The Implementation of an Innopolis Video Call Application Using Python Sockets

Rafik Hachana, Parth Kalkar, Igor Mpore
GitHub Repository

May 2021

Abstract

We describe various implementation aspects of a video network call by using Python sockets. The call is implemented as a peer-to-peer application and relies on TCP communication (except for a shared database that uses a client-server model). The application's video call only works for non-NATed peers, i.e. the call correspondents should either have a public IP address, or be in the same LAN. For the time being, the application only works in Linux environments.

1 Introduction

The project consists of the implementation of a peer-to-peer (P2P) video calling application (that supports both video and audio calls) using TCP sockets in Python. In order for a peer to call another peer, they should both be in the same private network or both should have a unique public IP address. As opposed to other P2P applications, this application does not distinguish a master peer that synchronizes other nodes, instead we use a remote shared database that stores usernames and their corresponding IP addresses. For now, the application does not use any security measures like encryption or authentication, that is due to the time constraints and the primary goal being a simple MVP that shows the basic functionality.
The application supports a quick registration each time a user opens it, with a username and no password. Once the user is registered he can call online users or receive a call. For the time being, the application can only run in Linux environments. However, it can be easily adapted and packaged for devices running Windows or MacOS.
In this report, we will get an overview of the implementation of the application, including the technologies used and the protocols designed for P2P communication, then we will offer a short installation and user guide, and finally we will discuss potential features that can be added and the challenges related to them.

2 Implementation details

In this section, we will go over the implementation details of the application, we will discuss the technologies used and how we used them to make different components, and we will explain how we implemented the P2P communication.

2.1 Technologies used

The technologies used can be classified by the component they belong to. The app has 2 main components: The frontend UI and the core of the application, as well as 2 external components consisting of 2 databases: The Redis local in-memory database and the MongoDB remote database.

2.1.1 Technologies for the application core

Python : Python was used to program the core of the application. Python was chosen primarily because of its simplicity which reduces the required development time, and also because of the team's profile. Many native python libraries were used, such as the `socket` library for implementing the TCP sockets, the `pickle` and `json` libraries for converting objects to bytes or strings before sending them over sockets, the `time` and `datetime` libraries for getting time and sleeping, as well as the `threading` and `multiprocessing` libraries for making processes and threads.
OpenCV library : The OpenCV library was used for capturing frames from the user's camera, and resizing and encoding them.
PyAudio library : The library was used for capturing and playing audio.
PyMongo library : The library was used for connecting to the remote MongoDB database. The library also requires the installation of the `dnspython` library that is used to connect to the database.
Redis library : The library was used to connect to the local Redis library, used for inter-process communication.

2.1.2 Technologies for the frontend UI

HTML, CSS and JavaScript : The UI was implemented using vanilla HTML, CSS and JavaScript, the same way that websites are implemented.
Electron.js package for Node.js : The Electron JavaScript framework was used to make the implemented web interface into a desktop application. It requires the use of a Node.js server.
The Redis package for Node.js : The Redis package was used to write to the Redis in-memory database.

2.1.3 Technologies used for databases

MongoDB : The application uses the MongoDB NoSQL database for storing the user's username, IP address and online status once the user is registered. The database is hosted on a remote distributed server cluster offered by MongoDB's Atlas service.
Redis : Redis is a very fast in-memory database, it is used for inter-process communication between the different components of the application. We can consider it as a shared memory space where all components can read or write data.

2.2 An overall application organization

From the user's perspective, the app has 4 states that correspond to 4 UI screens, the transitions between them is depicted in fig.1.

From a development perspective, the application's UI can be considered as one component, while the core has many

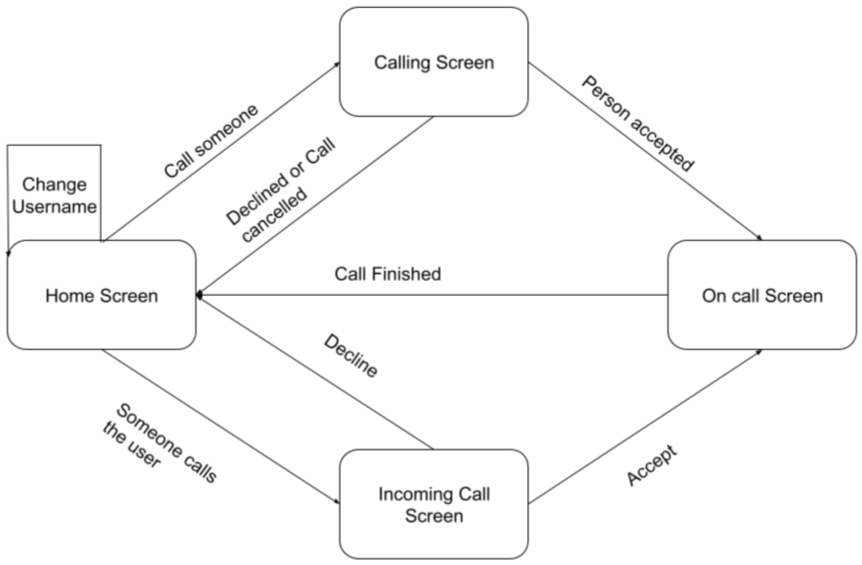


Figure 1: The UI flow diagram

individual components. All components communicate through Redis. A diagram of their relationship is shown in fig.2. We can see that the application core is made of the following components:
Main process: This process is run when the application starts, along with the Electron UI main process. It communicates with the MongoDB database, and it spawns the call listener when the user goes online, and the call listener when the user calls someone. It uses Redis to get the user input from the UI.
Call maker and call listener: These 2 processes make and receive calls respectively. The call maker in a peer connects to the call listener in another peer to initiate a call.
Call components: These components are started by the call maker or the call listener once the call is accepted. They can be divided into:

1. *The control component:* Has a client and server for full-duplex communication. It is responsible for toggling video and audio, and terminating the call.
2. *The video component:* Consists of the video server that streams the user's webcam, the video receiver that receives and buffers the video, and the video player that plays the video from the buffer.
3. *The audio component:* Consists of the audio server that streams the user's microphone, the audio receiver that receives and buffers the audio, and the audio player that plays the audio from the buffer.

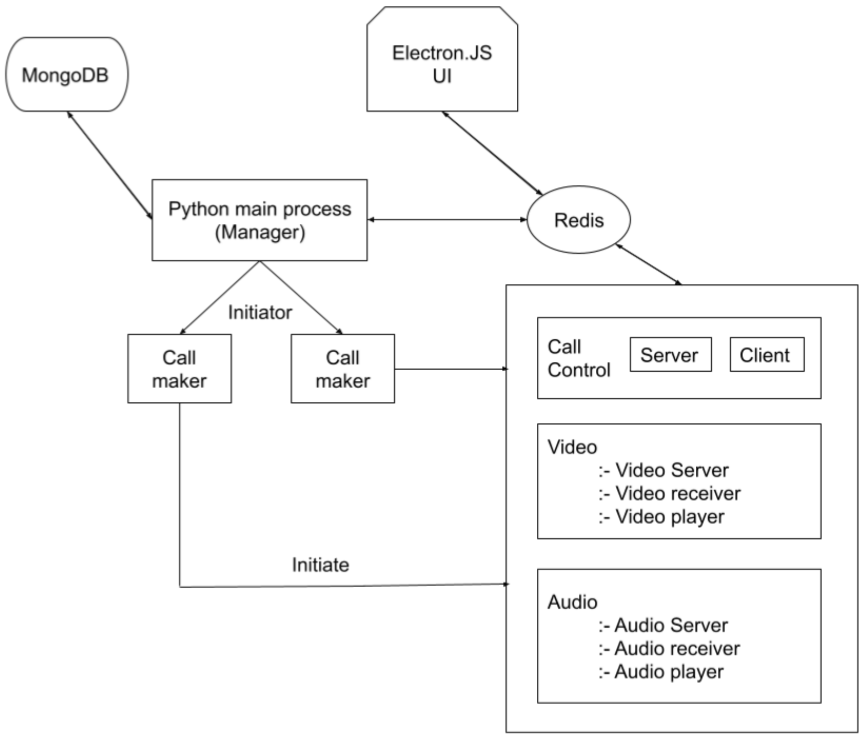


Figure 2: The overall application architecture

2.3 Peer implementation

Now we will discuss the components involved in each state of the application, they are depicted in fig.3. When the application is started the Manager Main Process waits for the user to input their username, then it starts the call listener and regularly fetches the list of online users from MongoDB. At this point, the user becomes an online peer that other peers can communicate with.
If the user makes a call, the call listener is stopped and the call maker is started with the IP address of the person to call. If the user receives a call, the call listener handles it. In both cases, the call listener or call maker terminate right after starting the call components. While on call, the call components control everything by using Redis. Once the call is finished, the call components terminate and the manager process takes control, and starts the call listener again.

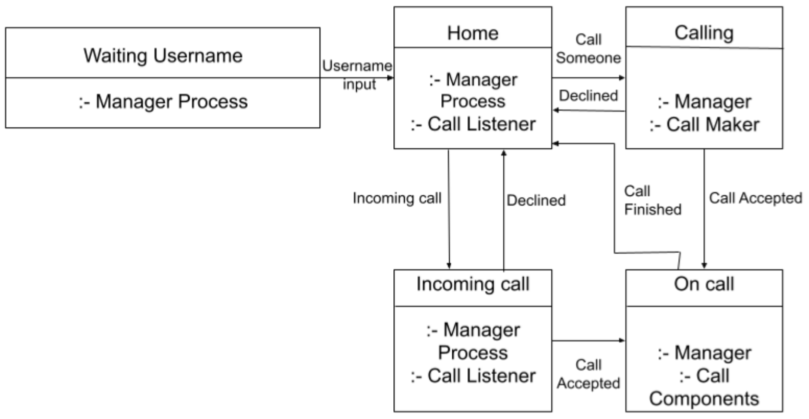
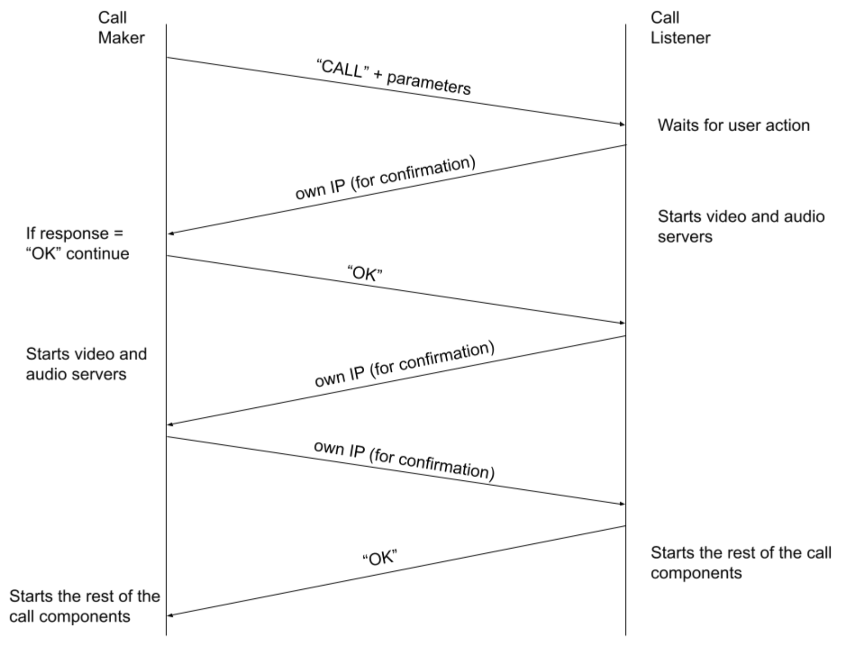


Figure 3: FSA of the application core, with components involved in each step

2.4 Peer-to-peer communication

Here we discuss the different communication protocols designed for P2P communication.



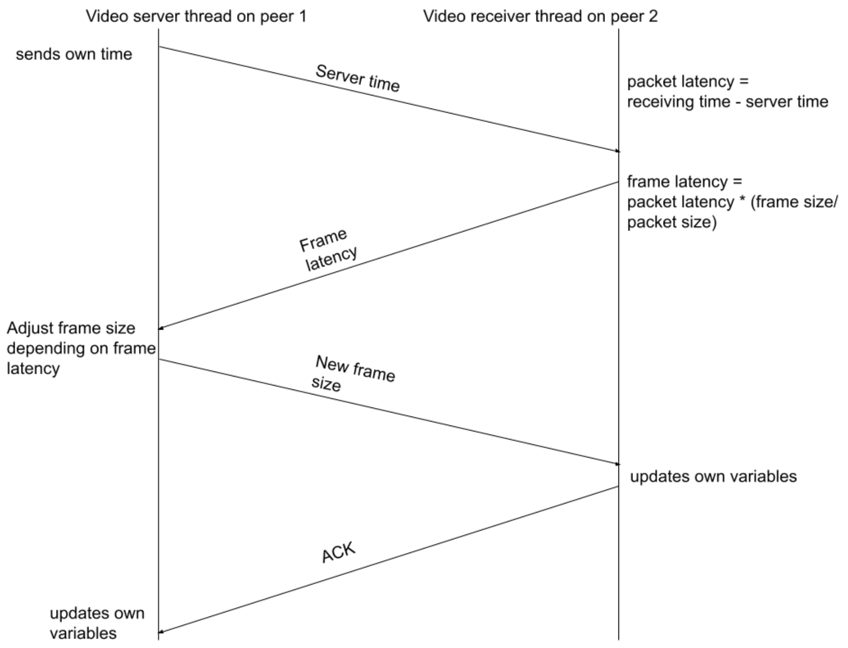
Call Maker
Call Listener
"CALL" + parameters
Waits for user action
own IP (for confirmation)
Starts video and audio servers
If response = "OK" continue
"OK"
Starts video and audio servers
own IP (for confirmation)
own IP (for confirmation)
Starts the rest of the call components
"OK"
Starts the rest of the call components

Figure 4: The call initation protocol

2.4.1 Call initiation protocol

When the peer makes a call, his call making process connects as a client to the other peer's call listener that has a server-side TCP socket. Then the protocol depicted in fig.4 is followed. Once the protocol is finished, each video receiver starts 20 parallel TCP to the other peer's video server, and each audio receiver starts 10 parallel TCP connections (in separate threads).

2.4.2 The adaptive frame size protocol



Video server thread on peer 1
Video receiver thread on peer 2
sends own time
Server time
packet latency = receiving time - server time
frame latency = packet latency * (frame size/ packet size)
Frame latency
Adjust frame size depending on frame latency
New frame size
updates own variables
ACK
updates own variables

Figure 5: The adaptive frame size protocol

The application supports an adaptive frame size feature, that reduces the video quality when the video latency is high. It works as in fig.5. The video server thread would execute this protocol at regular time intervals, and it uses the frame latency to approximate the video latency and adapt the frame size using an upper bound of tolerable latency.

2.4.3 Call control

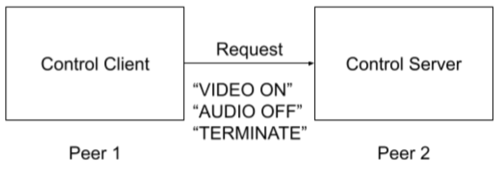


Figure 6: The call control model

During the call, the call control client would send requests to the other peer's call control server as depicted in fig.6. The requests include information about toggling audio and video, as well as the termination of the call.

3 Installation and usage guide

The application can be downloaded and installed on any Linux machine. The steps are as follows:

3.1 Installation guide

1. Download the .zip file of the project's source code here.
2. Extract the .zip file to the location where you want the app to be installed.
3. Open the terminal in the application's directory and run the command `bash install.sh`. This will install all the dependencies and requirements for the application.

3.2 User guide

1. Open the terminal in the application's directory and run the command `bash run.sh`.
2. Once the application's window opens, enter your username and click on "Save". Now you will be able to see the list of online users.
3. If you want to call a user, press the call icon next to their name.
4. Once the other user accepts the call, you will be able to see and hear them.
5. During the call you can toggle your microphone and webcam using the corresponding buttons.
6. To finish the call, use the corresponding button. You will be brought back to the home screen.

You can find the screenshot for the guide here.

4 Future extensions

The current version of the application is a minimum-viable product (MVP) that just supports the basic functionality of a video calling app. Many challenges were ignored which made the product limited in many ways. In this section, we will discuss potential future extensions and features of the app, and the challenges that need to be overcome.
Security: Right now, the application does not support any security features. In future versions, *encryption* of the data exchanged and authentication of users using a password will be implemented.
NAT traversal: The NAT (Network Address Translation) mechanism is a big challenge for P2P applications. It resulted in the current version being limited to local networks. Potential solutions include doing a NAT traversal (overcoming NAT routers using a hole-punching technique), using IPv6 (applicable only for users who support it), or having a fallback relay server that will be in the middle-man in the communication.
Better error signaling and handling: This ranges from handling errors and exceptions internally, and showing them to the user (e.g. "Unstable Internet connection", "User not reachable", "Username already taken" ...).
Text chat : Most video call applications have a text chat feature. We might implement it to be available both inside and outside the context of a call.
Video and audio performance: The biggest challenge here is optimizing TCP sockets for media streaming. This includes improving the adaptive quality protocols and algorithms.
Group calls: Another feature that most video call applications support. It poses challenges in term of bandwidth usage

and design decision of the P2P model (Either a master peer model can be used, or each peer would directly connect to every peer in the group call).
Compatibility: The application can be packaged for Windows and Mac users.

5 Conclusion

In this report, we have discussed the implementation details of the *Innopolis video call application*, we have also provided a small user guide for installation and usage and we presented potential extensions and features. For further details, you can check the source code in the GitHub repository linked at the first page.