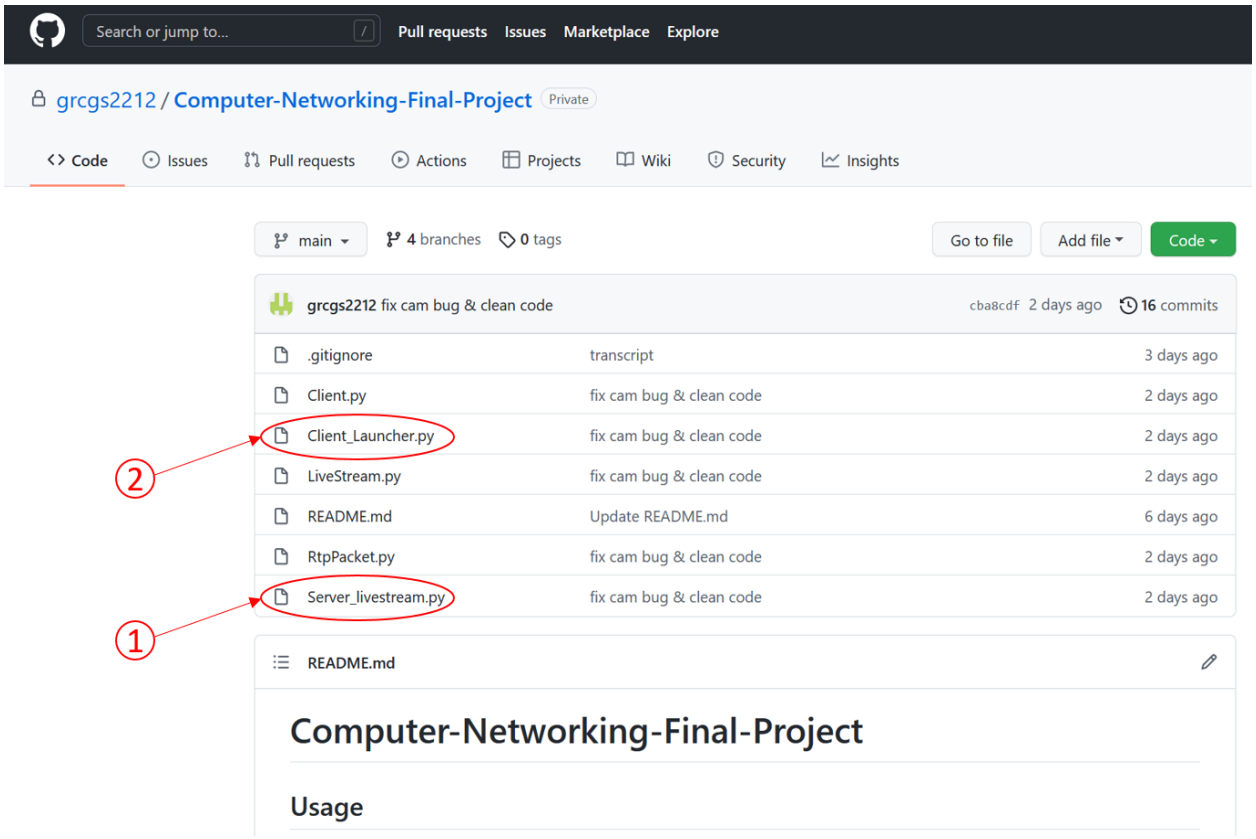
電腦網路導論期末報告

劉知穎 B07901039
陳韋丞 B07901060
廖甜雅 B07901113

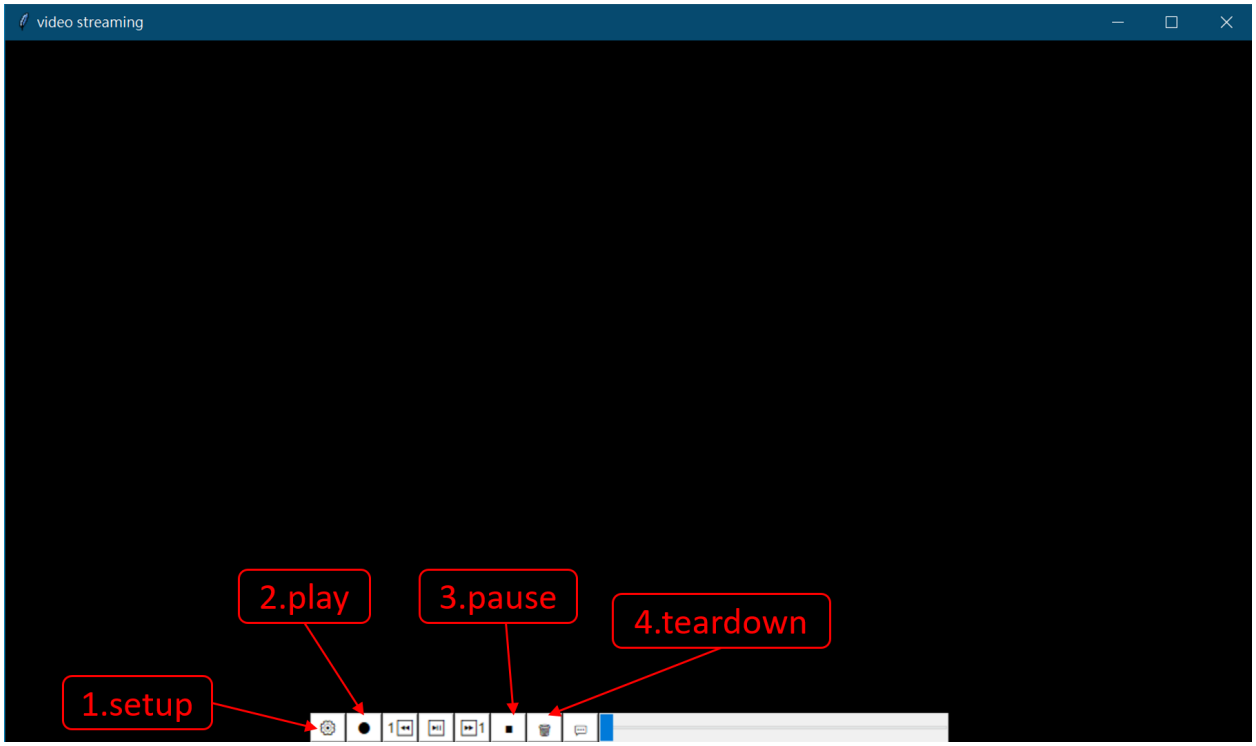
簡介

本次期末專題我們做了一個視訊通話的程式。起初我們先根據期末規定寫了一個video streaming的程式，並增加了回放以及進度條的功能，成功後，我們將其進一步改進成live streaming並增加了聲音傳輸；如此，我們便打下了視訊通話的基礎，其後我們為了解決通訊品質不穩定的問題，又增加了開啟字幕的功能，經過一番測試後，最終成功實現了兩人的視訊通話。惟通訊延遲平均大概2.5秒，仍有改進的空間；但大體而言溝通無礙。詳請請見我們的demo影片。
Computer Networking Final Project Demo

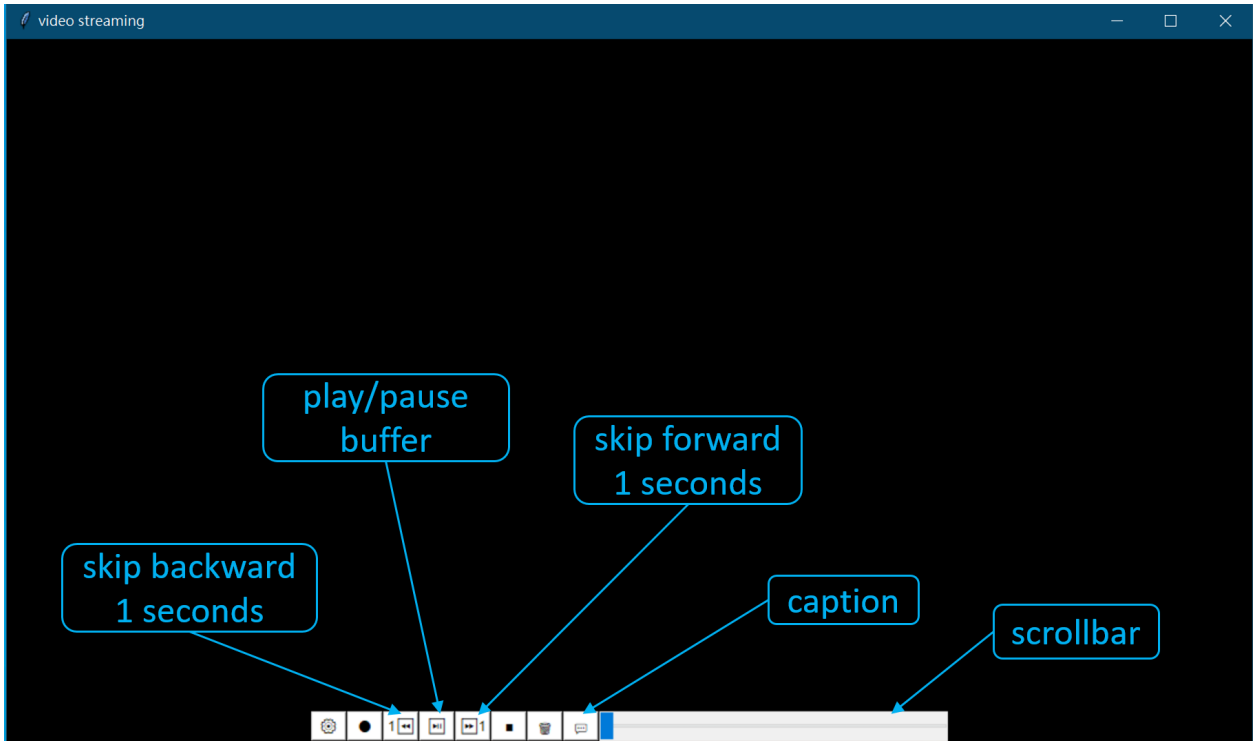
使用介紹



本次視訊通話程式在Python環境下運行，會使用到numpy、opencv-python、PyAudio、以及SpeechRecognition等模組。安裝完畢之後，播放者先執行Server_livestream.py檔。待terminal顯示RTSP socket listening...之後，接收者再執行Client_Launcher.py檔，便會跳出如下視窗。



根據RTSP protocol, client可以向server傳送setup、play、pause、teardown四種指令，我們以不同的符號來表示這些功能。另外，為了達成直播回放的功能，我們設計先將server回傳的資料儲存在buffer之中，而不直接播放。因此，協定中的play更趨像是錄製功能而非播放功能，故以錄製符號表示；而pause則更趨近於停止錄製功能，故以停止符號表示。



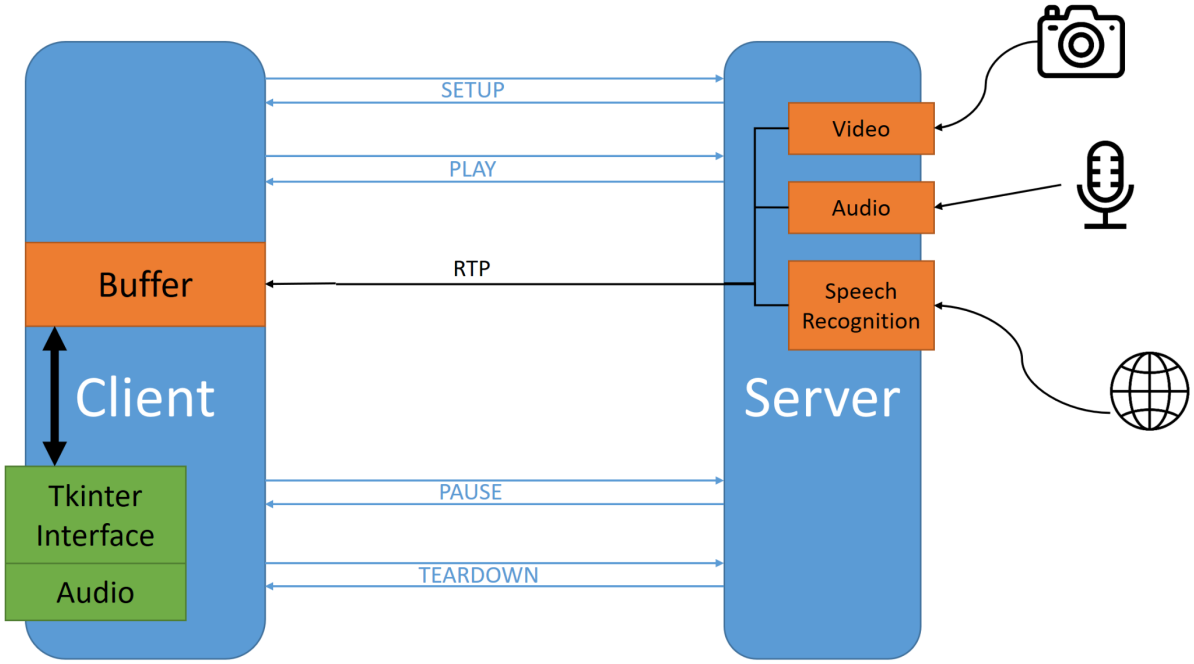
如上所述，我們設計先將直播資料錄製於buffer儲存，所以接收者需點擊上圖中間play/pause按鈕來播放或暫停影片。我們希望接收者可以有很高的自由度，故設計了快進按鈕、倒退按鈕、以及進度條(scrollbar)讓使用者可以任意調整自己想看的直播片段。另外，因為通訊品質會受限於網路環境，有時可能會聽不清楚對方在說甚麼，因此我們還設計了字幕功能。點擊caption符號開啟字幕功能後，server端會辨識播放者的聲音並轉換成文字，然後傳送給接收者。

如圖為實際直播畫面。可以注意到，若我們可以另外開通一個logical channel，並交換播放者與接收者，便能實現視訊通話的功能。操作方法僅需兩方交換，其餘不變。

上圖為我們實際視訊通話的成果，平均延遲大概2.5秒左右。詳情請見我們的demo影片。

程式介紹

在這個程式中，直播資料的提供者為Server端，直播資料的接收者為Client端。兩者以TCP傳遞請求與回應訊息，使用RTSP protocol。在播放直播的過程中，Server端的直播畫面、聲音、語音辨識字幕是以RTP傳送給Client端，而Client端處理從Server端接收的直播資料，將畫面顯示於以tkinter建立的使用者介面，同時播出音訊。



<RTP packet>

```python
import sys

HEADER_SIZE = 2 # only consider 2 bytes seq num

class RtpPacket:
    def __init__(self):
        self.seqnum = None
        self.payload = None

    def encode(self, seqnum, payload):
        self.seqnum = bytearray(seqnum.to_bytes(HEADER_SIZE, sys.byteorder))
        self.payload = payload

    def getPacket(self):
        return self.seqnum + self.payload

    def decode(self, bytestring):
        self.seqnum = bytestring[:HEADER_SIZE]
        self.payload = bytestring[HEADER_SIZE:]

    def getSeqNum(self):
        return int.from_bytes(self.seqnum, sys.byteorder)

    def getPayload(self):
        return self.payload
```

RtpPacket.py為Server端與Client端皆會使用到的檔案，檔案中定義class RtpPacket，包含RTP packet結構 -- header（序列號）+payload，以及encode、decode的方法。

<Server>

Server端使用的Python程式碼包含Server_livestream.py、RtpPacket.py與LiveStream.py。其中Server_livestream.py為主程式；RtpPacket.py如前面所述定義RtpPacket；LiveStream.py定義LiveStreamVideo與LiveStreamAudio兩個class。以下是LiveStream.py的內容與解釋。

```python
import cv2
import pyaudio
import time
from PIL import Image
import numpy as np


class LiveStreamVideo:
    def __init__(self):
        self.framNum = 0
        self.cap = cv2.VideoCapture(0)

    def getNextFrame(self):
        self.framNum += 1
        ret, frame = self.cap.read()
        try:
            frame = Image.fromarray(frame)
            frame = frame.resize((160, 120))
            frame = np.array(frame)
            frame = frame.tobytes()
            print('----- Next Video Frame (# {}), length: {} -----'.format(self.framNum, len(frame)))
        except: pass
        return frame
```

LiveStreamVideo負責接收設備攝影機所讀取的畫面。初始化時會開啟攝影機鏡頭；方法getNextFrame()會回傳當下視訊畫面的byte array，為避免攝影機鏡頭畫質過高導致訊息過長無法順利傳送，在讀取畫面後會先轉換成160*120的大小再加以編碼回傳。經過測試，視訊畫面為每秒30幀。

```python
class LiveStreamAudio:
    def __init__(self):
        self.framNum = 0
        self.NUM_FRAME_PER_CHUNK = 10
        self.RATE = 44100 # num frames per second
        self.CHUNK = int(44100 / 30 * self.NUM_FRAME_PER_CHUNK) # num frame
        self.FORMAT = pyaudio.paInt16
        self.CHANNELS = 2
        self.buffer = []
        self.p = pyaudio.PyAudio()
        self.stream = self.p.open(format=self.FORMAT,
                channels=self.CHANNELS,
                rate=self.RATE,
                input=True,
                frames_per_buffer=self.CHUNK)
        # self.stream.stop_stream() # for the first chunk delay

    def getNextChunk(self):
        # self.stream.start_stream()
        data = self.stream.read(self.CHUNK)
        self.buffer.append(data)
        # self.stream.stop_stream()
        self.framNum += self.NUM_FRAME_PER_CHUNK
        print('\t\t----- Next Audio Chunk (# {}), length: {} -----'.format(self.framNum, len(data)))
        return data
```

LiveStreamAudio負責接收麥克風所讀取的聲音。初始化時會開始接收麥克風聲音；方法getNextChunk()會回傳self.CHUNK大小的一段音訊。由於音訊無法以太短的單位接收，否則會導致Client端的聲音聽起來斷斷續續，大幅降低音質，因此每次傳送約10幀視訊長度的音訊，由此計算 self.CHUNK =（每幀視訊時間與每幀音訊時間的比值）* 10 = (44100 / 30) * 10，取整數。

以下為主程式Server_livestream.py的內容。為便於理解，下圖先展示Server_livestream.py 開始執行時首先執行的內容，再解釋其餘部分的程式碼。

```python
if __name__ == '__main__':

    HOST, PORT = '127.0.0.1', 8888 # For local network (server and client seperate on 2 computers)
    # HOST, PORT = '127.0.0.1', 8888 # For one host (on the same computer)
    rtsp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # RTSP: TCP socket
    rtsp_socket.bind((HOST, PORT))

    # to avoid delay, open live stream before listening
    live_stream_video = LiveStreamVideo()
    live_stream_audio = LiveStreamAudio()
    if not os.path.exists('./cache'):
        os.makedirs('./cache')
    print('RTSP socket listening...')
    rtsp_socket.listen(5)
    while True:
        rtsp_client, addr = rtsp_socket.accept()   # this accept {SockID,tuple object},tuple object = {clinet_addr,intNum}!!!
        ServerWorker(rtsp_client, addr, live_stream_video, live_stream_audio).run()
```

在剛開始執行主程式的時候，建立Server與Client對話的 RTSP socket，並開始等待Client的連線。為避免視訊延遲，在開始listen之前先開啟鏡頭畫面與麥克風聲音，並創建一個cache資料夾以便之後暫存欲進行語音辨識的聲音資料。

```python
import os
import sys
import socket
import threading
from random import randint
from LiveStream import LiveStreamVideo, LiveStreamAudio
from RtpPacket import RtpPacket
import speech_recognition as sr
import wave
import time


class ServerWorker:
    def __init__(self, socket, clientAddr, live_stream_video, live_stream_audio):
        self.rtsp_socket = socket
        self.rtp_socket_video = None
        self.rtp_socket_audio = None
        self.rtp_socket_word = None
        self.rtp_addr = clientAddr[0]
        self.rtp_port_video = None
        self.rtp_port_audio = None
        self.rtp_port_word = None
        self.state = 'INIT' # 'INIT', 'READY', PLAYING'
        self.live_stream_video = live_stream_video
        self.live_stream_audio = live_stream_audio
        self.session = None
        self.event = None
        self.worker_video = None
        self.worker_audio = None
        self.framNum_video = 0 # for aligning video and audio
        self.framNum_audio = 0

    def run(self):
        threading.Thread(target=self.receiveRTSPrequest).start()

    def receiveRTSPrequest(self):
        while True:
            data = self.rtsp_socket.recv(1024)
            if data:
                self.processRTSPrequest(data)
```

與Client成功連線時，初始化所有參數，起始狀態為INIT，並開啟一個新的執行緒聆聽Client的請求。當收到請求時，呼叫processRTSPrequest()以處理請求。

```python
    def processRTSPrequest(self, data):
        print('### RTSP request received: {}'.format(data))
        request = data.decode().split('\n')
        requestType = request[0].split(' ')[0]
        seqNum = int(request[1])
        if requestType == 'SETUP':
            if self.state == 'INIT':
                print('SETUP request received')
                self.state = 'READY'
                self.session = randint(100000, 999999)
                self.rtp_port_video = int(request[2].split(' ')[-3])
                self.rtp_port_audio = int(request[2].split(' ')[-2])
                self.rtp_port_word = int(request[2].split(' ')[-1])
                self.replyRTSP('OK_200', seqNum)
        elif requestType == 'PLAY':
            if self.state == 'READY':
                print('PLAY request received')
                self.state = 'PLAYING'
                self.rtp_socket_video = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                self.rtp_socket_audio = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                self.rtp_socket_word = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                self.event = threading.Event()
                self.worker = threading.Thread(target=self.sendRTP_video_and_audio)
                self.worker.start()
                self.replyRTSP('OK_200', seqNum)
        elif requestType == 'PAUSE':
            if self.state == 'PLAYING':
                print('PAUSE request received')
                self.state = 'READY'
                self.event.set()
                self.replyRTSP('OK_200', seqNum)
        elif requestType == 'TEARDOWN':
            print('TEARDOWN request received')
            self.event.set()
            self.replyRTSP('OK_200', seqNum)
            self.worker.join()
            self.rtp_socket_video.close()
            self.rtp_socket_audio.close()
            self.rtp_socket_word.close()
        else: pass

```

```python
    def replyRTSP(self, code, seqNum):
        if code == 'OK_200':
            reply = 'RTSP/1.0 200 OK\nCSeq: {}\nSession: {}'.format(seqNum, self.session).encode()
            self.rtsp_socket.send(reply)
```

Client可能傳送setup、play、pause、和teardown請求，完成請求後，傳送'OK_200'至Client。
收到setup請求時，切換狀態到READY，並設定收到的視訊、音訊與字幕的rtp port。
收到play請求時，切換狀態到PLAYING，建立視訊、音訊與字幕的rtp socket，並開啟新的執行緒執行sendRTP_video_and_audio()，持續傳送當前視訊、音訊以及字幕內容給Client。
收到pause請求時，回到READY狀態並暫停傳送影音與字幕。
收到teardown請求時，暫停傳送影音與字幕，待傳送資料的執行緒結束之後關閉所有socket。

```
83      def asr(self, framNum):
84          WAV_OUTPUT_FILE = os.path.join('cache', '{}.wav'.format(self.session))
85          wf = wave.open(WAV_OUTPUT_FILE, 'wb')
86          wf.setnchannels(self.live_stream_audio.CHANNELS)
87          wf.setsampwidth(self.live_stream_audio.p.get_sample_size(self.live_stream_audio.FORMAT))
88          wf.setframerate(self.live_stream_audio.RATE)
89          wf.writeframes(b''.join(self.live_stream_audio.buffer[framNum//10-10:framNum//10]))
90          wf.close()
91          r = sr.Recognizer()
92          WAV = sr.AudioFile(WAV_OUTPUT_FILE)
93          with WAV as source:
94              audio = r.record(source)
95          output = r.recognize_google(audio,  language="zh-TW", show_all=True)
96          if len(output) > 0:
97              sent = output['alternative'][0]['transcript']
98          else:
99              sent = ""
100         self.rtp_socket_word.sendto(self.makeRtpPacket(sent.encode(), framNum), (self.rtp_addr, self.rtp_port_word))
101         # print(framNum, sent)
102
```

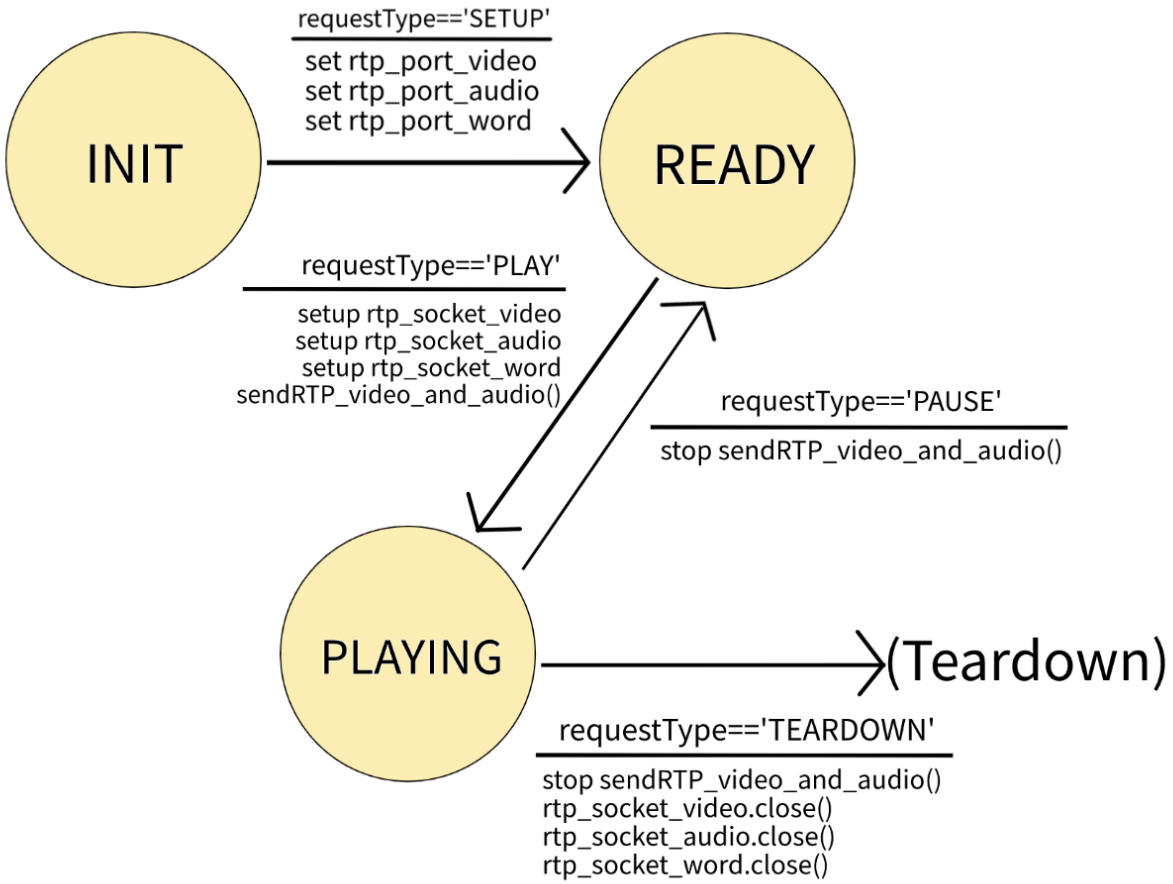
asr()對當前音訊執行語音辨識，並透過字幕port傳送字幕的rtp packet。

```
103     def sendRTP_video_and_audio(self):
104         while  True:
105             tot_start_time = time.time()
106             if self.event.isSet():
107                 asr_thread.join()
108                 break
109             # audio, delay 0.3 sec
110             start_time=time.time()
111             data = self.live_stream_audio.getNextChunk() # delay 0~0.008 sec
112             framNum = self.live_stream_video.framNum
113             if framNum > 9:
114                 asr_thread = threading.Thread(target=self.asr, args=(framNum,))
115                 asr_thread.start()
116             self.rtp_socket_audio.sendto(self.makeRtpPacket(data, framNum), (self.rtp_addr, self.rtp_port_audio))
117             end_time = time.time()
118             print('Total Audio Delay: {}'.format(end_time-start_time))
119             # video, delay 0.002
120             for i in range(10):
121                 if self.event.isSet():
122                     break
123                 start_time=time.time()
124                 data = self.live_stream_video.getNextFrame() # delay 0.002 sec
125                 framNum = self.live_stream_video.framNum
126                 self.framNum_video = framNum
127                 self.rtp_socket_video.sendto(self.makeRtpPacket(data, framNum), (self.rtp_addr, self.rtp_port_video))
128                 end_time = time.time()
129                 print('Total Video Delay: {}'.format(end_time-start_time))
130             tot_end_time = time.time()
131             # print('Total 10 frame delay: {} ({})'.format(tot_end_time-tot_start_time, tot_end_time-tot_start_time-1/3))
132
133     def makeRtpPacket(self, payload, framNum):
134         rtpPacket = RtpPacket()
135         rtpPacket.encode(framNum, payload)
136         return rtpPacket.getPacket()
```

當Server狀態在PLAYING時，會持續執行sendRTP_video_and_audio()，直到pause或teardown請求暫停傳送。其中上圖的109-118行負責接收並傳送當前音訊，同時呼叫asr()辨識與傳送字幕；119-129行負責傳送視訊畫面。因為接收的音訊長度與接收的視訊長度比為10:1，所以在這裡以1段音訊：10幀視訊的頻率傳送聲音與畫面。

makeRtpPacket以framNum作為header，將聲音、畫面或字幕製作成 RTP packet。

下圖為Server的Finite state machine:



<Client>

```python
import sys
import tkinter
from Client import Client


if __name__ == '__main__':

    client_rtp_port_video = 6666
    client_rtp_port_audio = 7777
    client_rtp_port_word = 5555
    filename = "video.mjpeg" # for video streaming
    root = tkinter.Tk()
    root.geometry("1072x603")
    root.configure(background='black')
    app = Client(root, client_rtp_port_video, client_rtp_port_audio, client_rtp_port_word, filename)
    root.mainloop()

```

如上圖，Client_Launcher.py會產生視訊、音訊、和字幕的rtp port，並利用tkinter產生使用者介面。視訊通話不會用到filename。下面我們接著看Client.py檔。

```python
import socket
import time
import threading
import numpy as np
from tkinter import*
from tkinter import messagebox as tkMessageBox
from tkinter import ttk
from RtpPacket import RtpPacket
from PIL import Image, ImageTk
import pyaudio


T_SEC = 1
BUFFER_SIZE = 1000
# SERVER_HOST, SERVER_PORT = '192.168.0.2', 8888 # communication on network
SERVER_HOST, SERVER_PORT = '127.0.0.1', 8888 # on the same host
CLIENT_HOST = '0.0.0.0'

```

如上圖，Client.py會產生五個constant，其中T_SEC表快進或倒退秒數。

```python
class Client:
    def __init__(self, master, rtpPort_video, rtpPort_audio, rtpPort_word, filename):
        ## tkinter
        self.master = master
        self.master.title("video streaming")
        self.master.protocol("WM_DELETE_WINDOW", self.handler) # handle closing GUI window
        self.playMovieButton = False # toggle
        self.tkwindow()
        self.audio()
        ## videostreamer
        self.buffer_video = [] # buffer to realize replay function
        self.buffer_audio = [] # buffer to realize replay function
        self.buffer_word = [] # buffer to realize replay function
        self.playIndex_video = 0
        self.playIndex_audio = 0
        self.playIndex_word = 0
        self.buffNum = 0 # buffer will reset when full, buffNum = Nth buffer we are now at
        self.SHOW_TRANSCRIPT = False
        self.RESET_STATE = False
        ## RTP
        self.rtpPort_video = rtpPort_video
        self.rtpPort_audio = rtpPort_audio
        self.rtpPort_word = rtpPort_word
        self.playRequestEvent = threading.Event()
        self.runEvent = threading.Event()
        ## RTSP
        self.state = 'INIT' # 'INIT', 'READY', 'PLAYING'
        self.sessionID = 0
        self.filename = filename
        self.requestSent = None
        self.SETUP_STR = 'SETUP {}\n1\n RTSP/1.0 RTP/UDP {} {} {}'.format(self.filename, self.rtpPort_video, self.rtpPort_audio, self.rtpPort_word)
        self.PLAY_STR = 'PLAY \n2'
        self.PAUSE_STR = 'PAUSE \n3'
        self.TEARDOWN_STR = 'TEARDOWN \n4'
        self.connectToServer()
        self.tearEvent = threading.Event()

```

Client端會同時開三個thread和buffer處理並儲存視訊、音訊、和字幕資料，第32-34行的playIndex紀錄螢幕上顯示的影片播放位置。

```python
    def connectToServer(self):
        ## RTSP / TCP session
        self.rtsp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.rtsp_socket.connect((SERVER_HOST, SERVER_PORT))

    def openRTPsocket(self):
        ## video
        self.rtp_socket_video = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        self.rtp_socket_video.settimeout(0.5)
        self.rtp_socket_video.bind((CLIENT_HOST, self.rtpPort_video))
        ## audio
        self.rtp_socket_audio = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        self.rtp_socket_audio.settimeout(0.5)
        self.rtp_socket_audio.bind((CLIENT_HOST, self.rtpPort_audio))
        ## word
        self.rtp_socket_word = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        self.rtp_socket_word.settimeout(0.5)
        self.rtp_socket_word.bind((CLIENT_HOST, self.rtpPort_word))

```

如上圖，初始時便會直接呼叫connectToServer()，其利用TCP與server端根據RTSP protocol通訊。setup時會呼叫openRTPsocket()，其會用UDP建立視訊、音訊、以及字幕的rtp_socket。

```
75      def recvRtspReply(self):
76          while not self.tearEvent.isSet():
77              try:
78                  data = self.rtsp_socket.recv(1024)
79                  if data:
80                      print('### RTSP reply recieved: {}'.format(data))
81                      self.parseRtspReply(data)
82              except: continue # handle time out
83
84      def parseRtspReply(self, data):
85          lines = data.decode().split('\n')
86          session = int(lines[2].split(' ')[1])
87          if self.sessionID == 0: self.sessionID = session # initiate
88          if self.sessionID == session and int(lines[0].split(' ')[1]) == 200:
89              if self.requestSent == 'SETUP':
90                  self.state = 'READY'
91                  self.openRTPsocket()
92              elif self.requestSent == 'PLAY':
93                  self.state = 'PLAYING'
94              elif self.requestSent == 'PAUSE':
95                  self.state = 'READY'
96              elif self.requestSent == 'TEARDOWN':
97                  self.state = 'INIT'
98                  self.rtp_socket_video.shutdown(socket.SHUT_RDWR)
99                  self.rtp_socket_video.close()
100                 self.rtp_socket_audio.shutdown(socket.SHUT_RDWR)
101                 self.rtp_socket_audio.close()
102                 self.rtp_socket_word.shutdown(socket.SHUT_RDWR)
103                 self.rtp_socket_word.close()
104         else: print("Session not in order")
105
```

如上圖，只要client不執行tear，recvRtspReply()便會一直接收RTSP資訊。當client接收到RTSP資訊後便會呼叫parseRtspReply()來處理接收資料，並改變client狀態。

```
106     def listenRtp_video(self):
107         while not (self.playRequestEvent.isSet() or self.tearEvent.isSet()):
108             try:
109                 data, addr = self.rtp_socket_video.recvfrom(80000)
110                 if data:
111                     rtpPacket = RtpPacket()
112                     rtpPacket.decode(data)
113                     currFrameNbr = rtpPacket.getSeqNum()
114                     ## late packet
115                     if currFrameNbr < self.buffNum * BUFFER_SIZE + len(self.buffer_video):
116                         continue
117                     ## correct packet
118                     else:
119                         self.buffer_video.append(rtpPacket.getPayload())
120                         if len(self.buffer_video) > BUFFER_SIZE:
121                             self.reset()
122             except: continue # handle out of time
123
```

如上圖，只要沒有pause或tear，listenRtp_video()便會不斷接收server端傳來的視訊資料，並儲存於buffer之中。注意到第120行，BUFFER_SIZE只要記憶體允許的話可以開得非常大，亦即可以將整段直播完全錄下來。但因為我們還沒找到非常有效得對齊視訊與音訊的方式，所以若不讓buffer reset的話，誤差會不斷累積；根據我們的實測，35秒的通話大概會造承3到4秒視訊音訊誤差。

```
124     def listenRtp_word(self):
125         while not (self.playRequestEvent.isSet() or self.tearEvent.isSet()):
126             try:
127                 data, addr = self.rtp_socket_word.recvfrom(4096)
128                 if data:
129                     rtpPacket = RtpPacket()
130                     rtpPacket.decode(data)
131                     currFrameNbr = rtpPacket.getSeqNum()
132                     ## late packet
133                     if currFrameNbr < self.buffNum * BUFFER_SIZE + len(self.buffer_word):
134                         continue
135                     ## correct packet
136                     else:
137                         self.buffer_word.append(rtpPacket.getPayload())
138                         if len(self.buffer_word) > BUFFER_SIZE//10:
139                             self.reset()
140             except: continue
141 
142     def listenRtp_audio(self):
143         while not (self.playRequestEvent.isSet() or self.tearEvent.isSet()):
144             try:
145                 data, addr = self.rtp_socket_audio.recvfrom(80000)
146                 if data:
147                     rtpPacket = RtpPacket()
148                     rtpPacket.decode(data)
149                     currFrameNbr = rtpPacket.getSeqNum()
150                     ## late packet
151                     if currFrameNbr < self.buffNum * BUFFER_SIZE + len(self.buffer_audio):
152                         continue
153                     ## correct packet
154                     else:
155                         self.buffer_audio.append(rtpPacket.getPayload())
156                         if len(self.buffer_audio) > BUFFER_SIZE//10:
157                             self.reset()
158             except: continue # handle out of time
159 
```

如上圖，listenRtp_word()以及listenRtp_audio()的功能與前述listenRtp_video()非常類似，惟注意到138和156行，可以發現BUFFER_SIZE//10與視訊的reset條件不同。原因是server傳過來的視訊和音訊的每一個chunk的時間比為1:10。

```
160     def reset(self):
161         ## clean both buffer and update buffer number (Nth)
162         self.buffer_video = []
163         self.buffer_audio = []
164         self.buffer_word = []
165         self.playIndex_video = 0
166         self.playIndex_audio = 0
167         self.playIndex_word = 0
168         self.bar.set(self.playIndex_video)
169         self.buffNum += 1
170 
```

如上圖，reset()將buffer清空並使 index歸零。self.bar為使用者介面之進度條。

```
171     def setupRequest(self):
172         if self.state == 'INIT':
173             threading.Thread(target=self.recvRtspReply, daemon=True).start()
174             self.rtsp_socket.send(self.SETUP_STR.encode())
175             self.requestSent = 'SETUP'
176 
177     def playRequest(self):
178         if self.state == 'READY':
179             self.playRequestEvent.clear()
180             threading.Thread(target=self.listenRtp_video, daemon=True).start()
181             threading.Thread(target=self.listenRtp_audio, daemon=True).start()
182             threading.Thread(target=self.listenRtp_word, daemon=True).start()
183             self.rtsp_socket.send(self.PLAY_STR.encode())
184             self.requestSent = 'PLAY'
185 
186     def pauseRequest(self):
187         if self.state == 'PLAYING':
188             self.playRequestEvent.set()
189             self.rtsp_socket.send(self.PAUSE_STR.encode())
190             self.requestSent = 'PAUSE'
191 
192     def tearRequest(self):
193         self.tearEvent.set()
194         self.rtsp_socket.send(self.TEARDOWN_STR.encode())
195         self.requestSent = 'TEARDOWN'
196         self.master.destroy() # Close the gui window
197         self.master.quit()
198 
```

上圖的四個函式分別處理RTSP中的setup、play、pause、和teardown請求。執行pauseRequest()和tearRequest()時，會分別將self.playRequestEvent和self.tearEvent設成True，讓前述listenRtp_()或recvRtspReply()迴圈停止。注意到這邊無法使影片播放，而僅會進行資料處理並存進buffer。

```
199     def playMovie(self):
200         self.runEvent.clear()
201         self.playIndex_video = ((len(self.buffer_video) - 1) // 10) * 10
202         self.playIndex_audio = self.playIndex_video + 10
203         self.playIndex_word = self.playIndex_audio
204         threading.Thread(target=self.run_video, daemon=True).start()
205         threading.Thread(target=self.run_audio, daemon=True).start()
206         threading.Thread(target=self.run_word, daemon=True).start()
207 
208     def stopMovie(self):
209         self.runEvent.set()
210 
211     def backwardMovie(self):
212         self.playIndex_video = ((self.playIndex_video - 30*T_SEC)//10) * 10
213         self.playIndex_audio = ((self.playIndex_video - 30*T_SEC)//10) * 10 + 10
214         self.playIndex_word = ((self.playIndex_video - 30*T_SEC)//10) * 10 + 10
215         if self.playIndex_video < 0 : self.playIndex_video = 0
216         if self.playIndex_audio < 0 : self.playIndex_audio = 0
217         if self.playIndex_word < 0 : self.playIndex_word = 0
218         self.bar.set(self.playIndex_video)
219         print("replay, frame # = ", self.playIndex_video)
220 
221     def forwardMovie(self):
222         self.playIndex_video = ((self.playIndex_video + 30*T_SEC)//10) * 10
223         self.playIndex_audio = ((self.playIndex_video + 30*T_SEC)//10) * 10 + 10
224         self.playIndex_word = ((self.playIndex_video + 30*T_SEC)//10) * 10 + 10
225         if self.playIndex_video > len(self.buffer_video) : self.playIndex_video = len(self.buffer_video)
226         if self.playIndex_audio > len(self.buffer_audio) * 10 : self.playIndex_audio = len(self.buffer_audio) * 10
227         if self.playIndex_word > len(self.buffer_word) * 10 : self.playIndex_word = len(self.buffer_word) * 10
228         self.bar.set(self.playIndex_video)
229         print("replay, frame # = ", self.playIndex_video)
230 
231     def play_pause_Movie(self):
232         if self.playMovieButton == True: # pause
233             self.playMovieButton = False
234             self.stopMovie()
235         else: # play
236             self.playMovieButton = True
237             self.playMovie()
```

如上圖，這邊實際進入控制影片播放的函式，可以注意到第200和209行的self.runEvent，其可以用來控制是否要進入播放影片的迴圈。注意到30*T_SEC的30為影片每秒貞數。

```
238     def run_word(self):
239         while  not (self.runEvent.isSet() or self.tearEvent.isSet()):
240             if self.playIndex_word > len(self.buffer_word)*10:
241                 self.playIndex_word = len(self.buffer_word)*10
242             if self.playIndex_word < len(self.buffer_word)*10 and len(self.buffer_word) != 0:
243                 try:
244                     if self.SHOW_TRANSCRIPT:
245                         self.word.set(self.buffer_word[self.playIndex_word//10].decode())
246                     self.playIndex_audio += 10 # NUM_FRAME_PER_CHUNK
247                 except: pass
248                 time.sleep(1/30) # wait for listenRTP_audio thread to finish pushing
249                 self.playIndex_word = (int(self.bar.get())//10)*10 + 10 # update index by scrollbar
250 
251     def run_audio(self):
252         while not (self.runEvent.isSet() or self.tearEvent.isSet()):
253             if self.playIndex_audio > len(self.buffer_audio)*10: self.playIndex_audio = len(self.buffer_audio)*10
254             if self.playIndex_audio < len(self.buffer_audio)*10 and len(self.buffer_audio) != 0:
255                 time.sleep(1/30) # wait for listenRTP_audio thread to finish pushing
256                 try:
257                     self.stream.write(self.buffer_audio[self.playIndex_audio//10])
258                 except: pass
259                 self.playIndex_audio += 10 # NUM_FRAME_PER_CHUNK
260                 self.playIndex_audio = (int(self.bar.get())//10)*10 + 10 # update index by scrollbar
261 
262     def run_video(self):
263         while not (self.runEvent.isSet() or self.tearEvent.isSet()):
264             self.playIndex_video = int(self.bar.get()) # update index by scrollbar
265             if self.playIndex_video > len(self.buffer_video):
266                 self.playIndex_video = len(self.buffer_video)
267                 self.bar.set(self.playIndex_video) # update scrollbar
268             if self.playIndex_video < len(self.buffer_video) and len(self.buffer_video) != 0:
269                 inp = np.asarray(bytearray(self.buffer_video[self.playIndex_video]), dtype=np.uint8).reshape(120, 160, 3)
270                 inp = inp[:, :, [2, 1, 0]]
271                 pilImage = Image.fromarray(inp)
272                 w = self.master.winfo_height() # current window height
273                 h = self.master.winfo_width() # current window width
274                 pilImage = pilImage.resize((h, w)) # rescale # , Image.ANTIALIAS
275                 imgtk = ImageTk.PhotoImage(pilImage)
276                 self.label.configure(image = imgtk, height=h, width=w)
277                 self.label.image = imgtk
278                 self.playIndex_video += 2
279                 self.bar.set(self.playIndex_video) # update scrollbar
280                 time.sleep(1/30) # 30 frames per second
281 
```

執行playMovie()之後，會開三個thread分別執行上圖run_word()、run_audio()、以及run_video()。只要不要點擊暫停播放或逕將socket teardown，三者便會不斷的執行。可以注意到run_video()第272到274行，目的為調整播放影片的邊長以隨著視窗大小變化。再注意到第278行的playIndex，其每次加二而非一。原因是經過我們的實測結果，run_video()跑一次迴圈需時0.06秒而run_audio()跑一次迴圈需時0.3秒，視訊速度為音訊的20倍，但前述已經處理過10倍的問題，所以影格播放的速度僅需變兩倍。

```python
    def tkwindow(self):
        """Build GUI."""
        bottom_frame = Frame(self.master)
        bottom_frame.pack(side=BOTTOM)
        top_frame = Frame(self.master)
        top_frame.pack(side=TOP)
        # Create Setup button
        self.setup = Button(bottom_frame, width=3, height=1, bg='white')
        self.setup["text"] = "⚙"
        self.setup["command"] = self.setupRequest
        self.setup.pack(side=LEFT)
        # Create Play button
        self.start = Button(bottom_frame, width=3, height=1, bg='white')
        self.start["text"] = "⏺"
        self.start["command"] = self.playRequest
        self.start.pack(side=LEFT)
        # Create backward button
        self.backward = Button(bottom_frame, width=3, height=1, bg='white')
        self.backward["text"] = "{}⏪".format(T_SEC)
        self.backward["command"] =  self.backwardMovie
        self.backward.pack(side=LEFT)
        # Create playMovie button
        self.play = Button(bottom_frame, width=3, height=1, bg='white')
        self.play["text"] = "⏯"
        self.play["command"] = self.play_pause_Movie
        self.play.pack(side=LEFT)
        # Create forward button
        self.forward = Button(bottom_frame, width=3, height=1, bg='white')
        self.forward["text"] = "⏩{}".format(T_SEC)
        self.forward["command"] =  self.forwardMovie
        self.forward.pack(side=LEFT)
        # Create Pause button
        self.pause = Button(bottom_frame, width=3, height=1, bg='white')
        self.pause["text"] = "■"
        self.pause["command"] = self.pauseRequest
        self.pause.pack(side=LEFT)
        # Create Teardown button
        self.teardown = Button(bottom_frame, width=3, height=1, bg='white')
        self.teardown["text"] = "🗑"
        self.teardown["command"] = self.tearRequest
        self.teardown.pack(side=LEFT)
        # Create Transcript button
        self.teardown = Button(bottom_frame, width=3, height=1, bg='white')
        self.teardown["text"] = "💬"
        self.teardown["command"] = self.setTranscript
        self.teardown.pack(side=LEFT)
        # Create a bar
        self.bar = ttk.Scale(bottom_frame, from_=0, to=BUFFER_SIZE, length=300)
        self.bar.pack(side=LEFT)
        # Creat a text box for transcript
        self.word = StringVar()
        self.word.set("")
        self.text = Label(top_frame, textvariable=self.word, bg='black', fg='white', width=20, height=3, font=('標楷體', 16))
        self.text.pack(side=LEFT)
        # Create a label to display the movie
        self.label = Label(bg='black')
        self.label.pack(side=TOP)
```

如上圖，利用tkinter模組，我們寫了一個使用者介面。

```python
    def handler(self): # handle closing GUI window directly without teardown button
        """Handler on explicitly closing the GUI window."""
        self.pauseRequest()
        if tkMessageBox.askokcancel("Quit?", "Are you sure you want to quit?"):
            self.tearRequest()
        else: # When the user presses cancel, resume playing.
            self.playRequest()
```

最後，handler會處理使用者按下視窗右上角叉叉按鈕時的情況。

問題與解決方式

1. 聲音影像不同步的問題

由於是livestream, 我們的傳送的聲音是從麥克風錄製得來的, 如果一次錄製的時常設計過短, 聲音的品質會非常差, 太長則無法用socket傳遞, 且遇到有loss與delay的環境, 就會一次損失一大段聲音。經過權衡, 我們影片以30 fps傳遞, 而聲音與影像傳遞頻率比率為1:10, 以此設計相對應一個聲音chunk的長度。

然而, 我們發現由於套件本身的delay, 在server端當我們用兩個thread各自傳遞聲音與影像時, 兩者的對其狀況並不如我們所想。在client端也是, 由於影像要多做些轉換和tkinter的delay, 影像遠比聲音慢得多。

為了解決這個問題, 我們詳細研究每個套件的運作方式和delay。例如, 我們發現pyaudio的stream.read(), 會在背景持續填充buffer, 因此這個函式本身的delay並不是我們預期的1/3秒, 而是將盡0s。因此, 在server端, 我們可以用單一線程去輪流處理聲音與影像的傳送, 再配合加入相應的delay, 如此便能保證他們的比例為1:10, 且影響速率為30 fps。

在client端, 由於用單一線程依序處理聲音與影像的話, 無法避免會聽到麥克風開開關關的聲音, 因此我們決定以兩個線程各自去播影片與聲音。經過delay觀察, 我們發現讓影像隔著播, 就可以大約讓影像對齊。剩下偏差的部分, 我們則在每次client端buffer滿了要清空時, 強制讓聲音影像對齊。

總而言之, 我們依靠設計multithread和調整delay, 成功讓聲音與影像在server與client端對齊。在使用local host, 不考慮網路延遲的情況下, 影像與聲音的delay在1秒之內，且聲音的撥放很穩定。

加分項目

1. Livestream

Server傳送從電腦鏡頭拍攝的影像與從麥克風錄製的聲音到client。更進一步, 只要兩端client互相連結到對方的server, 就可以達到視訊通話。用localhost連結時delay在1秒之內，用網路線連結時則約2.5秒。

2. 使用者介面

使用tkinter刻劃精美的使用者介面。使用者可以用按鈕發送RTSP Request、影片撥放/暫停/快進/倒退、開啟或關閉字幕，也可使用進度條控制撥放時間點。

3. 控制影片撥放的功能

包括撥放/暫停/快進/倒退, 使用進度條選擇撥放的時間點

4. 支援多個client連結到同個server

5. 字幕功能

使用speech_recoginition套件, 在server端製作字幕。用額外一個UDP socket傳送。Client可以選擇開啟或關掉字幕。

未來展望

因為音訊不像視訊可以完整的切分成一貞或一個影格, 故聽起來沒有到非常順暢, 未來我們需要進一步改進我們的音訊傳輸品質。另外, 我們或許可以壓縮資料的傳輸量, 並在client端利用機器學習的方式將畫質提高, 藉此改善平均2.5秒的通訊延遲。最後, 我們的測試還可以更充足, 希望可以進一步提高系統的穩定性並將所有的程式打包成執行檔, 以利一般使用者使用。