TEMPERATURE SENSOR NETWORK FOR HVAC CONTROL

By

Haige Chen

Ryan Finley

Heming Wang

Final Report for ECE 445, Senior Design, Spring 2019

TA: Dongwei Shi

1 May 2019

Project No. 33

Abstract

Our system aims to provide a low-cost solution for the problem of uneven heating and cooling among different rooms in typical households. This problem often arises from heating, ventilation, and air conditioning (HVAC) systems which perform temperature measurements at one central location. We created a system that mitigates this problem by measuring temperature across multiple rooms and manipulating air-vents to regulate temperature more precisely. We also provide a user-friendly interface to communicate historical temperature data and alerts to customers. The system is robust and scalable - providing an easy means to extend the home temperature control system if desired.

Contents

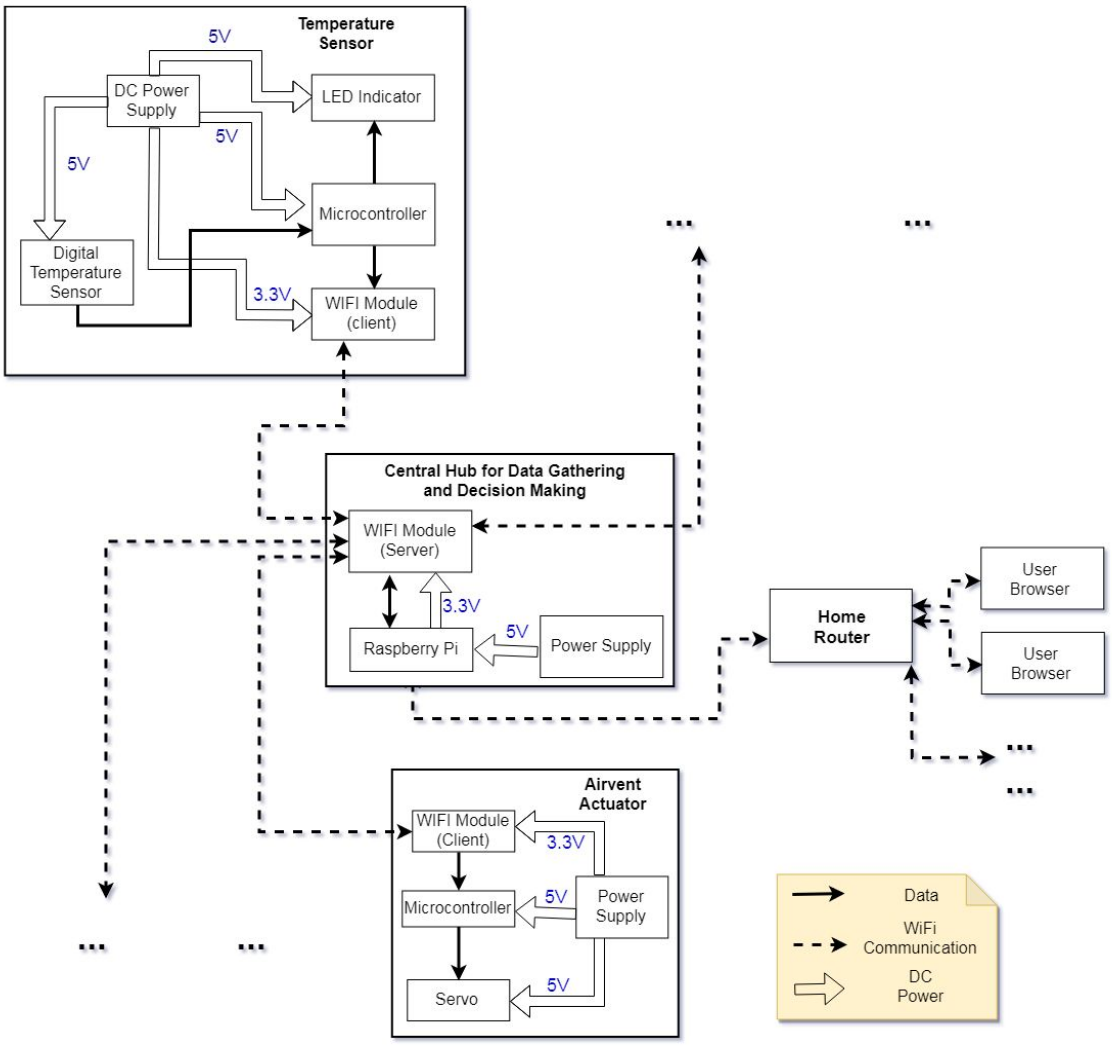

Figure 1. Top level Block Diagram of the HVAC Controller Network

1. Introduction

We probably have all experienced this scenario: on a hot summer day we set our HVAC to a comfortable temperature and proceed to enjoy our day. However, at night when we are about to go to bed, we realize that although our living rooms are at a cool 72°F, our bedrooms are at a scorching 80°F. This is because in most households, the HVAC systems are controlled through a single thermostat that is usually located in the living room, and as a result, the systems only know that their living room is at the desired temperature. Our HVAC network, as shown by its block diagram in Figure 1, aims to address the temperature difference among each room by adding wireless temperature sensors in those rooms, and wirelessly controlled air vents to regulate the temperature in those locations based on the temperature readings we receive.

Our system consists of three major parts: WiFi enabled temperature sensor modules, a central hub, and WiFi controlled air vent actuators. The sensor modules relay the temperature readings via WiFi to the central hub, which displays the data to the user and sends commands to the air vent actuators based on user settings. The actuators either open or close the air vents based on the commands they received. We have tested and confirmed that our system has good temperature accuracy, acceptable operation range, and reasonable (re)connecting time. Ultimately, the system performs as intended, since that we are able to connect individual components wirelessly, read data from the sensors, and have the air vents open and close based on user settings. We acknowledge that we can further develop our project to improve user friendliness, expand scalability, and lower power consumption.

2 Design

2.1 Temperature Sensor Module

The temperature sensor module takes accurate, real-time temperature measurements and send the data to central hub through WiFi. In our system there are multiple temperature sensor modules distributed in different rooms in a house.

Hardware components of this module include a 5V DC voltage regulator circuit, an ESP8266 WiFi module, a power indicator LED, a connection status indicator LED, a DS18B20 temperature sensor, an ATMega328p microprocessor and its peripheral components, as shown in Figure 2. The PCB (shown in Figure 3) does not include the WiFi module. The WiFi module requires additional voltage regulating circuit and is mounted on a perforated board connected to the printed circuit board (PCB) using jumper wires.

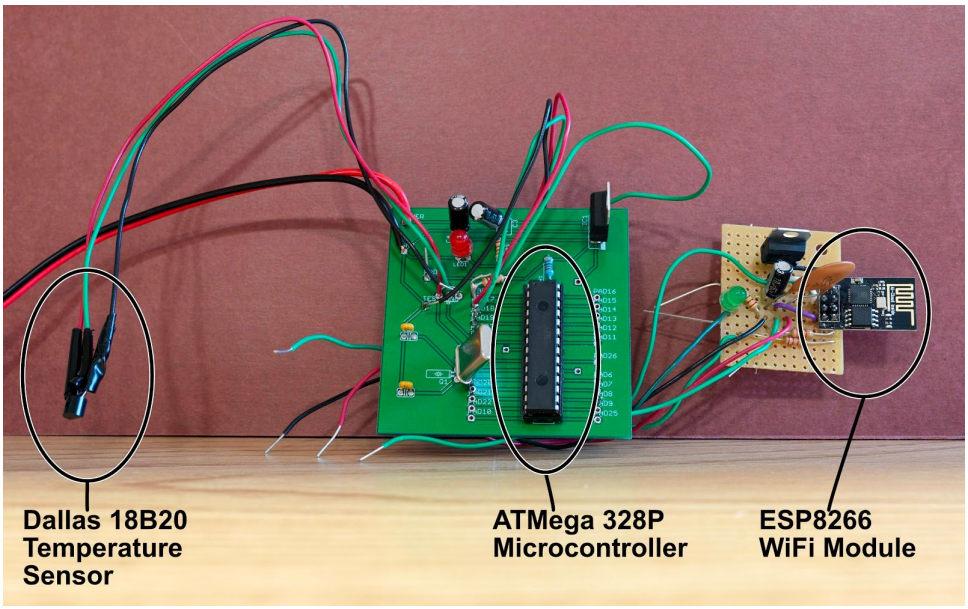

Figure 2. Picture of Temperature Sensor Module

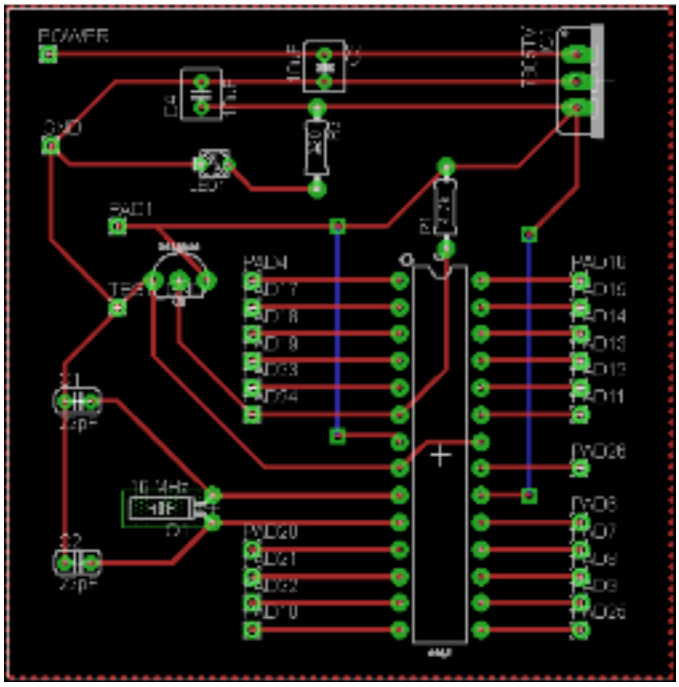

Figure 3. PCB of Temperature Sensor Module

2.1.1 Temperature Sensor DS18B20

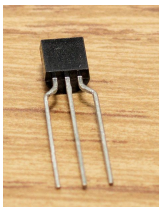

Figure 4. Photo of DS18B20

The sensor (shown in Figure 4) has one data line. Temperature data can be requested and read through a microcontroller using the Arduino DallasTemperature Library[1]. In the circuit a 4.7 kOhm pull-up resistor connects data line and VCC line.

The sensor can convert readings to 9-bit digital words. These readings measure temperature in the range 67°F to +257°F. The accuracy is ±0.9°F from 14°F to 185°F[2].

2.1.2 ESP8266 WiFi Module

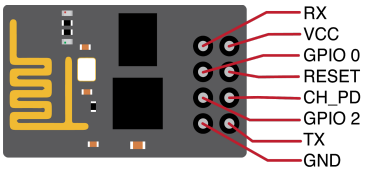

Figure 5. Pin Assignment and Board Layout Diagram[3]

We used ESP8266 WiFi module, shown in Figure 5, for wireless communications between temperature sensor modules and the central hub.

The ESP8266 module takes 3.3V voltage level. Because our main PCB uses 5V power supply, we designed a voltage regulator circuit to step down DC voltage from 5V to 3.3V. In addition, all inputs to ESP8266 should not exceed 3.3V. Therefore, we used a resistor voltage divider circuit (consisting of two 10k Ohm resistors) to convert 5V TX line to around 2.5V, which feeds into RX of ESP8266.

The ESP8266 module comes with firmware that supports AT commands[4]. We configure the ESP8266 on temperature sensor module as WiFi client and TCP client. The basic control flow involves the commands listed in Table 1.

Table 1. List of AT Commands Used

Sequence	Command	Explanation
1	AT+RST	Restart WiFi Module
2	AT+CWMODE=1	Set module in client mode
3	AT+CWJAP="hvac-network","hvachvac"	Connect to WiFi Access Point with ssid "hvac-network" and password "hvachvac"
4	AT+CIPSTART="TCP","192.168.4.1",333	Connect to TCP Host with IP "192.168.4.1" on Port 333
5	AT+CIPSEND=5	Send message of length 5 followed by the message

In order to have a robust system, our modules should be able to detect loss of connection and re-establish connection reliably. The details of auto-reconnection algorithm are shown in Figure 6 below.

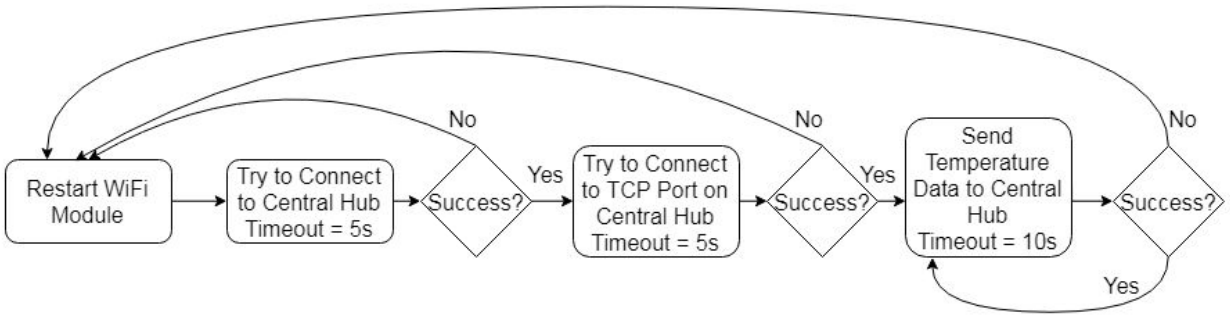

Figure 6. Flowchart of Link Loss Detection and Auto-Reconnection Algorithm

2.1.3 ATMega328p Microcontroller

We use ATMega328p to read data from temperature sensor and control ESP8266 module. In the circuit design, we used an external 16 MHz crystal oscillator as the clock. We used Arduino IDE and Arduino board to program it. Then we mounted it onto the IC socket on our PCB for testing.

2.2 Central Hub

The central hub's responsibilities include listening to the remote temperature sensor modules to gather temperature readings and sending commands to the air vent actuator modules for opening and closing. The air vent commands and client-facing alerts are generated through decision logic based on the latest temperature reading received from each temperature sensor module. Additionally, the central hub will store historical temperature data and generated alerts in a MongoDB instance for access the the client-facing web interface. The interface also includes configurations for users to specify the range in temperatures they desire the system to enforce.

2.2.1 Hardware

The central hub's hardware is created from two off-the-shelf components: a Raspberry Pi 3 Model B+ and an ESP8266 WiFi module. The Raspberry Pi model isn't crucial as long as necessary GPIO support is present. Connecting the two involved powering the ESP through the GPIO 3V3 pin[5], which outputs 3.3V and fits in the ESP's operating voltage range of 2.5V-3.6V . The CH_PD pin and RST pin are also powered with a 2.2k Ohm pull-up resistor. We also connect GPIO's TXD to ESP8266's RX and GPIO's RXD to ESP8266's TX. Finally, the grounds are connected between the components. The connection diagram is shown in Figure 7.

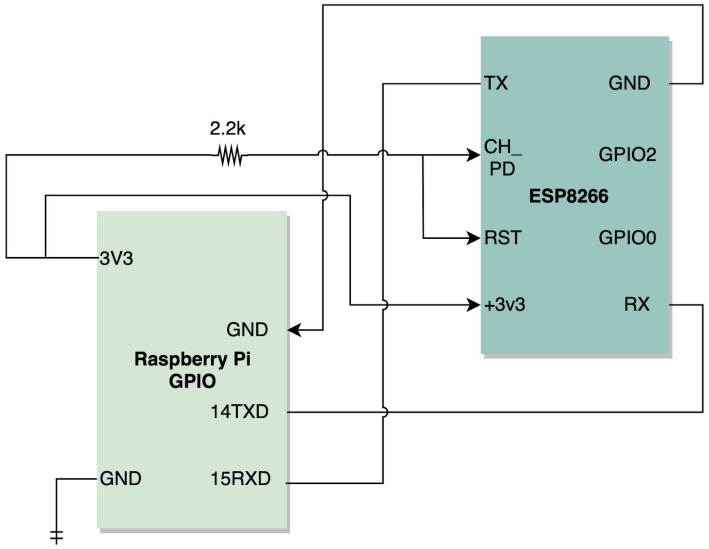

Figure 7. Block diagram of the central hub

2.2.1 Software Composition

The software is composed of two Docker containers which run on the Docker VM. Docker is ran atop Raspbian[6], a Linux Debian-based computer operating system for the Raspberry Pi. The first docker container is a pre-compiled 32-bit ARM MongoDB binary built specifically for the Raspberry Pi[7]

whereas the second docker hosts the server itself. The server container has full communication with the MongoDB container as well as the “serial0” UART port, allowing communication with the externally connected ESP8266. The server container itself is split into three threads which each have access to a different subset of resources. The GPIO Thread is the only thread with access to the GPIO pins and thus the ESP8266. The Processing Thread and Web Thread (a Python Flask app) have access to the MongoDB container. The Flask App hosts our web application, so it also communicates to external connections through the Raspberry Pi’s built-in WiFi chip. Relationship among software components is illustrated in Figure 8.

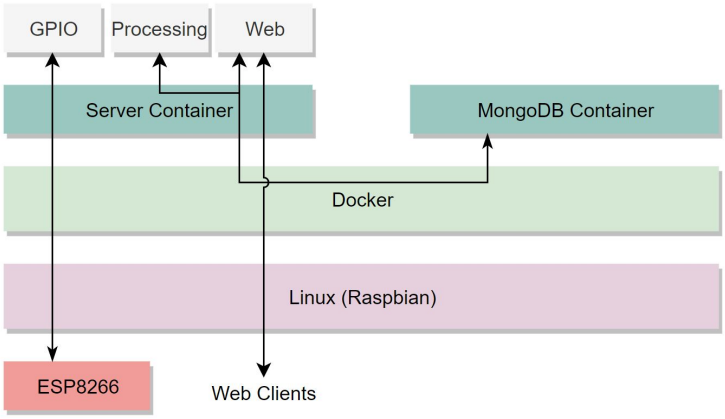

Figure 8. Block diagram of the central hub’s Software Composition

2.2.2 Server Data-flow

The server container is composed of three threads that run in a single python process: the Web Thread, the Processing Thread, and the GPIO Thread. Splitting the server into these three “sub-modules” allows for effective concurrency, crash resiliency, and modularity.

GPIO Thread: The main purpose of this thread is to act as an ESP8266 communication module through the use of AT command and Python’s Serial library. The GPIO Thread provides an interface for initializing communication through the ESP and sending/receiving messages simultaneously. This thread also maintains connections with clients by pinging each connection once every 3 seconds. Due to the nature of the AT command interface, it’s possible to receive messages while in the process of sending. The GPIO thread circumvents this problem by checking for and intercepting received messages during the send process. As a result, the ping protocol also serves as a receive poller. Communication with other threads is done exclusively through two message buffers: all temperature readings are placed into the incoming message buffer, and all air vent commands are received from the outgoing message buffer.

Processing Thread: This thread is the bridge between the GPIO Thread and the Web Thread. It also holds all the decision logic which is used to generate alerts and air vent commands. The current system supports three alerts and their corresponding air vent commands:

1. *temperature higher than desired*: This is set if received reading is greater than the configured maximum temperature (which is set in the front-end). The air vent of the corresponding temperature sensor module will be told to open.
2. *temperature lower than desired*: This is set if received reading is less than configured minimum temperature. The air vent of the corresponding temperature sensor module will be told to open.
3. *temperature variance higher than desired*: This is set if, among latest readings, the difference between the highest node temperature and lowest node temperature is greater than a configured value. The system sets this value to be 4°F. This alert does not affect air vents.

Air vents will be told to close in all other cases.

It's worth noting the benefits of having the Processing Thread since the system could still run if the processing and GPIO threads were combined. As an intermediary, the Processing Thread effectively separates the Web Thread and the GPIO Thread, making signal mocking trivial and easing development. The Processing Thread also improves crash resilience by monitoring the GPIO Thread and restarting it when it goes down (and visa-versa). Additionally, the Processing Thread enables more precise scaling. An imbalance in packet processing and ESP handling may require only one of the two threads to duplicate upon higher workloads.

The relation between the three threads is shown in the block diagram in Figure 9.

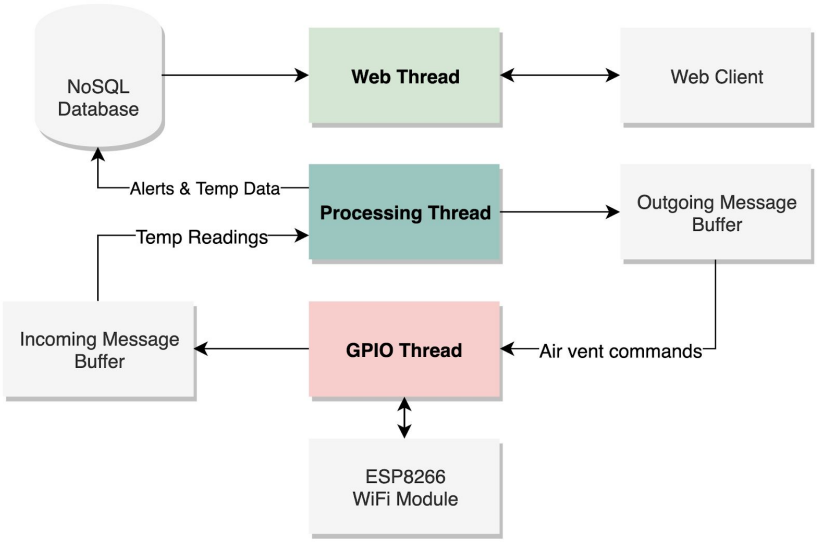

Figure 9. Block Diagram of central hub's server data-flow

Web Thread: This thread runs a Python Flask application to display alerts and historical temperature data to clients. All data exposed to this thread is stored in the MongoDB container. The only exception to this is globally shared configurations which are set from user input in the front-end and used to generate decisions in the Processing Thread. Historical temperature data is visualized using charts.js[8], and front-end styling is done through the Bulma web-framework[9]. An example of the front-end user web interface is shown in Figure 10.

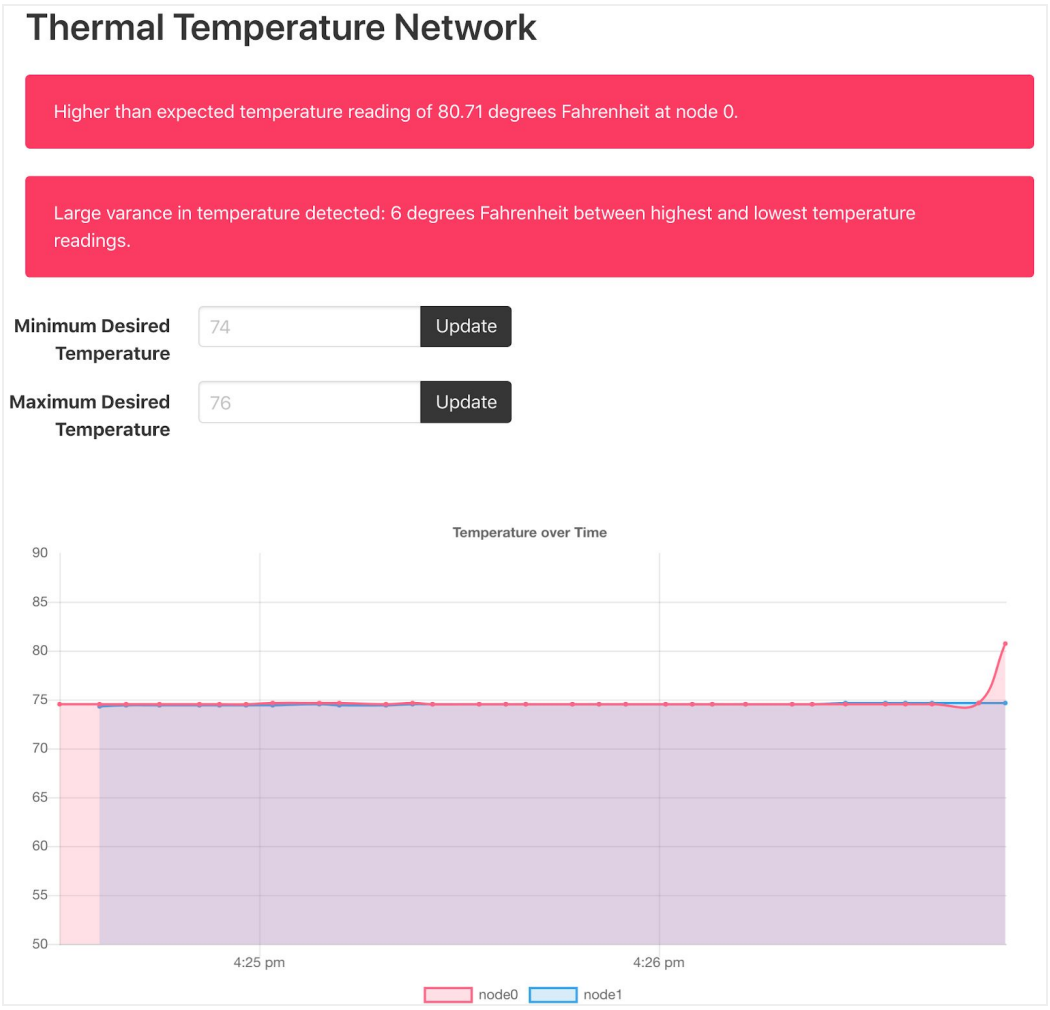

Figure 10. Web front-end with two alerts displayed

2.3 Air vent Actuator Module

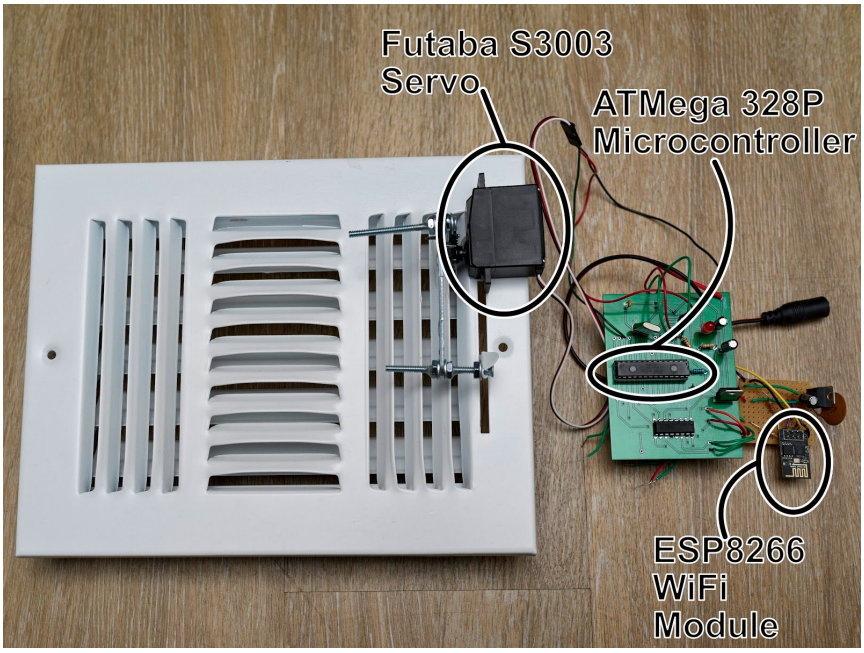

Figure 11. Picture of the air vent actuator module

The air vent actuator module acts as a temperature regulator by opening and closing the vent. The actuation is accomplished by a servo, which is designed to fit on top of any air vent with lever-operated shutters, so users with those vents do not have to replace their existing ones. As shown in Figure 11, it has the same microcontroller and WiFi module which we have elaborated in previous sections. Software-wise, the microcontroller is programmed to receive open/close commands from the central hub.

2.3.1 Servo Motor

The servo motor we use is a Futaba S3003. It is commonly used for radio controlled vehicles, and can be bought at a low price while still producing respectable torque. We run our servo motor at 5V, at which it produces about 3.2kg-cm of torque.

To control the servo, we utilize Arduino's built-in servo library that allows us to control the servo arm angle by simply entering the desired number in degrees in our code. The servo accepts pulse width modulation (PWM) signal as the control signal input: a 1.5ms wide pulse corresponds to 90-degree neutral position, while narrowing the pulse decreases the angle toward 0° and widening 180°.

We choose servo over the alternatives (e.g. stepper motor) for its simplicity and ease of use. Unlike a stepper motor, a servo does not need extra driving circuits, so we only need to connect power and ground besides the data pin that goes into the microcontroller.

3. Design Verification

3.1 Temperature Sensor Accuracy

-Methods: Because we do not have access to any high precision thermometer, it was very hard for us to compare temperature sensor readings to the actual temperature. Therefore, we used a more accessible measurement that is supposed to be correct: temperature readings from thermostats. We placed our temperature sensors within 1 meter of the thermostat in classrooms. After temperature readings stabilized, we recorded the average of 5 consecutive readings and compared them to the readings from the thermostat.

-Results: The measurement data and accuracy are shown in Table 2. With percentage of error below 1%, we claim the sensor to be accurate enough for our application.

Table 2. Temperature Sensor Accuracy Measurement

Sensor 1	Sensor 2	Thermostat Reading	Sensor 1 Accuracy	Sensor 2 Accuracy
74.75	75.09	75.5	0.993%	0.543%

3.2 WiFi Module Range Test

-Methods: We first measured line-of-sight range where there is no obstacle between the sender and the receiver. We set up the central hub at one end of a hallway and one temperature sensor module held by a person. After initial connection was established, the person started moving away from the central hub along the hall while holding the temperature sensor module (powered by battery in this case). We tested indoor range by setting up the central hub in the classroom. The person held the temperature sensor module in the hallway and started moving away from the room.

-Results: For line-of-sight experiment, after moving to the other end of the hall (about 50 meters away), connection is still maintained and auto-reconnection was successful. For indoor test, at about 10 meters away, the connection was lost. We observed there were two concrete walls in between the two nodes. For a typical house, walls are usually not concrete, and therefore should have a better range performance.

3.3 System Level: Auto-Reconnection Time

-Methods: At a system level, it is crucial to have a robust reconnection scheme so that no data is lost. We measured the number of seconds from powering on the WiFi module to successfully establishing connection. We ran three trials and took the average.

-Results: As shown in Table 3, we measured an average of 17.40 seconds and 13.97 seconds reconnection time for temperature sensor module and air vent actuator module respectively. They are

short compared to the rate we want to send data, which is around every 5 minutes. Therefore, the auto-reconnection scheme is fast enough to prevent data loss.

Table 3. Auto-Reconnection Time Measurement

	Sensor Module	**Air Vent Module**
Trial 1	20.10 s	14.31 s
Trial 2	15.88 s	14.41 s
Trial 3	16.21 s	13.19 s
Avg.	**17.40 s**	**13.97 s**

4. Costs

4.1 Parts

Table 4. Parts Costs

Part	Manufacturer	Retail Cost ($)	Bulk Purchase Cost ($)	Actual Cost ($)
ATMega328p	Atmel	4.30	21.50	21.50
ESP8266	Espressif Systems	6.95	41.70	41.70
DS18B20	Maxim Integrated	3.95	11.85	11.85
Raspberry Pi Model B+	Raspberry Pi Foundation	34.49	34.49	34.49
Servo Motor	Futaba	11.39	22.78	22.78
SD Card	SanDisk	5.79	5.79	5.79
Miscellaneous	N/A	10.00	10.00	10.00
Total				**148.11**

4.2 Labor

Rate: $40/hour

3 members x 16 weeks x 10 hours = 480 man hours

Total: 480 x 40 = $19,200

Table 5. Schedule

Week	Haige Chen	Ryan Finley	Heming Wang
2/18	Design review; prototyped wifi capability	Design review; configured Raspberry Pi; set up web server	Design review; tested sensor readings
2/25	Transferred Wifi submodule design to standalone ATMega328p	Developed Python script for interfacing between Wifi module and Raspberry Pi	Transferred temperature sensor submodule design to standalone ATMega328p
3/2	Designed ATMEGA328p peripheral circuit for temperature sensor module	Made sure Raspberry Pi receives correct data	Designed ATMEGA328p peripheral circuit for temperature sensor module
3/9	Tested 5V power supply circuits	Designed web front end; set up and test data display and alerts system	Tested 3.3V power supply circuits
3/16	Spring break	Spring break	Spring break
3/23	Prototyped air vent actuator circuit with Arduino	Tested actuator can correctly receive commands	Built air vent actuator (both circuit and mechanical design
3/30	Transferred actuator design to standalone ATMega328p	Designed decision algorithm	Tested actuator; tested power circuit of air vent actuator
4/6	Wrapped up any unfinished work from previous weeks;	Wrapped up any unfinished work from previous weeks	Wrapped up any unfinished work from previous weeks;
4/13	Put all the modules together; tweaked parameters in decision algorithm	Put all the modules together; tweaked parameters in decision algorithm;	Put all the modules together; tweaked parameters in decision algorithm;
4/20	Final debugging and testing	Final debugging and testing	Final debugging and testing
4/27	Final Presentation	Final Presentation	Final Presentation

5. Conclusion

5.1 Accomplishments

Overall, we managed to accomplish all of the objectives listed in our design document. We successfully connected two temperature sensors and two air vent actuators to our network. Our central hub functioned properly as the master controller. The temperature sensor modules could send temperature data to the central hub via WiFi, and the central hub parsed the data and displayed the values on our web interface. Additionally, the web interface could display the temperature data of each sensor against time on a graph. We could set upper and lower temperature thresholds on the web interface by simply typing in the numbers, and the central hub could update its decisions based on the updated parameters. Finally, we were able to send the open/close commands to the actuators via WiFi, and the servo motors can execute the commands accordingly.

5.2 Uncertainties

One uncertainty we faced was to choose between battery and wall supply to power the temperature sensor modules. We chose wall supply in the end because it eliminated the need to recharge/replace batteries. However, batteries allow more flexibility in sensor placement locations.

We considered a typical Li-ion battery with power ratings of (3.7V, 2000 mA h).

Table 6 System Requirements and Verifications

While Idle	While Transmitting
20.0 uA - Wifi module in deep sleep mode [5] 0.75 uA - ATMega328p microcontroller [10]	14 mA - ATMega328p microcontroller [10] 1mA - Temperature sensor [2] 140 mA - Wifi module in 802.11g active mode [5]

If we make measurements and transmit for 10 seconds in every 5 minutes, the power rating per day is

$$24 \text{ h} \times (10 \text{ s} / 300 \text{ s} \times (14+1+140 \text{ mA}) + 290 \text{ s} / 300 \text{ s} \times (20 \text{ uA} + 0.75 \text{ uA})) = 124.48 \text{ mA h/day} \quad (1).$$

The module can run on a single charge of the battery for

$$(2000 \text{ mA h}) / (124.48 \text{ mA h/day}) = 16.07 \text{ days} \quad (2).$$

By wall powering the modules, we can increase measurement precision to one sample per second and remove the need to recharge batteries every two weeks.

5.3 Ethical considerations

Data security is of major concern in the days of massive security breaches. Since our product collects room temperature data, which can be considered private information, steps need to be taken to ensure that the data is secured before our product can transmit it, thus preventing compromising users' privacy. Currently, the data we transmit between each module is unencrypted. Thus, it poses potential risk to users' privacy. It is also important that our product offers protection against hijacking should third parties attempt to collect sensitive personal data without the consent of the owners. All of the precautions are in accordance of ACM code of ethics 1.6, which states computing professionals should

respect privacy[11]. Our responsibility to user privacy also applies to ourselves such that we cannot deliberately engineer vulnerability and/or back doors to our product so we can collect our users' data for our own benefits.

We are aware that similar product already on the market, i.e. a Nest thermostat[12], uses machine learning to improve user experience, and requires extra data like the owner's daily schedule to train. Depending on how protective a given user is about his/her personal data, he/she may not feel comfortable surrendering the aforementioned data. In contrast, our project has an advantage of not collecting any data other than temperature readings, thus appealing to the more privacy-conscious users.

The temperature sensor, the central hub, and the actuators are low power devices. Hence, there is very little risk of electric shock. Besides, they are powered by aftermarket, UL certified DC power supplies, so we are not under the risk of fire and/or electrocution. However, we discovered that the 5V linear voltage regulator we used generates excessive heat, and we are concerned that it can cause burning to the users if touched directly. We aim to solve this problem by designing a more efficient power supply so that less energy is wasted in the form of heat.

5.4 Future work

Scalability can be improved by either updating the ESP8266's firmware to support more concurrent connections or foregoing the ESP8266 all together and relying on a more performant WiFi server source. A typical router supports about 250 connected devices - this is more than enough for most cases. Another option would be to implement communication as a mesh network which enables message multi-hopping and thus a longer viable distance of sensors from the central hub.

User-friendliness is also crucial, as customers would never use our product if they couldn't easily install it. By placing each sensor in a well-design encasing, we can make the temperature sensor less of an eye-sore and can provide visual clues about how the sensors should be installed.

Along with user-friendliness, making the product more customizable is vital for ensuring our product can function in most home environments. For example, in a single large room we may want multiple air-vents hooked up to a single temperature module. Exposing more configurations in the front-end is easy given the code structure, so adding options to configure temperature reader/vent actuator pairing would solve this problem. Another addition which wasn't implemented would be to specify the currently set HVAC mode: heating or AC. This could clarify when a vent should open or close.

Accessibility to our product can be further improved by making each module smaller and cheaper. Optimizing our hardware components (e.g. switching from an ATmega to an ATtiny) could accomplish both of these tasks. Another option would be to use the surface mount version of the processor to reduce size.

Power consumption is another point of improvement, and can be accomplished by designing a better power supply circuit and configuring the microprocessor and WiFi module to go into sleep mode whenever necessary[13].

References

[1] "milesburton/Arduino-Temperature-Control-Library", GitHub, 2019. [Online]. Available: https://github.com/milesburton/Arduino-Temperature-Control-Library. [Accessed: 01- May- 2019].

[2] Datasheets.maximintegrated.com, 2019. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf. [Accessed: 01- May- 2019].

[3] "How to Connect an Arduino to the Internet with an ESP8266 Module | Kunz, Leigh and Associates", Kunz, Leigh and Associates, 2019. [Online]. Available: https://kunzleigh.com/how-to-connect-an-arduino-to-the-internet-with-an-esp8266-module/. [Accessed: 01- May- 2019].

[4] Espressif.com, 2019. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf. [Accessed: 01- May- 2019].

[5] Espressif.com, 2019. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. [Accessed:01- May- 2019].

[6] “Quick Links,” Raspbian. [Online]. Available: https://www.raspbian.org/. [Accessed: 01-May-2019].

[7] Andresvidal, “andresvidal/rpi3-mongodb3,” GitHub, 01-Jan-2018. [Online]. Available: https://github.com/andresvidal/rpi3-mongodb3. [Accessed: 01-May-2019].

[8] “Chart.js,” Chart.js | Open source HTML5 Charts for your website. [Online]. Available: https://www.chartjs.org/. [Accessed: 01-May-2019].

[9] “Free, open source, & modern CSS framework based on Flexbox,” Bulma. [Online]. Available: https://bulma.io/. [Accessed: 01-May-2019].

[10] Ww1.microchip.com. (2019). [online] Available at: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf [Accessed 22 Feb. 2019].

[11] “ACM Code of Ethics and Professional Conduct,” Association of Computing Machinery. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: 08- Feb- 2019].

[12]Nest Learning Thermostat, 2019. [Online]. Available: https://nest.com/thermostats/nest-learning-thermostat/overview/. [Accessed 30- April- 2019].

[13] "ESP8266 Power Consumption - ESP8266 Developer Zone", Bbs.espressif.com, 2019. [Online]. Available: https://bbs.espressif.com/viewtopic.php?t=133. [Accessed: 22- Feb- 2019].

Appendix A Requirement and Verification Table

Table 7 System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
Temperature Sensor Module 1. Read temperature with accuracy up to 0.5 ˚C within the range of temperatures 10˚C to 30˚C 2. send correct temperature data to WIFI server reliably within 5-25 meters of range indoors 3. have an LED indicator that shows the status (power, connection, etc.) of the module.	1. Set up a heating coil and place the sensor inside the coil; compare the readings against the coil temperature settings 2. Place the sensor 5-25 meters away from the WIFI server and send known test sequence to server 3. Test LED indicator is displaying correct information under the following condition: looking for connection, connected, and battery low.	1. N Unable to obtain tool; used an alternative way 2. N It can reach 50 meters line-of-sight but only 10 meters indoors. 3. Y
Central Hub 4. hosting a web service and sending alerts to the user through email or text 5. listen and store new temperature readings from all clients 6. powered with standard wall AC outlet	4. Periodically send signals to the central hub and confirm it received and stored them 5. Access the web service on another computer in the same network and read the displayed sensor values 6. Measure the DC output of power supply and check whether it is within (5V±2.5%, 2A±2.5%)	4. Y 5. Y 6. Y
Air Vent Actuator 7. receives signals from Central Hub Wifi Module with instructions 8. Air-vent opens/closes vent upon order and maintains that state until a new signal is received	7. Send mock signals through the central hub, and confirm they were received by the air-vent actuator 8. Simulate signal response for both opening and closing by sending mock signals to the actuator and confirming the correct action was taken	7. Y 8. Y