StenoPhone

Author: Mitchell Yang, Ellen Seeser, Cambrea Earley: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of real-time generation of meeting transcripts and transmission of audio between conference rooms. During a meeting, microphone arrays interfaced to Raspberry Pis communicate over the internet with a centralized server to generate speaker-identified transcripts and relay the audio to the other rooms.**

***Index Terms* — Application Layer Networking, Audio Processing, Audio Streaming, Direction of Arrival, Machine Learning, Microphone Array, Speaker Identification, Speech to Text, Voice Over IP, Web Conferencing**

I. Introduction

In recent years, technology has facilitated a rise in the prevalence of geographically distributed workplaces. Even within a single team, one portion of the members may work in one city, a second portion in a second city, and so on. Responding to this phenomenon, the corporate world has adopted web conferencing tools such as Zoom, Microsoft Teams, and Google Meet. However, these solutions are optimized for the experience of individuals calling other individuals. With StenoPhone, we present a solution for audio-based web conferencing that's tailored towards the record-keeping and communication needs of semi-decentralized teams. Consisting of a wireless microphone and an associated web application, the product provides audio transmission and automatic transcription that includes the identification of speakers. StenoPhone supports multiple users per microphone and multiple microphones per meeting.

The goal of this project is to provide intelligible and low-latency audio transmission, as well as fast and accurate transcription, to the end users. The system must provide end-to-end audio latency of less than 150ms while maintaining a dropped audio packet rate of less than 5%. The system must also provide transcription of audio in less than three seconds, and the transcript's error must not exceed 25%.

II. Design Requirements

StenoPhone is intended for use in a conference room setting. Based on measurements we have taken during a survey of conference rooms, we define a conference room as a room with maximum dimensions of thirty-five square meters and featuring a large table. With the microphone placed in the middle of the table, the system is tested with participants sitting or standing no more than two meters away, in any direction around the table. We test with a maximum of three users per single microphone and a maximum of two microphones per meeting.

As described in the Introduction, the latency of audio across the system, also known as mouth-to-ear (M2E) latency, must be less than 150ms. This is a standard commonly accepted as maximum M2E latency in IP telephony; a round-trip delay greater than 250ms is noticeable to the users [1]. This latency may be tested by directing a packet from one microphone to the web server and back to the same microphone, capturing self-consistent timestamps at the beginning and end of this process. The second requirement related to the user's audio experience is that of limiting the percentage of dropped packets to below 5%. IP telephony theory again advises this rate to avoid audio distortion experienced by end users [2]. The dropped packet rate may be estimated using a long transmission of a known number of packets to the webserver and then to an end microphone-speaker, which can count the number of packets received.

Requirements pertaining to the speed and quality of transcription are also essential to meeting the goals of our project. The 3s transcript latency, also known as average word delay, that we require is a measure that comes from FCC guidelines pertaining to live closed-captioning [3]. This latency may be tested by comparing the time at which a spoken phrase was captured by the microphone to the time at which a packet containing the transcript arrived at the browser of the end user. This may be done manually.

The accuracy of the transcript can be measured with several different error rates. The first, word error rate (WER), measures the accuracy of the speech to text element itself. $WER = (S + I + D) / N$ where S is the number of erroneous substitutions, I is the number of word insertions, D is the number of deleted words, and N is the total number of words in the original text. The accuracy of the speaker identification system is measured by speaker identification error; because the participants in our system are known, this metric is simply computed as the number of misattributed words divided by the total number of words spoken. We require that neither WER nor speaker identification error rate (SIER) exceed 25%; research has shown that users find transcripts with error up to and including this rate to be useful [4]. We've also identified formatting errors that can arise from the combination of transcripts from multiple microphone streams: the chronology of the combined sections may be incorrect, or attribution may be missing or incorrect upon a switch between streams. A rate of these formatting errors greater than 5% will not be acceptable.

Similar tests may be used to measure the three aspects of transcript accuracy; all three tests require a 'known text.' A text may be considered 'known' if it is either predetermined before being read aloud or recorded as it is spoken. These texts remain constant over multiple tests; they consist of common English words formed into sentences, and they should be at least 100 words long. To test WER, a known text is spoken into the microphone; the transcript and the correct

text are compared, and WER is calculated. To test speaker identification error rate (SIER), a known text that features speaker changes and speaker movement is spoken into the microphone; the transcript and the correct text are compared, and SIER is calculated. To test formatting error rate (FER), a known text, one that features speaker changes where the speakers use microphones in different rooms, is spoken into the microphone; the transcript and the correct text are compared, and FER is calculated. Additionally, we test the three error rates in the situation defined by one group of at least two speakers using one microphone and another speaker using a second microphone.

Finally, the website will be required to provide certain functionality to the user: creating meetings, joining meetings, adding microphones, and viewing transcripts of current or past meetings.

server, to which are connected an arbitrary number of audio devices and web browsers.

Audio data originates at a microphone array, which performs some processing in hardware before passing the audio to a small WiFi-capable computer device. In our original design, we planned for audio processing (noise removal and voice detection) to occur on this device; this has been removed due to latency constraints and an already high quality of audio. A networking module then sends the audio information to the web server. Additionally, the web server sends audio from other microphones to each individual audio device within the same meeting, allowing the audio to be played on an attached speaker.

The networking component of the web server receives audio packets from the connected microphone devices. This audio is both sent out to other microphones and sent to the transcript

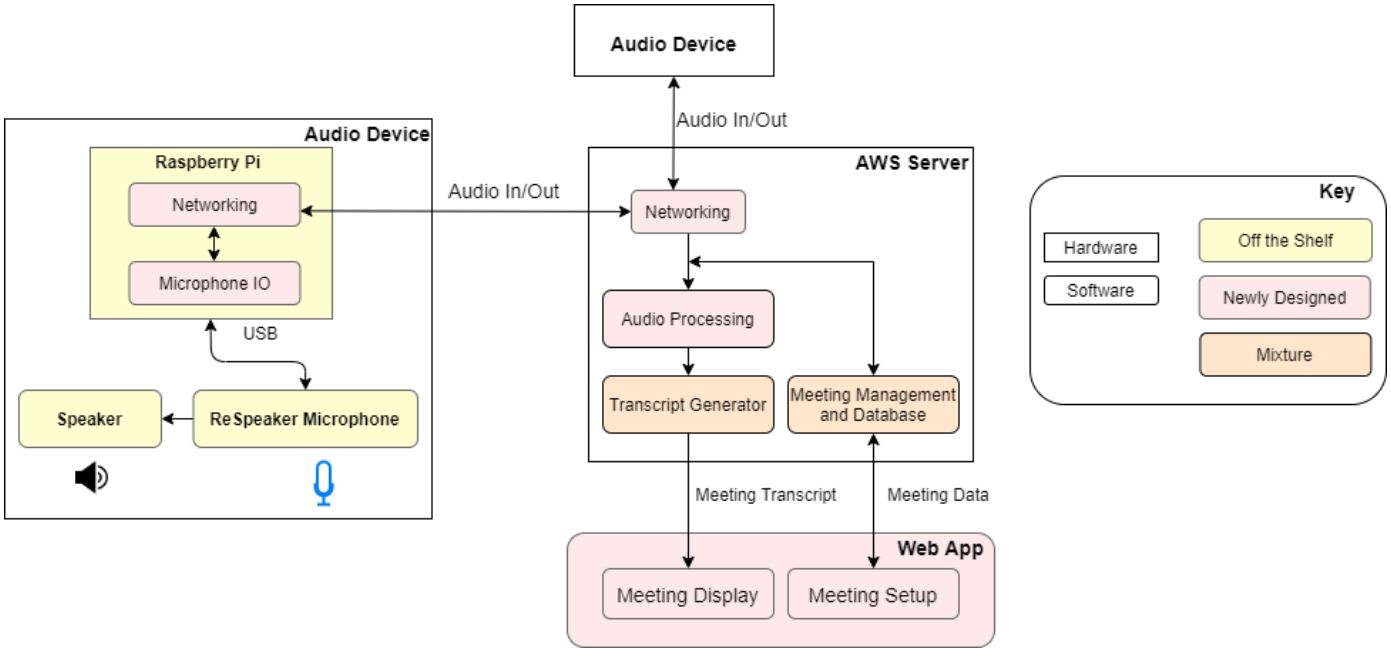

Fig. 1. System Data Flow Diagram

Table 1: Summary of Requirements

Requirement	***Metric***
Audio Transmission Latency	Mouth-to-Ear Latency (ms) < 150 ms
Audio Quality	Dropped packets (%) < 5%
Transcript Latency	Average Word Delay (s) < 3s
Transcript Accuracy	Word Error Rate (%) < 25%
Speaker Identification Accuracy	Speaker Identification Error Rate (%) < 25%
Formatting Accuracy	Formatting Error Rate (%) < 5%

III. Architecture and/or Principle of Operation

Fig. 1 shows the flow of information through our comprehensive system, including hardware and software modules as indicated. The system consists of a single web

generator module, where audio is preprocessed before speech to text and speaker identification processing are performed with the assistance of machine learning solutions. Speakers are identified from a pool of users who identify themselves to the microphone during the process of adding a microphone to a meeting.

Transcripts are compiled from multiple microphone streams by the meeting manager; adding transcripts to the database allows them to be viewed by users in the web application. Web application users can also create meetings, join existing meetings, and connect a microphone to a meeting on the website using a setup process. Users who join meetings, whether or not they connect a microphone, will be able to view the meeting transcript, whether or not the meeting is ongoing.

IV. Design Trade Studies

A. Audio Device: Microphone Array

The microphone array is one of the main advantages of using the StenoPhone over other existing conference room meeting products. For this microphone we needed to choose one that would work for our context; ideally it would sit in the center of a conference room table and detect audio from all surrounding angles. We were able to eliminate many microphone arrays that do not have this capability.

We then focussed in on the circular ReSpeaker Microphone arrays. These arrays have microphones placed in a circular formation for collecting audio surrounding the device at 360 degrees. Out of the ReSpeakers we decided to use the Mic Array v2.0. We found that this array had higher quality audio than the other microphone arrays. It also has an on board microcontroller IC, XVF-3000 from XMOS. This chip handles processing algorithms that come with the board, such as automatic gain control and detection of direction of arrival of audio.

The other option of ReSpeaker that we ultimately decided against was the ReSpeaker Core v2.0. This mic array has similar capabilities to the Mic Array v2.0 but it also has its own Linux system on the board with WiFi capabilities so that the microphone array functions as a single device without the need for a Raspberry Pi or computer to help with data transfer. Though this board is more powerful we decided against it since at that time we preferred to have a Raspberry Pi connection to the speaker to handle more processing of the audio and to package the data to send to the server over the network. The ReSpeaker Core was also a more expensive choice and seemed to be a higher level product which leaves less opportunity for us to configure and build upon the given microphone implementation for our specific project needs.

B. Audio Device: Computer

For the audio device, we needed to have a main system for processing the audio and sending and receiving data over the network. For this job we had two main choices, the Jetson Nano or the Raspberry Pi 3 B. We first started by looking at how much processing we would need to do on the device before we sent it over the network and thought that we mainly wanted to filter the audio, detect direction of arrival, route output audio to the server, and route input audio to the speaker. This design of the software allows for most of the long processing for both the speech to text and speaker identification solutions to be performed on the server which greatly reduces the amount of work that the audio device needs to perform locally. The specs for the Jetson Nano are 1.42 GHz processor, with 2GB RAM. This is comparable to the Raspberry Pi 3 B which has 1.4GHz processing speed, with 1GB RAM. Both of these devices also offer USB connection which we use to connect our microphone array and speaker.

A huge drawback of the Jetson Nano is that it does not have WiFi capabilities on the board, and we would need to buy an external WiFi connection device to use with the Jetson Nano. Though the Nano does have an Ethernet port, we were planning on using WiFi for our project since in conference rooms there are not always Ethernet connections available. The Raspberry Pi 3 does have dual-band WiFi capabilities and comes at a lower price than the Jetson Nano with comparable specifications for processing speed making it the best choice for our project.

In our original design, we intended to process the audio for noise removal before streaming it to the web server. We planned to queue incoming audio from the microphone as it waited to be processed. However, we found that this processing introduced unacceptable, audible inter-packet delays that made audio sound choppy. Because the audio without noise suppression already sounded quite clear, and introducing processing made the audio worse, we took the processing off of the audio device; some still occurs on the web server.

C. Web Server: Server

We chose to use a server for the site for this project's more complicated processing, such as speech to text and speaker identification ML. These components have higher latency so we decided to use the more powerful server to process this instead of processing this on the Raspberry Pi. Another reason why we chose to use the server is because we have structured our audio streaming system to be centralized, so each audio stream travels through the centralized server. This allows for us to quickly have access to all audio streams on the server so that we can generate the transcript in one place.

Amazon Web Services (AWS) was the obvious option for our web deployment because of its support from the course staff in terms of both knowledge and funding. To choose a specific Amazon EC2 instance we started by narrowing it down to the M5, M5a, and M4 classes of servers. We know that our system has audio streaming both to and from two devices potentially simultaneously (consisting of both audio and metadata at up to 44.1kHz with 16 bits per sample) as well as connections to a website, so we needed to focus on the network bandwidth with a lower limit of 3 Gbps. We also are running the speech to text and speaker identification ML, so we needed a balance of CPU power as well.

Comparing the EC2 servers we found that M5 had the highest processing speed 3.1 Ghz, with comparable network bandwidth up to 10 Gbps, but was the most expensive. The M4 had the lowest processing speed at 2.3 GHz with network performance defined as "moderate" and was the lowest cost. The M5a was right between these two servers with 2.5GHz processing speed and up to 10 Gbps network bandwidth with economical prices. We ultimately decided to use the M5a EC2 instance; we have run several ping tests on this server as well, to test the network. We found that hosting the server in the Ohio AWS Region resulted in the lowest latency from our location in Pittsburgh.

D. Web Server: Networking

In Fig. 1 we show two types of connections that need to be

made by the server: the first is the connection to the audio devices, and the second is the connection to the website. Both of the connections are handled in different ways with consideration to the type of data being sent and to the latency bounds of each.

For the connection to the audio devices, we are sending the audio, direction of arrival metadata, and the microphone ID so that the server can distinguish where the data is coming from. The server handles sending one copy of this data directly to the other audio device for audio streaming. The other copy of data is sent to the speech to text and speaker identification systems. It is important that this connection has very low latency since the audio stream output must appear at the second audio device in real-time and be intelligible to the users there. To easily measure the latency, we first did a ping test. Ohio, us-east-2 region, is consistent at around 42 ms, median 41ms. North Virginia, us-east-1, spikes at beginning with around 550 ms and decreases to 27-32 ms and stays there, median 32. We like consistency so are choosing to use the Ohio region, as latency spikes can cause inconsistent and poor audio.

We decided to use a UDP connection for our implementation since this connection allows us to send our data with low latency; we also found that UDP connection is common with real-time audio transmission. One issue that this connection could add to our project is that UDP allows for packets to be dropped on the network. Because of this we have made a requirement for less than 5% packet drop rate of our system to ensure that the audio stream is intelligible to the users. Since packet size plays a role in how many packets can be dropped on a network, we can tune the packet size to help enforce this requirement.

The next connection that the server has is the connection to the HTTP website. To transport the transcript from the server to the webpage, we used Django Channels to communicate with the websockets on the user's webpage hosted using ASGI using a docker run REDIS server on Apache 2.

E. Web Server: Database

The server's information is to be stored using a SQLite database. For the database, we have identified other solutions such as MySQL and PostgreSQL. A SQLite database was chosen as it is portable and that we would rather spend server cycles in processing than using a DBMS which requires an additional server process like MySQL or PostgreSQL. In addition, most of the website development will be done locally, so it is convenient to have the database contained and stored in the repository for ease of sharing and use.

F. Web Server: Speech to Text

The speech to text portion of the system is implemented by integrating a preexisting machine learning solution into the transcript generator software module. Tradeoffs in this implementation came down to the tradeoffs between different solution options.

Solutions that have 'streaming' constructs were automatically favored over others. These tools allow additional audio content to be appended to previous audio with the understanding that the new audio could be a continuation of the previous sentence or word. It's very helpful for our system since we receive small amounts of audio in each packet over the network, and these vocal fragments probably wouldn't make much sense if transcribed individually.

We identified three reputable speech to text solution options that contain streaming tools: Mozilla DeepSpeech, which is a software package; CMU PocketSphinx, another software package; and Google Speech-to-Text, which is a paid service. Because of our requirements associated with transcription, these options should be compared based on their latency and their accuracy, measured by word error rate (WER). DeepSpeech reports 7.5% WER [5]; PocketSphinx reports 10% WER [6]; Google reports a very low 4.9% WER [7]. Self-reported WER and timing information for each of these solutions, however, are not directly comparable; the best results on various different datasets are going to be reported.

In our implementation process, we found that Google Speech-to-Text was the only one of the solutions considered that provided us reasonable WER results. The latency was also low, so we naturally adopted the service.

The second ML element of the transcript generator was the speaker identification module. We had the option of either seeking out another open source software package, as was originally planned in our design, or using Google Speech-to-Text for this as well. Two factors influenced our decision to stick with the latter option. First, none of the packages we found included streaming constructs, which greatly complicated the implementation and accuracy prospects. Second, the Google product's speaker ID capability was already integrated with its transcription, making it possible to obtain the two results from the same single function call. There was no noticeable latency increase, and this decreased the number of threads on the server.

One additional design element of the transcript generator was audio preprocessing. Lowpass filtration was done to remove unnecessary elements of the audio before it was passed to the speech to text portion. This was done in scipy for ease of implementation.

G. Web Server: Website

The interface for users is a website served through AWS using the Django Framework. Django was chosen as it is well established, so resources are readily available if Django specific problems arise. It is also based in Python, so it is easier to integrate with other Python libraries like the ML solutions mentioned above.

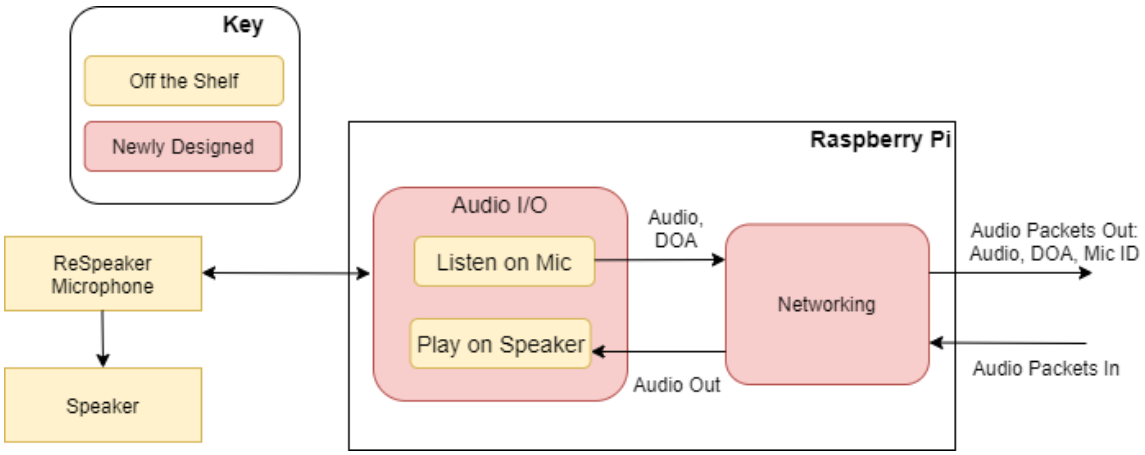

Fig. 2. Audio Device Diagram

V. System Description

A. Audio Device

The main function of the audio device, shown in Fig. 2, is to handle input and output audio. The 2 components to this are the direct Audio I/O to the microphone and speaker and the networking layer to transmit audio to and from the server. Audio input and output streaming happens on its own thread so that the networking component can keep working simultaneously. Each audio device has its own microphone ID so that users can register their microphone when they create and join meetings. This microphone ID is also used in the transmission of audio data, further discussed below.

Starting with the control flow of audio from the ReSpeaker mic to the server: the users will speak into the microphone; this audio is streamed, using the pyaudio streaming library, audio is sampled at 16KHz; and it's put into a queue of audio ready to be sent over the network.

The networking layer will read from the queue if there is an audio packet ready to send to the server. This packet contains the audio segment, direction of arrival information, and the microphone ID. We are using a UDP connection for this transmission to ensure low latency.

As for information coming from the server to the speaker, this consists of audio from users using a different audio device. The server sends the audio packet to the other audio devices in the meeting. The software on the Raspberry Pi will play this audio on the speaker connected to the mic array. The ReSpeaker mic array will block listening while audio is played on the speaker to prevent a feedback loop here.

B. Web Server

Two separate connection types are used on the AWS Server, shown in Fig. 3, since we need to connect to both audio devices and the website. First we discuss the connection to the audio devices.

We use UDP to send and receive the audio from the devices; this server operates on a single thread and performs both of these actions. One stream of audio is from one audio device to the other, for example, this audio is spoken by users in room 1 and will be heard by users in room 2. We also make a copy of the input audio stream to send to the speech to text and speaker identification ML portion of our project. To do this, we use separate queues to store input audio data, so that the server can continue sending and receiving data on its own thread, and will not be blocked by the transcript generation. One queue holds audio meant to be sent out by the server - this is audio that will be sent to an audio device to be played to the users. The other queue holds the audio packets that will be used by the transcript generator. We use queue implementations from both the python queue library and the python collections library. The queue library provides a thread safe queue that is used to store audio packets that are sent to the transcript generator, the collections library provides the queue, specifically the library is called deque, that is used to store the packets of audio to be sent to the audio devices. This "deque" operates in a single thread that is running the networking and is faster than using the queue implementation from the library Queue.

Packets headed for the transcript generator are dequeued from the server and placed in separate buffers depending on the microphone ID associated with them. Audio from a given microphone will accumulate in these buffers for a maximum of 1.5 seconds before being combined and sent as an audio file to the transcript generator. Accumulation may be cut short if the direction of arrival information in the packet differs significantly from the direction of arrival (DOA) associated with the other audio in the buffer.

The transcript generator software module receives short audio files from the networking layer. A microphone identifier and the audio's direction of arrival additionally accompany the file. Locking ensures that only one transcript generation task per microphone is active at one time. The transcript generator keeps a dictionary of information pertaining to each active microphone connection, including the timestamp of the last time audio from this microphone was processed and the DOA of that audio. These two pieces of information help with the transcript generator's first task: speaker change detection. A

speaker change is detected when the DOA of the new audio differs from the old DOA by more than ten degrees *or* it has been more than five seconds since the last audio. When a speaker change is detected, the ML submodules are reset so that the new audio is considered independent from the old audio; otherwise, if no speaker change is detected, the new audio is appended to the old preexisting stream of audio.

Before audio can be sent to speech to text and speaker identification, though, some preprocessing is performed on the file. First, a low pass filter with cutoff frequency of about 7kHz is applied. Human listeners may find high frequency audio information useful or interesting, but the machine learning models have less use for the higher frequencies. This filter is implemented as a fifth order Butterworth filter using scipy. The scipy package is also used to resample the audio to 16kHz if it is not already that frequency. Then, execution is passed to the speech to text component.

During the normal course of the meeting, speech to text and speaker identification proceed on the same thread. The speech to text submodule has its own dictionary containing references to state information for each active microphone connection. Given the new bit of audio, the speech to text submodule may return new words to append to the transcript, or it may defer until it receives more audio. Each new word has an associated speaker tag prediction; by analyzing the speaker tags for every new word in the output, the module makes a speaker ID prediction and assigns a confidence to that prediction. The speaker tag prediction is simply the speaker tag that is predicted most frequently among the new words. If there is only one new word, the confidence is 50%; otherwise, the confidence is calculated as the number of instances of the predicted speaker tag divided by the total number of new words.

The speaker ID prediction is further augmented by DOA information about the audio. If the DOA matches, within ten degrees in either direction, the recorded location of any speaker, the ML prediction is ignored in favor of that speaker. If the speaker tag prediction does not match the known speaker tags of any of the participants, the speaker with the closest recorded location is chosen. If the reported confidence of the speaker tag prediction is less than the "low confidence threshold" of 75%, the speaker prediction is chosen from the two best speaker tag predictions based on which speaker's recorded location is closest to the DOA. In cases where the confidence is high - greater than 80% - the speaker associated with the speaker tag has their location record updated to match the audio DOA.

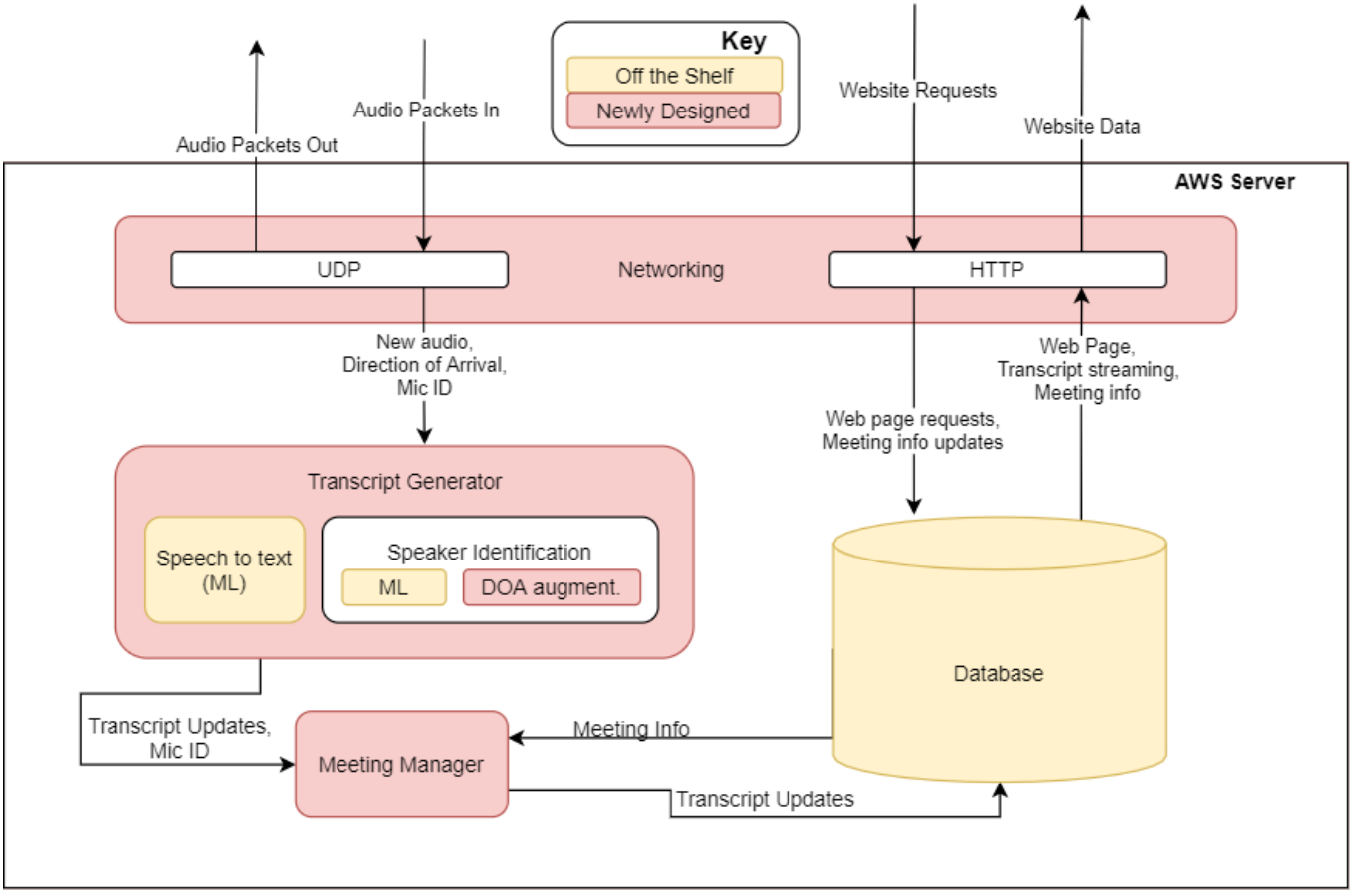

Fig. 3. Software Diagram

If the transcript generator receives new words from the speech to text submodule, it passes the new words, speaker prediction, and speaker change status to the meeting management module. During the normal course of a meeting, the meeting manager's job is to compile the official transcript. After receiving new information from the transcript generator, the manager looks up the meeting that this microphone is a part of and places a lock on that meeting's transcript so it can't

be modified by two threads simultaneously. Then, the manager detects microphone changes—if the current end of the transcript in the database does not match what the manager remembers having written last from this microphone, then a microphone change is detected. In the case of a microphone change, a speaker change reset is sent to the transcript generator of each microphone in the meeting; also, the speaker change status of the new audio is considered to be 'true.'

Finally, the meeting manager modifies the transcript to reflect the new information. If the speaker change status is 'true,' then the new speaker and speech content are appended to the end of the transcript. Otherwise, only the new speech content is added.

Next, a connection must be maintained with the website. We send the transcript, including live updates, using channels for the users to view during a meeting. This connection is HTTP

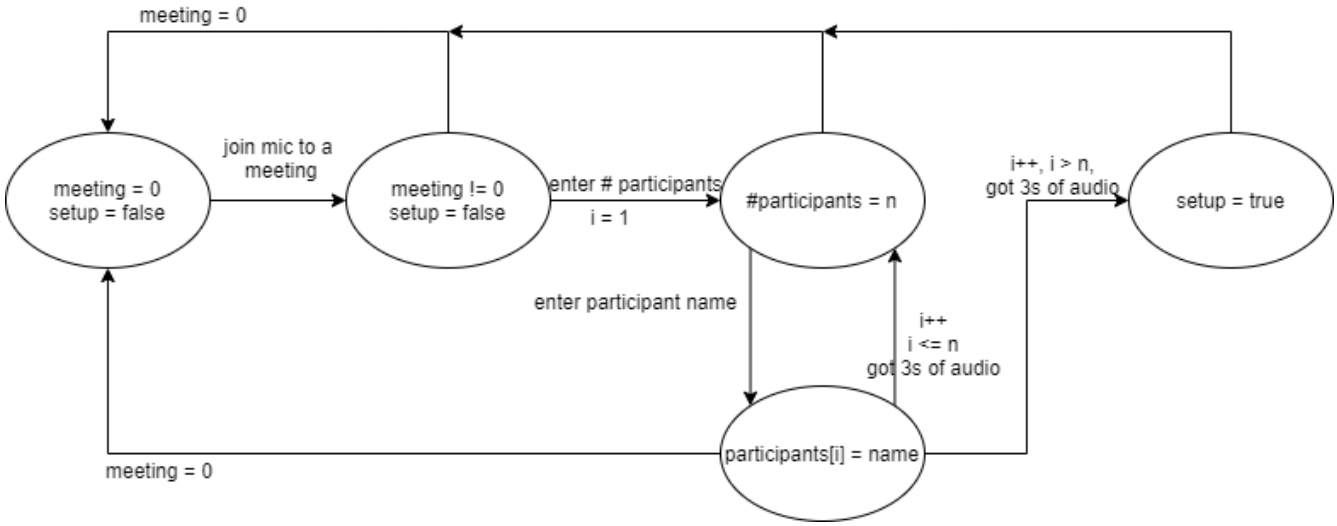

Fig. 4. Meeting Setup Flow

so that we can ensure packets arrive at the website and are in order. Here, we have two different types of data being sent. One is the transcript that is generated at the time of a meeting, transmitted with web sockets. The next data contains the meeting information, since the meeting database is stored on the server.

The type of data received by the server on this connection is meeting updates such as the joining, starting, or closing of a meeting. The updates are handled by the database and meeting manager.

When a microphone is added to a meeting, a setup process is followed in order to improve the transcript quality and help the speech to text submodule make good speaker identification predictions. A description of the setup state flow is shown in Fig. 4. During setup, transcription is not performed; a record of audio is compiled for each speaker and is used to initialize the speaker identification module after the setup process is finished. The setup process also allows the transcript to learn the name of each speaker.

VI. Test and Validation

A. Results for Audio Streaming

We had 2 separate requirements for Audio streaming. The first was that the mouth to ear (M2E) latency must be under 150ms, and the second was that the dropped packet rate should be under 5%.

For the M2E latency test, we measured the round trip time of the audio from an audio device to the server and back. We used one device for the test, sending 500 packets of audio to the server and comparing the timestamp of when they were sent to the timestamp at which they were received back at the Raspberry Pi. Our first implementation of the audio streaming included further audio processing on the Raspberry Pi before the audio was sent to the server. After 3 trials of testing we found this implementation to have latency values of about 115ms; this is within our threshold of 150ms. There was a drawback of this implementation however that the packet loss was >25%, which is significantly out of range and led to choppy audio output. This is due to the fact that we were running too many threads in our system to first get audio from the device, to then process the audio, and to finally send the audio over the network, that our system had a large latency. This meant that the networking portion of our system could not pick up packets off the network quickly enough, causing the buffer intended to hold incoming packets to become full. At this point many packets were then dropped from the network.

For our final version of the system, we changed our implementation to instead process audio exclusively on the AWS server so that the Raspberry Pi would only need to handle audio networking and playing and recording audio on the ReSpeaker. To test our latency of the final system we ran the test for 3 trials and found that the M2E latency was <= 32ms, which meets our requirements. Fig. 5 shows the comparison of the network latency results between the system with audio processing on the Raspberry Pi vs. with audio processing on the AWS server. We have also shown the average line of the data in this graph as well. In Fig. 6 we show the comparison of the dropped packet rates during the

latency tests between the system with audio processing on the Raspberry Pi versus audio processing on the AWS server. We gathered this data to correspond with our latency test values by calculating the percentage of packets sent during the latency test that were received back at the Raspberry Pi. We used the values from our latency test to calculate the dropped packet rate here, so that we can show how the latency tests results correspond with the dropped packets rates and subsequent choppy audio of our first implementation versus our final implementation.

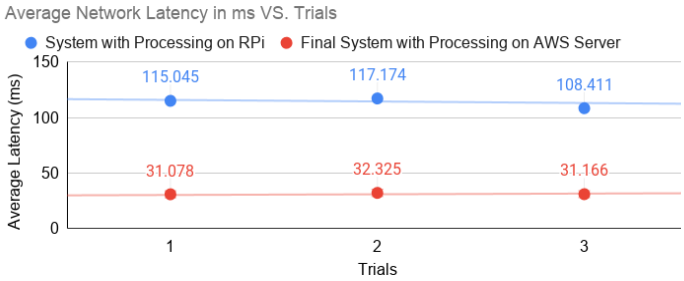

Fig. 5. Network Latency Result

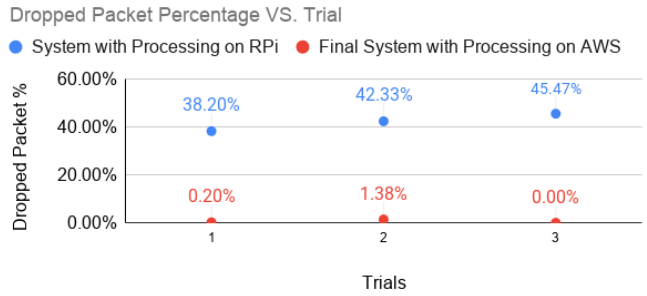

Fig. 6. Corresponding Dropped Packet Rate of Latency Test

To comprehensively measure the dropped packet rate of the final system we also use one audio device to send packets to the server, and we have the server send those same packets back to the audio device. We count the number of packets sent to the server and the number of packets received back at the audio device for a 10 minute transmission. After three trials of this test we found the dropped packet rate to be about .3%. This meets our requirement of having a dropped packet rate of below 5%. Both of these tests are also summarized in Table 2.

B. Results for Transcription

As described above, four separate requirements dictated our transcription performance, with at least one testing method assessing each in the finished system.

Transcript latency is the first transcription-related requirement. An average word delay of less than three seconds was never going to be difficult to meet, but it was going to be difficult to test. To gather data we took video of the transcript updating itself in a browser while the conversation was also recorded as it was spoken. The latency for each word of the conversation was then calculated as floor(second at which word appeared in browser) - floor(second at which word was spoken). Around 100 samples were analyzed in this way, and the results were very consistent; most words took about two seconds to arrive in the browser. This was unsurprising because packets might be held back and accumulated in the UDP server for up to 1.5 seconds, and the speech to text component might add additional latency. A histogram of the results is in Fig. 7 below. Since almost all packets took one or two seconds to arrive, the average word delay was 1.8 seconds.

TABLE II. Test Results

Requirement Metric	***Test***	***Results/ Tradeoffs***
Audio Transmission Latency (ms) < 150 ms	Route audio packets through server to find round trip time. ~1500 samples.	Tradeoffs in amount of processing done Latency <= 32 ms (>100ms w/processing)
Dropped packets (%) < 5%	Count original and final number of packets after transmitting. ~50k samples.	Tradeoffs in amount of processing done Dropped packets <= .3% (>25% w/processing)
Transcript Latency (s) < 3s	Compare time word spoken to transcript arrival in browser. ~100 samples.	Average Word Delay <= 1.8s
Transcript Word Error Rate (%) < 25%	Check transcript for word error after speaking a known text. ~500 samples.	Tradeoffs in paid services vs open source packages Average WER: 18% (>25% w/open source)
Speaker Identification Error (%) < 25%	Check transcript for ID error after reading known text with speaker switches. ~500 samples.	Average Speaker ID Error 9%

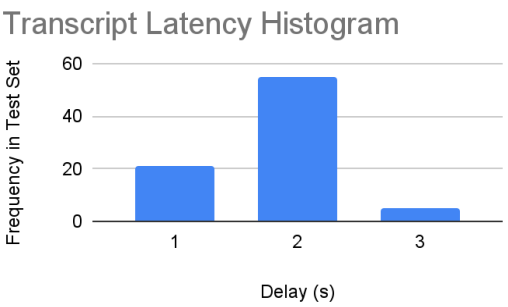

Fig. 7. Transcript Latency Data

The other requirements have to do with transcript accuracy; and the first of these is the word error rate. This is highly dependent on the machine learning model or service we decide to use for speech to text. We compared multiple options early on in development; out of the three we considered, Google Cloud Speech to Text was the only one that appeared capable of reaching a less-than-25% WER. Mozilla DeepSpeech's WER performance on our sample recording was always greater than or equal to 25%, and CMU PocketSphinx's was worse. So, we leaned in to Google Cloud and were not let down by the results from the full system tests. Our test text was a fake meeting script we wrote ourselves; it was 162 words long and contained very common words. We recorded the transcription results of this script in multiple scenarios, including three participants each reading different parts into the same microphone, one speaker reading the whole script, and three participants reading different parts to different microphones. We identified and counted the word errors in each transcription result. Having taken around 500 total samples, the average WER that resulted was 18%.

The speaker identification subsystem was developed *after* the speech to text subsystem, at a time when we were already optimistic about the results from Google Cloud Speech to Text, which also offered speaker identification in its speech to text results. The decision to use this for speaker ID has already been described above; no other alternatives were implemented. We did go through a long period of iteratively debugging our assumptions about the locations of participants and how the system responds to the participants moving around. Again, we took around 500 samples and measured speaker ID error as the number of misattributed words divided by the number of total words. In tests where speakers moved, the error was closer to 12%, higher than the total average of 9% error.

VII. Project Management

A. Schedule

As seen in Fig. 9 and 10 at the end of the document, our project schedule was divided into four phases. In the first phase, system setup, we ordered parts, configured the hardware we obtained, configured our web server, and set up some initial speech-to-text functionality. Phase two was focused on the backend of the product that the user would not generally see. The networking layer both on the Raspberry Pi and the web server was written. Non-moving transcriptioning, database and website backend, and filter design were also on the todo list for this phase. The third phase, then, was the front-end phase. The meeting setup and transcript streaming web pages were developed. Transcription capability was enhanced as well. Finally, the fourth phase left multiple weeks at the end of the project to account for system testing, making the site look better, and miscellaneous improvements such as tweaks to the speaker ID system. Integration occurred at the end of phase three, immediately before the interim demo deadline. Testing of submodules was continuous through development.

B. Team Member Responsibilities

Fig. 8 shows the general distribution of responsibilities among team members. Mitchell had prior experience with websites from his research; Ellen had used machine learning packages at previous jobs; and Cambrea had the most knowledge about hardware setup from the classes she had taken. The remaining tasks outside of these were divided equally between the members.

C. Budget

Our main source of spending was the ReSpeaker Mic array since we needed to order 2; this totaled to $158. Fortunately,

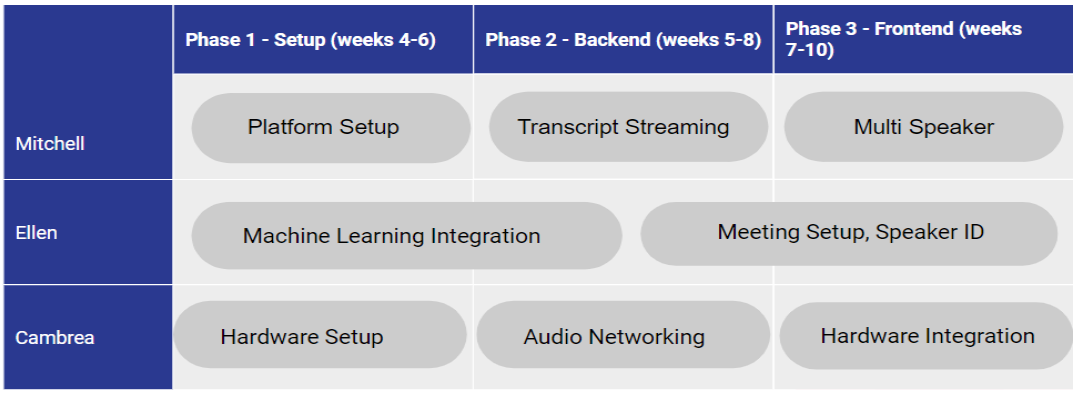

Fig. 8. Team Member Responsibilities

The formatting errors we looked out for had to do with switching between microphones in the same meeting. We never encountered any of this type of error; in section X we'll discuss what might happen if more than two microphones were introduced. It is also important to note that all transcript accuracy metrics are derived from testing both on one microphone and on two microphones in the same meeting so we could ensure the requirements held for both scenarios.

we were able to get the other high priced items from inventory, borrowing a JBL Speaker and a Raspberry Pi 3 B. Our budget and our table of parts can be found in Fig. 11. and Table 3.

Our project includes a central server that we host on AWS, specifically the M5a EC2 instance. We set up this server to include the django REDIS server that runs in the background to support our website. We also run our main networking code for audio streaming between the two audio devices and the ML speech to text and speaker identification components. We

lastly use the EC2 instance to run our transcript generation and meeting management code. Over the course of the semester we used about $25 worth of AWS credits. We were grateful to have the opportunity to use AWS resources in our project.

D. Risk Management

The risks to our project came in multiple forms. There were some associated with the technical design and others that came from the project plan. We had mitigation strategies in place for all the risks we identified, and we did in fact need to enact multiple as risks actually came into play.

The first category of risk we encountered was technical risk relating to our solution design. As described in Section ii., we identified a set of requirements for our design, and we put tests in place to verify whether or not our solution met the requirements after its implementation. We considered each requirement to be a potential source of risk, the risk being, of course, that our solution might fail our tests. Because of this, we planned fallback strategies corresponding to each requirement.

To delve into the specifics of these strategies, we'll describe them in the order the requirements were originally introduced. Audio transmission latency, and similarly transcript latency, might be improved by investing more money into the AWS server's networking specs or by relocating parts of the processing onto the server from the hardware. Both of these mitigations were actually necessary to achieve good audio streaming. We could improve the dropped packet rate by reducing packet size or selecting a different transport layer protocol; this, however, we found to be unnecessary.

Transcript error, both word errors and speaker identification errors, could be combated by trying alternative machine learning models or switching from free software to paid services. We decided to explore this path early on, adding paid Google services to our transcription arsenal given preliminary speech to text results, and ultimately we used Google's product for both speech to text and speaker identification. Transcript formatting error is a non-ML issue and can be corrected by sending more metadata, such as timestamps, to the meeting management software module. This risk did not come into play.

The project planning risks were related to, but separate from, the technical risks. First of all, there was a danger that, even if our solution worked, it could take longer to implement than we anticipated. This situation didn't arise, however. A similar risk was that, having implemented our solution according to the schedule, we might find that it doesn't pass some test in the end. Our schedule design was intended to help with this: we allocated multiple weeks at the end of the semester for testing, and this purposefully includes extra time for revision or further development. We used this time to incrementally add improvements to the speaker identification system.

VIII. Ethical Issues

As with any technology, there is ample opportunity for StenoPhone to be misused. The product could either be exploited intentionally or could accidentally create an adverse impact.

There are some features which would need to be added to StenoPhone before it could be introduced to widespread use. Security is lacking in the current implementation; anyone can create a meeting and anyone can enter the meeting room of an ongoing or finished meeting. A finished product would include user authentication and permissions and would have a secure, HTTPS compliant website.

StenoPhone of course falls under the wide umbrella of technologies that could be used for surveillance purposes. Anyone could set up a microphone to record transcripts of the conversations of unknowing individuals nearby. The same is true of many tools that include microphones or cameras.

Even the day to day, intended use of Stenophone, however, introduces potential ethical issues of which the user should be aware. These issues are in large part inherited from the Google Cloud Speech to Text service, which provides much of the content of our transcripts. As the machine learning technology is still imperfect, all users should be aware of the possibility of mistakes in transcription. Transcribed words may be misattributed to the wrong speaker, and the words themselves may be mistranscribed by substitution, deletion, or incorrect addition. This clearly creates the opportunity for problems and misunderstandings, especially if users are unaware of the imperfections in the technology.

These transcription errors may create even deeper ethical issues than simple misunderstandings, however. The accuracy of Google Cloud Speech to Text's service is not the same for every single speaker. This may result from any number of factors, including the choice of training data for their models. In any case, the effect is that when speaking in one's normal voice, one's speech characteristics, including accent, enunciation, and pronunciation, will influence the accuracy of the transcription outcome. People who don't speak the "right" way that Google best recognizes may lose out, and if carried to its logical conclusion, StenoPhone's adoption by a company could give rise to the needless standardization of Google-friendly accents and ways of speaking in the workplace.

IX. Related Work

The field of web conferencing solutions is well-saturated at this point, with seemingly every major software company offering their own alternative as an element of their suite of products. Here we'll highlight just a few major examples. We'd also like to point out the work of previous ECE Capstone group iContact, who created a project in the same field of distributed meetings with multiple participants per room. Their solution, though, focused on the video portion more than the audio portion.

The current most famous web conferencing tool is Zoom. The focus of this software is audiovisual meetings between

multiple individuals. Zoom allows live closed-captioning to be performed manually by participants or added by a third-party service [13]. It also offers live speech-to-text transcription in paid versions. Transcription does not indicate who is speaking [14]. Microsoft Teams, another prominent web conferencing solution, does offer a form of speaker attribution in its live transcription service. However, this very new feature assumes that there is only one participant per connected device [15]. Cisco's Webex provides the same level of speaker labeling as Microsoft, along with other interesting functionality using voice commands [16].

These and other common conferencing tools allow each instance of the application to run on different hardware. This is great for portability, but limits speaker identification capabilities.

The area of hardware microphone-speaker meeting tools is, however, one under active development in the industry. Only a few weeks ago, at the beginning of March, Microsoft demonstrated its plans to release "Intelligent Speakers, small puck-like devices that can identify up to 10 different voices in a Microsoft Teams meeting" [17]. The speakers can transcribe and translate meetings. It's interesting that a prominent company is developing their version of the product simultaneously as we develop ours. When we learned about this news, we believed it demonstrated that there is a real demand for this technology.

X. Summary

Although tradeoffs needed to be considered and design changes needed to be made along the way, our system was able to meet its design specifications well. Our audio transmission results were far better than our requirements, and our results for transcription were also within bounds.

It was somewhat easier to reach these requirements because of the small size of our deployment - with only two hardware devices and a web server in a nearby state. While we believe it is quite feasible to scale the StenoPhone product to include more microphones, some changes would need to be made in order to do this. Some of the UDP server code would need to be rewritten, even though overall much of the implementation does not rely on the assumption of a limited number of microphones. The transcript latency is also in small part a linear function of the number of microphones currently transmitting audio - each mic may add a few milliseconds to the processing time - so in a *very* large scale deployment, it could be advantageous to break this part of the code into multiple threads with a limited number of microphones to deal with per thread. Additionally, the more microphones are connected to a single meeting, the greater risk there would be that pieces of the transcript could be incorrectly ordered. This behavior would not be unbounded, however, with bounded waiting on the locks that synchronize transcript modification.

Even though we were content with the audio quality we obtained from our microphone devices without doing audio processing on the device itself, we would want to add this if we were to continue the project. This would probably involve switching out our hardware components, abandoning the Raspberry Pi for something more hefty. With a device capable of doing the processing without bad networking impacts, we'd first introduce voice activity detection (only sending audio with voice contents to the server) and also perhaps apply advanced noise reduction algorithms to get a really outstanding audio quality for users.

One limitation of the current product is that every meeting participant needs to have their own StenoPhone hardware device. This could be inconvenient for a traveling employee; it would be nice to have an option to join the meeting with an arbitrary audio source through the web application. The same transcription quality features might not be present, but at least the user would be able to participate.

After working on this project for a semester, we've come away with both new technical knowledge and design perspectives. Below, we'll discuss several of the lessons we learned.

As our project was developed in python, we encountered both pros and cons of the language. The benefit of our approach was that all components had python implementations, so it was easy to integrate without jumping around in languages; however, the documentation of the components were sparse and it was difficult to develop as the types were not well defined and we had to investigate the data manually.

We also learned that live audio streaming is quite delicate. Since all the modules that we used were in python we had to put in more consideration to the latency of the system. We originally had the audio processing performed on the raspberry pi, but it made the system too slow which resulted in choppy audio; therefore, we moved the processing to the AWS server where there is a lot more computing power and slowed down the system less.

We started the semester thinking of using open source software packages for our machine learning components - speech to text and speaker identification. But what we found was that documentation could be lacking and results were not up to our requirements; so in the end, we resorted to paying for software services for the same components, which had easier deployment and better results.

Overall, we thought it was interesting and cool to watch our design work, and hard thinking pay off, as our product took shape, and became operable.

Glossary of Acronyms

AWS - Amazon Web Services
ASGI - Asynchronous Server Gateway Interface
DOA - Direction of Arrival
FER - Formatting Error Rate
M2E - Mouth to Ear
WER - Word Error Rate

References

[1] VOIP-info.org, "QoS." https://www.voip-info.org/qos/

[2] vyopta.com, "Troubleshooting Packet Loss: How Much is an Acceptable Amount?" Dec. 2018. https://www.vyopta.com/blog/video-conferencing/understanding-packet-loss/

[3] B. McLaughlin, "Recommended Practices for Closed Captioning Quality Compliance," Jan. 2015. https://s3.amazonaws.com/eegent-assets/resources/McLaughlinB190115.pdf

[4] C. Muneanu et al., "Measuring the Acceptable Word Error Rate of Machine-Generated Webcast Transcripts," Jan. 2006. https://www.researchgate.net/publication/221485939_Measuring_the_acceptable_word_error_rate_of_machine-generated_webcast_transcripts

[5] R. Morais, "DeepSpeech 0.6: Mozilla's Speech-to-Text Engine Gets Fast, Lean, and Ubiquitous," Dec. 2019. https://hacks.mozilla.org/2019/12/deepspeech-0-6-mozillas-speech-to-text-engine/

[6] D. Huggins-Daines, "PocketSphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices", 2006. https://www.cs.cmu.edu/~dhuggins/Publications/pocketsphinx.pdf

[7] E. Protalinski, "Google's Speech Recognition Technology Now Has a 4.9% Word Error Rate," May 2017. https://venturebeat.com/2017/05/17/googles-speech-recognition-technology-now-has-a-4-9-word-error-rate/

[8] H. Bredin et al., "Pyannote.Audio: Neural Building Blocks for Speaker Diarization," May 2020. https://ieeexplore.ieee.org/document/9052974

[9] J. Patino, "EURECOM Submission to the Albayzin 2016 Speaker Diarization Evaluation," 2016. https://www.semanticscholar.org/paper/EURECOM-submission-to-the-Albayzin-2016-speaker-Patino/d8917a687edaaae061b59cf0baec95ced41e9dfc?p2df

[10] Y. Fujita et al., "End-to-End Neural Speaker Diarization with Self-Attention," Dec. 2019. https://ieeexplore.ieee.org/abstract/document/9003959

[11] F. Landini et al., "Bayesian HMM Clustering of X-Vector Sequences (VBx) in Speaker Diarization: Theory, Implementation, and Analysis on Standard Tasks," Dec. 2020. https://arxiv.org/abs/2012.14952

[12] DPA Microphones, "Facts About Speech Intelligibility," March 2021. https://www.dpamicrophones.com/mic-university/facts-about-speech-intelligibility

[13] Zoom Help Center, "Closed Captioning and Live Transcription." https://support.zoom.us/hc/en-us/articles/207279736-Closed-captioning-and-live-transcription

[14] Vanderbilt Brightspace, "How Can I Enable Live Transcription in Zoom?" Feb. 2021. https://www.vanderbilt.edu/brightspace/how-can-i-enable-live-transcription-in-zoom/

[15] R. Noureen, "Microsoft Teams Adds Live Transcription with Speaker Attribution for Meetings," March 2021. https://www.onmsft.com/news/microsoft-teams-adds-live-transcription-with-speaker-attribution-for-meetings

[16] A. Barave, "Turning Talk into Action: Bringing Voice Intelligence into Webex Meetings," Jan. 2020. https://blogs.cisco.com/collaboration/turning-talk-into-action-bringing-voice-intelligence-into-webex-meetings

[17] T. Warren, "Microsoft's New Intelligent Speakers Deliver its Promised Meeting Room of the Future," March 2021. https://www.theverge.com/2021/3/2/22308962/microsoft-intelligent-speaker-teams-translation-transcription-features

[18] Cloud Speech-to-Text Documentation. "Transcribing Audio from Streaming Input: Performing Streaming Speech Recognition on a Local File," May 2021. https://cloud.google.com/speech-to-text/docs/streaming-recognize#performing_streaming_speech_recognition_on_a_local_file

[19] Cloud Speech-to-Text Documentation. "Package google.cloud.speech.v1," March 2021. https://cloud.google.com/speech-to-text/docs/reference/rpc/google.cloud.speech.v1

Task	Team Member	Status	Week 3(Feb 15-21)			Week 4(Feb 22-28)			Week 5(March 1-7)			Week 6(March 8-14)			Week 7(March 15-21)			Week 8(March 22-28)			Week 9(March 29-April 4)		
			M	W	F	M	W	F	M	W	F	M	W	F	M	W	F	M	W	F	M	W	F
Phase 0: Design	all	done																					
Initial research	all	done	all	all	all																		
Project Proposal Presentation	all	done	all	all	all																		
Phase 1: System Setup	all	done																					
Network Design: RPi to AWS	cne	done				cne	cne	cne															
Network Design: AWS to rpi	cne	done						cne	cne	cne													
Hardware Setup	cne	done						cne	cne	cne	cne												
Respeaker Config on Raspberry Pi	cne	done								cne	cne	cne											
AWS Database + Django Initialization	mfy	done				mfy	mfy	mfy															
AWS EC2 Setup	mfy	done				mfy	mfy	mfy															
Speech to Text ML Initial Version	eseeser	done				eseeser	eseeser	eseeser															
Design Presentation & Report (presentation due 7th, report 17th)	all	done				all	all	all	all	all	all	all	all	all									
Backend Pre-ML Processing	eseeser	done						eseeser	eseeser	eseeser													
Test & Slack Time	all	done										all											
Phase 2: Backend	all	done																					
Database Creation and Integration	mfy	done						mfy	mfy	mfy													
Website Creation, Including Backend	mfy	done						mfy			mfy	mfy			mfy	mfy	mfy						
Non-Moving Speaker ID (no ML)	eseeser	done								eseeser	eseeser	eseeser	eseeser										
Server connection at AWS	cne	done										cne	cne	cne									
Microphone IO on RaspberrryPi	cne	done											cne	cne	cne	cne	cne						
Client Code on Raspberry PI	cne	done														cne	cne	cne					
Speech to Text ML Integration (first pass at meeting manager)	eseeser	done											eseeser	eseeser	eseeser								
Audacity Filter Design + Audio Process	mfy	done									mfy	mfy	mfy	mfy									
2 Audio Device Audio Streaming	cne	done																	cne	cne	cne	cne	
Mic Initialization Backend	eseeser	done																eseeser	eseeser	eseeser			
Testing & Slack Time	all	done																all	all	all			

Fig. 9. Schedule, Phases 0-2 View

Task	Team Member	Status	Week 7(March 15-21)			Week 8(March 22-28)			Week 9(March 29-April 4)			Week 10(April 5-11)			Week 11(April 12-18)			Week 12(April 19-25)			Week 13(April 26-May 2)			Week 14(May 3-5)	
			M	W	F	M	W	F	M	W	F	M	W	F	M	W	F	M	W	F	M	W	F	M	W
Phase 3: Frontend	all	done																							
Create housing	mfy	done				mfy	mfy																		
Transcript Support for Multi-mic Meetin	eseeser	done	eseeser	eseeser	eseeser																				
Full On-Website Meeting Setup	eseeser	done		eseeser	eseeser	eseeser																			
Moving Speaker ID (ML)	eseeser	done					eseeser	eseeser	eseeser	eseeser	eseeser														
Transcript Web Streaming	mfy	done						mfy	mfy	mfy	mfy	mfy	mfy												
Integration of RPi + Transcript + ML	all	done							all	all	all	all	all	all											
Complete System Debugging / Testing	all	done											all	all	all	all									
Phase 4: Touchup	all	done																							
Web CSS	all	done												all	all	all	all								
AWS Migration	all	done															all	all							
Testing	all	done																all	all	all	all	all	all		
Slack Time	all	done																					all	all	
Final Report, Presentation, Final Video	all	done														all	all	all	all	all	all	all	all	all	all

Fig. 10. Schedule, Phase 3-4 View

Item	Quantity	Price	Total Price	Link
Respeaker Mic Array v2.0	2	$79.00	$158.00	https://www.amazon.com/Se
SD card for Raspberry Pi	1	$6.57	$6.57	https://www.amazon.com/Sa
Raspberry Pi case	1	$9.99	$9.99	https://www.amazon.com/Bla
Raspberry Pi Power cable	1	$9.99	$9.99	https://www.amazon.com/Ca
ReSpeaker Case	2	$10.00	$20.00	
AWS Credits				
$50				
Parts From Inventory				
JBL Speaker	1	$0.00	$0.00	Parts Inventory: EDE0076
Raspberry Pi 3 B	1	$0.00	$0.00	Parts Inventory: EDE0130
			$600.00	Budget (approved vendors)
			$204.55	Total spent
			$395.45	Total remaining

Fig. 11. Budget and Parts List

TABLE III. BILL OF MATERIALS

Material	**Type**	**Quantity**
Google Cloud Speech to Text [18][19]	Software service	N/A
Django	Software package	N/A
SQLite	Software package	N/A
AWS EC2 M5a instance	Web deployment service	1
Micro SD Card	Hardware	2
JBL Speaker	Hardware	2
Raspberry Pi 3 B	Hardware	2
Raspberry Pi Power Cable	Hardware	2
ReSpeaker Microphone Array V2.0	Hardware	2
ReSpeaker Microphone Array Case	Hardware	2
3.5 mm Audio Cable	Hardware	2
USB A to micro USB Cable	Hardware	2