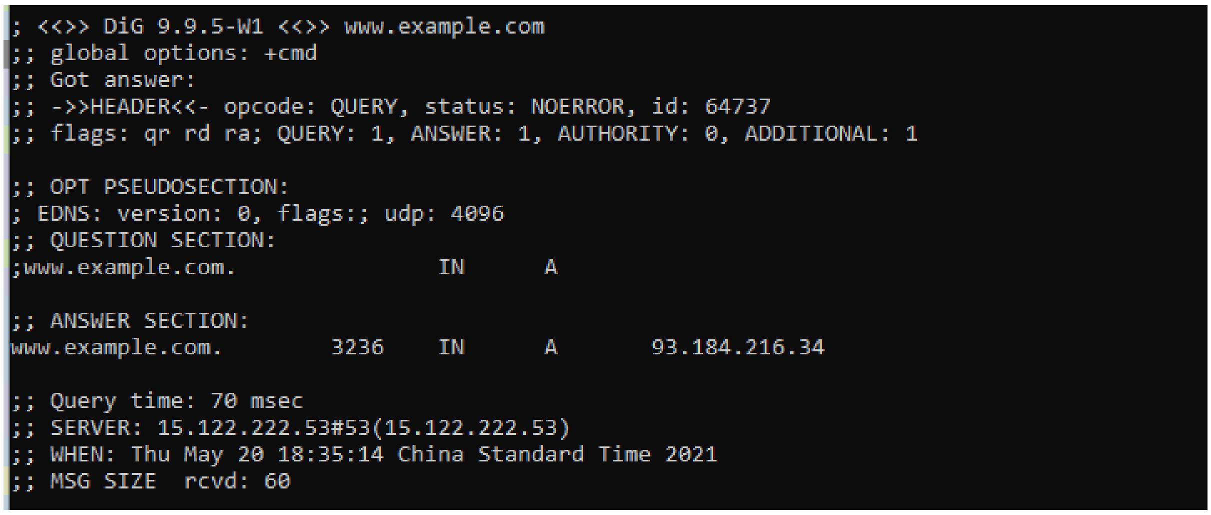
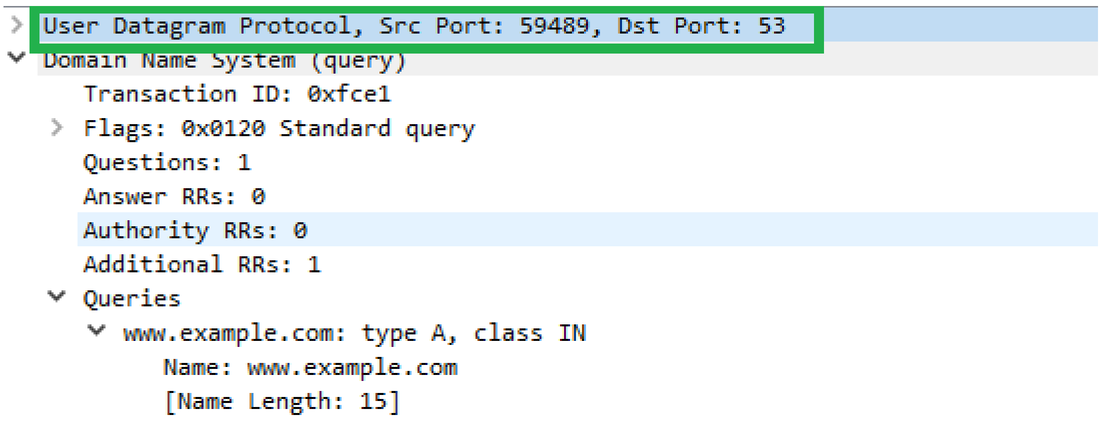
1. Command dig is used to find the ip address of a hostname such as www.yahoo.com. To do this, you can simply run dig www.google.com. You might see the following result. This is the result returned by your local DNS server when you run dig command to request it to find out the ip address of www.google.com.
www.google.com.	300	IN	A	172.217.13.4

a.



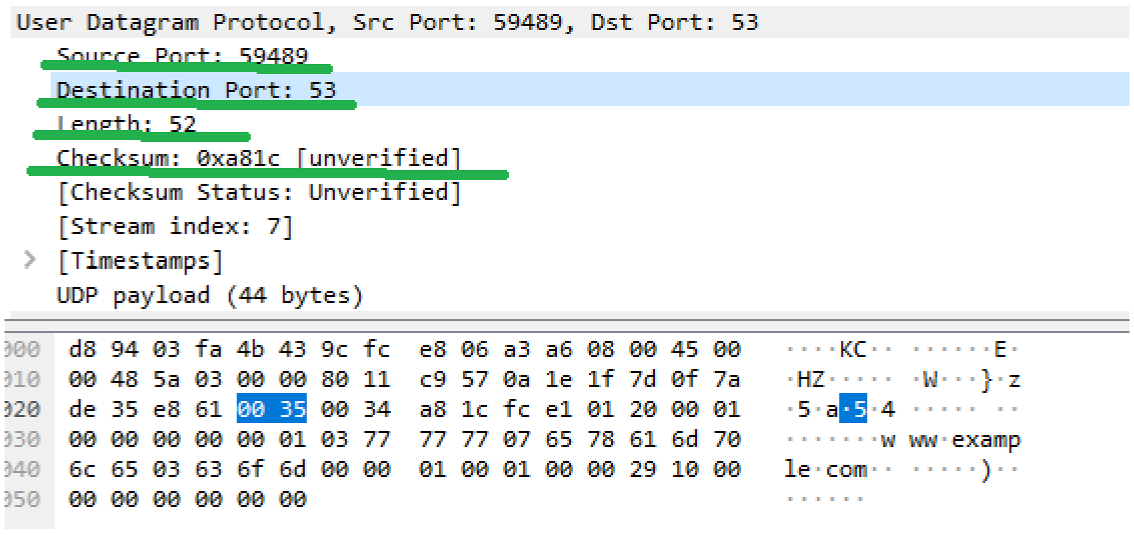
```
; <<>> DiG 9.9.5-W1 <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64737
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        3236    IN      A       93.184.216.34

;; Query time: 70 msec
;; SERVER: 15.122.222.53#53(15.122.222.53)
;; WHEN: Thu May 20 18:35:14 China Standard Time 2021
;; MSG SIZE  rcvd: 60
```

b.



```
> User Datagram Protocol, Src Port: 59489, Dst Port: 53
v Domain Name System (query)
    Transaction ID: 0xfce1
  > Flags: 0x0120 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
  v Queries
    v www.example.com: type A, class IN
        Name: www.example.com
        [Name Length: 15]
```

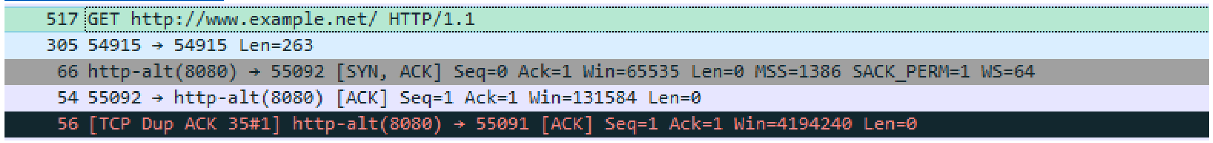
The UDP headers are as learned from the wireshark screenshot:
1) Source port
2) Destination port
3) Length of the package
4) Checksum



```
User Datagram Protocol, Src Port: 59489, Dst Port: 53
    Source Port: 59489
    Destination Port: 53
    Length: 52
    Checksum: 0xa81c [unverified]
    [Checksum Status: Unverified]
    [Stream index: 7]
  > [Timestamps]
    UDP payload (44 bytes)

00  d8 94 03 fa 4b 43 9c fc  e8 06 a3 a6 08 00 45 00   ····KC·· ······E·
10  00 48 5a 03 00 00 80 11  c9 57 0a 1e 1f 7d 0f 7a   ·HZ····· ·W···}·z
20  de 35 e8 61 00 35 00 34  a8 1c fc e1 01 20 00 01   ·5·a·5·4 ····· ··
30  00 00 00 00 00 01 03 77  77 77 07 65 78 61 6d 70   ·······w ww·examp
40  6c 65 03 63 6f 6d 00 00  01 00 01 00 00 29 10 00   le·com·· ·····)··
50  00 00 00 00 00 00                                  ······
```

c. my local DNS server's IP is 15.122.222.53, yes, I can confirm there is no data exchange between my local machine and DNS server

2. Yes I can confirm the three packets establishing the connection when browsing www.example.net



```
517 GET http://www.example.net/ HTTP/1.1
305 54915 → 54915 Len=263
 66 http-alt(8080) → 55092 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1386 SACK_PERM=1 WS=64
 54 55092 → http-alt(8080) [ACK] Seq=1 Ack=1 Win=131584 Len=0
 56 [TCP Dup ACK 35#1] http-alt(8080) → 55091 [ACK] Seq=1 Ack=1 Win=4194240 Len=0
```

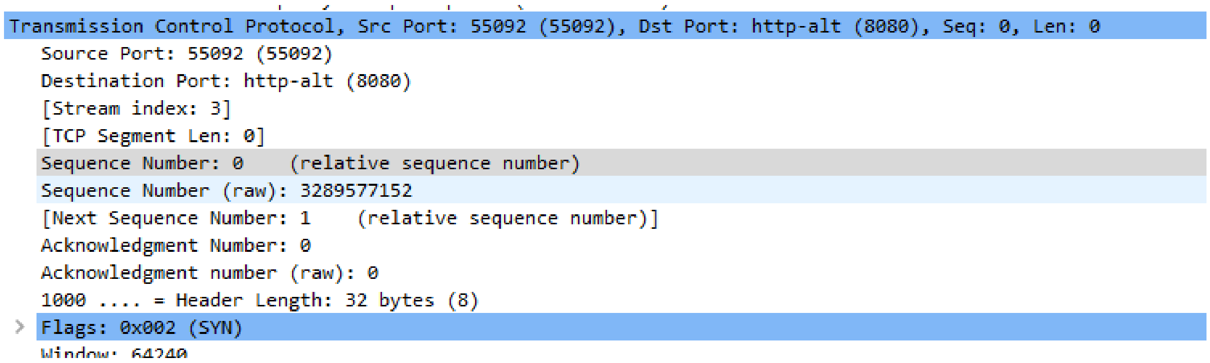
Since I was connecting via proxy so here you can see the leg between my server with the proxy

3. Describe what the programs are doing with inline comments:
   Client:

```
#!/usr/bin/python3
from socket import *
serverName = "10.199.62.80"
serverPort = 12000

# initialize a local socket which is similar to a local endpoint
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input("input your message:")

# through the local socket, send message to remote server on the given port
.
clientSocket.sendto(message.encode(), (serverName, serverPort))

# then receive response from server.
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

# print the response message
4. print(modifiedMessage.decode())
5.
6. # destruct the local socket in the end.
7. clientSocket.close()
```

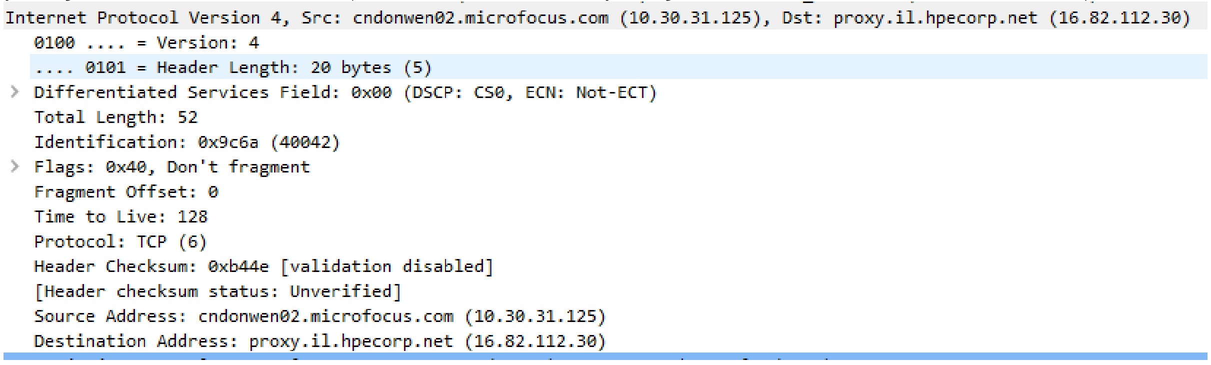
```
server:
#!/usr/bin/python3
from socket import *
serverPort = 12000

# initialize a server socket and bind it to a given port number, to listen/wel
come on the port.
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print('The server is ready to receive')

# loop on the port to receive incoming message.
while True:
    # receive message with maximum given size at a time
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()

    # answer the connection by sending message to the client.
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

4. a. source port is 55082, destination port is 8080 since I was connecting through proxy with port 8080. And I confirm they are in the TCP header as below:



```
Transmission Control Protocol, Src Port: 55092 (55092), Dst Port: http-alt (8080), Seq: 0, Len: 0
    Source Port: 55092 (55092)
    Destination Port: http-alt (8080)
    [Stream index: 3]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 3289577152
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
    Window: 64240
```

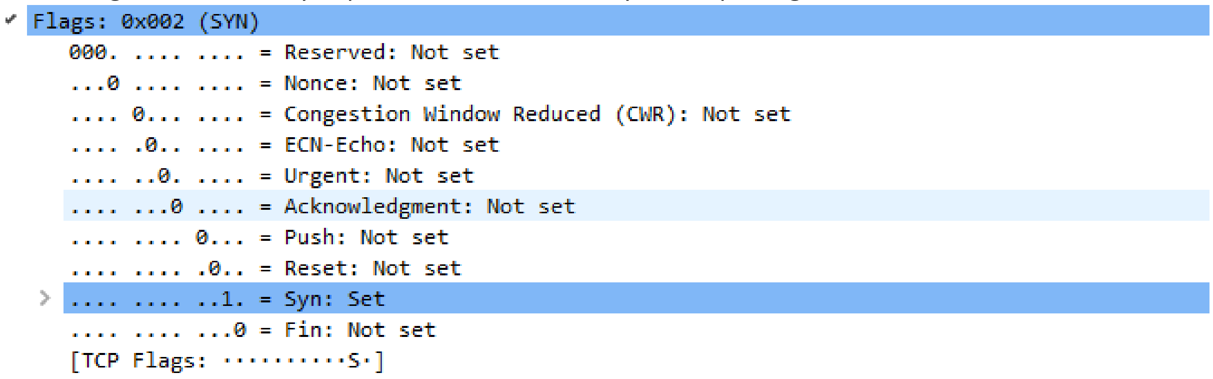
Confirm the source IP is my local IP 10.30.31.125, and destination port is my proxy IP 16.82.112.30 (have to use the proxy due to network limitation)



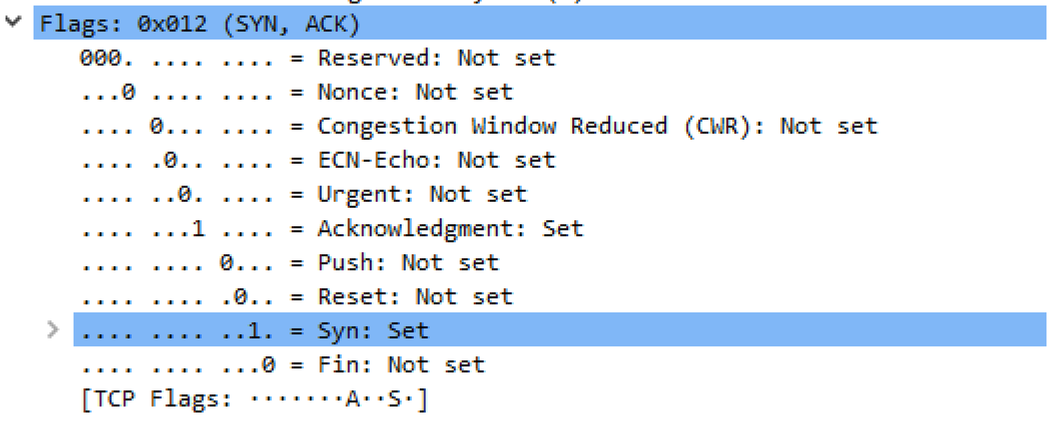
```
Internet Protocol Version 4, Src: cndonwen02.microfocus.com (10.30.31.125), Dst: proxy.il.hpecorp.net (16.82.112.30)
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x9c6a (40042)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0xb44e [validation disabled]
    [Header checksum status: Unverified]
    Source Address: cndonwen02.microfocus.com (10.30.31.125)
    Destination Address: proxy.il.hpecorp.net (16.82.112.30)
```

b. looked at the syn packet, the sequence number is 0, it is not a random number.

c. the flag bit is S in the sync packet, and A S in the sync-ack package as below:

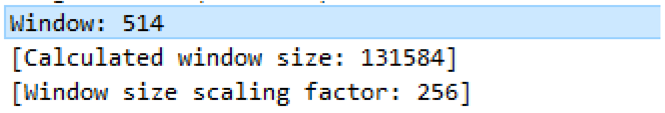


```
Flags: 0x002 (SYN)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Nonce: Not set
   .... 0... .... = Congestion Window Reduced (CWR): Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..0. .... = Urgent: Not set
   .... ...0 .... = Acknowledgment: Not set
   .... .... 0... = Push: Not set
   .... .... .0.. = Reset: Not set
 > .... .... ..1. = Syn: Set
   .... .... ...0 = Fin: Not set
   [TCP Flags: ··········S·]
```



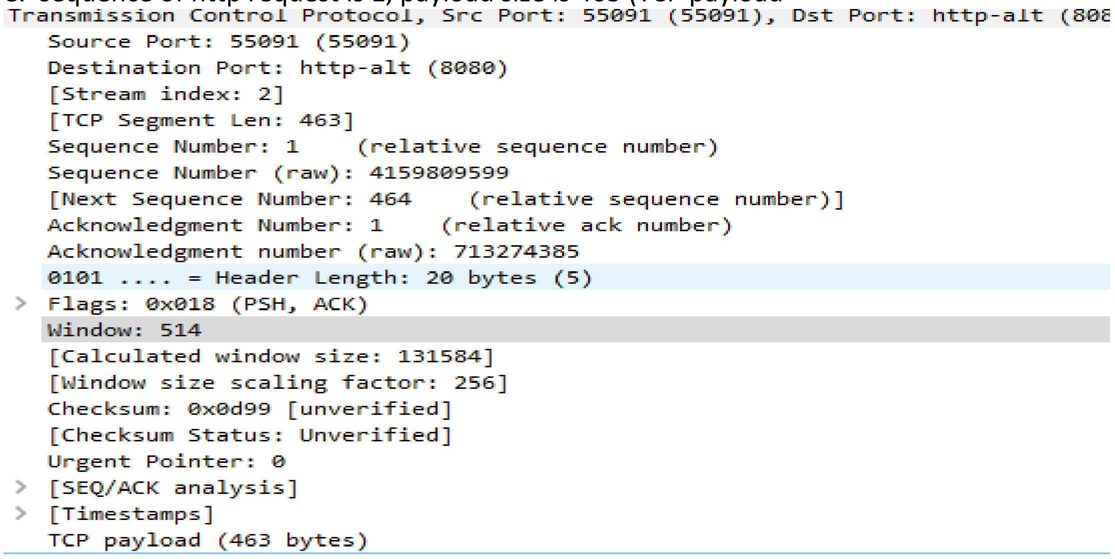
```
Flags: 0x012 (SYN, ACK)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Nonce: Not set
   .... 0... .... = Congestion Window Reduced (CWR): Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..0. .... = Urgent: Not set
   .... ...1 .... = Acknowledgment: Set
   .... .... 0... = Push: Not set
   .... .... .0.. = Reset: Not set
 > .... .... ..1. = Syn: Set
   .... .... ...0 = Fin: Not set
   [TCP Flags: ·······A··S·]
```

d. the window size is 65535 in the sync-ack package, not exactly equals with the http response size which is "window:514" and "Calculated window size : 131584"



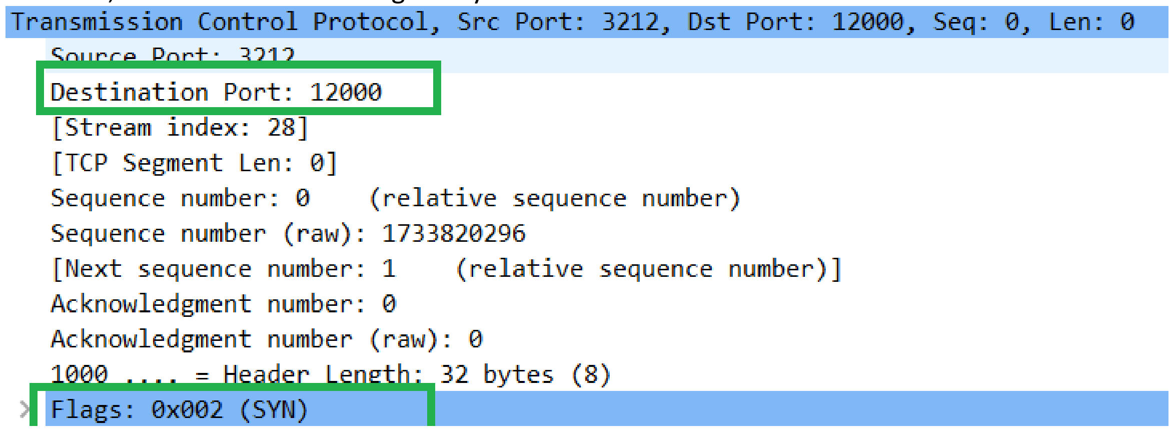
```
Window: 514
[Calculated window size: 131584]
[Window size scaling factor: 256]
```

e. sequence of http request is 1, payload size is 463 (TCP payload



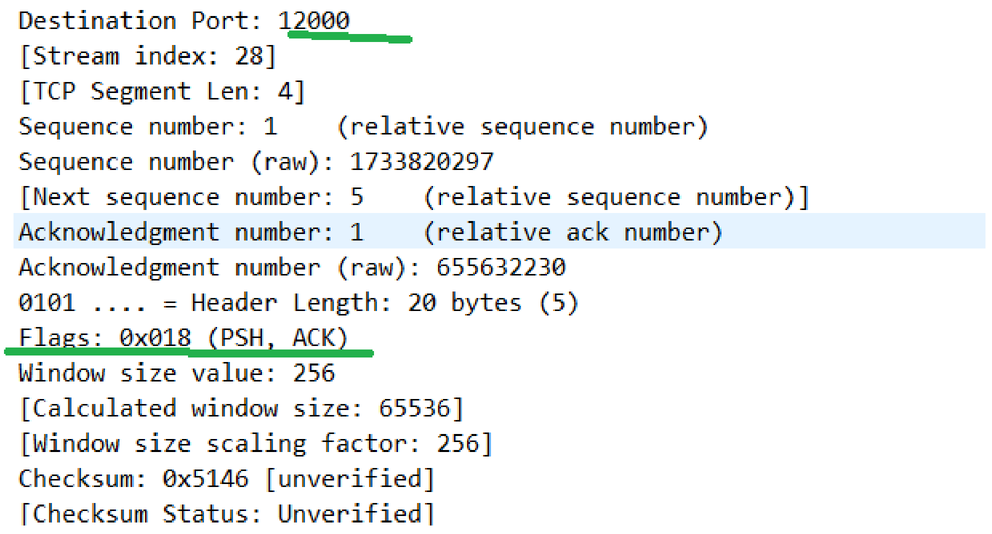
```
Transmission Control Protocol, Src Port: 55091 (55091), Dst Port: http-alt (808
   Source Port: 55091 (55091)
   Destination Port: http-alt (8080)
   [Stream index: 2]
   [TCP Segment Len: 463]
   Sequence Number: 1    (relative sequence number)
   Sequence Number (raw): 4159809599
   [Next Sequence Number: 464    (relative sequence number)]
   Acknowledgment Number: 1    (relative ack number)
   Acknowledgment number (raw): 713274385
   0101 .... = Header Length: 20 bytes (5)
 > Flags: 0x018 (PSH, ACK)
   Window: 514
   [Calculated window size: 131584]
   [Window size scaling factor: 256]
   Checksum: 0x0d99 [unverified]
   [Checksum Status: Unverified]
   Urgent Pointer: 0
 > [SEQ/ACK analysis]
 > [Timestamps]
   TCP payload (463 bytes)
```

5. yes, I can see the server always uses the same port 12000 during welcome socket and new socket as below, build connection during 3-way handshake



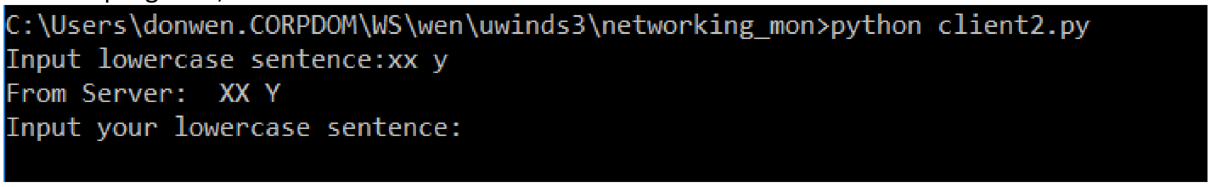
```
Transmission Control Protocol, Src Port: 3212, Dst Port: 12000, Seq: 0, Len: 0
    Source Port: 3212
    Destination Port: 12000
    [Stream index: 28]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Sequence number (raw): 1733820296
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 0
    Acknowledgment number (raw): 0
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
```

and data transfer after the connection was established:



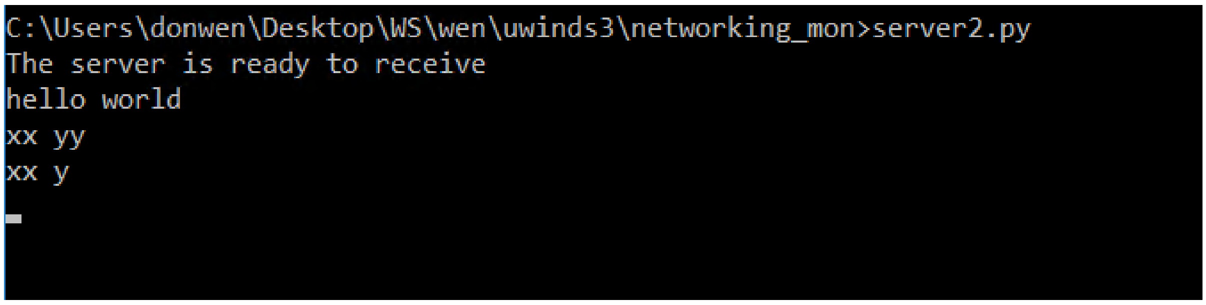
```
Destination Port: 12000
[Stream index: 28]
[TCP Segment Len: 4]
Sequence number: 1    (relative sequence number)
Sequence number (raw): 1733820297
[Next sequence number: 5    (relative sequence number)]
Acknowledgment number: 1    (relative ack number)
Acknowledgment number (raw): 655632230
0101 .... = Header Length: 20 bytes (5)
Flags: 0x018 (PSH, ACK)
Window size value: 256
[Calculated window size: 65536]
[Window size scaling factor: 256]
Checksum: 0x5146 [unverified]
[Checksum Status: Unverified]
```

Run the programs, client:



```
C:\Users\donwen.CORPDOM\WS\wen\uwinds3\networking_mon>python client2.py
Input lowercase sentence:xx y
From Server:  XX Y
Input your lowercase sentence:
```

Server:

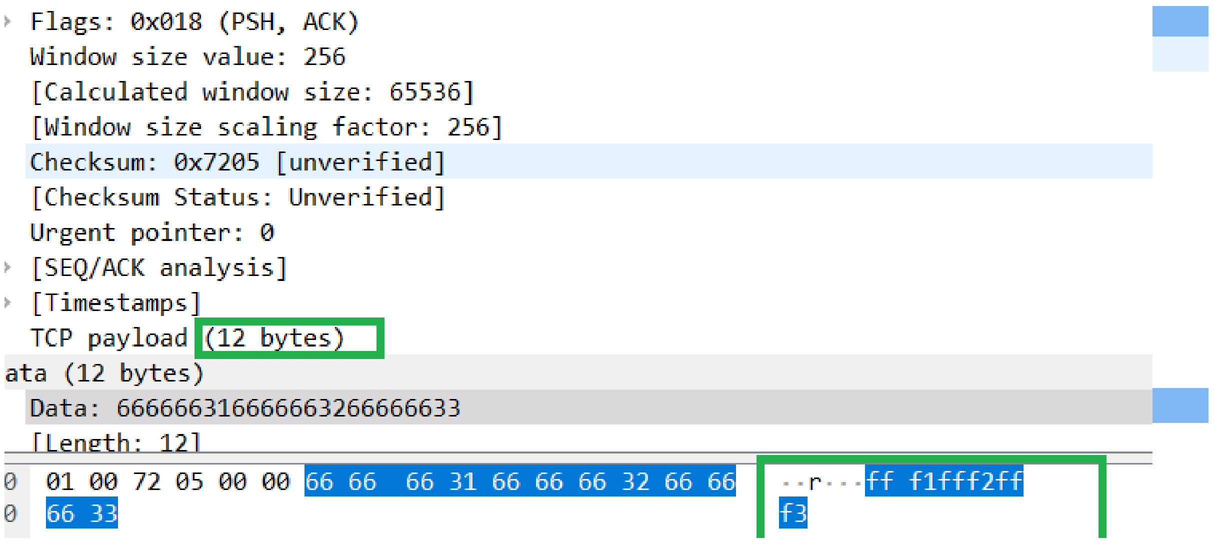


```
C:\Users\donwen\Desktop\WS\wen\uwinds3\networking_mon>server2.py
The server is ready to receive
hello world
xx yy
xx y
```

How the server works explained with inline comments:

```
#! /usr/bin/python3
from socket import *
from _thread import *
serverName = ""
serverPort = 12000

# create a socket in the server program
serverSocket = socket(AF_INET, SOCK_STREAM)

# bind it to a given port of the hosting server name, empty means any availabl
e name of the server.
serverSocket.bind((serverName, serverPort))

# start listening on the local port, 1 (backlog) means 1 unaccepted new incomi
ng connection is allowed in the waiting queue.
serverSocket.listen(1)
print("The server is ready to receive")
def multi_threaded_client(connectionSocket):
    # receive data from the connection.
    sentence = connectionSocket.recv(1024)
    # once there is incoming data received, read and process to the end.
    while sentence:
        capitalizedSentence = sentence.decode().upper()
        connectionSocket.send(capitalizedSentence.encode())
        print(sentence.decode())
        sentence = connectionSocket.recv(1024)
    # close the new connection when done.
    connectionSocket.close()

while True:
    # polling and welcoming new connection
    connectionSocket, addr = serverSocket.accept()

    # handle the new connection in a separate thread.
    start_new_thread(multi_threaded_client, (connectionSocket, ))
```

6. by running the program, I can see a single TCP packet was transferred to server for the three calls of send(..)



```
› Flags: 0x018 (PSH, ACK)
  Window size value: 256
  [Calculated window size: 65536]
  [Window size scaling factor: 256]
  Checksum: 0x7205 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
› [SEQ/ACK analysis]
› [Timestamps]
  TCP payload (12 bytes)
ata (12 bytes)
  Data: 666666316666663266666633
  [Length: 12]
0  01 00 72 05 00 00 66 66  66 31 66 66 66 32 66 66   ··r···ff f1fff2ff
0  66 33                                              f3
```