Live Video Broadcasting with Chat Rooms - Enigma
CS425 Course Project

Anuj Nagpal 14116
Anmol Porwal 150108
Chauhan Mayurdhwajsinhji Nitiraj 14195
Group 11

December 4, 2021

Introduction

Enigma is a live-video streaming platform with chat room support that allows a user to broadcast his/her live webcam video/static videos to multiple clients and at the same time chat with them. The user needs to first authenticate before entering any desired room. The application also supports streaming of static content in both audio and video format.

Our application has a wide reach in terms of applications:

- Live-stream content ranges from impromptu video blogging to produced efforts by industry veterans. For small businesses, live-streaming video may be a helpful, new way to engage an audience and establish a brand with little or no cost.
- Content is served in a way that allows files to play almost immediately after user has begun broadcasting. This way information is passed almost immediately without delay. Very useful in case of emergencies declarations.
- Live streaming also helps in piracy protection. Streaming video is harder to copy and prevents others to make a local copy of the data on their computer.
- It allows people to engage in real time so people can understand the current inclinements and thoughts of the uploader (unlike blogs which aren't recent and may serve old content).Hence, they also help better in learning since visual conetent is easier to interpret than written content.Moreover it is more time saving as compared to blogs.

Objective

Our objective was to create an application that can support live/static video streaming to multiple users along with chat room functionality.

Assumptions

- We are supposed to make an application layer project and therefore it is not necessary to implement everything from scratch from network layer and we are free to use libraries and frameworks.
- Video streaming consumes sufficient resources and local machine won't be able to handle too many clients simultaneously. Such performance issues can be handled with availability of good servers and can be ignored for this mini-project.

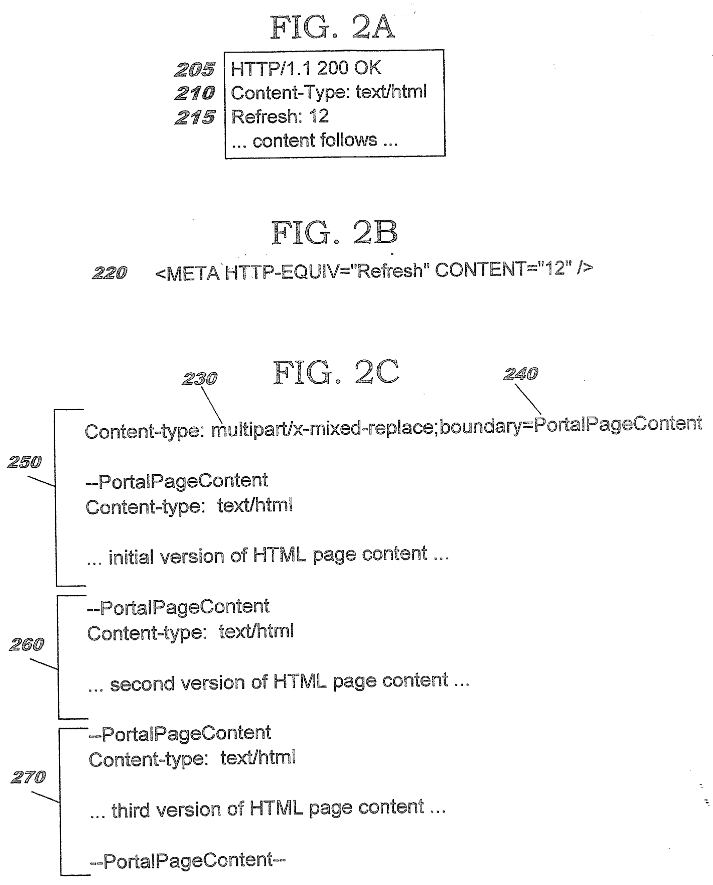
Implementation Environment

The application was made and tested on a personal laptop with 64 bit Linux OS, i5 Processor and 8GB RAM. Following tools and installations are required to run this application:

- Python 2.7 - The implementation programming language
- OpenCV (Open Source Computer Vision) Library - To record frames using webcam
- Flask Python Web Framework - To render HTTP requests on browser using HTML pages.
- Socket.IO library specific to Flask - For our general server-client communication.
- Gunicorn 'Green Unicorn' Python WSGI HTTP Server - To use more robust web server for multistreaming
- Python's rauth library to provide authentication support.

Architecture

- Live video streaming requires that each chunk of video data replace the previous chunk of data which would enable streams to play in the webpage. Hence, here chunk means an image which would allow a video to play in the page.
- The key to this concept is the multipart response. An HTTP multipart request instructs client to send sets of data to the HTTP server. There are various types of multipart responses which instruct how a set of data is to be manipulated in conjuction with another set of data. For our purpose, we have used multipart/x-mixed-replace content type which instructs the server to replace one set with another, thus making a stream.



```
HTTP/1.1 200 OK
Content-Type: text/html
Refresh: 12
... content follows ...
```

```
<META HTTP-EQUIV="Refresh" CONTENT="12" />
```

```
Content-type: multipart/x-mixed-replace;boundary=PortalPageContent

--PortalPageContent
Content-type:  text/html

... initial version of HTML page content ...

--PortalPageContent
Content-type:  text/html

... second version of HTML page content ...

--PortalPageContent
Content-type:  text/html

... third version of HTML page content ...

--PortalPageContent--
```

Figure 1: Structure of Multipart/x-mixed-replace

- Broadly , OpenCV helps in capturing a videocam feed and converting the feed to jpeg images after which Flask streams the data as a set of independent jpeg images which is served on the webpage using HTTP multipart response making it a stream again.

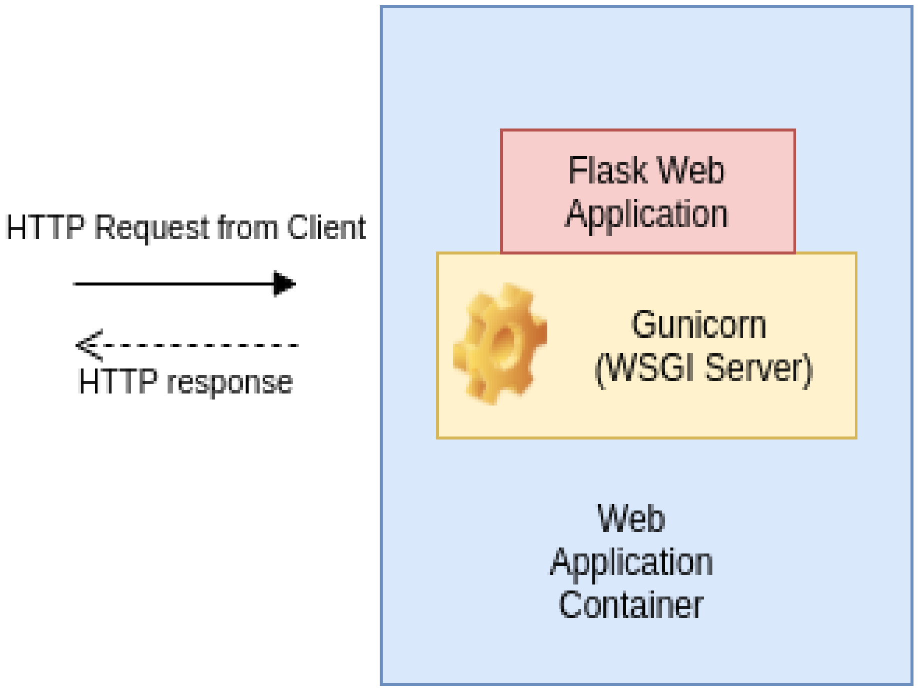


Figure 2: Overview of the Flask Application

- Chat Rooms were possible using flask socket-io libraries. Events and forms are handled using backend python scripts events.py and forms.py

Application Flow

- The user logs into the application using his Twitter account
- Next follows a page that asks the users to provide a user name (which will be public to others in the chat room) and a chat room number he wants to join.
- There you go, a video will be streamed in that chat room with multiple users watching that. You can side by side comment and chat with other users.

Summary

The application allows multiple features listed below:

- Multiple chat rooms allowing Live-video broadcasting from one server node to multiple client nodes while enabling the users at both ends to chat.
- Authentication via the OAuth protocol using Twitter as the third party OAuth provider.
- Support for streaming static audio(mp3) and video files(mp4).

Note : Providing audio feature to the application is a very difficult task. The Motion JPEG format does not support audio. We would need to stream the audio separately, and then add an audio player to the HTML page. Even after doing all this, synchronization between audio and video is not going to be very accurate.

Acknowledgement : Thanks to Miguel Grinberg, his blog and his flask implementations which helped us in completing this project.