Innopolis Video Chat: The Implementation of a Network Video Call Using Python Sockets

Rafik Hachana, Parth Kalkar, Igor Mpore
Innopolis University

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. Our implementation is publicly available[1].

1 Introduction

We describe our implementation experience of a custom network video call application called "Innopolis Video Chat", by using Python sockets. Many network video chats are available (e.g. Skype, Zoom, WhatsApp ...), however, the key feature of our tool is its peer-to-peer architecture, that makes it easily scalable. As opposed to many other P2P applications, we do not distinguish a master peer to synchronize all other peers. Instead, we use a different approach based on a remote shared database (storing usernames and IP addresses). Note that, the current implementation doesn't provide any security mechanisms, and it only works on Linux machines.
Next, 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 conclude and briefly mention potential features that can be added and the challenges related to them.

2 Implementation details

We first describe the implementation aspects of our network video chat application.

2.1 Technologies used

The technologies used can be classified into back-end technologies (depicted in Table 1) and front-end technologies (depicted in Table 3). Python is the main technology used on the back-end side, due to its ease of development and the abundance of libraries. The Python core libraries used are listed in Table 2.

Technology	Usage
Python	Network communication, multiprocessing, connection to databases ....
OpenCV	Video processing
PyAudio	Audio processing
MongoDB NoSQL	Store usernames, their IP addresses, online statuses.
PyMongo	To connect to MongoDB.
Redis	Inter-process communication on the same network host, using a shared memory space.

Table 1: Technologies used in the back-end side

2.2 An overall application organization

The application has many components (all of them developed in Python except the UI). All components communicate through Redis. A diagram of their relationship is shown in fig.1. We see that the application core has the following components:

Python libraries	Usage
socket	Network communication through TCP sockets
pickle, json, base64	Data encoding for network and inter-process communication
time, datetime	Yielding threads, calculating packet latency, network flow control
threading, multiprocessing	Concurrent execution of the different components

Table 2: Core Python libraries used

Technologies	Usage
HTML, CSS, JavaScript	The implementation of a web graphical user interface.
Electron.js and Node.js (JavaScript packages)	The implementation of a desktop graphical interface
Redis (JavaScript package)	To provide inter-process communication through Redis in-memory database.

Table 3: Technologies used in the front-end side

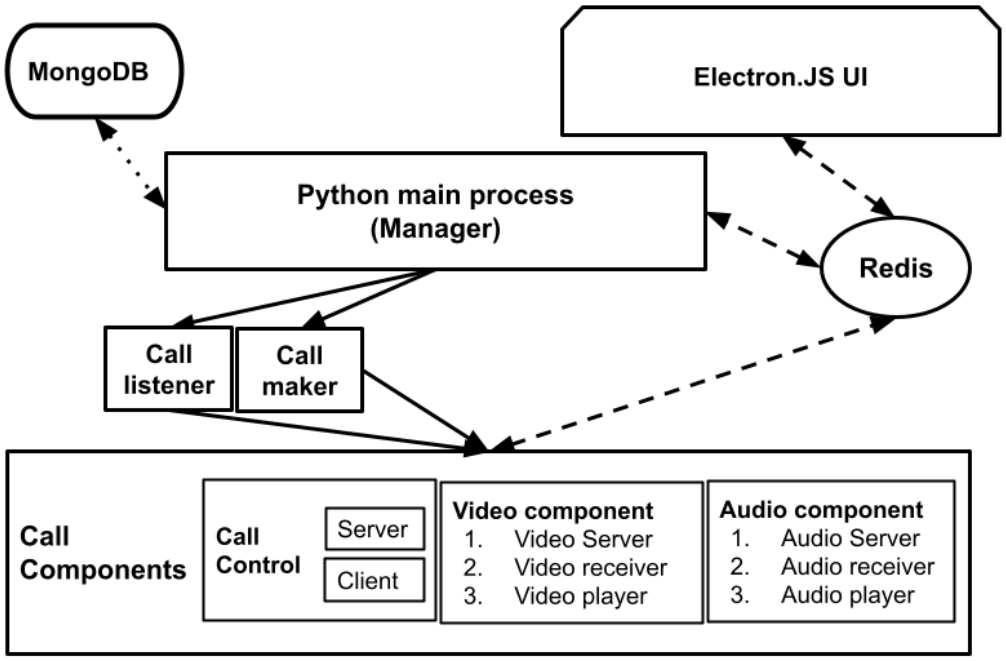

Figure 1: The overall application architecture (Continuous arrows are for components spawning other components, dashed lines are for IPC, dotted lines for remote communication)

Main process: Spawned when the application starts, along with the UI's main process. It communicates with the MongoDB database, and spawns the call listener when the user goes online, and the call maker when the user calls someone. It get user input through Redis.
Call maker and call listener: 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: Started by the call maker or call listener once the call is accepted. They include:

1. *The control component:* 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.

2.3 Peer implementation

Now we will discuss the components involved in each state of the application, they are depicted in fig.2. 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

[1] https://github.com/ParthKalkar/video-chatting-app-TCP

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.

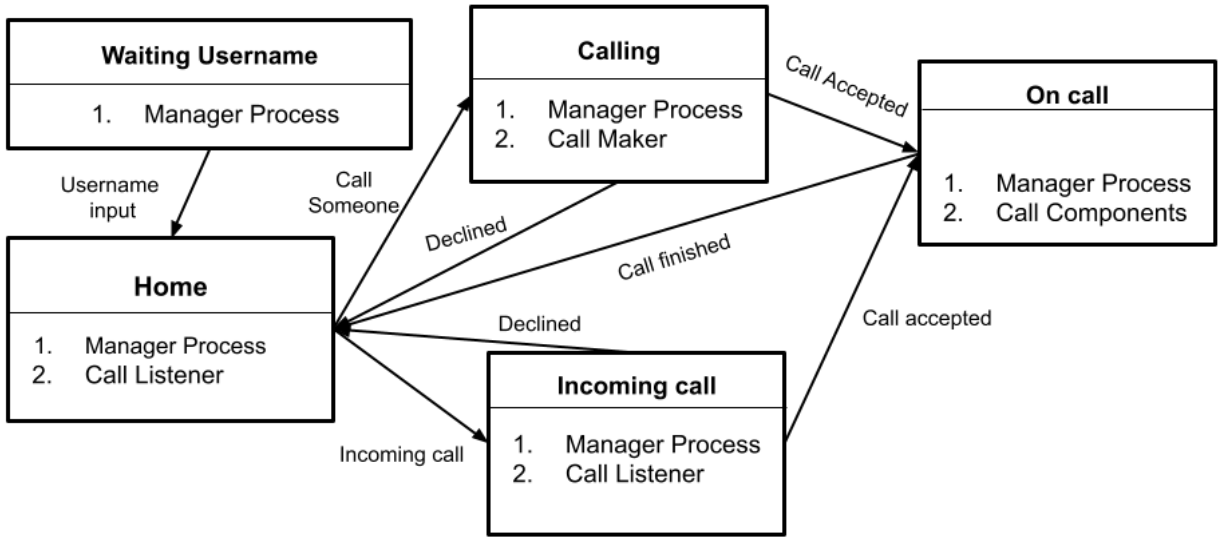

Figure 2: FSA of the application core, with components involved in each step

2.4 Peer-to-peer communication

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

2.4.1 Call initiation protocol

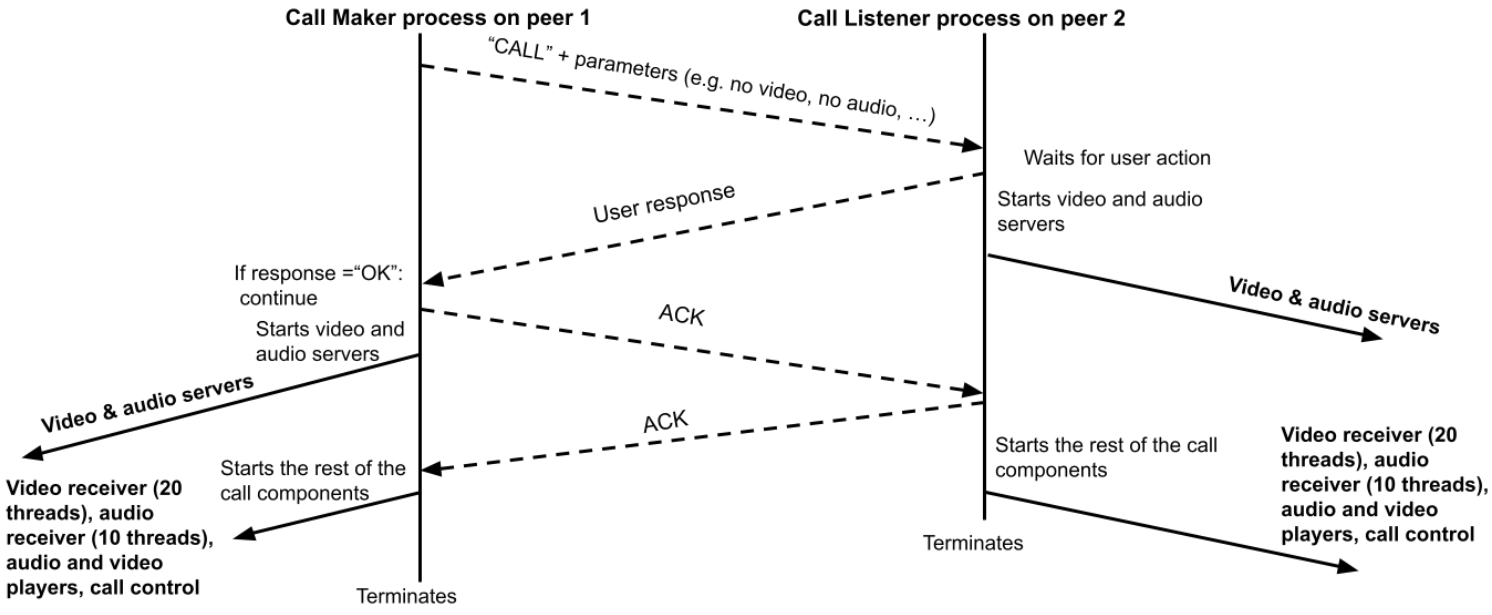

Figure 3: The call initation protocol

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.3 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

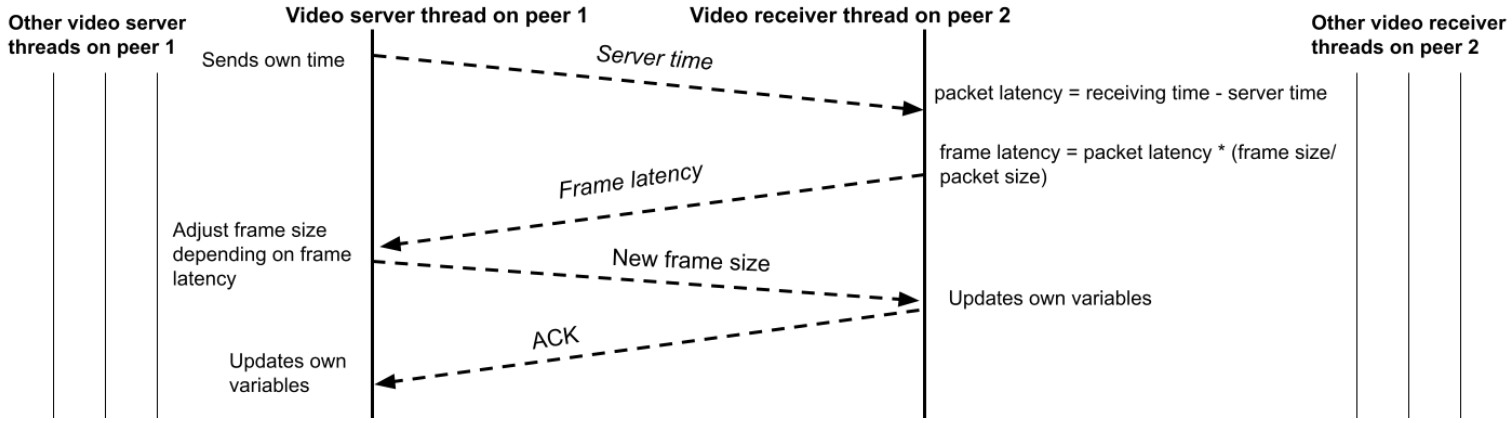

Figure 4: The adaptive frame size protocol

The application supports an adaptive frame size feature, that reduces the video quality when the video latency is high. It works as in fig.4. The video server thread would execute this protocol at regular time intervals, using the frame latency to approximate the video latency and adapt the frame size using an upper bound of tolerable latency. Note that this protocol involves only 1 thread on each host.

2.4.3 Call control

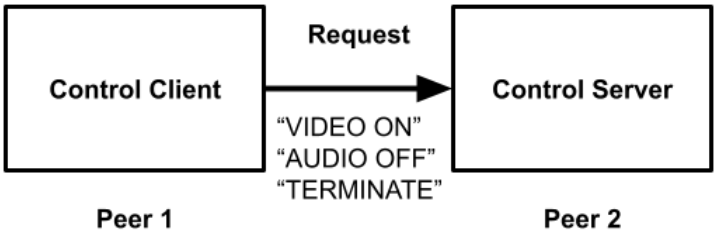

Figure 5: The call control model

During the call, the call control client would send requests to the other peer's call control server as depicted in fig.5. The requests include information about toggling audio and video, as well as the termination of the call.

3 Graphical User Interface

From the user's perspective, the application has 4 states that correspond to 4 UI screens, the transitions between them is depicted in fig.6. Images of the user panels are available publicly[2].

4 Installation and usage guide

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

4.1 Installation guide

1. Download the .zip file of the project's source code [3].
2. Extract the .zip file to the location where you want the app to be installed.

[2] https://github.com/ParthKalkar/video-chatting-app-TCP/tree/main/UI-panels
[3] https://github.com/ParthKalkar/video-chatting-app-TCP/archive/refs/tags/v0.1-alpha.1.zip

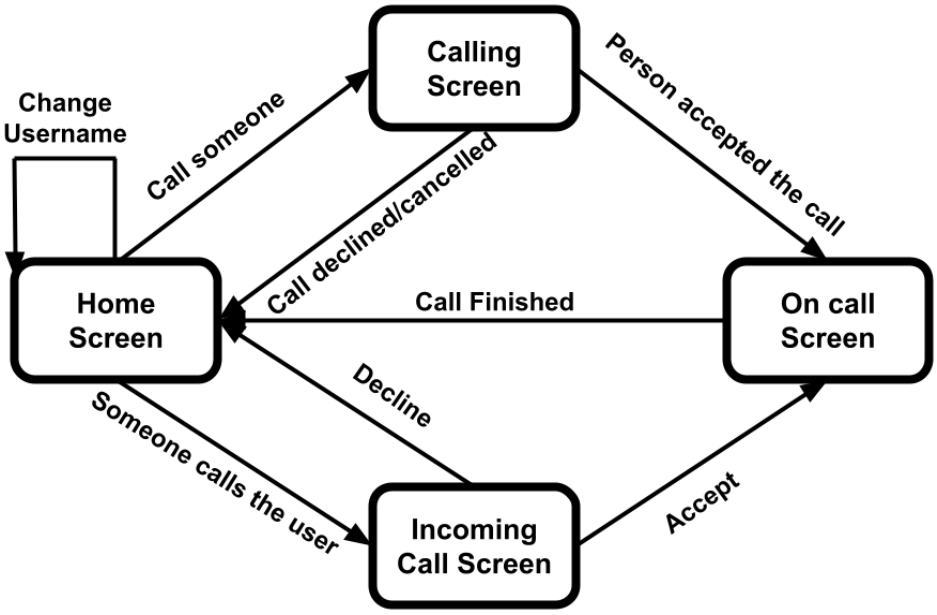

Figure 6: The UI flow diagram

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.

4.2 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.

.

5 Conclusion

In this report, we have discussed the implementation details of the *Innopolis Video Chat* application, we have described the graphical user interface, we have also provided a small user guide for installation.
The current implementation is a prototype of an "Innopolis Video Chat" for video calling. It supports video calls with toggling video and audio. However, there are several limitations that we aim to resolve in the near future. First, we will improve the security of communication by using encryption and authentication . In addition, we aim to introduce the support of a NAT traversal that is essential for many networks. We will improve the overall application reliability, add the support of text messages, introduce group calls, and the compatibility with Windows and MacOS.
For further details, you can check the source code in the GitHub repository[4].

Acknowledgements:
This project has been done in the scope the Networks course for undergraduates in Innopolis University, spring semester 2021.

[4] https://github.com/ParthKalkar/video-chatting-app-TCP