Department of Electrical and Computer Engineering

ECE 358: Computer Networks

Project 3: Socket Programming using Python

Table of Contents

1 Objective

In this lab, you will learn the basics of socket programming in Python: how to create a socket, bind it to a specific address and port, as well as send and receive a packet.

2 Programming Language

You must use Python 3 for this lab. TAs are trained to use Python 3.10 and above.

3 Task 1: Develop a web server that handles HTTP request.

3.1 Problem Description:

You will develop a program in Python for a web server that handles one HTTP request at a time.

3.2 Overview:

- You will develop a server program in Python.
- Your web server should accept and parse the HTTP request, then get the requested file from the server's file system. Your server should be able to handle GET and HEAD requests (other methods won't be tested). Use Postman to test GET and HEAD requests.
- Create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. The minimum header lines is six. It contains header fields of Connection, Date, Server, Last-Modified, Content-Length and Content-Type. For more information on header fields, please see RFC 2616 at https://datatracker.ietf.org/doc/html/rfc2616.
- If the requested file is not present in the server, the server should send an HTTP "404 Not Found" message back to the client.
- Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. The content of the html file is your choice, but keep it professional and useful for debugging.
- In this lab, we will be using the host's loop back address which is 127.0.0.1. You are free to specify the port number, but the port number must be a number bigger than 1023. We recommend the number in the range of [5,000, 11,000]. When your server starts up, the first thing it must print is the port number to stdout followed by a newline without any other output. For example, it must print "8001" if your port is 8001.
- From the same host, open a browser and provide the corresponding URL using the format http://Server_IP_Address:Port_Numbr/"requested_file_name". For example: http://127.0.0.1:6789/HelloWorld.html, where 'HelloWorld.html' is the name of the file you placed in the server directory.
- Note also the use of the port number after the colon. You need to replace this port number with whatever port number you have used in the server code. In the above example, we have used the port number 6789. The browser should then display the contents of HelloWorld.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.
- Next, try to get a file that is not present at the server. You should get a "404 Not Found" message.
- Your server is required to handle html files that are nested in the subdirectories of the directory where the server is in. You will be given a path to the HTML file you want to serve in the request. For instance, /templates/index/index.html
- We implement persistent http in this lab. By persistent, we mean the server socket stays open but the client socket closes after each sendall(). To avoid complication of implementing Timeout we have slightly modified this part. Everything else should work as persistent http.
- You must use the Python 'socket' library to implement your webserver. Any library that bypasses that requirement is not allowed. Otherwise, you are permitted to use any library that comes with a standard install of Python 3.10 and above. If you rely on anything that you had to use pip install for (or any other package installer) it is unlikely to work during testing on TA computers.
- We only ask for server code in Task-1, not the client code, client will be a WebBrowser in this case.
- Learn may modify the html file. Don't worry about that.

3.3 Deliverables:

- Fill in the report template from Learn for Task 1
- You must hand in the complete server code as a Python file named “webserver.py”. Your code should run with the Linux command: python3 webserver.py

4 Task 2: Design and implement an Authoritative DNS server using a Client-Server system that uses UDP socket.

4.1 Problem Description:

The client program sends a request containing domain-name to the server, and the server replies with IP Address/es that corresponds to the domain name. A DNS query will be sent from the client to the server, and the server will send a DNS response to the client.

4.2 Overview:

- You will develop a client program, that must be named “client.py” and a server program, that must be named “server.py” in Python. Open two separate terminal windows, one for the client and the other one for the server. After executing the programs in the respective terminals, communication between the server and the client will be established.
  Example (client terminal):
  ```
  eceubuntu1:~/ECE_358/Lab_2/HJ> python3 client.py

  Enter Domain Name
  ```
- The client initiates communication with a server. The server remembers the client for the entire duration of the communication session. The client then runs in an infinite loop where it accepts a domain-name from the user through the command line.
- A DNS query message is created on the client side. Follow the guidelines provided in the “DNS Message Format” section in this manual to create the DNS query. The query message (in Hex/byte array or any other format) is then forwarded to the DNS server.
- Once the DNS server receives the message, it will parse the message and accordingly generate a response message following the guidelines provided in the “DNS Message Format” section in this manual. The response message (in Hex) is then forwarded to the client.
- The server-side terminal should display both request and response messages (in Hex) in the following format.
  The server terminal should display according to the exact format below:
  ```
  Request:
  1a  2b  04  00  00  01  00  00  00  00  00  00  06  67  6f  6f
  67  6c  65  03  63  6f  6d  00  00  01  00  01
  Response:
  1a  2b  84  00  00  01  00  02  00  00  00  00  06  67  6f  6f
  67  6c  65  03  63  6f  6d  00  00  01  00  01  c0  0c  00  01
  00  01  00  00  01  04  00  04  c0  a5  01  01  c0  0c  00  01
  00  01  00  00  01  04  00  04  c0  a5  01  0a
  ```
- The client will parse the message and display the results. The client terminal should display output in the following format once a DNS request is made.
  Client terminal should display:
  Input from the user:
  ```
  > Enter Domain Name: google.com
  ```
  Output:
  ```
  > google.com: type A,  class IN,  TTL 260,  addr  (4) 192.165.1.1
  > google.com: type A,  class IN,  TTL 260,  addr  (4) 192.165.1.10
  ```
  *Note: (4) represents the length of the records in bytes.
- The server holds at least five domain names and the corresponding IP addresses (see table 1). The server runs an infinite loop where it keeps waiting for requests from the client.
- If the user enters “end,” the communication session ends (it must not send the message to the server, just close the client connection). Upon ending the session, it prints “Session ended” on the client terminal.
- We assume that the input will only be provided from Table 1.

Example format:

```
> Enter Domain Name: end
Session ended
```

- You will use the loop back address 127.0.0.1 for your server, and the client will start on the same host where the server runs. The port number can be any number bigger than 1023. Recommended port number range is [10,000, 11,000].
- You code should handle upper case and lower-case input scenarios. During testing, the TA will only input domains that do exist in the DNS server (Table-1).

4.3 Deliverables in report format:

- Fill in the TASK 2 report requirements in the template
- Submit Python code for the server program. The file name should be "server.py".
- Submit Python code for the client program. The file name should be "client.py".
    - server.py must be runnable via the Linux command python3 server.py. client.py must be runnable via the Linux command python3 client.py.
    - You may not use any libraries that are not included in a standard Python 3.10 installation

Table 1: The list of domain names and corresponding IP addresses.

Sl no.	Domain name	Type	Class	TTL	IP address
1	google.com	A	IN	260	192.165.1.1 192.165.1.10
2	youtube.com	A	IN	160	192.165.1.2
3	uwaterloo.ca	A	IN	160	192.165.1.3
4	wikipedia.org	A	IN	160	192.165.1.4
5	amazon.ca	A	IN	160	192.165.1.5

4.4 DNS Message Format:

DNS allows you to interact with devices on the Internet without having to remember long strings of numbers. Changing of information between client and server is carried out by two types of DNS messages:

- Query message
- Response message.

The format is similar for both types of messages.

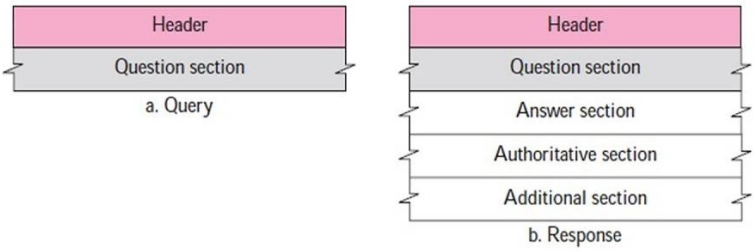

Fig 1: DNS Query and Response Message.

4.4.1 DNS Header:

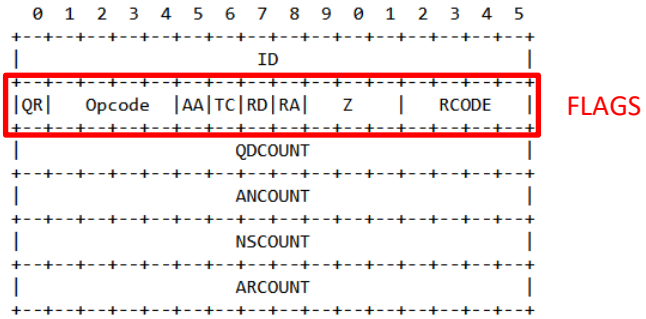

Fig 2: DNS Header Format

ID: A 16-bit identifier assigned by the program that generates any kind of query.
*Generate randomly. The response header id need to match the request header id
QR: A one bit field that specifies whether this message is a query (0), or a response (1).
*Select based on the type of message.
OPCODE: A four bit field that specifies kind of query in this message. This value is set by the originator of a query and copied into the response. The values are:
0 a standard query (QUERY).
1 an inverse query (IQUERY)
2 a server status request (STATUS)
3-15 reserved for future use.
*For this lab set OPCODE as 0.
AA: Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in question section.
* For this lab set AA as 1.
TC: TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel.
* For this lab set TC as 0.
RD: Recursion Desired - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively. Recursive query support is optional.
* For this lab set RD as 0.
RA: Recursion Available - this be is set or cleared in a response, and denotes whether recursive query support is available in the name server.
* For this lab set RA as 0.
Z: Reserved for future use. Must be zero in all queries and responses.
* For this lab set Z as 000.
RCODE: Response code - this 4-bit field is set as part of responses. The values have the following interpretation:
0 No error condition
1 Format error - The name server was unable to interpret the query.
2 Server failure - The name server was unable to process this query due to a problem with the name server.
3 Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.
4 Not Implemented - The name server does not support the requested kind of query.
5 Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation.
* For this lab set RCODE as 0.
QDCOUNT: an unsigned 16-bit integer specifying the number of entries in the question section.
* For this lab set QDCOUNT as 1.
ANCOUNT: an unsigned 16-bit integer specifying the number of resource records in the answer section.
* Set ANCOUNT based on message type.
NSCOUNT: an unsigned 16-bit integer specifying the number of name server resource records in the authority records section.
* For this lab set NSCOUNT as 0.
ARCOUNT: an unsigned 16 bit integer specifying the number of resource records in the additional records section.
* For this lab set ARCOUNT as 0.

4.4.2 Question section format:

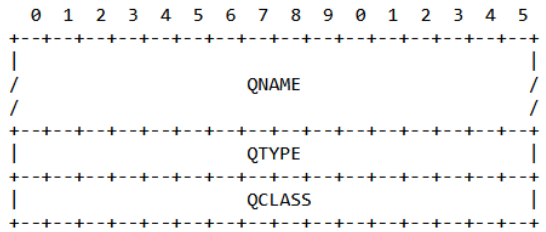

Fig 3: DNS Question Format

QNAME: a domain name represented as a sequence of labels, where each label consists of a length octet followed by that number of octets. The domain name terminates with the zero length octet for the null label of the root. Note that this field may be an odd number of octets; no padding is used.
*See the example below to set this field.
QTYPE: a two octet code which specifies the type of the query. The values for this field include all codes valid for a TYPE field, together with some more general codes which can match more than one type of RR.

TYPE	**value and meaning**
A	1 a host address
NS	2 an authoritative name server
MD	3 a mail destination (Obsolete - use MX)
MF	4 a mail forwarder (Obsolete - use MX)

* For this lab use TYPE A. Set this field according.
QCLASS: a two octet code that specifies the class of the query.
*Set QCLASS field as IN (00 01) (hex value) for the Internet.

4.4.3 Answer section format:

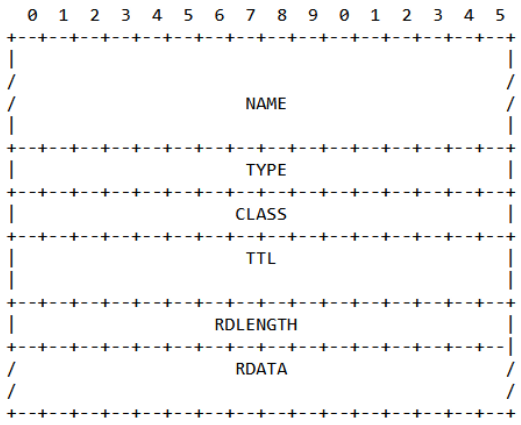

Fig 4: Resource Record (RR) format

NAME: an owner name, i.e., the name of the node to which this resource record pertains.
*set this field as (c0 0c) (hex value).
TYPE: two octets containing one of the RR TYPE codes.
*set this field accordingly. We only deal with TYPE A message.
CLASS: two octets containing one of the RR CLASS codes.
*set this field accordingly. We only deal with IN message.
TTL: a 32 bit unsigned integer that specifies the time interval that the resource record may be cached before the source of the information should again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. For example, SOA records are always distributed with a zero TTL to prohibit caching. Zero values can also be used for extremely volatile data.
*set this field accordingly. The info is provided in Table 1.
RDLENGTH: an unsigned 16 bit integer that specifies the length in octets of the RDATA field.
*set this field accordingly. The info is provided in Table 1.
RDATA: a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record.
*set this field accordingly. The info is provided in Table 1.

4.4.4 Authoritative section & Additional Section:

*Do not include these fields in the DNS response message.

4.4.5 Example:

The example shows a DNS Query. From the table, you can identify different fields in the message based on their location and length.

1a 2b 04 00 00 01 00 00 00 00 00 00 06 67 6f 6f
67 6c 65 03 63 6f 6d 00 00 01 00 01

Table 2: DNS Query.

Type	Key	Value
DNS HEADER	ID	1a 2b
	FLAGS	04 00
	QDCOUNT	00 01
	ANCOUNT	00 00
	NSCOUNT	00 00
	ARCOUNT	00 00
QUERY	QNAME	06 67 6f 6f 67 6c 65 03 63 6f 6d 00
	QTYPE	00 01
	QCLASS	00 01

5 Task 3: Short-Answer Questions

Answer the questions in the TASK 3 section of the report template.

6 Submission Guidelines

Submit the following in a *.zip file to the dropbox on LEARN. Your zip file may not include subdirectories. It must be named using your (maximum) 8-character UW ID plus the suffix _lab3.zip, such as “mstachow_lab3.zip”. Incorrectly named files will lose marks.

1. Submit your filled-in report template in *.pdf format. Do not leave it as *.docx.
2. Source code with proper comments. Source code submitted without comments or with insufficient comments will lose marks.

You may be asked to give a demo of your simulator.

7 References

[1] J. F. Kurose and K. W. Ross: Computer Networking, A Top-Down Approach. 8th Edition.
[2] P. Mockapetris, Domain Names – Implementation and Specification, document RFC 1035, 1987.