Problem 1

Consider the following string of ASCII characters that were captured by *Wireshark* when the browser sent an HTTP GET message (i.e., this is the actual content of an HTTP GET message). The characters <CR><LF> are carriage-return and line-feed characters. Answer the following questions, indicating where in the HTTP GET message below you find the answer.

```
GET /classes/spring17/cs118/project-1.html HTTP/1.1<CR><LF>
Host: web.cs.ucla.edu<CR><LF>
Connection: keep-alive<CR><LF>
Upgrade-Insecure-Requests: 1<CR><LF>
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/56.0.2924.87 Safari/537.36<CR><LF>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8<CR><LF>
Referer: http://web.cs.ucla.edu/classes/spring17/cs118/homeworks.html<CR><LF>
Accept-Encoding: gzip, deflate, sdch<CR><LF>
Accept-Language: en-US,en;q=0.8,lv;q=0.6,ru;q=0.4<CR><LF>
If-None-Match: "5a17-54c4847c4f640-gzip"<CR><LF>
If-Modified-Since: Mon, 03 Apr 2017 19:36:49 GMT<CR><LF>
```

1. What is the **full** URL of the document requested by the browser?
2. What version of HTTP is the browser running?
3. What type of browser initiates this message? Why is the browser type needed in an HTTP request message?
4. Can you find the IP Address of the host on which the browser is running from the captured HTTP request?

1. http://web.cs.ucla.edu/classes/spring17/cs118/project-1.html
   This URL was found by looking at lines 1 and 2 of the HTTP GET message by concatenating the host string in line 2 with the relative path found in line 1.
2. The browser is using version 1.1 of HTTP. The HTTP version run by the browser is found by examining line 1 of the HTTP GET message.
3. Because the user-agent software is listed in decreasing order of significance, the type of browser initiating this message is Chrome v56.0.2924.87 running on a Mac OS-X based machine. The type of browser is found on line 5 of the HTTP GET message under the User-Agent field. The browser type is needed in an HTTP request message because the server often has different versions of the same object for each browser. Different versions of the same object exist because each browser handles the same data differently due to the different capabilities offered by each browser.
4. As long as the host has a public IP address, you can find the IP address of the host by passing the host name into a DNS (Domain Name System) server. The host name can be found on line 2 of the HTTP GET message.

Problem 2

For each of the questions below, describe answer in terms of low-level packet sequences, drops, or network-level packet reordering.

1. A specific case where HTTP/1.1 wins in performance compared to HTTP/1.0
2. A specific case where HTTP with web caching wins in performance compared to HTTP without caching

1. HTTP/1.1 wins in performance over HTTP/1.0 when many objects need to be fetched on a Web page. HTTP/1.1 uses persistent TCP connections, so the delays of initiating the TCP connection to the server for each object needed is avoided in HTTP/1.1, as the same connection can be reused. Specifically, the low-level packets are sent over the same connection. In addition to persistent connections, pipelining is also an available feature in HTTP/1.1. This means that network-level packet sequences can be ordered such that packets are sent immediately after one another the moment data can be sent. This reduces the total delay of $2 \times N \times RTT$ where N is the number of objects when HTTP/1.0 is used to just 1 RTT + data transfer time with HTTP/1.1 with pipelining.
2. HTTP with web caching reduces load on the server, reduces traffic on the access link of the content provider, and reduces the response time for clients. Specifically, in the case where two clients request the same data objects and share the same proxy server, one client will most likely experience a cache hit in the proxy server from the other client requesting the same object in the past which avoids the overhead of requesting data from the origin server. This is a win for performance because instead of usually requesting low-level packets from the server which requires RTT delays from sending packets between the client and the server, a proxy hit in the cache completely eliminates any overhead associated with communicating with the server itself.

Problem 3

Suppose within your Web browser you click on a link to obtain a Web page. The IP address for the associated URL is cached in your local host, so a DNS look-up is not needed. Suppose that the Web page associated with the link is a small HTML file, consisting only of references to 100 very small objects on the same server. Let RTT_0 denote the RTT between the local host and the server containing the object. How much time elapses (in terms of RTT_0) from when you click on the link until your host receives all of the objects, if you are using:

1. HTTP/1.0 without parallel TCP connections?
2. HTTP/1.0 with parallel TCP connections?
3. HTTP/1.1 without parallel connections, but with pipelining?

Ignore any processing, transmission, or queuing delays in your calculation.

1. HTTP/1.0 uses non-persistent connections, so a new connection must be initiated for each object. $1RTT_0$ is required to initiate a connection to the server, and $1RTT_0$ is required to request and receive the Web page for a total of $2RTT_0$. After receiving the Web page, these two steps need to be repeated for all 100 objects on the Web page. Because there are no parallel TCP connections, each request and response is sequential, so each object on the page requires an additional $2RTT_0$ for initiating the connection to the server and fetching the object. Thus, the total elapsed time $= 2RTT_0 + 100 \times 2RTT_0 = 202RTT_0$.
2. HTTP/1.0 with parallel TCP connections results in a faster RTT time compared to HTTP/1.0 without parallel TCP connections. Although parallel TCP connections can essentially request all 100 objects at the same time, we don't know which objects to fetch until we have successfully requested the Web page that holds the objects. $1RTT_0$ is required to initiate a connection to the server, and $1RTT_0$ is required to request and receive the Web page for a total of $2RTT_0$. Once we've loaded the Web page, we can open 100 parallel TCP connections with the server and fetch each object in parallel because the problem does not state a maximum number of parallel TCP connections allowed. This requires another $1RTT_0$ to initiate a connection to the server and $1RTT_0$ to request and receive an object. Because 100 parallel TCP connections are set up, we can request each object in parallel after the Web page is received. Thus, the total elapsed time $= 2RTT_0 + 2RTT_0 = 4RTT_0$.
3. HTTP/1.1 uses persistent connections, so no new connections need to be initiated for each object once the Web page is received. Pipelining also reduces the overall RTT. $1RTT_0$ is required to initiate a connection to the server and $1RTT_0$ is required to request and receive the Web page for a total of $2RTT_0$. Because HTTP/1.1 uses persistent connections, no additional delay is required to initiate the connection to the server because the same connection can be used to fetch each object. Once the Web page is received, 100 requests can be pipelined and sent out, and data can be received in a pipelined manner as well. Pipelining 100 requests only requires $1RTT_0$ if we ignore any processing, transmission, and queuing delays. Thus, the total elapsed time $= 2RTT_0 + 1RTT_0 = 3RTT_0$.

Problem 4

BitTorrent is a communication protocol for peer-to-peer file sharing which is used to distribute data (or files) over the Internet.

1. Consider a new peer A that joins BitTorrent swarm without possessing any chunks. Since peer A has nothing to upload, peer A cannot become a top uploader for any of the other peers. How then will peer A get the first chunk?
2. Explain why BitTorrent is primarily useful for popular files but not for unpopular files.
3. Consider two DHTs (Distributed Hash Table) with a mesh overlay topology and a circular overlay topology, respectively. What are the advantages and disadvantages of each design?

1. BitTorrent has a mechanism called "Optimistic Unchoke" that essentially allows new peers to get their first chunks of data. This means that periodically, an existing peer will randomly select another peer and begin to send it chunks of data. If peer A gets randomly selected, it will begin to receive chunks. Now it has data to upload to its peers. At this point, the "Regular Unchoke" mechanism of uploading to the peers that are sending it chunks at the highest rate can ensure that peer A is motivated to stay in the torrent.
2. BitTorrent is useful for popular files because if many peers already own chunks of a file, then a peer A that is missing these chunks can simply receive the chunks from any of these peers, allowing peer A to quickly find any chunks that it needs due to the high availability of these required chunks. Meanwhile, receiving unpopular files may be troublesome. There are few peers to download from, making it difficult for peer A to find the peers that own the chunks it needs. If no other peer owns the chunks required by peer A, peer A will simply not be able to obtain its missing chunks.
3. Mesh overlay topology advantages:
   (a) Relies on the adjacent key
   (b) Accepts many paths of delivery

   Mesh overlay topology disadvantages:
   (a) High structural complexity
   (b) The complex delivery mechanism requires many resources

   Circular overlay topology advantages:
   (a) Simple in that there are only 2 peers, a predecessor and a successor

   Circular overlay topology disadvantages:
   (a) Requires key and inefficient hop delays
   (b) Messages are reduced

Problem 5

The server tries to distribute a file of $F = 15Gbits$ to N clients (peers). The server has an upload rate of $u_s = 30Mbps$, and each peer has a download rate of $d_p = 2Mbps$ and upload rate of $u_p = 1Mbps$. How long does it take to distribute if there are 1,000 peers for both **client-server distribution** and **P2P distribution**.

1. Client-server distribution:
   The server sequentially uploads N copies of the file in $N \times \frac{F}{u_s}$
   Each client p takes $\frac{F}{d_p}$ time to download the file
   Thus, the time to distribute a file of size F to N clients $= max(N \times \frac{F}{u_s}, \frac{F}{min_p(d_p)}) = max(1,000 \times \frac{15\times10^9}{30\times10^6}, \frac{15\times10^9}{2\times10^6}) = max(500000, 7500) = 500,000s$
2. P2P distribution:
   The server sends one copy of the file in $\frac{F}{u_s}$
   Each client p takes $\frac{F}{d_p}$ time to download the file
   A total of $N \times F$ bits must be downloaded (in aggregate)
   The maximum upload rate $= u_s + \sum u_p$
   Thus, the time to distribute a file of size F to N clients is at least $= max(\frac{F}{u_s}, \frac{F}{min_p(d_p)}, \frac{N\times F}{u_s+\sum u_p}) = max(\frac{15\times10^9}{30\times10^6}, \frac{15\times10^9}{2\times10^6}, \frac{1,000\times(15\times10^9)}{(30\times10^6)+1,000\times(1\times10^6)}) = max(500, 7500, 14563.1068) = 14,563.1068s$