Building an Open Source IoT Garage Controller

|  |  |
| --- | --- |
| Daniel A. Zajac  Department of Electrical and Computer Engineering  University of Michigan  danzajac@umich.edu | Jackson Harmer  Department of Electrical and Computer Engineering  University of Michigan  jharmer@umich.edu |

*Abstract*— Internet connected technologies have become mainstays of the modern household. From Internet of Things (IoT) connected coffee makers to sophisticated adaptive climate control systems, inexpensive wireless technology and popularity of voice activated digital assistants has enabled a wide variety of connected tech. Most of this technology remains closed source however. Many of the more popular connected technologies, such as Nest or ecoBee, rely on closed source protocols and cloud service backends. What happens when these companies go out of business or shut down older services? VueZone shut down its services leaving owners with severely crippled expensive IoT cameras. [1] This paper focuses on building a proof of concept IoT connected garage door real time controller and fully published interface and source: The GarageRTC! This paper reviews the features, design, and implementation of a reference architecture built on an ESP32 microcontroller built on FreeRTOS software as well as performance and possible future improvements.

Keywords—GarageRTC, IoT, Embedded, FreeRTOS, ESP32, NodeMCU, RTOS

# Introduction

As IoT become more common place in our daily lives we become increasingly dependent on the connected services that support those systems. Some of the more popular devices, such as Nest thermostat, are the product of small teams operating as a startup. As these teams grow they are sometimes bought out by larger corporations interested in entering into the IoT market. For some less fortunate start‑ups, they never break through into profitability and slowly descend into obscurity.

What happens to the devices that depend on the cloud services previously maintained and supported by those teams? Sometimes they can continue to function but with crippled or limited performance. Often, they are rendered useless as registration and web-based configuration tools become obsolete. [2] For inexpensive devices, they can be cannibalized for parts or simply disposed of. For more expensive equipment, such as the Juicero, the founding company dissolved leaving many users with useless $400 IoT juice machines. [3]

The only guaranteed way to ensure these devices can be indefinitely supported is to implement an open source methodology including web APIs. Preferably, a completely open source hardware and software reference design would be created and published. Startups wishing to develop derivative works, could leverage the technology could then extend the designs. If the startup dissolves or can no longer support their design, if compatibility with the reference design was maintained, an internet user community would be able to continue support. This concept has been demonstrated profitable and sustainable in large software efforts such as IBM supported Red Hat Linux. [4]

To prove out this concept, the research team designed and built a reference design for an IoT connected Garage real time system, or simply, the GarageRTC.

The GarageRTC is an automation system for a consumer garage connected as an IoT device, Figure 1. The system makes the status of garage available to the user connected remotely through a web-based interface or locally via a display and control panel. Inside the garage, the system is connected to the garage door opener, a garage light, and an alarm. The system uses sensors to detect door position, accumulation of carbon monoxide, temperature, and objects in the path of the door. The user can check the status or manipulate the controls from the control panel or web interface.

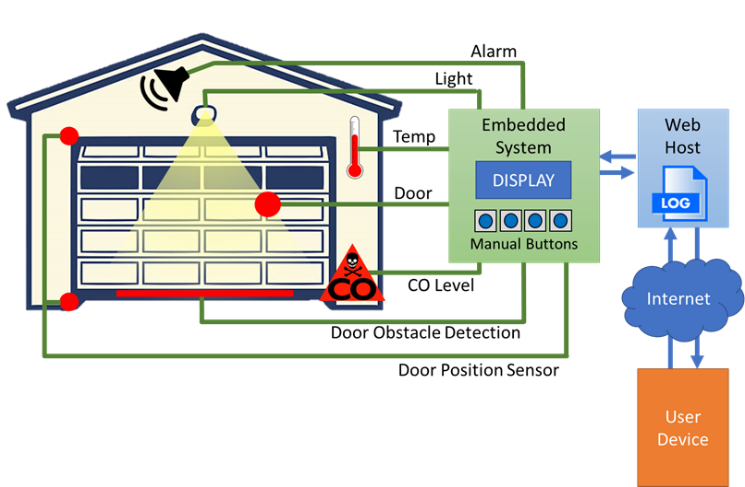


Figure 1. GarageRTC Concept

This paper documents the design process, development, implementation, and testing of a WiFi connected garage monitoring system. It implements a basic JSON based API and includes a reference web application that can easily be hosted in a personal cloud or Linux based development board.

## Organization of this Paper

This paper is organized into ten parts:

* **Introduction** – Introduces the motivations and organization of this project.
* **Objectives and Functional Description** – Outlines the high-level objectives for the project and reviews the design process employed.
* **Requirements** – Reviews the high-level requirements the GarageRTC was designed to meet.
* **High Level Design** – Describes the process used to decompose and partition the system and generate high level designs and test plans.
* **Detail Design** – Discussion on the detailed design decisions including selection of hardware, software, and web application.
* **Testing** – Review of the tests and stages used to ensure proper functioning of the system.
* **Accomplishments** – Review of the objectives met, limitations of the current design, possible improvements, and lessons learned.
* **Comparison of Commercial Offerings** – A comparative discussion about similar products commercially available.
* **Conclusions and Closing Remarks**
* **References**

# Objective and Functional Description

The objective of the GarageRTC project was to develop an open source real-time embedded system that can be integrated with an existing opener system. The GarageRTC collects data from obstruction detectors, limit switches, carbon monoxide (CO) concentration, and temperature. It maintains a connection to a basic web server. From the control panel or web interface, the user can observe the following:

* Status of the garage door (open, moving, or closed)
* Status of the light (on or off)
* Level of carbon monoxide
* Temperature in the garage
* Errors or Messages

From either interface, the user can execute commands including:

* Open or Close the door
* Stop door movement
* Turn on/off the Light
* Silence the alarm

The system monitors the environmental conditions of the garage and can sound an alarm if the any of the following issues are detected:

* Carbon Monoxide level too high
* Low temperature/freezing risk
* High temperature/possible fire
* Movement of the door interrupted by an object

This system provides the user an easy way to operate and monitor and control the status of their garage. The scope of this effort it to create the garage interface control and status architecture and basic web interface.

## Design Process

This project was developed using a waterfall software development lifecycle. Adherence to a classic waterfall model was attempted, but during implementation it was discovered that the original display selected was not functioning properly. The display originally selected, a .72 in OLED, had partially implemented drivers and the display was smaller and more difficult to read than expected. The team returned to the design phase briefly to select another display that met the requirements and had better driver support. The waterfall software development lifecycle with the feedback into the design phase is illustrated in Figure 2.

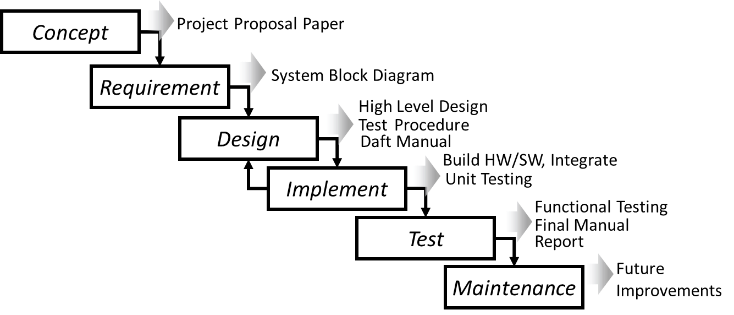


Figure 2. Waterfall Software Development Lifecycle

The maintenance phase is not applicable in this instance as this paper represents the final stage of this effort. The team did make several recommendations for follow on improvements and the source is available in the public GitHub repository. The proposed improvements can be found in section VII of this paper.

# Requirements

The high-level requirements were developed as part of the initial project proposal paper during the concept portion of the design process. Functionally the system needed to have the following interfaces:

* Electrical interface with a garage door opener
* Electrical interface with a mains voltage light
* Web interface for remote operation
* CO and temperature monitoring
* Limit switches and obstacle detection
* Local display and buttons

Design detail within the system was mostly left up to the design activity but was loosely defined as a conventional web server interfaced to the embedded system. Further detail was provided on the specific functions of each button and the display content. The web display would mirror the local display and provide buttons to interact with the embedded system as if the user was local. Response timing goals was also provided and are summarized in Table 1.

Table 1. Response Timing Goals

| **Parameter** | **Timing Requirement** |
| --- | --- |
| Local Display | Every 500 mS or better |
| Obstacle Detection | Toggle opener within 150 mS from detection |
| Limit Switch | Toggle opener within 150 mS from detection |
| Door movement | Toggle opener within 300 mS from press |
| Light | Latch light within 300 mS from press |
| Idle Post to Server | Post to the server every 5000 mS |
| Event push to server | Post to the server within 500 mS from event. |
| Check Server | Check remote messages every 1000 mS |

From the timing goals and requirements, a system block diagram, Figure 3, was composed to illustrate the major partitioning of the system and the system inputs and outputs. The block diagram was annotated to show the type of signal (resistive, digital, voltage, UDP data, etc…) and the response times for outputs.

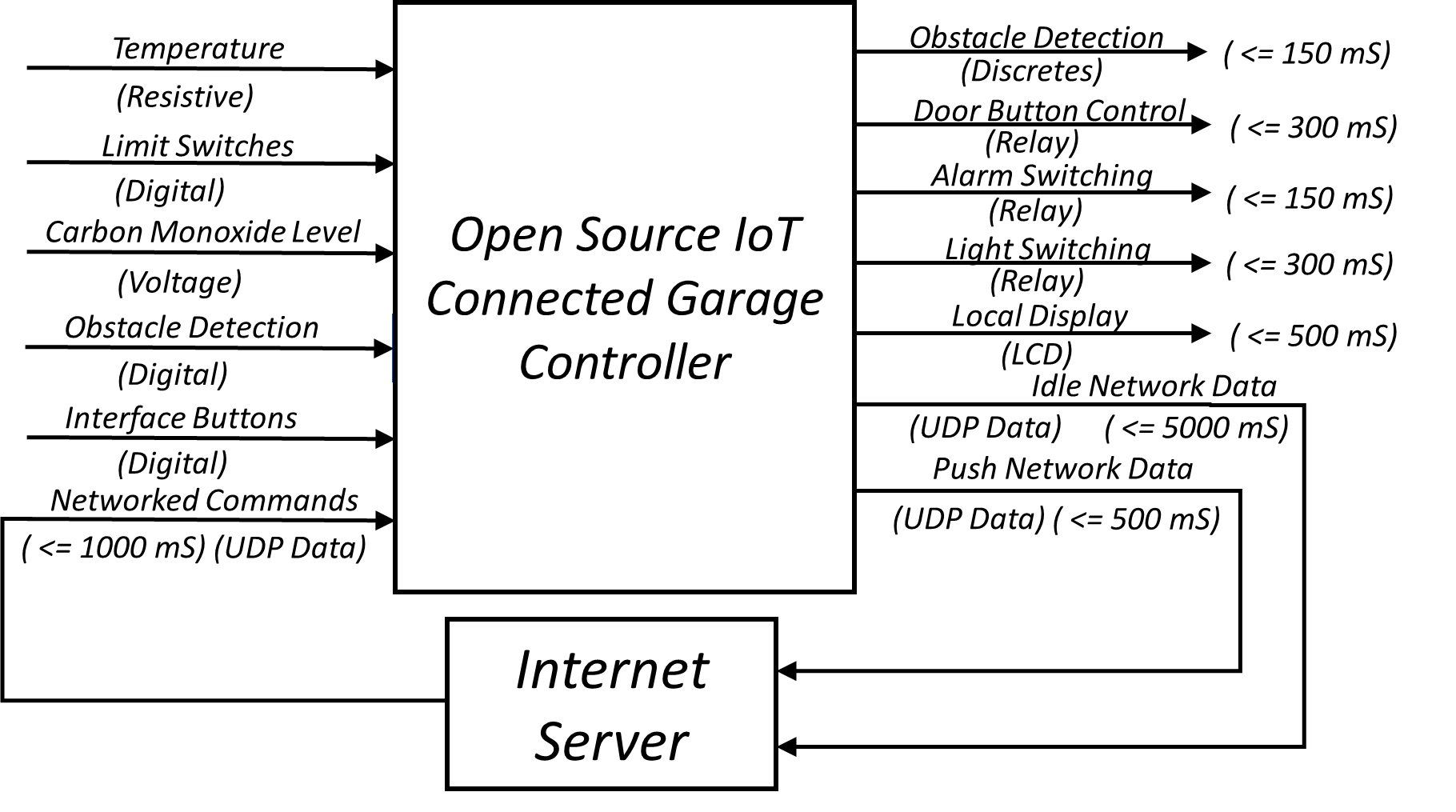


Figure 3. GarageRTC System Block Diagram

# High Level Design

From the block diagram, the major partitioning of the system could be determined. The functions of the internet server were could be developed independently from the embedded system assuming good interface constraints could be established between the two. Since this project team was not regionally co-located, this made for a good way of separation of efforts. Figure 4 shows the decomposition of functions and responsibilities of entire system. This section reviews the activities performed as part of the high-level design for the GarageRTC project.

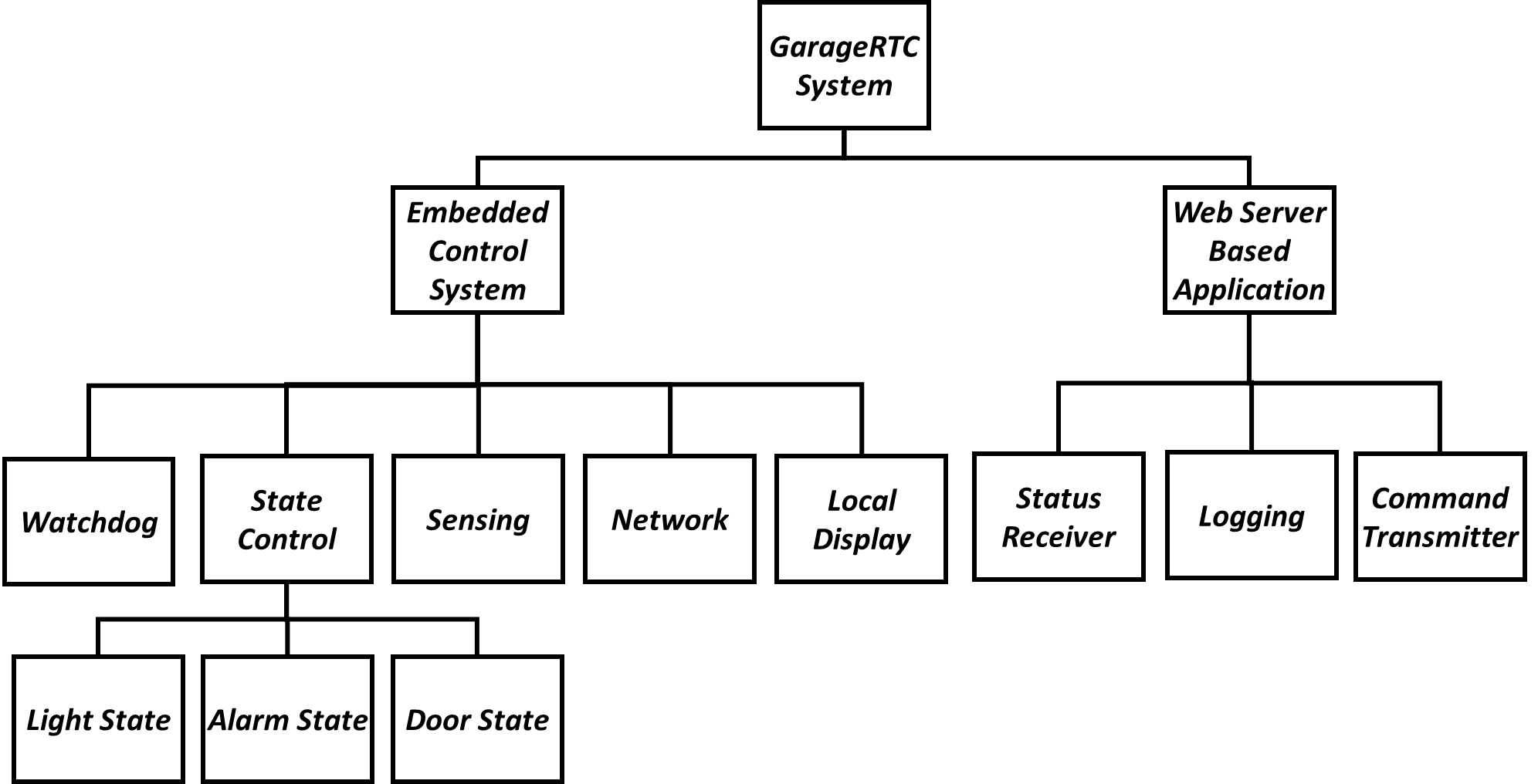


Figure 4. GarageRTC System Organization

## Interface specification

## Languages

## Major Hardware

## OS

## Documentation

Draft Manual

Test Plan

## Embedded System

For the embedded system, a real-time OS was necessary to provide predictable response to inputs while simultaneously executing multiple tasks in a timely fashion. While scratch development of a RTOS was considered, ultimately the FreeRTOS project was selected. FreeRTOS implements a real-time kernel and schedulers targeted as low resource microcontrollers. The project is professionally developed and is available for free use in commercial embedded systems. It supports a wide variety of hardware and offers excellent documentation and, most importantly, is freely available including source. [5]

## Web Application

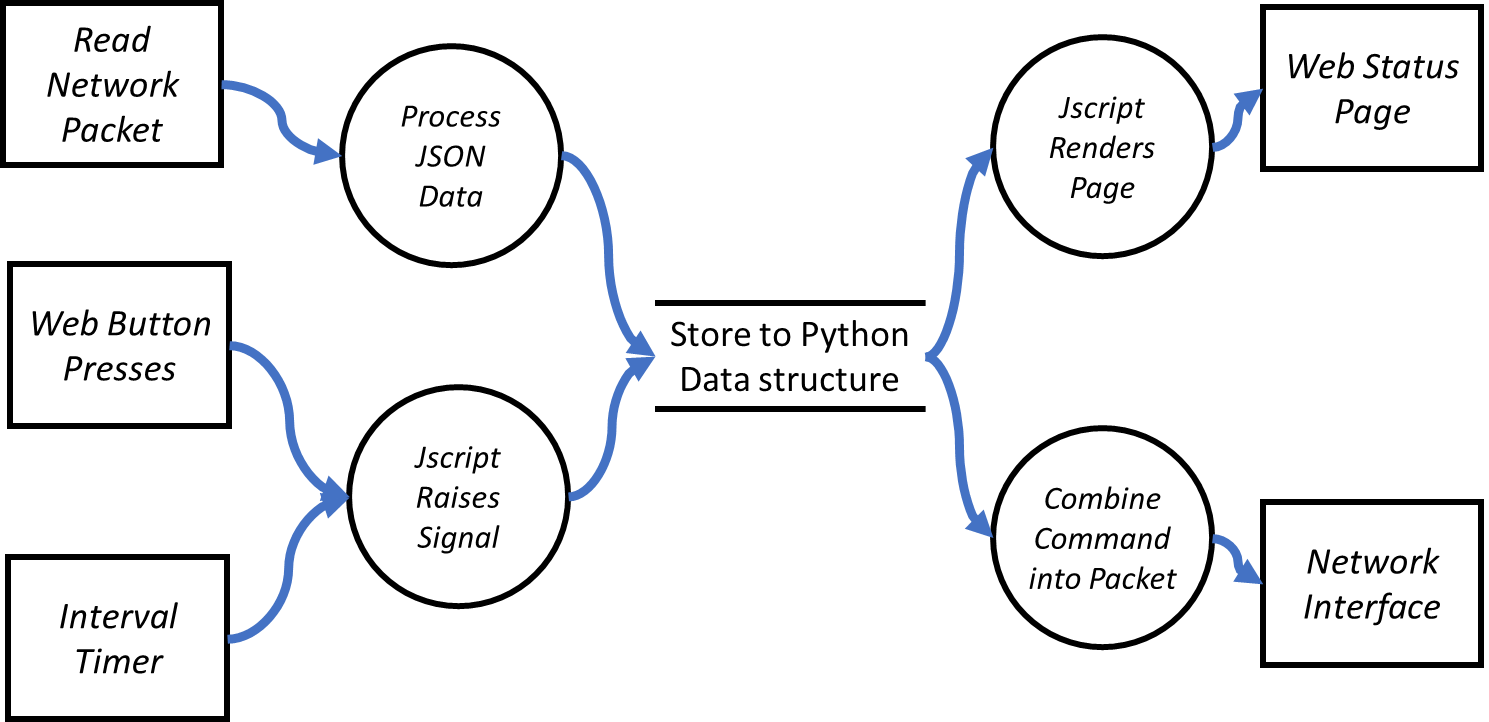
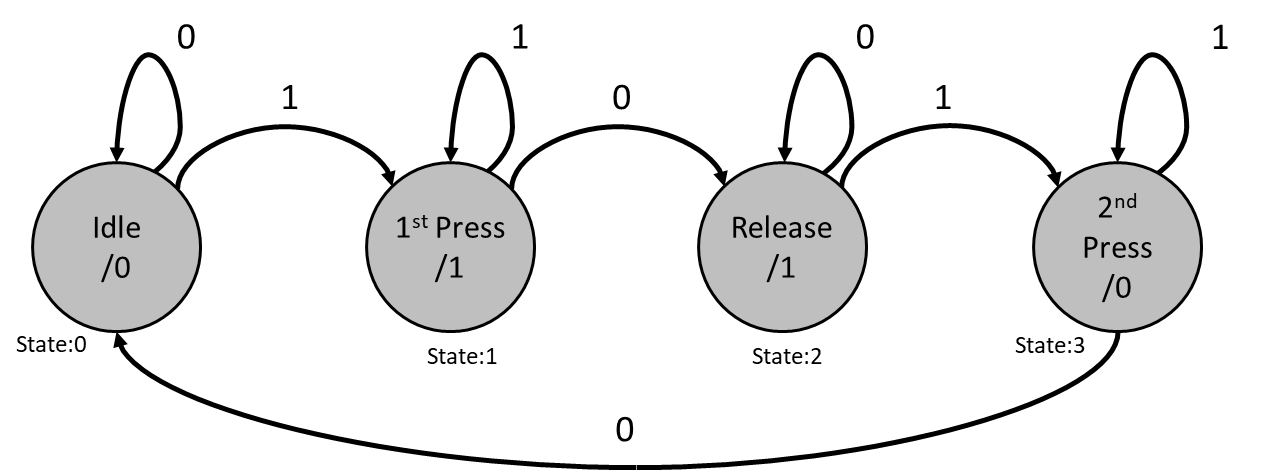


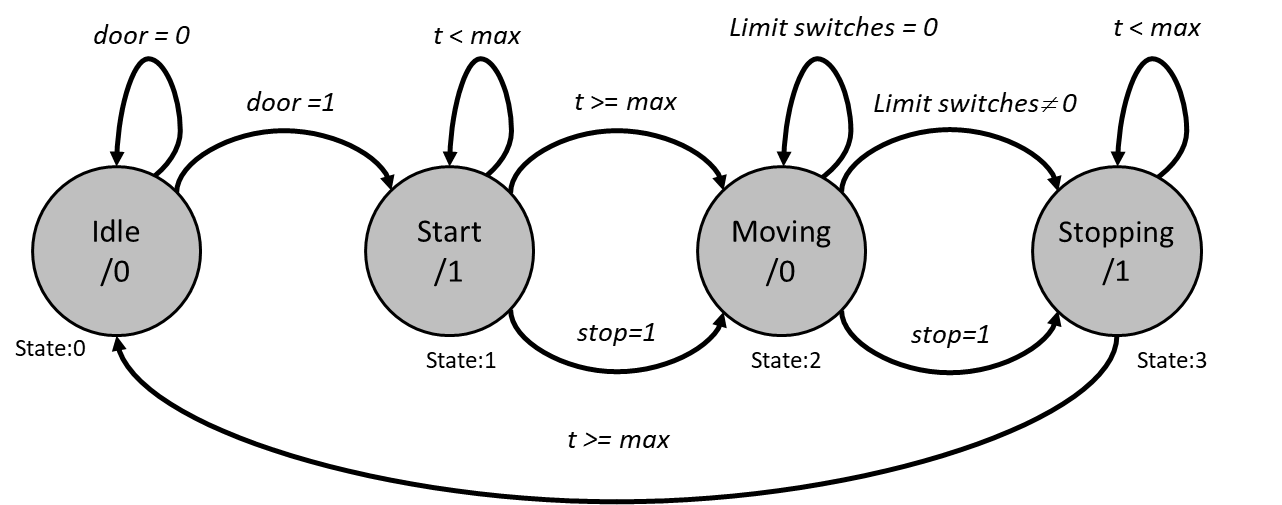
Figure 5. Data Flow Graph of the Web Server Subsystem

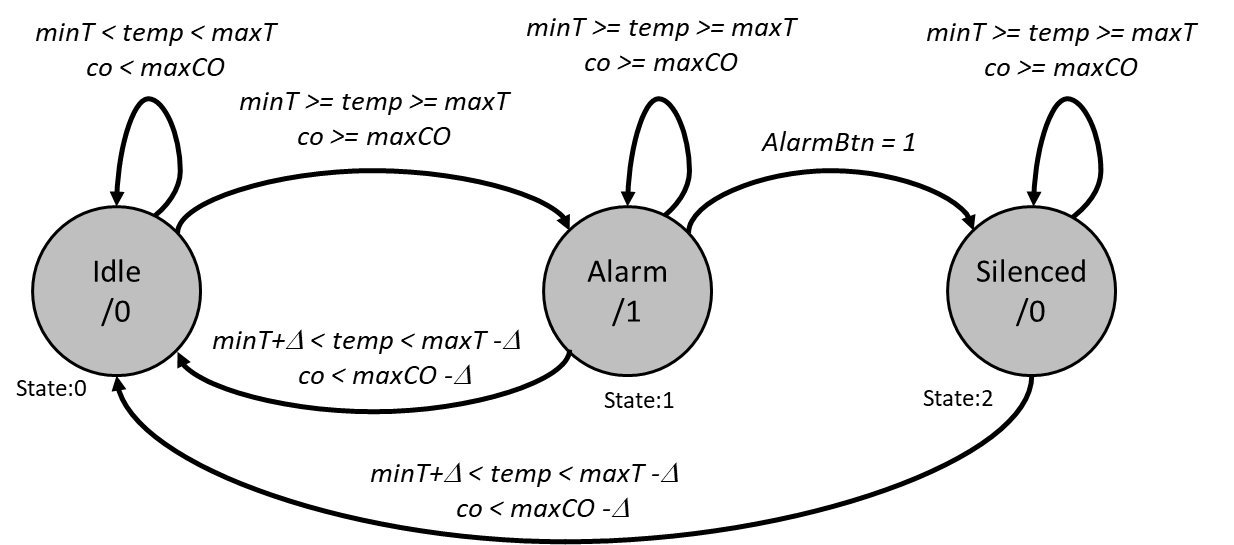
# Detail Design

## Implementation

## Embedded System







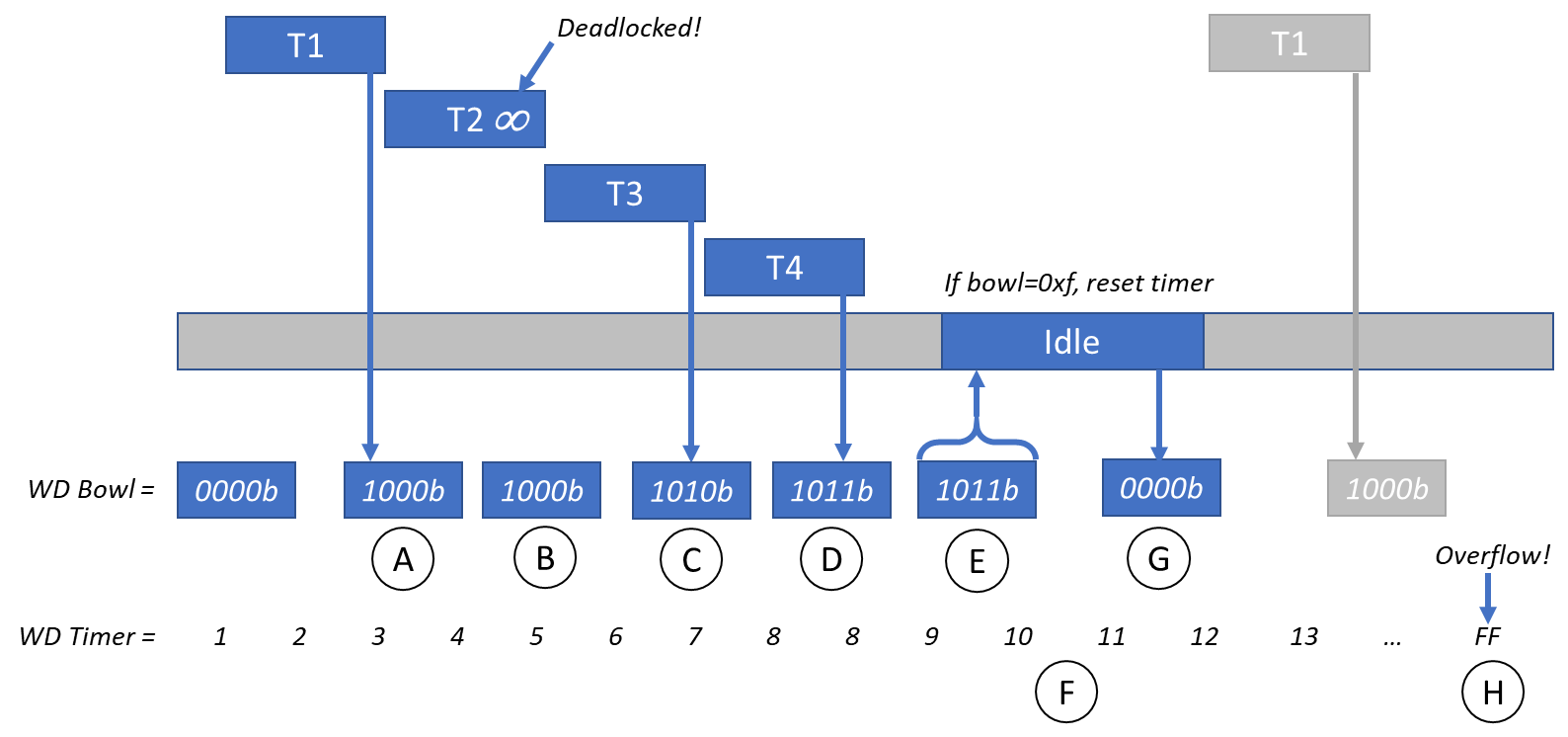
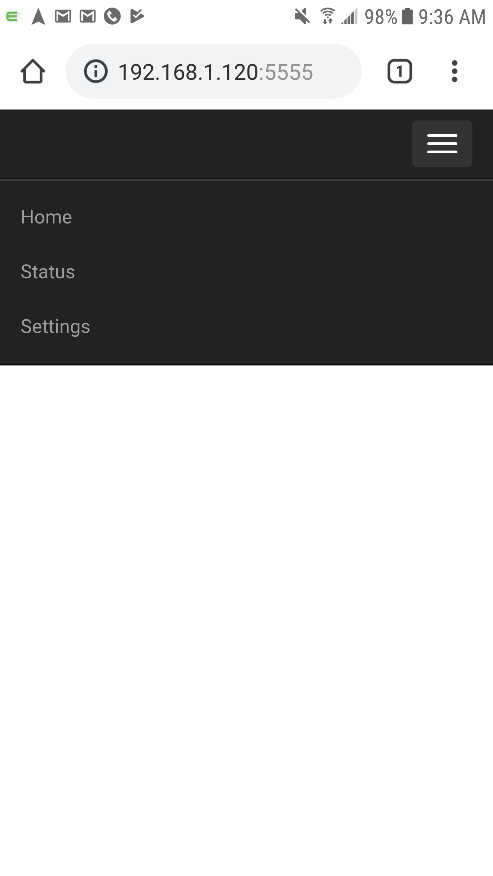
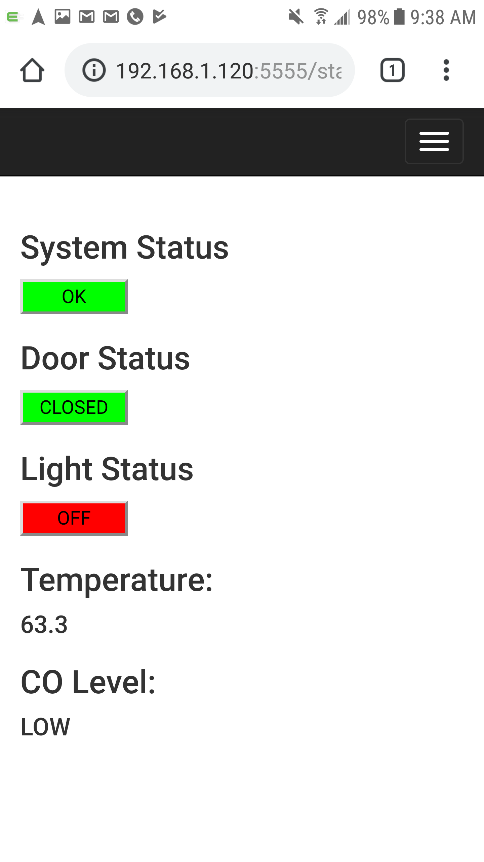


Table 2. Bill of Materials

|  |  |  |  |
| --- | --- | --- | --- |
| **REF** | **Part** | **PN** | **Qty** |
| U1 | ESP32\_NodeMCU | B07F877YZQ | 1 |
| U2 | 4x20 LCD Display | 030003LA | 1 |
| D1 | Laser Diode Assy | 1172 | 1 |
| D2 | Laser Detector | B01M8PFZRC | 1 |
| S1-S5 | Mom PCB Switch | B06XT3FLVM | 4 |
| S6-S8 | Limit Switch | V-153-1C25 | 2 |
| K1 | 4x Relay Assy | 4450182 | 1 |
| J1 | USB Cable | 7T9MV4 | 1 |
| CO | CO Detector | FTC010-MQ-7 | 1 |
| R1 | 10k Thermistor | MF52-103 | 1 |
| R2 | 10k ohm ±1% 1/4W | MFP-25BRD52-10K | 1 |
| - | Breadboard | B01EV6LJ7G | 1 |

## Web Service





# Testing

## Performance

### Utilization

### Time Loading

# Accomplishments

## Objectives Met

## Limitations

## Lessons Learned

## Possible Improvements

The project made several design decisions in the interest of simpler development. To ease network integration, the system employs UDP and no encryption. These are considered experimentation only type configurations and would need to be resolved in a more production ready version of the product. The system also interfaces to the garage door lift motor through the operator button, this too could be improved. Finally, the system depends on the obstacle sensor being well aligned and can easily be spoofed by jumpering the signal wires. This section discusses potential improvements to solve these issues intended in a future revision of the GarageRTC project.

UDP simplifies the design by reducing the network configuration within the web app and embedded unit, but routing traffic to any area larger than the immediate subnet is impractical. The excessive packets will hinder network congestion and are not easily forwarded to another network if the server was not on the same subnet. To solve this, the system should be moved over to TCP based networking between the server and GarageRTC. This would also more easily enable security features such as TLS.

The lack of encryption is certainly simplified design by not having to create, manage keys, and develop paring processes for the server and embedded unit. Obviously in this day an age, any IoT device is under heavy scrutiny from security researchers. With the current design, anyone on the network could easily replay a packet and get the garage door to move. The solution would involve adding application layer encryption between the embedded and server. For further protections, TLS could also be added by embedding certificates that are mutually authenticated.

Another limitation is that the GarageRTC interacts with the garage door lift motor through the opener button interface. Each garage door operates a little differently, such as longer button pulses or automatic reversing after a down trigger. This makes the interaction between the GarageRTC and the opener not ideal. A potential solution would be to incorporate a direct dive reversable H-Bridge controller into the project. That would give the controller better control over the door. The downside is there are significant safety impacts that need to be considered when taking over control of the door lift mechanism.

Lastly, the laser transmitter and receiver used by the project could be improved by replacing it with a pair of conventional garage break beam eyes. These devices typically use a proprietary protocol but do incorporate anti-spoof circuitry, so the door controller can tell if they have been intentionally jumpered. They also are more tolerant of misalignment. Replacing the laser with libraries to interface with different conventional eyes would reduce system cost, improve safety, reliability, and performance.

# Comparison of Commercial Offerings

Several commercial IoT garage devices are available to consumers. These devices are typically developed by major opener manufacturers and marketed as an upgrade to the existing opener electronics. This section provides a short review of several of these offerings and their compatible hardware. Surprisingly, none of these technologies offered native integration with digital assistants such as Amazon Alexa. We provide a comparison between the GarageRTC features and the commercial offerings and discuss the benefits and drawbacks of these paired solutions.

## Commercial Offerings Surveyed

This section provides a short review of several of these offerings and their compatible hardware. Surprisingly, none of these technologies offered native integration with digital assistants such as Amazon Alexa. We provide a features comparison between the GarageRTC and commercial offerings reviewing the benefits and drawbacks of these paired solutions.

## Commercial IoT Garage Interfaces

The market was surveyed for commercial IoT garage interfaces that offer similar capabilities as to what was implemented for the GarageRTC project. The two most prominent opener companies, Chamberlain and Genie, offered hardware designed to interface with their respective openers. There was also a couple of notable general-purpose add-on modules designed to interface with a variety of openers. There were several minor part number differences, but the technologies surveyed were grouped as follows:

* Chamberlain MYQ-G0301
* Aladdin Connect
* Garage Door Buddy
* GoControl GD00z-4

## Comparison with GarageRTC Features

Next the team created collected documentation publicly available from the device’s respective marking pages. From this the research team composed a matrix considering connectivity, digital assistant integration, sensing and control, local interface, and openness of design. The compiled data is summarized in Table 3 below. References to the respective manufacturer’s site is included in the appendix of this document.

Table 3. Comparison of Commercial Devices

| **Feature** | MYQ-G301 | Aladdin Connect | Door Buddy | GoControl | GarageRTC |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Connectivity** |  |  |  |  |  |  |
| WiFi | ✔ | ✔ | ✔ |  | ✔ |  |
| Bluetooth |  |  |  |  | *1* |  |
| ZWave |  |  |  | ✔ |  |  |
| **Digital Assistant** |  |  |  |  |  |  |
| Alexa |  |  |  |  | *2* |  |
| Google |  |  |  |  | *3* |  |
| Siri |  |  |  |  | *3* |  |
| **Sense/Control** |  |  |  |  |  |  |
| Aux Control | *4* |  |  |  | ✔ |  |
| Amb. Temp |  |  |  |  | ✔ |  |
| Amb. CO |  |  |  |  | ✔ |  |
| Door Position | ✔ | ✔ | ✔ | ✔ | ✔ |  |
| **Local Control** |  |  |  |  |  |  |
| LCD Display |  |  |  |  | ✔ |  |
| Door | ✔ | ✔ | ✔ | ✔ | ✔ |  |
| Aux/Light |  |  |  |  | ✔ |  |
| **Open Source** |  |  |  |  |  |  |
| Web App |  |  |  |  | ✔ |  |
| FOSS |  |  |  |  | ✔ |  |
| Published API |  |  |  |  | ✔ |  |
| HW Schematic |  |  |  |  | ✔ |  |
| Universal |  |  | ✔ | ✔ | ✔ |  |
| **Reference** | [6] | [7] | [8] | [9] |  |  |
| 1. ESP32 contains the hardware but reference design does not implement.  2. Libraries exist that could integrate this capability. [10]  3. Can be integrated through webhook/REST API calls. [11]  4. Requires purchase of additional hardware. | | | | | | |

## How GarageRTC differs from current solutions

The GarageRTC is comparable to commercial offerings in all categories and is stand-out in design openness. The open source nature of the GarageRTC makes it considerably more adaptable and extensible than the consumer offerings.

It is important to note that security was omitted from the table. The team found that, while the manufacturers advertised various “security assurances” they did not publish details on how these were achieved. The GarageRTC does not currently employ any sort of link encryption and depends completely on the security of the wireless access point. The team expected the security category to be the location where the commercial offerings would outshine the project. However, since the details of their security mechanisms could not be determined, it is difficult to say if the commercial systems are offering much more than security through obscurity. The research work done by J. Margulies and XXXX indicate verifying levels of security effectiveness. For these reasons, the security offerings cannot be effectively compared without a more thorough evaluation. A proposal for security on the GarageRTC project can be found in section VII.D of this paper.

The commercial offerings did have the advantage in that they tended to offer applications with credentials. All solutions surveyed offered Android and IOS native apps for their remote interface. Frequently, these applications are merely application wrapping around a web interface. This simplifies development and can help with security if the site is properly protected. The location of the IoT service could not be conclusively determined from the documentation surveyed, but it is reasonable to assume that it is most likely a closed source cloud hosted service. If the support for the cloud service is suspended, then these devices would likely fail to function. Since the GarageRTC hosts a generic web page, it’s reasonable to say that the GarageRTC offers equivalent performance.

## Benefits and drawbacks of the integrated solution

The primary benefit of the commercially available devices is they have corporate backed development support. This translates to more refined software and user-friendly interfaces as they have superior developer resources as compared to the GarageRTC research team. They have the capacity to fully implement and test their solutions and maintain professional assistance hotlines. Further, mass production of their assemblies’ results in greater availability and prices typically ranging from $50 to $100. However, if the popularity of GarageRTC caught on, the resulting user community could quickly make up this gap. Further development could produce layouts to meet off the shelf housings or 3D printed enclosures.

The major drawbacks of the commercial units are their closed source nature. Vulnerabilities have been identified in the garage door solutions and countless other issues may exist. [12] These units have not undergone any sort of security community analysis and publish very little documentation on security controls available. The GarageRTC, however, doesn’t currently employ any security as it was not the focus of this effort, could easily have any flavor of security controls the developer prefers. Since the source is open, the security community could then review the implementation, identify potential flaws and make recommendations for remediation.

The closed source communication protocols also mean that the lifetime of these products is directly tied to the supplier’s willingness to continue support. If the companies go out of business or cease support in favor of a newer generation part, customers will be out of luck. The GarageRTC project offers the ability for anyone to stand up or share their own web application servers and therefore can indefinitely support hardware built to meet the GarageRTC protocols.

# Conclusions and Closing Remarks

The GarageRTC met most of the performance and design requirements as identified in the concept and requirements phase.

High level design

Detailed design

Testing

Accomplishments

Comparison with commercial offerings

# References

|  |  |
| --- | --- |
| [1] | A. Zeichick, "Throwing our IoT investment in the trash thanks to NetGear," *Network World,* 11 July 2016. |
| [2] | J. Perlow, "When your IoT goes dark: Why every device must be open source and multicloud," *ZDNet,* 12 March 2019. |
| [3] | S. Levin, "Squeezed out: widely mocked startup Juicero is shutting down," *The Guardian,* 1 September 2017. |
| [4] | R. Suehle, "Q. What is the secret of Red Hat's Success?," *Technology Innovation Management Review,* January 2012. |
| [5] | R. Barry, "Mastering the FreeRTOS Real Time Kernal," Real Time Engineers Ltd, 2016. |
| [6] | Chamberlain, "MYQ Smart Garage Hub," Chamberlain, 2019. [Online]. Available: https://www.chamberlain.com/myq-smart-garage-hub/p/MYQ-G0301-D. [Accessed 13 April 2019]. |
| [7] | Genie, "Aladdin Connect," Genie, 2019. [Online]. Available: https://www.geniecompany.com/aladdinconnect/default.aspx. [Accessed 13 April 2019]. |
| [8] | GarageDoorBuddy, "GarageDoorBuddy - Remotely Control your garage doors from Anywhere," Garage Door Buddy, 2019. [Online]. Available: http://garagedoorbuddy.com/. [Accessed 13 April 2019]. |
| [9] | GoCOntrol, "GD00Z-4: Z-Wave Garage Door Opener Remote Controller Accessory," GoControl, 2019. [Online]. Available: http://www.gocontrol.com/detail.php?productId=4. [Accessed 13 April 2019]. |
| [10] | X. Pérez, "FauxmoESP - Amazon Alexa support for ESP8266 and ESP32 devices," 2019. |
| [11] | X. Pérez, "Using Google Assistant to control your ESP8266 devices," Tinkerman, 2019. [Online]. Available: https://tinkerman.cat/using-google-assistant-control-your-esp8266-devices/. [Accessed 13 April 2019]. |
| [12] | J. Margulies, "Garage Door Openers: An Internet of Things Case Study," *IEEE Security & Privacy,* 2015. |
| [13] | C. Goudie and R. Weidner, "Home hackers: Digital invaders a threat to your house," *ABC 7 News, Chicago IL,* 12 February 2015. |