CS305 Computer Networks Project: Remote Meeting

Introduction

Video meetings have become essential in today's digital landscape, transforming how people connect, collaborate, and communicate. With the rise of remote work, online learning, and global partnerships, **video meeting** allows individuals and teams to interact across vast distances, fostering real-time collaboration without the need for physical presence. The implementation of a video meeting project relies heavily on core computer networking principles and protocols, making it an ideal opportunity to consolidate and apply the knowledge gained throughout the course.

Requirements

In this project, you are required to construct a simple **video meeting system** in **Python** that enables participants to communicate over the network using video, audio, and text in real time. This project emphasizes building network frameworks and utilizing network protocols. The system framework is divided into two parts: **Client-Server (CS)** and **Peer-to-Peer (P2P)**. Implementing the system with the **CS** framework forms the basis component, while implementing it with the **P2P** framework is an optional ***bonus*** (details provided later). Meanwhile, a video meeting system relies on several key network protocols that enable real-time audio, video, and text communication, such as **TCP, UDP, SIP, RTP, RTCP, STUN**, and **TURN**. These protocols facilitate the smooth transmission of media streams, ensure security, and manage connectivity between participants. *Determining how to choose and implement these protocols is the primary challenge you will address. Alternatively, you have the option to design your own protocol.*

Basic Part (85%)

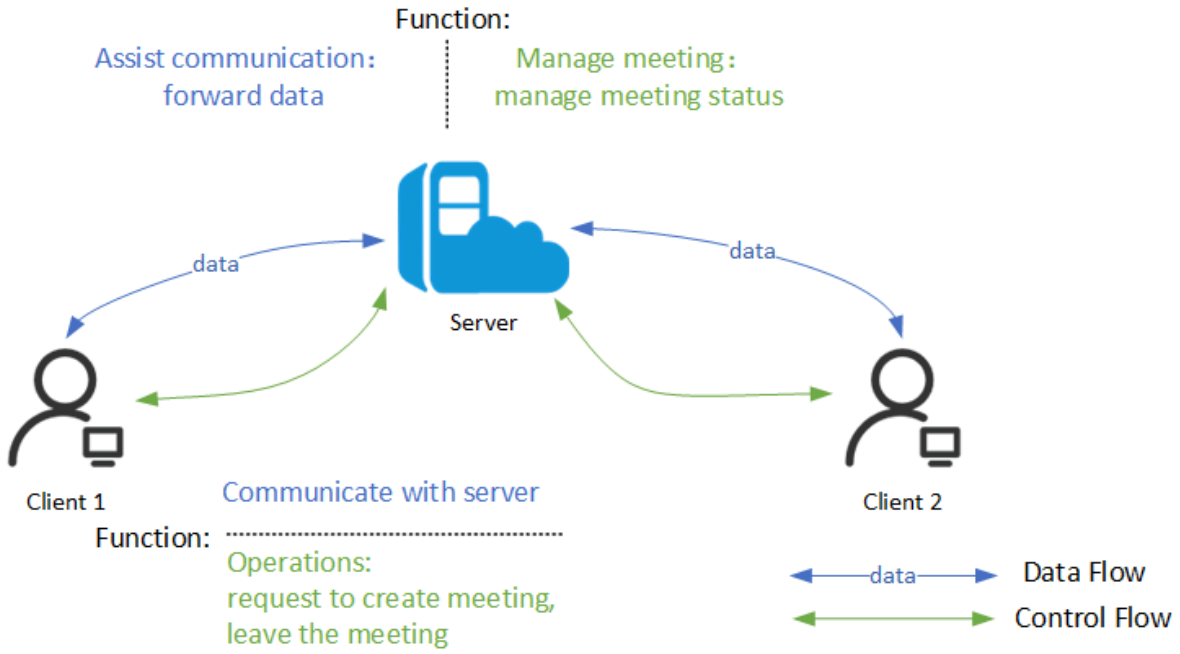
In the basic part, you are required to implement a video meeting system based on **Client-Server (CS)** framework. In this system, the **server** manages and records multiple conference rooms and user operations, while users can act as **clients** to communicate with the server by creating, joining, and exiting meetings.

In a detail, functions of server:

- Manage and record meeting statuses (such as creating and terminating meetings) and track client actions (such as joining and leaving).
- Forward client data to facilitate communication among participants.

Functions of client (Clients are divided into creators and participants):

- Creators: request to create and terminate meeting.
- Participant: join and exit meetings.
- All clients: communicate with the server.



In each meeting, all users can send video, audio, and text messages at the same time. The server is responsible for handling the **broadcast** of this information and ensuring that each client can correctly receive and display content from other users. The specific requirements are as follows:

Basic transmission function (25%)

- **Text transmission(such as chat messages) function (5%)**
  - Each client can send the text messages to server.
  - The server forwards the text messages to all other participants in the meeting.
  - The client should display all text messages from senders.
  - Each message should include the sender's name and the time-stamp to help participants quickly identify the source of the information.
- **Video transmission function (10%)**
  - Each user can choose to turn their camera on or off. Once the camera is activated, the client should send the video stream to the server.
  - The server forwards the video stream to all other participants in the meeting.
  - The client should display the screens of all users who have their cameras turned on and can show real-time video from multiple users using a split-screen or similar methods.
- **Audio transmission function (10%)**
  - Each user can choose to turn their microphone on or off. Once the microphone is activated, the client should send the audio stream to the server.
  - The server must receive and mix the audio streams from multiple users, distributing the combined audio stream to all participants.
  - The client should play audio from other users and be capable of handling the play of multiple audio streams simultaneously.

Note: Streaming data processing is a difficult aspect of the project. Grades will be assigned based on performance, with smoother display and play resulting in higher scores during the presentation.

Meeting function(60%)

Single meeting mode (20%)

In this mode, **two client** will interact with **a server**: one client requests to create a meeting and joins it, while the other client also joins the same meeting.

- **Create a Meeting (5 points)**: The client can send a request to the server to create a new meeting. Once the meeting is successfully created, other users will be allowed to join.
- **Join a Meeting (5 points)**: The client can view a list of currently available meetings and choose to join any of them. After joining, the client will participate in the meeting.
- **Exit the Meeting (5 points)**: The client can exit the current meeting at any time. Leaving the meeting will not affect the ability of other client to continue participating.
- **Cancel the Meeting (5 points)**: The creator of the meeting can cancel it at any time. Once the meeting is canceled, no new clients will be able to join, and those already in the meeting will be removed. Additionally, the meeting will automatically end when all participants, including the creator, have exited.

Complex Meeting Scenarios (40%)

1. **Multiple participants participate in a meeting (20%).** The meeting system must support the simultaneous participation of **multiple clients**(at least 3 clients) to ensure smooth communication through audio, video, and text. This extended version of the single meeting mode emphasizes that each meeting must be capable of supporting multiple client participants (*tips:* ***asynchronous io or multi-thread***).
2. **Multiple conferences in parallel (20%).** The server should support multiple meetings occurring at the same time. The founders of each meeting can be different, allowing various users to create independent meetings. The specific requirements are as follows:
    - **Independent Meeting Life Cycle (10%)**: The start and end of each meeting should be independent of one another, ensuring that the status of one meeting does not affect other meetings. **All meetings should be able to proceed normally**.
    - **Content Isolation (10%)**: The content of each meeting should be completely isolated from others; no meeting should receive audio or video data from other conferences. The server must ensure that audio and video streams are transmitted exclusively between users in the same meeting, thereby maintaining the privacy and independence of each conference.

Bonus Part (20%)

In the bonus part, you are required to implement a video meeting system based on **Peer-to-Peer (P2P)** framework. This system is similar to the **Client-Server (CS)** system, but in this case, the server's role is limited to managing and recording meetings **without forwarding data.** The specific requirements are as follows:

- **Two clients scenarios(10%)**: The client can request to create a meeting. In this meeting, two clients communicate directly with each other.
- **Mode switch(10%)**: The server can determine the most suitable mode for different meetings and switch accordingly between **Client-Server (CS)** and **Peer-to-Peer (P2P)** (For instance, when two clients are in a meeting, it operates in P2P mode, and when three clients are in a meeting, it switches to CS mode.).

Note: The bonus part is relatively loose, and you can do it yourself. But at the same time, how to show that your data is not forwarded by server in P2P scenarios and whether the mode is switched in different scenarios are also issues you need to consider.

Presentation (5%)

You will present your project in the lab class **after December 20**. Each group has a maximum of **10 minutes** to showcase their work. TAs may ask **questions** about your project during the presentation, so a thorough understanding of your implemented functions is essential.

Provided Code

We have provided demo code for you to further understand the fucntions of **server** and **client**, as well as the system structure (see Demo for Video Conference). It includes four files `conf_client.py`, `conf_server.py`, `config.py`, `util.py`.

Note that this demo is just a reference. You don't need to follow it exactly and are encouraged to implement a more efficient video conference service.

`conf_client.py`

There is a `ConferenceClient` class, with necessary fields to record client status or meeting information. Following are some necessary function for client, each with description for its functionality. All a client need to do are **create connections**, and **do transmission** (sending and receiving).

In the `start()` method, we have provided a simple command line user interface to execute certain function. But we won't mind that some of you can provide a better GUI for your clients.

`conf_server.py`

This is a simple demo for server with original implementation of **asynchronized IO** (`asyncio`). There are two classes `MainServer` and `ConferenceServer` for possible **multi-conference** implementation, where `ConferenceServer` handles clients within a single conference and `MainConference` manages multiple conference by handling multiple instances of `ConferenceServer`.

Note that any conference (with mode C-S or P2P) should be managed by server, while the data transmission among clients depends on the mode of conference.

`config.py`

This file stores some necessary configuration of the conference service, including but not limited to the service ports, server's IP address and other parameters to control the stream data and its transmission.

`util.py`

This file includes some useful functions for data capture and image processing.