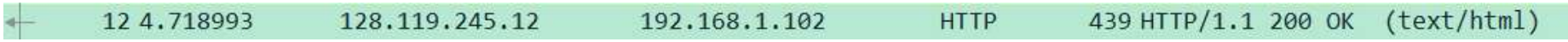
Exercise 3: Using Wireshark to understand basic HTTP request/response messages

- Step 1: Start Wireshark by typing wireshark at the command prompt.
- Step 2: Load the trace file http-ethereal-trace-1 by using the File pull down menu, choosing Open and selecting the appropriate trace file. This trace file captures a simple request/response interaction between a browser and a web server.
- Step 3: You will see a large number of packets in the packet-listing window. Since we are currently only interested in HTTP we will filter out all the other packets by typing “http” in lower-case in the Filter field and press Enter. You should now see only HTTP packets in the packet-listing window.
- Step 4: Select the first HTTP message in the packet-listing window and observe the data in the packetheader detail window. Recall that since each HTTP message was carried inside a TCP segment, which was carried inside an IP datagram, which was carried within an Ethernet frame, Wireshark displays the Frame, Ethernet, IP, and TCP packet information as well. We want to minimize the amount of non-HTTP data displayed (we’re interested in HTTP here, and will be investigating these other protocols is later labs), so make sure the boxes at the far left of the Frame, Ethernet, IP and TCP information have a right-pointing arrowhead (which means there is hidden, undisplayed information), and the HTTP line has a down-pointing arrowhead (which means that all information about the HTTP message is displayed).

Question 1: What is the status code and phrase returned from the server to the client browser?
Answer:
The status code is 200 and its phrase is 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?
Answer:
The last modified time is 23 Sep 2003 05:29:00.
There is a DATE header contained by response.
DATE contains the date and time at which the message was originated.
Last-Modified indicates the date and time at which the original server
believe the variant was last modified. It is used to reduce the load time of the page when the last-modified attribution in the user’s request

is same as last-modified attribution on the server.

```
Last-Modified: Tue, 23 Sep 2003 05:29:00 GMT\r\n
Date: Tue, 23 Sep 2003 05:29:50 GMT\r\n
```

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

Answer:

The connection established persistently. Because The Connection general header controls whether or not the network connection stays open after the current transaction finishes. If the value sent is keep-alive, the connection is persistent and not closed, allowing for subsequent requests to the same server to be done. As the picture blow,

```
Connection: Keep-Alive\r\n
```

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

Answer:

73 bytes

```
Content-Length: 73\r\n
File Data: 73 bytes
```

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

Answer:

The data contained by HTTP response packet is text/html file. The content is as follow,

```
Line-based text data: text/html (3 lines)
    <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

- Step 1: Start Wireshark by typing wireshark at the command prompt.

- Step 2: Load the trace file http-ethereal-trace-2 by using the File pull down menu, choosing Open and selecting the appropriate trace file. This trace file captures a request response between a client browser and web server where the client requests the same file from the server within a span of a few seconds.
- Step 3: Filter out all the non-HTTP packets and focus on the HTTP header information in the packet header detail window.

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?

Answer:

Yes,

```
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?

Answer:

Yes, the If-Modified-Since request HTTP header makes the request conditional is that the server will send back the requested resource, with a 200 status, only if it has been last modified after the given date. If the request has not been modified since, the response will be a 304 without any body.

“IF-NONE-MATCH” is a hash value return from server in the first response time. In 2nd HTTP request from browser to server, if this value equal to Etags which calculate by files in the server, return 200 OK, else 304 Not Modified. This can help reduce network load.

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

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.

Answer:

The status code is 304 and phrase is Not Modified. The server don't explicitly return the contents of the file because file is Not modified. As we talks above, if Not Modified, then the content of file will load from brower cache.

```
> HTTP/1.1 304 Not Modified\r\n
```

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 1 stresponse message was received?

Answer:

The value of Etag field as follow, it is 1bfef-173-8f4ae900. It is similar as "IF-MODIFIED-SINCE", in order to be more efficient, and saves bandwidth, as a web server does not need to send a full response if the content has not changed. On the other side, if the content has changed, etags are useful to help prevent simultaneous updates of a resource from overwriting each other ("mid-air collisions").

This value don't changed since the 1 st response message was received. Like blow,

Etag 2nd response message: `ETag: "1bfef-173-8f4ae900"\r\n`

Etag 1nd response message: `ETag: "1bfef-173-8f4ae900"\r\n`

Exercise 5: Ping Client

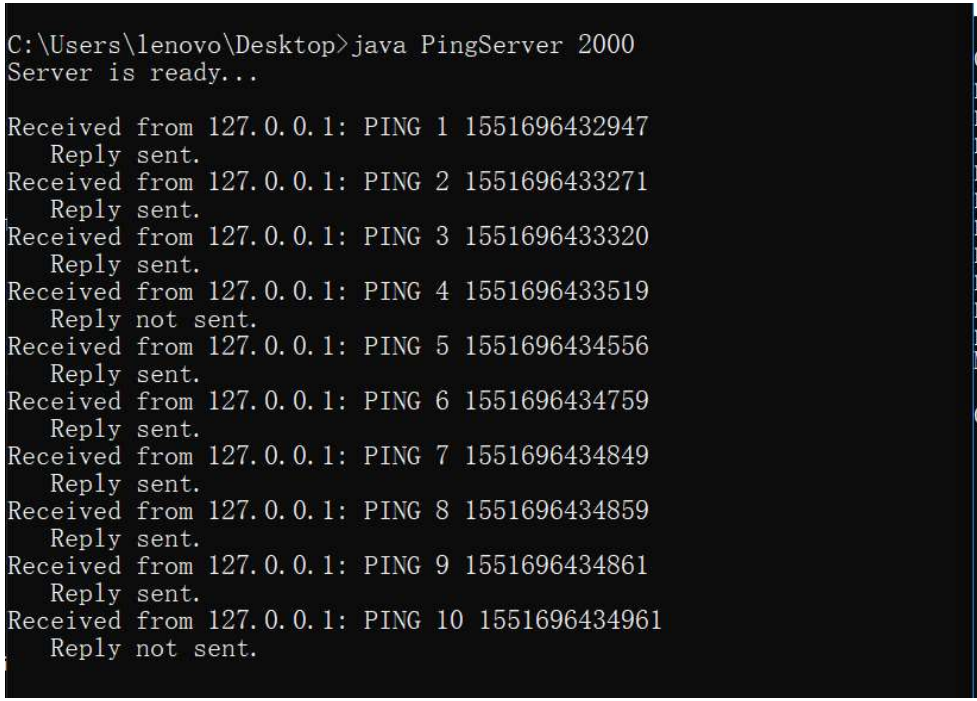
Here is my code in java:

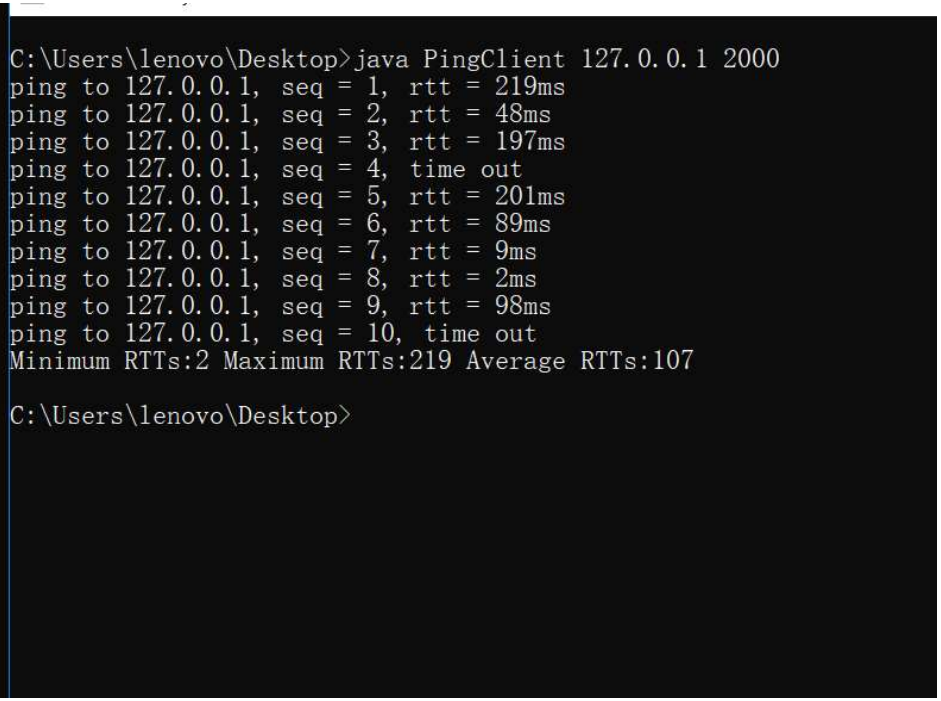
```
import java.io.*;
import java.net.*;
import java.util.*;

public class PingClient {
    static private final int ping_time = 10;
    static private final int TIMEOUT = 1000;

    public static void main(String[] args) {
        String server = args[0];
```

```
int port = Integer.parseInt(args[1]);
long t1 = 0, t2 = 0;
long[] RTTs = new long[10];          // list for storing rtts
int RTT_succ = 0;

SocketAddress local_addr = new InetSocketAddress("localhost", 3231);
try {
    DatagramSocket sender = new DatagramSocket(local_addr);
    byte[] buf1 = new byte[1024];
    DatagramPacket dp_receive = new DatagramPacket(buf1, 1024);
    sender.setSoTimeout(TIMEOUT);                // set timeout
    int count = 0;

    while(count < ping_time){
        count++;
        long timeStamp = System.currentTimeMillis();
        byte[] buf = ("PING " + count + " " + timeStamp +"\r\n").getBytes();
        SocketAddress receive_addr = new InetSocketAddress(server, port);
        DatagramPacket data = new DatagramPacket(buf, buf.length, receive_addr);
        try {
            sender.send(data);
            t1 = System.currentTimeMillis();
        } catch (Exception e) {
            e.printStackTrace();
        }

        // try to revice response from server.
        while (true) {
            try {
                sender.receive(dp_receive);
                t2 = System.currentTimeMillis();    // already receive message
                RTTs[RTT_succ] = t2 - t1;
                RTT_succ ++;
                System.out.println("ping to "+ server + ", seq = "+ count + ", rtt = " + (t2 - t1) + "ms");
```

```
                    break;
                }
                catch (InterruptedIOException e){       // Time out
                    System.out.println("ping to "+ server + ", seq = "+ count + ", time out");
                    break;
                }
                catch (IOException e1){                 // can not receive any message
                    continue;
                }
            }
        }
        sender.close();
    } catch (SocketException e) {
        e.printStackTrace();
    }

    if (RTT_succ != 0) {                                // not all requests are time out
        long max = RTTs[0], min = RTTs[0], avg = 0;

        for (int i = 0; i < RTT_succ; i ++) {
            if (RTTs[i] > max)
                max = RTTs[i];
            if (RTTs[i] < min)
                min = RTTs[i];
            avg += RTTs[i];
        }
        System.out.println("Minimum RTTs:" + min + " Maximum RTTs:" + max + " Average RTTs:" + avg / RTT_succ);
    }
  }
}
```

This is the result:


C:\Users\lenovo\Desktop>java PingServer 2000
Server is ready...
Received from 127.0.0.1: PING 1 1551696432947
Reply sent.
Received from 127.0.0.1: PING 2 1551696433271
Reply sent.
Received from 127.0.0.1: PING 3 1551696433320
Reply sent.
Received from 127.0.0.1: PING 4 1551696433519
Reply not sent.
Received from 127.0.0.1: PING 5 1551696434556
Reply sent.
Received from 127.0.0.1: PING 6 1551696434759
Reply sent.
Received from 127.0.0.1: PING 7 1551696434849
Reply sent.
Received from 127.0.0.1: PING 8 1551696434859
Reply sent.
Received from 127.0.0.1: PING 9 1551696434861
Reply sent.
Received from 127.0.0.1: PING 10 1551696434961
Reply not sent.


C:\Users\lenovo\Desktop>java PingClient 127.0.0.1 2000
ping to 127.0.0.1, seq = 1, rtt = 219ms
ping to 127.0.0.1, seq = 2, rtt = 48ms
ping to 127.0.0.1, seq = 3, rtt = 197ms
ping to 127.0.0.1, seq = 4, time out
ping to 127.0.0.1, seq = 5, rtt = 201ms
ping to 127.0.0.1, seq = 6, rtt = 89ms
ping to 127.0.0.1, seq = 7, rtt = 9ms
ping to 127.0.0.1, seq = 8, rtt = 2ms
ping to 127.0.0.1, seq = 9, rtt = 98ms
ping to 127.0.0.1, seq = 10, time out
Minimum RTTs:2 Maximum RTTs:219 Average RTTs:107
C:\Users\lenovo\Desktop>