# COMS3200 Assignment 1

Due: Wednesday 27th March 2019, 20:00

100 marks total (scaled to 15% of final grade)

## Part A (25 marks)

A client requests a webpage from a remote server on a remote island via a slow satellite link above the earth in geostationary orbit.

The goal of this exercise is to use the following information to calculate the time for the request to be completed. The scenario is in no way intended to be realistic.

There is a client (C), a server (V), and a DNS server (D). These are connected by three switches (S1, S2, S3) and five transmission links (L1 to L5). L2 is the satellite link. The following tables provide further information about the network (bps == bits per second, 1 kbps == $10^3$ bps, 1 Mbps == $10^6$ bps).

### Transmission Links

| Link Name | Connected to | Connected to | Transmission Rate | Length | Propogation Speed |
|---|---|---|---|---|---|
| L1 | C | S1 | 100Mbps | 150m | $2 \times 10^8$ m/s |
| L2 | S1 | S2 | 5kbps | 36,000km | $3 \times 10^8$ m/s |
| L3 | S2 | V | 100Mbps | 150m | $2 \times 10^8$ m/s |
| L4 | S1 | S3 | 100Mbps | 6,000km | $2 \times 10^8$ m/s |
| L5 | S3 | D | 100Mbps | 150m | $2 \times 10^8$ m/s |

### Switches

| Switch | Links on Ports | Processing Delay |
|---|---|---|
| S1 | L1, L2, L4 | 500$\mu$s |
| S2 | L2, L3 | 1ms |
| S3 | L4, L5 | 500$\mu$s |

## Assumptions

You may make the following assumptions about the network:

- The network is packet-switched
- The network starts with no other packets in queues - only packets that are a part of this question
- Each link is bidirectional and can concurrently handle bits travelling in opposite directions
- All required records are stored in the DNS server
- C, D, and V have no processing delays (while this is unrealistic, this simplifies calculations)
- DNS packets and TCP SYN, ACK, FIN, SYN/ACK, and FIN/ACK packets are 100 bytes long, including all headers, preamble, etc
- HTTP GET and HTTP response packets are 1000 bytes long, including all headers, preamble, etc

## Scenario

At time $t = 0$, a program on the client (C), starts the process of retrieving a webpage from the server (V). The following events happen sequentially:

1. C sends a DNS request to D
2. D sends a response with V's IP address to C
3. C sends a TCP request (SYN) to open a connection to V
4. V acknowledges (SYN/ACK) the TCP request and opens the connections
5. C sends a HTTP GET request to V (which includes the ACK back to V for its SYN)
6. V sends the web page to C, which requires 5 packets, which includes the HTTP response plus the HTML file.
7. After receiving each data packet, C sends an ACK message back to V. Note that V does not need to wait for an ACK before sending the next data packet.
8. After receiving the last data packet acknowledgement, V sends a TCP FIN packet to close the connection.
9. After receiving FIN, C sends one FIN/ACK packet back to V.
10. After receiving FIN/ACK, V sends a final ACK packet back to C.
11. When this final ACK packet is received at C, the connection is finally closed.

## Questions

Answer each of the following questions in the associated quiz on blackboard, following the specified instructions. All answers should be in milliseconds (ms) rounded to at least four decimal places.

1. At what time does D receive the DNS request? (3 marks)
2. At what time does C receive the DNS response? (3 marks)
3. At what time does V receive the TCP SYN request? (3 marks)
4. At what time does C receive the TCP SYN/ACK acknowledgement? (3 marks)
5. At what time does V receive the HTTP GET packet? (3 marks)
6. At what time does C receive the fifth (last) packet of the HTTP response? (5 marks)
7. At what time is the connection closed? (5 marks)

*HINT: Make sure you correctly convert between units*

## Part B (25 marks)

This question requires you to analyse the Wireshark file *Assignment1.pcapng*. This packet capture shows a client connecting to an email server from their home PC.

You will need to read the relevant IETF RFCs for the application level protocols used in this trace to answer some of these questions.

Answer each of the following questions in the associated quiz on blackboard, following the specified instructions. Each question is worth 2.5 marks.

The first six questions relate to the message being sent:

1. What application layer protocol is used to send the email message?
2. What is the IP address and port number of the mail client?
3. What is the IP address and port number of the mail server?
4. What mail client is the sender using?
5. What is the first line of the body of the message? Omit any ASCII characters with a decimal value less than 32 or greater than 126.
6. What is the maximum message size (in bytes) that the mail server will accept?

The following questions relate to the message being received:

7. What application layer protocol is used to read the email message from the server?
8. What is the username and password that is used to login to the server?
9. How many packets are used to transmit the body of the message that is being received?
10. What is the size (in bytes) of the first email in the in the maildrop? (i.e. the message with ID 1)

## Part C (50 marks)

Your task for this part is to write a program to retrieve a file on a webserver via HTTP. Your program should make use of sockets to send and receive HTTP requests/responses and must be written in Python, Java, C, or C++. It is recommended that you use Python as more support will be available for it in this course.

### Description

**Base Functionality (25 marks)**

Your program should take a HTTP URL as a command line input, leading to a file on a webserver. This URL could just be a domain name (eg. http://www.my.server.com) or include the resource location (eg. http://www.my.server.com/about or http://www.my.server.com/file.json). The exact method of program invocation is described later.

Your program should open a TCP connection to the webserver and print the following information to stdout:

```
URL Requested:  [url]
Client:  [client-ip-addr] [client-port-num]
Server:  [server-ip-addr] [server-port-num]
```

You do not need to handle invalid domains. Upon successful connection, your program should make a HTTP request to the server. After receiving the associated response, the following information should then be printed:

```
Retrieval Successful
Date Accessed:  [dd/mm/yyyy] [hh:mm:ss] AEST
Last Modified:  [dd/mm/yyyy] [hh:mm:ss] AEST
```

Both fields should be as per the value given in the HTTP response. Both time fields should be converted from UTC to AEST if required. The last modified field may not always be given - if this is the case, the text "`Last Modified not available`" should replace that line.

The contents of the file retrieved from the webserver should be written to a file named "output.[extension]". [extension] should be replaced with an appropriate extension based on the MIME type of the retrieved file. You only have to support the MIME types given in the table below. You can read more about MIME types at `https://tools.ietf.org/html/rfc6838`. Note that the extension of the URL is not always indicative of a file's MIME type, you will need to retrieve this from the HTTP response.

**Supported MIME Types**

| MIME Type | File Extension |
|---|---|
| text/plain | .txt |
| text/html | .html |
| text/css | .css |
| text/javascript or application/javascript | .js |
| application/json | .json |
| application/octet-stream | No extension |

*HINT: Because TCP is a stream-based protocol, long HTTP responses may be transmitted over multiple packets. Your program should make sure it has received the entire file before terminating.*

### Handling Unsupported URLs (10 marks)

Your program does not need to work for HTTPS URLs. If a HTTPS URL is requested your program should print the following and terminate:

```
URL Requested:  [url]
HTTPS Not Supported
```

If the status code of a HTTP response is in the range of 400-599, your program should notify the user of this and terminate, as follows:

```
URL Requested:  [url]
Client:  [client-ip-addr] [client-port-num]
Server:  [server-ip-addr] [server-port-num]
Retrieval Failed ([code])
```

### Redirection (15 marks)

The final feature your program should support is handling of 301 and 302 status codes. If a resource has been moved, you should repeat the above process until either the resource is found, or an invalid/unsupported URL is given. For example:

```
URL Requested:  [url]
Client:  [client-ip-addr] [client-port-num]
Server:  [server-ip-addr] [server-port-num]
Resource [temporarily/permanently] moved to [url]
Client:  [client-ip-addr] [client-port-num]
Server:  [server-ip-addr] [server-port-num]
Retrieval Successful
Date Accessed:  [dd/mm/yyyy] [hh:mm:ss] AEST
Last Modified:  [dd/mm/yyyy] [hh:mm:ss] AEST
```

It is possible that you are redirected multiple times, in which case, the "moved to" line and client/server info should be repeated for each redirection. You can complete redirections over a persistent TCP connection or you may choose to create a new socket on each redirection. If you choose to do the former, you will have to handle the server closing the connection (and you should still re-print the client/server information if the connection is not closed).

## Program Invocation

Your program should be able to be invoked from a UNIX command line as follows. `url` is the URL of the webpage to request.

### Python

```
python3 assign1.py url
```

### C/C++

```
make
./assign1 url
```

### Java

```
make
java Assign1 url
```

## Example Output

Note that for the following examples, the client/server information and dates may not be accurate.

The following is an example output for a request to `http://uq.edu.au/`:

```
URL Requested:  http://uq.edu.au/
Client:  192.168.12.15 54321
Server:  10.187.2.85 80
Retrieval Successful
Date Accessed:  04/03/2019 10:37:33 AEST
Last Modified:  04/03/2019 10:35:01 AEST
```

In this case, the user should be able to view the contents of this webpage in a file named "output.html".

An example for a request to `http://uq.edu.au/missing` is below:

```
URL Requested:  http://uq.edu.au/missing
Client:  192.168.12.15 54321
Server:  10.187.2.85 80
Retrieval Failed (404)
```

An example for a request to `http://abc.net.au/` is below:

```
URL Requested:  http://abc.net.au/
Client:  192.168.12.15 54321
Server:  10.187.2.85 80
Resource permanently moved to http://www.abc.net.au/
Client:  192.168.12.15 54321
Server:  10.187.2.85 80
Resource temporarily moved to https://www.abc.net.au/
HTTPS Not Supported
```

It should be noted that all output uses single-spacing, not double-spacing (as it may appear in some examples).

## Library Restrictions

- You should use standard socket libraries to open the TCP connection and communicate between the client and server
- You should NOT use higher level libraries, packages, or programs which retrieve data from HTTP servers, such as the python `requests` and `urllib` libraries.
  - The only exception is that you may use text parsing functions which are a part of `urllib` (i.e. the `urllib.parse` module
- If you are unsure about whether you may use a certain library, please ask the course staff on Piazza

### Submission

Submit all files necessary to run your program. At a minimum, you should submit a file named `assign1.py`, `assign1.c`, `assign1.cpp` or `Assign1.java`. If you submit a C or Java program, you should also submit a makefile to compile your code into a binary named `assign1` or a .class file named `Assign1.class`.

### Marking

Your code will be automatically marked, so it is essential that your program's output is exactly as specified above. It is expected that you will receive a small sample of tests before the submission deadline.

There are no marks for coding style.

## Academic Misconduct

Students are reminded of the University's policy on student misconduct, including plagiarism. See the course profile and the School web page `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`.

## Late Submission and Extensions

Please view the ECP for policies on late submissions and extensions.

## Version History

### v1.0

- Released assignment

### v1.1 (11/03/19)

Fixes:

- Added missing port to S1 in part A
- Added missing "Retrieval Successful" to redirection example

Clarifications:

- Reworded part B Q9
- Clarified spacing between words in part C output

Additions:

- Added C++ as one of the allowed languages for part C

### v1.2 (14/03/19)

Additions:

- Changed the URL for the abc example to include a trailing slash
- Redirection now requires the client/server info to be re-printed

### v1.3 (21/03/19)

Fixes:

- Fixed inconsistency between redirection outputs