Q1(a)

Answer: The Transport layer is responsible for process-to-process delivery.

Difference from host-to-host delivery: The Network layer (IP) delivers packets host-to-host using IP addresses and routing. The Transport layer delivers between processes on those hosts using port numbers, and may provide reliability, ordering, flow control and congestion control (TCP) or best-effort datagrams (UDP).

Q1(b)
Answer: DNS differentiates services by record type and the client's protocol/port. Example workflow:

For web: client resolves www.example.com via CNAME records and connects to port 80/443.
For mail: senders look up the domain's MX record (e.g. example.com MX 10 mail.example.com), then resolve that MX host via CNAME and connect to SMTP ports (25, 587).
Even if both hostnames resolve to the same IP, the DNS record type (MX vs A/CNAME) and the port/protocol separate mail and web services.

Q1(c) Answer:

(I) Yes. Client G can download from the current top uploaders.It is connected to three of the top four (B, E, A).

(II) If D's upload increases to 40 Mbps and becomes selected,

After D increases:

Top-4 now: B (100), D (40), E (15), F (14).

Total aggregate upload U,

100+40=140 ,100 + 40 = 140, 100+40=140.
140+15=155,

155+14=169.

So, U=169 Mbps.

File size S = 890 megabits.

T=S÷U

Download Time, T = 890/169 =5.266 s

Q1(d)

Answer:

Benefits of DASH: adaptive bitrate switching (reduces rebuffering, improves QoE), uses standard HTTP/CDN infrastructure (cacheable segments), efficient bandwidth utilization, supports live and on-demand streaming with smooth quality switching.

Manifest (MPD) purpose: The manifest (Media Presentation Description) lists available representations (bitrate/resolution/codecs), segment URLs and their timing/duration, and adaptation sets. The client reads the manifest to choose which segments/bitrates to request and when to switch, it is the roadmap for adaptive streaming.

Q2(a)

Answer :Web browsers store cookies, localStorage, and other browsing state in browser-specific profiles. Chrome and Firefox maintain separate storage; data written by Chrome are not accessible to Firefox.Therefore searches for site data saved while using Chrome will not appear when you later open the same site in Firefox, because Firefox has its own cookie/localStorage store. Also cookies could have expired, been set with restrictive SameSite/path attributes, or the site could store data server-side tied to a login.

Q2(b)

Answer:Proxy reuses cached data via a conditional GET request (If-Modified-Since or If-None-Match).

The proxy asks the origin server “Has this resource changed since the date. If the resource has not changed, the server returns 304 Not Modified. The proxy reuses its cached body and forwards it to the client, avoiding transfer of the full object.
Alternative / related method: HEAD returns only the response headers. A proxy can use HEAD to check headers without downloading the object body. HEAD saves bandwidth because the server does not send the body but conditional GET is the common cache-validation mechanism that still leverages GET semantics and allows a 304 response.
How it saves time: On 304 Not Modified , the server sends only headers, not the full response body. This reduces network bandwidth and latency because the large object body does not need to be re-transmitted from origin to proxy. That reduces overall response time and server/load on the network.

Q2(c) Answer:

Connection is persistent; object requests are sent sequentially.
(I) Total RTT to fetch the IP address (DNS):

Iterative lookup typically requires 3 sequential RTTs.
Each RTT = 19 ms

total DNS = 3×19= 57ms.

(II) Total time to fetch all objects after the IP address has been retrieved

Interpretation of TCP timing:

one full RTT=2×35ms=70ms.

per-object time = 35+5+125 ms = 165 ms.

Total objects = 21

Total object time = 21×165ms= 3465 ms.
Total after IP retrieval = 70+3465=3535 ms.

(III) Total time to load the webpage

DNS = 57 ms (from part I)
Fetch all objects after IP = 3535 ms (from part II)
Total = 57+3535=3592 ms.

Q3(a) Answer:

The UDP checksum provides an end-to-end integrity check over the UDP pseudo-header, UDP header, and UDP data.
It detects bit errors that may occur during transmission:The pseudo-header includes the source IP, destination IP, protocol number, and UDP length — ensuring the packet hasn't been misdelivered.
In IPv4, the checksum is optional.
In IPv6, it is mandatory.
The checksum detects corrupted data but does not correct it.

Q3(b)

Answer: TCP Urgent Data
Urgent pointer value = 700
Urgent data range = bytes 3001 to 3700

URG = 1
Urgent pointer = 700

This tells PCB that bytes up to sequence 3700 are urgent.

Q3(c) Answer:

(I) Sequence & ACK for Client's HTTP Request 2

Client’s first data byte after SYN = 2045 + 1 = 2046

After Request 1 (320 bytes), next seq = 2046 + 320 = 2366

Server data after DS1–DS3 = 952 + 378 + 455 = 1785 bytes

Server first data byte after SYN = 8935 + 1 = 8936

Next expected byte from server = 8936 + 1785 = 10721

Sequence = 2366

Acknowledgment = 10721

(II)

Total server bytes received (DS1–DS5) = 952 + 378 + 455 + 300 + 99 = 2184 bytes

Client buffer = 3020 bytes

While data is buffered: available space = 3020 − 2184 = 836 bytes

After data is delivered to the application: buffer empties, rwnd = 3020 bytes

(III) Sequence & ACK After Client Receives DS-4

After sending HTTP Request 2 (389 bytes): next seq = 2366 + 389 = 2755

Server bytes after DS1–DS4 = 952 + 378 + 455 + 300 = 2085

Next expected server byte = 8936 + 2085 = 11021

Client next sequence number = 2755

Acknowledgment number = 11021