

CAPSTONE 2, FALL 2015

BY: Connor Igo, DAN CARDIN, JOE GORSE, CHRIS HOLTSNIDER, GUNNAR BOEHM

ADVISOR: JOHN KIMAni

Nosferatu

Northeastern university | 360 Huntington Ave, Boston, MA 02115

2015

Table of Contents

[Abstract 1](#_Toc437722420)

[Introduction 3](#_Toc437722421)

[Body 6](#_Toc437722422)

[Hardware 6](#_Toc437722423)

[Software 8](#_Toc437722424)

[Budget 11](#_Toc437722425)

[Conclusion 12](#_Toc437722426)

[References 13](#_Toc437722427)

[Appendix 14](#_Toc437722428)

[Hardware Schematics 14](#_Toc437722429)

[Relay Circuit 14](#_Toc437722430)

[Nosferatu Schematic 14](#_Toc437722431)

[ESP Schematic 15](#_Toc437722432)

# Abstract

Nosferatu improves the lives of its users by adding convenience and removing annoyances from their daily routines. At times it can be inconvenient or even dangerous to stumble around in the dark looking for a light switch. Accidentally leaving the lights on when leaving the house leads to a more expensive electricity bill while leaving them off while on vacation can make a home seem abandoned and a prime target for burglars. Current wireless light bulbs are expensive and will eventually need to be replaced. Nosferatu replaces a basic light switch and box with its own self-contained unit that will add improved functionality without needing to replace each individual bulb. This unit adds a motion sensor as well as remote control through a web interface. The motion sensor allows a user to turn on a light simply by walking into a room while the web interface allows them to turn their lights on or off from anywhere. The web interface allows the user to turn off their lights from bed before going to sleep or turn them on after waking up. While the user is out and wonders whether or not they remembered to turn off their lights they are able to check the status on the page and turn them off if needed. A user can also turn them on if they are not going to be home and want a light on to prevent their home from appearing empty. The web interface also allows a user to schedule times for their lights to turn on or off.

Each of the nodes is a NodeMCU board with a built in ESP Wireless Chip that can act as both an Access Point and Station, if the node has no active connection it will broadcast otherwise it will connect to the station. While broadcasting the node will just host a web page for a user to enter the SSID and Password of the station. Once a user enters their WiFi information, the node will stop broadcasting and connect to their home WiFi. After the node connects to the station, the user is able to search for all nodes on their network and name them to better identify them later. From there the user can add rules to individual nodes, which each can be set to change the status of a node’s light or motion sensing based on whatever criteria the user wants. This can be either based on a schedule, specific times of day, or even the statuses of other nodes. Finally, there is also a button on the node allowing manual control of the node’s status. Manual control of the node from the button or the web interface will automatically disable motion control for the node to prevent motion control from contradicting the user’s intentions.

In order to test Nosferatu, the team created multiple test circuits and connected one full circuit that would be found in a typical node to a desk lamp. A server was hosted locally on a Raspberry Pi. With this setup it was possible to easily test the motion controls as well as the button and control from the web interface. Testing of these features was met with success. Due to time constraints the Nosferatu team was unable to create a 3D printed case to house the circuit. Due to safety and time limitations the prototype was also not put to test in an actual home lighting scenario where the node is replacing a light switch.

# Introduction

You leave your house in the morning, you get to work, and you remember that you left your bedroom light on. You think, “Great. Money wasted”. With the use of the Nosferatu system, this is no longer a problem. Nosferatu will be a direct replacement for your current lighting setup. Each of your conventional light switches will be replaced with a new network-enabled switch, giving it the ability to connect to a central hub in your home, and automate all of your home lighting.

In order to automate a given light switch in your home, that switch will be replaced with a Nosferatu switch. Each of these new “switches” will look similar to a normal light switch plate, except instead of a simply a switch sticking out, there will be a few differences. The most prominent will be a small dome shape, which is the front of a motion sensor. Then there will also be a button, which replaces the manual input that the original switch controlled as well as serving as a manual override to any automated commands.

Any given switch can be manually turned on and off, through the web service, which simply acts as a toggle. Each switch can be configured to turn on/off on a schedule, independent of any other switches. Finally, more complex rules can be configured to turn on/off a switch. Rules include input such as motion sensor data, time of day (such as sunrise and sunset), the status of other lights, and more.

Each switch also has a Wi-Fi chip inside of it, allowing it to network to the other switches through a centralized hub. This hub will be responsible for communicating with all the switches in your network, as well as hosting a web service that is used to do any configuration of the system. Through any Internet browser, any switch can be connected to, and then controlled through a number of inputs.

Currently there are very few existing consumer available solutions for automated lighting. Wi-Fi enabled light bulbs and the Belkin WeMo are the only real competition in the market. Even in the space of patents, where there isn’t necessarily a product yet, there are only a few, and they tend to not directly impact the space that Nosferatu fills.

Wi-Fi enabled light bulbs like the LumenBulb[1] have been the most common to date. Their main drawback, however is that any time a bulb burns out, the whole light needs to be replaced. Given that each light costs many times more than a normal light bulb, this can become expensive very quickly. This also become unnecessarily expensive when a single switch turns on more than a single light bulb, or actually impossible if the bulbs being controlled are more esoteric and not in a size where Wi-Fi light bulbs are sold. Even ignoring all of these issues, they tend to be sold singularly, so any attempt at controlling the lights as a whole system, rather than individually essentially requires a lot of manual setup.

For in wall solutions, the Belkin WeMo[2] is the only real similar product on the market. However, the WeMo is also a large step backwards from Nosferatu. On the hardware side there is not much different, other than the WeMo’s lack of a motion sensor. Belkin does sell a motion sensor that is compatible the WeMo, however it is sold separately (and when paired with the WeMo, costs nearly twice as much as Nosferatu’s prototypes), and does not fit into the wall unit itself, and needs to be placed on something nearby. On the software side, while they do have the ability to control the unit on a schedule, there is much less the WeMo can do with rules; basically limiting it to time based automation. Finally, by default the WeMo is unable to be controlled through anything but their proprietary app. Third parties have found ways around this, and created self-hosted solutions, however Nosferatu is built with this in mind so that out of the box, it can be interacted with through any device that can access the internet.

#### BlockDiagramFigure 1 – Block Diagram

# Body

## Hardware

The hardware of Nosferatu is a fairly straightforward spoke-and-hub setup. There is a Raspberry Pi which acts as the server and the hub, and a series of nodes connected over a network broadcasted by the Pi, which act as the spokes. The server contains the rules, schedules and all other configuration as mentioned in the software section. In its current configuration, the server hosts its own separate Wi-Fi network. This is not ideal from a usability perspective, however there is no hardware or software reason that the server cannot be piggybacked into the user’s network directly or even be remotely hosted. In an ideal household scenario, there would be as many nodes as there are light switches with the idea that every light or wall-driven appliance in the house could be able to be controlled by Nosferatu.

The nodes themselves are made up of a few parts, with each of the schematics included later in this document. The central piece of the node is the ESP8266 NodeMCU Arduino that controls everything within the box via GPIOs. The motion sensor and button work in tandem with the Arduino to drive the relay to act as a switch. We tie the motion sensor to a pull up which gets driven low when motion is detected. The GPIO will sense this and respond according to the rules and schedules set and programmed to the chip. The button is also connected to a GPIO that ties it high when the button is pressed, which we use as a hardware override on the current state of the light. The Arduino also has an ESP8266 Wi-Fi module physically attached to it that receives instructions from the pi to then relay on to the rest of the box. It was convenient to have the Wi-Fi module directly attached and built into the framework of the Arduino. The power proved the most difficult task to accomplish, as working with mains is never an easy task. Stepping down a stripped USB 5V 500ma adapter and fitting it to the 120VAC lines, cutting and soldering much thicker wires and providing power for a relay that required that max amount of voltage we could give it were just some of the tasks we had to overcome to power everything in the box correctly. The ‘transformer’ was a USB plug that would normally charge a smart phone of some sort or anything else powered by USB. It stepped down the 120AC to 5V that powered the Arduino. From USB to microUSB, the 5V rail on the Arduino through a small breadboard provided power to the rest of the circuit: the motion sensor, the relay (although we had to get creative in wiring it,) and the Wi-Fi module. The relay was hooked up differently than the way the spec originally called for, but ended up working just as intended. The power comes in through a BJT that when in active region, allows current to flow through the coil that closes to complete the light circuit. We need a fly back diode here to regulate the spike in voltage that occurs when current is no longer flowing through the coil.

In many ways, the hardware that is in this iteration of the Nosferatu nodes are not how they would be in a final product. The whole enclosed device would be in a small box, similar in size to a wall switch; with an appearance meant to mimic the switches that most people are used to. Also, much or all of the circuitry that is currently in each node would be put onto a single integrated circuit. Not only would this be much cheaper than individually sourced components, but it would also result in decreased power consumption.

To measure success for the hardware, our only real requirement was that it save the user electricity. Luckily, even in it’s current iteration, with a current standby power draw of 0.09 amperes at 5 volts, or 0.18 amps while power a light. This means that in order for a node to use less electricity than it saves, a node only needs to save 15 minutes per light per day. If a single light were to be turned off for an hour a day when it would otherwise have been on, then it could save upwards of $50 per year. This is not even considering power improvements that could be had through utilizing the low power modes in the ESP8266 or ATMega chips, or a more power efficient relay. A savings of 15 minutes is actually significantly less than what we believe a typical user would end up saving, so in this regard it was certainly adequate.

## Software

The software that powers Nosferatu is built into a few layers that all interoperate with each-other. The top layer, and the only layer that a user can actually directly interact with is the front-end web-app. The middle layer is the actual server software that actually performs all of the functionality that the web-app offers. The bottom layer consists of the software that directly interacts with the database, where we save all the users, nodes, rules, and schedules so that they are saved in the case of reboots. Finally, the node itself runs firmware which directly modifies the state of the hardware components in Nosferatu to actually turn the lights and sensors on and off.

Starting with the front-end, we decided to write the front end in the form of a web-app to reach the largest number of use cases while still being useable. By creating a web-app, a user is able to control their lights from their local computer, tablet, phone, or even a remote device that queries our REST API. As a web-app the base technologies that it must be built on top of are HTML (for layout), CSS (for styling), and JavaScript (for dynamic functionality). However, we used SCSS for styling and CoffeeScript for dynamic functionality, both of which can be compiled into CSS and JavaScript respectively. We decided on this route because the maintainability of the technologies we used is ultimately much better and easier to use than their alternatives. We actually used Angular as well, which is a framework for creating dynamic content on web-apps and displaying them to the user. It also includes a library for asynchronous HTTP request, of which we made extensive use. Almost every action that a user can take on the web-app such as searching for a node, turning it on and off, or creating rules is not directly performed through interface. When a user tries to turn a light on for example, it kicks off an asynchronous HTTP request which is sent to the REST API as a request to the server software to create the rule. This is actually a key point because turning the light on requires both modifications to the database and wireless communication with a node, both of which can take arbitrarily long. If we used blocking requests, the entire UI would be unresponsive for the duration of the request, which would make for a bad user experience. Finally, once the server responds, the front-end will update the UI to show that the light has been turned on.

The server software essentially acts as the brains of Nosferatu. It coordinates communication between the front-end, the database, and the nodes themselves. Given the number of responsibilities it has, it is necessarily built around a number of technologies. The server itself is written in Python using the Flask framework for creating web-apps. This allowed us to create a REST API that the front end could query to communicate. We had originally been planning to use Celery (a task queuing application) in order to make the REST API asynchronous, however we decided to go a different route for a few reasons. It turns out that Celery was not ideal for our use case because it requires a separate process to be running, requires a restart on each code change (which hindered development), and finally because it was somewhat unnecessary. It turns out that the asynchronous functionality that Angular in the front-end gives us overlapped with Celery and made the whole system overly complicated. Removing Celery actually vastly simplified the code and improved performance. We then used PostgreSQL as a backend database to store all the users, nodes and rules. We had originally planned to use SQLite for this, however PostgreSQL is a much more performant for the data we were storing. Finally, we had planned to use Cron (a scheduling framework built into Linux) for the light schedules but switched instead, to a separate in-memory SQLite database that could run an action on an arbitrarily complex schedule. Cron would have been far too complex to dynamically create and delete rules whenever the server starts and stops, in addition to intrinsically tying the server software to Linux. The SQLite database in this particular case actually turned out to be ideal because SQLite databases do not need to be stored on disk like other databases. So every time the server starts, it creates an in-memory table of schedules stored in the actual PostgreSQL database so that they can be run dynamically.

The firmware running on each node was developed in the Arduino IDE environment, which is a C/C++ based environment. The advantage the NodeMCU board brings over a different Arduino is the use of built in user libraries for wireless communications, which allows for a lightweight and diverse setup. Using TCP connections, the communication between the board and server were able to complete in less than 5ms timeframes, allowing for fast processing of commands and updates. The motion sensor, relay and LED were all controlled via digital pins which is very fast and using simple logic the controls for these components was completed. The board also allows access to a file system in its onboard memory through C file stream operators, which allows for us to save any information or extra forms in separate files from the core code.

To measure success for the software we wanted to be: readily available in real-world circumstances, be able to quickly change the state of the lights, have advanced features like scheduling and rules, and use minimal bandwidth. The fact that we created a web-app that can be accessed through a number of different views including mobile, and remotely through a web browser serves to account for being readily accessible. For quickness, we had initially been limited to a somewhat random latency that reached upwards of five seconds. However, by the end of the project we managed to consistently reduce that time to be less than a quarter of a second. We also succeeded with features because we were able to create features for rules and schedules, as well as extra features like advanced configurability, and dynamic rules. To minimize bandwidth, we only communicate while a user has the web-app open with saved nodes, but when it is communicating we issue requests once per second. After measuring the network traffic of our page loaded with three nodes over the course of 10 minutes, we were able to calculate that it would take our web-app 72 minutes to reach the same amount of traffic as loading the front page of reddit.com. Given that it’s likely that someone would only be actually viewing the web-app for a few minutes at a time, this was a resounding success.

# Budget

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Units | Cost per unit | Total cost |
| FTDI Basic Breakout | 1 | 14.95 | 14.95 |
| Arduino Pro Mini | 4 | 9.95 | 39.8 |
| Relay Control Kit | 4 | 7.95 | 31.8 |
| Motion Sensor | 4 | 9.95 | 39.8 |
| Wifi Module | 9 | 6.95 | 62.55 |
| Raspberry Pi | 1 | 41.64 | 41.64 |
| Wifi Adapter | 1 | 19.99 | 19.99 |
| Pi Power Supply | 1 | 6.99 | 6.99 |
| Micro SD Card w/ Reader | 1 | 11.99 | 11.99 |
| Apple Transformer | 4 | 5.75 | 23 |
| Pin Headers | 2 | 2.06 | 4.12 |
| Arduino Pro Mini Serial Adapter | 1 | 7.1 | 7.1 |
| IRF4905 MOSFET | 3 | 7.61 | 22.83 |
| IRLB3034 MOSFET | 3 | 5.21 | 15.63 |
| Pack of 12 Tact Switch | 1 | 5 | 5 |
| Raspberry Pi Case | 1 | 6.95 | 6.95 |
| Solderless Breadboard | 1 | 4.7 | 4.7 |
| 200mm Male to Female Cables | 2 | 1.99 | 3.98 |
| Experiment Breadboard w/ Wires | 1 | 7.48 | 7.48 |
| NodeMCU Devkit | 3 | 15.13 | 45.39 |
| Micro-USB to USB Cable | 2 | 4.69 | 9.38 |
| NodeMCU | 1 | 15.12 | 15.12 |
|  |  |  |  |
| Total |  |  | 440.19 |
| Server BOM Cost |  |  | 89.73 |
| Estimated BOM Cost per Node |  |  | 36.66 |

We originally planned to buy enough components to build four nodes, but throughout development we ended up needing to expand our original budget. We needed to purchase additional Wi-Fi modules as during the soldering process they would occasionally stop working. We made the decision to switch from the Arduino Pro Mini to the NodeMCU as the Arduino was giving us an unreliable connection. We purchased two different types of MOSFETS so we would be able to test which one was better for our node. Various parts including the headers and wires. The Raspberry Pi and accessories for it are one-time costs and do not contribute to each the node cost.

For an actual product, these costs are almost completely unrepresentative of what they could be. For each node, given that the cost of individually sourcing each component was only about $36.66, printing ICs and mass production vastly reduce that cost, even including an added cost for and enclosure. As for the server, in a productized version the server would not even need to cost anything to a user. A remotely hosted solution server or a locally hosted always-on machine that the user already owns could easily server as free alternatives to the Raspberry Pi. Even if we sold a separate server, it would likely not be an actual Raspberry Pi, which because of all the required extra components led something a fair amount more expensive than necessary.

# Conclusion

Nosferatu will replace your current lighting system and improve upon it in every way. Nosferatu adds convenience and lowers costs over your default lighting setup and over its competitors. With Nosferatu you no longer need to worry about whether or not you turned off your lights. You can check and change the status of any of your lights from anywhere you have an internet connection. After getting in bed and realizing you forgot to turn off the lights or after waking up and not quite wanting to get up yet you can use Nosferatu to turn off or on the lights without needing to get up. You can also set schedules to automatically change your lights at certain times of the day or after a certain period of inactivity from the motion sensor. These features not only add the convenience of not needing to remember to turn off your lights, but also save you money by turning them off when they aren’t in use.

Nosferatu does have room for improvement. There are multiple features that could be added with continued development. The web UI could use improvement with its design and could have added functionality. There is room to add additional rules and rule types and add space for third party integration through features such as a temperature sensor to allow connection to a thermostat. By adding a deep sleep mode to various components in the node the power costs could be lowered beyond where they already are.

Compared to its competitors Nosferatu is both cheaper and easier to use. Each unit even with the additional hub is cheaper when compared to the WeMo, even without a motion sensor, and Nosferatu can be used from any device with a web browser. Unlike simply buying a smart light bulb Nosferatu can control an entire room of lights with one unit and does not need to be replaced if a bulb burns out. Nosferatu also improves over its competitors by having more options for control with the rules that govern turning on and off the lights which can only be improved with future development. These features again make Nosferatu a cheaper and more convenient alternative for home lighting control.

# References

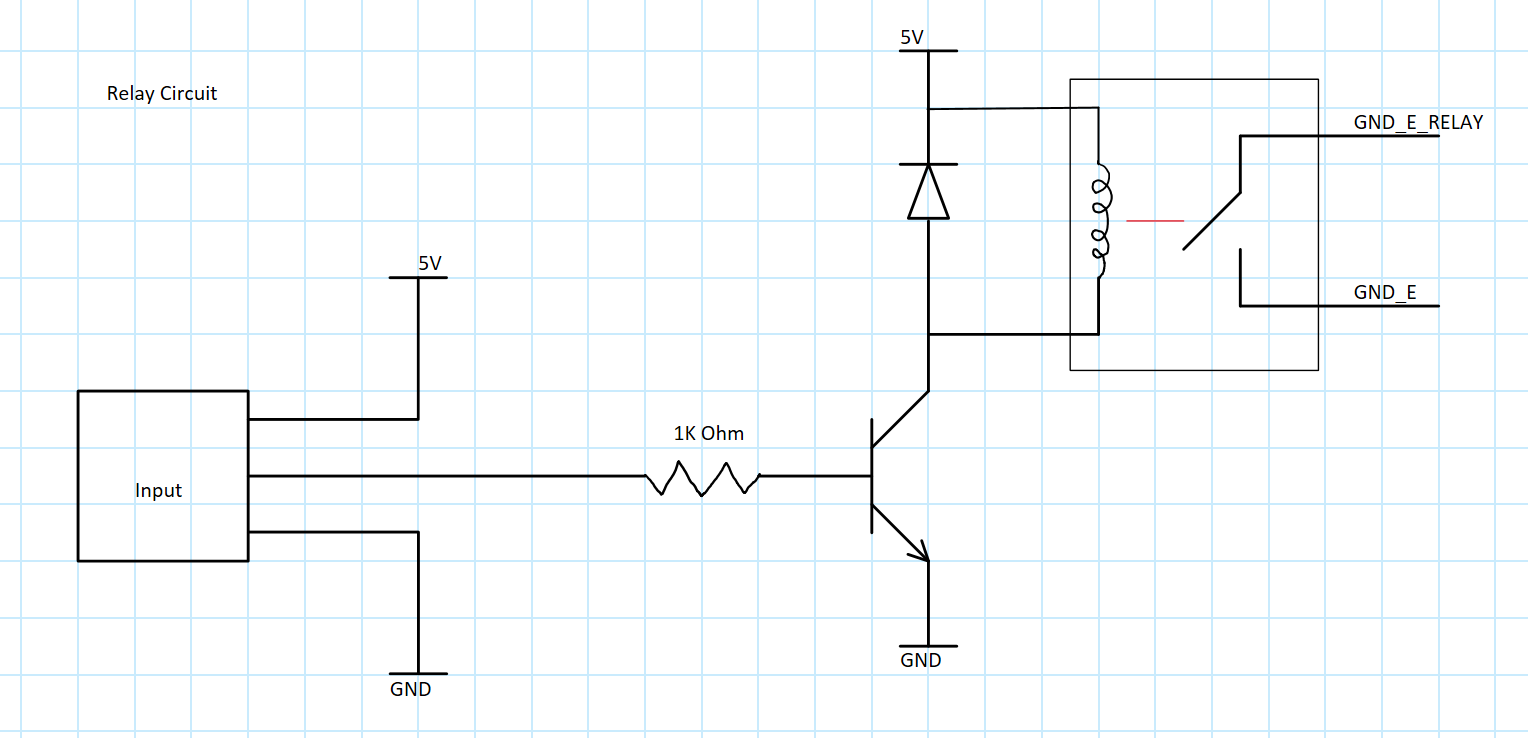
[1] <http://www.lumenbulb.net/>

[2] <http://www.belkin.com/us/Products/home-automation/c/wemo-home-automation/>

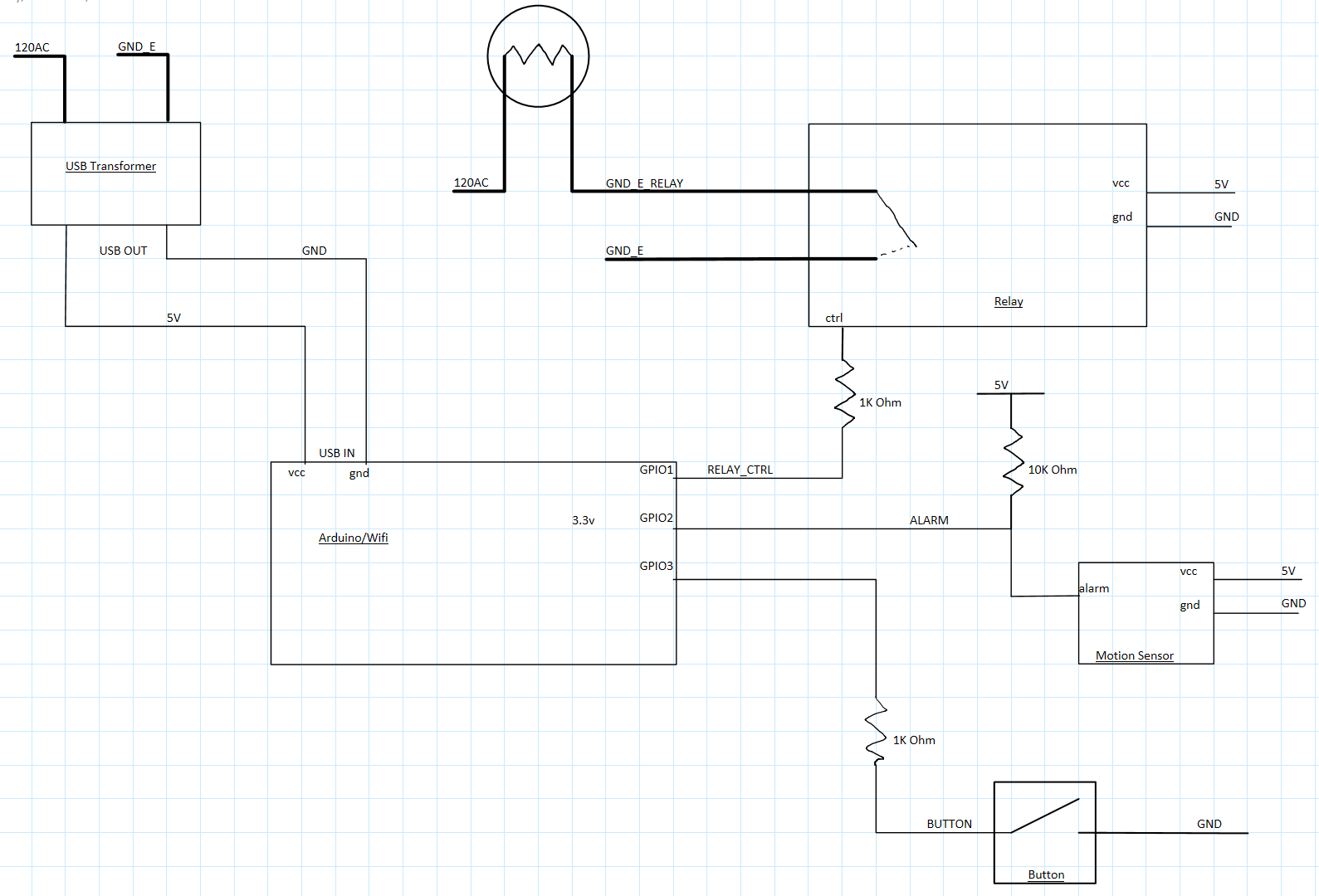
# Appendix

## Hardware Schematics

### Relay Circuit



### Nosferatu Schematic



### ESP Schematic

