Department of Computer Science and Engineering,
Anna University, Chennai- 600025
CS23502- NETWORKS AND DATA COMMUNICATION
Practical
July- Oct 2026
Year/Sem/Batch : III/V/ N

Exercise: 5	Persistent & Non-persistent HTTP protocol	06- 08- 2026

Observation questions:

1. Explain, with a labeled timeline diagram, how a web page containing 3 embedded images is fetched under **non-persistent HTTP**. How many TCP connections are established in total, and why?
2. Explain how the same page (with 3 embedded images) is fetched under **persistent HTTP**. What role do the Connection: keep-alive and Connection: close headers play in deciding when the underlying TCP connection is closed?
3. Compare non-persistent and persistent HTTP in terms of the number of TCP 3-way handshakes required and the resulting total response/latency time for fetching a page with several embedded objects. Which one is more efficient and why?
4. Explain the working of an **HTTP caching proxy**. Describe, step by step, what the proxy does on a **cache miss** versus a **cache hit**, and how this reduces load on the origin server and improves response time for the client.
5. What is a **conditional GET** request? Explain the role of the If-Modified-Since / If-None-Match request headers and the 304 Not Modified response status code in ensuring that a proxy cache does not serve stale content.

Execution Questions (Programming Language - Java)

1. **Non-Persistent HTTP Server and Client** Write a Java **NonPersistentHTTPServer** that:
   - Listens on a fixed port using ServerSocket
   - Accepts one connection, reads the HTTP GET request line and headers
   - Sends back a valid HTTP response (status line, headers, and an HTML body) for the requested file
   - Closes the connection immediately after sending the response
   -

Write a Java **HTTPClient** that:

- Connects to the server, sends a GET request for a given file
- Reads and displays the response
- Repeats this process (opening a *new* connection each time) for every embedded object on the page
-

2. **Persistent HTTP Server and Client** Write a Java **PersistentHTTPServer** that:
   - Accepts a connection and services *multiple* sequential requests on the same socket using a loop
   - Reads the Connection header from the request; keeps the socket open if it is keep-alive
   - Closes the connection only when the client sends Connection: close or after an idle timeout

   Write a Java **PersistentHTTPClient** that:

   - Opens a single socket connection to the server
   - Sends multiple GET requests (for the main page and its embedded objects) over the same connection, one after another
   - Reads each response correctly using the Content-Length header to know where one response ends and the next begins
   -
3. **HTTP Proxy Server with Caching** Write a Java **HTTPProxyServer** that:
   - Listens for HTTP requests from a client (or browser)
   - Parses the requested URL and checks an in-memory (HashMap) or on-disk cache for the object
   - **On a cache miss:** forwards the request to the actual origin server (using Socket/URL/HttpURLConnection), stores the response in the cache, and returns it to the client
   - **On a cache hit:** serves the cached response directly to the client without contacting the origin server
   - Prints a log line for every request indicating HIT or MISS and the response time

   Write a simple **ProxyClient** (or configure a browser) that sends the same request multiple times through the proxy to demonstrate caching behavior. *Deliverable:* Source code + console log clearly showing cache MISS on the first request and cache HIT on subsequent identical requests.

Spot Questions

1. Force and demonstrate a 304 Not Modified response.

   On the spot, modify your proxy/server code so that a cached object is re-validated with a conditional GET (If-Modified-Since or If-None-Match). Run it against the origin server without changing the file, and show the resulting 304 Not Modified response in the console/log. Then modify the origin file and show that the same request now returns 200 OK with the updated content.

2. Convert a persistent connection to non-persistent (or vice versa) at runtime.

   On the spot, take your already-running persistent HTTP server/client pair and modify only the Connection header handling (client request header and/or server response logic) so that the connection now closes after a single request instead of being reused — or, if starting from non-persistent, modify it so multiple requests are now served over one socket. Re-run the client and demonstrate, via console output, that the connection behavior has changed as required.