CS335a: Computer Networks

Professor: Maria Papadopouli
TA: Giorgos Mellios (csdp1395@csd.uoc.gr)
Deadline: 05/11/2025

SUBMISSION GUIDELINES

- **Your report should be in PDF format.**
- **Please submit your assignment via the e-learn platform.**
- **The maximum grade you can get is 120 with 20 out of the 120 points being BONUS.**
- **To prevent plagiarism, in each assignment series a random sample of students will be selected for further oral examination.**

Assignment 2: Application Layer

Exercise 1 (25pts + 5pts)

i) *(3p)* Explain the main differences between client-server and peer-to-peer architectures. Provide one modern real-world example of each.

ii) *(3p)* What are the main benefits and drawbacks of running applications over UDP instead of TCP? Mention at least two examples of real applications that use UDP and explain why.

iii) *(3p)* Describe the term application-layer protocol. What are the key elements it defines, and how do they enable interoperability among systems?

iv) Imagine you are designing a real-time multiplayer educational platform (for example, an interactive quiz, collaborative lab, or simulation game) where multiple users interact simultaneously over the Internet.

1. *(4p)* Specify the type of transport service your application would require in terms of:
    - Reliability

- Timing / delay tolerance
- Throughput
- Security

Justify each requirement based on the expected user experience.

2. *(2p)* Choose an appropriate transport protocol (e.g., TCP, UDP, or a hybrid approach such as UDP with application-level reliability).
   Explain your choice in terms of performance, scalability, and interactivity.

3. *(2p)* Propose one application-layer feature that would improve user experience under varying network conditions (e.g., packet loss, latency spikes, or limited bandwidth).
   Describe how this feature adapts to changing network performance.

4. *(+5 bonus points)* Suggest a novel or non-trivial feature that distinguishes your system from existing real-time platforms (such as Kahoot, Quizizz, or multiplayer online games).
   Your innovation may relate to:

   - Network protocol design,
   - Synchronization and fairness mechanisms,
   - Adaptive content delivery,
   - Real-time feedback, or
   - Cross-platform optimization.

v) ***(8p)*** Compare **DNS**, **SMTP**, and **HTTP** in terms of:

1. Type of communication (push/pull)
2. Statefulness
3. Typical transport protocol used
4. Default port numbers

Indicative Answers:

i.

Client-Server:

- **Architecture**: Relies on an always-on host (server) that provides services to many clients. The server has a fixed, known IP address.
- **Scalability**: Can be a bottleneck. A single server must handle all client requests.
- **Example**: The World Wide Web (HTTP). Your browser (client) requests a webpage from a web server.

Peer-to-Peer (P2P):

- **Architecture**: Uses direct communication between intermittently connected hosts, called peers. It does not rely on a single, always-on server.
- **Scalability**: Highly scalable. As more peers join, the total capacity of the system increases (peers are both clients and servers).
- **Example**: BitTorrent. Peers download file chunks from each other simultaneously.

ii.

Benefits:

- **Low Overhead**: No connection setup (no 3-way handshake).
- **Speed**: No congestion or flow control mechanisms, so data can be sent as fast as desired.

Drawbacks:

- **Unreliable**: No guarantee of delivery.
- **Unordered**: Packets may arrive out of sequence.

Real-World Examples:

- **DNS**: A fast, simple request-response is needed. The overhead of a TCP connection is unnecessary. If a query packet is lost, the client simply times out and resends it.
- **Real-time Video/Audio Streaming (VoIP)**: Speed is more important than perfect reliability. It is better to skip a lost packet (a minor glitch) than to wait for a retransmission, which would cause noticeable lag.

iii.

Definition: An application-layer protocol defines the "language" that applications use to communicate with each other over a network. It dictates the rules for how messages are exchanged between end systems.

Key Elements Defined:

- **Message Types**: e.g., Request messages and Response messages.
- **Message Syntax**: The exact fields in a message, their order, and how they are delineated.
- **Message Semantics**: The meaning and interpretation of the information in each field.
- **Rules**: When and how a process should send and respond to messages.

Interoperability: By standardizing these rules (e.g., in an RFC), developers can build different software (like a Chrome browser and an Apache web server) on different operating systems, and they will be able to communicate ("interoperate") perfectly.

iv.

1. Transport Service Requirements :

- **Reliability:** Mixed. Critical data like submitting a final answer, user login, or game-state changes (e.g., "User A buzzed in") must be reliable. Non-critical updates like real-time cursor movements can tolerate some loss.
- **Timing / Delay:** Low latency is critical. High delay ruins the "real-time" aspect of an interactive quiz or simulation, creating an unfair and frustrating user experience.
- **Throughput:** Moderate. Does not need massive throughput like video streaming, but requires enough bandwidth to handle frequent, small updates from all users simultaneously.
- **Security:** High. Must use encryption (like TLS) to protect user data (passwords, scores) and prevent cheating (e.g., intercepting the correct answer).

2. Transport Protocol Choice:

- **Choice:** A hybrid approach, using both TCP and UDP.
- **Justification:**
    - **TCP:** Use for all critical, state-changing actions: user login, loading the quiz questions, and submitting final answers. This guarantees reliable delivery.
    - **UDP:** Use for all real-time, high-frequency, loss-tolerant updates: seeing other players' cursors move, live "typing..." indicators, or voice chat. This minimizes latency for interactivity.

3. Application-Layer Feature:

- **Feature:** Adaptive Rate Control / Dynamic Prioritization.
- **Description:** The application actively monitors network conditions (packet loss, RTT).
    - **Under poor conditions (high lag):** The app automatically reduces the frequency of low-priority updates (e.g., sends avatar positions 5 times/sec instead of 30).
    - **Under good conditions:** It provides the full, high-fidelity experience. This ensures that critical messages (like "I buzz in") are prioritized, maintaining core functionality even on bad networks.

v.

Protocol	Communication Type	Statefulness	Typical Transport	Default Port(s)
DNS	Pull (Client requests/pulls a record)	Stateless (Each query is independent)	UDP (primarily); TCP (for zone transfers)	53
SMTP	Push (Client pushes email to the server)	Stateful (Commands must be in sequence)	TCP	25 (server-to-server), 587 (client submission)
HTTP	Pull (Client requests/pulls objects)	Stateless (Protocol is stateless; cookies add state at the app-level)	TCP	80 (HTTP), 443 (HTTPS)

Exercise 2 (20pts)

i) *(3p)* Explain the difference between **non-persistent** and **persistent** HTTP connections. How many TCP connections are typically established when a page with 10 embedded objects is loaded under each approach?

ii) *(4p)* Describe how **pipelining** in HTTP/1.1 reduces response time compared to standard persistent HTTP. What limitations caused browsers to eventually disable pipelining by default?

iii) *(5p)* Consider an institutional network using a web proxy cache.

1. Explain how **proxy caching** reduces overall response time and bandwidth consumption.
2. What HTTP headers are used by clients and servers to control caching behavior?
3. How can a cache validate that its stored copy of an object is still fresh?

iv) *(4p)* Explain how **cookies** allow HTTP servers to maintain state across multiple interactions of the client
Include in your answer the four components that are involved in cookie-based state management.

v) *(4p)* A user reports that some pages load outdated versions of content even though they were updated on the server.

1. Identify and explain one HTTP mechanism that could help prevent this issue.
2. Include an example of an HTTP header used for that purpose.

Indicative Answers:

i.

Non-persistent: A new TCP connection is established for every single object (the base HTML file and each embedded object).
Persistent: A single TCP connection is opened and reused to download the base HTML and all embedded objects.
Connections for 10 objects:

- **Non-persistent:** 11 TCP connections (1 for the base HTML + 10 for the objects).
- **Persistent:** 1 TCP connection (it is reused for all 11 requests).

ii.

- **How it works:** With a persistent connection, pipelining allows the client to send multiple requests back-to-back without waiting for the first response to arrive. The server must send the responses back in the order the requests were received.
- **Benefit:** It reduces RTT-induced delays, as the client isn't idle waiting for a response before sending the next request.

- **Limitations:** It suffered from Head-of-Line (HOL) blocking. If the first request in the "pipeline" was for a large or slow-to-generate object, it blocked all other responses (even small, ready ones) behind it. This, and poor proxy support, led to it being disabled.

iii. Web Proxy Caching

1. **Benefits:**
    - Reduces Response Time: If a requested object is in the proxy's cache (which is close to the client), the client gets it almost instantly, avoiding the RTT to the origin server.
    - Reduces Bandwidth: If 100 users request the same object, the proxy downloads it once from the origin, saving 99 downloads worth of institutional bandwidth.
2. **HTTP Headers:**
    - Cache-Control: The primary header used by both client and server to define caching rules (e.g., max-age, no-cache, private).
    - Expires: An older header from the server specifying when a resource becomes stale.
3. **Validation**: The cache uses a conditional GET. It sends the client's request to the origin server but adds an If-Modified-Since header (with the date of its cached copy). If the object has not changed, the server replies with 304 Not Modified (an empty body). If it has changed, the server sends 200 OK with the new object.

iv. Cookies and HTTP State

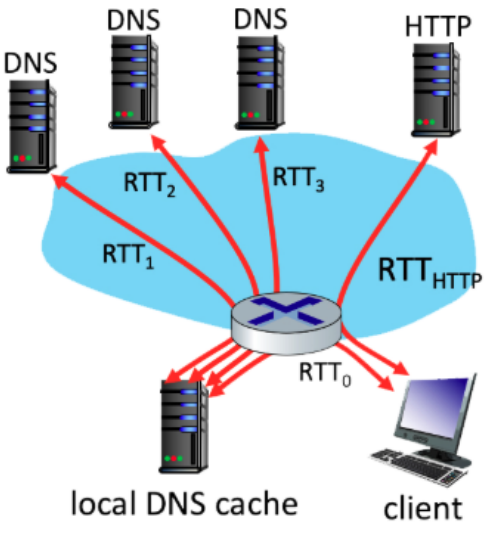
Cookies allow stateless HTTP to maintain state (like a shopping cart or login session).

- **Four Components:**
    1. **Server Response:** The server includes a Set-Cookie: header in its response to the client, assigning a unique ID.
    2. **Client Storage:** The client's browser stores this cookie on the user's local machine, associating it with the server's domain.
    3. **Client Request:** On every subsequent request to that same server, the browser includes a Cookie: header with the stored ID.
    4. **Server Database:** The server uses this ID to look up the user's state (e.g., shopping cart contents) in its backend database.

v. Outdated Content

1. **Mechanism:** This is a caching issue. The client (or a proxy) is serving a stale version of the content. The mechanism to fix this is Cache Validation, where the server provides a way for caches to check if their copy is fresh before using it.
2. **Header Example:**
    - **ETag (Entity Tag):** The server provides a "version hash" for the file (e.g., ETag: "abc-123"). The cache must check this ETag (using an If-None-Match request header) before serving its copy. If the ETag on the server is different, the cache knows its copy is stale and must download the new one.
    - (Alternatively: **Cache-Control**: no-cache forces validation on every request).

Exercise 3 (20pts)



Suppose within your Web browser you click on a link to obtain a Web page. The IP address for the associated URL is not cached in your local host, so a DNS lookup is necessary to obtain the IP address. Suppose that four DNS servers are visited before your host receives the IP address from DNS. The first DNS server visited is the local DNS cache, with an RTT delay of RTT_0 = 2 msecs. The second, third and fourth DNS servers contacted have RTTs of 17, 10, and 27 msecs, respectively. Initially, let's suppose that the Web page associated with the link contains exactly one object, consisting of a small amount of HTML text. Suppose the RTT between the local host and the Web server containing the object is RTT_HTTP = 87 msecs.

i) *(4p)* Assuming zero transmission time for the HTML object, how much time (in msec) elapses from when the client clicks on the link until the client receives the object?

ii) *(4p)* Now suppose the HTML object references 8 **very small objects** on the same server. Neglecting transmission times, how much time (in msec) elapses from when the client clicks on the link until the base object and all 8 additional objects are received from the web server at the client, assuming **non-persistent** HTTP and **no parallel** TCP connections?

iii) *(4p)* Suppose the HTML object references 8 very small objects on the same server, but assume that the client is configured to support a maximum of 5 parallel TCP connections, with **non-persistent HTTP**.

iv) *(4p)* Suppose the HTML object references 8 very small objects on the same server, but assume that the client is configured to support a maximum of 5 parallel TCP connections, with **persistent HTTP**.

v) *(4p)* What are the differences between these methods:

1. Non-Persistent HTTP (without parallel connections)
2. Persistent HTTP without pipelining
3. Persistent HTTP with pipelining

Indicative Answers:

i. 1 Object, Non-persistent
Total Time = (Total DNS) + (TCP Handshake) + (HTTP Req/Resp)
Total Time = 56 ms + (1 * RTT_HTTP) + (1 * RTT_HTTP)
Total Time = 56 + 87 + 87 = 230 ms

ii. 8 Objects, Non-persistent, No parallel
Total Time = (Total DNS) + (Base HTML) + (8 objects)

- DNS = 56 ms
- Base HTML = 2 * RTT_HTTP = 174 ms
- 8 Objects = 8 * (2 * RTT_HTTP) = 8 * 174 = 1392 ms

Total Time = 56 + 174 + 1392 = 1622 ms

iii. 8 Objects, Non-persistent, 5 parallel

Total Time = (Total DNS) + (Base HTML) + (8 objects in parallel)
DNS = 56 ms
Base HTML = 2 * RTT_HTTP = 174 ms
8 Objects (5 parallel):
    Batch 1 (Objects 1-5): 2 * RTT_HTTP = 174 ms
    Batch 2 (Objects 6-8): 2 * RTT_HTTP = 174 ms
Total Time = 56 + 174 + 174 + 174 = 578 ms

iv.

8 Objects, Persistent, 5 parallel
Total Time = (Total DNS) + (Parallel TCP Setups) + (Object Batches)
Total 9 objects (1 base + 8 embedded).
DNS = 56 ms
Open parallel TCP connections: 1 * RTT_HTTP = 87 ms
Batch 1 (Req/Resp for Base HTML + 4 objects): Base HTML + 1 * RTT_HTTP = 87 + 87 = 174 ms
Batch 2 (Req/Resp for remaining 4 objects, reusing 4 connections): 1 * RTT_HTTP = 87 ms
Total Time = 56 + 87 + 174 + 87 = 404 ms

v.

- **Non-Persistent (no parallel):** Very slow. Requires 2 * RTT_per_object (1 for TCP setup, 1 for req/resp).
- **Persistent (no pipelining):** Faster. Pays the 1 * RTT_TCP setup cost only once. Then, it's 1 * RTT (for req/resp) per object, sent sequentially.
- **Persistent (with pipelining):** Fastest (in RTTs). Pays 1 * RTT for TCP setup, then can send all requests at once. If N objects, it's roughly 1 * RTT_setup + N * (small req time) + 1 * RTT (for all responses). Its main drawback is **Head-of-Line (HOL)** blocking.

Exercise 4 (20pts)

i) *(5p)* Suppose a client at **student.csd.uoc.gr** needs to resolve **www.harvard.edu**. List in order, all DNS servers contacted during an iterative resolution process and describe what each returns.

ii) *(5p)* Now assume the same query is done using recursive resolution. What changes?

iii) *(5p)* Explain DNS caching and how stale entries can affect performance or correctness.

iv) *(5p)* Explain how DNS-based content delivery enables users to be directed to geographically closer servers. Reference how CDNs like Akamai use DNS for performance optimization.

Indicative Answers:

i. Iterative Resolution

In iterative resolution, the local resolver does all the work:

1. Client asks Local DNS Server (e.g., dns.csd.uoc.gr): "**Where is www.harvard.edu?**"
2. Local Server asks Root Server: "**Where is .edu?**"
    - Root Server replies: "**I don't know, ask the .edu TLD server at IP A.**"
3. Local Server asks .edu TLD Server (at IP A): "**Where is harvard.edu?**"
    - TLD Server replies: "**I don't know, ask the harvard.edu authoritative server at IP B.**"
4. Local Server asks harvard.edu Auth. Server (at IP B): "**Where is www.harvard.edu?**"
    - Auth. Server replies*:* "**The IP is 1.2.3.4**"
5. Local Server replies to Client: "**www.harvard.edu is at 1.2.3.4**"

ii. Recursive Resolution

The key change is that the client only talks to its local server and expects a final answer.

1. Client asks Local DNS Server: "**Where is www.harvard.edu? Please find this recursively.**"
2. The Local DNS Server then performs the entire iterative process (**steps 2-4 from part i**) on the client's behalf.
3. Local DNS Server returns only the final answer (1.2.3.4) to the client.

The client is not bothered with referrals; the local server does all the work.

iii. DNS Caching

- **Explanation**: When a DNS resolver learns a mapping (e.g., www.uoc.gr → 1.2.3.4), it stores (caches) this entry for a period of time called the **TTL** (Time To Live), which is set by the authoritative server.
- **Stale Entries**: If the IP for www.uoc.gr changes, but other resolvers still have the old entry in their cache (because the **TTL** hasn't expired), they will continue to send users to the wrong, old IP address. This breaks (correctness) or degrades (performance) the service.

iv. DNS-based Content Delivery (CDNs)

- **Explanation:** Content Delivery Networks (CDNs) distribute content (like videos or images) to many servers geographically around the world.
- **How DNS is used:** When a user requests a resource (e.g., video.cdn.com), the CDN's authoritative DNS server intelligently answers the query. It detects the geographic location of the user's querying DNS resolver and returns the IP address of the CDN server that is physically closest to that user.
- **Example (Akamai):** A user in Athens querying for a Netflix video (hosted on Akamai) will get the IP of an Akamai server in Athens, not one in California. This drastically reduces latency and improves streaming performance.

Exercise 5 (15pts)

i) *(3p)* Outline the role of each of the following components in email delivery:

1. User Agent (UA)
2. Mail Server
3. Mail Access Protocol

ii) *(4p)* Describe the steps involved when Alice (alice@csd.uoc.gr) sends an email to Bob (bob@mit.edu) using SMTP.

iii) *(4p)* Compare SMTP and HTTP in terms of:

1. Direction of data transfer (push vs pull)
2. Connection persistence
3. Message format

iv) *(4p)* IMAP allows messages to remain on the server. List two advantages and one drawback of this design.

Indicative Answers:

i. Email Components

1. **User Agent (UA):** The client software used to read, compose, and manage email (e.g., Outlook, Thunderbird, Gmail web interface).
2. **Mail Server:** The "post office." It stores received mail in user mailboxes and has an outgoing queue to send messages to other mail servers using SMTP.
3. **Mail Access Protocol:** The protocol a UA uses to retrieve messages from its mail server (e.g., IMAP, POP3).

ii. Email Delivery Steps (Alice to Bob)

1. Alice (UA) composes her email and instructs her mail server (mail.csd.uoc.gr) to send it (using SMTP).
2. Alice's mail server (mail.csd.uoc.gr) performs a DNS query for the MX (Mail Exchange) record of mit.edu to find the IP address of Bob's mail server (e.g., mail.mit.edu).
3. Alice's server opens a TCP connection to Bob's server (mail.mit.edu) on port 25.
4. Alice's server pushes the email to Bob's server using a sequence of SMTP commands (e.g., HELO, MAIL FROM:, RCPT TO:, DATA).
5. Bob's server stores the message in Bob's mailbox.
6. Bob later uses his UA (e.g., Outlook) to connect to his server (mail.mit.edu) via IMAP or POP3 to download and read the message.

iii. SMTP vs. HTTP

1. **Direction**: SMTP is a push protocol (it pushes mail from a client/server to a server). HTTP is primarily a pull protocol (it pulls resources from a server to a client).
2. **Connection Persistence**: HTTP/1.1 uses persistent connections by default, reusing one TCP connection for many objects. SMTP uses a new connection for each email "transaction" (though one transaction can send to multiple recipients).
3. **Message Format**: SMTP messages must be in 7-bit ASCII. All binary data must be encoded (e.g., using MIME). HTTP allows unencoded binary data within its message body.

iv. IMAP

IMAP (**Internet Message Access Protocol**) keeps messages on the server.

- **Advantages**:
    1. **Synchronization**: All folders and read/unread status are synchronized across all devices (phone, laptop, web).
    2. **Accessibility**: Emails are accessible from any device with an internet connection, as they aren't tied to a single machine.
- **Drawback**: Offline Access: It is "online-by-default." Accessing old emails requires an active internet connection (unless the client is configured to cache local copies).

Exercise 6 (BONUS 15pts)

In this exercise, you will use the command-line tool **dig (Domain Information Groper)** to explore how the **Domain Name System (DNS)** operates in practice. **dig,** provides detailed information about DNS queries, responses, and authoritative name servers.

You can perform all commands on the department's Linux machines or any Unix-based system.

i) Run the following command:

dig www.google.com

1. *(1p)* What is the IP address returned for www.google.com?
2. *(1p)* Which DNS server answered your query? Was it authoritative or non-authoritative? Explain how you can tell.
3. *(1p)* Run the same command again immediately. Did the query time change? Why?
4. *(1p)* Briefly explain how DNS caching reduces latency in name resolution.

ii) Retrieve the MX (Mail Exchange) records for the University of Crete:

dig uoc.gr MX

1. *(1p)* List all returned mail servers and their priorities.
2. *(1p)* Which mail server has the highest preference (lowest number)?

Query for the NS (Name Server) records of the top-level domain .gr:

dig gr NS

3. *(1p)* How many TLD servers are listed?
4. *(1p)* What is the purpose of having multiple TLD servers?

Query for the A record of www.uoc.gr:

dig www.uoc.gr A

5. *(1p)* Identify the IP address returned and the Time To Live (TTL). What does TTL indicate?

iii)

Use the trace option to follow the full resolution path:

dig +trace www.google.com

Answer the following:

1. *(1p)* List the sequence of DNS servers contacted (root, TLD, authoritative).
2. *(1p)* Which server provides the final IP address for www.google.com?
3. *(1p)* How is iterative resolution visible in the output?
4. *(1p)* Explain why dig might only show a single response line on some networks, and how to fix this.

iv)

Run:

dig @8.8.8.8 www.csd.uoc.gr

1. *(1p)* What does @8.8.8.8 specify? Compare the response time with your local resolver.
2. *(1p)* Why might the results differ between your local resolver and Google's DNS in terms of authority and latency?

Indicative Answers:

i.

```
; <<>> DiG 9.20.15-1~deb13u1-Debian <<>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42522
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         300     IN      A       172.217.218.99
www.google.com.         300     IN      A       172.217.218.105
www.google.com.         300     IN      A       172.217.218.104
www.google.com.         300     IN      A       172.217.218.103
www.google.com.         300     IN      A       172.217.218.106
www.google.com.         300     IN      A       172.217.218.147

;; Query time: 60 msec
;; SERVER: 147.52.16.1#53(147.52.16.1) (UDP)
```

1. **IP Address:** The ANSWER SECTION shows multiple A records for www.google.com, including 172.217.218.99, 172.217.218.105, 172.217.218.104, 172.217.218.103, 172.217.218.106, and 172.217.218.147.
2. **Answering Server:** The SERVER: line at the bottom indicates the query was answered by 147.52.16.1. This is a **non-authoritative** answer, as the aa (Authoritative Answer) flag is missing from the flags: line in the header.

3. **Query Time:** The initial query time was 60 msec. When the command is run again immediately, the query time drops to near 0 msec. This is because the result is retrieved from the local resolver's cache instead of being fetched over the network.
4. **DNS Caching:** DNS caching reduces latency by storing recent DNS responses locally. When the same query is made again, the resolver can provide the answer from its cache almost instantly, avoiding the RTT delays of contacting the root, TLD, and authoritative servers.

ii.

```
; <<>> DiG 9.20.15-1~deb13u1-Debian <<>> uoc.gr MX
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32242
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;uoc.gr.                                IN      MX

;; ANSWER SECTION:
uoc.gr.                 600     IN      MX      50 mx01.uoc.gr.
uoc.gr.                 600     IN      MX      50 mx02.uoc.gr.

;; Query time: 8 msec
;; SERVER: 147.52.16.1#53(147.52.16.1) (UDP)
```

```
; <<>> DiG 9.20.15-1~deb13u1-Debian <<>> gr NS
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5615
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;gr.                            IN      NS

;; ANSWER SECTION:
gr.                     600     IN      NS      estia.ics.forth.gr.
gr.                     600     IN      NS      grdns.ics.forth.gr.
gr.                     600     IN      NS      gr-c.ics.forth.gr.
gr.                     600     IN      NS      gr-m.ics.forth.gr.
gr.                     600     IN      NS      gr-d.ics.forth.gr.
gr.                     600     IN      NS      gr-at.ics.forth.gr.

;; Query time: 20 msec
;; SERVER: 147.52.16.1#53(147.52.16.1) (UDP)
```

```
; <<>> DiG 9.20.15-1~deb13u1-Debian <<>> www.uoc.gr A
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62805
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.uoc.gr.                    IN      A

;; ANSWER SECTION:
www.uoc.gr.             600     IN      A       147.52.201.74

;; Query time: 8 msec
;; SERVER: 147.52.16.1#53(147.52.16.1) (UDP)
```

1. **dig uoc.gr MX**: The ANSWER SECTION lists two mail (MX) servers: mx01.uoc.gr. and mx02.uoc.gr.. Both are listed with a priority of 50.
2. **Highest Preference**: Both servers share the highest preference (lowest number), as both have a priority of 50. This is a common setup used for load balancing.
3. **dig gr NS**: The ANSWER SECTION lists six TLD Name Servers (NS) for the .gr domain, including estia.ics.forth.gr, grdns.ics.forth.gr, and gr-c.ics.forth.gr.
4. **Multiple TLD Servers**: The purpose of having multiple TLD servers is to provide redundancy and load balancing. If one server fails, others can handle the requests (fault tolerance), and the query load is distributed among them.
5. **dig www.uoc.gr A**: The ANSWER SECTION shows the IP address for www.uoc.gr is 147.52.201.74. The TTL (Time To Live) is 600 seconds. This TTL indicates that a resolver may cache this record for 600 seconds (10 minutes) before it must request a fresh copy from the authoritative server.

iii.

```
; <<>> DiG 9.10.6 <<>> +trace www.google.com
;; global options: +cmd
.                       486730  IN      NS      l.root-servers.net.
.                       486730  IN      NS      h.root-servers.net.
.                       486730  IN      NS      j.root-servers.net.
.                       486730  IN      NS      i.root-servers.net.
.                       486730  IN      NS      k.root-servers.net.
.                       486730  IN      NS      e.root-servers.net.
.                       486730  IN      NS      f.root-servers.net.
.                       486730  IN      NS      c.root-servers.net.
.                       486730  IN      NS      g.root-servers.net.
.                       486730  IN      NS      d.root-servers.net.
.                       486730  IN      NS      a.root-servers.net.
.                       486730  IN      NS      b.root-servers.net.
.                       486730  IN      NS      m.root-servers.net.
.                       486731  IN      RRSIG   NS 8 0 518400 20251124050000 202
51111040000 61809 . R9kC1ovDDzMbmNj4yZys8xowO4Vs/Ur8SmdL+P2V/m7OJB8AZZhBZJK1 xfi
u4s+8O0ntX3+vl3j/G1BNkoZ0bVLXuh7bnAuFj7/VXNvPJctEJp5m nbQIGktI80KcLe0OK9Sq+Hk4vq
Kq283VqkHJqxMl0l5cpwy+t8cV/Jju LOnVCmKluuYES2zVfseHTH8O/ewI34mNrgce2iiWI0If/EqDE
KxSe/wz 9M7cksUVvjm0mjECjL1XjtSVeJTUs7AuxkO1CysyAqvHwir/cJCTP2mF I5jyy/eRulq0FQZ
F1pRaFbm7zX2y3nxztqmsu49Rs0M/y3Qwq3MWY/P3 ioKIMw==
;; Received 525 bytes from 147.52.80.1#53(147.52.80.1) in 23 ms

com.                    172800  IN      NS      i.gtld-servers.net.
com.                    172800  IN      NS      l.gtld-servers.net.
com.                    172800  IN      NS      m.gtld-servers.net.
com.                    172800  IN      NS      d.gtld-servers.net.
com.                    172800  IN      NS      b.gtld-servers.net.
com.                    172800  IN      NS      h.gtld-servers.net.
com.                    172800  IN      NS      e.gtld-servers.net.
com.                    172800  IN      NS      g.gtld-servers.net.
com.                    172800  IN      NS      a.gtld-servers.net.
com.                    172800  IN      NS      f.gtld-servers.net.
com.                    172800  IN      NS      j.gtld-servers.net.
com.                    172800  IN      NS      k.gtld-servers.net.
com.                    172800  IN      NS      c.gtld-servers.net.
com.                    86400   IN      DS      19718 13 2 8ACBB0CD28F41250A80A4
91389424D341522D946B0DA0C0291F2D3D7 71D7805A
com.                    86400   IN      RRSIG   DS 8 1 86400 20251124170000 2025
1111160000 61809 . 0aHasShjNPW4I2N8yzTYH3U2MlczThWFmYQ29bkNLworhqyvBI8EBDU/ hQVK
NJoESGqls+wVNQ+Anj8psrpCxD9g0XdPB7dst2vHX6Rgtaf2aIMF COuHNRAOsNLyjjMZiG+uVqGK58E
eOytHVpO38Ksa9PdNNbdRw7UdczL1 spCNw6E5Q4tQ++lgOM1sGktPACP5DLMlR7u8fFp41ND89nLkIi
xItDHe M0GlQuDOHO+Cjkcw91vqrI5vtFFlWHPqvG6R7hDXCAkEgdG8SM+vkJjM oAi22eXNh461K0ig
aphbf5QHzkRpG2cvmKvlrWX22Bt4cEuSWmYYjmSs b2n5MA==
;; Received 1177 bytes from 193.0.14.129#53(k.root-servers.net) in 30 ms

google.com.             172800  IN      NS      ns2.google.com.
google.com.             172800  IN      NS      ns1.google.com.
google.com.             172800  IN      NS      ns3.google.com.
google.com.             172800  IN      NS      ns4.google.com.
CK0POJMG874LJREF7EFN8430QVIT8BSM.com. 900 IN NSEC3 1 1 0 - CK0Q3UDG8CEKKAE7RUKPG
CT1DVSSH8LL  NS SOA RRSIG DNSKEY NSEC3PARAM
CK0POJMG874LJREF7EFN8430QVIT8BSM.com. 900 IN RRSIG NSEC3 13 2 900 20251115002630
 20251107231630 46539 com. dxQ1aDzMstWomzJXQiHSR7VgWKs6hBNgpy7YlauK93WaXlKdi2pcZ
F5F tp8mSfqLDnAB4nX+TU5OrouEjdJj7A==
S84BOR4DK28HNHPLC218O483VOOOD5D8.com. 900 IN NSEC3 1 1 0 - S84BR9CIB2A20L3ETR1M2
415ENPP99L8  NS DS RRSIG
S84BOR4DK28HNHPLC218O483VOOOD5D8.com. 900 IN RRSIG NSEC3 13 2 900 20251116035935
 20251109024935 46539 com. ceLGLwGiBB1ed/ayz4Q22jCck7zxF/ru+xuzBz5CBcnOn63H1rJlx
I/l J6VcwQMBueFiR6kep52LTNV6elI4gA==
;; Received 648 bytes from 192.26.92.30#53(c.gtld-servers.net) in 174 ms

www.google.com.         300     IN      A       216.58.204.228
;; Received 59 bytes from 216.239.32.10#53(ns1.google.com) in 95 ms
```

- The sequence of server types contacted is:
    1. **Root Server:** The client first receives a response from k.root-servers.net..
    2. **TLD Server:** Following the referral from the root, the client receives a response from c.gtld-servers.net. (a TLD server for the .com domain).
    3. **Authoritative Server**: Following the TLD's referral, the client receives the final answer from ns1.google.com. (an authoritative name server for google.com).

2. The final IP address is provided by the authoritative server. In the last block of the output, the server ns1.google.com. provides the A record: www.google.com. 300 IN A 216.58.204.228.

3. The output itself demonstrates iterative resolution. The dig tool, acting as the client, performs the following steps visible in the trace:

    1. It first queries a root server, which does not provide the IP but refers the client to the .com TLD servers.
    2. The client then follows that referral and queries a TLD server, which also does not provide the IP but refers the client to the google.com authoritative servers.
    3. Finally, the client follows the second referral and queries the authoritative server, which provides the final answer. This step-by-step process of the client following referrals is the definition of iterative resolution.

4. This can occur when a transparent DNS proxy is operating on the network (often at an ISP or corporate level). This proxy intercepts all outbound DNS queries (on UDP port 53) and forces them to be recursive, even if dig +trace is attempting to perform an iterative query. The proxy intercepts the first query (to the root) and, instead of letting it pass, performs the full lookup itself and returns only the final answer to the dig client, resulting in what looks like a single response.

- Fix: Bypassing this is not always possible as it's a network-level interception. However, potential fixes include using a different protocol like DNS-over-HTTPS (DoH), which uses port 443 and would not be intercepted by a simple port 53 proxy.

iv.

```
; <<>> DiG 9.10.6 <<>> @8.8.8.8 www.csd.uoc.gr
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40443
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.csd.uoc.gr.                        IN      A

;; ANSWER SECTION:
www.csd.uoc.gr.         300     IN      CNAME   csd.uoc.gr.
csd.uoc.gr.             300     IN      A       147.52.16.73

;; Query time: 147 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
```

- **@8.8.8.8:** The @8.8.8.8 syntax specifies that dig should send its query directly to Google's public DNS resolver at 8.8.8.8, bypassing the system's default local resolver. The query time to 8.8.8.8 was 147 msec, which is significantly higher than the 8 msec query to the local resolver (147.52.16.1) for a similar uoc.gr domain. This is due to the local resolver being closer on the network (lower RTT).
- **Differences**: Latency differs due to network proximity; the local resolver (147.52.16.1) is on the same local network and thus much faster. Authority does not differ, as both Google's DNS and the local resolver are recursive resolvers and provide non-authoritative answers (the aa flag is not set). The results (the actual IPs) could differ if one resolver's cache is stale while the other has a fresh, updated record

For this exercise, provide screenshots of each command's output. Write short explanations (1–3 sentences per question). Make sure your answers clearly indicate which section and question they correspond to.