
3) Rules

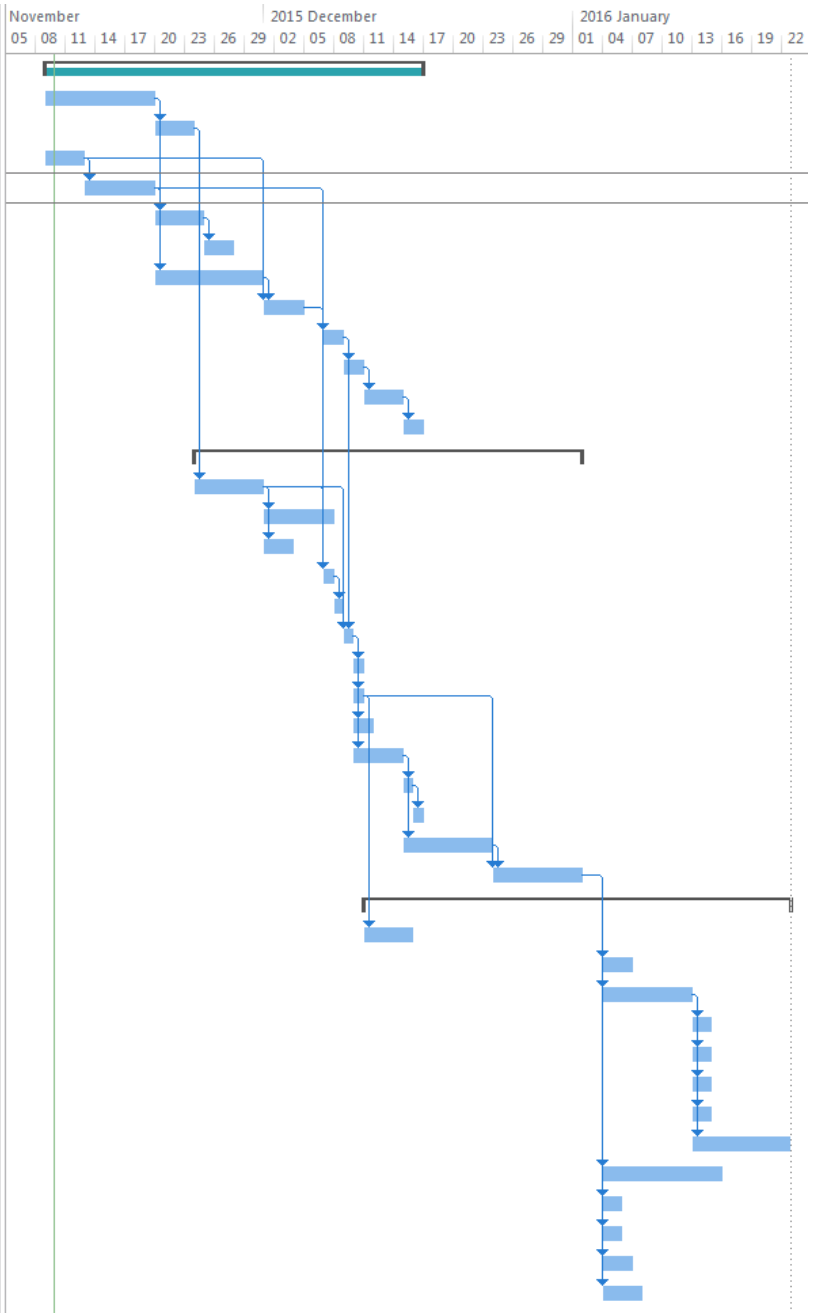
a) Lights can be toggled based on weather conditions (e.g. lights can turn on automatically if it starts raining). This would require an Internet connection or light/moisture sensors.
b) Lights can be toggled based on a schedule in the user's calendar (e.g. a user could create a lighting schedule on their Google calendar and it would "sync" with the lighting system). This would require Internet connection.
c) Lights can be toggled based on moon position or other celestial data (e.g. lights could turn off during a full moon or solar eclipse for better viewing). This would likely require either an Internet connection or a cache of preloaded data and the user's location.
d) Lights can be toggled based on input from attached sensors
    i) Motion sensors (e.g. lights turn on when someone walks up to the front door)
    ii) Light sensors (e.g. lights turn off at a certain brightness level, regardless of time of day, weather, moon phase, etc. This would also account for an area's brightness changing with the seasons and tree cover).
    iii) Moisture sensors (e.g. lights turn on when it's wet outside, negating the need for an Internet connection to determine if it's raining and possibly providing more accurate results. Could also potentially be used in combination with sprinkler integration to stop watering once a certain moisture level has been reached).
    iv) Sound sensors (e.g. lights turn on after a loud noise, or can strobe according to the beat of music that is playing).

### *D. Preliminary Timetable*

By the end of fall term, we plan to have a working proof-of-concept, using the Raspberry Pi and a basic control program to toggle lights on and off. By the end of winter term, we plan to have all required features at least partially implemented, at which our project enters the beta phase. As we finish up by ironing out bugs and perhaps completing stretch goals, we will release version 1.0 prior to the expo.

*1) Gantt Chart:* Gantt chart available at our project SharePoint website: https://oregonstateuniversity-my.sharepoint.com/personal/rettigs_oregonstate_edu/capstone22_project/_layouts/15/start.aspx#/Lists/Tasks/gantt.aspx

| Task Name | Duration | Start | Finish | Predecessors |
|---|---|---|---|---|
| **1. Device control** | **28 days** | **Mon 15-11-09** | **Wed 15-12-16** | |
| (a) Server can boot with GUI and touchscreen support | 9 days | Mon 15-11-09 | Thu 15-11-19 | |
| (b) Server starts control program on power on | 2 days | Fri 15-11-20 | Mon 15-11-23 | 2 |
| (c) Clients are loaded with client control program | 4 days | Mon 15-11-09 | Thu 15-11-12 | |
| (d) Clients discover connected relays/lights | 5 days | Fri 15-11-13 | Thu 15-11-19 | 4 |
| (e) Clients can toggle lights individually | 3 days | Fri 15-11-20 | Tue 15-11-24 | 5 |
| (f) Lights can be toggled over time | 3 days | Wed 15-11-25 | Fri 15-11-27 | 6 |
| (g) Server acts as wireless access point for clients | 7 days | Fri 15-11-20 | Mon 15-11-30 | 2 |
| (h) Clients can be paired with servers | 4 days | Tue 15-12-01 | Fri 15-12-04 | 8,4 |
| (i) Clients tell server which lights are available | 2 days | Mon 15-12-07 | Tue 15-12-08 | 9,5 |
| (j) Server can send light instructions to clients | 2 days | Wed 15-12-09 | Thu 15-12-10 | 10 |
| (k) Clients can receive light instructions from server | 2 days | Fri 15-12-11 | Mon 15-12-14 | 11 |
| (l) Clients can perform light instructions from server | 2 days | Tue 15-12-15 | Wed 15-12-16 | 12 |
| **2. User interface** | **29 days** | **Tue 15-11-24** | **Fri 16-01-01** | |
| (a) Server serves web site with control interface for user | 5 days | Tue 15-11-24 | Mon 15-11-30 | 3 |
| (b) Accessible via the system's built-in touchscreen | 5 days | Tue 15-12-01 | Mon 15-12-07 | 15 |
| (c) Accessible to other devices like phones | 3 days | Tue 15-12-01 | Thu 15-12-03 | 15 |
| (d) Displays list of connected clients | 1 day | Mon 15-12-07 | Mon 15-12-07 | 15,9 |
| (e) Can nickname clients | 1 day | Tue 15-12-08 | Tue 15-12-08 | 18 |
| (f) Displays list of connected lights | 1 day | Wed 15-12-09 | Wed 15-12-09 | 15,10 |
| (g) Can nickname lights | 1 day | Thu 15-12-10 | Thu 15-12-10 | 20 |
| (h) Buttons to toggle lights individually | 1 day | Thu 15-12-10 | Thu 15-12-10 | 20 |
| (i) Sliders to change light intensity | 2 days | Thu 15-12-10 | Fri 15-12-11 | 20 |
| (j) Can put lights into groups | 3 days | Thu 15-12-10 | Mon 15-12-14 | 20 |
| (k) Displays list of groups | 1 day | Tue 15-12-15 | Tue 15-12-15 | 24 |
| (l) Can nickname groups | 1 day | Wed 15-12-16 | Wed 15-12-16 | 25 |
| (m) Groups can be nested | 7 days | Tue 15-12-15 | Wed 15-12-23 | 24 |
| (n) Can toggle lights/groups based on rules | 7 days | Thu 15-12-24 | Fri 16-01-01 | 27,22 |
| **3. Rules** | **31 days** | **Fri 15-12-11** | **Fri 16-01-22** | |
| (a) Lights can be toggled manually, overriding any rules | 3 days | Fri 15-12-11 | Tue 15-12-15 | 22 |
| (b) Rules can be temporarily disabled | 3 days | Mon 16-01-04 | Wed 16-01-06 | 28 |
| (c) Lights can be toggled based on time of day | 7 days | Mon 16-01-04 | Tue 16-01-12 | 28 |
| (d) Lights can be toggled based on day of week | 2 days | Wed 16-01-13 | Thu 16-01-14 | 32 |
| (e) Lights can be toggled based on day of month | 2 days | Wed 16-01-13 | Thu 16-01-14 | 32 |
| (f) Lights can be toggled based on day of year | 2 days | Wed 16-01-13 | Thu 16-01-14 | 32 |
| (g) Lights can be toggled based on specific dates | 2 days | Wed 16-01-13 | Thu 16-01-14 | 32 |
| (h) Lights can be toggled based on sunrise/sunset times | 8 days | Wed 16-01-13 | Fri 16-01-22 | 32 |
| (i) Multiple rules per light/group | 10 days | Mon 16-01-04 | Fri 16-01-15 | 28 |
| (j) Lights can be toggled based on parent group | 2 days | Mon 16-01-04 | Tue 16-01-05 | 28 |
| (k) Rules can toggle early/late by random time | 2 days | Mon 16-01-04 | Tue 16-01-05 | 28 |
| (l) Lights can toggle early/late randomly within groups | 3 days | Mon 16-01-04 | Wed 16-01-06 | 28 |
| (m) Lights can gradually toggle over time | 4 days | Mon 16-01-04 | Thu 16-01-07 | 28 |



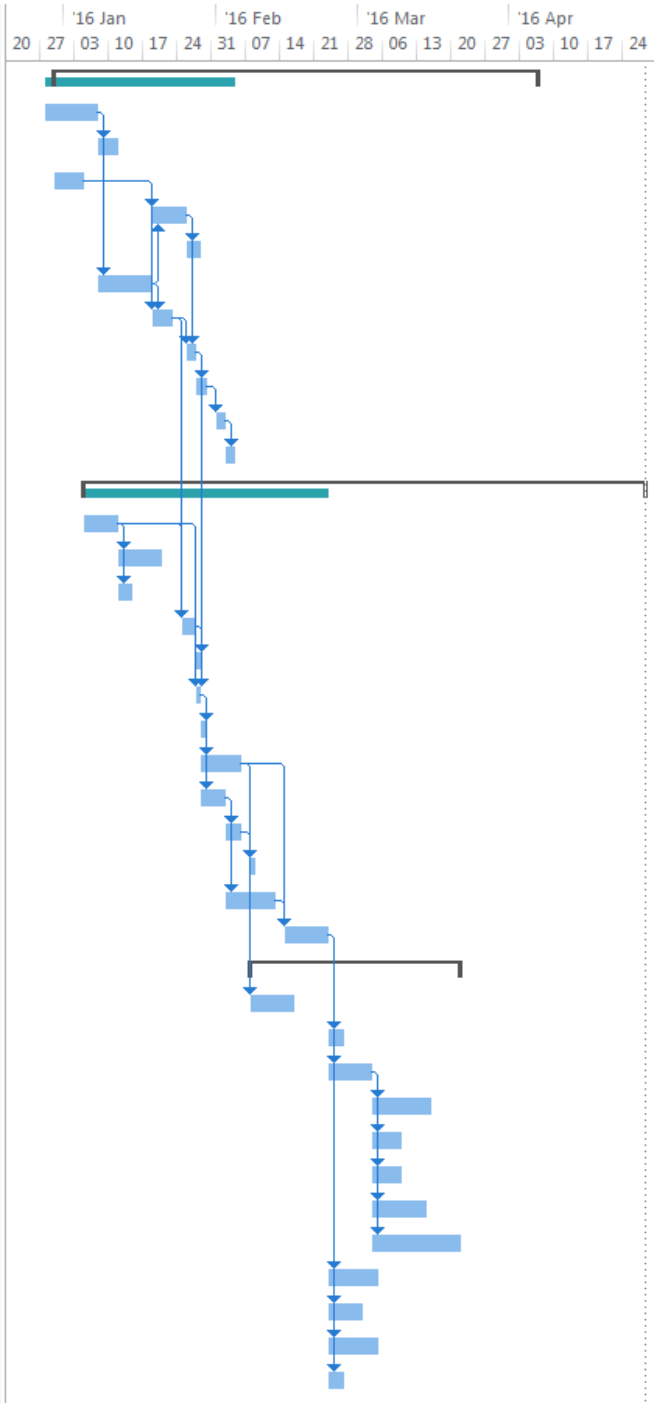
## III. Updated Requirements

Since the original requirements document, only three of our requirements have been updated, and all for the same reason; they were made into stretch goals due to hardware limitations.

| | Requirement | Result | Comments |
|---|---|---|---|
| 1f | Lights can be toggled over a user-specified period of time, e.g. a light can gradually turn on and grow brighter over the course of 30 seconds. | Changed to stretch goal. | We decided that this functionality was not possible with the provided relays, which can only toggle lights on or off, not vary their intensity. |
| 2i | The web interface contains sliders for each light that can be used to change the intensity of lights individually. | Changed to stretch goal. | We decided that this functionality was not possible with the provided relays, which can only toggle lights on or off, not vary their intensity. |
| 3m | Lights/groups can be set to gradually dim/brighten over a set period of time. For example, lights can turn on and gradually brighten from sunset to 30 minutes after sunset, after which they are at full brightness. | Changed to stretch goal. | We decided that this functionality was not possible with the provided relays, which can only toggle lights on or off, not vary their intensity. |

### *A. Updated Gantt Chart*

| Task Name | Duration | Start | Finish | Predecessors |
|---|---|---|---|---|
| **1. Device control** | **71 days** | **Wed 15-12-30** | **Wed 16-04-06** | |
| (a) Server can boot with GUI and touchscreen support | 9 days | Mon 15-12-28 | Thu 16-01-07 | |
| (b) Server starts control program on power on | 2 days | Fri 16-01-08 | Mon 16-01-11 | 2 |
| (c) Clients are loaded with client control program | 4 days | Wed 15-12-30 | Mon 16-01-04 | |
| (d) Clients discover connected relays/lights | 5 days | Tue 16-01-19 | Mon 16-01-25 | 4,7 |
| (e) Clients can toggle lights individually | 3 days | Tue 16-01-26 | Thu 16-01-28 | 5 |
| (g) Server acts as wireless access point for clients | 7 days | Fri 16-01-08 | Mon 16-01-18 | 2 |
| (h) Clients can be paired with servers | 4 days | Tue 16-01-19 | Fri 16-01-22 | 7,4 |
| (i) Clients tell server which lights are available | 2 days | Tue 16-01-26 | Wed 16-01-27 | 8,5 |
| (j) Server can send light instructions to clients | 2 days | Thu 16-01-28 | Fri 16-01-29 | 9 |
| (k) Clients can receive light instructions from server | 2 days | Mon 16-02-01 | Tue 16-02-02 | 10 |
| (l) Clients can perform light instructions from server | 2 days | Wed 16-02-03 | Thu 16-02-04 | 11 |
| **2. User interface** | **83 days** | **Tue 16-01-05** | **Thu 16-04-28** | |
| (a) Server serves web site with control interface for user | 5 days | Tue 16-01-05 | Mon 16-01-11 | |
| (b) Accessible via the system's built-in touchscreen | 7 days | Tue 16-01-12 | Wed 16-01-20 | 14 |
| (c) Accessible to other devices like phones | 3 days | Tue 16-01-12 | Thu 16-01-14 | 14 |
| (d) Displays list of connected clients | 3 days | Mon 16-01-25 | Wed 16-01-27 | 14,8 |
| (e) Can nickname clients | 1 day | Thu 16-01-28 | Thu 16-01-28 | 17 |
| (f) Displays list of connected lights | 1 day | Thu 16-01-28 | Thu 16-01-28 | 14,9 |
| (g) Can nickname lights | 1 day | Fri 16-01-29 | Fri 16-01-29 | 19 |
| (h) Buttons to toggle lights individually | 6 days | Fri 16-01-29 | Fri 16-02-05 | 19 |
| (j) Can put lights into groups | 3 days | Fri 16-01-29 | Tue 16-02-02 | 19 |
| (k) Displays list of groups | 3 days | Wed 16-02-03 | Fri 16-02-05 | 22 |
| (l) Can nickname groups | 1 day | Mon 16-02-08 | Mon 16-02-08 | 23 |
| (m) Groups can be nested | 8 days | Wed 16-02-03 | Fri 16-02-12 | 22 |
| (n) Can toggle lights/groups based on rules | 7 days | Mon 16-02-15 | Tue 16-02-23 | 25,21 |
| **3. Rules** | **31 days** | **Mon 16-02-08** | **Mon 16-03-21** | |
| (a) Lights can be toggled manually, overriding any rules | 7 days | Mon 16-02-08 | Tue 16-02-16 | 21 |
| (b) Rules can be temporarily disabled | 3 days | Wed 16-02-24 | Fri 16-02-26 | 26 |
| (c) Lights can be toggled based on time of day | 7 days | Wed 16-02-24 | Thu 16-03-03 | 26 |
| (d) Lights can be toggled based on day of week | 8 days | Fri 16-03-04 | Tue 16-03-15 | 30 |
| (e) Lights can be toggled based on day of month | 4 days | Fri 16-03-04 | Wed 16-03-09 | 30 |
| (f) Lights can be toggled based on day of year | 4 days | Fri 16-03-04 | Wed 16-03-09 | 30 |
| (g) Lights can be toggled based on specific dates | 7 days | Fri 16-03-04 | Mon 16-03-14 | 30 |
| (h) Lights can be toggled based on sunrise/sunset times | 12 days | Fri 16-03-04 | Mon 16-03-21 | 30 |
| (i) Multiple rules per light/group | 8 days | Wed 16-02-24 | Fri 16-03-04 | 26 |
| (j) Lights can be toggled based on parent group | 5 days | Wed 16-02-24 | Tue 16-03-01 | 26 |
| (k) Rules can toggle early/late by random time | 8 days | Wed 16-02-24 | Fri 16-03-04 | 26 |
| (l) Lights can toggle early/late randomly within groups | 3 days | Wed 16-02-24 | Fri 16-02-26 | 26 |



# IV. Design Document

## A. Introduction

In the twenty-first century, your home should be smart and responsive. An automated home should be able to control appliances such as lights through single-board computer units that have Internet connectivity. Many commercial options exist, such as those from Nest Labs, but such devices are prohibitively expensive for most consumers, especially for simple tasks such as lighting automation. This project aims to provide a low-cost alternative for simple home automation, for devices like lights and garage door openers. Using a low-cost embedded computer such as the Raspberry Pi and cheap wireless communication modules such as the ESP8266, this project will provide a cheap, easy-to-install home automation system.

The project is called "Not Exactly the Internet-of-Things" because it's not designed as a typical IoT product. The network is entirely local, not worrying about an external connection to the Wide Area Network. Some additional components such as consulting an external API could be considered, but that's an addon to the finished product. As it is made as a home application, it is readily accessible through smartphones and web interfaces. With a clean user interface that enables both manual control and automation profiles, the application will require a minimal learning curve and be simple to set up.

## B. Design View

The aim of this project is to create a home lighting automation system that is inexpensive, wireless, and easy to use. The user of this product should the following features:

1) Lights and Groups
    a) The lights can be controlled from the built-in touchscreen on the control unit, or a mobile device such as a phone, tablet, or laptop
    b) Lights can be put into "groups" so that they can be controlled all at once
    c) Groups can also be nested into other groups to create hierarchies of lights, in order to control many lights at once
    d) The user will be able to nickname lights and light groups
    e) The intensity of each light can be adjusted using a slider
2) Rules / Types of Control
    a) Lights can be toggled on and off manually, but they can also be set to be activated on specific schedules using "rules"
    b) Rules can be set for lights or groups of lights
    c) There are 6 main types of schedules that rules can trigger on
        i) Time of day (e.g. "turn on at 8pm and turn off at 6am")
        ii) Day of week (e.g. "turn on during Wednesdays")
        iii) Day of month (e.g. "turn on every 1st of the month")
        iv) Day of year (e.g. "turn on every January 1st")
        v) Specific dates (e.g. "turn on from Dec. 24th at 2am to Dec 27th at 8pm")
        vi) Sunrise / sunset (e.g. "turn on when the sun sets, turn off when the sun rises")
    d) Multiple rules can be applied to each light or group of lights by using AND or OR logic
    e) Lights can be toggled based on the toggle state of its parent group
    f) Rules can be enabled or disabled temporarily
    g) Lights or groups can be set to gradually dim/brighten over a set period of time instead of toggling
3) Stretch Goals
    a) The system can control devices other than just lights:
        i) Garage door openers
        ii) Music players
        iii) Holiday decorations
        iv) Sprinkler systems
    b) The web interface is accessible over the Internet (Instead of only on the local network)
        i) The interface is accessible for only the homeowner or authorized individuals
        ii) The interface hosted on a remote server if the user does not want to deal with port forwarding their home network.
    c) Additional types of triggers for rules
        i) Weather conditions (e.g. lights can turn on automatically if it starts raining)
        ii) Schedule in a Google calendar

iii) Moon position or other celestial data (e.g. lights could turn off during a full moon or solar eclipse for better viewing)
iv) Motion sensors (e.g. lights turn on when someone walks up to the front door)
v) Light sensors (e.g. lights turn off at a certain brightness level)
vi) Moisture sensors (e.g. lights turn on when it's wet outside)
vii) Sound sensors (e.g. lights turn on after a loud noise, or strobe to the beat of music that is playing)

### *C. Design Viewpoints*

We will be using a wide variety of open source technologies and inexpensive hardware to provide a low-cost home automation system. We will be using a varied set of tools to accomplish our goal, including:

1) The Yocto Project
    a) The Yocto Project is a kernel building system designed for embedded systems. It easily incorporates additional features and custom packages directly from Git repositories. It makes it easy to build a customized image with only exactly what we want and no unnecessary packages [1]. It also allows us to customize elements of the kernel such as startup scripts in init.d, kernel modules, and network scripts [2]. The kernel will be compiled for the Raspberry Pi and copied to an SD card using Yocto's filesystem script.
2) Raspberry Pi 2
    a) The Raspberry Pi 2 is an ARMv7-processor-based microcomputer designed for educational and embedded projects [3]. We will use it as a low-cost server solution for the lighting automation system. It can serve as the web server running NGINX and transmit TCP commands to the ESP8266 Wi-Fi modules [4]. The Pi will be outfitted with a 320x280 TFTLCD display [5] so that the interface for control can be accessed directly. The Pi will also act as a wireless router for the plug units to connect to, and run a DHCP server to distribute IP addresses to the plug units.
3) ESP8266
    a) The ESP8266 is a self-contained SoC (System on a Chip) that has a built-in Wi-Fi radio [6]. Our module is outfitted with custom firmware, written in Lua and flashed to the device using the Arduino IDE, that allows the module to automatically connect to the Raspberry Pi (control unit) that acts as a WAP (Wireless Access Point) [7]. The board we have is designed to connect to an electronic relay using jumper cables. Using simple TCP commands, the board will flip the relay pins on and off [8].

### *D. User Experience*

The final deliverable of a commercial version of this product would likely include the following:

- A central control unit consisting of the Raspberry Pi, its power cable, the touchscreen module, a case, and an SD card with the server software preloaded onto it.
- One or more plug units which would likely resemble a power strip, with each plug being individually controllable by the system.

To set up the system, the user simply plugs both the control unit and the plug unit(s) into wall power. The plug units will preconfigured to connect to the control unit. The user can then plug devices into the plug units, such as lamps, and the display on the control unit will update itself automatically with a list of lights. The lights can have either numbered (e.g. "Zone 1, Light 1") or randomized but memorable names by default (e.g. "Blue Koala") which can be later edited through the web interface. To access the web interface, the user connects to the Wi-Fi network provided by the control unit using the credentials included with the manual, or perhaps printed on the control unit itself. They then open a web browser and connect to a specific URL, provided to the user with the Wi-Fi password. From the web interface, all features are accessible, such as renaming lights, creating light groups, and applying rules for when to toggle lights.

### *E. Testing*

To test the functionality and correctness of our system, we will test the various components both individually and when integrated with other components:

- We plan to test the wireless communication capabilities of the system by attempting to send packets between the control unit and plug units at varying distances and in varying environments, including open and walled areas. Ideally, the range should be great enough to fully cover the average American house/apartment. Once this is complete, it will likely not need to be changed or extended much, reducing the need for regression testing.
- We plan to test the actual light toggling (as performed by the plug units) manually. Once this is complete, it will likely not need to be changed or extended much, reducing the need for regression testing.

- We plan to test the functionality of the touchscreen interface through manual human interaction testing. Once this is complete, it will likely not need to be changed or extended much, reducing the need for regression testing.
- We plan to test the functionality of the web interface through a combination of manual human interaction testing, automated API-driven testing, and automated GUI testing (using a platform such as Selenium). The plug units can send back acknowledgement packets to confirm what actions they took in response to the tests, allowing nearly complete integration tests to be performed automatically when changes are made to the control program and web interface. These automated tests will help prevent regressions as new features are developed and save time in ensuring the correctness of the system.

### *F. Timeline*

| **Device control** | **Due Date** |
|---|---|
| Server can boot with GUI and touchscreen support | Thu 11/19/15 |
| Server starts control program on power on | Mon 11/23/15 |
| Clients are loaded with client control program | Thu 11/12/15 |
| Clients discover connected relays/lights | Thu 11/19/15 |
| Clients can toggle lights individually | Tue 11/24/15 |
| Lights can be toggled over time | Fri 11/27/15 |
| Server acts as wireless access point for clients | Mon 11/30/15 |
| Clients can be paired with servers | Fri 12/4/15 |
| Clients tell server which lights are available | Tue 12/8/15 |
| Server can send light instructions to clients | Thu 12/10/15 |
| Clients can receive light instructions from server | Mon 12/14/15 |
| Clients can perform light instructions from server | Wed 12/16/15 |
| **User Interface** | |
| Server serves web site with control interface for user | Mon 11/30/15 |
| Accessible via the systems built-in touchscreen | Mon 12/7/15 |
| Accessible to other devices like phones | Thu 12/3/15 |
| Displays list of connected clients | Mon 12/7/15 |
| Can nickname clients | Tue 12/8/15 |
| Displays list of connected lights | Wed 12/9/15 |
| Can nickname lights | Thu 12/10/15 |
| Buttons to toggle lights individually | Thu 12/10/15 |
| Sliders to change light intensity | Fri 12/11/15 |
| Can put lights into groups | Mon 12/14/15 |
| Displays list of groups | Tue 12/15/15 |
| Can nickname groups | Wed 12/16/15 |
| Groups can be nested | Wed 12/23/15 |
| Can toggle lights/groups based on rules | Fri 1/1/16 |
| **Rules** | |
| Lights can be toggled manually, overriding any rules | Tue 12/15/15 |
| Rules can be temporarily disabled | Wed 1/6/16 |
| Lights can be toggled based on time of day | Tue 1/12/16 |
| Lights can be toggled based on day of week | Thu 1/14/16 |
| Lights can be toggled based on day of month | Thu 1/14/16 |
| Lights can be toggled based on day of year | Thu 1/14/16 |
| Lights can be toggled based on specific dates | Thu 1/14/16 |
| Lights can be toggled based on sunrise/sunset times | Fri 1/22/16 |
| Multiple rules per light/group | Fri 1/15/16 |
| Lights can be toggled based on parent group | Tue 1/5/16 |
| Rules can toggle early/late by random time | Tue 1/5/16 |
| Lights can toggle early/late randomly within groups | Wed 1/6/16 |
| Lights can gradually toggle over time | Thu 1/7/16 |

### *G. Changes to the Design Document*

Over the course of the year, there were a few changes to the project that had to be made to allow for new problems that came up when implementing some of the features specified in the document. The only change that we have made to the design document since it was created was that we had to remove the lines that stated “The intensity of each light can be adjusted using a slider”, and “Lights or groups can be set to gradually dim/brighten over a set period of time instead of toggling.” This had to be done because relays only allow us to have two states, and unless we used a different implementation and way to connect to the lights, there is physically no way for us to do dimming or brightening.

# V. Technology Review

## A. Client

Victor Hsu
Oregon State University
Phone: 541-737-4398
Email: hsuv@onid.orst.edu

## B. Problem Statement

Outdoor lighting seems like a simple problem to solve, but the solutions on the market today are less than ideal. The standard transformer/timer combos available in the big box stores are rudimentary and clunky at best (constantly needing to be adjusted for the changing sunset and sunrise times), but are reasonably priced. The new-generation smart apps for home automation are flexible and fancier, but are quite spendy and lock you into a specific protocol. So why not use an open platform running on commodity hardware? Easy to use, reasonably priced, and highly customizable–that is our goal.

## C. Project Description

Our system will consist of a wireless network of tiny client computers that each control up to 4 sets of lights and are controlled by a central ”server” computer, which will automatically send out commands to the clients when it’s time to turn on or off. The central node will run a control program that can be easily accessed via a touch screen, a web browser, or a mobile device, where the user can locally or remotely control each light individually. Want your lights to turn on at sunset and then dim gradually as the sun rises? Simple. Want your lights to flash when you’re throwing a party? Just press a button. The control interface will allow users to easily set ”rules” for what their lights do and when, depending on the time of day, the sun/moon position, and potentially even triggers such as weather conditions or calendar dates. This system will be easily extensible to potentially control other devices as well, such as garage doors, sound systems, and more.

## D. Pieces

1) OS for the PI
   a) Raspbian
      Custom built for the PI, this operating system would provide many important features, and be a solid foundation to run on. Installation is very straightforward, and it has support all across the Internet. This means that the Pi would have to run the entire Raspbian operating system, so there will be many utilities we don’t need that will be included anyway. The image is easily acquired and it can be burned to a SD card and booted straight away. It will likely require heavy configuration to get all the necessary services running and the extra ones to never run.
   b) Yocto meta-recipe
      We will not need all of the fancy features that Raspbian offers, and one way we can strip our OS down to the features we need is by building a Linux system using Yocto. The Yocto build system is complicated, but our group has experience building layers for various images. It will allow us to build a custom distribution with every package we need, init.d scripts, and service scripts for only the services we need. It will require build time, and time to burn to installation media. Kevin noted that we could get a build server for the project by the end of the term, which would give us good hardware for Yocto builds.
   c) OpenWRT
      An open source router firmware package that allows embedded devices to perform complicated network tasks. Only recently was it recompiled to work with the Raspberry Pi. Our collective experience with the firmware is limited, so it would have an exceptional learning curve. It looks like it would be able to handle all the services we would need, but there isn’t a whole lot of flexibility for other additions. It’s also very difficult to debug or run tests with.

   We have decided to use a Yocto build of Linux for the final build. This decision was made since the Yocto build will only contain exactly what we need, will allow us to easily incorporate new packages with the recipe system, and can be easily shared on our Github repository as a meta-layer for others to build. Proof of concept builds will just use Raspbian with the packages installed.
2) Server Program Implementation
   a) Python / MicroPython
      The ESP8266 can run MicroPython for GPIO access, making Python the obvious choice for the server program

as well. Using Python's socket API, we can easily connect the devices and execute code to work with the GPIOs. The overhead of Python is minimal when considering the ease of use, and MicroPython was practically designed for the ESP8266, so there's lots of support available. Installing it is as simple as flashing a binary to the ESP8266, and the Python shell is supported. However, we would likely load in a Python script to handle incoming commands from the server. The ESP8266 has sufficient storage for one large Python script.

b) C / libmraa
Running the system through C is possible, thanks to libmraa which provides useful abstractions from the /sys/class/gpio and puts it into easy-to-use C libraries. While we could use these libraries, the Python variant provides an easier interface with only slightly more overhead. Additionally, the library contains SWIG-generated wrappers for a variety of languages, including Python, C++, and Perl. The repository is maintained by Intel's IoT Development team, and our team has contribution experience with libmraa.

c) sysfs
The most direct, but perhaps least reliable method is to just use the sysfs interface. This involves doing things like echoing values into files and reading the raw files for values. For instance, accessing the SPI interface for a device registered with major number 0 would look like: `cat /sys/class/spi_master/spi0/spi0.0/iio:device0` which is not ideal or clean. Additionally, it could break if we try to use it on systems with different sysfs layouts. It would be useful to use sysfs for testing, but we should not use bash scripts like this for our final product.

We have decided to use Python and MicroPython because of compatibility issues and ease-of-use for the devices. Python will run on both the ESP8266 and the Raspberry Pi with all the libraries we need to run the software. Our concerns about overhead are minuscule since the Python will be executing very simple code; it is just interacting with a relay switch. It will mesh well with our web interface, so we'll be using it as a CGI script in the web server.

3) Web Site Implementation

a) JavaScript
An easy language to work with, Javascript will allow us to use many different, responsive templates such as react.js and node.js for control. Node even has wrappers for sysfs functionality which may even allow us to deploy it on the ESP8266 devices for remote control. Using JS for the web interface would offload the majority of work from the Pi to the web browser, which could be useful depending on the complexity of our interface. If we end up using a GPIO abstraction library such as libmraa, it should be noted that most of them have wrappers to Javascript code.

b) Django
A web framework known best for rapid deployment, we could examine this technology as a way to power our entire web interface and backend system. It would have a longer ramp-up time since none of our group members are fully comfortable with Django. However, if we decide that the ramp-up time is worth it, we could see this platform as an all encompassing management tool. Django is well-supported, but does have large learning curve and would require exhaustive group research to understand the entire system.

c) PHP
The hypertext preprocessor is, at first glance, perhaps not the best system to run our backend off of, but we're considering it for a few reasons. It can execute our CGI scripts with the mod_php module for the web server, our group has significant working experience with the language, and it just works out of the box. It may require additional configuration, but from a 10,000 foot view it accomplishes all the tasks we need. It can mesh well with a wide variety of other technologies we may end up using, and is highly modular.

We have decided to use the HTML/JS/PHP stack for our project, which means we'll be building a system from the ground up. We can point the Javascript to the CGI files to execute the Python that will change the relays at the remote end, use PHP for the web backend and database access, and use HTML for the pages themselves (although this one will likely just be generated through the PHP code too) to create our full web server stack. This does not rule out us using other technologies such as node or Django as a subsystem.

4) Server/Client Communication

a) AD-HOC
In this communication method, the devices are configured for AD-HOC mode, or a packet-radio system. The wireless adapters need to be able to support AD-HOC mode and be set to a specific channel and IP range. Given the difficulty we've had with AD-HOC networks in the past, we'll maybe try to avoid this mode, but it would be useful as a proof-of-concept for socket communication. However, this is likely not a long-term solution.

b) Central WAP
We would configure one Raspberry Pi as a central Wireless Access Point and have the ESP8266 units connect to it as wireless clients. The advantage to this mode is that we could use the Raspberry Pi to connect to the Internet to perform tasks such as talking to the Wunderground API, which is not possible in AD-HOC mode.

This would require that the Pi run a DHCP server, along with other routing services.

c) CAS
A Central Authority Service model leans more toward the well-defined Internet of Things model. All of our devices rely on an external server for almost all their actions. It would coordinate between the devices, actually store all the data and perform all the heavy lifting, relying on the Pi and ESP8266 devices only for GPIO access. A drawback to this model is the additional server we'd need to configure and rely on, especially since the system would stop working if the server ever becomes inaccessible.

We have decided to use the WAP model because it provides the benefits of the AD-HOC and CAS models without all the additional configuration burdens. A Yocto meta-recipe for the Pi's WAP functionality would be simple, and connections between the ESP8266 devices would become a trivial task. This does mean the Pi has to run extra services (like DHCP), but it should be more than capable of running the limited services we will require of it.

### *E. Changes to the Technology Review*

The only large change that we made in regard to the decisions in our technology review was that for the website, we used no PHP to modify the database, and instead used a python Object Relational Mapper (ORM) called SQLAlchemy. Additionally, we used a python web framework for our website called Flask. Combined, these allowed for much cleaner code and a much easer to manage database access.

## VI. Blog

### *A. Fall Week 3 Update*

Progress since last week:

- We have finished the requirements document after having our client review it.
- We received our Raspberry Pi and touchscreen hardware.

Problems:

- No problems this week.

Plans for the coming week:

- Begin working on the Pi starting with an operating system and then branching out to communication between the devices and touchscreen capabilities.

Blog Date: 10-29-15

### *B. Fall Week 4 Update*

Most of what we did this week involved trying to get a functioning proof of concept for our project, which turned out to be filled with more problems than anticipated. We began to investigate software packages and technologies that would help us meet the requirements outlined on our document. We will hopefully have a functional proof-of-concept by the next week.

Progress since last week:

- Tested the devices with various wireless networking methods, including ad-hoc and in access-point mode
- Continued to develop our documents such as the technical requirements and elevator pitch

Problems:

- The ad-hoc system did not work as expected due to the USB WiFi adapters we were using, and we had to reevaluate our network implementation

Plans for the coming week:

- Continue to develop a proof-of-concept, working with a new structure with the wireless components that will function to expectations.

Blog Date: 11-5-15

### *C. Fall Week 5 Update*

Progress since last week:

- We now have a working proof of concept where the Pi is able to wirelessly communicate to the ESPs and cause each of the connections on the relays to be activated individually.
- Completed technology review and revision of requirements document.
- Started on expo poster.

Problems:

- No real problems other than having to revise the requirements document to be more detailed and fine-grained.

Plans for the coming week:

- Finish Poster
- Revise elevator pitch
- Demonstrate proof of concept
- Possibly attach actual lights to the relay rather than just using the relay's LEDs to see which switches are enabled.

Blog Date: 11-13-15

### *D. Fall Week 6 Update*

Progress since last week:

- Our proof of concept is now portable, and booting the pi connects it to the ESP8266 module and cycles the lights on the relay. It has also been shown to the TA.
- Finished expo poster
- Re-wrote elevator pitch

Problems:

- No real problems this week

Plans for the coming week:

- Continue to develop the proof of concept, possibly by adding actual lights to the relay instead of the LED's.
- Present the elevator pitch

Blog Date: 11-19-15

### *E. Fall Week 7 Update*

Progress since last week:

- We have connected actual lights to our relay and verified that they can be toggled on/off.
- We have presented our elevator pitch.
- We have begun learning how to use Flask for our website.

Problems:

- No real problems this week

Plans for the coming week:

- Work on our design document
- Become more familiar with Flask and potentially start on the website Use the Yocto build server to build the new system rather than using Raspbian.
- Work on progress report

Blog Date: 11-28-15

### *F. Fall Week 8 Update*

Progress since last week:

- We retrieved our build server from Kevin and installed Linux. We also installed the Yocto build system and built core-image-minimal to verify that it works.
- We booted the Raspberry Pi with core-image-minimal to verify that the images would work
- We investigated options for connections and user experiences, and decided on a wireless connection method for our system
- We completed our design document and ran it by our TA. We also continued work on the progress report.

Problems:

- We had a bit of a confused discussion on the user experience complexity and the wireless communication methods we would use, as we did not know the expectations for ease-of-use. We contacted Victor and had our questions answered.

Plans for the coming week:

- Work on our report
- Over break, start to build the web framework with Flask.

Blog Date: 12-4-15

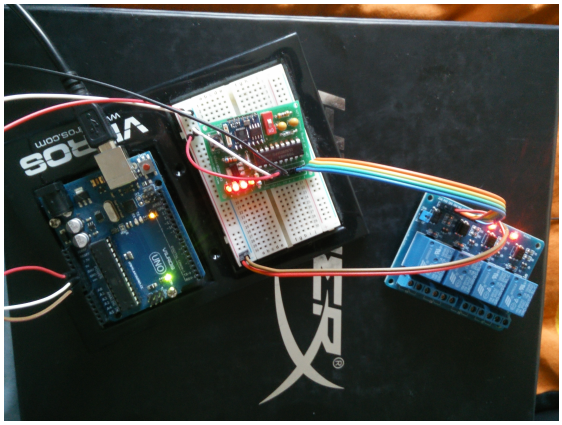


Fig. 1. An example of basic wiring, using an Arduino for powering the ESP8266 for 5v

### G. Winter Week 1 Update

Over the break, our group refreshed our working knowledge of Flask, configured our new device for our Yocto build server and continued testing with the Raspberry Pi. We resumed work on the project and plan to meet during the weeks throughout the term to coordinate work, although most of our work will happen outside of our meetings.

Progress over break:

- Reviewed Python Flask to be better prepared to start working on it to build our web interface.
- Configured the Raspberry Pi with software to run the web interface such as the ISC DHCP server and a basic DNS server.
- Used Hostapd and wpa_supplicant to have the Wi-fi adapter on the Pi broadcast an access point.

Week 1 work:

- Met to discuss web layout, decided on various Javascript libraries to run the application
- Continued to develop software configuration for the Raspberry Pi and the corresponding Yocto image.

Problems:

- Some minor nonviolent disagreements on design for the front end of the application

Plans for the coming week:

- Continue development of front end application
- Finish back-end scripts to send the commands to the ESP8266 module
- Finish Python ISC-DHCP autodiscovery scripts

Blog Date: 1-8-16

### H. Winter Week 2 Update

Progress since last week:

- Changed requirements document to show that adjusting the brightness of lights is now a stretch goal. (requirements 1f, 2i, and 3m)
- Created our first way to send the TCP command for turning on and off the lights.
- Added database capability for the ESP devices.
- Reformatted the layout of the website to be more compartmentalized and manageable.
- Added python data structures for groups and lights

Problems:

- We realized that we would not be able to accomplish our goal of being able to adjust the brightness of the lights with the hardware we have available. We changed this goal to a stretch goal in the requirements document.

Plans for the coming week:

- Continue development of front end application
- Work more on connecting the website to the functions of the hardware

Blog Date: 1-15-16

### *I. Winter Week 3 Update*

Progress since last week:

- Added interface to add and view current connected ESP8266 devices, along with error handling.
- Performed minor formatting and usability updates to the front page.
- Created a basic Advanced page to accept dates/times for when to turn lights on/off and a basic query string that is able to live update via Javascript.

Problems:

- No notable problems this week.

Plans for the coming week:

- Malcolm plans to work on making the front page of the site more usable by allowing the editing of names, the creation/deletion of groups, and the sending of reordering operations to the server from the web browser.
- Sean plans to continue work on the Advanced page for creating rules for lights, including creating a proper query string and actually sending the updates back to the server.

Blog Date: 1-22-16

### *J. Winter Week 4 Update*

Progress since last week:

- Decided on a database layout for creating and displaying light groups on the front page of the site.
- Continued to add support for the advanced page and improving the Javascript creating the custom queries.
- Added login functionality and created a default user when the database is created. We'll actually apply the security to the pages when the project is near completion.

Problems:

- Minor disagreements on what to use for the group table structure, but resolved during Thursday's meeting.

Plans for the coming week:

- Malcom will add the group display functionality to the front page, and will apply the *-jquery_editable* functions to the group entries to allow easy editing of group attributes.
- Sean will flesh out the logic generation subsystem as it interacts with the new light group management so that manual overrides are possible without disruption customized schedules.
- Evan will finish the devices display and modification page by applying the *jquery_editable* package to it and will then work on the group layout with Malcom.

### *K. Winter Week 5 Update*

Progress since last week:

- Re-Wrote querybuilding on the server and started on the client's querybuilding.
- Finalized the database schema and the flask sql-alchemy object relational model format.
- Additional changes to the advanced page

Problems:

- No major problems this week.

Plans for the coming week:

- We will work on and complete the Midterm Progress Report, as well as the accompanying video.

Blog Date: 2-8-16

### *L. Winter Week 6 Update*

Progress since last week:

- Updated the formatting, fixed general bugs, and cleaned up the code for the website in both the advanced page and the main page
- Finished the query builder for the advanced page
- Rules now appear on the side when created
- Finished both the progress report and the video to go along with it.

Problems:

- No major problems this week.

Plans for the coming week:

- We will continue to work on achieving beta-level functionality for all parts of the project

Blog Date: 2-12-16

*M. Winter Week 7 Update*

Progress since last week:

- Implemented sorting on the Flask app's front page for groups
- Added JS for deleting groups and adding of new groups
- Continued logical development for the advanced page and custom user queries
- Fixed some minor styling bugs with custom CSS

Problems:

- No major problems this week.

Plans for the coming week:

- Continue working on the advanced user queries
- Implement group change interaction with the database on the front page

Blog Date: 2-19-16

*N. Winter Week 8 Update*

Progress since last week:

- Reorganized advanced page layout for usability according to feedback from user study.
- Removed ability to apply multiple rules to each light/group to improve usability, since users found the concept of multiple rules per light/group confusing. No functionality is lost, however, since parent group rules can still be applied, and custom queries can be created that achieve the same result as multiple rules.
  - Since each light/group only has one rule now, the rules table was removed from the database and a rule column was added to the light and group tables.
- Advanced page now sends rule changes to the server via AJAX, updating the database immediately whenever a change is made. This negates the need for the user to hit a "save changes" button or the like, which users responded poorly to in the user study.
- Fixed bug with deleting groups that have nested group children on the main page.
- Fixed additional minor styling bugs, such as with the light/group name editing box on the main page appearing on a new row.

Problems:

- No major problems this week.

Plans for the coming week:

- Implement job for re-evaluating rules on startup and periodically, including calculating proper sunset/sunrise times, and then sending updates to each light.
- Make further style/usability tweaks to site, including front page and advanced page, such as making buttons bigger, adding help text, fixing alignment of elements, etc.
- Create settings page
  - Currently, only planned setting is for geographical location for calculating accurate sunset/sunrise times.
- Integration testing
  - Starting from an empty database, add a new client device and toggle the lights from the main web UI and through rules.
  - Test the use of the website on the Pi's touchscreen.

Blog Date: 2-26-16

*O. Winter Week 9 Update*

Progress since last week:

- As we near the end of our project, we have started using Github's issue tracker for all bugs and yet-to-be-implemented features:
  - https://github.com/capstone22-Pilight/cs-senior-capstone/issues
- Essentially finished beta version of project, completing all high priority issues.
- The main portion of the project that was finished was the advanced page and rule system; the advanced page has been reformatted to make it easier to use, and the rule system was overhauled to use custom datetime wrapper objects to elegantly handle early/late toggle times and randomized toggle times.
  - We also implemented sunset/sunrise time calculation using the Astral Python module and adding a settings page that allows the user to select their geographic location.

  - Furthermore, the scheduler that actually runs the rules periodically has been implemented, using the Schedule Python module.
- We have performed integration testing involving using all of the main features of the project (e.g. toggling lights manually, using rules, using groups, etc.) on the provided hardware and all of the functionality is there, save for a few minor bugs.
- We have updated our poster to reflect the current status of the project.
- We have created a slideshow presentation for use as an outline for our final video presentation.

Problems:

- It was fairly minor, but originally, we had issues with the rule system ignoring manual overrides; whenever a user manually toggled a light, the rule system would overwrite it the next time it ran.
  - This was solved by making sure that the overridden state of a light was stored in the database and making rules check if a light is overridden before updating it.

Plans for the coming week:

- Resolve all medium-priority issues. Fixing low-priority issues will be our goal next term, as they are mostly minor usability and maintainability improvements that are not necessary for a beta release.
- Finish poster and slideshow
- Finish video and progress report

Blog Date: 3-8-16

### *P. Winter Week 10 Update*

Progress since last week:

- Continued work on issues for the project, including adding new ones since week 9 and resolving outstanding ones
  - Total issues currently open: 9 (all low-priority)
  - 19 issues closed: 11 medium, 3 high priority
- The full unit test with the screen, mobile device, and relay connected to LEDs was conducted. The tests revealed bugs that were caught and fixed.
  - For instance, we found out that we needed to reverse the logic for generating the TCP commands to make the relay function properly.
- We created our presentation video for the physical device and began shooting video for our PowerPoint presentation
- We finalized our poster after review and submitted the final revision

Problems:

- The lights on the relay were out of sync, we fixed this by logically flipping the TCP command sent to the ESP module
- The override function did not match the requirements document, so Sean revised the override logic to match the requirement.

Plans for the coming week:

- Finish video and progress report

Blog Date: 3-11-16

### *Q. Spring Week 1 Update*

At this point, our project is at a completed stage, but we have identified a few bugs and usability issues that we will spend this term working on.

Progress at start:

- We met to discuss adding new issues to work on this term as enhancements to the final product. Our first priority is to complete two new medium-level issues before the Expo:
  - Create a JS poller so that the lights update on all open pages when the rules or a user changes them.
  - Change the default names for lights so that they're user-friendly.
- Evan has been using the system at home to run his own lights as part of a long term test. The test began on 3/31/16 and will be updated as we get closer to expo.
- We added the Yocto layer for the build and the site itself to the Github repo, and will be updating the Readme to add the proper Yocto version and machine type.

Problems:

- We had no major issues, and since the finished product (minus usability changes and bug fixes) has been demonstrated and is in long-term testing, we don't expect many production issues at this point.

Plans for the coming week:

- Resolve all medium-priority issues.

Blog Date: 4-4-16

*R. Spring Week 2 Update*

Progress this week:

- We met with our client, Victor, to demonstrate the state of our project and receive feedback. Some of the suggestions from the meeting include:
  - Make sure to account for daylight savings time.
  - Make rules run more often than every 60 seconds, especially for demos.
  - Make rules run immediately when rules are changed.
  - Include a way to simulate time running faster, especially for demos. This could potentially allow a whole daily cycle to be simulated in just seconds, showing off the rule system in a timely fashion.
  - Improve our physical demo setup; our current setup involves a breadboard, an extra Arduino for power, and a lot of stray wires; our new plan is to have a small model house made from PCB with the LEDs soldered in and hooked up directly to a 120V to 5V converter. We also plan to have 2 ESP/relay units for the final demo, which Victor has ordered for us.
- We also started work on some of our highest-priority issues, particularly the client-side polling to automatically update light statuses without refreshing, and checks to prevent light statuses from changing if a toggle action failed (such as if the power strip was unplugged or there was wireless interference).

Problems:

- One problem we had was that during the demo with our client, the rules did not seem to trigger when they should have. However, we later realized that we had been manually toggling the lights immediately beforehand, which overrode the rules; the behavior was actually intended. We need to remember this in future demos. The faster time simulation could help with this since even if we override the lights, the override period will end much more quickly.

Plans for the coming week:

- Finish and submit final version of expo poster
- Complete high priority issues
- Prepare model for expo

Blog Date: 4-13-16

*S. Spring Week 3 Update*

Progress this week:

- Added a function to the test branch for the int_bool conversion to make the code involved with clicking the group buttons cleaner.
- Removed the duplicate add_device function, and cleaned up the original by making better default names.
- Started building a rough model house to use at expo using junk PCB.
- Started work on the poller which checks the status of the lights and updates them for the client if they have changed.

Problems:

- Our int_bool conversion branch is still not toggling all of the lights in a group correctly when one of the group control buttons is pressed

Plans for the coming week:

- Debug and fix the problem with the int_bool conversion branch
- Continue working on and possibly finish the PCB model house
- Polish up the poster so that it can be submitted at the end of next week
- Complete high priority issues

Blog Date: 4-15-16

*T. Spring Week 4 Update*

Progress this week:

- Polished the poster and submitted it to the TA for review
- Constructed the first prototype for the house model for use at expo (pictured below)
- Completed changing entry in database for light status from integers to booleans

- Completed the poller to update the switch status across windows.

Problems:

- No major issues encountered this week

Plans for the coming week:

- Complete all medium-level issues
- Complete the next prototype for the house model for expo

Blog Date: 4-24-16

### *U. Spring Week 5 Update*

Progress this week:

- Began work on final report
- Finalized our expo poster
- Submitted our expo poster for printing
- Merged the fix for the Integer to Boolean conversion issue

Problems:

- No major issues encountered this week

Plans for the coming week:

- Complete the video for submission next Friday
- Complete the final report for submission next Friday
- Resolve all medium level issues

Blog Date: 5-6-16

### *V. Spring Week 6 Update*

Progress this week:

- Completed final version of the house/castle we will use at expo for demonstration
- Completed and submitted the final report
- Completed and submitted the video
- Resolved the issue to add reset buttons

Problems:

- No major issues encountered this week

Plans for the coming week:

- Continue fixing and resolving all medium level issues
- Prepare for expo presentation

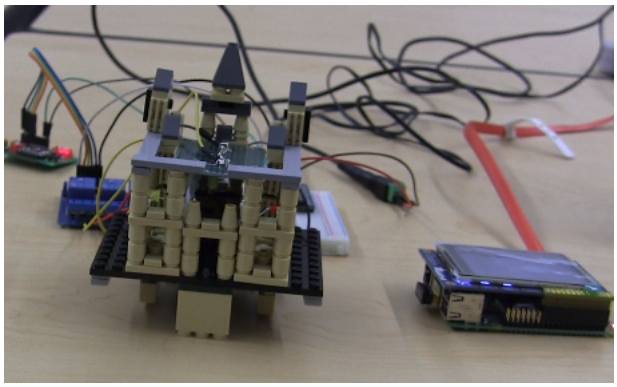


Fig. 2. Our model for the Expo, containing 4 breadboard LEDs

Blog Date: 5-6-16

### *W. Spring Week 7 Update*

Progress this week:

- Bug fixing on JavaScript poller
- Bug fix on footer section
- Completed and submitted the video
- Bug fix on reset button

Problems:

- No major issues encountered this week

Plans for the coming week:

- Continue fixing and resolving all medium level issues
- Create final presentation for Expo
- Collect all materials (Evan!) and make sure that no major technical errors will occur at expo

Blog Date: 5-14-16

### *X. Spring Week 8 Update*

Progress this week:

- Prepared for Expo by practicing our presentation and fixing any outstanding bugs
- Participated in Expo

Problems:

- No major issues encountered this week

Plans for the coming week: Blog Date: 6-5-16

### *Y. Spring Week 9 Update*

Progress this week:

- Received 2nd place award for our performance at the Expo

Problems:

- No major issues encountered this week

Plans for the coming week: Blog Date: 6-5-16

### *Z. Spring Week 10 Update*

Progress this week:

- Assigned parts to complete for the final report

Problems:

- No major issues encountered this week

Plans for the coming week:

- Complete and submit the final report and presentation

Plans for the coming week: Blog Date: 6-5-16

## VII. Poster

This page to be replaced with a full-page color printout of our poster.

# VIII. The Project

## A. System Overview

This project is divided into several sections that can be easily described through a graphic. Below is a block diagram that lays out the basic structure of the system.

As the diagram shows, the lights are connected to the back end of the server through the ESP devices. These devices are

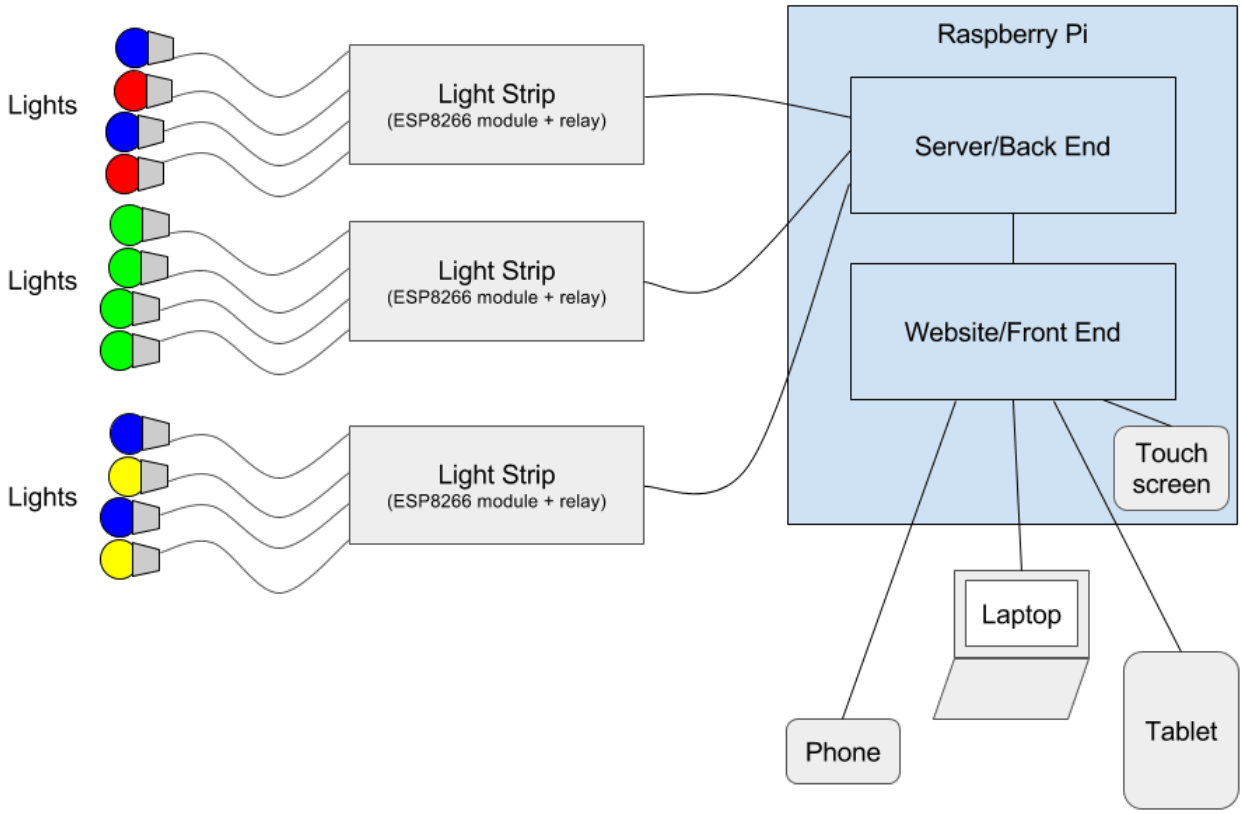



Fig. 3. A block-diagram of the structure of PiLight

configured to wirelessly connect to the server and receive commands to turn the lights on or off. Each ESP device connects to a relay which can serve up to four different lights. The server then hosts a website that is used to control the lights and the schedules. This website can be viewed by any device that is able to connect to a Wi-Fi network, such as a laptop, smartphone, or tablet. It is also viewable from the touchscreen on the Pi itself.

## B. Theory of Operation

PiLight should provide a way to more easily manage the lights in your home and provide an easy way to schedule times for them to turn on and off.

## C. Installation

As part of our installation, we have two modes of putting the image on your Raspberry Pi. These are tailored to users who just want an express installation with no difficulties, and for users who wish to build their own Pilight image from scratch using the Yocto Builder.

### D. Express install

The wiki on Github has a link to a prebuilt image of the Pilight system we had in place at expo. This image is updated using Yocto's autobuilder program Toaster. Once downloaded, it can be installed like any other image for a Raspberry Pi.

*1) Windows:* Windows will use the graphical utility Win32DiskImager to install our image to the SD card.

- Download the image from the Pilight Wiki at https://github.com/rettigs/cs-senior-capstone/wiki
- Insert the SD card into your SD card reader and check which drive letter was assigned. You can easily see the drive letter, such as G:, by looking in the left column of Windows Explorer. You can use the SD card slot if you have one, or a cheap SD adapter in a USB port.
- Download the Win32DiskImager program from http://sourceforge.net/projects/win32diskimager/, which will provide a way to put the image on the SD card
- Extract the executable from the zip file and run the Win32DiskImager utility; you may need to run this as administrator. Right-click on the file, and select Run as administrator.
- Select the image file you extracted earlier.
- Select the drive letter of the SD card in the device box. Be careful to select the correct drive; if you get the wrong one you can destroy the data on your computer's hard disk! If you are using an SD card slot in your computer and can't see the drive in the Win32DiskImager window, try using an external SD adapter.
- Click Write and wait for the write to complete.
- Exit the imager and eject the SD card.

*2) OSX:* On Macs we can use the command-line utilities for creating our image.

- Insert your SD card to an SD card reader
- Use the disk reader program to identify your SD card:

```
diskutil list
```

- Identify the disk (not partition) of your SD card e.g *disk4*, not *disk4s1*
- Unmount the SD card

```
diskutil unmountDisk /dev/disk<disk# from diskutil>
```

where *disk* is your BSD name e.g. *diskutil unmountDisk /dev/disk4*

- Copy the data to the SD card:

```
sudo dd bs=1m if=pilight.img of=/dev/rdisk<disk# from diskutil>
```

where *disk* is your BSD name e.g. */dev/disk4*

  - This may result in a *dd: invalid number '1m'* error if you have GNU coreutils installed. In that case, just change the **bs=1m** to **bs=1M**

*3) Linux:* Our Linux install is similar to OSX, but using Linux tools instead

- Run *df* to find where your SD card is mounted:

```
df -h
```

It will be listed as something like */dev/mmcblk0p1* or */dev/sdd1*. For later steps, you'll need the name without the partition number, which would change the values to */dev/mmcblk0* and *dev/sdd*

- Unmount the drive. For */dev/sdd* it would be:

```
umount /dev/sdd1
```

- Write the data to the SD card. Again for */dev/sdd* it would be:

```
dd bs=4M if=pilight.img of=/dev/sdd
```

- Run sync to flush the write cache

### E. Yocto Build

Using the Yocto autobuilder allows you to build the image from scratch, however it does require some advanced knowledge of the Linux command line. You'll also need about 30 GB of drive space for the build.

- Start by fetching the builder and all the source repositories we'll need

```
mkdir source
cd source
git clone -b dizzy git://git.yoctoproject.org/poky
cd poky
git clone -b dizzy git://git.openembedded.org/meta-openembedded
git clone -b dizzy git://git.yoctoproject.org/meta-intel-iot-middleware
git clone -b dizzy https://github.com/Pilight/meta-pilight.git
```

- Initialize the build environment

```
source oe-init-build-env
```

- Now add the layers to the bblayers file and the configuration options to the local.conf file:

```
echo "BBLAYERS += \"$HOME/source/poky/meta-pilight\"" >> conf/bblayers.conf
echo "BBLAYERS += \"$HOME/source/poky/meta-openembedded/meta-python\"" >> conf/bblayers.conf
echo "BBLAYERS += \"$HOME/source/poky/meta-openembedded/meta-oe\"" >> conf/bblayers.conf
echo "BBLAYERS += \"$HOME/source/poky/meta-openembedded/meta-networking\"" >> conf/bblayers.
    conf
echo "MACHINE = \"raspberrypi2\"" >> conf/local.conf
echo "CORE\_IMAGE\_EXTRA\_INSTALL += \"kernel-dev\"" >> conf/local.conf
echo "CORE\_IMAGE\_EXTRA\_INSTALL += \"pilight\"" >> conf/local.conf
echo "MACHINE\_FEATURES\_BACKFILL\_CONSIDERED += \"rtc"\" >> conf/local.conf
echo "EXTRA\_IMAGE\_FEATURES = \"debug-tweaks package-management dev-pkgs tools-sdk dev-pkgs\""
     >> conf/local.conf
```

- Now build your image. This could take several hours depending on the system:

```
bitbake maker-image
```

- Using Yocto's built-in installer script, we can write the image to the SD card. Assuming it's at */dev/sdd*:

```
sudo poky/scripts/contrib/mkefidisk.sh \
/dev/sdd \
poky/build/tmp/deploy/images/raspberrypi2/pilight-image-raspberrypi2.hddimg \
/dev/sda
```

We now have a bootable SD card

## IX. Usage

*1) Hardware setup:* Setting up the hardware depends on how you want to use the Pilight system. If you want to control a standard light switch, for example, you'll need to replace the light switch itself. Before connecting the existing power line to the relay, cut off the power to the switch for safety. Simply connect the live wire to the relay as shown here:

To connect the ESP8266 module to the relay, you'll need to connect each one of the connections you want to use on the

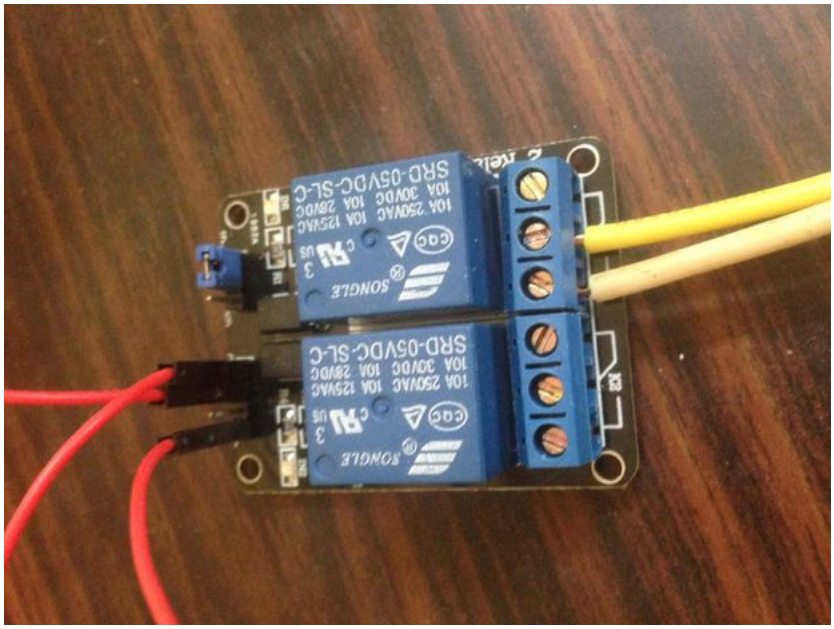



Fig. 4. An example of a two-relay setup.

ESP module to the corresponding relay pin. For instance, connecting pin 1 on the ESP to relay in IN1 will allow you to control the light on IN1 with pin 1 on the ESP module.

Both the relay and the ESP module will need 5V power supplied. You can either run these through a wall outlet with a 5V power supply, or try a long-term battery. Use what works best for your electrical setup. To set up the Raspberry Pi, just provide it with standard wall power over a micro USB cable.

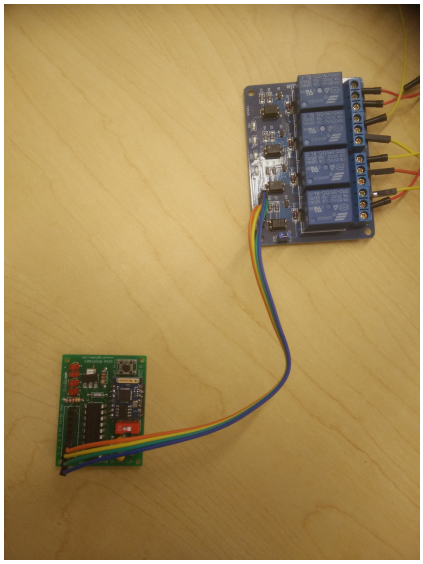


Fig. 5. An example of a connection from the ESP device to a relay board that controls lights.

*2) Software Setup:* Once the system is booted on the SD card, you'll see the web interface appear on the TFT LCD screen. You can also access it by logging onto the "Pilight_Access" wireless access point on a smartphone, tablet or computer. The default password is "RaspberryPi" for connecting. Once you're on, go to **http://192.168.42.1** in your web browser to access the interface.

Assuming the ESP8266 modules are powered on with 5v and connected up to the relay, you can add the devices on the

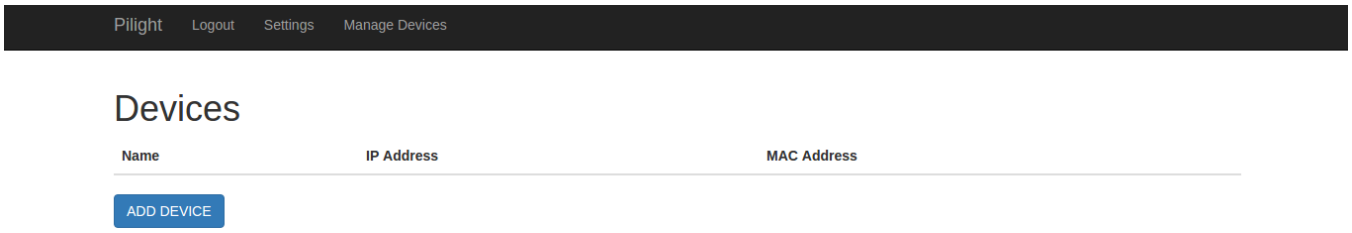



Fig. 6. The devices page before any devices are added.

"Manage Devices" page. Click on the "Add Devices" button to add the ESP8266 devices to the list. Once it's done checking for devices, all your modules should appear.

With these devices added, the front page will automatically populate itself with listings of your new devices. The ESP8266 modules used for this project have 4 GPIOs for controlling the lights, so you'll find 4 entries per device. The front page lets you reorganize these lights into groups for control, and with it you can set intelligent rules for turning these lights on and off

*3) Controlling Lights:* To control groups of lights, you can create an empty group using the "Create Group" button at the bottom of the page. Clicking and dragging lights and groups on the movement icon on the left side of the button will allow you to create groups. You can nest groups within other groups or have lights independent of groups. Clicking the gear icon to the right of the name of a light or group will take you to the configuration page.

## X. New Technology

There were several technologies we used that many people in our group were not familiar with, but learning them was not too great of a challenge.

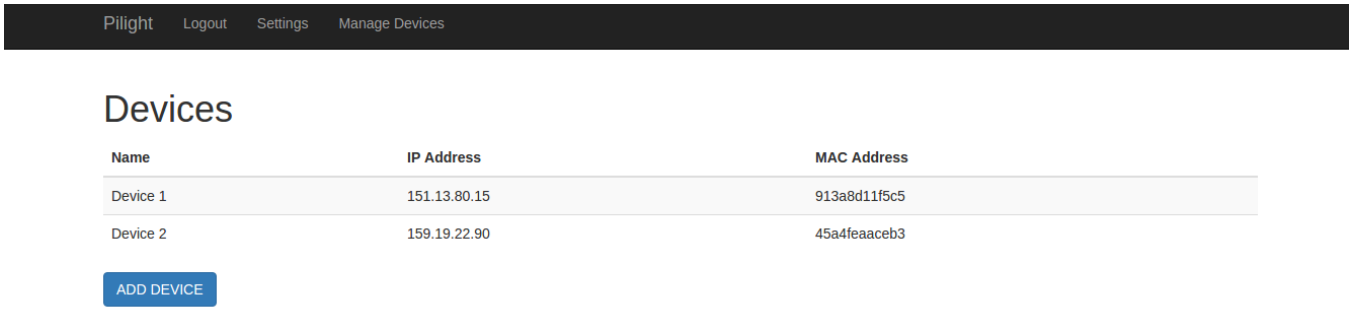



Fig. 7. The devices page after two devices are detected and added.

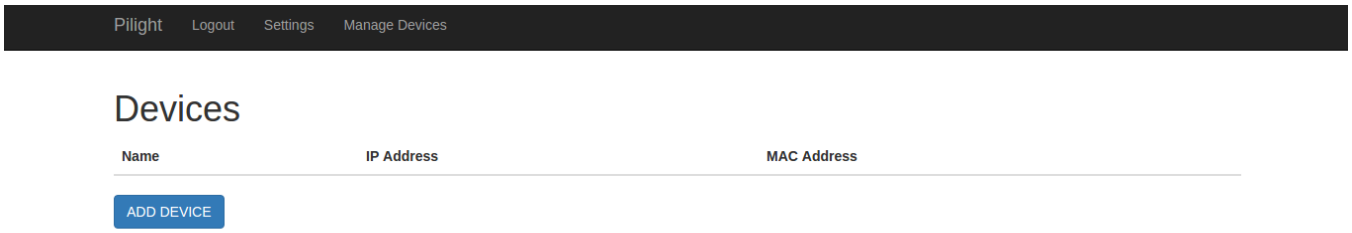



Fig. 8. The advanced page for a light named ”Light 3”

- Python
  While we had all had experience with python before starting on this project, for most of us this was our first large scale project that was almost entirely Python-based. For most questions we had about syntax particulars, we used the standard Python Documentation [9].
- SqlAlchemy
  For database management, we used a Object Relational Mapping (ORM) python library called SqlAlchemy. For information on how to implement it and use it accordingly, we used the SqlAlchemy Docs [10] on the SqlAlchemy Website.
- Flask
  We decided to use Flask as a framework for our website, which ended up being extremely useful, as it allows for a very seamless integration between the front and back ends of our server. As none of us had used Flask before, the Flask Documentation [2] was extremely helpful. Another resource that was useful for learning Flask in a more in-depth sense was The Flask Mega-Tutorial [11].
- Bootstrap
  While some people in the group had used bootstrap before, others were new, and even those who were already familiar with it sometimes needed to look up specifics about how to use it. A resource we found useful was the main Bootstrap Documentation [12], as well as the Bootstrap Switch [13] page for information on how to use the Switches plugin.
- Jquery
  The Jquery API proved very useful when writing the JavaScript for our website. As some of us were new to it and others had used it a decent amount, the Jquery Documentation [14] proved quite useful, especially when using the query functionality.
- Yocto
  Only one of our group members had used the Yocto build environment before, but he was able to provide the advice that we needed to make our project work.
- Firmware
  We needed to write firmware to make the ESP 8266 module do what we want. This would have been very obscure and

hard to do without a reference of some sort, and the one we found useful was a github repository [**?**] that was fairly well organized. We based our firmware off of this code with some slight modifications.

- Google
  As always when learning new technology, google proved to be an invaluable resource, as the ability to search the web was very useful indeed.

## XI. Team Discussion

### A. Malcolm Diller

Throughout the past few terms, this project has been a large portion of the technical work that I have been a part of. One way that it has affected my technical experience is that it has opened my eyes to python which has for the moment become my go-to language. As the project is mostly written in Python, I have learned much about the language in general, and have also learned Flask. Inspired by the effectiveness of Flask in this project, I actually used Flask in a side project that I was doing by myself, which worked out quite well. This was my first real in-depth experience working with HTML and Javascript, and it taught me a lot about the difficulties and usefulness that working with them provides. I also learned a lot more about Git and how to use it effectively in a team environment.

This was also my first experience of working closely as a team with two other people for several months. The three of us worked well together and were able to assign tasks equally and without much effort. Working as a team was made much easier with source control tools like Git, and the use of an instant messaging service. We chose to use an IRC channel as our main chat room for discussing project-related activities, which was helpful in communication. Our frequent meetings allowed us to stay on task and up to date on what each of us were doing with the project.

If we were to do this project over again, there is not much that I would change. Our plan and course of action worked out well, and there were no grand mistakes with how we organized the project, although there were a couple points in the project where we ran a little bit close to the deadlines.

### B. Sean Rettig

Throughout this project, I learned a lot on the technical side, having not had much web development experience in the past. Not only did I get a basic HTML/CSS refresher, but I also learned basic JavaScript/JQuery and became more familiar with Flask and Bootstrap. On the non-technical side, I also was able to improve my project planning and time management skills; we had to schedule regular meetings, keep track of all issues, review each other's PRs, and assign each other tasks to be accomplished by the next meeting. Using this system, we were able to make measurable incremental progress throughout the year and didn't have to scramble to completion in our final weeks. Overall, I have found the most important aspect of working in a team to be communication; we scheduled meetings for at least 2-3 times a week and also kept in constant contact over IRC in our own channel, allowing us to easily leave messages or bounce ideas off of each other between meetings. At the end of each meeting, I always made sure everyone knew what their action items were and what our expectations were for the completion of various tasks. This system made sure that everyone knew what they needed to do and when, and prevented us from accidentally duplicating effort.

If I could do it all over, I don't know that I would change much. For the most part, our original project goals hadn't changed; we were able to establish realistic expectations right from the start, and so we were able to successfully complete our project to our client's satisfaction in the time allotted. Overall, the experience was great; the project itself was interesting, our client was helpful, and we all worked together quite nicely as an organized group.

### C. Evan Steele

My experience with the Pilight project gave me the opportunity to merge my existing technical knowledge with an application that I initially had no interest or experience in. While I had gained significant experience with Raspberry Pi electronic work at my Intel internship before the entire project started, I had never worked on any "practical" applications using the technology yet. Most of my work was at a much lower level, such as writing new SPI device drivers or compatibility testing with $I^2C$ peripherals. Moving toward a project that would require me to step out of my C comfort zone into a higher level was shaky at first, but became a crucial part of my development as a software engineer.

The codebase in Python made up the most of my technical learning on this project. I learned how a web server could be set up in Python and how certain packages, such as Flask, could greatly improve the development process of complex web-centric systems. I learned about optimizing development for mobile devices, since it was the primary way users would be interacting with the Pilight web interface. Additionally, I was able to learn more about the Pi hardware itself and how to manage the limited resources available. Running a traditional web server such as Apache is very intensive on the hardware, but I learned about Flask optimizations that can improve performance. On a side note, I learned about the dangerous power

of Git's *rebase* command the hard way.

My documentation writing skills have greatly improved during this project. Writing endless pages of reports has forced me to improve how I document tasks as the project moves forward, rather than just waiting until the whole project is done. Sean introduced heavy usage of the Github issue tracker, which I initially met with great hostility, but warmed up to over time as it provided an interface to match commits with issue resolutions. My commit messages became clearer and more descriptive with some useful feedback from my group, and I learned how to communicate in an IRC group.

The project taught me how to abandon my ego and accept that I can make code contributions that can later be deleted as the project progresses. I had always resisted change to my own code, even when it was painfully obvious that it needed to be drastically changed or removed. It was difficult to do at first, as having your code criticized feels like a personal attack sometimes. However, I learned to distance myself from these criticisms and see the changes for the good of the end result.

Working for three terms taught me that management is fluid, in that there's never a set leader for the entirety of the project. Each one of us switched management duties as our schedules changed over time, to the point where different periods of time had different group members in charge. Our group worked well together and complimented one another's skills well, so we relied on our collective expertise when it was needed for each leg of the project.
It was emphasized to be honest in answering the question "If you could do it all over, what would you do differently?", yet I feel that I would not change anything about the project. Sean, Malcolm, and I were a perfect team for this project, where each one of us possessed a unique skillset that made the project work in the end. Along with a flexible client, and a project that was right at our difficulty level, gave us an ideal set of conditions for the project that other groups could only dream of.

## XII. Code

The *send_command(light,action)* function sends the TCP command to the ESP8266 module from the Raspberry Pi. It takes two arguments:

- *light* = Light object chosen from the database
- *action* = Boolean for light status (”True” = ON, ”False” = OFF)

```
def send_command(light, action):
    ip = str(light.device.ipaddr)
    tcp_port = 9999

    command = ""
    for i in range(0,4):
        if light.device.lights[i].port == light.port:
            command += '1' if not action else '0'
        else:
            command += '1' if not light.device.lights[i].status else '0'

    if not debug:
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.connect_ex((ip,tcp_port))
            sock.send(command)
        except socket.error:
            return "Connection refused to device: " + ip

    model.Light.query.filter_by(id=light.id).first().status = action
    model.db.session.commit()
```

In the query generator, we have to evaluate the queries from the database to determine what the current state of the light should be. This loop runs every few seconds in a seperate process.

```
# For an explanation of this logic, see here:
        # https://github.com/rettigs/cs-senior-capstone/issues/27#issuecomment-194592403
        if bool(e.status) == previous_rule_state and bool(e.status) != current_rule_state:
            if debug >= 2:
                print "New state:\t{}".format(current_rule_state)

            # Send update to light, or update the group in the database
            if isinstance(e, model.Light):
                send_command(e, current_rule_state)
            else:
                e.status = current_rule_state
        else:
            if debug >= 2:
                print "Not changing state"
```

# REFERENCES


[1] "Yocto Project yocto build server," https://www.electronickits.com/4-channel-auto-roll-rf-relay-board-assembled/.
[2] "Flask flask python microframework," http://flask.pocoo.org/.
[3] "Raspberry Pi embedded devices," https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.
[4] "NGINX nginx web server," https://www.nginx.com/resources/wiki/.
[5] "TFTLCD pi display," http://www.adafruit.com/products/1774.
[6] "ESP8266 wifi module," https://www.tindie.com/products/hbouzas/general-purpose-wifi-control-board-x4/.
[7] "Lua lua scripting language," http://www.lua.org/.
[8] "Relay 10a relay board," https://www.electronickits.com/4-channel-auto-roll-rf-relay-board-assembled/.
[9] "Python python documentation," https://docs.python.org/.
[10] "SqlAlchemy sqlalchemy documentation," http://docs.sqlalchemy.org/.
[11] "Flask Mega the flask mega-tutorial," http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world/.
[12] "Bootstrap bootstrap documentation," http://getbootstrap.com/.
[13] "Bootstrap Switch bootstrap switch documentation," http://bootstrap-switch.org/.
[14] "Jquery jquery documentation," http://api.jquery.com/.