

# Internship Report

Fahim Imaduddin Dalvi

Meeting Translation Project

## Abstract

The Meeting translation project aims to provide a platform for multi-lingual meetings. In order for the system to work efficiently, a robust backend is required to augment the automatic recognition and translation services. The current backend is a very simple proof-of-concept that supports a single user only. The goal of this internship is to develop a backend that will support the realtime needs of this project.

# Table of Contents

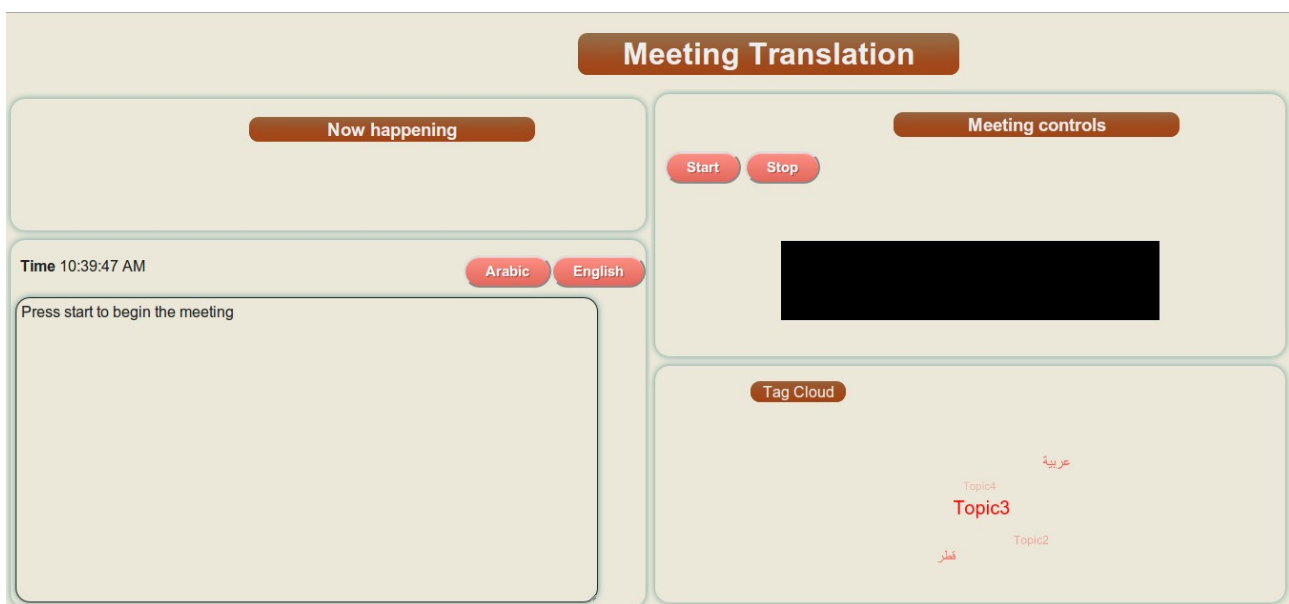
Section	Page Number
Introduction	3
Initial State	3
Objectives	4
Related Work	5
Technologies	7
Architecture	8
Statistics Collection	10
Results	12
Audio Processing Statistics	12
Realtime Factor	13
Latency	14
Future Work	16
Known Issues	17
Other tasks	18
Conclusion	20
References	21

# Introduction

The Meeting Translation project aims to make meetings involving multiple languages more efficient. Currently, when multiple languages are involved, either human interpreters are brought in to bridge the gap between the meeting participants, or most of the participants do not completely understand what is going on, and hence their participation is limited. This project will provide an online platform where participants can create and join meetings. The platform will provide real-time speech recognition and translation services, and hence will bridge the linguistic gap between the users in a timely fashion.

## Initial State

Before the internship, a proof of concept system was already in existence. This system was a single-user, single-meeting system. The backend was mainly built on the Native WebRTC Tutorial<sup>1</sup>. The tutorial was heavily dependent on a GUI component, which was acceptable for a proof of concept system, but was not suitable for deployment on an end-system. The rest of the backend that sent the audio for processing to the recognition and translation servers was also user/meeting unaware.

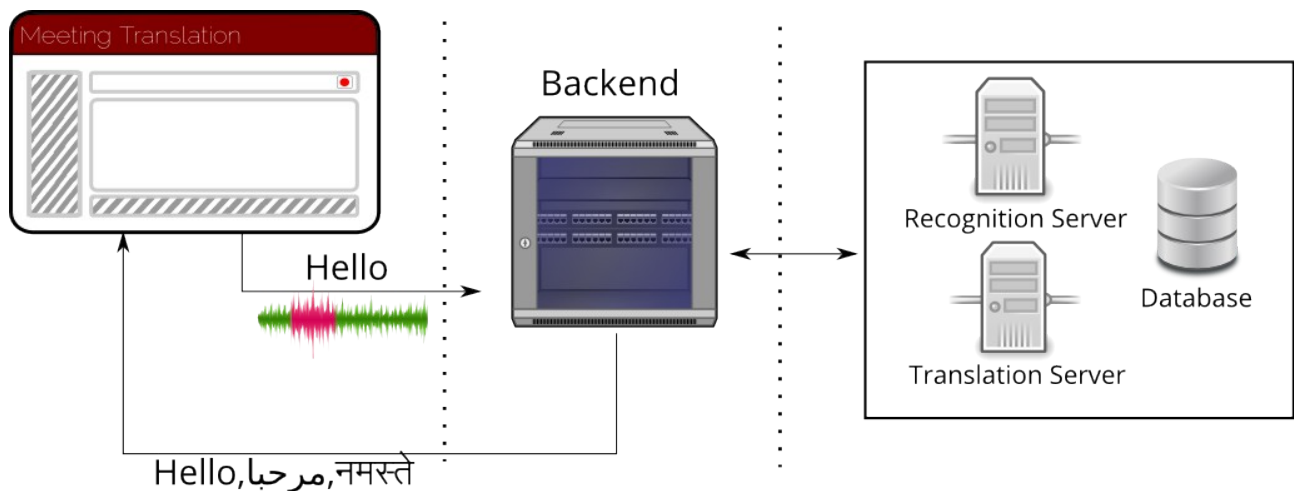


1 <http://www.webrtc.org/reference/getting-started>

# Objectives

The main objective of this internship was to build a robust backend for the Meeting Translation Project. The idea was to integrate a completely redesigned frontend with the Recognition and Translations services available at the backend. The integration would consist of the following:

1. Get the user's voice from the frontend to our servers
2. Transcribe the user's voice
3. Translate the transcriptions(into multiple languages if required)
4. Get the transcriptions/translations back to the user



The system was also to be multiplexed, so that multiple users in multiple meetings can use the system at the same time. This is necessary for the end system, as the goal of the project is to enable interaction between multiple users speaking multiple languages in a meeting.

# Related Work

There are a few systems that resemble part of what the Meeting translation project aspires to do:

- **Google Translate:** Google translate currently supports speech translation<sup>2</sup> between multiple languages. The main difference between this project and Google translate is that Google translate is not built for meetings, and it does not provide the specific functionality the the Meeting translation project will provide in order to enhance meetings. Also, although Google translate supports speech, it does not support continuous speech. Hence, all audio must be sent to the server before receiving any results.
- **Verbalizeit:** This mobile application pairs up the user with another human that would act as a live interpreter and translator<sup>3</sup>. This is what currently happens in meetings like the UN, and events where multiple languages are involved. The Meeting translation project is very different from such services, as its employs a completely different ideology of automating both the recognition and translation of speech, in order to reduce the inefficiency and costs that arise from the presence of human interpreters and translators.
- **OmNovia:** This platform provides an online meeting platform<sup>4</sup>, just like the Meeting translation project. It supports multiple languages, but again, the entire process of interpreting and translating is manual, which is a key difference between this platform and our project. Since the meeting translation project uses automatic speech recognition and translation, the speed at which the results are available will be much higher. The quality might be slightly reduced, but this will only improve as new research presents better language models and systems as time will progress.

The above platforms/system provide a brief overview of the kinds of systems out there.

---

2 <http://googleblog.blogspot.com/2011/01/new-look-for-google-translate-for.html>

3 <https://www.verbalizeit.com/>

4 <http://www.omnovia.com/multi-language/>

There are a lot of *meeting platforms*, that allow multiple people to meet and interact online, but only a few of them support multiple languages, and even these are backed by live human translators. Other projects provide automated translation services, but few of them are realtime. The Meeting translation project aims to bridge the gap between these two kinds of systems, and provide an unique platform for multi-lingual meetings.

# Technologies

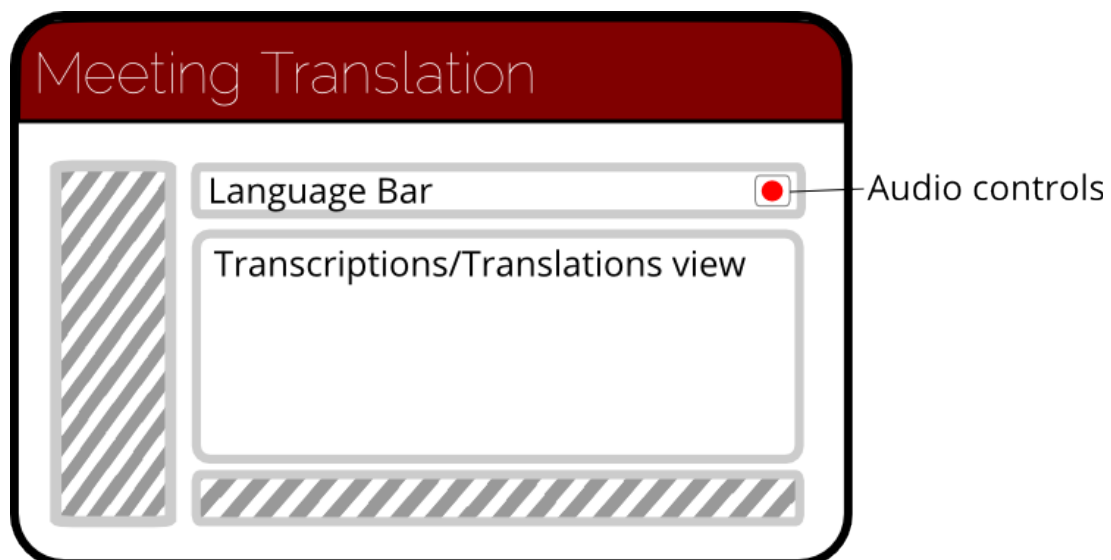
The following technologies were used for the project:

- **WebRTC:** WebRTC is a new and upcoming web standard. It is built right into HTML5. Since HTML5 is a web standard now, it will be available in all browsers, including on mobile devices. This allows us to instantly develop our system for multiple platforms without worrying about device specific restrictions. Also, since it's baked right into the browser as a standard, installation of third party plugins is also not required. WebRTC also supports noise cancellation by default, and hence is suited for a multi-party meeting. All of these features enhance the quality of our system for the end users.
- **NodeJs:** The decision to use NodeJs for the server side processing was made due to its robustness and ease of management. NodeJs requires minimal setup, and is incredibly powerful thanks to JavaScript. It also has a wide contributor base, which provides us with extensions and support that are required for our system. NodeJs essentially replaced the C++ server in this project. This helps us leverage the ease of use of JavaScript while still maintaining the powerful functionality of the C++ server.
- **Apache:** Apache is used to serve static content for the end user. It is also used to carry out session/user management tasks.
- **MySQL:** MySQL is used as the database backend for the system. All the information required by the system such as users, meetings, sessions, transcriptions, translations etc. are stored in the database.
- **Python:** Python is used heavily to perform the task of generating current meeting topics. It is also used in part of the pipeline that connects the NodeJs servers and the Recognition/Translation servers. This pipeline is responsible for carrying all audio data from the NodeJs servers to the recognition server, and the results from the recognition server again use this pipeline to provide input to the translation server.

# Architecture

The architecture of the backend consists of two NodeJs servers to dynamically serve and process clients. An apache server will serve all the clients with static content, and will also take care of user/session management.

As far as the server side is concerned, only four elements are required from the frontend.



1. The language bar – This decides what language to user currently wants to view the transcriptions/translations in
2. The Transcriptions/Translations view – This is where the user can view the results
3. Audio controls – These defines when the audio is being sent to the server
4. Session metadata – This is provided by apache's session management. The bare-minimum information that is required for the servers to perform correctly is user-id, meeting-id and language-id

On the server-side, there are two NodeJs servers running all the time.

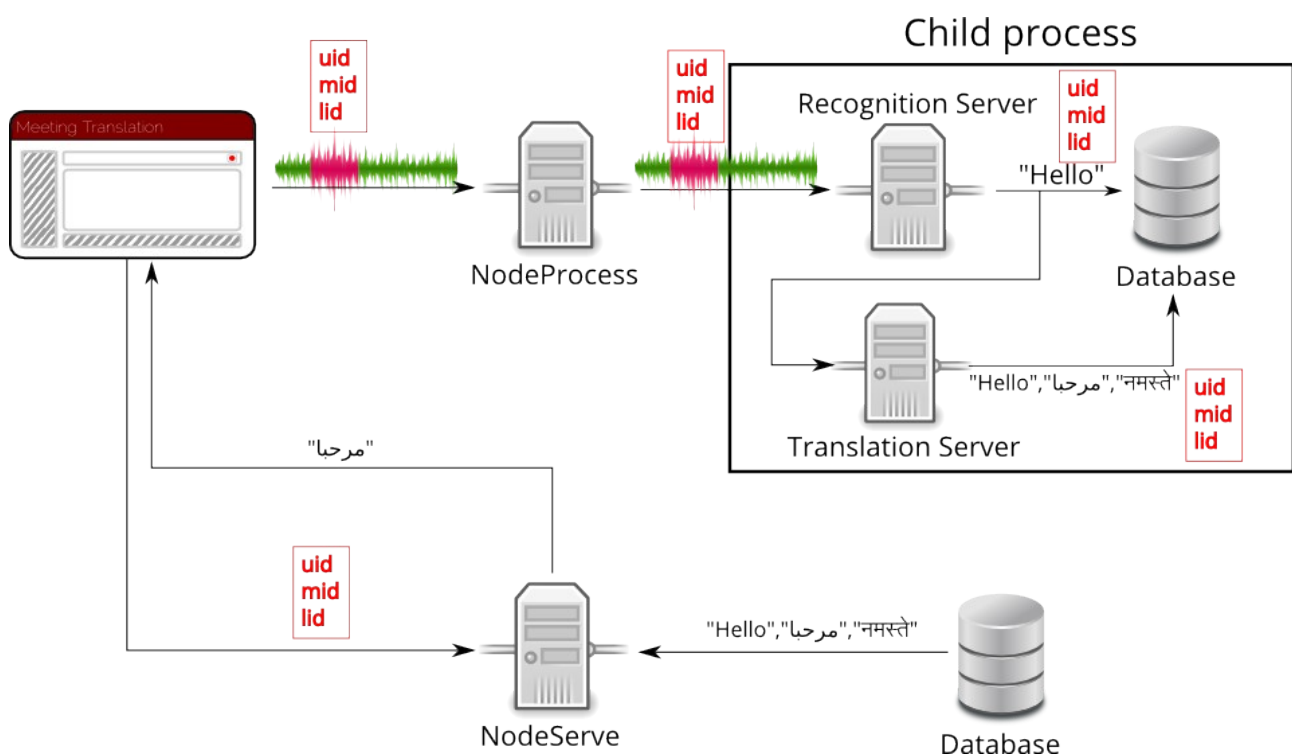
1. **NodeProcess server:** This server is responsible for receiving audio from the clients and passing it on to the recognition server, translation server, and finally storing the results in the database. This is based on [node-peerconnection](https://github.com/Rantanen/node-peerconnection)<sup>5</sup>, an open source project on GitHub that tries to implement the WebRTC bindings from

<sup>5</sup> <https://github.com/Rantanen/node-peerconnection>



C++ in Node.js. For every user, a pipeline is created by this server. This pipeline connects the server itself to the Recognition/Translation servers. Hence, when audio is received from the user, it is sent through this pipeline first to the Recognition server. After receiving the results from the recognition server, they are saved in the database, packaged again and are sent to the translation server, results from which are again stored in the database. This module also implements a statistics collection scheme, which stores all statistics related to recognition (Audio length, Real time factor), translation (translation time) and the latency of the results. These statistics are very useful for analyzing the performance of the overall system, as well as its individual components (Actual statistics and observations can be found in the *Results* section).

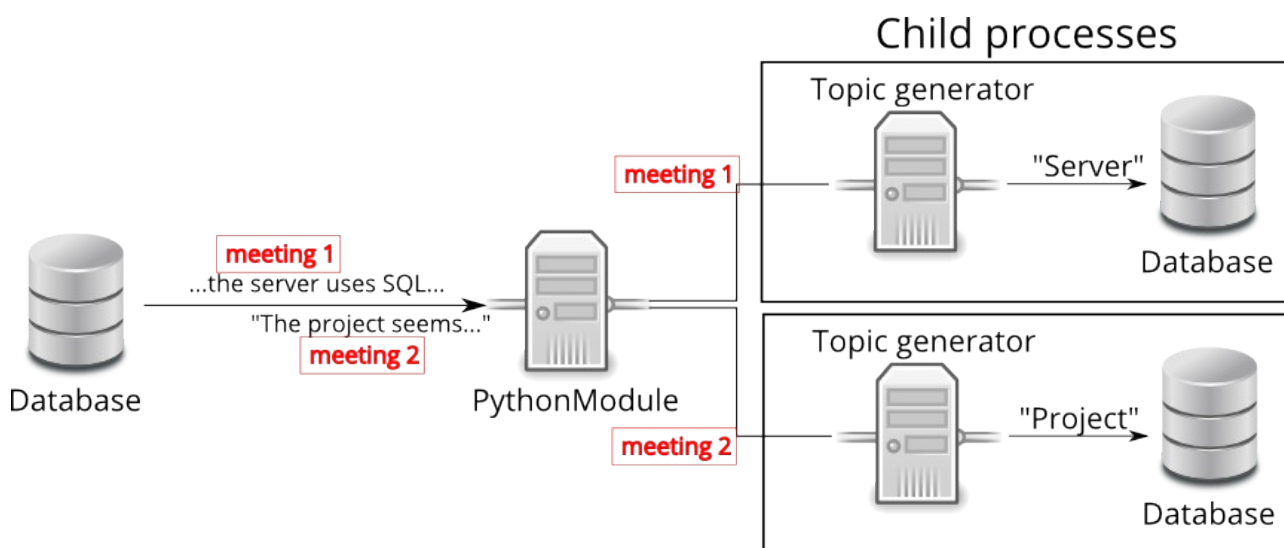
2. **NodeServe server:** This server is responsible for pushing the results stored in the database to the connected clients. This is a more efficient method of getting the results to the clients, instead of each client polling the database regularly. This server also manages the state of the current meetings and the users connected to them, in order to send only necessary information to each user.



Overview of the system architecture

Another process running on the server-side is a python module that is responsible for generating topics from the meeting transcriptions. This module basically analyzes the transcriptions and tries to figure out what topics are being currently talked about, and how important each topic is. This information is then passed on the the Frontend by the nodeServe server as they are made available by the python module.

The python module constantly checks for which meetings are in progress. For each of these meetings, it checks if new transcriptions are available, and sends these transcriptions to the *Topic Generator*. The *Topic Generator* has historical knowledge of all the transcripts of the current meeting, and after collecting sufficient number of transcriptions, it outputs and saves the important topics in the database. One *Topic Generator* is run for every meeting.



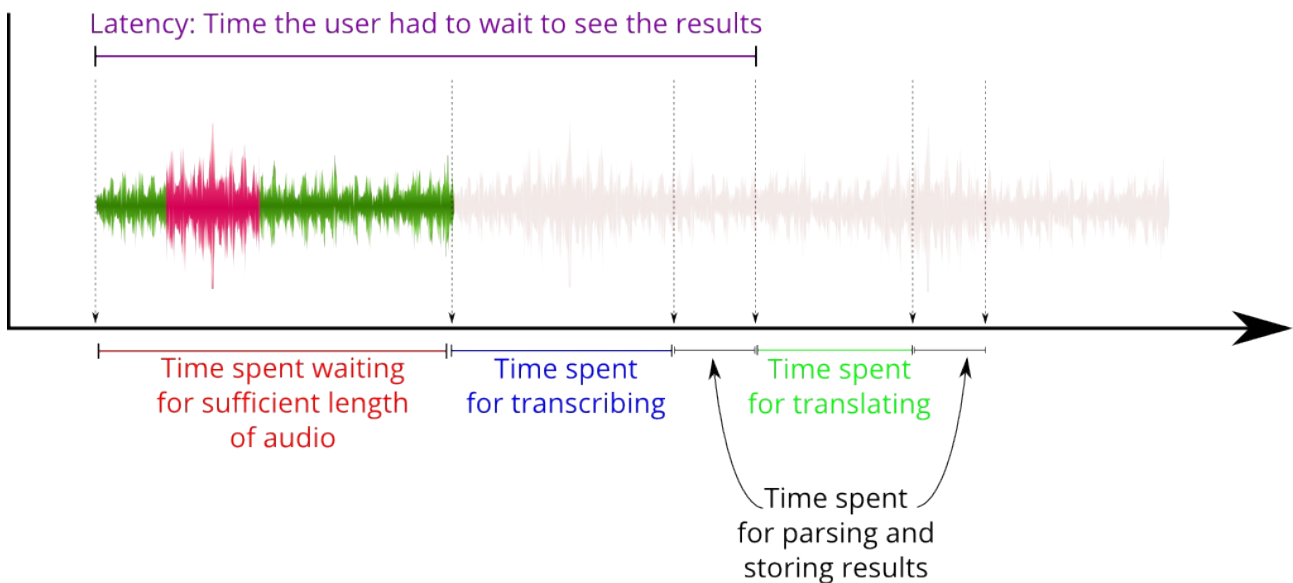
Overview of the Python-Module

## Statistics Collection

As mentioned earlier, the nodeProcess server has built-in capabilities for collecting statistics. Before we understand how the statistics are collected, we need to understand the basic concept of *segments*. When continuous audio is sent over to the recognition server for processing, it is segmented into smaller chunks (based on several heuristics like silence). Cutting the audio into smaller chunks, or *segments*, helps the server process the

audio in pseudo-realtime (As the server does not have to wait for the entire speech of the user to finish). We receive the transcriptions corresponding to each of these *segments*, and send each of these transcriptions for translation in the very same *segments*.

Now, we can determine the time each segment spent in the pipeline, and the time the decoder spent to process each segment. Hence, for each segment, we collect the necessary information in the following way:



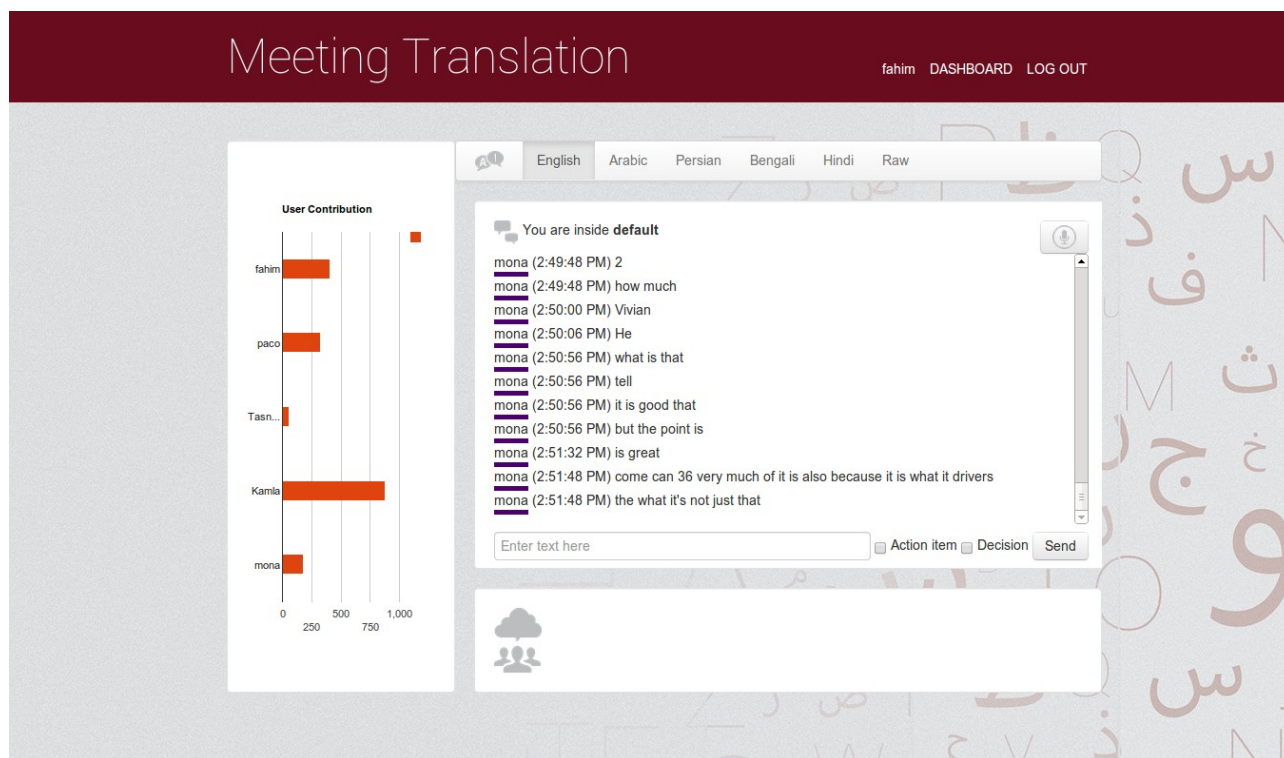
We can use the decoding time (time spent for transcribing) and the audio length (time spent waiting for sufficient length of audio) to calculate the realtime-factor of the system:

$$\text{Realtime Factor} = \frac{\text{Decoding Time}}{\text{Audio Length}}$$

The latency before which the user can see the results would be the sum of the audio length, the decoding time, and the time it takes to parse and store the recognizer results.

# Results

A fully working system was created and tested against multiple users and meetings. The complete cycle (Audio Collection → Recognition → Translation → Topic Generation) was tested thoroughly with Arabic and English.



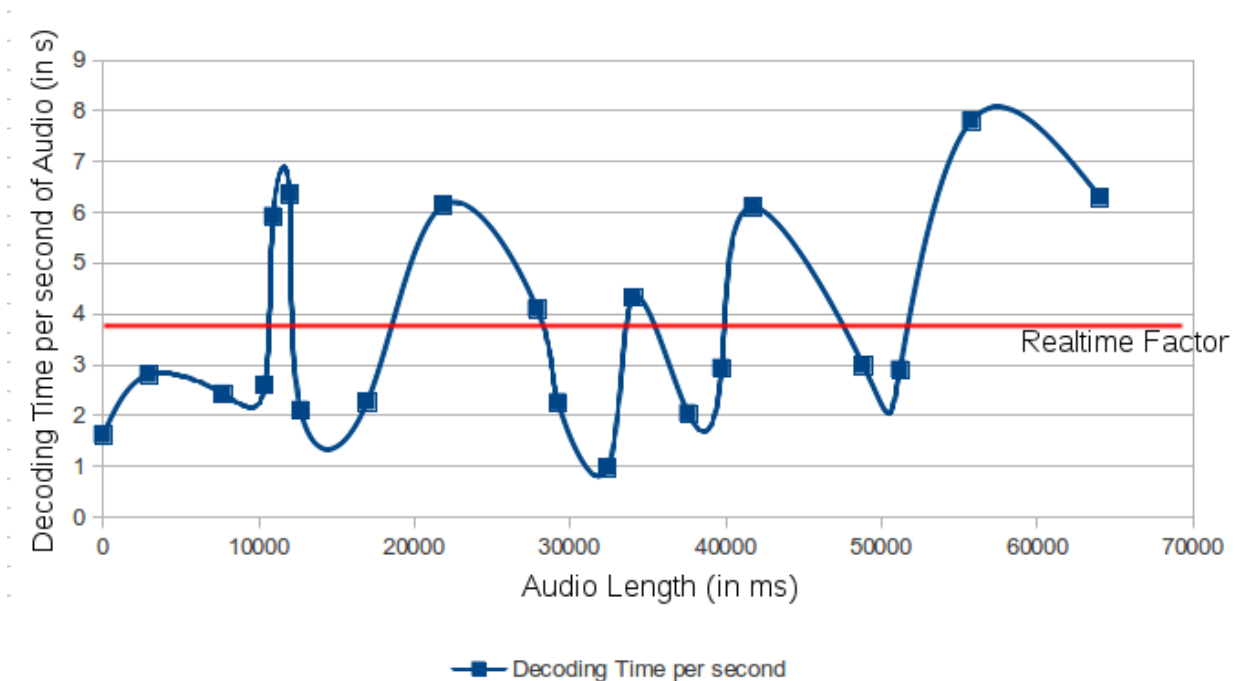
Complete integration (Language Selection, Transcriptions view, Translations view, Contribution calculation, Topics view) was achieved with the frontend. The integration also includes proper error handling. For example, the client will automatically reconnect in case the server-side fails for an unforeseen reason. The user is also notified of the reconnection in a subtle manner.

## Audio Processing Statistics

The method for collecting statistics described in the nodeProcess section above was also tested in the final system. For this test, the translation server was inactive, and hence only statistics related to the recognition process were collected. The system was tested with 1 Person speaking for 1 minute. The language used was English. The number of processes on the recognition server were 4, which allowed 4 segments to be processed

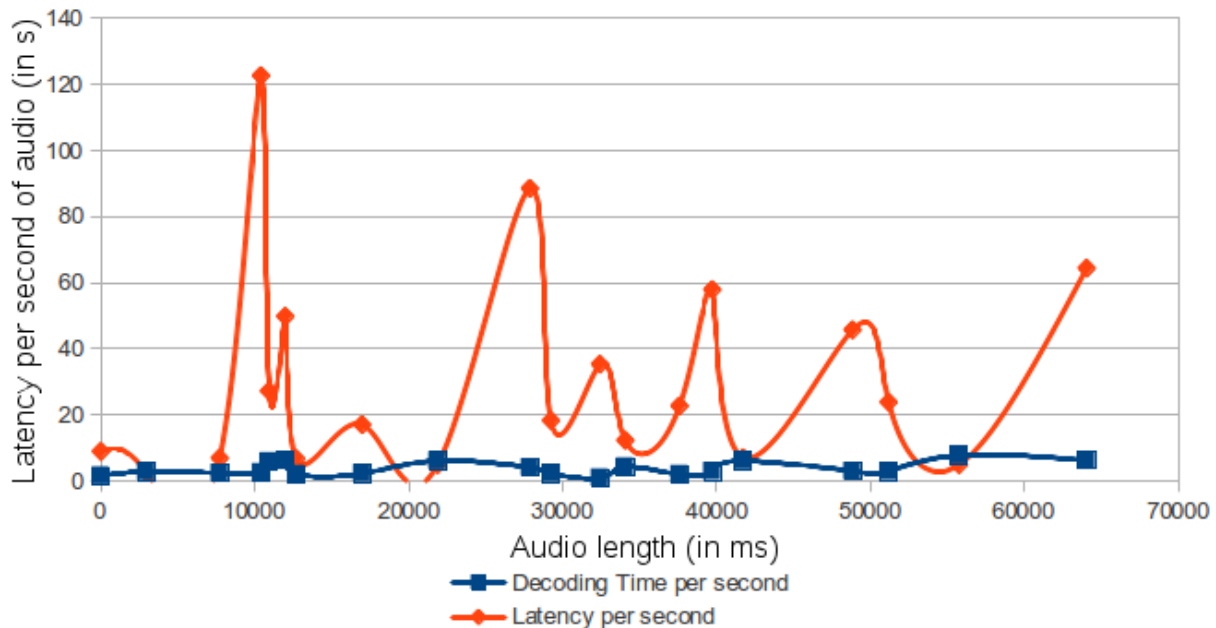
concurrently. The following results were obtained:

## Realtime Factor



The decoding time (per second of audio input) here has been plotted against the start time of each segment. Although the reasoning for the several peaks in the graph is unclear, it could be due to the fact that not all segments were of the same complexity. The recognizer has to maintain history and make a decision between several choices. Since the number of choices varies from segment to segment, the peaks may represent the segments where the number of choices was high. The average decoding time was computed as 3.71, which is also known as the realtime factor.

# Latency



A more interesting result that was obtained was of the latency. Latency is basically the time the user has to wait after speaking before the transcriptions and translations are available. In the above graph, the latency (per second of audio input) has been plotted with the decoding time. As it can be seen, the latency for each segment is much higher than the realtime factor. This can be attributed to two reasons:

1. The number of processes on the server is low (i.e it should be higher than 4)
2. When the number of segments to be processed becomes greater than four, the segments are queued. Hence if the serving time (decoding time in this case) is greater than the time in which a new segment is added to the queue, the backlog for each recognition process increases, and all subsequent segments experience a higher latency.

These findings are incredibly helpful for the person working on the recognition side to find out where the bottleneck in the system lies. The important point here is that this kind of statistics collection can be performed on any queue based processing system.

The quality of the results was rather poor due to the lack of training data for the recognition and translation systems. The systems themselves perform very well. This is evident from the fact that if the input to the system is analogous to what the system was originally trained on (Example: News data for Arabic Recognizer, TED-Style talks for translator), the results are very promising. The accuracy is quite reasonable in these cases, suggesting that the systems themselves are fine, only the training data was insufficient. Hence, by collecting and analyzing these statistics, we can find which part of the system requires the most attention and tweaking.

# Future Work

The system has come a long way in these eight weeks. There are a few limitations and known bugs (explained further below), but overall the system performs well. A lot of work still remains in order for the system to be deemed complete. Some features that would be augment the usefulness of the system would be:

- **Continuous Audio:** Currently, the system works akin to a push-to-talk system, where each user unmute's his/her microphone when he/she wants to speak. It would be much better if all the user has to do is speak, and the system will take care of identifying when the user actually spoke. This is technically possible right now, but a higher level of integration in order for the system to perform well (better noise cancellation, crosstalk identification etc).
- **Define Meeting End:** Due to shortage of time, the frontend currently does not provide a way to end a meeting. This may seem trivial, but this is extremely important for the backend. Both the nodeServe server and the python module keep track of meetings currently in progress. This is done in order to avoid unnecessary computation each time a new user logs in, i.e. if a meeting is already in progress, the transcripts are stored locally on nodeServe to avoid redundant polling. Moreover, the topic generation takes into regard all the past results to compute the current topics. Hence, if we prematurely end a meeting, all this past computation will have to be recomputed.

Currently, a meeting is considered 'not-in-progress' if there are no users in the meeting. This may not be the ideal case in the final system, as meetings can be created before hand by users in advance, and meetings may have 'breaks' where all users leave temporarily.

- **Support any number of languages:** Currently, parts of the system are centered around the fact that there are two languages(For example, the script that sends the transcripts for translation). It would be quite important to make this generic, as the system may support more languages in the future.



# Known Issues

There are currently two issues that are known to exist:

## 1. Audio tempo is modified on reception

- On receiving audio in the NodeProcess server, the audio is approximately 0.29 times slower than the original. The pitch of the audio is unchanged, suggesting that this is a change in tempo. The 0.29 factor was identified by recording audio both at the source and after it was received by the NodeProcess server. Upon careful inspection using Audacity(an audio manipulation tool), it was found that increasing the tempo of the audio by a factor of 0.7122 would restore the audio to its original form.
- This is currently fixed by introducing 'SoX' in the audio processing pipeline. Basically, all audio on reception is piped through SoX before being sent off to the servers. SoX has a 'tempo' effect which increases the tempo of the audio in real time.

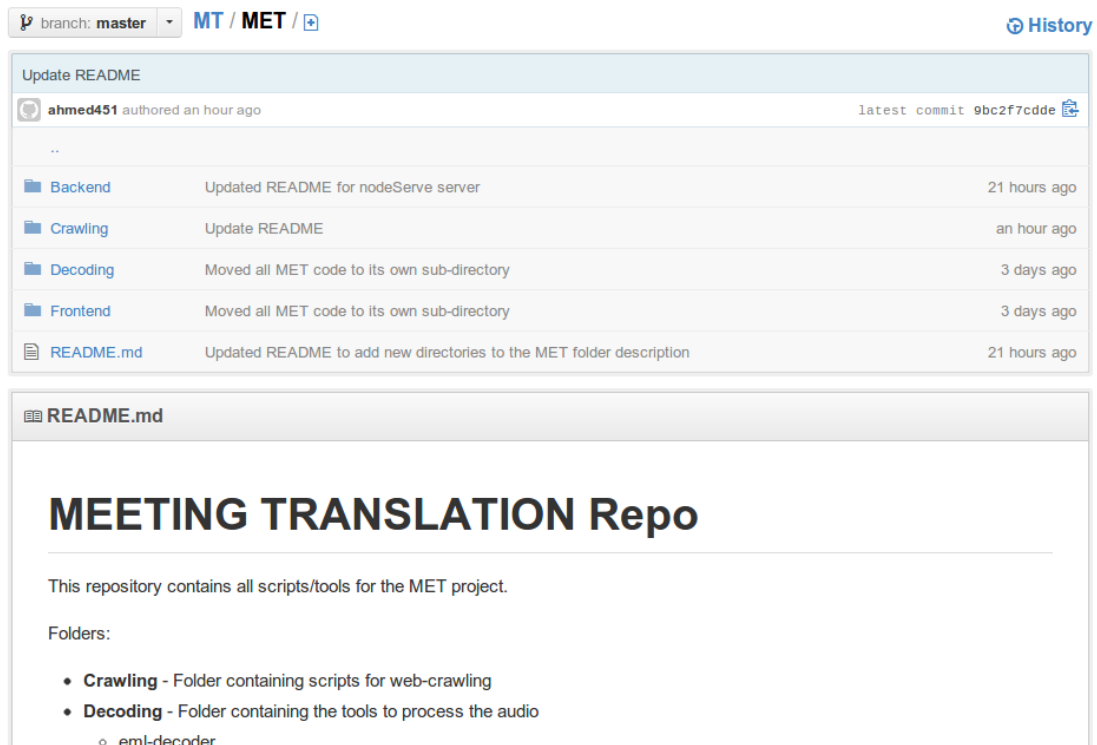
## 2. Segmentation Fault in NodeProcess

- The NodeProcess server uses the C++ bindings from the NativeWebRTC code. This code segfaults on rare occasions. The exact cause of this has not been determined, as attempts of reproducing this usually fail. Running the process under *gdb* revealed that the segfault happens in the *MakeFastBuffer* function, part of *node\_buffer.cc*.
- A temporary fix is to use the node-forever module, which automatically restarts a server when it crashes, and logs all output for future examination. This is a disruptive fix, as in the clients would suddenly notice that their audio is not sent anymore. They can ofcourse unmute their microphone again and communications should resume.

# Other tasks

Apart from creating the backend, several other goals were achieved during the internship period, two of which have been described below:

## 1. Code restructuring and organization



The screenshot shows a GitHub repository interface. At the top, it indicates the current branch is 'master' and the repository name is 'MT / MET'. A 'History' link is visible in the top right. Below this, a commit history table is displayed, showing the following entries:

Commit	Message	Time
ahmed451	authored an hour ago	latest commit 9bc2f7cdde
Backend	Updated README for nodeServe server	21 hours ago
Crawling	Update README	an hour ago
Decoding	Moved all MET code to its own sub-directory	3 days ago
Frontend	Moved all MET code to its own sub-directory	3 days ago
README.md	Updated README to add new directories to the MET folder description	21 hours ago

Below the commit history, the 'README.md' file is shown. The title is 'MEETING TRANSLATION Repo'. The content of the README is as follows:

This repository contains all scripts/tools for the MET project.

Folders:

- **Crawling** - Folder containing scripts for web-crawling
- **Decoding** - Folder containing the tools to process the audio
  - eml-decoder

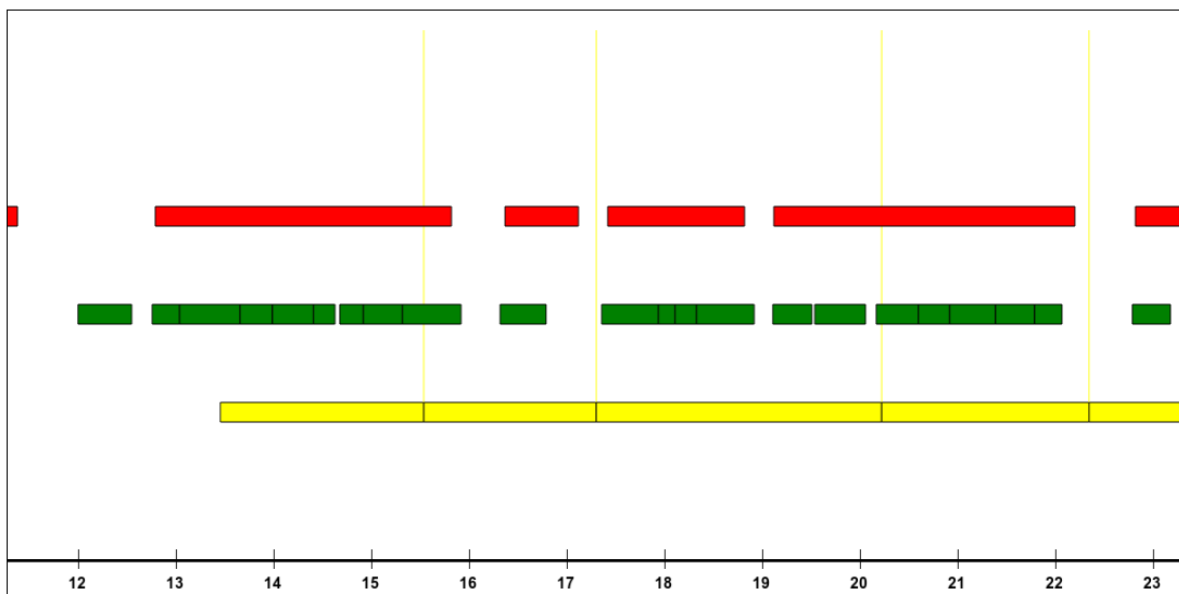
- The codebase was originally placed under the MT repositories, with different parts of the system in different folders with no instructions as to what belongs where. The codebase was successfully reorganized into a new repository, where system modules were grouped by their position in the system. For example, all server side code went into the 'Backend' folder, while all the Recognition and transcription parsing code went into the 'Decoding' folder. Documentation was created for each of these modules. Instructions were also added for each of the server-side components regarding the per-requisites for their setup and compilation.

## 2. Visualizer module for SRT-Word alignment

- A lot of Arabic data is required to train the Meeting Translation system to recognize Arabic speech. Part of this data comes from the AMARA database,

which has videos with subtitles created by volunteers. Another member in the team was investigating the usefulness of this data, and a roadblock that was encountered in the process was that the subtitles were not aligned properly with the speech (partly due to human error and lack of high-quality tools for subtitle creation). Hence, a simple module was created that allowed a user to see the information received from the subtitle file, speech recognizer, and a simple segmenter (that separates speech based on silence).

- This tool helped investigating the similarities and differences in a visual form, and this proved to be a much quicker method of identifying issues rather than manually keeping track of the timestamps of different segments.



# Conclusion

In conclusion, the Meeting translation project has grown from a proof-of-concept to a full-blown multi-lingual, multi-user and multi-meeting system. The technologies used by the system are emerging as standards in each of their fields, which will allow us to deploy the system quickly and efficiently on multiple platforms in the near future, with little or no extra effort.

A lot of knowledge was acquired in the process of building the system, especially regarding WebRTC. Since WebRTC is quite new, the support available was limited, and there were more problems out there than solutions regarding this technology. This posed as a very interesting challenge, and also pushed the limits to what I personally had to do. A lot of reading and research was involved in this process, which has better equipped me now to deal with this new and emerging technology. It was also very satisfying and rewarding to see the complete system actually work all the way from speaking to translation.

Another important technologies that was involved in the project was MySQL. MySQL is used in a lot of major systems, and it was very interesting to learn the tidbits and little things involved in managing such databases. The most important lesson learned in this case was that before the database is even created, it is important to layout the entire schema on paper first, as this helps avoid a lot of redundancy and enhances the overall structure of the database. All in all, this was a very fruitful period with a lot of knowledge involved.

# References

- A new look for Google Translate. (n.d.). *Google Blog*. Retrieved July 29, 2013, from <http://googleblog.blogspot.com/2011/01/new-look-for-google-translate-for.html>
- Getting started - WebRTC. (n.d.). *WebRTC*. Retrieved July 29, 2013, from <http://www.webrtc.org/reference/getting-started>
- Introduction. (n.d.). *Verbalizeit*. Retrieved July 29, 2013, from <https://www.verbalizeit.com/>
- Multi Language. (n.d.). *omNovia Interactive Webinar and HD Webcasting Platform – Online Training – Marketing – Live Event Webcasting*. Retrieved July 29, 2013, from <http://www.omnovia.com/multi-language/>
- Rantanen, M. (n.d.). Node-Peerconnection. *GitHub*. Retrieved July 29, 2013, from <https://github.com/Rantanen/node-peerconnection>
- WebRTC-Overview. (n.d.). *WebRTC*. Retrieved July 29, 2013, from <http://www.webrtc.org/>
- WebRTC 1.0: Real-time Communication Between Browsers. (n.d.). *W3C Public CVS Repository*. Retrieved July 29, 2013, from <http://dev.w3.org/2011/webrtc/editor/webrtc.html>