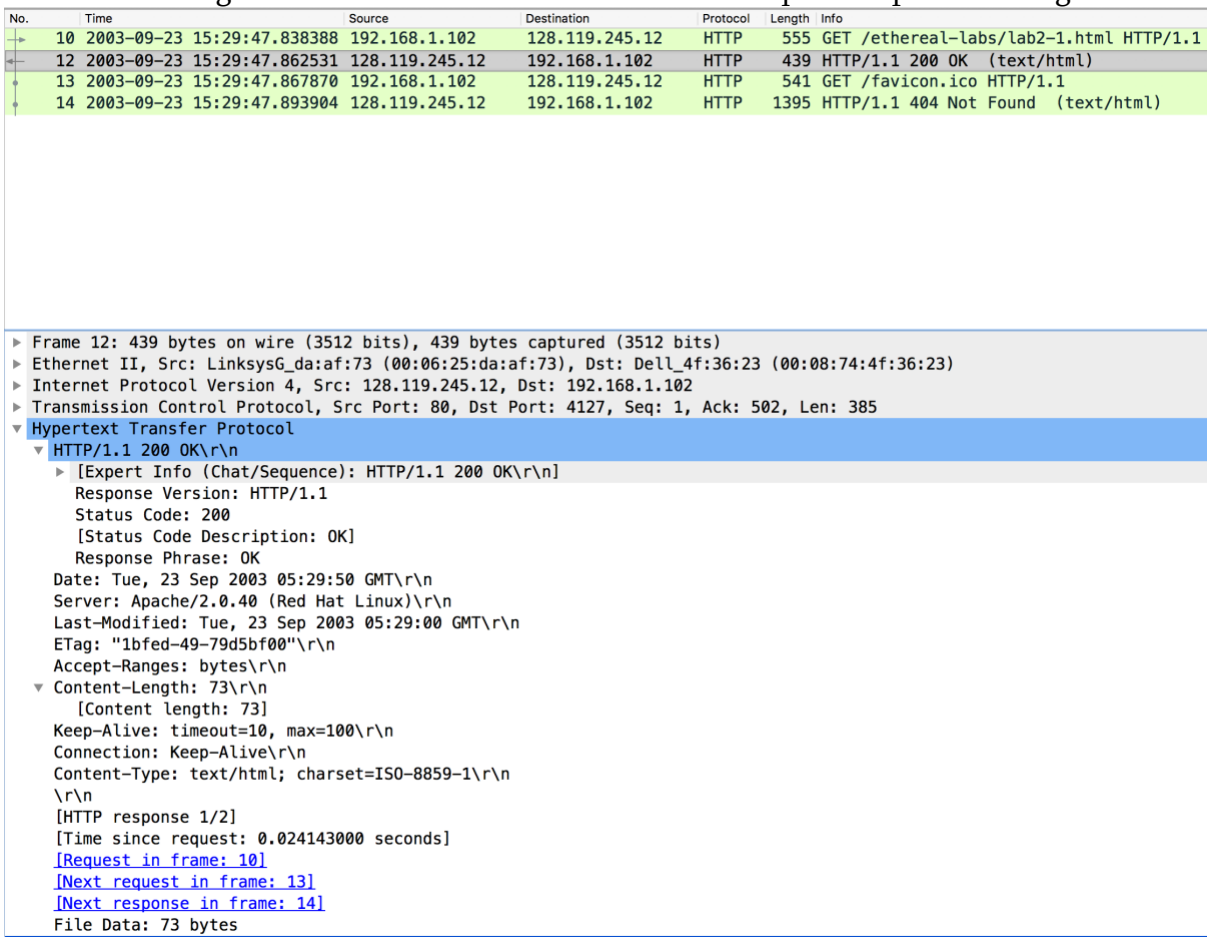
Exercise 3: Using Wireshark to understand basic HTTP request/response messages



```
No.  Time                       Source          Destination     Protocol Length Info
  10 2003-09-23 15:29:47.838388 192.168.1.102   128.119.245.12  HTTP     555 GET /ethereal-labs/lab2-1.html HTTP/1.1
  12 2003-09-23 15:29:47.862531 128.119.245.12  192.168.1.102   HTTP     439 HTTP/1.1 200 OK  (text/html)
  13 2003-09-23 15:29:47.867870 192.168.1.102   128.119.245.12  HTTP     541 GET /favicon.ico HTTP/1.1
  14 2003-09-23 15:29:47.893904 128.119.245.12  192.168.1.102   HTTP    1395 HTTP/1.1 404 Not Found  (text/html)

▶ Frame 12: 439 bytes on wire (3512 bits), 439 bytes captured (3512 bits)
▶ Ethernet II, Src: LinksysG_da:af:73 (00:06:25:da:af:73), Dst: Dell_4f:36:23 (00:08:74:4f:36:23)
▶ Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.1.102
▶ Transmission Control Protocol, Src Port: 80, Dst Port: 4127, Seq: 1, Ack: 502, Len: 385
▼ Hypertext Transfer Protocol
  ▼ HTTP/1.1 200 OK\r\n
    ▶ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
    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
    ETag: "1bfed-49-79d5bf00"\r\n
    Accept-Ranges: bytes\r\n
  ▼ Content-Length: 73\r\n
      [Content length: 73]
    Keep-Alive: timeout=10, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=ISO-8859-1\r\n
    \r\n
    [HTTP response 1/2]
    [Time since request: 0.024143000 seconds]
    [Request in frame: 10]
    [Next request in frame: 13]
    [Next response in frame: 14]
    File Data: 73 bytes
```

Question 1: What is the status code and phrase returned from the server to the client browser?

```
Status Code: 200
[Status Code Description: OK]
Response Phrase: OK
```

The status code and phrase are 200 OK

Question 2: 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 last modified time is :

```
Last-Modified: Tue, 23 Sep 2003 05:29:00 GMT\r\n
```

The response contains a DATE header, which indicates the time when the HTTP response message was actually generated by the server. Since these two fields are showing the same time in the current response, it is likely that the server is simply setting the file's last-modified time to the time when the HTTP response message is created.

Question 3: Is the connection established between the browser and the server persistent or non-persistent? How can you infer this?

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

Both the client and server are using persistent HTTP connections.

we can infer this by looking its connection, header in both the request and response.

Question 4: How many bytes of content are being returned to the browser?

73bytes

Question 5: What is the data contained inside the HTTP response packet?

The file being returned the browser is an HTML file

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

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

Question 1: 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?

Answer: No

Question 2: Does the response indicate the last time that the requested file was modified?

Yes,it is and the time is:

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

Question 3: 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?

Yes. The information following is:

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

which is the date of the last modification of the file from the previous get request.

Question 4: 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 status code and phrase returned from the server is HTTP/1.1 304 Not Modified. The server didn't return the contents of the file since the browser loaded it from its cache.

Question 5: 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 is :

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

It is used to match the value of previous etag value, if the ETag values do not match, meaning the resource has likely changed, then a full response including the resource's content is returned,and if it match,then the server may send back a very short response with a HTTP 304 Not Modified status. The 304 status tells the client that its cached version is still good and that it should use that.

The value didn't change.

(*) Exercise 5: Ping Client

Server

Port = 5000

```
us579:Desktop us579$ java PingServer 5000
Received from 127.0.0.1: PING 1 Tue Aug  7 00:11:21 2018
   Reply sent.
Received from 127.0.0.1: PING 2 Tue Aug  7 00:11:21 2018
   Reply sent.
Received from 127.0.0.1: PING 3 Tue Aug  7 00:11:21 2018
   Reply sent.
Received from 127.0.0.1: PING 4 Tue Aug  7 00:11:21 2018
   Reply sent.
Received from 127.0.0.1: PING 5 Tue Aug  7 00:11:21 2018
   Reply sent.
Received from 127.0.0.1: PING 6 Tue Aug  7 00:11:21 2018
   Reply sent.
Received from 127.0.0.1: PING 7 Tue Aug  7 00:11:22 2018
   Reply sent.
Received from 127.0.0.1: PING 8 Tue Aug  7 00:11:22 2018
   Reply sent.
Received from 127.0.0.1: PING 9 Tue Aug  7 00:11:22 2018
   Reply sent.
Received from 127.0.0.1: PING 10 Tue Aug  7 00:11:22 2018
   Reply not sent.
```

Client

Host = local host 127.0.0.1

Port = 5000

```
us579:Desktop us579$ python3 Pingc.py 127.0.0.1 5000
ping to 127.0.0.1, seq = 1, rtt = 144.56 ms
ping to 127.0.0.1, seq = 2, rtt = 162.04 ms
ping to 127.0.0.1, seq = 3, rtt = 94.39 ms
ping to 127.0.0.1, seq = 4, rtt = 127.86 ms
ping to 127.0.0.1, seq = 5, rtt = 89.22 ms
ping to 127.0.0.1, seq = 6, rtt = 42.46 ms
ping to 127.0.0.1, seq = 7, rtt = 115.41 ms
ping to 127.0.0.1, seq = 8, rtt = 144.82 ms
ping to 127.0.0.1, seq = 9, rtt = 80.15 ms
ping to 127.0.0.1, seq = 10, rtt = timeout
10 packets transmitted, 9 received, 10.00% packet loss
min = 42.46 ms, max = 162.04 ms, avg = 111.21 ms
us579:Desktop us579$
```

For the client, I use python and the code is below.

```python
from socket import *
import sys
import time

#The function to caculate the min,max,avg time and the successful rate of transmition.
def detials(rtt,sent,received):
    num = len(rtt)
    min_time = min(rtt) if num > 0 else 0
    max_time = max(rtt) if num > 0 else 0
    avg_time = sum(rtt) / num if num > 0 else 0
    print("{} packets transmitted, {} received, {:.2f}% packet loss".format(sent,received, 100 -
        ((received/sent)*100)))
    print('min = {:.2f} ms, max = {:.2f} ms, avg = {:.2f} ms'.format(min_time,max_time,avg_time))
argv = sys.argv
if len(argv) != 3:
    #if arguments from the commend line do not follow the format below, Print message and exit
    print('Incorrect input,format: Python3 <host name/IP> <Port number>')
    sys.exit(1)
#the local host from commend line arguments
host = argv[1]
#port number
port = int(argv[2])
#create an UDP client socket
client_socket = socket(AF_INET,SOCK_DGRAM)
client_socket.settimeout(1)
seq_num = 0
#the list that store rtt time
rtt_list = []
received = 0
#loop for pinging 10 times
while seq_num < 10:
    seq_num += 1
    #data send to the server
    data ='PING {} {}\r\n'.format(seq_num,time.asctime(time.localtime()))
    try:
        Rtt_start = time.time()
        client_socket.sendto(data.encode(),(host,port))
        message,address = client_socket.recvfrom(1024)
        Rtt_end = time.time()
        received += 1
        rtt_time = (Rtt_end - Rtt_start)*1000
        rtt_list.append(rtt_time)
        print('ping to {}, seq = {}, rtt = {:.2f} ms'.format(address[0],seq_num,rtt_time))
    except:
        print('ping to {}, seq = {}, rtt = timeout'.format(argv[1],seq_num))
        continue
detials(rtt_list,seq_num,received)
#close the client socket
client_socket.close()
```