

Final report

B08901046 詹侑昕

B08901058 陳宏恩

B08901073 王維芯

GitHub Link: https://github.com/redxouls/Magic_Streaming

RTP

介紹

RTP (Real-time Transport Protocol)是隸屬於transport layer的網路傳輸協定，普遍是基於UDP協議，主要用來進行多媒體資料的即時傳輸服務。由於RTP強調的是即時性，因此他不保證資料的可靠性也不具備服務品質QoS，較容易出現packet loss。RTP可以針對不同的傳輸地址 (IP address) 建立不同的通道，同時將資料傳給多個目標。常見的使用情境有視訊會議和即時串流媒體等。

用法

`RTP.py`

```
header = packet[: cls.HEADERSIZE]

padding = (header[0] >> 5) & 0b1
ext = bool((header[0] >> 4) & 0b1)
CSRC_count = header[0] & 0xF
Marker = (header[1] >> 7) & 0b1
payload_type = header[1] & 0x7F
seq_num = (header[2] << 8) | header[3]
timestamp = (header[4] << 24) | (header[5] << 16) | (header[6] << 8) | header[7]
SSRC = (header[8] << 24) | (header[9] << 16) | (header[10] << 8) | header[11]

# if not ext:
payload = packet[cls.HEADERSIZE :]
```

對輸入資料進行位元的對應，再將其打包成RTP packet的格式。

RTP封包

Bit Offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	Padding	Ext.	CSRC Count	Marker	Payload Type	Sequence Number
32	Timestamp						
64	Synchronization Source (SSRC) Identifier						
96	Contributing Source (CSRC) Identifier						
96+32*CC	Payload						

source:

https://www.researchgate.net/figure/presents-the-RTP-packet-header-format-In-RTP-the-Synchronization-Source-SSRC-field_fg2_221352750 [accessed 21 Jan, 2022]

RTSP

介紹

RTSP屬於TCP/IP體系的應用層協議，主要是用於串流媒體的建立與控制，例如影片的播放與暫停。RTSP是雙向的串流協議

, client/server端皆可以發送請求，並且同時支援多個串流的控制需求。在一個傳輸流程當中，RTSP會經歷4個重要的階段，分別是SETUP, PLAY, PAUSE, TEARDOWN，分別負責啟動RTSP連線、控制資料的播放與暫停、以及終止連線狀態。

用法

`RTSP.py`

分別處理請求與回應，並將其包裝成RTSP格式。

Request -> Response

接收到client端的請求以後，對內容進行解析。若語句不符合格式或有所缺漏則回報錯誤，若檢查皆正確則將特定的資訊按照格式取出，再包裝成RTSP物件回傳給server端。[圖]

Response -> Request

檢查server端回傳的資訊是否符合格式，若無誤則包裝成RTSP物件回傳給client。

```
Sending request: 'TEARDOWN rtsp://mic RTSP/1.0\r\nSeq: 8\r\nSession: 0\r\n'
Received from server: 'RTSP/1.0\r\nSeq: 8\r\nSession: 0\r\n'
```

Video streaming

Used Packet: OpenCV, pillow, numpy, io

Server side:

setup:

1. 建立影像傳送 rtp 專用的 udp socket，並記錄 client side 的 ip 和 port。
2. 初始化裝置，使用 cv2 選取想要的相機裝制。

play:

1. 使用 opencv 獲取鏡頭的 raw_img
2. 使用 resize 將圖片的解析度變小並水平翻轉，再將圖片轉為 JPEG 的 raw bytes 壓縮度為 50。
3. 使用 BytesIO 的 Object 接受 cv2 的 buffer，將指針設為最起點並將該 BytesIO object 中的 buffer 取出
4. 取出的 buffer 便是串流影片中的一個 frame，使用 RTP packet 將該 frame 包住，並以 UDP 傳送至 rtsp 設定的 client 端 ip 和 port
5. 將影像串流的函數放到 thread 中，背景執行。

pause:

1. 停止背景執行中的影像串流 thread 。

tearDown:

1. 切斷 rtp socket 的 connection
2. 將相機裝置妥善關閉，避免佔據到其他的程序的使用。

Client side:

setup:

1. 建立接收 rtp 影像專用的 udp socket。

play:

2. 從 udp socket listen 來自 server side 的 rtp packet.
3. 根據 rtp 中 header 所標註的 application mode 來偵測 EOF
4. 收到一個完整的封包後，將 payload 和 timestamp 取出
5. 將接收影像串流的函數放到 thread 中，背景執行。

6. thread 負責收 video 的 RTP packet, 先將video存進一buffer中, 與audio端進行對時後, 選則對應的frame並且透過CV2進行影像處理並輸出。

pause:

1. 送出RTSP pause request 給 server 端。
2. 停止背景執行接收影像串流的 thread 。

tearDown:

1. 切斷 rtp socket 的 connection
2. 將相機裝置妥善關閉, 避免佔據到其他的程序的使用。

Audio streaming

Used Packet: pyaudio, io

Server side:

接收到 RTSP setup request => setup :

1. 建立聲音傳送 rtp 專用的 udp socket。並記錄 client side 的 ip 和 port
2. 使用 pyaudio 初始化音源裝置, 選取聲音的輸入裝置, 並建立聲音的 stream。
3. 設定 pyaudio 的參數:
FORMAT=pyaudio.paInt16, CHANNELS = 1, RATE = 44100 , CHUNK = 1024
4. 回傳 RTSP response

接收到 RTSP play request => play:

- 1.
2. 設定完以後就可以從 sound stream 中擷取出 binay 的聲音資訊。
3. 將取出的 buffer 使用 RTP packet 將該 frame 包住, 並以 UDP 傳送至 rtsp 設定的 client 端 ip 和 port
4. 將聲音串流的函數放到 thread 中, 背景執行。
5. 回傳 RTSP response

接收到 RTSP pause request => pause:

1. 停止背景執行的thread , 來停止聲音串流函數。
2. 回傳 RTSP response

接收到 RTSP teardown => teardown:

1. 切斷 rtp socket 的 connection
2. 將錄音裝置妥善關閉, 避免佔據到其他的程序的使用。
3. 回傳 RTSP response

Client side:

setup

1. 建立聲音接收 rtp 專用的 udp socket。
2. 使用 pyaudio, 選取聲音的輸出裝置。
3. 初始話
4. 與 server 端相同的 pyaudio 參數
5. 設定完以後就可以 sound stream 中擷取出 binay 的聲音資訊。
6. 將取出的 buffer 使用 RTP packet 將該 frame 包住, 並以 UDP 傳送至 client 端。

play:

1. 送出RTSP play request 給 server 端。
2. 建立 stream_audio 這個 thread。
3. stream_audio 負責收 audio 的 RTP packet, 並透過pyaudio進行輸出。

pause:

1. 送出RTSP pause request 給 server 端。
2. 將 play 階段的 audio thread中止。

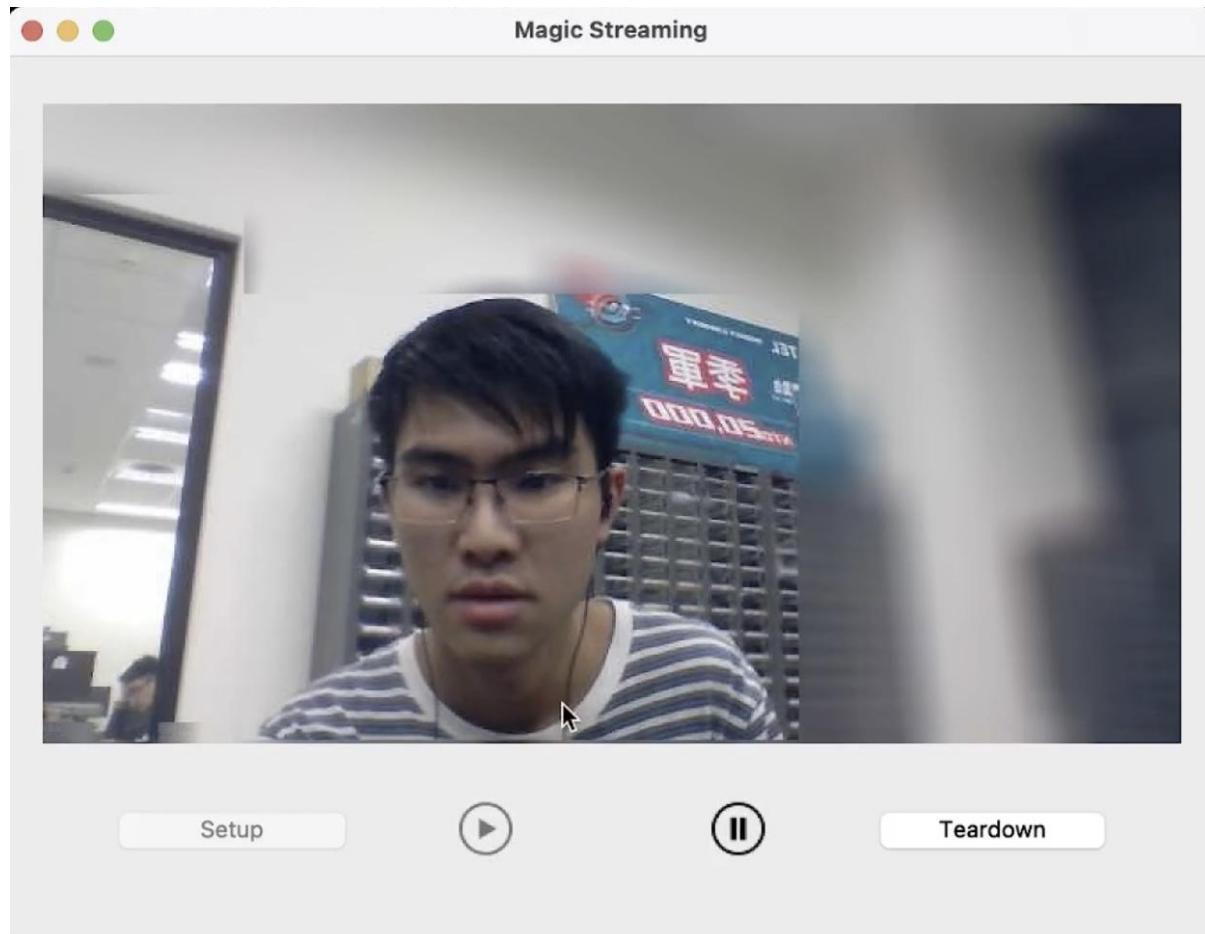
teardown:

1. 送出RTSP teardown request 給 server 端。
2. 將 class 中每個參數進行初始化。

GUI

串流播放介面

使用PyQt5製作串流播放介面，畫面正中央為影像窗口，下方由左到右分別是RTSP的4個狀態，SETUP、PLAY、PAUSE和TEARDOWN。



Qt是跨平台的GUI框架，廣泛用於編寫各種前端介面。我們用的PyQt5是結合Python和Qt的套件庫，適合用來製作簡易的串流播放介面。

播放影像的方式是將圖片一幀一幀傳到前端。在Qt框架當中，物件溝通的方式是透過signals和slots, signal用來產生訊號而slot用來接收訊號。我們將signal定義為切換的照片，再將其connect到前端的reload_image，並且在啟動timer後連接切換的照片，即signal.emit，藉此達成播放串流影像的效果。

Bonus: Object Detection

Object detection已經被廣泛的使用在各個層面上，在本次專題中，我們之所以需要用到object detection，是用以達成人物聚焦的效果，如在google meet中，近年來也加入了背景模糊的功能，除了可以讓大家目光更加的聚焦在另一端的使用者身上，也可以讓使用者的隱私更加受到保護。

在這次專題的實做上，我們使用了yolov5s這個object detection的model進行辨識，之所以會選擇使用這個model，是因為其在準確度及效能上達到了非常好的平衡，眾所皆知，一個機器學習的模型是需要大量的運算資源進行training以及inference的，在一般的電腦上，是非常難以run起一個過於龐大的模型的，這邊使用的yolov5s僅具有12.7M個參數，就算是在一台只有CPU沒有GPU的電腦依然可以運行。我們這次object detection的流程為：

1. 由server端將影像傳入client端
2. client端收到影像後將其decode成numpy array
3. 將numpy array經過dataloader後進行resize以及轉型
4. 輸入model
5. 得到object detection的結果，結果裡有被辨識出物件的class、其bounding box的頂點
6. 將不在bounding boxes內的部份做模糊化處理

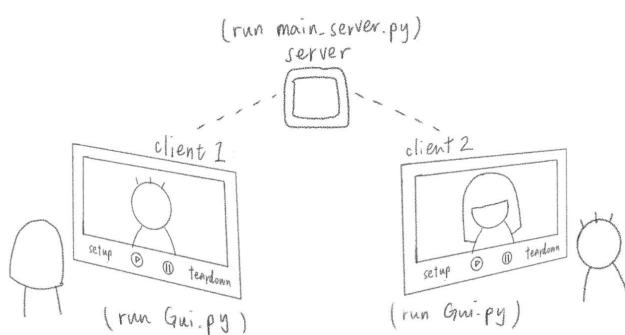
這樣一來可以得到不錯的結果，但其中有一個問題在於若每個frame都去進行object detection的話，frame rate會有顯著的下降，後來我們想到的作法為：每十個frame進行一次的inference，因為以視訊會議為場景的話，使用者並不會在短時間內有太大幅度的動作，因此每隔0.5~1秒更新一次即可。

Application: online meeting

疫情以後，無論是上課、討論、開會都常常以視訊會議的方式進行，而串流平台的主要應用之一就是視訊，因此想藉這個機會架設一個RTSP/RTP架構的程式，透過這次期末專題的機會對於視訊會議背後的架構有更深的理解。Online meeting在實做上並不僅僅只是傳送影像以及音訊，以一般的影音內容來說，要有良好的傳輸效果可以直接透過TCP進行傳送，但Online meeting必須要作到即時，所以透過RTSP/RTP架構會有較好的效果，此外，在影像及音訊的結合也是非常值得深入研究的問題，透過RTP header的timestamp可能可以做到較好的mapping，也因此Online meeting可以說是RTSP/RTP架構的最好的應用範例。

操作方法

1. 開啟Server
`'python3 main_server.py --ip <host IP> --port <host port>'`
2. Client端建立連線
`'python3 Gui.py --ip <IP> --port <port>'`



參考資料

1. <https://github.com/ultralytics/yolov5>
2. <https://github.com/gabrieljablonski/rtsp-rtp-stream>
3. https://en.wikipedia.org/wiki/Real-time_Transport_Protocol
4. https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol