**HTTP**

1. Both my browser and the server are running HTTP 1.1.
2. English (US) and Russian.
3. The IP Address of my computer is **10.0.0.5**, while the IP Address of the link is **128.119.245.12**.
4. My browser receives the status code **200**.
5. The HTML file that I received was last modified on Sat, 26 Sep 2015 at 18:30:48 GMT, 11 seconds before I received that data.
6. **128** bytes are being caught by my browser.
7. No, the raw data matches up with what is shown in the packet-listing window.
8. No, I do not see an “IF-MODIFIED-SINCE” line.
9. Yes, see image “HTTP\_P2” for reference. It returns the original content of the webpage.
10. Yes. I**f-Modified-Since: Sat, 26 Sep 2015 05:59:01 GMT\r\n**
11. The code is HTTP/1.1 304 Not Modified. This is much shorter than the previous return, which displayed all the contents of the web page
12. Only one HTTP GET response was sent. Packet 93 contains the status code and phrase that is associated with the response to the GET message.
13. Packet 100 contains the status code and phrase associated with the response.
14. HTTP/1.1 200 OK
15. 4 TCP segments were needed.
16. 4 HTTP GET messages were sent. They were sent to **128.119.245.12**, **165.193.140.14**, and **128.119.240.90**.
17. The images seem to be downloaded in parallel. Packet 273 and 294 sent requests for the two images, and packet 283 and 410 were responses that the images were received. Thus, they were downloaded in parallel.
18. HTTP/1.1 401 Unauthorized
19. **Authorization: Basic d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms**

**DNS**

1. The IP Address of the server is **69.172.200.121**.
2. The authoritative DNS for Cambridge University was **ipreg.csi.cam.ac.uk**.
3. The address is **2001:4998:58:2201::50**.
4. They are sent over UDP.
5. Source Port is **65291**, and the Destination Port is **53**.
6. The IP Address that the DNS message is sent to is **75.75.75.75**. This is the same as the IP address of my local DNS server.
7. It is a type A, and has no answers.
8. Only one answer is provided, and this answer contains the address of the website that it was queried for.
9. Yes, the destination IP of the SYN packet corresponds to the address that is provided by the DNS response, which is **4.31.198.44**.
10. Yes, the host issues new DNS queries for every image.
11. The source port is **61884**, and the destination port is **53**.
12. It is sent to IPV6 destination 2001:558:feed::1 – this is the same as the source port of the response.
13. It is type AAAA, and has no responses.
14. The response provides 4 answers. The first two answers have a CNAME (Canonical Name for an alias), while the second two answers contain the IPv6 Addresses and AAAA Addresses for the second CNAME.
15. See ‘DNS\_P2\_14’ PNG file in the attached .zip file.
16. The IP Address that the DNS message is sent to is **75.75.75.75**. This is the same as the IP address of my local DNS server.
17. It is a type A query and has no responses.
18. The response provides [www.mit.edu](http://www.mit.edu), [www.mit.edu.edgekey.net](http://www.mit.edu.edgekey.net), and e9566.dscb.akamaiedge.net. The response does not provide the IP Addresses of the first two answers.
19. See ‘DNS\_P2\_14’ and ‘DNS\_P2\_16’ PNG files
20. The query is sent to 18.72.0.3, which is not the same as my local DNS server. This IP address instead corresponds to [www.aiit.or.kr](http://www.aiit.or.kr). Bitsy.mit.edu no longer exists.
21. The Type is PTR (Domain name pointer) and contains no answers.
22. There were no response messages.
23. See ‘DNS\_P2\_20’ PNG file.

**Socket Programming and TCP**

1. Skipped b/c of professor instruction.
2. The udpClient file does not have a clientSocket.connect piece of code – instead, it has a ‘sendto’ line that accepts an additional message argument (along with the serverName and serverPort arguments), while the tcpClient file instead has a ‘connect’ line that accepts the serverName and serverPort, while there is a separate ‘send’ line that sends the message. The UDP code does this all in one line, while tcpClient has to break it up.

Similarly, the udpServer and tcpServer files have differences. The tcpServer listens for a message, while the udpServer does not. Also, the socket argument in the tcpServer file accepts SSOCK\_STREAM rather than udpServer’s SOCK\_DGRAM. In the while loop, while the udpServer code accepts the message and client address, while also using sendto, the tcpServer must accept the ConnectionSocket and addr every time it exists, while also sending the modified message and then closing the connection socket.

1. The extra socket is created in order for the server to accept new data if it comes in. The server accepts the data that it receives, and then closes the data so that no other data can come in from that stream.
2. I receive a socket.error that states that ‘Only one usage of each socket address (protocol/network address/port) is normally permitted. I could change the code to allow for the server to listen for multiple clients – if it listens for two clients, it will prevent the clients from trying to access the same port and network address.
3. The entire file is not being transmitted to the server; it is being broken up into chunks and then sent to the server.