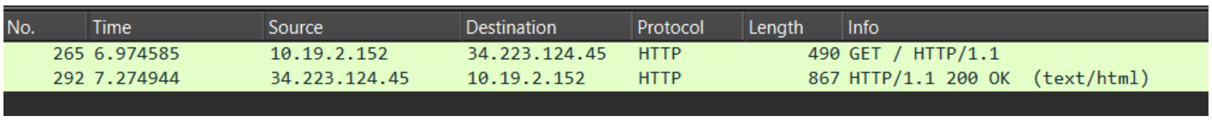
Computer Networks Lab (CS 342) - 2

Divyansh Chandak 220101039
Arush Shaleen Mathur 220101017
Udbhav Gupta 220101106
Tanay Goenka

Drive link for files of each task: files_to_be_submitted

Q1) HTTP using Wireshark



No.	Time	Source	Destination	Protocol	Length	Info
265	6.974585	10.19.2.152	34.223.124.45	HTTP	490	GET / HTTP/1.1
292	7.274944	34.223.124.45	10.19.2.152	HTTP	867	HTTP/1.1 200 OK (text/html)

a) Identify an HTTP Request

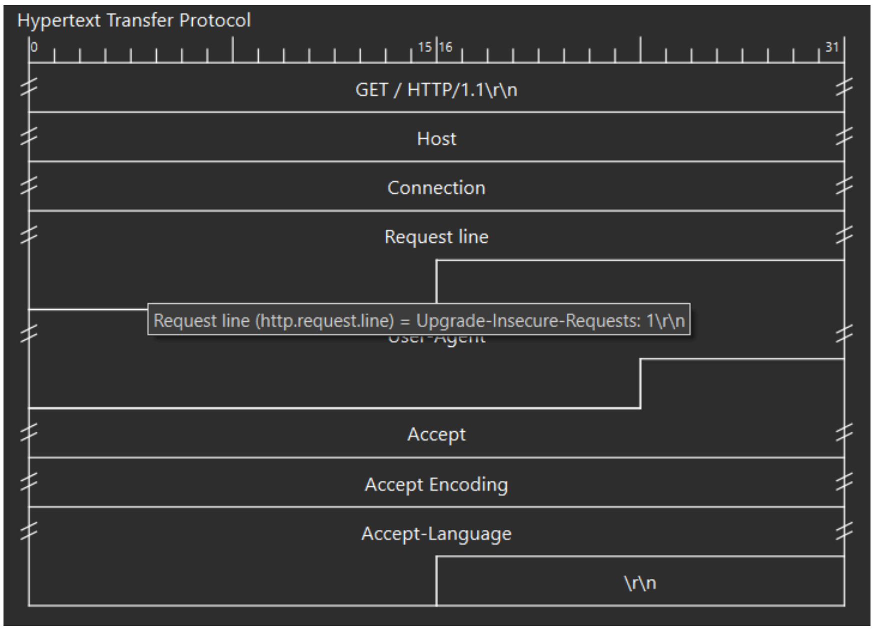
- HTTP Method: The request method is GET.
- URL of the Requested Resource: The requested URL is http://neverssl.com/.
- HTTP Version: The HTTP version used is HTTP/1.1 (which means it's a persistent http).

b) Analyze HTTP Request

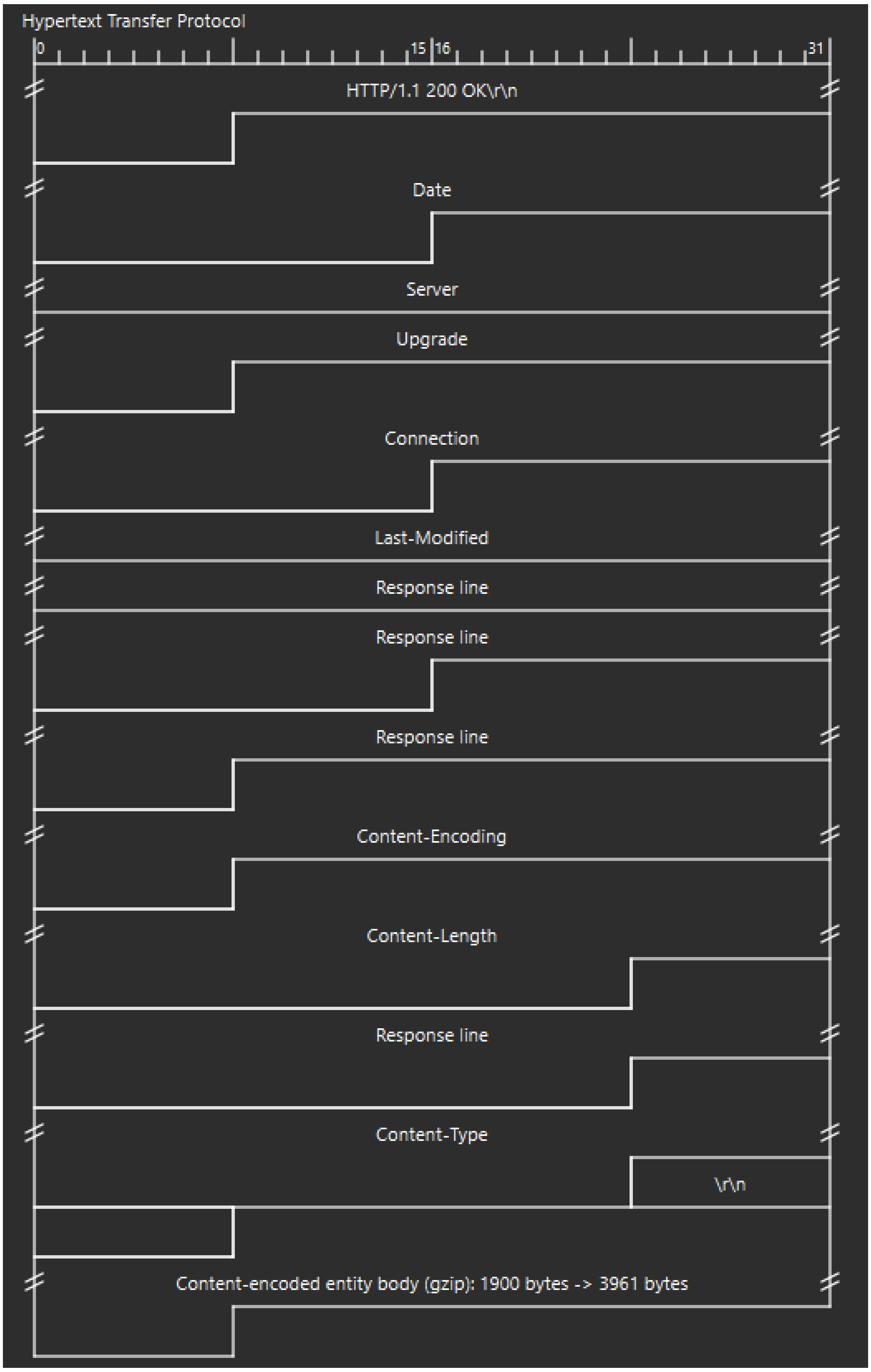
- Status Code: The status code returned by the server is 200 OK. 200 OK indicates that the request was successful, and the server returned the requested resource.

c) Inspect HTTP Headers

This is the Request header:



This is the response header:



Request Headers (from the first screenshot):

- Host: neverssl.com
- Connection: keep-alive
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.0.0 Safari/537.36
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
- Accept-Encoding: gzip, deflate
- Accept-Language: en-US,en;q=0.9,hi;q=0.8

Response Headers (from the second screenshot):

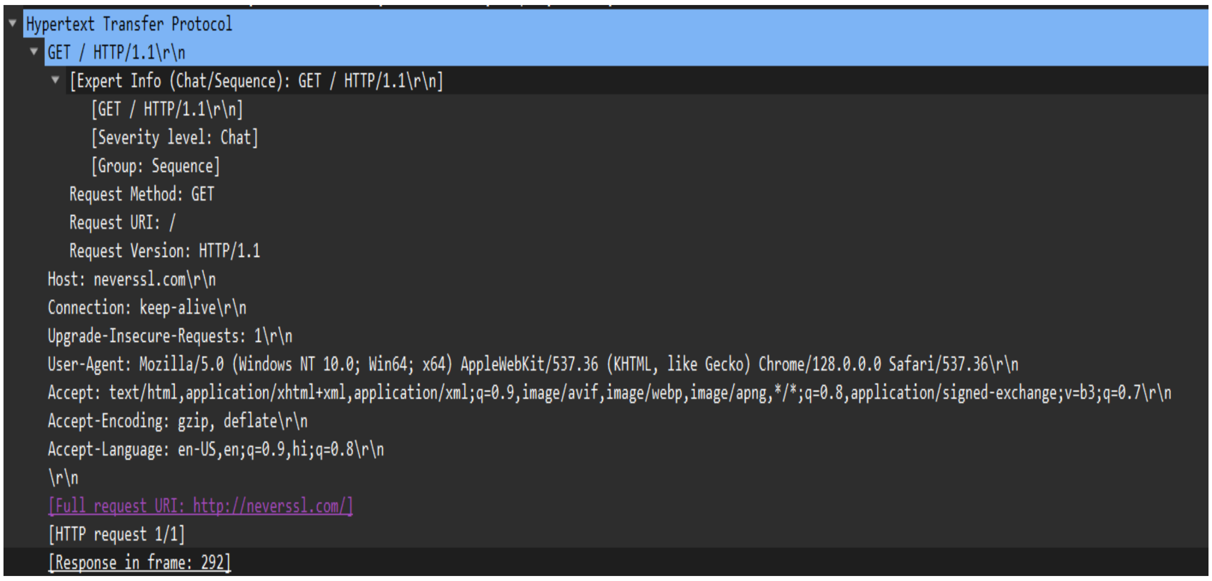
- Date: Fri, 30 Aug 2024 18:56:31 GMT
- Server: Apache/2.4.58 (Ubuntu)
- Upgrade: h2,h2c
- Connection: Upgrade, Keep-Alive
- Last-Modified: Wed, 29 Jun 2022 00:23:33 GMT
- ETag: "f79-5e28b29d38e93-gzip"
- Accept-Ranges: bytes
- Vary: Accept-Encoding
- Content-Encoding: gzip
- Content-Length: 1900
- Keep-Alive: timeout=5, max=100
- Content-Type: text/html; charset=UTF-8

Request Headers:

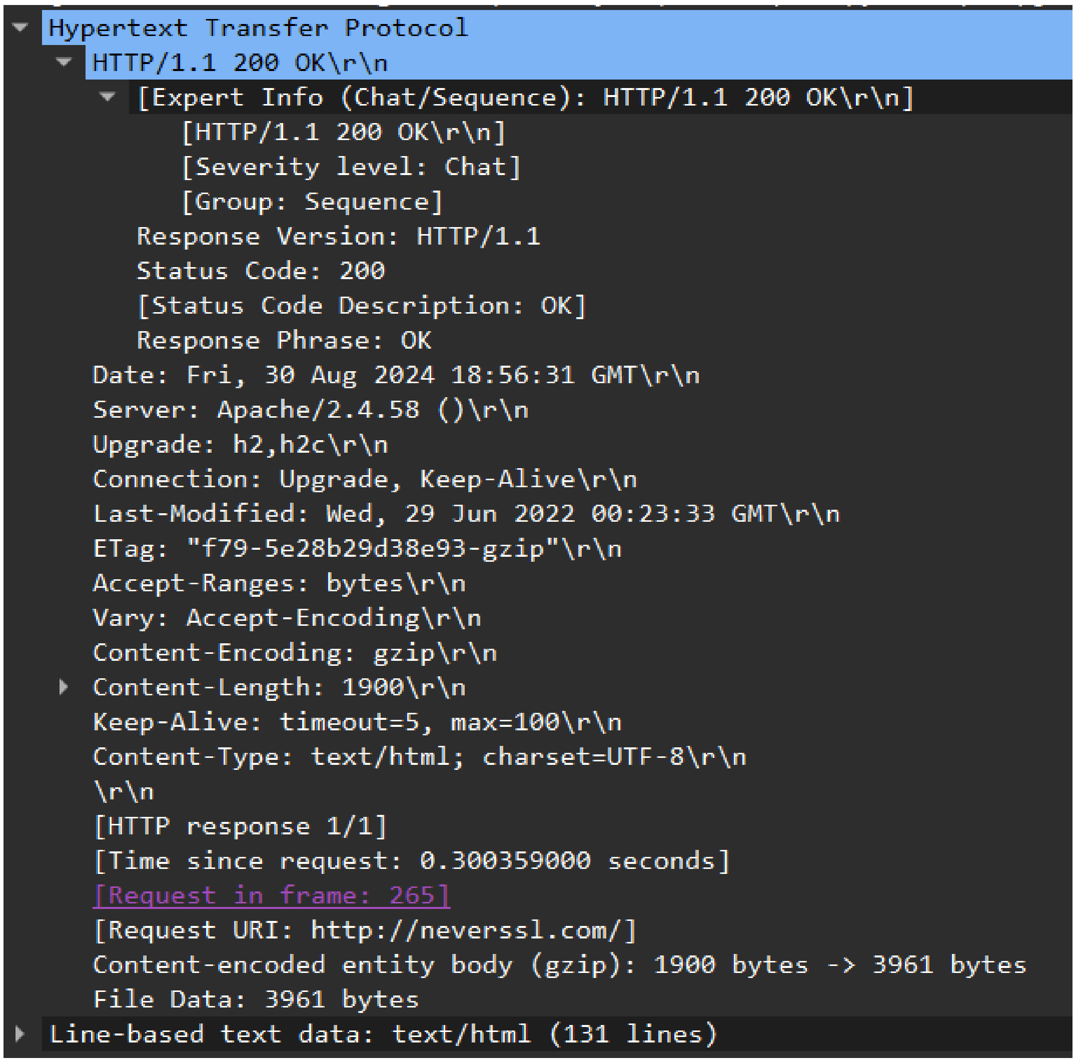
- Provide details about the client's preferences (e.g., content types, language, and encoding).
- The Host header specifies the domain of the server.
- The Connection header suggests keeping the connection open.

Response Headers:

- Provide details about the server's response, such as the date, server software, and the type of content being returned (text/html).
- The Content-Encoding header indicates that the content is compressed using gzip.
- The Content-Length header indicates the size of the response body in bytes.
- The Last-Modified header provides the last modification date of the resource.
- The ETag header provides a unique identifier for the resource, useful for caching.



```
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
        [GET / HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
    Request Method: GET
    Request URI: /
    Request Version: HTTP/1.1
  Host: neverssl.com\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.0.0 Safari/537.36\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en-US,en;q=0.9,hi;q=0.8\r\n
  \r\n
  [Full request URI: http://neverssl.com/]
  [HTTP request 1/1]
  [Response in frame: 292]
```



```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
        [HTTP/1.1 200 OK\r\n]
        [Severity level: Chat]
        [Group: Sequence]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  Date: Fri, 30 Aug 2024 18:56:31 GMT\r\n
  Server: Apache/2.4.58 ()\r\n
  Upgrade: h2,h2c\r\n
  Connection: Upgrade, Keep-Alive\r\n
  Last-Modified: Wed, 29 Jun 2022 00:23:33 GMT\r\n
  ETag: "f79-5e28b29d38e93-gzip"\r\n
  Accept-Ranges: bytes\r\n
  Vary: Accept-Encoding\r\n
  Content-Encoding: gzip\r\n
  Content-Length: 1900\r\n
  Keep-Alive: timeout=5, max=100\r\n
  Content-Type: text/html; charset=UTF-8\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.300359000 seconds]
  [Request in frame: 265]
  [Request URI: http://neverssl.com/]
  Content-encoded entity body (gzip): 1900 bytes -> 3961 bytes
  File Data: 3961 bytes
Line-based text data: text/html (131 lines)
```

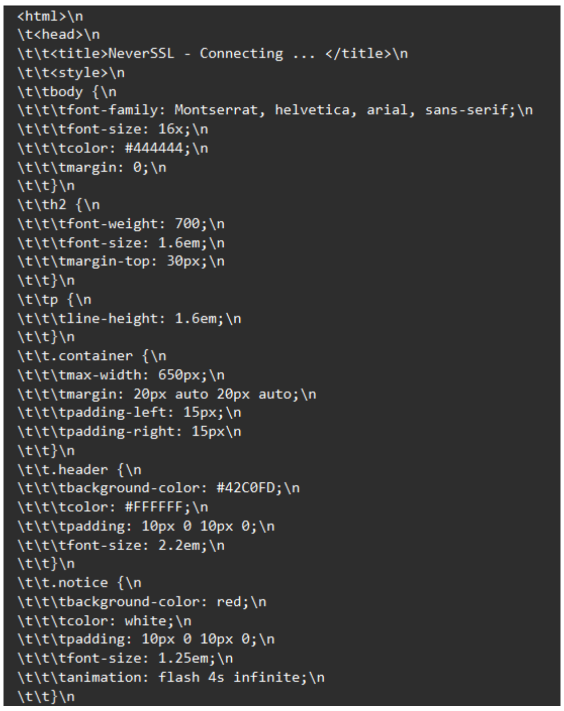
d) Calculate Response Time

- Response Time:
  - The request was sent at 6.974585 seconds.
  - The response was received at 7.274944 seconds.

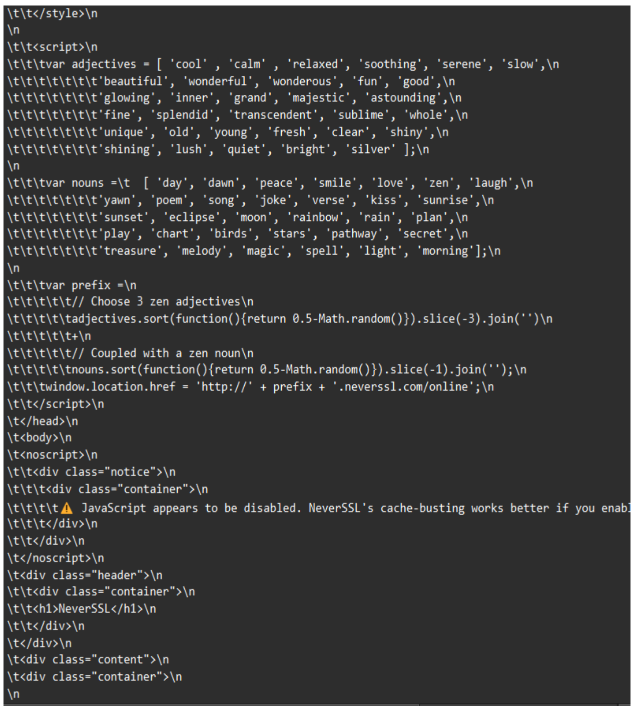
Response Time = 7.274944 s − 6.974585 s=0.300359 s = 300.359 ms (milliseconds).

e) Examine HTTP Content

The content type of the response is text/html; charset=UTF-8, meaning the server returned an HTML document. We observed that it contained 131 lines of HTML, below are some sample images:



```
<html>\n
\t<head>\n
\t\t<title>NeverSSL - Connecting ... </title>\n
\t\t<style>\n
\t\tbody {\n
\t\t\tfont-family: Montserrat, helvetica, arial, sans-serif;\n
\t\t\tfont-size: 16x;\n
\t\t\tcolor: #444444;\n
\t\t\tmargin: 0;\n
\t\t}\n
\t\th2 {\n
\t\t\tfont-weight: 700;\n
\t\t\tfont-size: 1.6em;\n
\t\t\tmargin-top: 30px;\n
\t\t}\n
\t\tp {\n
\t\t\tline-height: 1.6em;\n
\t\t}\n
\t\t.container {\n
\t\t\tmax-width: 650px;\n
\t\t\tmargin: 20px auto 20px auto;\n
\t\t\tpadding-left: 15px;\n
\t\t\tpadding-right: 15px\n
\t\t}\n
\t\t.header {\n
\t\t\tbackground-color: #42C0FD;\n
\t\t\tcolor: #FFFFFF;\n
\t\t\tpadding: 10px 0 10px 0;\n
\t\t\tfont-size: 2.2em;\n
\t\t}\n
\t\t.notice {\n
\t\t\tbackground-color: red;\n
\t\t\tcolor: white;\n
\t\t\tpadding: 10px 0 10px 0;\n
\t\t\tfont-size: 1.25em;\n
\t\t\tanimation: flash 4s infinite;\n
\t\t}\n
```



```
\t\t</style>\n
\n
\t\t<script>\n
\t\t\tvar adjectives = [ 'cool' , 'calm' , 'relaxed', 'soothing', 'serene', 'slow',\n
\t\t\t\t\t\t\t'beautiful', 'wonderful', 'wonderous', 'fun', 'good',\n
\t\t\t\t\t\t\t'glowing', 'inner', 'grand', 'majestic', 'astounding',\n
\t\t\t\t\t\t\t'fine', 'splendid', 'transcendent', 'sublime', 'whole',\n
\t\t\t\t\t\t\t'unique', 'old', 'young', 'fresh', 'clear', 'shiny',\n
\t\t\t\t\t\t\t'shining', 'lush', 'quiet', 'bright', 'silver' ];\n
\n
\t\t\tvar nouns =\t  [ 'day', 'dawn', 'peace', 'smile', 'love', 'zen', 'laugh',\n
\t\t\t\t\t\t\t'yawn', 'poem', 'song', 'joke', 'verse', 'kiss', 'sunrise',\n
\t\t\t\t\t\t\t'sunset', 'eclipse', 'moon', 'rainbow', 'rain', 'plan',\n
\t\t\t\t\t\t\t'play', 'chart', 'birds', 'stars', 'pathway', 'secret',\n
\t\t\t\t\t\t\t'treasure', 'melody', 'magic', 'spell', 'light', 'morning'];\n
\n
\t\t\tvar prefix =\n
\t\t\t\t\t// Choose 3 zen adjectives\n
\t\t\t\t\tadjectives.sort(function(){return 0.5-Math.random()}).slice(-3).join('')\n
\t\t\t\t\t+\n
\t\t\t\t\t// Coupled with a zen noun\n
\t\t\t\t\tnouns.sort(function(){return 0.5-Math.random()}).slice(-1).join('');\n
\t\t\twindow.location.href = 'http://' + prefix + '.neverssl.com/online';\n
\t\t</script>\n
\t</head>\n
\t<body>\n
\t<noscript>\n
\t\t<div class="notice">\n
\t\t\t<div class="container">\n
\t\t\t\t⚠ JavaScript appears to be disabled. NeverSSL's cache-busting works better if you enabl
\t\t\t</div>\n
\t\t</div>\n
\t</noscript>\n
\t<div class="header">\n
\t\t<div class="container">\n
\t\t<h1>NeverSSL</h1>\n
\t\t</div>\n
\t</div>\n
\t<div class="content">\n
\t<div class="container">\n
\n
```

Challenges encountered

While we were trying to capture some of the famous websites like flipkart, amazon, alibaba youtube, we could not find an HTTP request in the capturing process. We later realised that it was because:

1) These websites use HTTPS (Hypertext Transfer Protocol Secure) instead of plain HTTP. HTTPS encrypts the data exchanged between your browser and the server using TLS (Transport Layer Security). This encryption ensures that the content of the HTTP requests and responses cannot be easily read or intercepted by third parties, including tools like Wireshark.

2) As a result, when we captured traffic in Wireshark, we saw TLS packets instead of the plain HTTP traffic. The actual HTTP requests and responses are encrypted within these TLS packets, making them unreadable without decryption.

3) HTTPS is used to protect user privacy and secure data transmission, especially on sites that handle sensitive information like passwords, payment details, or personal data.

Q2) Web Server, HTTP Proxy with Caching, and DNS Lookup Simulation

Report: Web Server, HTTP Proxy with Caching, and DNS Lookup Simulation

Introduction

This report outlines the implementation of a simplified HTTP Proxy Server with a caching mechanism and simulates the DNS lookup process. The goal is to understand the intricacies of web communication, particularly how DNS lookups interact with HTTP requests and caching. The assignment involves simulating DNS lookups with random delays and failures and implementing a caching mechanism to store web pages for a proxy server, reducing retrieval times and improving overall performance.

1+2. DNS Lookup Process Explanation

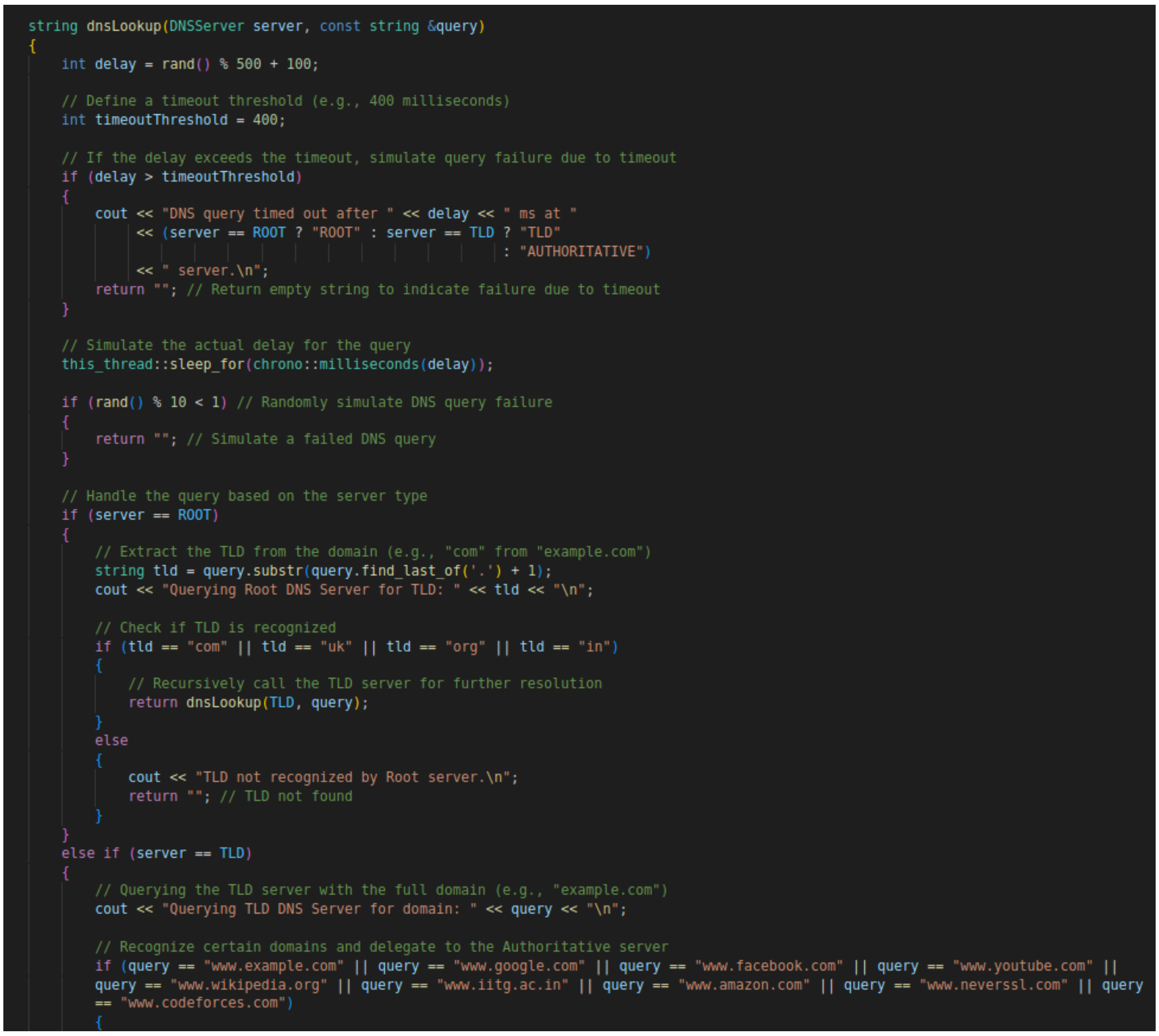
1. DNS Lookup Process When IP Address is Not Cached Locally

When the IP address of a requested URL is not cached locally, the DNS lookup process involves querying multiple DNS servers sequentially until the IP address is resolved. The steps are as follows:

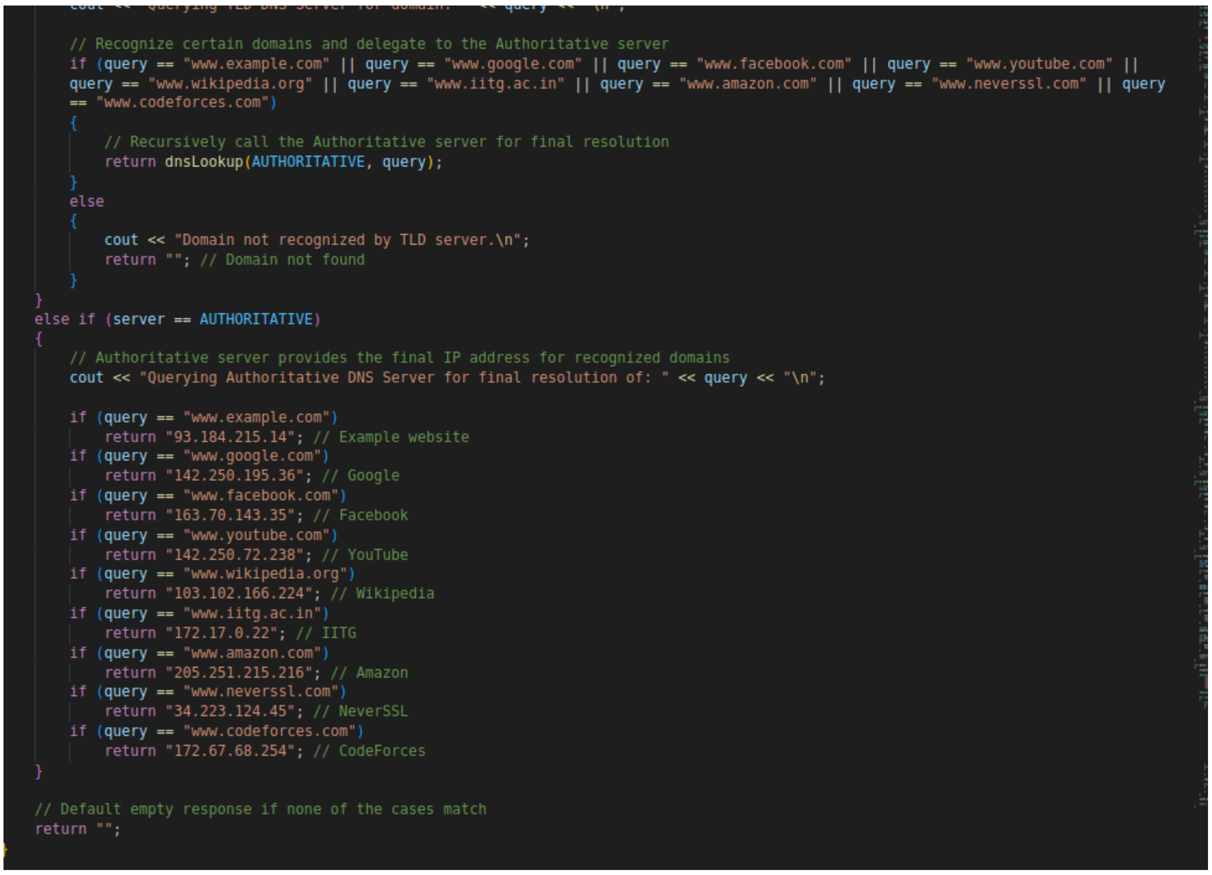
1. **Initial Query**:
   The browser first checks for the IP address in the local cache. If it is not found, it sends a DNS query to the local DNS resolver, typically provided by the ISP.
2. **Querying the Root DNS Server**:
   If the local DNS resolver does not have the IP address cached, it forwards the query to a root DNS server. The root server does not know the IP address but directs the resolver to the appropriate TLD DNS server.
3. **Contacting the TLD DNS Server**:
   The root DNS server directs the local DNS resolver to the correct TLD DNS server (e.g., .com, .org). The local resolver queries the TLD server.
4. **Finding the Authoritative DNS Server**:
   The TLD server provides a referral to the authoritative DNS server that holds the specific details for the domain being queried.
5. **Getting the IP Address from the Authoritative DNS Server**:
   The local DNS resolver queries the authoritative DNS server, which provides the IP address associated with the requested domain.
6. **Caching and Returning the IP Address**:
   The local DNS resolver caches the IP address for future use and returns it to the browser, which initiates the HTTP request to the web server.

2. Code Explanation for DNS Lookup Simulation

To simulate the DNS lookup process, the attached code represents the behaviour of each DNS server, introducing random delays or failures.The dnsLookup function introduces random delays and checks if the delay exceeds a timeout threshold. If so, it simulates a DNS server failure. The code will try DNS lookup with backup server in case of timeout or failure.



```
string dnsLookup(DNSServer server, const string &query)
{
    int delay = rand() % 500 + 100;

    // Define a timeout threshold (e.g., 400 milliseconds)
    int timeoutThreshold = 400;

    // If the delay exceeds the timeout, simulate query failure due to timeout
    if (delay > timeoutThreshold)
    {
        cout << "DNS query timed out after " << delay << " ms at "
             << (server == ROOT ? "ROOT" : server == TLD ? "TLD"
                                                          : "AUTHORITATIVE")
             << " server.\n";
        return ""; // Return empty string to indicate failure due to timeout
    }

    // Simulate the actual delay for the query
    this_thread::sleep_for(chrono::milliseconds(delay));

    if (rand() % 10 < 1) // Randomly simulate DNS query failure
    {
        return ""; // Simulate a failed DNS query
    }

    // Handle the query based on the server type
    if (server == ROOT)
    {
        // Extract the TLD from the domain (e.g., "com" from "example.com")
        string tld = query.substr(query.find_last_of('.') + 1);
        cout << "Querying Root DNS Server for TLD: " << tld << "\n";

        // Check if TLD is recognized
        if (tld == "com" || tld == "uk" || tld == "org" || tld == "in")
        {
            // Recursively call the TLD server for further resolution
            return dnsLookup(TLD, query);
        }
        else
        {
            cout << "TLD not recognized by Root server.\n";
            return ""; // TLD not found
        }
    }
    else if (server == TLD)
    {
        // Querying the TLD server with the full domain (e.g., "example.com")
        cout << "Querying TLD DNS Server for domain: " << query << "\n";

        // Recognize certain domains and delegate to the Authoritative server
        if (query == "www.example.com" || query == "www.google.com" || query == "www.facebook.com" || query == "www.youtube.com" ||
        query == "www.wikipedia.org" || query == "www.iitg.ac.in" || query == "www.amazon.com" || query == "www.neverssl.com" || query
        == "www.codeforces.com")
        {
```



```
    cout << "Querying TLD DNS Server for domain: " << query << "\n";

    // Recognize certain domains and delegate to the Authoritative server
    if (query == "www.example.com" || query == "www.google.com" || query == "www.facebook.com" || query == "www.youtube.com" ||
    query == "www.wikipedia.org" || query == "www.iitg.ac.in" || query == "www.amazon.com" || query == "www.neverssl.com" || query
    == "www.codeforces.com")
    {
        // Recursively call the Authoritative server for final resolution
        return dnsLookup(AUTHORITATIVE, query);
    }
    else
    {
        cout << "Domain not recognized by TLD server.\n";
        return ""; // Domain not found
    }
}
else if (server == AUTHORITATIVE)
{
    // Authoritative server provides the final IP address for recognized domains
    cout << "Querying Authoritative DNS Server for final resolution of: " << query << "\n";

    if (query == "www.example.com")
        return "93.184.215.14"; // Example website
    if (query == "www.google.com")
        return "142.250.195.36"; // Google
    if (query == "www.facebook.com")
        return "163.70.143.35"; // Facebook
    if (query == "www.youtube.com")
        return "142.250.72.238"; // YouTube
    if (query == "www.wikipedia.org")
        return "103.102.166.224"; // Wikipedia
    if (query == "www.iitg.ac.in")
        return "172.17.0.22"; // IITG
    if (query == "www.amazon.com")
        return "205.251.215.216"; // Amazon
    if (query == "www.neverssl.com")
        return "34.223.124.45"; // NeverSSL
    if (query == "www.codeforces.com")
        return "172.67.68.254"; // CodeForces
}

// Default empty response if none of the cases match
return "";
```

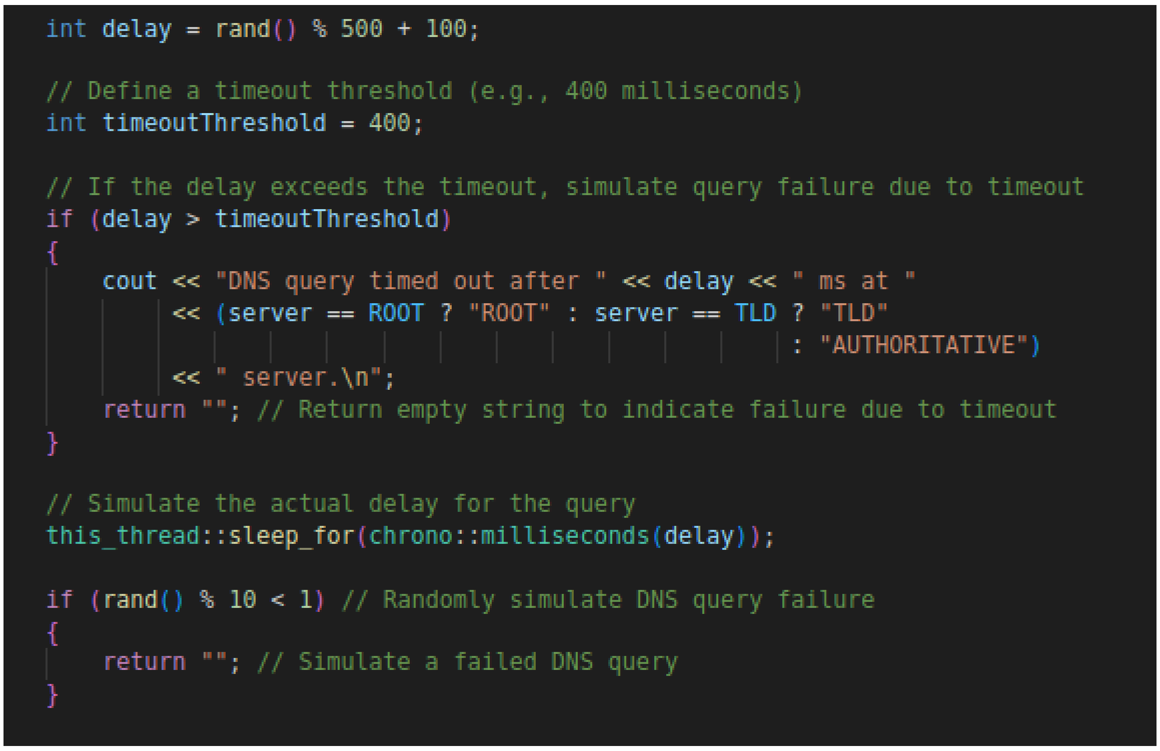
3. Explanation of Sequential DNS Queries and Their Implications

Querying multiple DNS servers sequentially can increase the overall time to retrieve a web page. If a server fails or is slow to respond, the time to resolve the domain name can significantly increase, impacting the user experience. However, caching the IP address locally after a successful lookup can improve performance for subsequent requests.

3. Simulating the DNS Lookup Process with Random Delays and Failures

1 Introducing Random Delays and Failures

The DNS lookup simulation introduces random delays and a probability of failure at each server level. This helps mimic real-world network conditions where DNS servers may be slow or temporarily unavailable.



```
int delay = rand() % 500 + 100;

// Define a timeout threshold (e.g., 400 milliseconds)
int timeoutThreshold = 400;

// If the delay exceeds the timeout, simulate query failure due to timeout
if (delay > timeoutThreshold)
{
    cout << "DNS query timed out after " << delay << " ms at "
         << (server == ROOT ? "ROOT" : server == TLD ? "TLD"
                                                     : "AUTHORITATIVE")
         << " server.\n";
    return ""; // Return empty string to indicate failure due to timeout
}

// Simulate the actual delay for the query
this_thread::sleep_for(chrono::milliseconds(delay));

if (rand() % 10 < 1) // Randomly simulate DNS query failure
{
    return ""; // Simulate a failed DNS query
}
```

2 Mechanism to Handle Delays and Failures Gracefully

The code decides whether to wait for a delayed response or switch to a backup server based on a timeout threshold. If a query takes too long, it is considered a failure, and the system immediately retries with a backup server.

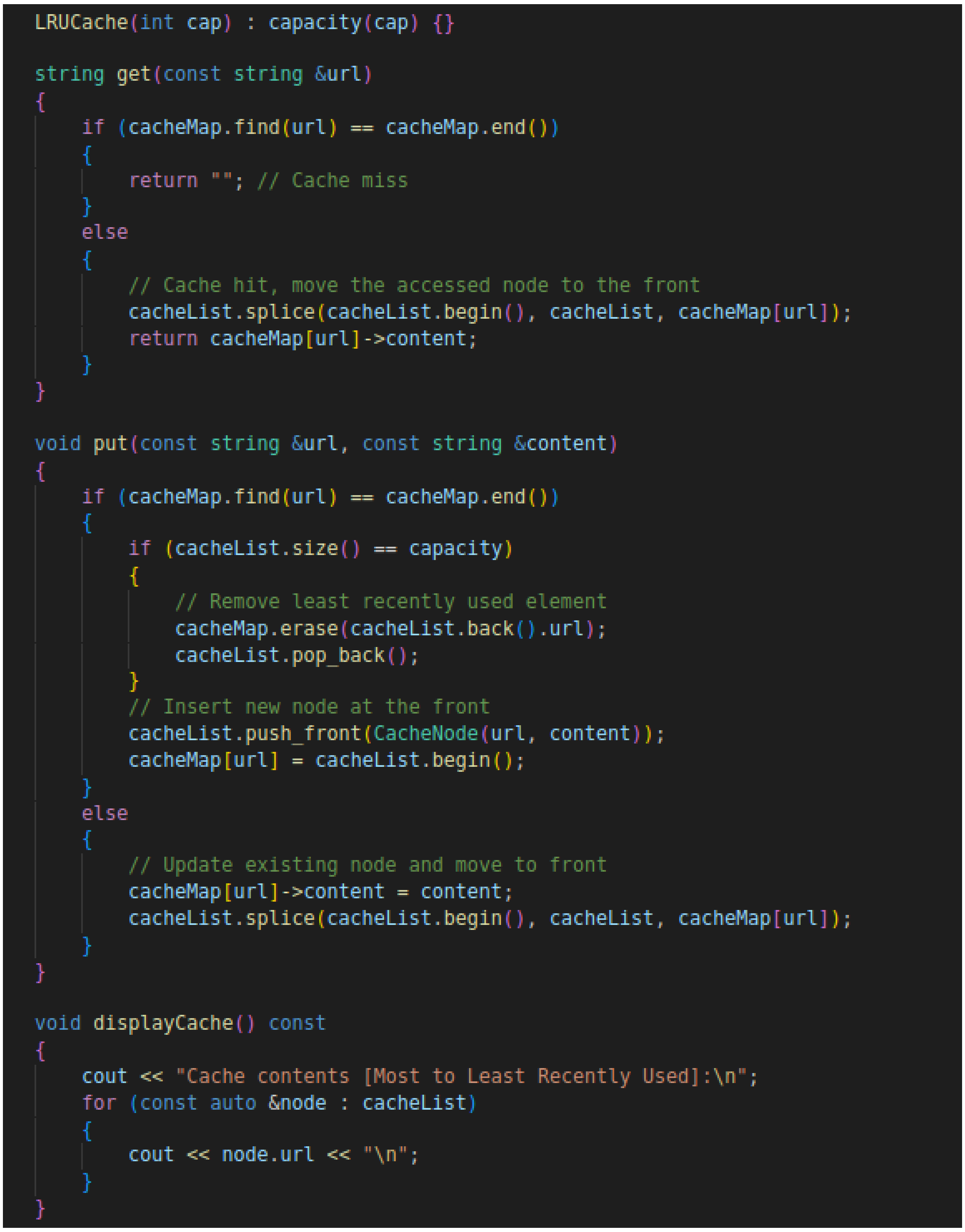
4.1 HTTP Proxy Server with Caching Mechanism

1 Explanation of the HTTP Proxy Server with Caching

An HTTP proxy server can cache frequently accessed pages to reduce retrieval times. The caching mechanism is implemented using an LRU (Least Recently Used) cache, which stores a limited number of pages. When a page is requested, the proxy server first checks the cache:

- If the page is in the cache, it returns the cached content.
- If not, it performs a DNS lookup, fetches the page using an HTTP GET request, stores it in the cache, and returns the content.

2 Code Explanation for LRU Cache Implementation



```cpp
LRUCache(int cap) : capacity(cap) {}

string get(const string &url)
{
    if (cacheMap.find(url) == cacheMap.end())
    {
        return ""; // Cache miss
    }
    else
    {
        // Cache hit, move the accessed node to the front
        cacheList.splice(cacheList.begin(), cacheList, cacheMap[url]);
        return cacheMap[url]->content;
    }
}

void put(const string &url, const string &content)
{
    if (cacheMap.find(url) == cacheMap.end())
    {
        if (cacheList.size() == capacity)
        {
            // Remove least recently used element
            cacheMap.erase(cacheList.back().url);
            cacheList.pop_back();
        }
        // Insert new node at the front
        cacheList.push_front(CacheNode(url, content));
        cacheMap[url] = cacheList.begin();
    }
    else
    {
        // Update existing node and move to front
        cacheMap[url]->content = content;
        cacheList.splice(cacheList.begin(), cacheList, cacheMap[url]);
    }
}

void displayCache() const
{
    cout << "Cache contents [Most to Least Recently Used]:\n";
    for (const auto &node : cacheList)
    {
        cout << node.url << "\n";
    }
}
```

Explanation:

- The cache uses a doubly linked list to keep track of the order in which pages are accessed. When a page is accessed, it is moved to the front of the list.
- When the cache reaches its maximum capacity, the least recently used page is removed.

3 Benefits of Caching:

Benefits of Caching in an HTTP Proxy Server

1. **Faster Response Times:** Caching allows quick access to frequently requested content, reducing the need for repeated DNS lookups and HTTP requests.
2. **Reduced Bandwidth Usage:** Cached content minimizes data transfer over the network, saving bandwidth and lowering costs.
3. **Lower Load on Servers:** Caching reduces the number of requests to origin servers, decreasing their workload and preventing overload.
4. **Improved Scalability:** The system can handle more users efficiently by serving cached content, enhancing its scalability.
5. **Enhanced Performance:** Users in different locations get faster access to cached content, reducing delays caused by geographical distance.
6. **Resilience to Failures:** Cached data ensures users can access content even if the origin server is temporarily unavailable.
7. **Cost Savings:** Less bandwidth and reduced server load translate to lower operational costs.
8. **Offline Access:** Cached content can be accessed even when the network is down, improving user experience.

4.2 Cache Management and Performance Analysis

1. Impact of DNS Lookups on Web Page Retrieval Time

Discussion:

The series of DNS lookups significantly impacts the time required to retrieve a web page. Each DNS query involves:

- **Initial Query:** Sent to the local DNS resolver.
- **Root DNS Server:** Directs the resolver to the appropriate TLD server.
- **TLD DNS Server:** Provides the authoritative DNS server.
- **Authoritative DNS Server:** Resolves the domain name to an IP address.

Implications of Querying Multiple DNS Servers Sequentially:

- **Increased Latency:** Each step adds a delay, compounded by potential network delays and server processing times. Sequential queries can significantly increase the overall retrieval time.
- **Timeouts and Failures:** If any DNS server is slow or unresponsive, it can delay the entire resolution process. Implementing backup DNS servers can help mitigate this risk, but it may also add to the complexity and time.
- **Retry Mechanisms:** Handling failures or delays often involves retrying queries or switching to backup servers, further impacting retrieval time.

2. Impact of Cache Size on Proxy Server Performance

Reflection:

2. Cache Size Impact:

- **Larger Cache:**
    - **Pros:** Can store more pages, reducing cache misses and potentially decreasing retrieval times for frequently accessed content.
    - **Cons:** Requires more memory and can lead to slower cache management due to increased overhead.
- **Smaller Cache:**
    - **Pros:** Requires less memory and can be managed more efficiently. Eviction policies are simpler.
    - **Cons:** Higher likelihood of cache misses, leading to more frequent DNS lookups and HTTP requests.

Trade-offs:

- **Memory Usage vs. Performance:** A larger cache improves hit rates but uses more memory. A smaller cache is memory-efficient but may result in higher retrieval times due to more frequent misses.
- **Eviction Policies:** Larger caches may require more sophisticated eviction policies (e.g., LRU) to manage content efficiently, while smaller caches may rely on simpler approaches.

3. Challenges in Integrating DNS Lookups with HTTP Communication

Analysis:

- **Coordination Issues:** Synchronizing DNS lookups with HTTP requests requires careful handling to ensure the IP address is resolved before making an HTTP request.
- **Error Handling:** Must handle DNS resolution failures gracefully and implement fallback mechanisms, such as querying backup DNS servers.
- **Delay Management:** Integrating delays from DNS lookups with HTTP communication involves managing response times and ensuring that DNS resolution does not excessively delay HTTP requests.

Coordination Approach:

- **Sequential Process:** Ensure DNS resolution is completed before initiating the HTTP request.
- **Timeout and Retry Mechanisms:** Implement mechanisms to handle delays and retries, ensuring that DNS failures or timeouts are managed effectively.
- **Caching:** Use caching to minimise repeated DNS lookups and reduce the impact of delays.

4. Performance Implications of Cache Misses

Considerations:

- **Increased Retrieval Time:** When a page is not in the cache, the proxy server must perform a DNS lookup and HTTP request, increasing retrieval time compared to serving cached content.

- **User Experience:** Frequent cache misses lead to slower page loads, impacting the user experience negatively. Users may experience delays while waiting for DNS resolution and content retrieval.

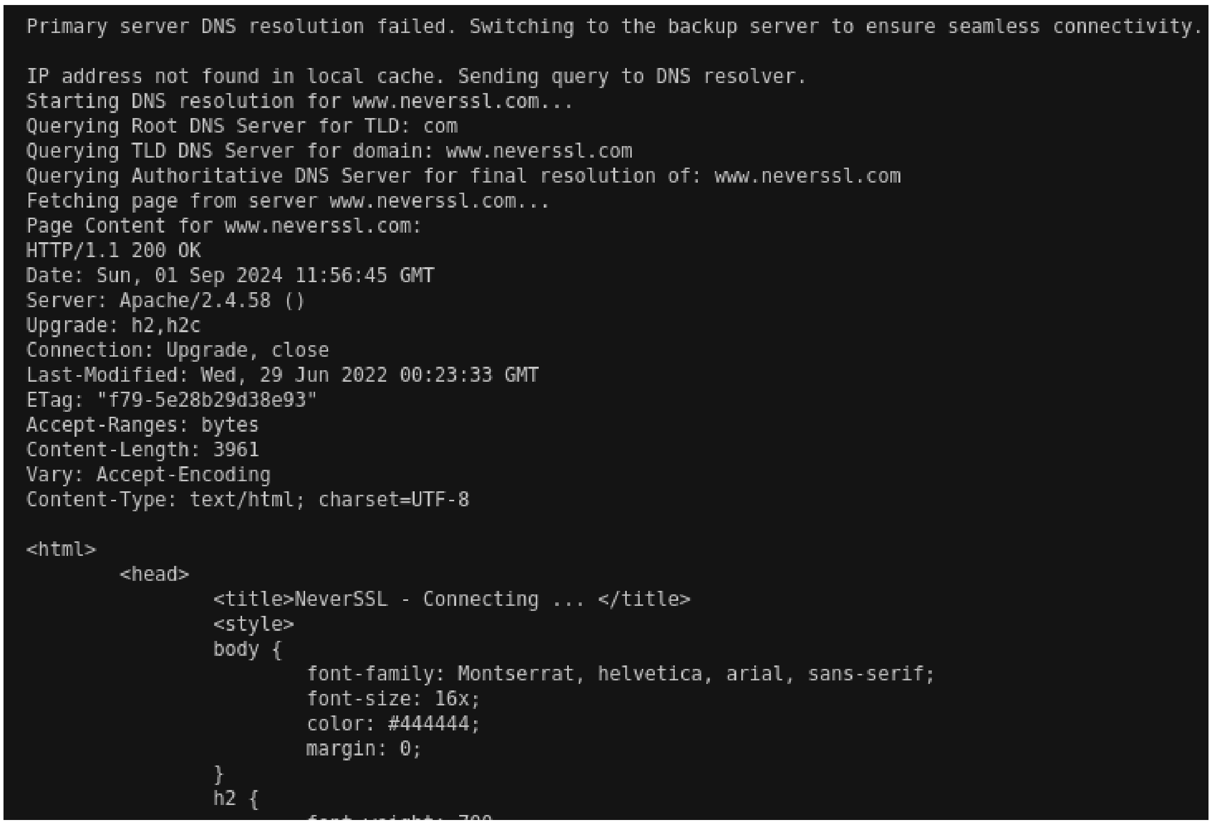
Influence on User Experience:

- **Perceived Performance:** Users perceive slower performance during cache misses due to increased latency from DNS lookups and HTTP requests.
- **Content Availability:** Cache misses can lead to temporary unavailability of content, affecting user satisfaction.

5. Attached Screenshots

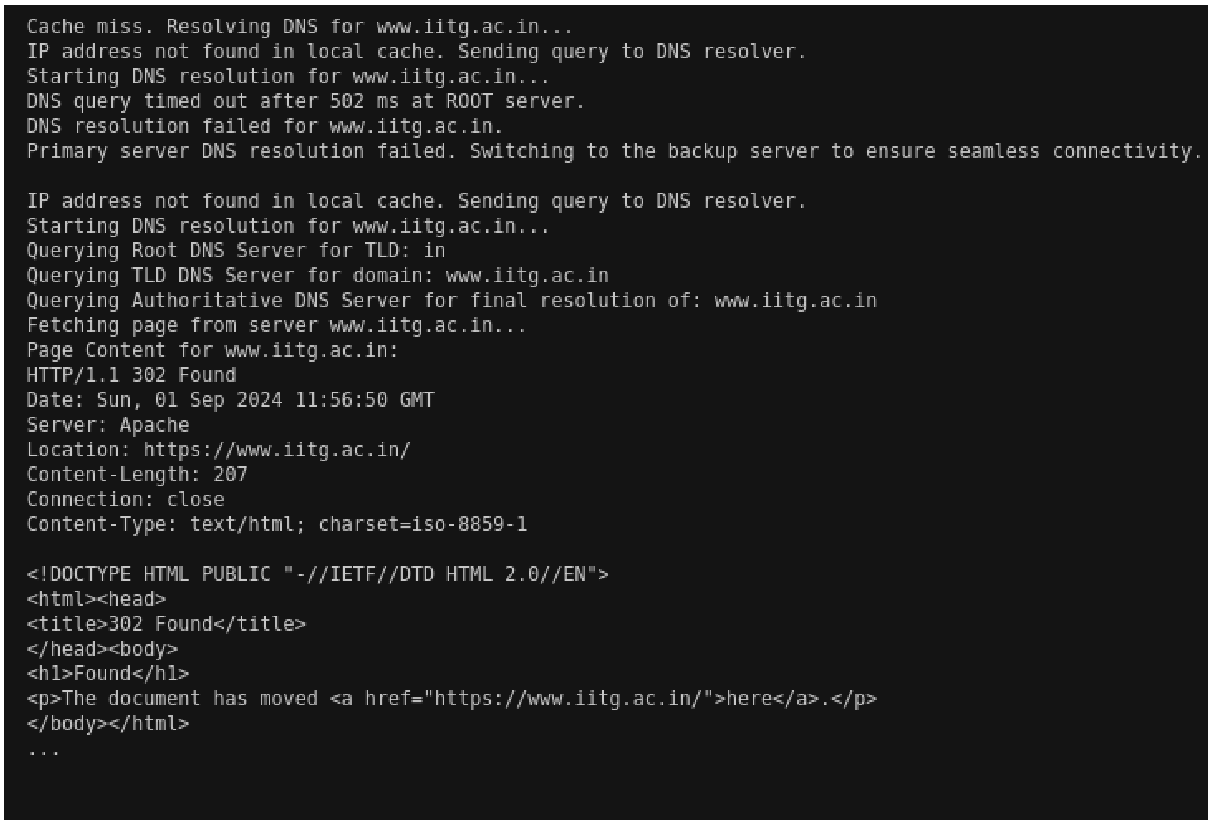
Attached screenshots of:

1. DNS lookup failure then success in backup server.



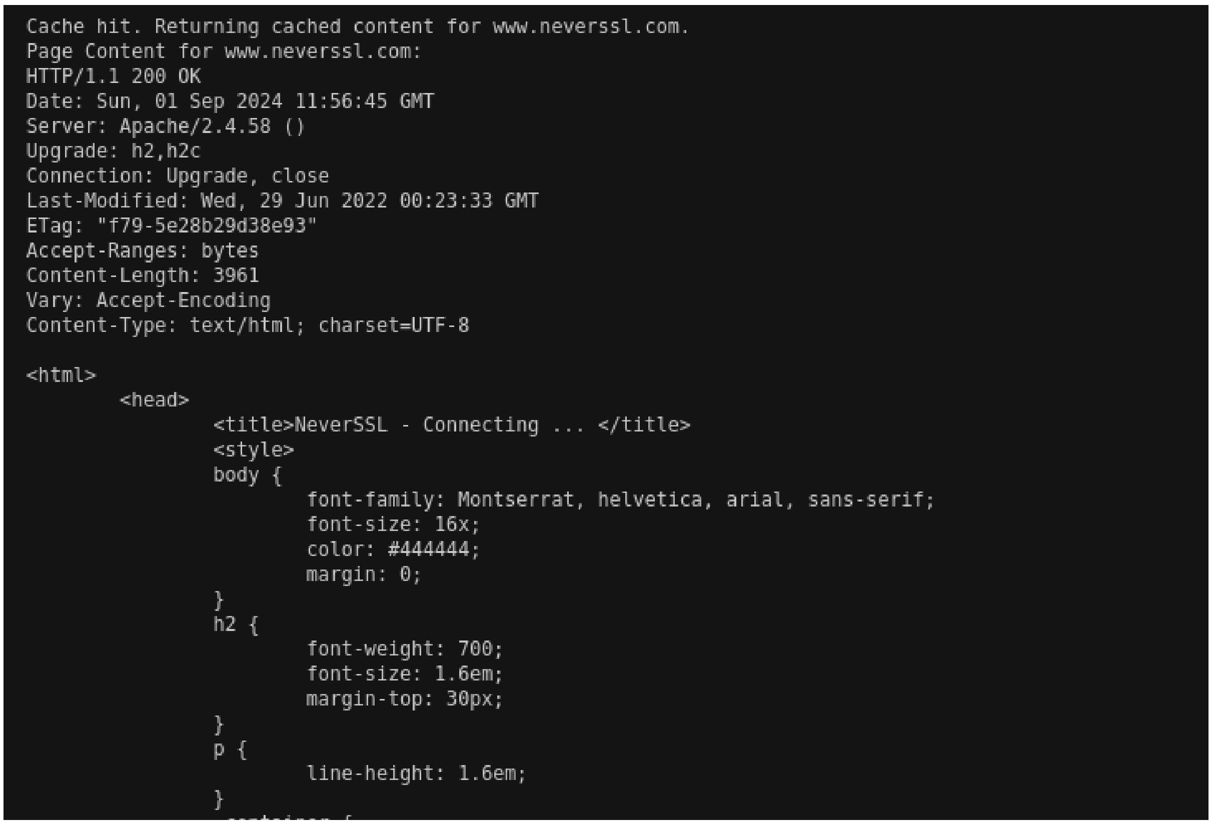
```
Primary server DNS resolution failed. Switching to the backup server to ensure seamless connectivity.

IP address not found in local cache. Sending query to DNS resolver.
Starting DNS resolution for www.neverssl.com...
Querying Root DNS Server for TLD: com
Querying TLD DNS Server for domain: www.neverssl.com
Querying Authoritative DNS Server for final resolution of: www.neverssl.com
Fetching page from server www.neverssl.com...
Page Content for www.neverssl.com:
HTTP/1.1 200 OK
Date: Sun, 01 Sep 2024 11:56:45 GMT
Server: Apache/2.4.58 ()
Upgrade: h2,h2c
Connection: Upgrade, close
Last-Modified: Wed, 29 Jun 2022 00:23:33 GMT
ETag: "f79-5e28b29d38e93"
Accept-Ranges: bytes
Content-Length: 3961
Vary: Accept-Encoding
Content-Type: text/html; charset=UTF-8

<html>
        <head>
                <title>NeverSSL - Connecting ... </title>
                <style>
                body {
                        font-family: Montserrat, helvetica, arial, sans-serif;
                        font-size: 16x;
                        color: #444444;
                        margin: 0;
                }
                h2 {
```

2. Cache misses in the proxy server.



```
Cache miss. Resolving DNS for www.iitg.ac.in...
IP address not found in local cache. Sending query to DNS resolver.
Starting DNS resolution for www.iitg.ac.in...
DNS query timed out after 502 ms at ROOT server.
DNS resolution failed for www.iitg.ac.in.
Primary server DNS resolution failed. Switching to the backup server to ensure seamless connectivity.

IP address not found in local cache. Sending query to DNS resolver.
Starting DNS resolution for www.iitg.ac.in...
Querying Root DNS Server for TLD: in
Querying TLD DNS Server for domain: www.iitg.ac.in
Querying Authoritative DNS Server for final resolution of: www.iitg.ac.in
Fetching page from server www.iitg.ac.in...
Page Content for www.iitg.ac.in:
HTTP/1.1 302 Found
Date: Sun, 01 Sep 2024 11:56:50 GMT
Server: Apache
Location: https://www.iitg.ac.in/
Content-Length: 207
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://www.iitg.ac.in/">here</a>.</p>
</body></html>
...
```

3.Cache hits in the proxy server.



```
Cache hit. Returning cached content for www.neverssl.com.
Page Content for www.neverssl.com:
HTTP/1.1 200 OK
Date: Sun, 01 Sep 2024 11:56:45 GMT
Server: Apache/2.4.58 ()
Upgrade: h2,h2c
Connection: Upgrade, close
Last-Modified: Wed, 29 Jun 2022 00:23:33 GMT
ETag: "f79-5e28b29d38e93"
Accept-Ranges: bytes
Content-Length: 3961
Vary: Accept-Encoding
Content-Type: text/html; charset=UTF-8

<html>
        <head>
                <title>NeverSSL - Connecting ... </title>
                <style>
                body {
                        font-family: Montserrat, helvetica, arial, sans-serif;
                        font-size: 16x;
                        color: #444444;
                        margin: 0;
                }
                h2 {
                        font-weight: 700;
                        font-size: 1.6em;
                        margin-top: 30px;
                }
                p {
                        line-height: 1.6em;
                }
```

6. Conclusion

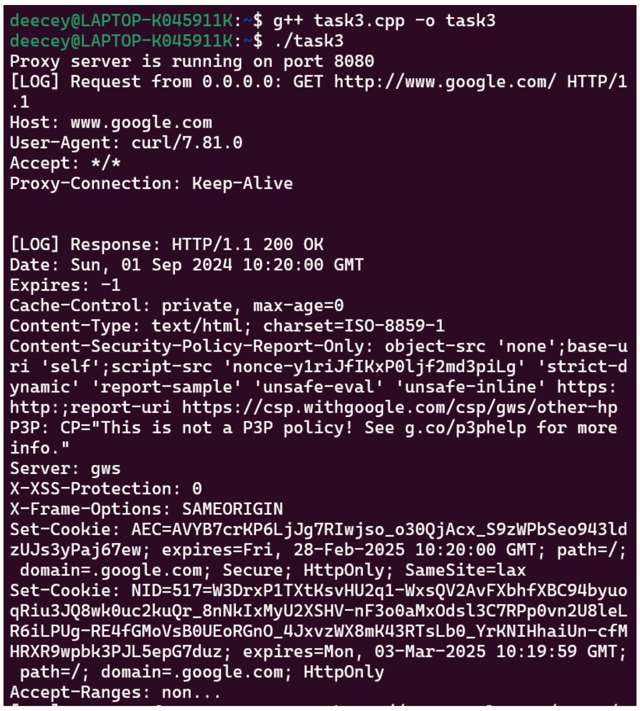
This report has demonstrated the implementation of a DNS lookup simulation and an HTTP proxy server with a caching mechanism. The DNS lookup simulation captures the potential delays and failures that occur during the domain resolution process. The HTTP proxy server demonstrates how caching can improve performance by reducing the need for repeated DNS lookups and HTTP requests. Overall, these mechanisms are essential for optimising web communication and improving user experience.

Q3) Develop an HTTP Proxy Server

Implement an HTTP Proxy server capable of handling HTTP GET requests from clients.

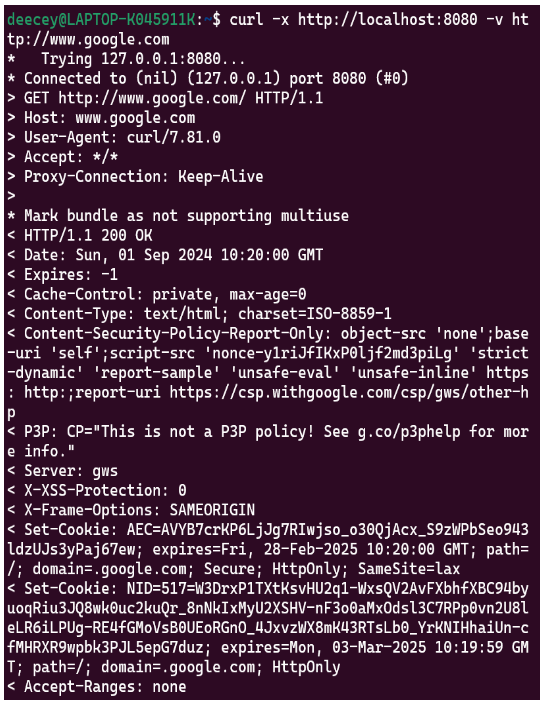
To implement an HTTP Proxy server that handles HTTP GET requests from clients, we developed a server in C++ that listens for incoming connections and processes client requests. The server is capable of parsing the HTTP GET requests, identifying the target server, and forwarding the requests to the appropriate destination. Upon receiving the server's response, the proxy relays it back to the client while logging the interaction details. Additionally, the server manages cookies, particularly for interactions with www.google.com, storing and forwarding them as necessary for subsequent requests.

SERVER



```
deecey@LAPTOP-K045911K:~$ g++ task3.cpp -o task3
deecey@LAPTOP-K045911K:~$ ./task3
Proxy server is running on port 8080
[LOG] Request from 0.0.0.0: GET http://www.google.com/ HTTP/1
.1
Host: www.google.com
User-Agent: curl/7.81.0
Accept: */*
Proxy-Connection: Keep-Alive


[LOG] Response: HTTP/1.1 200 OK
Date: Sun, 01 Sep 2024 10:20:00 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src 'none';base-u
ri 'self';script-src 'nonce-y1riJfIKxP0ljf2md3piLg' 'strict-d
ynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https:
http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more
info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: AEC=AVYB7crKP6LjJg7RIwjso_o30QjAcx_S9zWPbSeo943ld
zUJs3yPaj67ew; expires=Fri, 28-Feb-2025 10:20:00 GMT; path=/;
 domain=.google.com; Secure; HttpOnly; SameSite=lax
Set-Cookie: NID=517=W3DrxP1TXtKsvHU2q1-WxsQV2AvFXbhfXBC94byuo
qRiu3JQ8wk0uc2kuQr_8nNkIxMyU2XSHV-nF3o0aMxOdsl3C7RPp0vn2U8leL
R6iLPUg-RE4fGMoVsB0UEoRGnO_4JxvzWX8mK43RTsLb0_YrKNIHhaiUn-cfM
HRXR9wpbk3PJL5epG7duz; expires=Mon, 03-Mar-2025 10:19:59 GMT;
 path=/; domain=.google.com; HttpOnly
Accept-Ranges: non...
```

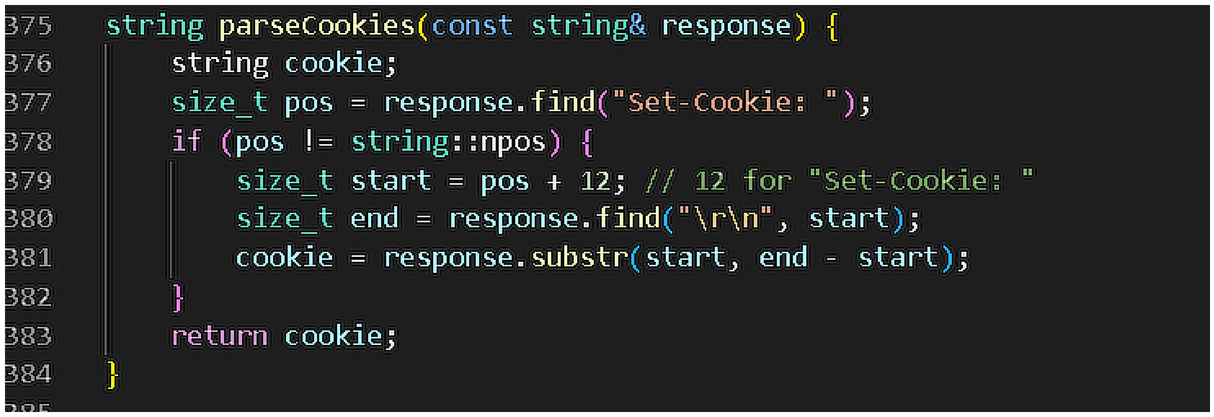
CLIENT



```
deecey@LAPTOP-K045911K:~$ curl -x http://localhost:8080 -v ht
tp://www.google.com
*   Trying 127.0.0.1:8080...
* Connected to (nil) (127.0.0.1) port 8080 (#0)
> GET http://www.google.com/ HTTP/1.1
> Host: www.google.com
> User-Agent: curl/7.81.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Sun, 01 Sep 2024 10:20:00 GMT
< Expires: -1
< Cache-Control: private, max-age=0
< Content-Type: text/html; charset=ISO-8859-1
< Content-Security-Policy-Report-Only: object-src 'none';base
-uri 'self';script-src 'nonce-y1riJfIKxP0ljf2md3piLg' 'strict
-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https
: http:;report-uri https://csp.withgoogle.com/csp/gws/other-h
p
< P3P: CP="This is not a P3P policy! See g.co/p3phelp for mor
e info."
< Server: gws
< X-XSS-Protection: 0
< X-Frame-Options: SAMEORIGIN
< Set-Cookie: AEC=AVYB7crKP6LjJg7RIwjso_o30QjAcx_S9zWPbSeo943
ldzUJs3yPaj67ew; expires=Fri, 28-Feb-2025 10:20:00 GMT; path=
/; domain=.google.com; Secure; HttpOnly; SameSite=lax
< Set-Cookie: NID=517=W3DrxP1TXtKsvHU2q1-WxsQV2AvFXbhfXBC94by
uoqRiu3JQ8wk0uc2kuQr_8nNkIxMyU2XSHV-nF3o0aMxOdsl3C7RPp0vn2U8l
eLR6iLPUg-RE4fGMoVsB0UEoRGnO_4JxvzWX8mK43RTsLb0_YrKNIHhaiUn-c
fMHRXR9wpbk3PJL5epG7duz; expires=Mon, 03-Mar-2025 10:19:59 GM
T; path=/; domain=.google.com; HttpOnly
< Accept-Ranges: none
```

The command curl -x http://localhost:8080 -v http://www.google.com uses the curl tool to send an HTTP request to www.google.com through a specified proxy server running on localhost at port 8080.

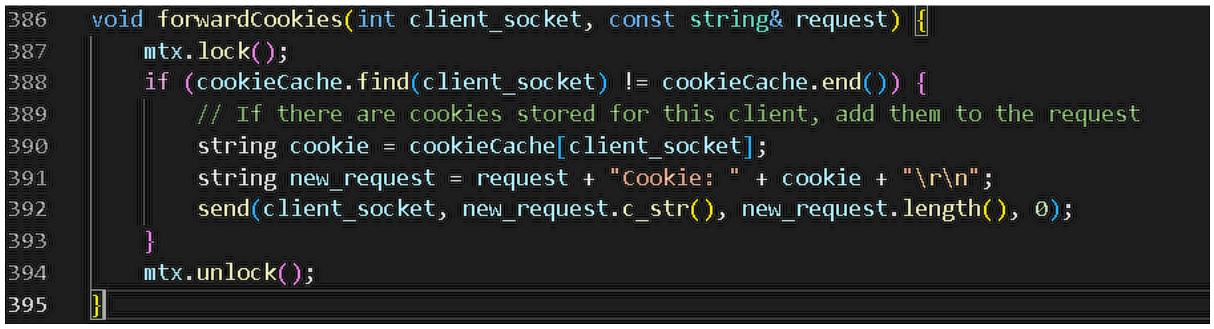
Cookie Parsing: Parse cookies from server responses



```cpp
string parseCookies(const string& response) {
    string cookie;
    size_t pos = response.find("Set-Cookie: ");
    if (pos != string::npos) {
        size_t start = pos + 12; // 12 for "Set-Cookie: "
        size_t end = response.find("\r\n", start);
        cookie = response.substr(start, end - start);
    }
    return cookie;
}
```

When a server sends an HTTP response to a client (like a web browser), it can include a "Set-Cookie" header to store a small piece of data (a cookie) on the client side.This function takes a single argument response, which is a string representing the entire HTTP response from the server and searches for the first occurrence of "Set-Cookie" header. If found ,it extracts its value and returns that.

Cookie Forwarding: Forward stored cookies in subsequent client requests to the server



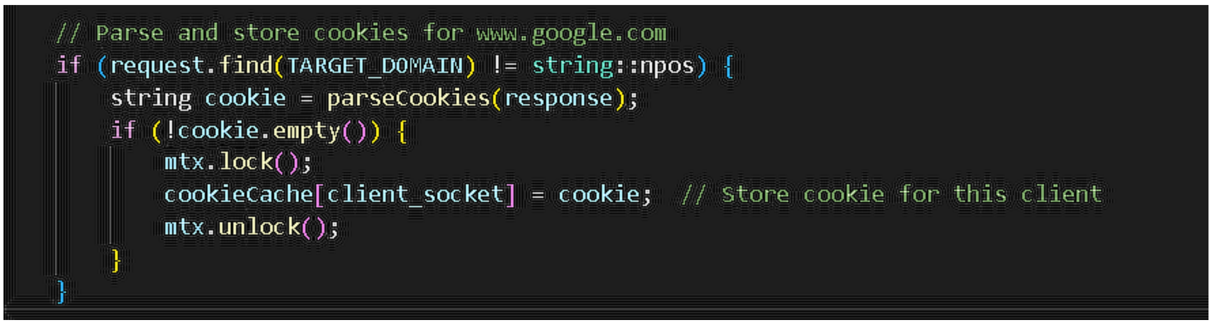
```cpp
void forwardCookies(int client_socket, const string& request) {
    mtx.lock();
    if (cookieCache.find(client_socket) != cookieCache.end()) {
        // If there are cookies stored for this client, add them to the request
        string cookie = cookieCache[client_socket];
        string new_request = request + "Cookie: " + cookie + "\r\n";
        send(client_socket, new_request.c_str(), new_request.length(), 0);
    }
    mtx.unlock();
}
```

When a client sends a request, the server may respond with a "Set-Cookie" header, which the client stores in the cookieCache (a map that associates cookies with the client socket). On subsequent requests, the function forwardCookies checks if a cookie exists for the client in the cookieCache :

- If a cookie is found, it appends this cookie to the outgoing HTTP request's headers under the "Cookie" field.
- The modified request (with the "Cookie" header) is then sent to the server.

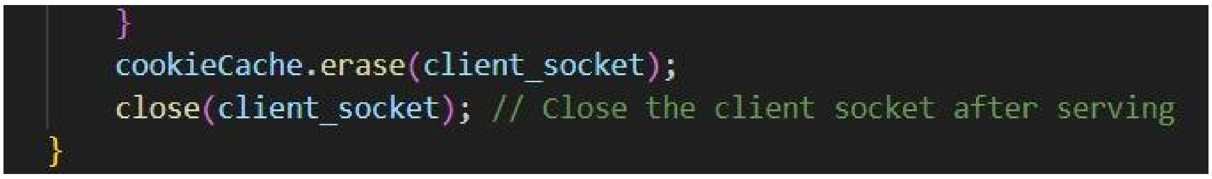
Session Management: Maintain a cache of cookies for each client session, ensuring that cookies are managed efficiently.

We have maintained a Cookie Cache as an unordered map from int to string. Here we have the session id of the client as the key (int) and cookie of the client from www.google.com as the value in this map.



```cpp
// Parse and store cookies for www.google.com
if (request.find(TARGET_DOMAIN) != string::npos) {
    string cookie = parseCookies(response);
    if (!cookie.empty()) {
        mtx.lock();
        cookieCache[client_socket] = cookie;  // Store cookie for this client
        mtx.unlock();
    }
}
```

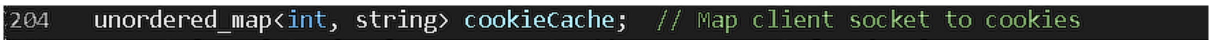
When the proxy server receives the response from the actual server , it searches for the cookie in the response. If it finds one, it stores it in cookieCache corresponding to that client session id. This process is repeated for subsequent requests.



```cpp
    }
    cookieCache.erase(client_socket);
    close(client_socket); // Close the client socket after serving
}
```

Before we close the client connection , we erase the the previously stored cookie so we can start fresh with new client session.

<u>Utilize an appropriate data structure to efficiently manage cookies for each client session</u>



```cpp
unordered_map<int, string> cookieCache;  // Map client socket to cookies
```

To efficiently manage cookies for each client session, the choice of data structure is crucial. It must allow for fast lookups, insertions, deletions, and must be scalable to handle potentially large numbers of client connections with minimal memory overhead.
We have used unordered map(hash map) because
1) it provides average O(1) time complexity for lookups, insertions, and deletions.
2) Memory usage is relatively low compared to structures like balanced binary search trees.
3)Efficiently handles large numbers of clients,
4)Allows integration with threading mechanisms (e.g., using mutexes) to ensure thread safety

Some other data structures we could have used were :-

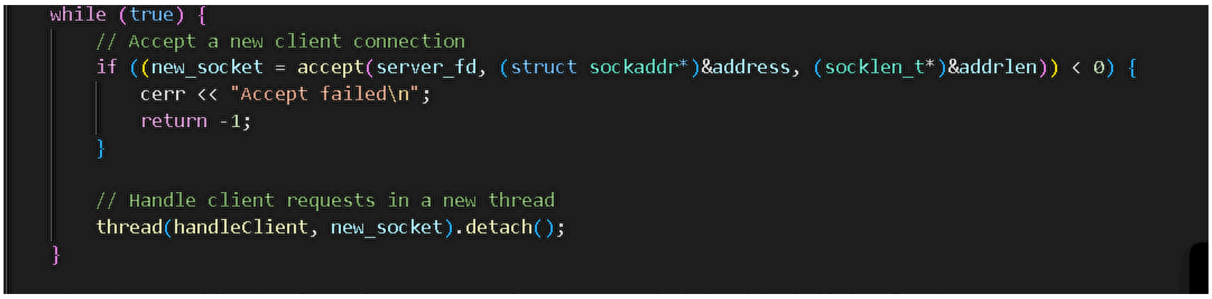
std::map (Red-Black Tree):

- **Performance:** Provides O(log n) time complexity for lookups, insertions, and deletions.
- **Memory Usage:** Typically higher memory usage due to the overhead of maintaining tree structure and node pointers.
- **Use Case:** Better suited when ordered traversal of cookies is needed (though rarely required for cookie management).

std::vector or std::list:

- **Performance:** O(n) lookups make them inefficient for scenarios with many clients.
- **Memory Usage:** Lower memory overhead for a small number of clients but poor scalability.
- **Use Case:** Not ideal for high-concurrency scenarios; suited for simple cases with few clients or cookies.

<u>Ensure the proxy server can handle multiple client sessions concurrently</u>

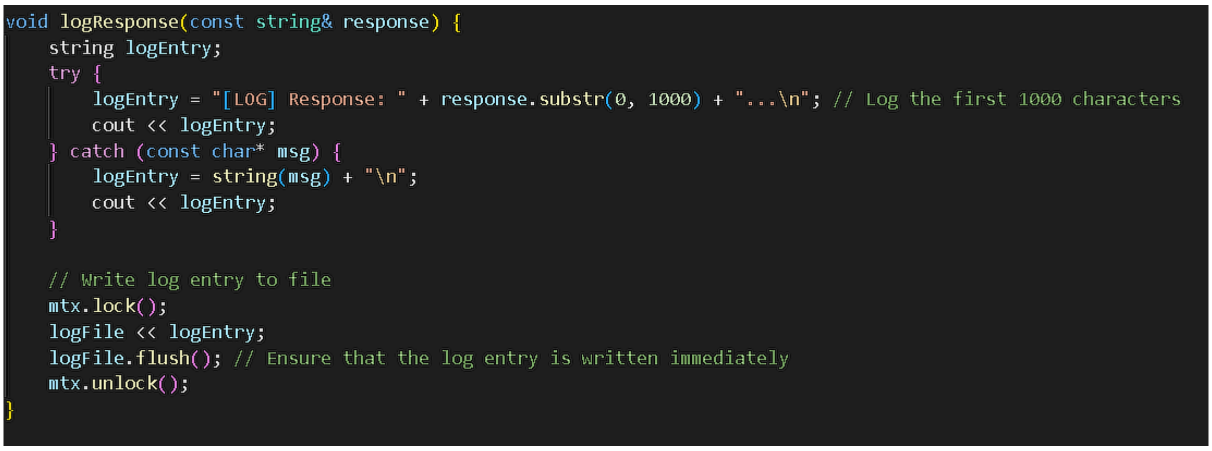


```
while (true) {
    // Accept a new client connection
    if ((new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
        cerr << "Accept failed\n";
        return -1;
    }

    // Handle client requests in a new thread
    thread(handleClient, new_socket).detach();
}
```

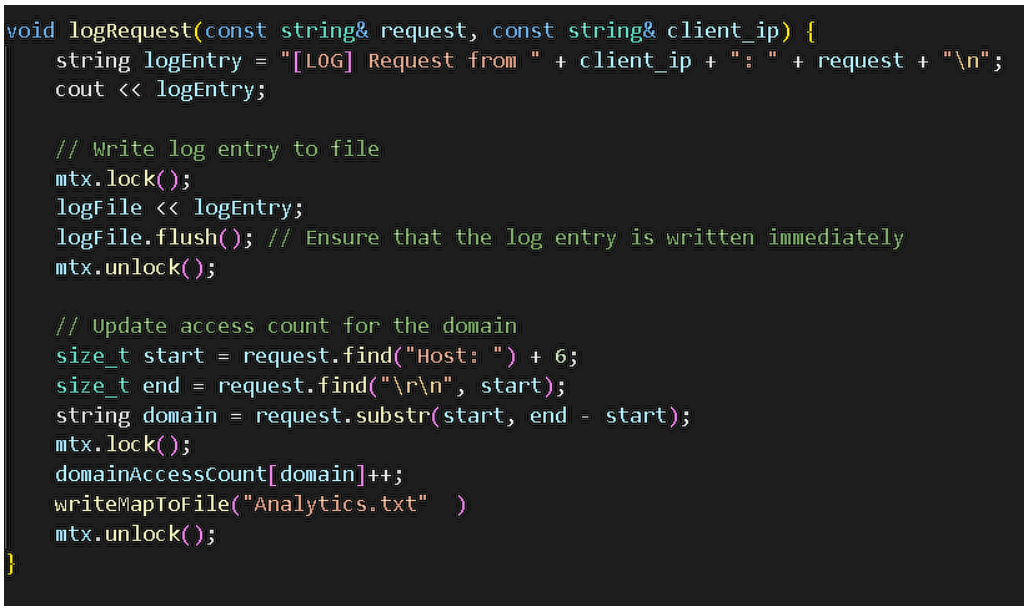
The above code helps us to maintain concurrency because by creating a new thread for each client connection, the server can handle multiple clients at the same time. Each thread runs independently and can process requests concurrently, The handleClient function can handle the client's request independently of other client requests. The detach method means the thread will clean up after itself when it finishes execution,

<u>Implement detailed logging to capture each request and response, including the cookies involved</u>

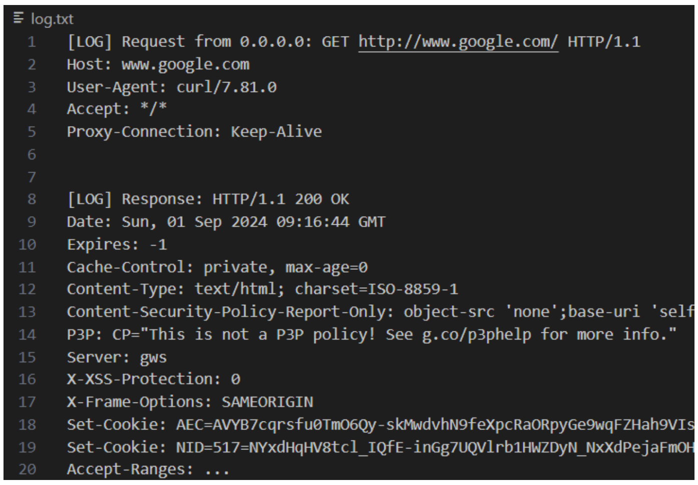
The following two functions allow us to log response and requests into the log files.



```
void logResponse(const string& response) {
    string logEntry;
    try {
        logEntry = "[LOG] Response: " + response.substr(0, 1000) + "...\n"; // Log the first 1000 characters
        cout << logEntry;
    } catch (const char* msg) {
        logEntry = string(msg) + "\n";
        cout << logEntry;
    }

    // Write log entry to file
    mtx.lock();
    logFile << logEntry;
    logFile.flush(); // Ensure that the log entry is written immediately
    mtx.unlock();
}
```



```
void logRequest(const string& request, const string& client_ip) {
    string logEntry = "[LOG] Request from " + client_ip + ": " + request + "\n";
    cout << logEntry;

    // Write log entry to file
    mtx.lock();
    logFile << logEntry;
    logFile.flush(); // Ensure that the log entry is written immediately
    mtx.unlock();

    // Update access count for the domain
    size_t start = request.find("Host: ") + 6;
    size_t end = request.find("\r\n", start);
    string domain = request.substr(start, end - start);
    mtx.lock();
    domainAccessCount[domain]++;
    writeMapToFile("Analytics.txt"   )
    mtx.unlock();
}
```

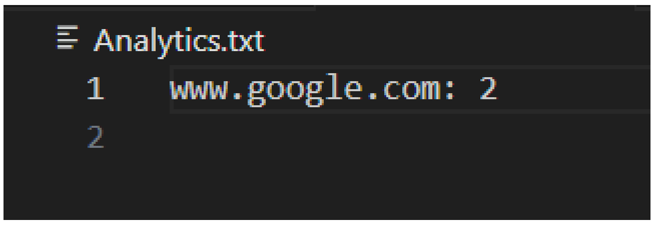


log.txt

```
[LOG] Request from 0.0.0.0: GET http://www.google.com/ HTTP/1.1
Host: www.google.com
User-Agent: curl/7.81.0
Accept: */*
Proxy-Connection: Keep-Alive


[LOG] Response: HTTP/1.1 200 OK
Date: Sun, 01 Sep 2024 09:16:44 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: AEC=AVYB7cqrsfu0TmO6Qy-skMwdvhN9feXpcRaORpyGe9wqFZHah9VIs
Set-Cookie: NID=517=NYxdHqHV8tcl_IQfE-inGg7UQVlrb1HWZDyN_NxXdPejaFmOH
Accept-Ranges: ...
```

As we can see from the log file, first a request is made to www.google.com . Then we receive a response from the server. As we can see we have Set-Cookie in the response header which we subsequently save and append to the next request that is made by the same client.

Provide analytics on the most frequently accessed domains and cookies.



Analytics.txt

```
www.google.com: 2
```

Along with the log file, we have created an analytics file on the most frequently accessed domains. Whenever a new request to the domain is made , the file is updated to reflect the changes.