Rahul Padhi - 919975938

Arun Khanijau - 919496132

11/25/24

Project 2 Part 1 Report

Part 1: DNS client from scratch

Name of my Python source code file: Part1DNSClient.py

Name of my Python source code file from ChatGPT: Part1DNSClient_GPT.py

Link to ChatGPT session: https://chatgpt.com/share/67453760-e8dc-8008-a236-4e4bfcf77880

Name of HTML output files submitting: DNSOutputEx1.html, DNSOutputEx2.html, DNSOutputEx3.html

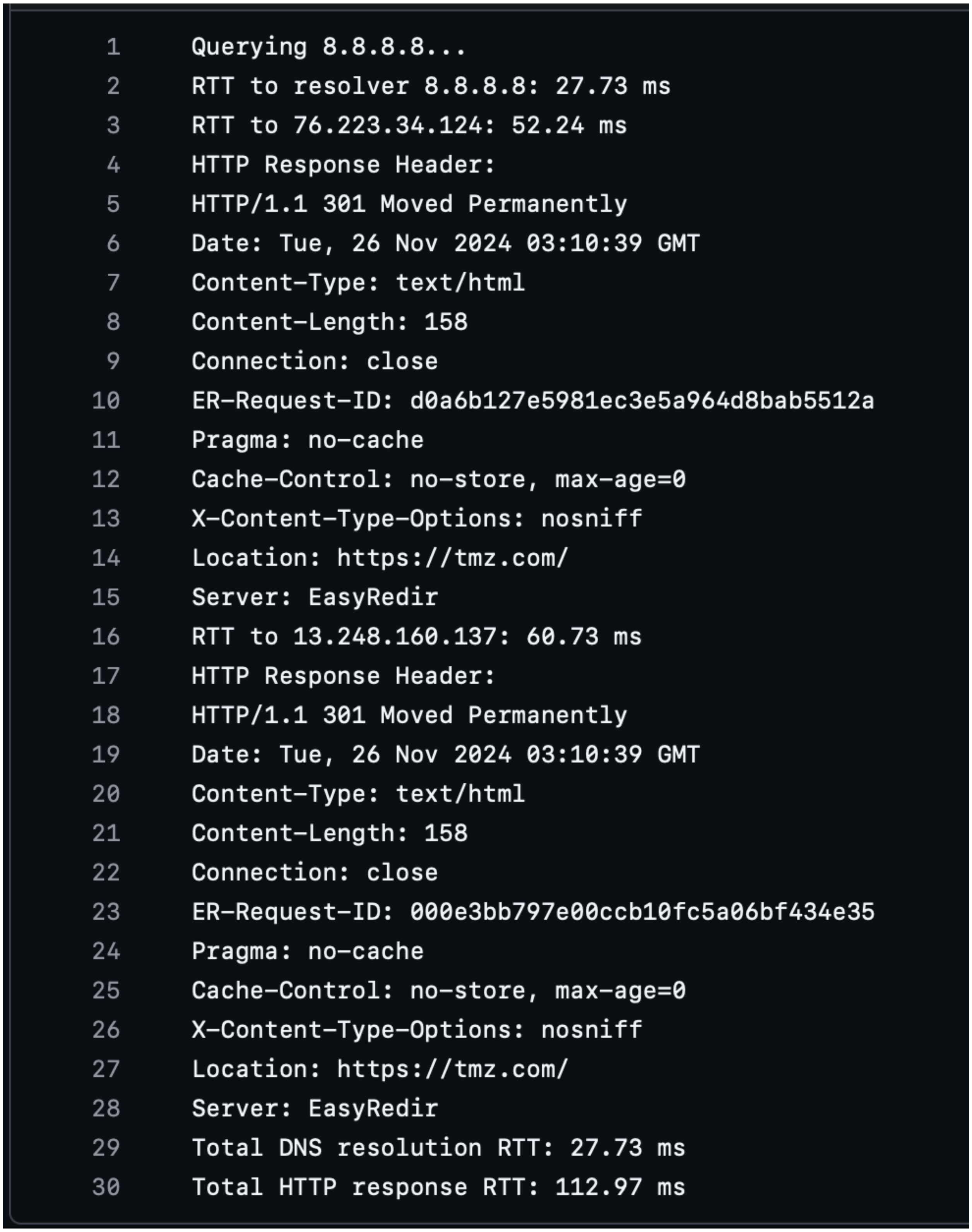
To implement the DNS client for this project, I followed a systematic approach that involved constructing raw DNS queries, sending them to a public DNS resolver, and parsing the responses manually. I started by creating the DNS query packet from scratch. This required understanding the structure of a DNS query and translating it into raw bytes. I began with a randomly generated transaction ID, followed by setting the flags for a standard query. For the domain name, tmz.com, I encoded it in a dotted-decimal format where each segment of the domain was prefixed with its length, ending with a null terminator. I added the query type as A (IPv4 address) and the class as IN (Internet) to complete the packet.

Once the query was ready, I used Python's socket API to send it over UDP to the public DNS resolver 8.8.8.8. I chose this resolver because it is widely used and reliable. To handle potential issues like timeouts, I implemented a mechanism to switch to another resolver if no response was received within 10 seconds. When a response was received, I parsed it manually to extract the resolved IP addresses. This involved navigating the DNS response structure, skipping the header and question sections to locate the answer section, and identifying A records based on their type field. Extracting the IPv4 addresses from these records confirmed the correctness of the query construction and response parsing.

After resolving the domain to its IP addresses, I moved on to making an HTTP request to each IP. For this, I created a GET request with the necessary headers, including the Host header to specify tmz.com. Using a TCP connection, I sent this request and captured the response headers to verify communication with the server. It was interesting to see that the server consistently redirected requests to its secure HTTPS endpoint, which is a common practice for modern websites.

Throughout the process, I measured round-trip times (RTTs) for both the DNS query and the HTTP request. To do this, I recorded timestamps before sending the packet and after receiving the response, calculating the difference to get the RTT. Debugging was also a big part of my implementation, and I often logged intermediate results to understand what was happening at each step. This iterative process helped me fine-tune the client and ensure it met all the project requirements. The results, including RTT measurements and response behavior, are detailed in output HTML files in our submission, with a sample photo of one below.

DNSOutputEx2.html



```
Querying 8.8.8.8...
RTT to resolver 8.8.8.8: 27.73 ms
RTT to 76.223.34.124: 52.24 ms
HTTP Response Header:
HTTP/1.1 301 Moved Permanently
Date: Tue, 26 Nov 2024 03:10:39 GMT
Content-Type: text/html
Content-Length: 158
Connection: close
ER-Request-ID: d0a6b127e5981ec3e5a964d8bab5512a
Pragma: no-cache
Cache-Control: no-store, max-age=0
X-Content-Type-Options: nosniff
Location: https://tmz.com/
Server: EasyRedir
RTT to 13.248.160.137: 60.73 ms
HTTP Response Header:
HTTP/1.1 301 Moved Permanently
Date: Tue, 26 Nov 2024 03:10:39 GMT
Content-Type: text/html
Content-Length: 158
Connection: close
ER-Request-ID: 000e3bb797e00ccb10fc5a06bf434e35
Pragma: no-cache
Cache-Control: no-store, max-age=0
X-Content-Type-Options: nosniff
Location: https://tmz.com/
Server: EasyRedir
Total DNS resolution RTT: 27.73 ms
Total HTTP response RTT: 112.97 ms
```