CS342: Computer Networks Laboratory

Deadline: 11:59 pm, 1 September 2024

Instructions:

- This assignment is to be done in groups.
- The programming questions can be written in C/C++/Java.
- Your code should have a readme file, a makefile, and it should be well commented. These will carry separate marks for each question.
- Submit a soft copy of the report, preferably in PDF format, together with your code in a zip file on teams. The name of the zip file should be like "Your_Groupno.zip" (example: "Group11.zip").
- Files submitted without proper naming format will not be evaluated.
- If your trace file size is larger than 2 MB, you are advised to provide the OneDrive/Google Drive/Dropbox link of the traces in your report.
- No extensions in submission are allowed. Delay in submission will lead to penalty in marks.
- Assignments submitted before the deadline will only be considered for evaluation.
- Please do not email your assignments separately to the TAs, it will not be considered for evaluation.
- Your code will be checked for plagiarism. Any kind of academic dishonesty, plagiarism, etc. will lead to penalties.
- No sharing of code between students, submission of downloaded code is allowed.
- The first instance of code copying will result in ZERO marks for the assignment. The second instance of code copying will result in a ` F' grade. Students may also be reported to the Students Disciplinary Committee, which can impose additional penalties.
- Total Marks:100, 75(25x3, including 20 for programming and 5 for README/Makefile/comments) + 10 for the report + 15 for VIVA

Q1) HTTP using Wireshark

a) Identify an HTTP Request: Use Wireshark to capture HTTP traffic and identify a GET request. What is the URL of the requested resource, and which HTTP version is used?

b) Analyze HTTP Request: Find an HTTP response packet in the capture. What is the status code returned by the server, and what does it indicate about the request's outcome?
c) **Inspect HTTP Headers:** Locate an HTTP request packet and an HTTP response packet. Compare the headers in both. What headers are present in the request and response, and what information do they contain?
d) **Calculate Response Time:** Determine the response time for a specific HTTP request. How do you calculate the time difference between the request and the response?
e) **Examine HTTP Content:** Open an HTTP response packet. What is the content type of the response, and can you view the actual data sent back by the server (e.g., HTML content)?

Deliverables:

- Prepare a brief report (in PDF format) answering each of the questions above. Include screenshots from Wireshark to support your findings where applicable.
- Annotated Wireshark capture file (.pcap or .pcapng) highlighting the specific packets referenced in your report.
- Explanation of the process you followed for each task, including any challenges you encountered and how you resolved them.

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

Implement a simplified HTTP Proxy Server with a caching mechanism and simulate the DNS lookup process. The assignment is designed to help you understand the intricacies of web communication, including how DNS lookups interact with HTTP requests and caching.

Scenario:

When a user clicks on a link in their web browser, several key processes occur to retrieve the requested web page. One of these processes is the DNS (Domain Name System) lookup, which resolves the URL to an IP address.

1. Consider a scenario where four DNS servers are queried before your host receives the IP address from DNS. Describe the process of a DNS lookup when the IP address of the requested URL is not cached locally. How does the system query multiple DNS servers sequentially until the URL is resolved to an IP address?

2. To better understand this process, let's break down the steps of a recursive DNS lookup:
    - **Step 1: The Initial Query**

      When the user enters a URL in their browser, the browser first checks if the IP address is cached locally. If it's not found, the browser sends a DNS query to the local DNS resolver, typically provided by the Internet Service Provider (ISP).

    - **Step 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 of the requested domain but knows which TLD (Top-Level Domain) server to refer the query to.

    - **Step 3: Contacting the TLD DNS Server**

      The root DNS server responds by directing the local DNS resolver to the appropriate TLD DNS server (e.g., .com, .org). The local resolver then queries this TLD server.

    - **Step 4: Finding the Authoritative DNS Server**

      The TLD server doesn't have the IP address either but provides a referral to the authoritative DNS server that has the specific details for the domain being queried (e.g., www.example.com).

    - **Step 5: Getting the IP Address from the Authoritative DNS Server**

      The local DNS resolver queries the authoritative DNS server, which finally provides the IP address associated with the requested domain.

    - **Step 6: Caching and Returning the IP Address**

      The local DNS resolver caches the IP address for future requests and returns it to the user's browser, allowing the browser to initiate the HTTP request to the web server.

3. Now that you understand the DNS lookup process, simulate a series of DNS lookups by creating functions or processes that represent each DNS server. M**odify the**

traditional DNS lookup by introducing a random delay or failure at each DNS server level. For example:

- Introduce a probability that a DNS server might take longer to respond or might fail to return a result, forcing the system to retry or move to a backup DNS server.
- Implement a mechanism where the DNS resolver must decide whether to wait for the delayed response or immediately switch to a backup server.
- Design your code to handle these delays and failures gracefully, ensuring that the system can still resolve the URL within a reasonable time.

4. After the DNS resolution, explain how an HTTP Proxy Server can cache frequently accessed pages to reduce retrieval times for future requests. How would you simulate this caching process in a proxy server, and what are the potential benefits of such a system?

   Requirements:

   Develop a cache data structure capable of storing a limited number of web pages (e.g., 5 pages).

- The cache should allow efficient access and management of stored pages, ensuring that only the most relevant pages are retained. Create a function that retrieves a web page from the cache. If the page is present, return the cached content.
- If the page is not in the cache, simulate the DNS lookup process to resolve the URL to an IP address, perform an HTTP GET request to fetch the page, store it in the cache, and return the content.
- Implement HTTP GET requests using sockets to communicate with the Web server.
- Ensure that your implementation handles socket creation, connection, request sending, and response receiving correctly.
- Use a linked list to keep track of the order in which pages are accessed.
- When a page is accessed (from the cache or fetched a new), move it to the front of the list, representing it as the most recently used page.
- Implement cache eviction by removing the least recently used (LRU) page when the cache reaches its maximum capacity.
- Implement a function to display the current contents of the cache, showing pages from most recently accessed to least recently accessed.

Questions:

1. Discuss how the series of DNS lookups affects the overall time to retrieve a web page. What are the implications of querying multiple DNS servers sequentially?
2. Reflect on how the size of the cache impacts the performance of the Proxy server. What are the trade-offs of different cache sizes?
3. Analyze the challenges involved in integrating DNS lookups with HTTP communication. How did you handle the coordination between these processes?
4. Consider the performance implications of cache misses (when a requested page is not in the cache). How does this influence the user's experience?

Provide a brief summary in your report about your findings and reflections on the performance aspects discussed.

Q3) Develop an HTTP Proxy Server

Develop an HTTP Proxy server in C++ that handles client HTTP GET requests, manages cookies specifically for www.amazon.com, and provides detailed logging and analytics.

Requirements:

- Implement an HTTP Proxy server capable of handling HTTP GET requests from clients. The proxy should be able to parse and forward these requests to the appropriate servers.
- **Cookie Parsing:** Parse cookies from server responses specifically for www.amazon.com.
- **Cookie Storage:** Store cookies in a cache for www.amazon.com.
- **Cookie Forwarding:** Forward stored cookies in subsequent client requests to www.amazon.com.
- **Session Management:** Maintain a cache of cookies for each client session, ensuring that cookies are managed efficiently.
- Utilize an appropriate data structure to efficiently manage cookies for each client session. Consider the implications of different data structures on performance and memory usage.
- Ensure the proxy server can handle multiple client sessions concurrently.
- Implement detailed logging to capture each request and response, including the cookies involved. Logs should provide insight into the processing and handling of cookies for www.amazon.com.

- Provide analytics on the most frequently accessed domains and cookies. Specifically, include statistics on interactions with www.amazon.com, such as access frequency and cookie patterns.

Provide a summary in your report including answers to questions on the design choices for cookie management, concurrency, and logging and an analysis of the performance and efficiency of the implemented system.