

1091438 郁宸瑋

1090953 陳元彬

## Experiment 6 :

### (a)

Berkeley Socket I/F 支援主機傳輸資訊到其他主機。使用者與其中一個

socket 對接後，就可以和網路上其他通訊端點連線以傳輸資料。同時

Berkeley Socket I/F 也是 UDP 和 TCP/IP 使用的基層服務。

以下是以 python 的 socket 建構的 UDP 與 TCP 連接：

### UDP：

從 WireShark 的擷取結果可以得知 client 端傳遞了一個 UDP 協定到 server

端，且因為沒有相關的回饋，server 端沒有回傳任何封包回來，python 的接

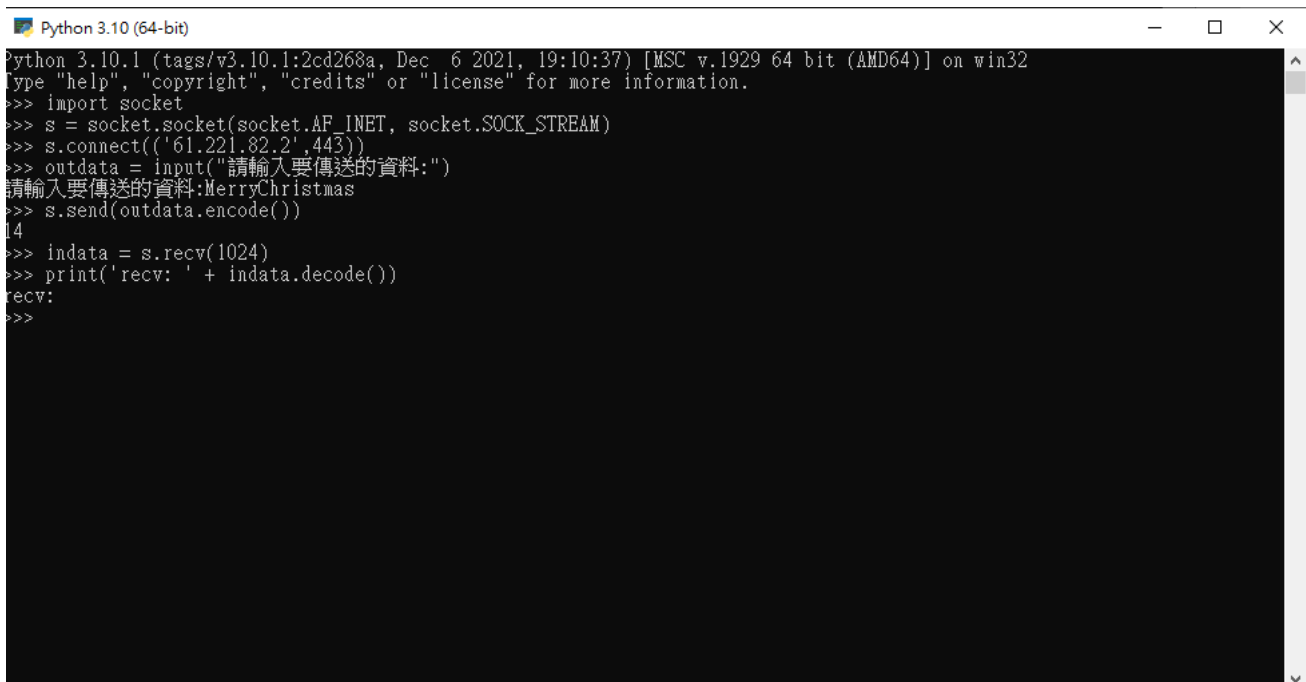
收函式也因此卡住，無法執行下一步驟

The screenshot displays two windows. The top window is Wireshark, showing a packet capture on the 'eth0' interface. The selected packet is a UDP packet from 192.168.1.104 to 192.0.2.1, port 56507. The packet details show the IP header, UDP header, and a data payload of 15 bytes. The bottom window is a Python 3.10 terminal. It shows the execution of a UDP socket program. The user enters the IP address '192.0.2.1' and the port '56507'. The program sends a message '請輸入要傳送的資料: 我好開心囉' to the server. However, the program then enters an infinite loop, repeatedly calling 'recv\_data = udp\_socket.recvfrom(1024)' without receiving any data, which is why it appears to be stuck.

This screenshot is similar to the one above, showing the same Wireshark packet capture and Python terminal. The Wireshark packet list shows the same UDP packet. The Python terminal shows the same code and user input. The key observation is that the server (Python) is stuck in a loop, waiting for a response from the client (Wireshark) that never arrives, illustrating a one-way communication scenario.

## TCP:

雖然傳輸的為無意義的封包, TCP 仍回傳了一個空的封包回來, 如下圖



```
Python 3.10 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> s.connect(('61.221.82.2', 443))
>>> outdata = input("請輸入要傳送的資料:")
請輸入要傳送的資料:MerryChristmas
>>> s.send(outdata.encode())
14
>>> indata = s.recv(1024)
>>> print('recv: ' + indata.decode())
recv:
>>>
```

## (b)

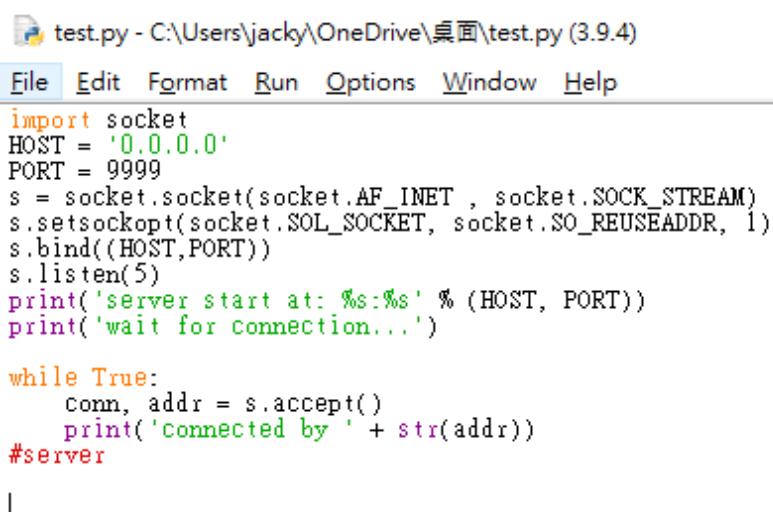
UDP 屬於傳輸層的協定, 從圖二(b)可看出 UDP 會在 server 端不知道 client 端要傳送訊息的情況下直接傳送封包至 Client 端。因為只進行一次連線, 和 TCP 相比速度較快但不精確。而 TCP 會先向 Client 端建立連線請求, 等待 server 端確認並回應後便開始傳輸資料。由於 TCP 會確認封包完整的送達 Client 端, 因此和 UDP 比較起來較為可靠。綜上所述, 如果要傳送大且不須準確的資料可以選擇 UDP; 而需要精確傳輸的資料則以較可靠的 TCP 為主。

## (c)

我們用 Python 在同一台主機上同時建立了 Server 和 Client,

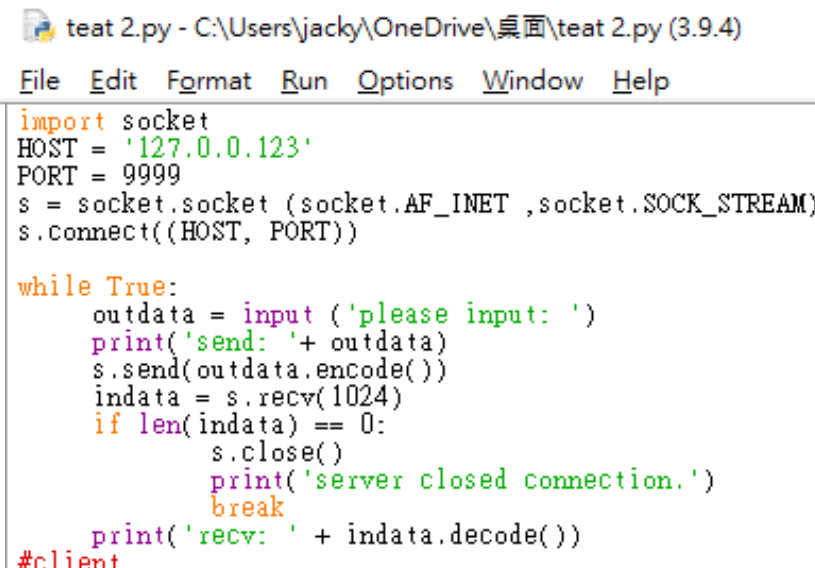
(下方為)Server 的程式碼

Client 的程式碼



```
test.py - C:\Users\jacky\OneDrive\桌面\test.py (3.9.4)
File Edit Format Run Options Window Help
import socket
HOST = '0.0.0.0'
PORT = 9999
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST, PORT))
s.listen(5)
print('server start at: %s:%s' % (HOST, PORT))
print('wait for connection...')

while True:
    conn, addr = s.accept()
    print('connected by ' + str(addr))
#server
```



```
teat 2.py - C:\Users\jacky\OneDrive\桌面\teat 2.py (3.9.4)
File Edit Format Run Options Window Help
import socket
HOST = '127.0.0.123'
PORT = 9999
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

while True:
    outdata = input('please input: ')
    print('send: ' + outdata)
    s.send(outdata.encode())
    indata = s.recv(1024)
    if len(indata) == 0:
        s.close()
        print('server closed connection.')
        break
    print('recv: ' + indata.decode())
#client
```

## (下方為)Server 的執行結果

```
*IDLE Shell 3.9.4*
File Edit Shell Debug Options Window Help
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021, 13:40:21) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\jacky\OneDrive\桌面\test.py =====
==
server start at: 0.0.0.0:9999
wait for connection...
connected by ('127.0.0.1', 63753)
```

## Client 的執行結果

```
*IDLE Shell 3.9.4*
File Edit Shell Debug Options Window Help
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021, 13:40:21) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\jacky\OneDrive\桌面\teat 2.py =====
==
please input: helle Mr.zhong
send: helle Mr.zhong
```

Wireshark(以 Adapter or lookback traffic capture 介面)捕捉到的封包, 如下圖:

\*Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.addr = 127.0.0.123

No.	Time	Source	Destination	Protocol	Length	Info
921	16.005105	127.0.0.1	127.0.0.123	TCP	52	63753 → 9999 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1
922	16.005143	127.0.0.123	127.0.0.1	TCP	52	9999 → 63753 [SYN, ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=65495 SACK_PERM=1
923	16.005172	127.0.0.1	127.0.0.123	TCP	44	63753 → 9999 [ACK] Seq=1 Ack=1 Win=65495 Len=0
1535	27.078140	127.0.0.1	127.0.0.123	TCP	58	63753 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=65495 Len=14 [TCP segment of ...
1536	27.078154	127.0.0.123	127.0.0.1	TCP	44	9999 → 63753 [ACK] Seq=1 Ack=15 Win=65481 Len=0

> Frame 921: 52 bytes on wire (416 bits). 52 bytes captured (416 bits) on interface \Device\NPF Loopback, id 0

```
0000 02 00 00 00 45 00 00 30 a2 f9 40 00 80 06 00 00 ...E..0 ..@....
0010 7f 00 00 01 7f 00 00 7b f9 09 27 0f 19 6e 82 70 .....{ ...'.n.p
0020 00 00 00 00 70 02 ff d7 ce af 00 00 02 04 ff d7 ....p....
0030 01 01 04 02
```

wireshark\_NPF\_LoopbackB3HVE1.pcapng | Packets: 1873 · Displayed: 5 (0.3%) | Profile: Default

在這次傳輸中總共傳遞了五次封包，其中只有第四次是有長度的，故

第一次：連線建立，client 端 to server 端

第二次：同意請求，server 端 to client 端

第四次：傳輸 data

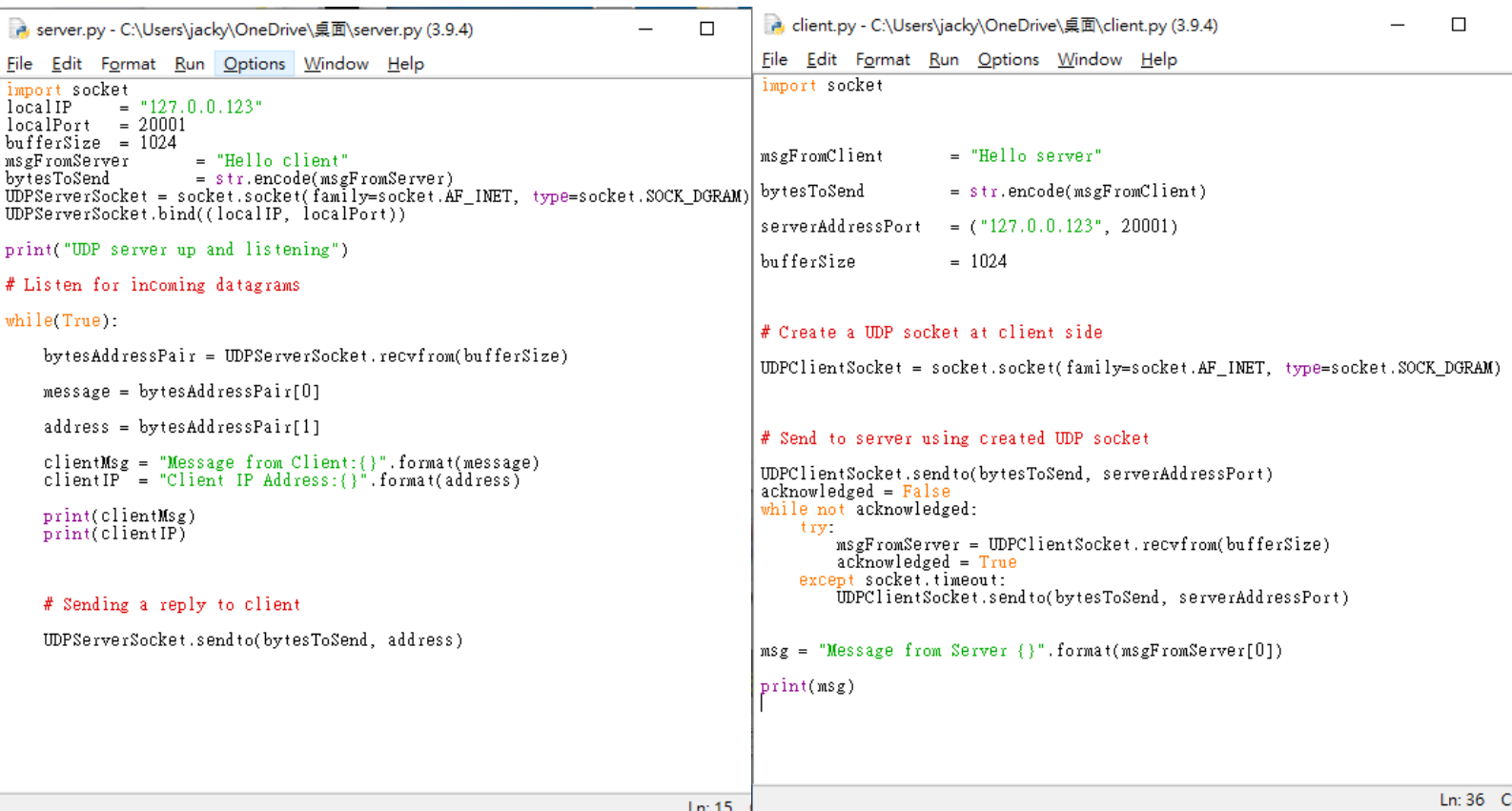
第五次：確認封包的完整性，server 端對 client 端

### Experiment 7:

下圖使用了 stop-and-wait ARQ 的 UDP Protocol 的 Server 和 Client 程式碼。

左邊是 Server 程式碼，右邊是 Client 程式碼。

可以看出在 Client 有進行 stop-and-wait ARQ。



```
server.py - C:\Users\jacky\OneDrive\桌面\server.py (3.9.4)
File Edit Format Run Options Window Help
import socket
localIP = "127.0.0.123"
localPort = 20001
bufferSize = 1024
msgFromServer = "Hello client"
bytesToSend = str.encode(msgFromServer)
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
UDPServerSocket.bind((localIP, localPort))

print("UDP server up and listening")

# Listen for incoming datagrams
while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]

    clientMsg = "Message from Client:{}".format(message)
    clientIP = "Client IP Address:{}".format(address)

    print(clientMsg)
    print(clientIP)

    # Sending a reply to client
    UDPServerSocket.sendto(bytesToSend, address)

Ln: 15 C

client.py - C:\Users\jacky\OneDrive\桌面\client.py (3.9.4)
File Edit Format Run Options Window Help
import socket

msgFromClient = "Hello server"
bytesToSend = str.encode(msgFromClient)
serverAddressPort = ("127.0.0.123", 20001)
bufferSize = 1024

# Create a UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Send to server using created UDP socket
UDPClientSocket.sendto(bytesToSend, serverAddressPort)
acknowledged = False
while not acknowledged:
    try:
        msgFromServer = UDPClientSocket.recvfrom(bufferSize)
        acknowledged = True
    except socket.timeout:
        UDPClientSocket.sendto(bytesToSend, serverAddressPort)

msg = "Message from Server {}".format(msgFromServer[0])
print(msg)

Ln: 36 C
```

## (下方為)Server 的執行結果



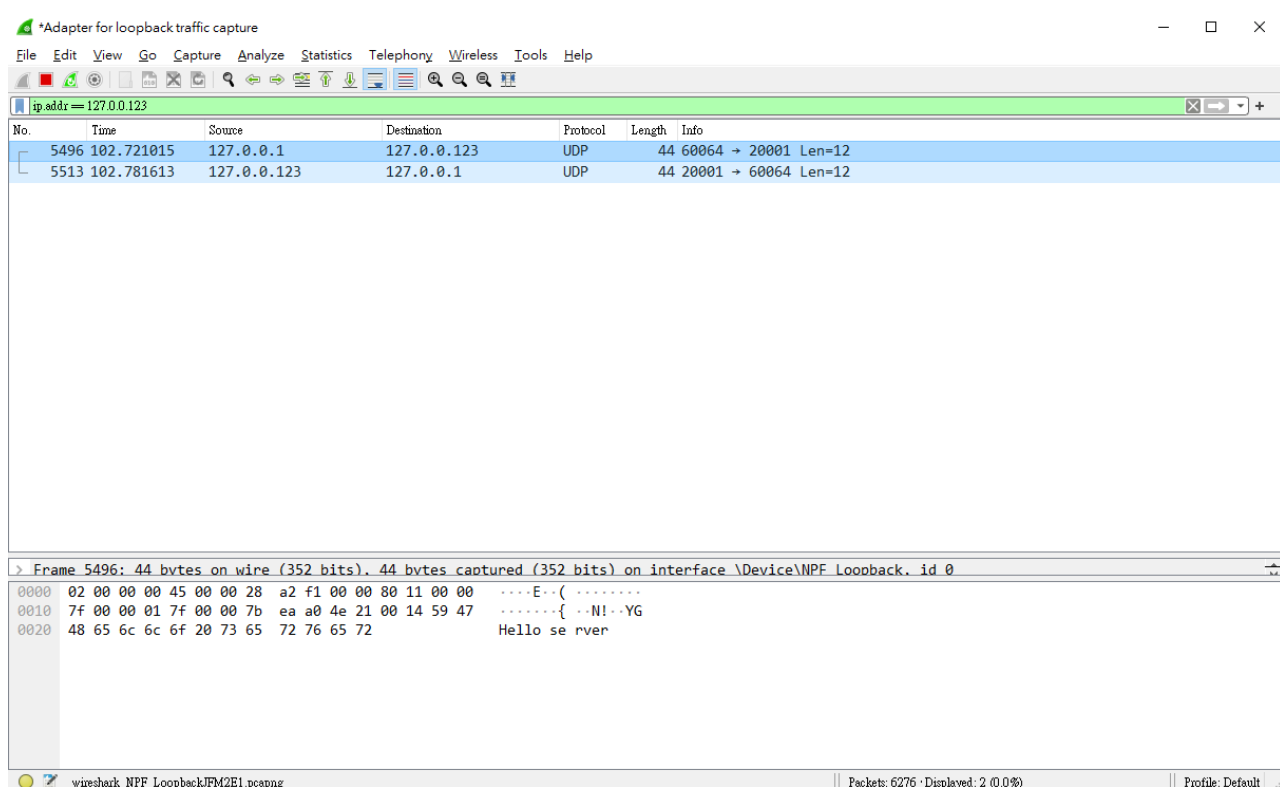
```
*IDLE Shell 3.9.4*
File Edit Shell Debug Options Window Help
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021, 13:40:21) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\jacky\OneDrive\桌面\server.py =====
==
UDP server up and listening
Message from Client:b'Hello server'
Client IP Address:('127.0.0.1', 59317)
```

## Client 的執行結果



```
IDLE Shell 3.9.4
File Edit Shell Debug Options Window Help
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021, 13:40:21) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\jacky\OneDrive\桌面\client.py =====
==
Message from Server b'Hello client'
>>>
```

Wireshark(以 Adapter or lookback traffic capture 介面)捕捉到的封包,如下圖:



由上圖可知, Server 有回應 Client 的程式碼, 與實驗六(a)不同, 除了傳遞到 Server 的封包之外, 另外還有回覆 Client 的封包, 符合圖二(b)