Final Report - RFID Attendance System

April 29, 2019
Github Repo: https://github.com/hbh7/RFID-Attendance-System

Introduction

In this project, we created a distributed RFID attendance system that can be used to take attendance for large classes or events. It can be implemented with many card readers so that all entrances to the location of the event can be covered. The underlying relationship between the devices is one Raspberry Pi acts as the server, and then at least one Pi acts as the client. In our project, we utilized 2 client Raspberry Pis. This relationship is similar to the Bluetooth connection model of master-slave, with the clients attached to Mifare RC522 card readers modules and the server controlling the attendance. Our goal was to create a full application for attendance recording. The application supports two modes: enrollment and attendance. New classes or events are able to be added to the system, and then they can be opened again at a later time to start attendance again. The application also allows for the viewing of attendance and of a server log which records all actions that happen within the system.

Hardware Implementation

The RFID attendance system is a fairly software-extensive application, however, some interesting hardware components are required to achieve the capabilities which our project supports. Our project currently utilizes two Raspberry Pi 3 Bs and a Raspberry Pi 3 B+, but any model 3 B or above is supported. The Raspberry Pis do not inherently support RFID reading therefore it must be added via some sort of external reader. For our project we attempted to utilize a RF IDeas PCPROX reader to read the 125Khz RFID tags which exist in every RPI student's id cards as this would be most applicable to a real world scenario which would occur at RPI, however, the TAs advised against reading RPI ids for this particular demo. Therefore, we went with the Mifare RC522 modules.

The Mifare RC522 modules were chosen for this project for a variety of reasons. The modules offers support for varieties of Mifare cards which are an extremely popular format. According to Wikipedia over 10 billion Mifare cards have been sold. More information about the mifare technology can be found here. The module also offered a low entry cost, currently selling for anywhere from $5 - $12 on Amazon depending on how many cards you want and the location which the module was manufactured. Lastly, the module offers extensive Raspberry Pi support via various python packages and is easy to wire to the Raspberry Pi due to the 3.3v power requirement which the Raspberry Pi supports without the need for any external power source or resistors.

For this project two RC522 modules were purchased as a proof of concept, but the software we supported supports many more modules. The modules were wired to the Raspberry Pi as follows, SDA was connected to pin 24, SCK to pin 23, MOSI to Pin 19, MISO to pin 22, GND to pin 6, RST to pin 22, and the 3.3v to pin 1. More info about wiring this module can be found here. The RQ pin is unused. The RC522 module is wired in this manner to allow communication with the Raspberry Pi's Serial Peripheral Interface (SPI). The SPI allows the

module to communicate the bits it reads from the card to the Raspberry Pi. More information about the Raspberry Pi's SPI bus can be found here. The modules were wired to both clients, but not the server as the server merely acted as a controller to process data in our application.

Software Implementation

The RFID attendance system relies on a plethora of different programming languages and technologies to allow for the features which our system supports. The program can be broken down into two parts, the client code and the server code. The client code utilizes Python to communicate with the RC522 via the Raspberry Pi's GPIO pins. The client code also contains a systemd script to allow for auto starting of services. The server code utilizes Node.JS, MongoDB, CSS 3, HTML5, as well as JQuery and Javascript to support the various features which we have implemented for this application.

Server Code

The server code for our purposes can be further broken down into the user interface and the backend. The backend is an Node.JS express application which serves the various web pages via port 3000. The Node.JS express application contains various API endpoints such that a user connecting to it either directly or via the web interface can execute tasks against the database such as retrieving a list of users who attenended during a certain time period, creating classes / events, retrieving a list of class, etc.

The Node.JS application also acts as the controller for the clients (the Raspberry Pi's which have readers and are running client code) to communicate with. The server provides various socket endpoints for the client to communicate. The socket endpoints supported by the server currently are getting the current class which attendance is being taken for, logging to the websites server log, taking attendance, and enrolling a user.

This particular socket implementation was done via Socket.io as it allowed for simple and secure communication between the client and server. Socket.io also allows for the number of clients to be scalable supporting upwards of 600,000 clients, however, we felt the raspberry pi would limit us before that limit was hit and therefore only tested it with around 50 "fake" clients to ensure the scalability was working. More info regarding Socket.IO can be found here.

The server code stores data such as enrolled users, attendance, and classes / events in a NoSQL database, in this case we chose MongoDB due to its fast processing time, easy Node.JS integration, and scalability. The database and Node.JS interact between one another via the mongoose node module which allows us for better relational support.

We created a responsive front-end for users to be able to interact with the attendance system easily. These pages were designed using HTML5, Bootstrap 4, and CSS 3. The home page offers many options, including Take Attendance, View Attendance, and View Server Log. If the user clicks on Take Attendance, they are then presented with 2 options: Create a New Class or Take Attendance for an Existing Class. If the user chooses to View Attendance, then they can enter a timeframe and a class for which they want to see attendance records. If attendees were not enrolled, then they will just be listed as "Unenrolled User." Enrolled attendees' names and RINs will be displayed in the list otherwise. The View Server Log is an interesting feature that will show all activity that the server recorded, even if it goes offline for an amount of time. It is important for users to be able to monitor the server at all times so that all cards can be

processed for attendance. We implemented alerts and other feedback-providing displays that will notify the users of all errors and successes when utilizing the attendance system.

Client Code

The client code uses Python and has 2 parts, the normal reader program and the enrollment program. The programs use multiple different modules to facilitate the communications it needs to, mainly the RPi.GPIO module to interface with the RFID module over the GPIO pins, the mfrc522 module to communicate with the RFID module, and the socket.io library to communicate with the server, plus a few extra modules like time, random, and sys.

The reader program is broken up into a few parts, mainly the read function, the transmit function, and the main function. The main function checks that you provided the program with a server address to connect to, and creating a separate thread to manage communications with the server and transmitting the learned IDs. It then goes into a loop to facilitate repeatedly calling read on a loop, and if it reads data, then adding it to the queue.

The reading function creates a reader object based off the mfrc522 module. It then tries to read the ID and text from the card, and if it gets data, it prints it and returns it to the main function.

The transmit function runs in its own infinite loop, constantly testing if the queue has any items in it. If it does, it attempts to connect to the server specified in argv[1]. If the connection is successful, it then loops through the queue and transmits each ID to the server. It then waits for a response, and if it receives the ID back, it then removes it from the queue. This happens until the queue is empty, at which point it then goes back to just checking if there's anything in the queue, and waiting until there is to do anything else.

The enroll program does nearly the same thing as the reader program, but with the loops and functions omitted and the functionality fine tuned. On startup, it checks to make sure the server address was given, and then attempts to connect to it. If it cannot connect, it exits, but if it can, it then goes into a loop asking for information about the person to enroll in the server database. This includes the attendee's name and RIN. It then asks for their card to be scanned, and upon doing so, it links the information together and tells the server of the new linking. This process then repeats until the user types in EXIT.

Both programs secure their transmissions by using HTTPS SSL encryption where the clients have a certificate and key file, as well as the server certificate to verify the connection is to the correct machine and not an imposter. In the real world, these would be regenerated by the person setting up the system for added security rather than using the ones provided in the repo, however for testing, this is adequate.

Obstacles

Our project incorporates many different technologies and therefore was bound to run into a few obstacles along the way. The two main obstacles we experienced were with the Pi's WiFi chip and the RC522 readers. The Pi's WiFi chip is very weak and therefore when set to transmit rather than receive it was highly susceptible to interference. Moving our laptops around (which

have metal casings) was enough to completely block the signal. The Pi was also unable to perform at all in the RPI Student Union due to the high level of interference. In a permanently deployed scenario it is recommended that the Pi act as the server and an external access point / router handles the network part of the application. The second obstacle which we encountered involved the speed of the RC522 readers interacting with the Python library. In order for the card to be considered a valid format the library does some verification to ensure valid data exists in the correct sectors on the card. Scanning cards had to be restricted to about once per second per reader in order to allow the library proper time to validate the sectors without throwing invalid card errors. In practice we found this to be a nonissue as by the time we switched hands to scan the other card the reader was ready to scan again.

Future Plans

In the future, we would like for our system to work on actual RPI IDs, so that students would not have to hold a separate identification card as the current readers are not equipped to read RPI IDs. We would also like to implement some sort of feedback for towards the user when scanning their IDs such as a beep similar to that used when you scan into a building on campus. Lastly, it would be interesting from an analytical standpoint to allow for the viewing of trends per user per class or for the entire class. This was a lot more difficult to implement than expected in the short amount of time and therefore was left as a great feature to add in the future.

Conclusion

Overall, the project set out to create what was intended of it: A Master-Slave type of model that allows the distribution of readers throughout a venue or classroom which would communicate the attendance to a separate server. This was all implemented independent of existing infrastructure which allowed for easy deployment. We also succeeded in creating a nice front end user interface which increased the usability of our application for the end user who might not be as tech-savvy. Lastly, we succeeded in implementing client-side caching to prevent a single point of failure (the server going down) causing us to lose all the attendance data for that period. While this project was very challenging, we were able to implement just about everything we planned.