CS 6043 Computer Networking

PROJECT 1

By Mitra Sasanka Darbha, M#13523468

Part 1: HTTP

Question 1:

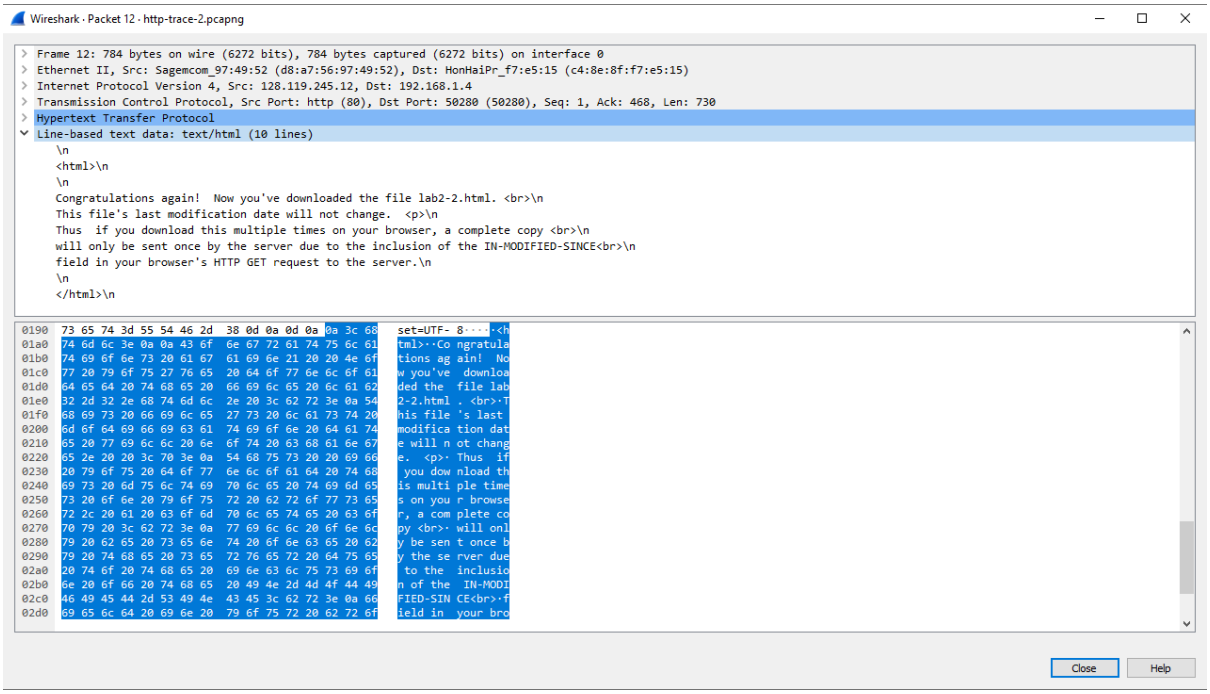
(a) Languages supported are: **en-US, en.** HTTP version 1.1 is running.

(b) **404 not found** is returned from the server to client browser for the second GET request. It means that when the user requests a page via an URL, the corresponding page is deleted or moved to another URL.

Question 2:

(a) The server implicitly returned the contents of file. The server returned HTTP status Code 200 – OK.

It returned HTTP-wireshar-file2.html



(b) The server returned **304 Not Modified** in response to the second HTTP GET. No, the server did not return the contents of the file since it returned only the status code. **304 Not Modified** is returned to the client when the cached copy of a particular file is up to date with the server.

Question 3:

The client downloaded the first two images in parallel.

If the packet capture is observed, the two images requests are packet 141 and 203. Responses came later in packets 456 and 483 respectively. That means that the responses are obtained after the two requests are completed. If it is in serial, the first image would be sent before the second image was requested.

Question 4:

(a) The response was **401 Unauthorized**.

No.	Time	Source	Destination	Protocol	Length	Info
81	1.655444	192.168.86.22	128.119.245.12	HTTP	445	GET /wireshark-labs/protected_pages/HTTP-wireshark-file5.html HTTP/1.1
90	1.686891	128.119.245.12	192.168.86.22	HTTP	771	HTTP/1.1 401 Unauthorized (text/html)
178	19.778179	192.168.86.22	128.119.245.12	HTTP	504	GET /wireshark-labs/protected_pages/HTTP-wireshark-file5.html HTTP/1.1
180	19.807951	128.119.245.12	192.168.86.22	HTTP	544	HTTP/1.1 200 OK (text/html)
182	20.089710	192.168.86.22	128.119.245.12	HTTP	321	GET /favicon.ico HTTP/1.1
183	20.118288	128.119.245.12	192.168.86.22	HTTP	538	HTTP/1.1 404 Not Found (text/html)

(b) The newly added field is the **Authorization** field. The login credentials are sent as plain-text.

```
Hypertext Transfer Protocol
  GET /wireshark-labs/protected_pages/HTTP-wireshark-file5.html HTTP/1.1\r\n
  Host: gaia.cs.umass.edu\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) Gecko/20100101 Firefox/69.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Authorization: Basic d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=\r\n
    Credentials: wireshark-students:network
  \r\n
  [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html]
  [HTTP request 1/2]
  [Response in frame: 180]
  [Next request in frame: 182]
```

Part 2: DNS

Question 1:

nslookup for determining authoritative DNS server and its IP for www.uc.edu

The command is: ***nslookup -type=soa uc.edu***

The screenshot is as shown:

```
Command Prompt
Microsoft Windows [Version 10.0.18362.388]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\MSD>nslookup -type=soa uc.edu
Server:  intucdnsa.uc.edu
Address:  10.27.3.2

wifi.uc.edu
        primary name server = intucdnsa.uc.edu
        responsible mail addr = noc.uc.edu
        serial  = 458090508
        refresh = 10800 (3 hours)
        retry   = 3600 (1 hour)
        expire  = 2419200 (28 days)
        default TTL = 900 (15 mins)

C:\Users\MSD>
```

Therefore, the Authoritative DNS server for uc.edu is **intusdnsa.uc.edu** and it's IP is **10.27.3.2**

Question 2:

nslookup for mail.uc.edu from Authoritative DNS of uc.edu

the command is: ***nslookup mail.uc.edu intucdnsa.uc.edu***

The screenshot is shown below:

Command Prompt

```
C:\Users\MSD>nslookup mail.uc.edu intucdnsa.uc.edu
Server:  intucdnsa.uc.edu
Address:  10.27.3.2

Name:    mail.uc.edu
Address:  10.23.135.36


C:\Users\MSD>
```

The IP address of mail.uc.edu is **10.23.135.36**

Question 3:

(a) DNS query and response messages are sent over UDP. The destination port for DNS query message and the source port for DNS response message is **53**.

(b) DNS query message is Type "**A**".

(c) Two answers are provided.
Each answer contains Name, Type, Class, Time to live, Data Length and IP address of www.ietf.org

```
Queries
  > www.ietf.org: type A, class IN
Answers
  v www.ietf.org: type A, class IN, addr 132.151.6.75
      Name: www.ietf.org
      Type: A (Host Address) (1)
      Class: IN (0x0001)
      Time to live: 1678
      Data length: 4
      Address: 132.151.6.75
  v www.ietf.org: type A, class IN, addr 65.246.255.51
      Name: www.ietf.org
      Type: A (Host Address) (1)
      Class: IN (0x0001)
      Time to live: 1678
      Data length: 4
      Address: 65.246.255.51
  [Request In: 8]
  [Time: 0.000844000 seconds]
```

Question 4:

(a) Type: A
Class: IN (0x0001)
Ip Address: 216.58.217.238

(b) Refresh Interval: 1200 (20 minutes)
Minimum TTL: 86400 (1 day).

Part 3: Socket Programming

Question 1: Web Server

webserver.py

```
# Import socket module

from socket import *


# Create a TCP server socket

#(AF_INET is used for IPv4 protocols)

#(SOCK_STREAM is used for TCP)

serverSocket = socket(AF_INET, SOCK_STREAM)

# Prepare a server socket

# FILL IN START

# Assign a port number

serverPort = 12000

# Bind the socket to server address and server port

serverSocket.bind(('', serverPort))

# Listen to at most 1 connection at a time

serverSocket.listen(1)

# FILL IN END

# Server should be up and running and listening to the incoming connections

while True:

        print ('Ready to serve...')

        # Set up a new connection from the client

        connectionSocket, addr = serverSocket.accept()
```

```
# If an exception occurs during the execution of try clause
# the rest of the clause is skipped
# If the exception type matches the word after except
# the except clause is executed
try:
        # Receives the request message from the client
        message =  connectionSocket.recv(1500)

        # Extract the path of the requested object from the message
        # The path is the second part of HTTP header, identified by [1]
        filepath = message.split()[1]

        # Because the extracted path of the HTTP request includes
        # a character '\', we read the path from the second character
        f = open(filepath[1:])

        # Read the file "f" and store the entire content of the requested file in a temporary
buffer
        outputdata = f.read()

        # Send the HTTP response header line to the connection socket
        # Format: "HTTP/1.1 *code-for-successful-request*\r\n\r\n"
        # FILL IN START
        connectionSocket.send(bytes("HTTP/1.1 200 OK\r\n\r\n","UTF-8"))
        # FILL IN END
        # Send the content of the requested file to the connection socket
        for i in range(0, len(outputdata)):
                connectionSocket.send(bytes(outputdata[i],"UTF-8"))
        connectionSocket.send(bytes("\r\n","UTF-8"))

        # Close the client connection socket
```

```
            connectionSocket.close()

        except IOError:
            # Send HTTP response message for file not found
            # Same format as above, but with code for "Not Found"
            # FILL IN START
            connectionSocket.send(bytes("HTTP/1.1 404 Not found\r\n\r\n","UTF-8"))
            # FILL IN END
            connectionSocket.send(bytes("<html><head></head><body><h1>404 Not found</h1></body></html>\r\n","UTF-8"))
            # Close the client connection socket
            # FILL IN START
            connectionSocket.close()
            # FILL IN END

serverSocket.close()
```

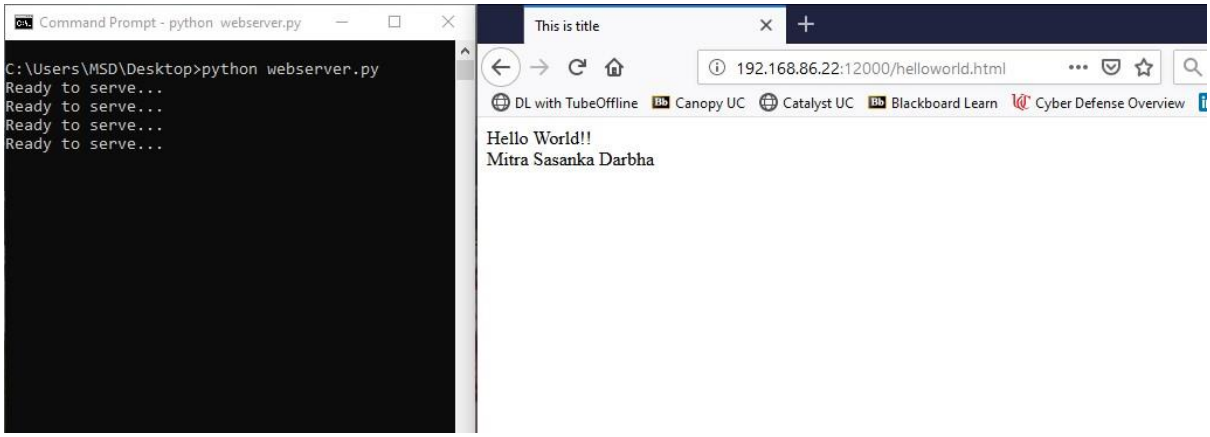
helloworld.html

```
<html>
<header><title>This is title</title></header>
<body>
Hello World!!
</br>
Mitra Sasanka Darbha
</body>
</html>
```

The code is executed in command prompt and Firefox browser is used as client.

It is executed on port 12000, and the file name is helloworld.html

The screenshots are below:



If any other file is executed “404 Not found” message is displayed.

Since “hello.html” does not exist, “404 not found” is displayed as shown.

Question 2: Simple Mail Client

send_mail.py

```
from socket import *

import base64

import time

import ssl


msg = "\r\n Mitra Sasanka Darbha Computer Networks!"

endmsg = "\r\n.\r\n"

mailserver = ("smtp.gmail.com",465) #Fill in start #Fill in end

ssl_version=ssl.PROTOCOL_SSLv23)

clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocketSSL = ssl.wrap_socket(clientSocket)

clientSocketSSL.connect(mailserver)

recv = clientSocketSSL.recv(1024)
```

```
recv = recv.decode()
print("Message after connection request:" + recv)
if recv[:3] != '220':
    print('220 reply not received from server.')
    ## Send HELO command and print server response.
heloCommand = 'EHLO Alice\r\n'
clientSocketSSL.send(heloCommand.encode())
recv1 = clientSocketSSL.recv(1024)
recv1 = recv1.decode()
print("Message after EHLO command:" + recv1)
if recv1[:3] != '250':
    print('250 reply not received from server.')

#Info for username and password
username = "sasanka.msd97@gmail.com"
password = "xxxxxxxxxx" #my password here
base64_str = ("\x00"+username+"\x00"+password).encode()
base64_str = base64.b64encode(base64_str)
authMsg = "AUTH PLAIN ".encode()+base64_str+"\r\n".encode()
clientSocketSSL.send(authMsg)
recv_auth = clientSocketSSL.recv(1024)
print(recv_auth.decode())

mailFrom = "MAIL FROM:<sasanka.msd97@gmail.com>\r\n"
clientSocketSSL.send(mailFrom.encode())
recv2 = clientSocketSSL.recv(1024)
recv2 = recv2.decode()
print("After MAIL FROM command: "+recv2)
rcptTo = "RCPT TO:<sasanka.msd97@gmail.com>\r\n"
clientSocketSSL.send(rcptTo.encode())
recv3 = clientSocketSSL.recv(1024)
```

```
recv3 = recv3.decode()
print("After RCPT TO command: "+recv3)
data = "DATA\r\n"
clientSocketSSL.send(data.encode())
recv4 = clientSocketSSL.recv(1024)
recv4 = recv4.decode()
print("After DATA command: "+recv4)
subject = "Subject: testing my client\r\n\r\n"
clientSocketSSL.send(subject.encode())
date = time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmtime())
date = date + "\r\n\r\n"
clientSocketSSL.send(date.encode())
clientSocketSSL.send(msg.encode())
clientSocketSSL.send(endmsg.encode())
recv_msg = clientSocketSSL.recv(1024)
print("Response after sending message body:"+recv_msg.decode())
quit = "QUIT\r\n"
clientSocketSSL.send(quit.encode())
recv5 = clientSocketSSL.recv(1024)
print(recv5.decode())
clientSocketSSL.close()
clientSocket.close()
```

When this code is executed in shell the output is as follows:

```
Python 3.7.4 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/MSD/Documents/1 UC/CN/Project/Project 1/smtp/smtp_email.py
Message after connection request:220 smtp.gmail.com ESMTP 178sm4368658ywb.66 - gs
mtp

Message after EHLO command:250-smtp.gmail.com at your service, [74.83.207.65]
250-SIZE 35882577
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-CHUNKING
250 SMTPUTF8

235 2.7.0 Accepted

After MAIL FROM command: 250 2.1.0 OK 178sm4368658ywb.66 - gsmtp

After RCPT TO command: 250 2.1.5 OK 178sm4368658ywb.66 - gsmtp

After DATA command: 354  Go ahead 178sm4368658ywb.66 - gsmtp

Response after sending message body:250 2.0.0 OK  1570499779 178sm4368658ywb.66 -
gsmtp

221 2.0.0 closing connection 178sm4368658ywb.66 - gsmtp

>>>
```

The mail received is as shown: