# COMP3331 LAB 2
# Abanob Tawfik
# Z5075490

## Contents

# Exercise 3.1: Using Wireshark to understand basic HTTP request/response messages

## What is the status code and phrase returned from the server to the client browser?

The status code returned was 200, and the phrase associated with status code 200 is OK. This is highlighted in Figure 1.

| | | | | | |
|---|---|---|---|---|---|
| 10 | 4.694850 | 192.168.1.102 | 128.119.245.12 | HTTP | 555 GET /ethereal-labs/lab2-1.html HTTP/1.1 |
| 12 | 4.718993 | 128.119.245.12 | 192.168.1.102 | HTTP | 439 HTTP/1.1 200 OK (text/html) |

*Figure 1 Status Code and phrase returned*

## When was the HTML file that the browser is retrieving last modified at the server? Does the response also contain a DATE header? How are these two fields different?

The html file was last modified on **Tuesday, 23rd of September 2003 05:29:00**. The response does contain a DATE header and the date was **Tuesday, 23rd of September 2003 05:29:50**. These two fields are different as the last-modified is the date in which the HTML file was last changed, and the DATE header is the date the packet was created. This is highlighted in Figure 2.

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Tue, 23 Sep 2003 05:29:50 GMT\r\n
  Server: Apache/2.0.40 (Red Hat Linux)\r\n
  Last-Modified: Tue, 23 Sep 2003 05:29:00 GMT\r\n
```

*Figure 2 Dates for last modified and date header*

## Is the connection established between the browser and the server persistent or non-persistent? How can you infer this?

The connection established between the browser and the server is persistent as the connection header is Keep-Alive illustrated in Figure 3. However, there is a timeout of 10 seconds when there is no activity between the client and the server, and a maximum of 100 requests before the connection is forcefully closed.

```
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
```

*Figure 3 Connection Header*

## How many bytes of content are being returned to the browser?

The number of bytes for the payload was 73 bytes as illustrated in Figure 4.

```
File Data: 73 bytes
```

*Figure 4 payload size*

## What is the data contained inside the HTTP response packet?

The data contained inside the HTTP response packet was a html page, that had the message congratulating the user that they've downloaded the file for lab2-1.html. This is displayed in Figure 5.

```
<html>\n
Congratulations.  You've downloaded the file lab2-1.html!\n
</html>\n
```

*Figure 5 the HTTP response payload content*

## Exercise 3.2: Using Wireshark to understand basic HTTP request/response messages for our own machines.

This was performed on the website [http://cgi.cse.unsw.edu.au/~cs2041/18s2/index.html](http://cgi.cse.unsw.edu.au/~cs2041/18s2/index.html) displayed in Figure 6.

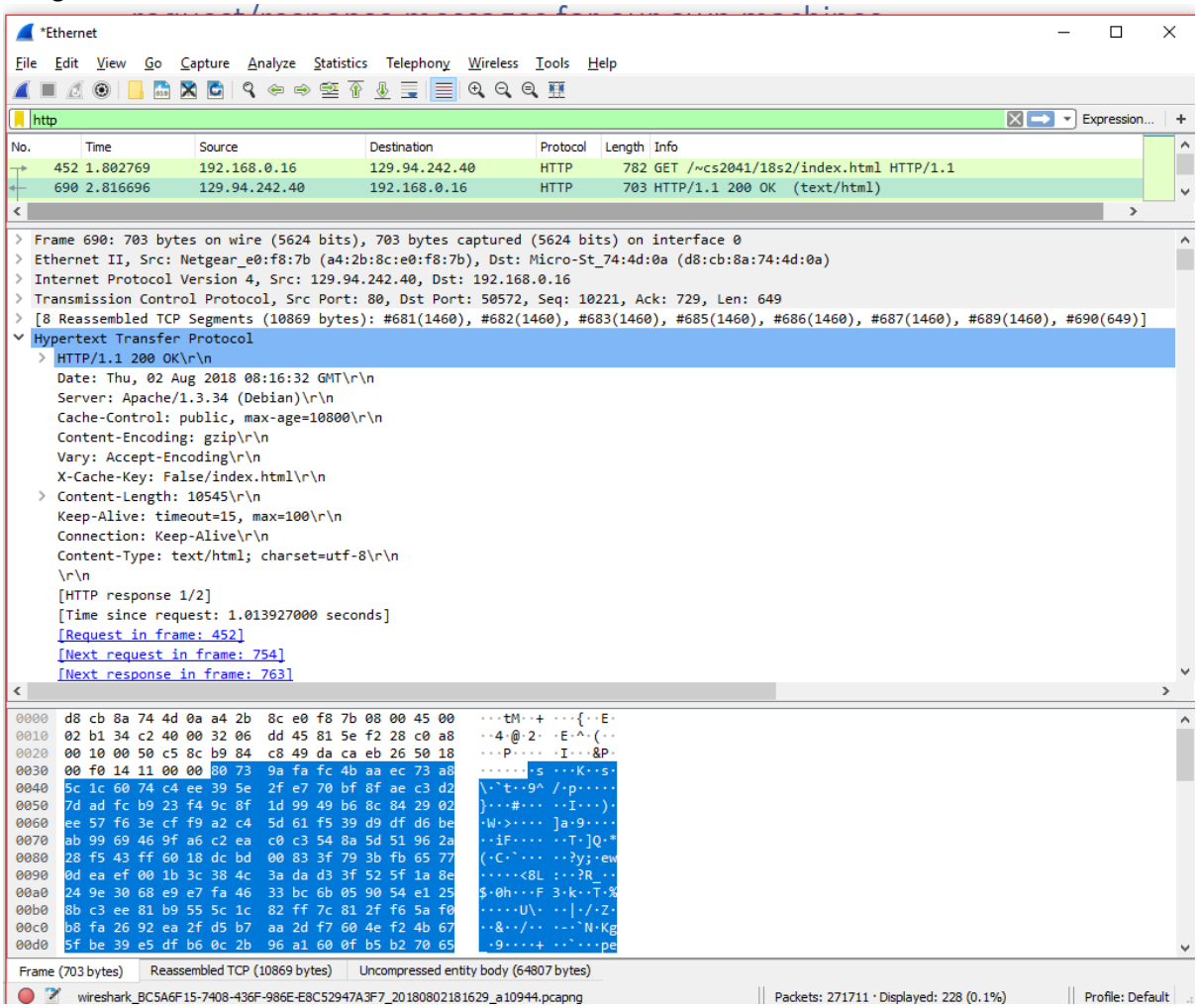


*Figure 6 wireshark on packets for cs2041 website*

### What is the status code and phrase returned from the server to the client browser?

The status code returned was 200, and the phrase associated with status code 200 is OK. This is highlighted in Figure 7.

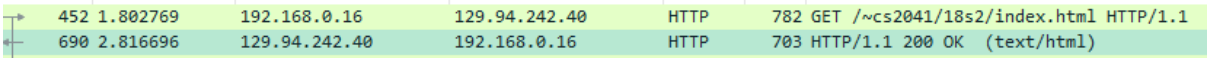


*Figure 7 Status Code and phrase returned*

## When was the HTML file that the browser is retrieving last modified at the server? Does the response also contain a DATE header? How are these two fields different?

The html file did not contain a last modified date. The response does contain a DATE header and the date was **Tuesday, 2nd of August 2018 08:16:32**. These two fields are different as the last-modified is the date in which the HTML file was last changed, and the DATE header is the date the packet was created. This is highlighted in Figure 8.

```
> HTTP/1.1 200 OK\r\n
  Date: Thu, 02 Aug 2018 08:16:32 GMT\r\n
  Server: Apache/1.3.34 (Debian)\r\n
  Cache-Control: public, max-age=10800\r\n
  Content-Encoding: gzip\r\n
  Vary: Accept-Encoding\r\n
  X-Cache-Key: False/index.html\r\n
> Content-Length: 10545\r\n
  Keep-Alive: timeout=15, max=100\r\n
  Connection: Keep-Alive\r\n
  Content-Type: text/html; charset=utf-8\r\n
  \r\n
  [HTTP response 1/2]
```

*Figure 8 Date header*

## Is the connection established between the browser and the server persistent or non-persistent? How can you infer this?

The connection established between the browser and the server is persistent as the connection header is Keep-Alive illustrated in Figure 9. However, there is a timeout of 15 seconds when there is no activity between the client and the server, and a maximum of 100 requests before the connection is forcefully closed.

```
Keep-Alive: timeout=15, max=100\r\n
Connection: Keep-Alive\r\n
```

*Figure 9 Connection Header*

## How many bytes of content are being returned to the browser?

The number of bytes for the payload was 64807 bytes as illustrated in Figure 10.

```
File Data: 64807 bytes
```

*Figure 10 payload size*

## What is the data contained inside the HTTP response packet?

The data contained inside the HTTP response packet was a html page, that contained all the contents to the comp2041 18s2 website, displayed in Figure 11.

```
Line-based text data: text/html (2729 lines)
    <!doctype html>\n
    <html lang="en">\n
    <head>\n
    <meta charset="utf-8" />\n
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />\n
    \n
    <title>\n
    \n
    COMP2041 Software Construction\n
    \n
    </title>\n
    \n
    \n
    \n
    \n
    <!-- <link href="/~cs2041/18s2/flask.cgi/static/bootstrap.min.css" rel="stylesheet"> -->\n
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous"
```

*Figure 11 the HTTP response payload content*

# Exercise 4: Using Wireshark to understand the HTTP CONDITIONAL GET/response interaction

## Inspect the contents of the first HTTP GET request from the browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?

The first HTTP GET request did not contain an IF-MODIFIED-SINCE line. Its content is shown below in Figure 12.

```
Hypertext Transfer Protocol
  GET /ethereal-labs/lab2-2.html HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /ethereal-labs/lab2-2.html HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /ethereal-labs/lab2-2.html
    Request Version: HTTP/1.1
  Host: gaia.cs.umass.edu\r\n
  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.2) Gecko/20021120 Netscape/7.01\r\n
  Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1\r\n
  Accept-Language: en-us, en;q=0.50\r\n
  Accept-Encoding: gzip, deflate, compress;q=0.9\r\n
  Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66\r\n
  Keep-Alive: 300\r\n
  Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://gaia.cs.umass.edu/ethereal-labs/lab2-2.html]
  [HTTP request 1/2]
  [Response in frame: 10]
  [Next request in frame: 14]
```

*Figure 12 content of the HTTP GET request*

## Does the response indicate the last time that the requested file was modified?

The response contained the last time the requested file was modified displayed in Figure 13. This was on **Tuesday, 23rd of September 2003 5:35:00.**

```
Last-Modified: Tue, 23 Sep 2003 05:35:00 GMT\r\n
```

*Figure 13 Last modified time*

## Now inspect the contents of the second HTTP GET request from the browser to the server. Do you see an "IF-MODIFIED-SINCE:" and "IF-NONE-MATCH" lines in the HTTP GET? If so, what information is contained in these header lines?

The HTTP GET request contained both header lines displayed in Figure 14. The information contained in the If-Modified-Since is a conditional check that will only return status code 200 ok, if the resource was modified after the date specified. The information contained in the IF-NONE-MATCH is a conditional check that will only return status code 200 ok, if the resource does not contain the matching Etag given.

```
If-Modified-Since: Tue, 23 Sep 2003 05:35:00 GMT\r\r
If-None-Match: "1bfef-173-8f4ae900"\r\n
```

*Figure 14 if-modified lines*

## What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

The HTTP status code and phrase returned was 304 not modified displayed in Figure 15. The server did not return the contents of the file, it instead returned the status code and phrase. This is because the Etag in the resource matched the IF-NONE-MATCH, which returns the 304-status code.

```
Hypertext Transfer Protocol
  HTTP/1.1 304 Not Modified\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 304 Not Modified\r\n]
    Response Version: HTTP/1.1
    Status Code: 304
    [Status Code Description: Not Modified]
    Response Phrase: Not Modified
  Date: Tue, 23 Sep 2003 05:35:53 GMT\r\n
  Server: Apache/2.0.40 (Red Hat Linux)\r\n
  Connection: Keep-Alive\r\n
  Keep-Alive: timeout=10, max=99\r\n
  ETag: "1bfef-173-8f4ae900"\r\n
  \r\n
  [HTTP response 2/2]
  [Time since request: 0.022826000 seconds]
  [Prev request in frame: 8]
  [Prev response in frame: 10]
  [Request in frame: 14]
```

*Figure 15 response from the HTTP request*

## What is the value of the Etag field in the 2nd response message and how it is used? Has this value changed since the 1st response message was received?

The value of the Etag field in the second response message is **1bfef-173-8f4ae900** underlined in Figure 15**.** The Etag field is used for web cache validation, this is used to compare whether two different representations of a response are the same. The Etag field is used in conjunction with the IF-NONE-MATCH field of a request to check if the cached version of the resource is still valid. This is done by comparing the current Etag value for the website and the cached Etag value stored in the browser. The Etag value hasn't changed since the first response was received displayed in Figure 16, as the etag value for the first response was **1bfef-173-8f4ae900.** This means that the resource has not been modified, thus giving a 304 response in the second response.

```
ETag: "1bfef-173-8f4ae900"\r\n
```

*Figure 16 First Etag value*

# Exercise 5: Ping Client

## Implementing Ping Client

The code for Ping Client is supplied in the file PingClient.java as well as in the next page of the report. To run the following demonstration, run two separate terminals.
On the first terminal type the following command:
Javac PingServer.java
Java PingServer 8888 ← port number
On the second terminal type the following command:
Javac PingClient.java
Java PingClient 127.0.0.1 8888 ← host + port number respectively
When the following was completed this will send 10 pings separated by 1 second, and the following output is shown in Figure 17.

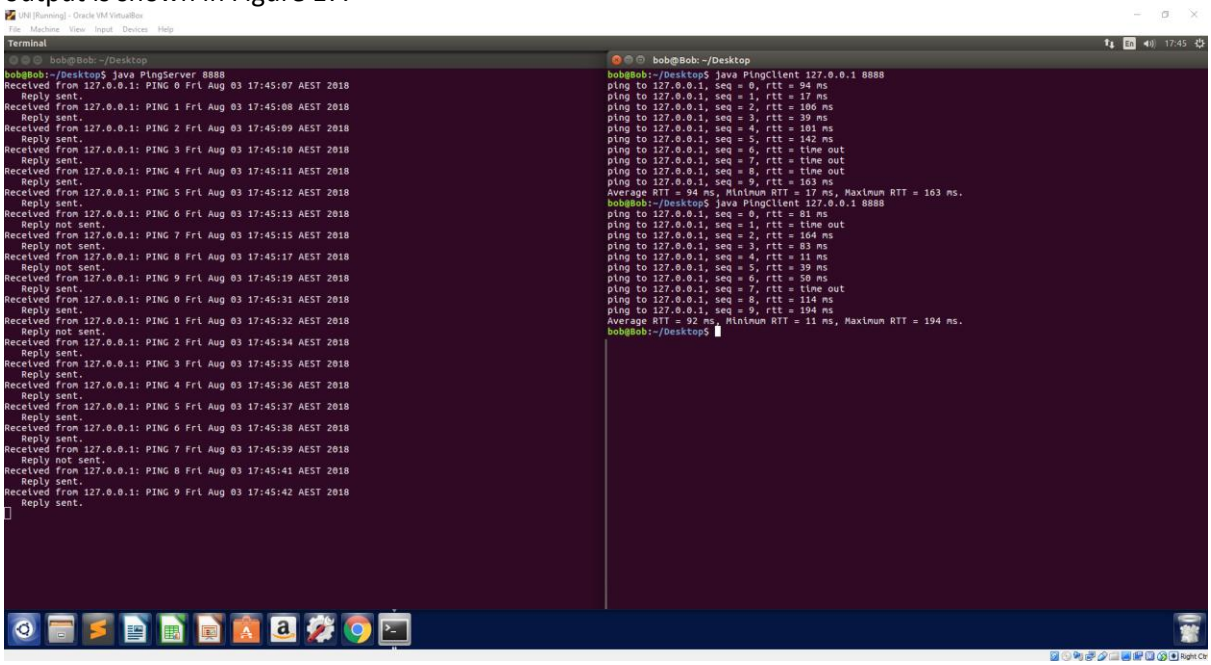


*Figure 17 Output from PingClient and PingServer*

Please note that the 1 second delay between each ping was implemented.

```java
import java.net.*;
import java.util.*;

public class PingClient {
    private static final int REQUEST_TIME_OUT = 1000;  // milliseconds

    public static void main(String[] args) throws Exception {
        // Get command line argument.
        if (args.length != 2) {
            System.out.println("Required arguments: (Host) + (Port)");
            return;
        }
        //the server we are pinging
        InetAddress server = InetAddress.getByName(args[0]);
        // Create a datagram socket for receiving and sending UDP packets
        // through the port specified on the command line.
        DatagramSocket port = new DatagramSocket();
        port.setSoTimeout(REQUEST_TIME_OUT);
        //initalising values for end analysis
        long averageRTT = 0;
        long maximumRTT = 0;
        long minimumRTT = 5000;
        int validRTT = 0;
        for (int i = 0; i < 10; i++) {
            //get the current time for  time-stamp-start
            Date time = new Date();
            String s = time.toString();
            long timeStampStart = time.getTime();
            //created the message for the ping
            String message = "PING " + i + " " + s + "\r\n";
            //this is the packet used in the ping request containing the message.
            DatagramPacket pingRequest = new DatagramPacket(message.getBytes(),
message.length(), server, Integer.parseInt(args[1]));
            //now we want to send the ping request to the port
            port.send(pingRequest);
            //now we want to receive the response from the server.
            DatagramPacket serverReply = new DatagramPacket(new byte[1024], 1024);
            //to setup for timeout we want to use a try catch, where we check if there is a
response within 1s, else catch the exception
            try {
                //if there is a response from the server within 1s since timeout at 1s will
trigger the exception
                port.receive(serverReply);
                //use the current time after message received for time-stamp-end
                Date time2 = new Date();
                //calculate the RTT from this
                long timeStampEnd = time2.getTime();
                //the round trip time is the final time subtracted from the initial time
                long RTT = timeStampEnd - timeStampStart;
                //display the ping message
                System.out.println("ping to " + args[0] + ", seq = " + i + ", rtt = " + RTT +
" ms");
                averageRTT += RTT;
                validRTT++;
                //if the RTT from this iteration is larger than the maximum update maximum
                if (RTT > maximumRTT)
                    maximumRTT = RTT;
                //if the RTT from this iteration is smaller than the minimum update minimum
                if (RTT < minimumRTT)
                    minimumRTT = RTT;
                //program waits for 1 second - the amount of time for the RTT to simulate 1
second delay
                Thread.sleep(1000 - RTT);
            } catch (Exception e) {
                //time out exception
                System.out.println("ping to " + args[0] + ", seq = " + i + ", rtt = time
out");
                //1 second delay between the ping
                Thread.sleep(1000);
            }
        }
        //print the average minimum and maximum RTT
        System.out.println("Average RTT = " + (averageRTT / validRTT) + " ms" + ", Minimum RTT
= "
                + minimumRTT + " ms" + ", Maximum RTT = " + maximumRTT + " ms.");
    }
}
```