LAB 1 (HTTP AND WEB SERVER LAB)

PROBLEM STATEMENT

In this lab, you will learn the basics of socket programming using TCP connections in Python:

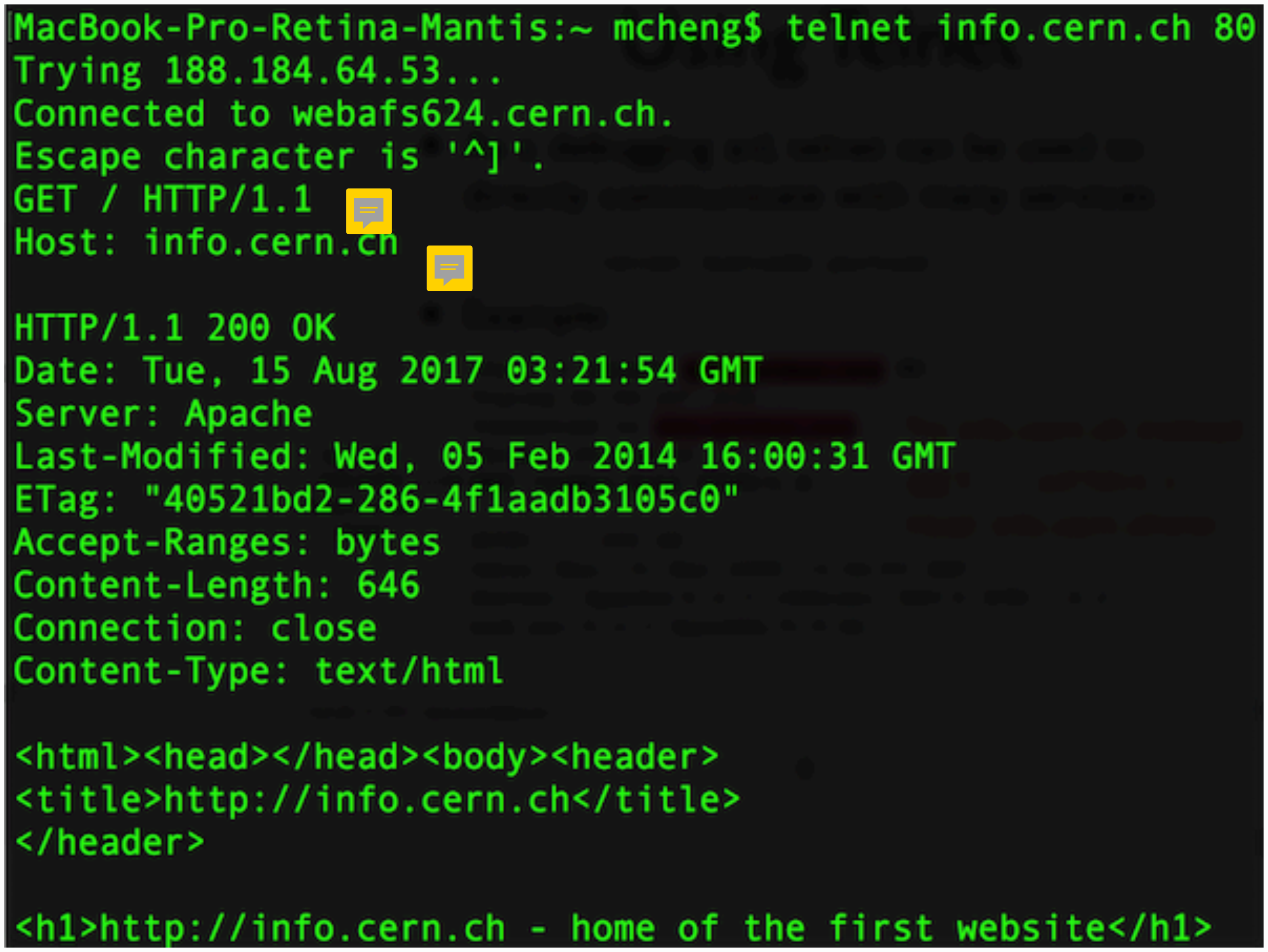
- how to create a socket,
- bind it to a specific address and port, as well as
- send and receive a HTTP packet.

You will also learn some basics of HTTP header format.

You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP `404 Not Found` message back to the client. You are also **required** to implement a simple client in Python after you have completed your server.

HTTP PROTOCOL

Start `CSC361-VM` , use `telnet` into `info.cern.ch` with port 80 and request the world's first HTML page. Now, do it again with the `If-Modified-Since:` header line in your request, so that the server returns a status `304 Not Modified` . (**Note:** Follow the steps shown below.)



```
MacBook-Pro-Retina-Mantis:~ mcheng$ telnet info.cern.ch 80
Trying 188.184.64.53...
Connected to webafs624.cern.ch.
Escape character is '^]'.
GET / HTTP/1.1
Host: info.cern.ch

HTTP/1.1 200 OK
Date: Tue, 15 Aug 2017 03:21:54 GMT
Server: Apache
Last-Modified: Wed, 05 Feb 2014 16:00:31 GMT
ETag: "40521bd2-286-4f1aadb3105c0"
Accept-Ranges: bytes
Content-Length: 646
Connection: close
Content-Type: text/html

<html><head></head><body><header>
<title>http://info.cern.ch</title>
</header>

<h1>http://info.cern.ch - home of the first website</h1>
```

This helps you understand the HTTP based protocol. There is nothing to submit for this part. It is intended for you to learn what HTTP protocol is about.

SAMPLE CODE

Included is a skeleton code for the Web server. You are required to complete the skeleton code `server.py`. The places where you need to fill in code are marked with `???????????`. Each place may require one or more lines of code. (**Note:** You may modify `server.py` any way you like, as long as it meets the above requirements. You may even ignore this sample code and start all over from scratch.)

RUNNING THE SERVER

Determine the IP address of the host that is running the server (e.g., `128.238.251.26`). Then run a working version of your server program `server.py`. From another host, open a browser and provide the corresponding URL. For example:

```
http://128.238.251.26:6789/hello.html
```

`hello.html` is the name of the file you placed in the server directory, which is also included. **Note** also the use of the port number after the colon. You need to replace this port number with whatever port you have chosen in your server code. In the above example, we have used the port number `6789`. The browser should then display the contents of `hello.html`. If you omit `:6789`, the browser will assume port `80` and you will get the web page from the server only if your server is listening at port `80`. (This is **not** recommended.) Then try to get a file that is not present at your server. You should get a `404 Not Found` message.

RUNNING THE CLIENT

Instead of using a browser, write your own HTTP client to test your server. Your client will connect to the server using a TCP connection, send an HTTP request to the server, and display the server response as an output. You can assume that the HTTP request sent is a GET method. The client should take command line arguments specifying the server IP address or host name, the port at which the server is listening, and the path at which the requested object is stored at the server. The following is an input command format to run the client.

```
client.py 128.338.251.26 6789 hello.html
```

WHAT TO HAND IN

You will hand in the complete client/server code along with the screen shots of your client. Please verify that you actually receive the contents of the HTML file from the server. Submit your solutions as attachments to Connex.

EVALUATION

You must demonstrate your working code inside the `CSC361-VM` using **Wireshark**. Using `CSC361-VM`, the server and the client must use different IP addresses and port numbers, i.e., not just `127.0.0.x`. Your Wireshark demo must capture all related packets, including ARP, TCP and HTTP. Test your HTTP client/server using the provided `hello.html`.

You are **also** required to answer a few questions related to this lab during evaluation in the lab (ECS 360).

Our evaluation scheme is:

- (50%) Correctness of Python code
- (20%) Wireshark demo
- (30%) Questions and Answers (at least 5 questions)