Lucas Jakin

Lab 3: HTTP and QUIC

Reti di Calcolatori II – Università di Trieste – Martino Trevisan

In this lab, you will practice with experiment with the **TCP**, **HTTP** (various versions) and **QUIC** protocols. You will practice with the **Google Chrome debugger**, measure the **performance** of different protocols, and learn basics of ***scarping*** (also called crawling). The tools you will use are:

- **Google Chrome Debugger:** by means of which you can inspect all HTTP transactions issued by the browser. Use the "Network" Tab and select the "Preserve Log" flag
- **BrowserTime**: a dockerized test suite to run experiments with automated web browsing.
- **`curl`:** a command line tool to issue HTTP request. You will use the classical `curl` command-line version and also a version compiled with a QUIC implementation, so that you can use HTTP/3.

Google Chrome Debugger

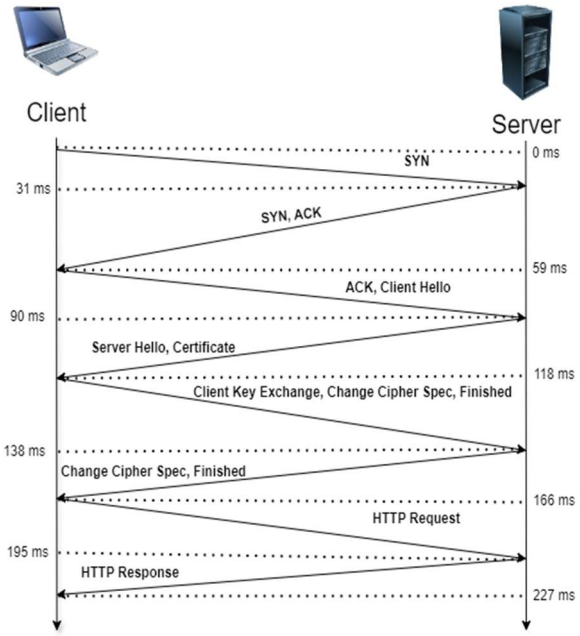
Practice with Google Chrome and with its **Debugger**. You must learn how to use the "*Network*" tab, which provides details on all HTTP requests. Check what happens when:

1. You access a webpage (e.g., www.units.it)
2. You scroll a web page
3. You log in into a website (hint: check what the "*Preserve Log*" flag does)

Question: When you log in into a website, which is the HTTP transaction carrying the **credentials**? Can you see the credentials through the **debugger**? Can a network observer (e.g., using **Wireshark**)?

> When logging in to Esse3 from "www.units.it", there are some HTTP transactions that are in action. The POST transaction "SSO?execution=e5s1" contains all the credentials used to access the webpage. These credentials are in the payload of the transaction. This differs from network observers such as Wireshark that cannot look at the credentials or even at the whole payload, that is because the transactions are encrypted using HTTPS so everything is hidden.
>
> Through debuggers all the credentials are visible instead.

Question: Check the "*Timing*" details of various HTTP transactions. Create the trellis graph (such as this one) for a standard HTTP/1.1 transaction over TLS, showing the most important packets. Map each of the timings reported by Chrome in time intervals in the trellis graph.



Scraping

Scraping means automatically downloading a webpage to extract any useful information. Your goal is to make a Bash script printing the **current temperature** in a nice sea place: **Muggia**

You must gather the data by inspecting the webpage: https://www.3bmeteo.com/centraline-meteo

Use the Chrome debugger, the `curl` tool and other bash tools (e.g., `grep` or `tail`), to make a script which prints ***uniquely*** the temperature in Celsius degree.

Hint: use the Network Tab of the debugger to find the HTTP request carrying the data on the weather stations, and use the "***Copy as Curl***" option to obtain a Bash command that replicates the specific HTTP request.

Report the **script** in the following box.

After opening the webpage "https://www.3bmeteo.com/centraline-meteo" I opened the Chrome DevTools and in the Network tab I looked for some type of API request that fetches weather station data. The data about the weather in some specific station is stored in a JSON file "rete_json.json". In order to get the information I copied it as a cURL and used it in a bash command file to obtain the final solution.

In order to achieve the desired solution it is possible to execute directly the inline bash commands instead of creating a separated bash file. The sequence of commands is the following:

curl 'https://retemeteo.lineameteo.it/rete_json.json?key=ZfzYzBpyeuPg6lve8HGitIfPlem4T7dw' -H 'Accept: */*' -H 'Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7' -H 'Connection: keep-alive' -H 'Cookie: TESTCOOKIESENABLED=1' -H 'If-Modified-Since: Tue, 10 Dec 2024 16:25:03 GMT' -H 'Referer: https://retemeteo.lineameteo.it/index_3bm.php' -H 'Sec-Fetch-Dest: script' -H 'Sec-Fetch-Mode: no-cors' -H 'Sec-Fetch-Site: same-origin' -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36' -H 'sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"' -H 'sec-ch-ua-mobile: ?0' -H 'sec-ch-ua-platform: "Linux"' | tail -c +11 | jq . | grep -i muggia -A -10 | grep temperature | cut -d"" -f 4

Performance Measurement

Now, you will use **BrowserTime**, a tool to measure web page performance. It is a JavaScript tool that instruments Google Chrome (and other browsers) to automatically visit a webpage. It also collects various **statistics** on the issued HTTP transactions and performance.

You will use the **dockerized** version, so that you don't have to install all the dependencies. The image name is: `sitespeedio/browsertime`. To test a webpage, visiting it `N` times, you only need to run the following command line:

```
sudo docker run --rm sitespeedio/browsertime -n <N> <webpage>
```

On the standard output, you will get various statistics on performance, while the full log of the visit is stored **in the container** at the path: `/browsertime`

To access it, you must choose local path and **mount** it in `/browsertime` in the container. Use the option:

```
-v <localpath>:/browsertime
```

Assignment: select a **website supporting HTTP/3** and measure how its performance varies with **different HTTP versions**. You must visit the website instrumenting Chrome to support i) only HTTP/1.1, ii) HTTP/1.1 and 2, iii) All HTTP versions. By default Chrome supports all HTTP versions (case iii). You can disable HTTP/2 with the Chrome's command line option `--disable-http2` and HTTP/3 with `--disable-quic`. To pass an option to Chrome within BrowserTime, you should add the **`--chrome.args`** option in the command line. For example:

```
docker run --rm sitespeedio/browsertime <page> --chrome.args="--disable-quic"
```

You goal is to visit the website **20 times with each protocol version** and compare the value of:

1. **Page Load Time**: you can use the BrowserTime standard output or the `browsertime.json` log file
2. **Speed Index**: same hint as above

Assignment: Compute and compare the average values of the above metrics for the three cases.

I decided to use a known website for testing which is www.google.com, since it also supports HTTP/3. I did not manage to do all steps and to code a successful bash file for outputting the final average values, but I tried it anyway and managed to do the initial steps. I started by trying to measure the performance in three different cases: with just HTTP/1.1, with both HTTP/1.1 and HTTP/2, and with all three HTTP versions. In each case, the webpage should have been visited 20 times using the BrowserTime in a Dockerized environment. Then the logs for each version were stored locally in the directory browsertime_logs and the information inside was ready to be analyzed. To compute the average Page Load Time and Speed Index I should write a bash script and extract the wanted metrics using the "jq" tool.

Optional – Artificial Delay

Repeat the same experiments as above imposing an **artificial delay** on the network.

To impose an artificial delay on Linux, you will use the **TC Netem** tool as follows:

1. Identify the physical WiFi or Ethernet interface name, e.g., **`eth0`**
2. Impose extra latency with the command:

   ```
   tc qdisc add dev eth0 root netem delay 100ms
   ```

3. Run the experiments using BrowserTime
4. Delete the artificial delay with: `tc qdisc del dev eth0 root`

Report your results making plots in the form of a **CDF** (Cumulative Distribution Function).

Hint: to create the plots from data, you can use your favourite library, tool or programming language. We recommend `Python` and `MatPlotLib`. Otherwise, `gnuplot` or Excel are fine.