Nosferatu

Capstone 1, Summer 2 2015

By: Connor Igo, Joe Gorse, Dan Cardin, Gunnar Boehm, Chris Holtsnider

Advisor: John Kimani





Table of Contents

Abstract 3

Introduction 3

Problem Formulation / Impact 4

Hardware Design 5

Software Design 7

Tasks 8

Conclusion 9

Appendices 10

References 11

# Abstract

Nosferatu is an energy efficient, convenient new way to control your home lighting. Connecting to your current home network Nosferatu will be able to automatically control the lights in your house. Bringing Wi-Fi and motion sensors to your light switches allows you to set schedules and rules for managing your home lighting, saving money, energy and time.

# Introduction

Nosferatu will be a direct replacement for your current lighting setup, going directly inside the wall and replacing the existing switch. Using each switch as an endpoint they will be networked with the central hub that will be responsible for controlling all switches. The hub will be accessible through any Internet browser, which will support remote control of individual switches, rules, and scheduling. Each switch will also have a manual override button, a motion sensor, as well as a status LED.

# Problem Formulation / Impact

Currently there are very few existing consumer available solutions for automated lighting. Wi-Fi light bulbs are the most common to date, with the main drawback being that every time a bulb needs to be replaced a new module needs to be purchased and re-setup, as well as needing proprietary applications to run properly. Light bulbs also lack scheduling and rules that Nosferatu can provide. For in wall solutions Belkin has a similar product that is exclusive to its WeMo in home setup. We have more features and are cooler, and we hope ours costs less.

# Hardware Design

The key design aspect we took into consideration is the connectivity between our endpoints and server. The two main considerations were utilizing a Bluetooth mesh network, with endpoints interconnecting between each other starting from a centralized hub. Compared to a Wi-Fi setup where each point directly connects to the hub, which is more common in home networking and generally an easier setup. Though utilizing the Bluetooth mesh could provide a lower power alternative with a larger potential range, the current kits and interfaces for developing mesh networks requires Bluetooth LE, which is a newer technology and the required hardware wouldn’t meet our physical requirements for the design.

After deciding to use Wi-Fi to connect our endpoints, the next biggest issue faced is powering each endpoint. The easiest would be to use batteries to power the boards, but we would lose out on the convenience this device should have by needing to replace batteries whenever they die. We decided to put a transformer in to our design, which limits the product to be used in newer lighting systems that have a ground, hot and common wire. In comparison to other products similar to our design they have the same wiring requirement for the switch, which is the most common in home lighting and is more future proof.

We looked into a few different choices for where and how to host our server. Our initial discussion was whether to host the server locally, on a small local server, or to host remotely using a cloud service such as Amazon Web Services. We decided that hosting the server locally would give us more flexibility in software design. We then settled on either a Raspberry Pi or a BeagleBone Black to be our local server. Comparing price, processing power, I/O connectivity, community development support, and our group members’ familiarity with each, we chose to use a second generation Raspberry Pi to host our web server and act as our hub.

Safety is the biggest concern we have as a general design aspect, due to adding a device that goes in to a wall and will be running off live wires. Our relay was a choice between an SPDT relay and a Solid State relay, the SPDT use much less power, but also are designed to fail “open” which means if it does fail the circuit will be broken and avoid an electrical fire, unlike a Solid State relay which can fail closed and cause many problems. Our transformer is also a key aspect in safety; it is responsible for taking a 120V input and converting it to the necessary 5V to power the entire unit, while also not generating a lot of heat. We chose an existing product that we will take apart and convert because it is much safer than trying to design a transformer-less system to convert high voltage AC input to DC. Heat is the hardest factor to determine in planning, though each endpoint is expected to only be “active” for a very short amount of time, as of now the heat generated seems benign, but in the event it becomes a problem we will put a small vent in the cover plating of our switch to disperse heat outside of the wall.

# Software Design

There are two main software blocks in our system; the firmware running on the endpoint, and the server running on our hub. The server will have three software layers; frontend (HTML, JavaScript), middleware (Python), and backend (database, Linux system calls). The node firmware will be written in C.

While the software for the nodes is almost entirely direct hardware interaction, the software for the server is mostly http requests. The general flow of data will start with a user logging into the web service. After being authenticated, they will see the main page of the web application.

Upon connection, the frontend will kick off an asynchronous background task on a timer to send requests to the nodes for status updates. This means the user will get a close to real-time view of what all of the nodes are doing. From there, a user can do any number of actions, such as controlling the lights, rules and schedules. Any given action a user can do will send off an asynchronous JavaScript request which either will make modifications to the database (or Cron job for schedules), or request updated data from the server to update the UI elements on the webpage.

For example, if a user wants to add a new rule, they'll fill out the information on the webpage, and submit it, which will send the request to the middleware running Python, which will then add the new rule to the database. If that rule happens to immediately update the state of the node, a request will also be sent to the node to change its state. Finally this will kick off a response from the node, which will communicate back to the server with its updated status.

The firmware running on our nodes will have three core functional tasks: identifying itself to register with a hub, changing the state of the relay to turn its light on/off, and responding to ping requests with the current status of the relay (and therefore the light).

# Tasks

Node Assembly: Chris

Raspberry Pi Setup: Joe

Create Web Server: Dan

Flask Setup: Gunnar

Hub -> Microcontroller Connection: Joe

Microcontroller -> Hub Connection: Connor

Relay Interface (Software): Connor

Motion Sensor Logic: Joe

Web Server Logic (Basic): Dan

Expand UI: Gunnar

Design Node Enclosure: Dan/Gunnar

Setup Test Environment: Everyone

Running Tests in Lab: Everyone

Advanced Features: Everyone

Presentation Prep: Everyone

# Conclusion

# Appendices

# References

Patents:

<https://www.google.com/patents/US6531836>

<https://www.google.com/patents/US5086385>

WeMo:

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

LumenBulb:

<http://www.lumenbulb.net/>