Problem 1 (20 points)
Short and quick questions
(1) A web browser sets up a TCP connection to fetch data from a web server.
Assuming the browser uses HTTP/1.0: is it possible for the browser to send a single TCP segment carrying 2 distinct HTTP requests? Explain your answer.

No, it is not possible/not useful. Every request/response belongs to a separate TCP connection. Though it is possible to send 2 requests in the same TCP segment, only first request will be answered (HTTP 1.0 doesn't support request pipelining)

Assuming the browser uses HTTP/1.1: is it possible for the browser to send a single TCP segment carrying 2 distinct HTTP requests? Explain your answer.

Yes. Because HTTP 1.1 supports pipelining, nothing prevents anybody to send two distinct requests in the same TCP segment. Both requests will be processed by the server and be answered to the requester.

(2) Is it correct to claim that neither the HTTP status code nor the corresponding reason string in HTTP response messages (see slides 53, 58 in week1-L2 slide deck) is *necessary* for HTTP to function correctly? Please explain your answer.

No. While HTTP response message is only useful for humans to debug the problem, everybody using HTTP protocol rely on HTTP status code to determine if the request returned useful data (200), there is need to resend request to a different place (301), or some other error happened (404, 500, or others)

(3) Because DNS uses UDP for data delivery, packets may get lost, so DNS caching resolvers must detect lost requests and retransmit. We learned in class that TCP uses a fine-tuned adaptive retransmission timer. Does DNS also use an adaptive retransmission timer? Why or why not?

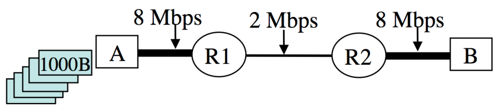
No. DNS does not use adaptive timer. It uses fixed timer (e.g., 3 sec). Caching resolvers do not usually communicate with the same DNS server, and if communicate, only for a short period of time. As a result, there are not enough samples to establish a good estimation for the round-trip time / retransmission timeout.
** Some DNS requests are performed using TCP, so there exists a RTO adaptation within such connections. However, these cases are rare. Most of DNS queries are connectionless.*

(4) Recent years have witnessed large scale DDoS attacks against the DNS system: the attackers sent large volumes of DNS queries to DNS root servers. The first one occurred in October 2002 and disrupted availability of about half of the root servers. However, according to news reporters: "There were no known reports of end-user-visible error conditions during, and as a result of, this attack". Can you explain why? List all the reasons you can identify.

a) Redundancy: more than half of the root servers were still intact
b) Caching: for most of the queries there is no need to go down to the root servers, information is cached within caching resolvers.

Problem 2 (20 points)

Consider sending 5 packets from Node A to Node B via routers R1 and R2 (see the figure below). The packet length is 1000 bytes each. The propagation delay of all the 3 links is 2 msec (0.002 second). The bandwidth of Links A-R1 and R2-B is 8 Mbps (8x10^6 bits per second), and link R1-R2's bandwidth is 2 Mbps.



Assume A starts transmitting the first packet at time t = 0,

a) What is the time gap between the first and second packets when they arrive at B? (i.e. the time gap between receiving the last bit of the first packet and the last bit of 2nd packet)

The link between R1 and R2 is bottleneck link. When the second packet arrives at R1, it has to be queued until the first packet leaves R1. Therefore, the gap between last bits of two packets is the transmission delay of the second packet on R1.

d_{gap} = (1000 Bytes * 8 bits/Bytes) / (2 *10^6 bits/sec) = 4 msec.

b) When will B receive all the 5 packets?

d_{transA} = (1000 Bytes * 8 bits/Byte) / (8 * 10^6 bits/sec) = 1ms

$d_{transR1}$ = (1000 Bytes * 8 bits/Byte) / (2 * 10^6 bits/sec) = 4ms

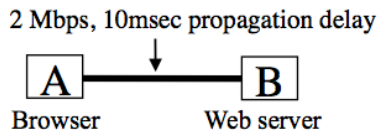
$d_{transR2}$ = (1000 Bytes * 8 bits/Byte) / (8 * 10^6 bits/sec) = 1ms

$d_1 = d_{transA} + d_{prop1} + d_{transR1} + d_{prop2} + d_{transR2} + d_{transA}$ = 1ms + 2ms + 4ms + 2ms + 1ms + 2ms = 12ms

Total delay = $d_1 + 4*d_{gap}$ = 12ms + 4 * 4ms = 28 ms

Problem 3 (20 points)

A web browser on the host A is connected to a web server on the host B over a link with 2Mbps bandwidth and 10 msec propagation delay, as shown in the figure. A needs to send 4 HTTP requests to B, each HTTP request will retrieve 800 bytes of data.



Assuming the size of HTTP requests is small enough so that their transmission delay can be ignored. Also assume that TCP flow and congestion control window sizes are big enough so that they do not slow down data transmission.

(1) Assuming the browser uses HTTP/1.0 to retrieve the data. To speed up the retrieval the browser opens 4 TCP connections in parallel. How long will it take for the browser to receive all 4 pieces of data?

As connections are in parallel, they share the same channel, thus packets has to be queued before actually being transmitted. For SYN, SYN-ACK, and ACK packets transmission/queuing delay can be ignored, but for DATA packets it has to be considered.

delay = T_{prop}*(SYN)* + T_{prop}*(SYN-ACK)* + T_{prop}*(HTTP REQ)* + T_{prop}*(HTTP RESPs+ACKs)* + $T_{trans+queueing}$*(HTTP RESPs+ACKs)*

$T_{trans+queueing}$*(HTTP RESPs+ACKs) ~= 4**$T_{transmission}$*(HTTP RESP) ~= 4*(800 Bytes *8 bits/Byte) / (2**10^6 *bits/sec) = 12.8 ms ~ 13ms*

delay ~= 10 ms + 10 ms + 10 ms + 10 ms + 13 ms = 53 ms

(2) Assuming the browser uses HTTP/1.1 with pipelining to retrieve the data. How long will it take for the browser to receive all 4 pieces of data?

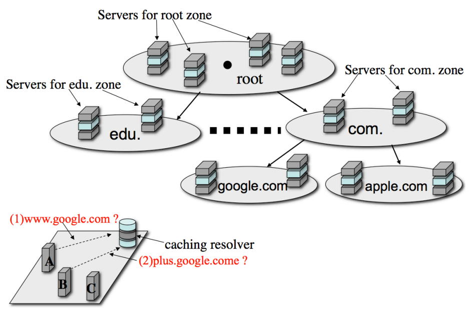
The only difference with pipelining that there is no additional overhead for establishing multiple connections. As long as we are ignoring transmission delay for SYN and SYN-ACKs, the calculation is exactly the same. It will take T_{prop} *for SYN to get to the server,* T_{prop} *to get back with SYN-ACK,* T_{prop} *to get request to the server,* $T_{prop}+T_{trans}$*(4 data packets) to get data back. There still the same channel and we need to get all four data packets through it, one by one.*

delay = T_{prop}*(SYN)* + T_{prop}*(SYN-ACK)* + T_{prop}*(HTTP REQ)* + T_{prop}*(HTTP RESPs+ACKs)* + $T_{trans+queueing}$*(HTTP RESPs+ACKs)*

$T_{trans+queueing}$*(HTTP RESPs+ACKs) ~= 4**$T_{transmission}$*(HTTP RESP) ~= 4*(800 Bytes *8 bits/Byte) / (2**10^6 *bits/sec) = 12.8 ms ~ 13ms*

delay ~= 10 ms + 10 ms + 10 ms + 10 ms + 13 ms = 53 ms

Problem 4 (20 points)
Consider the following DNS resolution process: at time T=0, the caching resolver in the figure has an empty cache, and Host-A sends a query to resolve the DNS name of *www.google.com*. 10 minutes after Host-A received the answer from the caching resolver, Host-B sends a query for the name *plus.google.com*. Then 10 minutes after Host-B receives its answer, Host-C sends a query for DNS name *www.apple.com* (not shown in the figure). Assuming that it takes 0 (zero) seconds for all packet exchanges between the local hosts and the caching resolver, and it takes 100 msec for the caching resolver to get a reply from any of the DNS servers. All the DNS data has a TTL of 24 hours.



(1) How long does it take for Host-A to get the answer back for the IP address of *www.google.com*?

300ms

Host-A ..... Caching resolver (0ms)
Caching resolver .... Root --- get information about .Com (100ms)
Caching resolver .... a .Com name server --- get information about .Google.com (100ms)
Caching resolver .... a .Google.com name server --- get information about Www.google.com (100ms)
Caching resolver ... Host-A (0ms)

(2)How long does it take for Host-B to get the answer back for the IP address of *plus.google.com*?

100ms

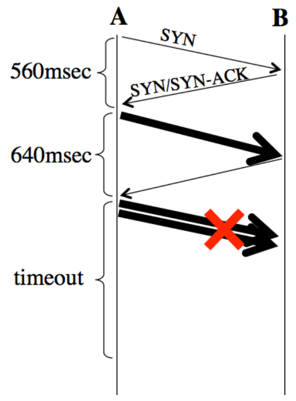
Host-B .... Caching resolver (0ms)
Information from Root about .Com is in the cache (no need to query .Root)
Information about .Google.com is in the cache (no need to query .Com name server)
Caching resolver ... a .Google.com name server - get information about Plus.google.com (100ms)
Caching resolver ... Host-B (0ms)

(3)How long does it take for Host-C to get the answer back for the IP address of *www.apple.com*?

200ms

Host-C ... Caching resolver (0ms)
Information from Root about .Com is in the cache (no need to query .Root)
Caching resolver ... a .Com name server --- get information about .Apple.com (100ms)
Caching resolver ... a .Apple.com name server --- get information about Www.apple.com (100ms)
Caching resolver ... Host-C (0ms)

Problem 5 (20 points)
Host-A sets up a TCP connection to host-B to transmit a large file to B, which takes many packets. TCP performs Slow-Start congestion control. Let us consider A's retransmission timer value calculation: SRTT and DevRTT are initialized to 3-second each. The first round trip delay measure is 560msec (SYN and SYN-ACK, please refer to slide-27 in week4), the 2nd is 640msec (first data and ACK).



5.1 What is the RTO value when A sends the 3rd and 4th packets out? (Use the parameters suggested on slides in week4)

alpha = 1/8
beta = 1/4

t0 (initial): SRTT = 3s, DevRTT = 3s

t1 (SYN-ACK received): SRTT(t1) = 560s, DevRtt(t1) = SRTT(t1)/2 = 280ms (rfc2988, rule 2.2)

t2 (after first data is acknowledged, rfc2988, rule 2.3):

*DevRTT(t2) = (1-1/4) * 280 + 1/4 * abs(560 - 640) = 210 + 20 = 230 ms*
*SRTT(t2) = (1-1/8) * 560 + 1/8 * 640 = 490 + 80 = 570 ms*

*RTO(t2) = SRTT + 4*DevRTT = 570 + 4*230 = 1490 ms = 1.49 seconds*

5.2 Unfortunately both the 3rd and 4th packets are lost. A's TCP connection times out and retransmit the 3rd packet. What is the RTO value this time?

Double the value of RTO calculated in 5.1

*t3 (after 3rd packet of TCP connection / 2nd data packet timed out): RTO(t3) = 2*RTO(t2) = 2*1.49 = 2.98 seconds*

rfc2988:
```
When the retransmission timer expires, do the following:
(5.4) Retransmit the earliest segment that has not been acknowledged by the TCP
receiver.
(5.5) The host MUST set RTO <- RTO * 2 ("back off the timer").  The maximum value
discussed in (2.5) above may be used to provide an upper bound to this doubling
operation.
(5.6) Start the retransmission timer, such that it expires after RTO
seconds (for the value of RTO after the doubling operation outlined in 5.5).
```