1 True or False

1. One of the goals of a CDN is to lower latency for clients.

   Solution: True. CDNs place data closer to clients and thus reduce the latency of loading. Furthermore, they also spread load.

2. HTTP 1.0 Requests are not human readable.

   Solution: False. HTTP 1.0 Requests are in plaintext and are thus *mostly* human readable.

3. If an HTTP request includes the header field "If-Modified-Since", then the HTTP server responds with only a header when the requested object has not been recently modified

   Solution: True. The server responds with a "304: Not Modified", a header and **no** body.

4. Pipelined connections are frequently used in practice.

   Solution: False. Pipelined connections are not commonly used because of a) bugs and b) head-of-line blocking. HTTP2 uses multiplexing which avoids these shortcomings.

2 Performance

We want to download a webpage. We must first download the HTML (size P). This HTML includes URLs for two embedded images of size M which need to then be loaded. Assume the following:

- SYN, ACK, SYNACK, and HTTP request packets are small and take time z to reach their destination in either direction.
- Each of our HTTP connections can achieve throughput T for sending files and web pages across the network unless there are concurrent connections, in which case each connection's throughput is divided by the number of concurrent connections.
- You never need to wait for TCP connections to terminate.

For each of the following scenarios, compute the total time to download the web page and both media files.

1. Sequential requests with non-persistent TCP connections.

   Solution:

   (a) $2z$ (SYN + SYNACK) $+z$ (ACK/HTTP request) + $(\frac{P}{T}+z)$ (actual webpage)
   (b) $2z$ (SYN + SYNACK) $+z$ (ACK/HTTP request) + $(\frac{M}{T}+z)$ (first media file)
   (c) $2z$ (SYN + SYNACK) $+z$ (ACK/HTTP request) + $(\frac{M}{T}+z)$ (second media file)

   Total $= \boxed{12z+\frac{P}{T}+2\frac{M}{T}}$

2. Concurrent requests with non-persistent TCP connections.

   Solution:

   (a) $2z$ (SYN + SYNACK) + z (ACK/HTTP request) + $(\frac{P}{T} + z)$ (actual webpage)

   (b) $2z$ (SYNs + SYNACKs) + z (ACKs/HTTP requests) + $(\frac{M}{(\frac{T}{2})} + z)$

   Total $= \boxed{8z + \frac{P}{T} + 2\frac{M}{T}}$

3. Sequential requests with a single persistent TCP connection.

   Solution:

   (a) $2z$ (SYN + SYNACK) $+z$ (ACK / HTTP request) $+(\frac{P}{T} + z)$ (actual webpage)

   (b) z (HTTP request) $+(\frac{M}{T} + z)$ (first media file)

   (c) z (HTTP request) $+(\frac{M}{T} + z)$ (second media file)

   Total $= \boxed{8z + \frac{P}{T} + 2\frac{M}{T}}$

4. Pipelined requests within a single persistent TCP connection.

   Solution:

   (a) $2z$ (SYN + SYNACK) $+z$ (ACK / HTTP request) + $(\frac{P}{T} + z)$ (actual webpage)

   (b) z (Both HTTP requests)

   (c) $\frac{M}{T}$ (first media file)

   (d) $(\frac{M}{T} + z)$ (second media file)

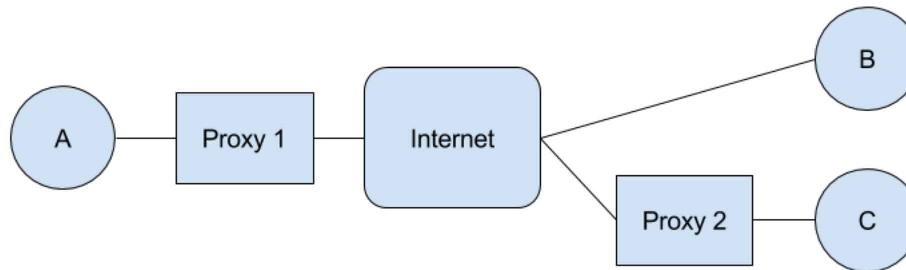
   Total $= \boxed{6z + \frac{P}{T} + 2\frac{M}{T}}$

   Note that above, the second media file can begin to be sent immediately after the first is pushed onto the wire; hence, we don't need to account for the propagation delay of the first file in our calculation.

5. We have been assuming that the throughput for sending media files is T for a single connection, and $\frac{T}{n}$ for n concurrent connections. Remember that the throughput for sending the media files depends on both its transmission delay and propagation delay. So far we ignored this finer granularity division but depending on the size of the media files, we can make more inferences about how fast we can send the media files. If the media files are very small, what kind of delay would dominate the time it would take to send them? What if the files are very large?

   Solution: If the media files are very small, then transmission delay is small so propagation delay dominates. If the media files are very large, then transmission delay is large, and so dominates.
   The takeaway from the 5 parts of this question is that adding concurrency, re-using existing TCP connections and pipelining all help to speed up end-to-end page loads.

3 HTTP



Consider the (abstracted) network topology above. Hosts A and C are connected to HTTP proxies that cache the results of the last two HTTP requests they've seen to improve performance. The proxy will perform any TCP handshakes or teardowns with the client and server concurrently. That is, when it gets a SYN from a client, it will respond and immediately send a SYN to the server, and similarly for other messages.

As an example, let's suppose that A sends a request to B. A sends a SYN to B, which is intercepted by the proxy. The proxy sees the request is going to B, and initiates its own TCP handshake with B, all the while completing the original, separate handshake with A. By the time this has completed, there will be 2 TCP sessions. The first between A and the proxy, and the second between the proxy and B. When A sends a request, the proxy will forward it to B if it is not cached, or respond if it is cached. A similar process to the handshake is followed when A tears the connection down.

For the purposes of this problem, assume that the latency on each link is L, and the latency through the internet is I. Processing delay at all points and packet size is assumed to be negligible (don't consider transmission and processing delay). Assume that TCP connections use the 3 message teardown, and that no data is sent in ACK packets.

Suppose that Host A issues the following list of requests (in order):

- berkeley.edu to Host C
- eecs.berkeley.edu to Host C
- stanford.edu to Host B
- mit.edu to Host B
- stanford.edu to Host B
- berkeley.edu to Host C

1. What is the total time to complete all requests if they are issued one at a time? That is, each completes before the next is started with a separate TCP session (no need to wait for the session to be torn down).
   Solution: To set up a connection to B, A will initiate a TCP connection with P1, which will establish a connection to B. The total time to establish the connection is the time for the SYN to reach P1 plus the time it takes P1 to establish a connection to B. This is $L+2(2L+I)$, since the proxy will send the request from A right after sending the ACK. **Note:** the SYNACK from P1 to A and SYN from P1 to B occurs concurrently so we could write the term $2(2L+I)$ in the equation above more precisely as $\max\left(L,\ 2(2L+I)\right)$ but that is equivalent to $2(2L+I)$

   To set up a connection to C, A will initiate a TCP connection with P1, which will do so with P2 (who it thinks is C), which will do so with C. Similarly to before, the time this takes is $L+2(2L+I)$, since the longest delay is between the two proxies, that's all we need to consider. Since this time is the same for both servers, we'll call it S, the setup time.

In all cases, A will complete its handshake with its proxy before the proxy finishes its handshake with the server, and so the HTTP request from Host A will have already arrived at P1 and will be sent immediately after the proxy P1 finishes its handshake. This means that in the total request time for requests to B and C, the latency from A to P1 doesn't matter. This total request time is only affected by the latency from P1 to hosts B and C.

The latency from P1 to B is $2L+I$, we'll call this L_B.

The latency from P1 to C is $3L+I$, we'll call this L_C.

Even though they share the same subdomain base, the first request (berkeley.edu) and second request (eecs.berkeley.edu) go to different servers - eecs.berkeley.edu is hosted in a different place than berkeley.edu) - so won't be helped by the cache. Therefore they take $S+2L_C+L$ time each. Look at the diagram below to see why this is so (purple is the request for the page and green is the response).

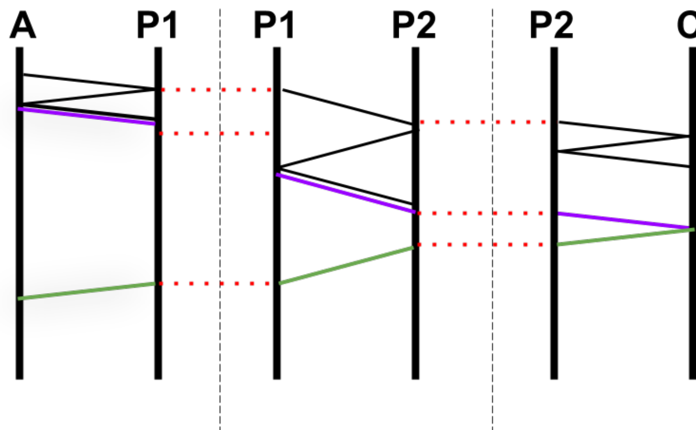
The third and fourth requests are similar, taking $S+2L_B+L$ time each.

The fifth request hits P1's cache, only taking $2L+2L$ time.

The sixth request hits P2's cache, taking $S+2(L_C-L)+L$ time.

Therefore the total time is:

$2(S+2L_C+L)+2(S+2L_B+L)+4L+S+2(L_C-L)+L =$
$2S+4L_C+2L+2S+4L_B+2L+4L+S+2L_C-2L+L =$
$5S+6L_C+4L_B+7L =$
$5(5L+2I)+6(3L+I)+4(2L+I)+7L =$
$25L+10I+18L+6I+8L+4I+7L =$
$58L+20I$



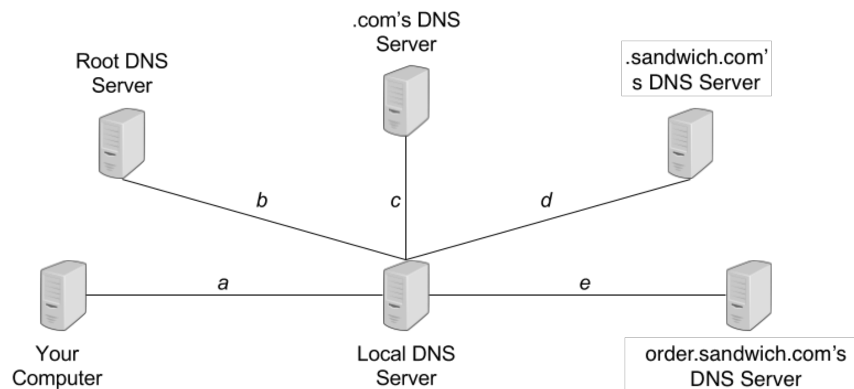
2. What is the total time to complete all requests if they are performed concurrently such that everything is done in parallel and there is no possibility for caching?

Solution: Since all the requests occur concurrently, there is no opportunity for caching by the proxies. Therefore the total time is just the max of all the times for the individual requests, which is the first request. It takes:

$S+2L_C+L =$
$5L+2I+2(3L+I)+L =$
$5L+2I+6L+2I+L =$
$12L+4I$

4 DNS

A sandwich ordering website www.order.sandwich.com is accepting online orders for the next T minutes. Consider the following setup of DNS servers, with annotated latencies between servers:



Assume that the latency between your computer and the website's server is t, that once you send an order for a sandwich you must wait for a confirmation response from the website before issuing another, and that your computer does not cache website's IP address.

In each of the following three scenarios below, determine how many sandwiches you can order for the next T minutes:

1. Your local DNS server doesn't cache any information.

   Solution: Your computer begins by issuing a DNS query for www.order.sandwich.com to its local DNS server, which takes time a. Your local DNS server then iteratively queries the root DNS server, .com's DNS server, .sandwich.com's DNS server, and order.sandwtch.com's DNS server, which takes time $2b+2c+2d+2e$. It then returns the result of the query to you, which takes time a. Your computer can then issue a sandwich request to www.order.sandwich.com, which responds with an order confirmation, which takes time $2t$. The total time per query is hence

   $$2a+2b+2c+2d+2e+2t$$

   Each sandwich order takes this much time, since our local DNS server doesn't cache the IP address corresponding to www.order.sandwich.com, and so the number of orders we can make in this time is

   $$\boxed{\#\text{ sandwiches} = \left\lfloor \frac{T}{2a+2b+2c+2d+2e+2t} \right\rfloor}$$

   (rounding down to the nearest integer with the floor function $\lfloor x \rfloor$).

2. Your local DNS server caches responses, with a time-to-live $L \geq T$.

   Solution: If $L \geq T$, then this effectively means that once the result of the DNS query for www.order.sandwich.com is cached, it will remain cached in our local DNS server until the website's server ultimately goes down. The first query thus takes the same amount of time as a query from the previous part

   $$2a+2b+2c+2d+2e+2t$$

so long as this time is less than T. All subsequent queries only take time $2a+2t$ thanks to caching. Hence we can model the number of sandwiches we get as follows:

$$\text{\# sandwiches} = \begin{cases} 1 + \left\lfloor \frac{T-(2a+2b+2c+2d+2e+2t)}{2a+2t} \right\rfloor & T \geq 2a+2b+2c+2d+2e+2t \\ 0 & T < 2a+2b+2c+2d+2e+2t \end{cases}$$

3. Let $T = 600$ seconds and $a = b = c = d = e = t = 1$ second. Your local DNS server caches responses with a finite time-to-live of 30 seconds.

   Solution: When you make your first DNS query, the response gets cached in your local DNS server at time $a+2b+2c+2d+2e = 9$ seconds, and will remain cached for the next 30 seconds, until time 39.

   Once the response is cached, your local DNS server sends it to your computer (which arrives at time 10) and then you issue a sandwich request (and receive confirmation at time 12). This gives you $39-12 = 27$ additional seconds to make more requests until the TTL for the cached entry expires, during which time you can make $\left\lceil \frac{27}{4} \right\rceil = 7$ additional sandwich requests, each request is completed at time $12, 16, 20, 24, 28, 32, 36$ and 40.

   At this point, the cached DNS response has expired, and 40 seconds have elapsed. The events above can repeat a maximum of $\frac{600}{40} = 15$ times in our 600 second window of opportunity, and so we can order at most $15 \cdot (7+1) = \boxed{\text{120 sandwiches}}$. Not bad!