yz709-networking-sup5

Question 1(y2016p5q4)
Question 2(y2014p5q5(a))
Question 3
Question 4
Question 5
Question 6
Question 7
Question 8
Question 9

Question 1(y2016p5q4)

Two objects are being retrieved from B by A using HTTP over a network where the TCP-style transport protocol has no maximum segment size and the network experiences no loss. Each object is 125 kByte (i.e., one Mbit) in size. We assume all connection setup packets and HTTP request packets are negligible in size; we ignore the connection tear-down.

(a) Suppose the Round-Trip-Time (RTT) between A and B is 10 milliseconds and the bandwidth between the sites is 10 Mbit/s. Illustrate with a simple diagram in each case how long it takes for A to retrieve both files under the following circumstances: [2 marks each]

(i) Sequential (one-at-a-time) requests with non-persistent connections.

(ii) Concurrent requests with non-persistent connections.

(iii) Sequential requests within a single persistent connection.

(iv) Pipelined requests within a single persistent connection.

(a) RTT = 10 ms

B = 10 Mbits/s

(i)

125 KByte = 1 Mbits

1 Mbits / B = 0.1 s = 100 ms

(2RTT + 100 ms) x2 = 240 ms

(ii)

concurrent connections - use multiple TCP connections in parallel

So each has half of the bandwidth

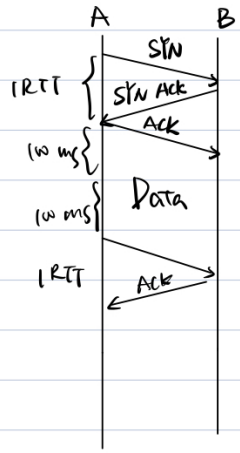
1 Mbits / 5 Mbits/s = 0.2 s = 200 ms

2RTT + 20 ms = 220 ms

(iii)

3RTT + 100x2 = 320 ms

(iv)



2 RTT + 200 ms = 220 ms

assume TCP segment size has no limitations

(*b*) Suppose A is connected to a cache C by a link with 10 Gbit/s bandwidth and negligible RTT. C connects to B with the link originally specified in part (*a*): 10 Mbit/s and 10 millisecond RTT.

The cache operates as follows:

- If the object is not in the cache, the request is forwarded to B which responds with the object, which the cache stores with an object timeout and then forwards to A.
- If the object is in the cache, and the cache entry has not timed out (i.e., the cache Time To Live (TTL) has not expired), the object is returned to the client.
- If the object is in the cache, but the cache entry has timed out, the cache issues a conditional GET to the original server, asking if the object has changed since this object was cached. If the original server responds that it has not, the cache returns the cached object, otherwise the original server responds with the updated object which the cache forwards to the client.

Showing your working, how long does it take for A to retrieve one file under the following circumstances: [3 marks each]

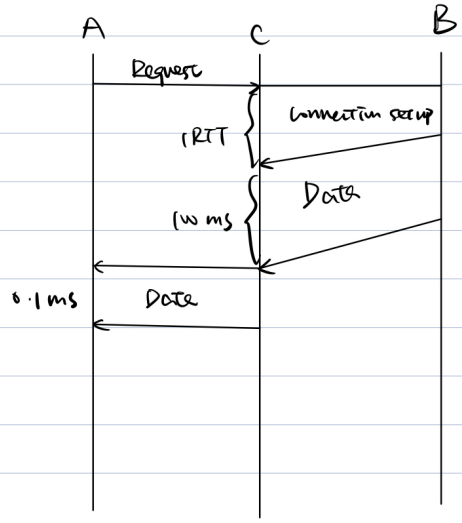
(*i*) The file is not in the cache

(*ii*) The file is in the cache and the TTL has not expired

(*iii*) The file is in the cache, the TTL has expired, but the file has not changed

(*iv*) The file is in the cache, the TTL has expired, and the file has changed

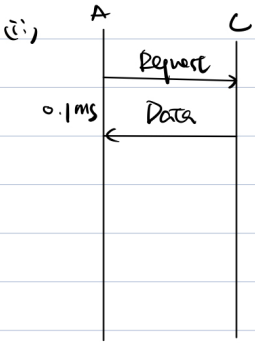
(i)



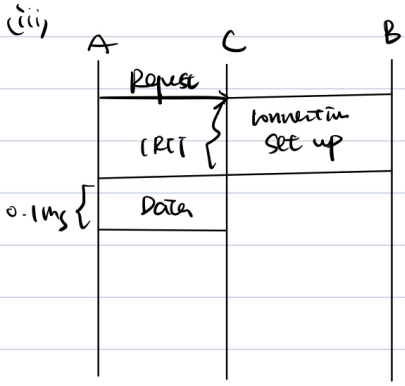
$$1\ \text{Mbit} / 10\ \text{Gbit/s} = \frac{1\ \text{Mbits}}{10 \times 10^3\ \text{Mbits/s}} = 10^{-4}\ \text{s} = 0.1\ \text{ms}$$

$$1\ \text{RTT} + 10\ \text{ms} + 0.1\ \text{ms} = 120.1\ \text{ms}$$

(ii)

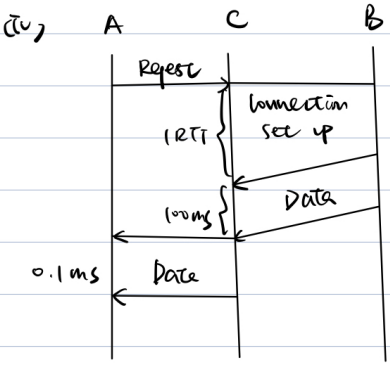


(iii)



1 RTT + 0.1 ms = 20.1 ms

(iv)



1 RTT + 100 ms + 0.1 ms = 120.1 ms

Comments:

Question 2(y2014p5q5(a))

(a) Below is an excerpt from the DNS record for a fictitious corporation, Lemon:

Name	Type	Value	TTL (seconds)
lemon.co.uk	A	91.45.20.24	86400
lemon.co.uk	NS	grove.lemon.co.uk	86400
lemon.co.uk	NS	tree.lemon.co.uk	86400
lemon.co.uk	MX	stem.lemon.co.uk	60
grove.lemon.co.uk	A	91.45.23.22	86400
tree.lemon.co.uk	A	91.45.23.23	86400
orchard.lemon.co.uk	A	91.45.23.82	86400
stem.lemon.co.uk	A	91.45.23.85	86400
www.lemon.co.uk	CNAME	orchard.lemon.co.uk	86400

(*i*) If you type `http://www.lemon.co.uk` into your web browser, to which IP address will your web browser connect? [1 mark]

- DNS records with type CNAME redirect a domain to a different domain and type A maps domain names to IP addresses.
- The final entry is retrieved from the table with the value `orchard.lemon.co.uk` which has an IP address of `91.45.23.82`

(*ii*) If you send email to `support@lemon.co.uk`, to which IP address will the message get delivered? [1 mark]

- DNS records with type MX provide the domain names of mail servers that receive emails on behalf of a domain. Hence the value `stem.lemon.co.uk` will be retrieved which maps to an IP address of `91.45.23.85`

(*iii*) The TTL field refers to the maximum amount of time a DNS server can cache the record. Most of the TTLs in this record were chosen to be 86400 seconds (1 day). What is the trade-off between choosing a shorter or a longer time? Why was the MX record specifically chosen to have a 60 second TTL? [4 marks]

- A shorter TTL may have a low throughput because fewer requests can get the record information within a short time period.
- A long TTL may lead to requests getting an out-of-date record.
- MX record is a mail exchange record, a short TTL of 60 seconds means this entry will need to be updated very often in order to prevent any messages sent to the mail server from being directed to an out-of-date IP address.

(*iv*) Explain why the Internet DNS uses caching. [2 marks]

- (1) reduce the overhead of performing queries before the actual communication;
- (2) popular sites are visited often, reducing the overhead of getting the same information from the root server;
- (3) The local server can serve multiple clients with the same query information.

(v) Comment on how the provision of name servers for `lemon.co.uk` affects the availabilty of the name service. [2 marks]

- DNS records with type `NS` provide a list of authoritative name servers responsible for the domain, there are 2 name servers for `lemon.co.uk`, this redundancy improves the reliability of the name service because if one name server is down, the other can still serve incoming requests.

(vi) Outline two strategies to improve availability of the DNS server for the `lemon.co.uk` domain. [2 marks]

- Make sure the DNS name servers for the domain are resident in different locations and are less likely to be attacked at the same time.
- Adding more name servers for the domain, replication increases reliability.

Comments:

Question 3

In HTTP version 1.0, a server marked the end of a transfer by closing the connection. (based on PD, ch 9, q 10)

1. Explain why, in terms of the TCP layer, this was a problem for servers.
    - This is inefficient because at least 2RTT are required to set up and tear down a TCP connection, the HTTP client needs to initiate a three-way-handshake TCP connection to the server for set-up and termination exchange, it introduced more latency for sending more objects, e.g., $2nRTT$ for sending n small objects.
2. How does HTTP version 1.1 avoid this?
    - HTTP 1.1 comes in two flavours:
        - (1) Persistent connection allows multiple requests and response messages over the same TCP connection to eliminate connection setup overhead.

    - (2) Pipelining connection improves on top of persistent connection, it batches requests and responses so that multiple requests can be sent in a single batch, leading to much less latency.

💡 Comments:

Question 4

When an HTTP server initiates a **close()** at its end of a connection, it must then wait in TCP state FIN WAIT 2 for the client to close the other end. What mechanism within the TCP protocol could help an HTTP server deal with noncooperative or poorly implemented clients that don't close from their end? (based on PD, ch 9, q 13)

- HTTP server could do an abrupt termination using the RST flag, once the client-side receives the RST flag, it will be forced to close the connection, thus avoiding a half-closed state. The reset packet sent by the server is one with no payload and with the RST bit set in the TCP header flags.

💡 Comments:

Question 5

SOAP has been largely deprecated in favour of REST protocols. (based on PD, ch 9, q 16)

1. Explain two differences between them.
    - Both are used to define how to build application programming interfaces that allow data to be communicated between web applications. But SOAP is a protocol with specific requirements like XML messaging and REST is a set of architectural principles that offer a flexible implementation.
    - SOAP cannot make use of REST since SOAP is a protocol and REST is an architectural pattern, but REST can make use of SOAP as the underlying protocol for web services.

2. Why has REST won out over SOAP?
    - REST APIs are lightweight, suitable for the Internet of Things, mobile application development and serverless computing, while SOAP web services are much heavier as they have built-in security and transaction compliance.

> 💡 Comments:

Question 6

Consider the host *here.eye.am*, with a local DNS server *nameserver.eye.am*. *here.eye.am* asks *nameserver.eye.am* to resolve the hostname *there.you.ar.* Assume there are no cached entries relevant to this request and that *nameserver.eye.am* utilises recursive resolution by default. (based on supervision exercises, topic 6, q 23)

1. Write down the steps taken to resolve *there.you.ar* and respond to *here.eye.am.*
    - The host `here.eye.` sends a query to the local DNS server `nameserver.eye.am` requesting the IP address of the host `there.you.are`.
    - The local DNS server contacts the root DNS server, which may contact a top-level DNS server (e.g., `ar`), and recursively, the TLD DNS server may contacts an authoritative DNS server `you.ar`, which in turn contacts a local DNS server `nameserver.you.ar` that stores the relevant IP address of the host `there you are`.
    - Once the response is sent back along the path to the local DNS server `nameserver.eye.am`, it stores the entry inside its local cache, and then sends the response back to the client `here.eye.am`
2. Describe the differences between this solution and one achieved using an iterative DNS enquiry.
    - In an iterative DNS enquiry, the burden of name resolution is put on the local DNS server only, it needs to contact other DNS servers iteratively to find the server which stores the required IP address. While in a recursive

DNS enquiry, the burden is of name resolution is spread across all contacted DNS servers, the task will be delegated along the path.

- In a long term, an iterative DNS query will be much faster than a recursive query because more entries will be cached in the cache of a local DNS server, but it also implies the workload of the local DNS server is higher in the iterative DNS enquiry method.

💡 Comments:

Question 7

Future DNS: (based on supervision exercises, topic 6, q 24.d (which should be 25?))

1. Why did DNS use UDP?
    - Most DNS queries are simple and have a small data payload, thus using UDP rather than TCP can help reduce query overhead, thus much faster to transfer and the client will have a better interactive experience.
2. What considerations may drive DNS to different transport protocols, like TCP?
    - UDP lacks reliability assurance which TCP can offer, and can only support a packet with limited size. In cases where the packet size is too large (e.g., > 512 byte) to push through in a single UDP packet, DNS will fall back to use TCP.
    - With the introduction of IPv6, modern DNS systems may more likely to transfer packets with a larger combined size.

💡 Comments:

Question 8

Imagine a CDN configured as a caching hierarchy, with end-users accessing content from edge caches, which fetch the content for a parent cache upon a cache miss,

and so on up to a root cache, which ultimately fetches content from an origin server. When would you decide to: (based on PD, ch 9, q 40)

(a) add more storage capacity to a given cache

- If the number of clients for a local cache has increases and exceeded the capacity (e.g., 70% of cache entries are actively used), then we should increase the storage capacity of the cache to prevent cases where evicted victim cache entries being requested again shortly.

(b) add an additional level to the caching hierarchy.

- If the geological span of one cache is too big, then some clients further away from the cache will experience a high latency, by adding an additional level, we could enable clients to have a closer distance to the local cache.

Comments:

Question 9

Consider the following simplified BitTorrent scenario. There is a swarm of 2^n peers and, during the time in question, no peers join or leave the swarm. It takes a peer 1 unit of time to upload or download a piece, during which time it can only do one or the other. Initially, one peer has the whole file and the others have nothing. (based on PD, ch 9, q 42)

1. If the swarm's target file consists of only 1 piece, what is the minimum time necessary for all the peers to obtain the file? Ignore all but upload/download time.
    - At t = 0, the single peer a_1 takes 1 unit of time to upload the file.
    - At t = 1, one peer a_2 downloads the file from that single peer a_1, which takes 1 unit of time.
    - At t = 2, one peer a_3 downloads the file from the first peer, the second peer a_2 uploads the file
    - At t = 3, two peers a_4 and a_5 download the file from the first two peers a_1 and a_2 who uploaded the file, and the third peer a_3 uploads the file

- It is a Fibonacci sequence 1 + 1 + 1 + 2 + 3 + 5 + 8 + ... ≥ 2^n, i.e., $F_0 + F_1 + F_2 + ... \geq 2^n - 1$, since the sum from F_0 to F_m is $F_{m+2} - 1$, we need to find m such that $F_{m+2} \geq 2^n$, and the total time taken would be $m + 1$ units of time.

2. Let *x* be your answer to the preceding question. If the swarm's target file instead consisted of 2 pieces, would it be possible for all the peers to obtain the file in less than 2*x* time units? Why or why not?

    - At t = 0, peer a_1 uploads piece 1 of the file; at t = 1, peer a_1 uploads piece 2 of the file; at t = 2 and t = 3, peer a_2 downloads piece 1 and 2 of the file from peer a_1; at t = 4 and t = 5, peer a_2 uploads piece 1 and 2 of the file, peer a_3 downloads two pieces from a_1.
    - At t = 6 and t = 7, peer a_4 and a_5 can download file pieces concurrently from a_1 and a_2, while peer a_3 uploads the file pieces.
    - At t = 8 and t = 9, peer a_6, a_7 and a_8 can download file pieces concurrently from a_1, a_2 and a_3, while a_4 and a_5 uploads the file pieces.
    - Hence we have 2 × (1 + 1 + 1 + 2 + 3 + 4 + ...) ≥ 2^n, and we can simplify that to 1 + 1 + 1 + 2 + 3 + 4 + ... ≥ 2^{n-1}, since the sequence grows much slower than the fibonacci sequence, we are not able to ensure all the peers obtain the file in less than $2x$ time units.

Comments: