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](http://www.google.com).

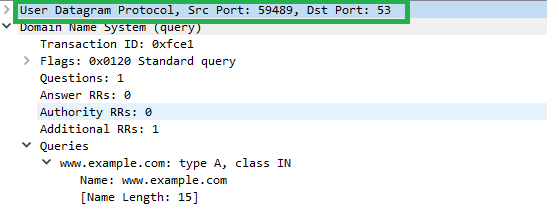
www.google.com. 300 IN A 172.217.13.4

a.

Text

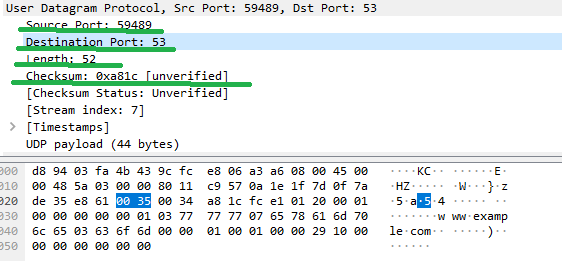
Description automatically generated

b.



The UDP headers are as learned from the wireshark screenshot:

1. Source port
2. Destination port
3. Length of the package
4. Checksum



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

1. Yes I can confirm the three packets establishing the connection when browsing [www.example.net](http://www.example.net)

A picture containing text

Description automatically generated

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

1. 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

1. print(modifiedMessage.decode())
2. # destruct the local socket in the end.
3. 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/welcome 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:

Graphical user interface, text

Description automatically generated

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)

Graphical user interface, text, application

Description automatically generated

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:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

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”

Graphical user interface, text, Word

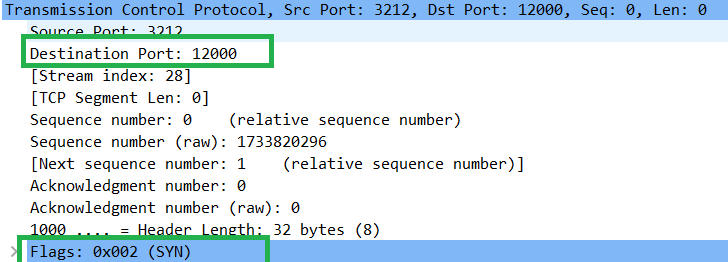
Description automatically generated

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

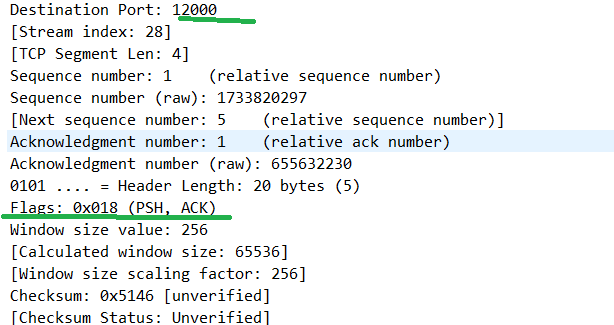
Text

Description automatically generated

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



and data transfer after the connection was established:



Run the programs, client:

Text

Description automatically generated

Server:

Text

Description automatically generated

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 available name of the server.

serverSocket.bind((serverName, serverPort))

# start listening on the local port, 1 (backlog) means 1 unaccepted new incoming 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(..)

