HTTP/3 LAB

Bar Binyamin Varsulker, ..

Ori Meir Kushnir, ..

Preparation Questions

a. What is the main difference between TCP and UDP?

b. What are the advantages and disadvantages of using TCP versus UDP?

c. What is QUIC? And on which protocol is it based on?

d. What are the advantages of HTTP/3 compared to previous protocols (HTTP/1.1, HTTP/2)?

e. What problems of HTTP/1.1 does HTTP/2 solve?

f. What is Head-of-Line Blocking? How does it manifest in TCP?

g. How does HTTP/3 address Head-of-Line Blocking?

h. What is SCTP? What are the fields in the header?

i. What is multi-homing and multi-streaming in SCTP?

j. What are SCTP chunks?

k. Give 2 advantages and 2 disadvantages of SCTP.

l. What is a Docker container, and how is it different from a virtual machine?

m. Why is Docker useful for running a simple HTTP server in a lab environment?

Part 1 - HTTP/1.1, HTTP/2 and HTTP/3 with Caddy Web Server

Step 1: Create an Ubuntu Virtual Machine

- You can use virtualization tools such as VirtualBox or VMware.
- Required OS: Ubuntu 22.04.
- Minimum Resources: 1 CPU, 2GB RAM, 5GB free disk space.
- Update the VM by running the following command in the terminal:

  ```
  sudo apt update && sudo apt upgrade -y
  ```

Step 2: Install Docker on Ubuntu

- Install Docker engine:

  ```
  sudo apt install docker.io -y
  ```

- Enable and start Docker service:

  ```
  sudo systemctl enable docker
  sudo systemctl start docker
  ```

  Verify Docker is working:

  ```
  docker -version
  ```

Step 3: Configure a Simple Caddy Server

- Create a working directory:

  ```
  mkdir ~/site
  cd ~/site
  ```

- Create the Caddyfile:

  ```
  nano Caddyfile
  ```

- Paste the following content:

  ```
  localhost:443 {
      root * /srv
  ```

```
    file_server
    templates
    tls internal
}

localhost:80 {
    root * /srv
    file_server
    templates
}
```

Question 1: Explain each block in Caddyfile

- Create a webpage:

  ```
  nano ~/site/index.html
  ```

- Paste the following configuration and save it:

  ```
  <!DOCTYPE html>
  <html>
  <head><title>Connection Info</title></head>
  <body>
    <h1>Hello lab students!</h1>
    <p><strong>Using protocol:</strong> {{.Req.Proto}}</p>

  </body>
  </html>
  ```

Step 4: Run the Caddy Server Using Docker

- Run the following command:

```
docker run -d \
  -p 80:80 \
  -p 443:443 \
  -p 443:443/udp \
  -v ~/site:/srv \
  -v caddy_data:/data \
  caddy:latest \
  caddy run --config /srv/Caddyfile
```

Question 2: Explain the above command.

- Confirm that the container is running:

```
sudo docker ps
```

1. Send an HTTP/1.1 Request

- Install curl:

```
sudo apt install curl -y
```

  Question 3: Explain about curl.

- Open Wireshark and start capture on lo (loopback) interface.
  Use capture filters:

```
  - tcp.port == 80 (HTTP/1.1 plaintext)
  - tcp.port == 443 (HTTP/2)
  - udp.port == 443 (HTTP/3 QUIC)
```

- Run the command:

```
curl --http1.1 http://localhost
```

- Stop the capture.

Question 4: Explain what is a major disadvantage in HTTP/1.1 that you can see in the Wireshark pcap and add supporting screenshots.

2. Send an HTTP/2 Request

- Start a new Wireshark capture and run the following command:
  ```
  curl --http1.1 http://localhost
  ```

Question 5: Analyze HTTP/2 packet from the Wireshark pcap and add the packet to the final report. Compare It with the HTTP/1.1 packets from the previous question.

3. Send an HTTP/3 Request

- Start a new Wireshark capture and run the following command:
  ```
  curl --http3 -k --trace-ascii - https://localhost
  ```

- You'll probably get an error which says that the currently installed curl version doesn't support HTTP/3. Follow instructions in the appendix and afterwards run this command again.

Question 6: Compare the output of the above command with the HTTP/3 packets from Wireshark. What differences can be observed?

Part 2 - HTTP/2: Demonstrating TCP-Level Head-of-Line Blocking

In this part of the lab, you will observe and understand the TCP-level Head-of-Line blocking issue inherent in HTTP/2.

HTTP/2 allows multiple streams to be multiplexed over a single TCP connection. However, TCP guarantees in-order delivery across all bytes in the connection. If a packet is lost during transmission, TCP must hold back all following packets — even those belonging to different HTTP/2 streams — until the lost packet is successfully retransmitted.

This behavior leads to head-of-line blocking: a delay in delivering data from one stream blocks the progress of other streams sharing the same TCP connection.

Later, we will see how HTTP/3 resolves this issue.

Step 1 - Preparing the Test Files

Before starting the HTTP/2 HOL blocking experiment, we need to create and set up the files that the server will serve: one **small file** and two **large files**.

- **Create the files on your VM:**

  Run the following commands in ~/site directory to create the required files:

  ```
  # Create a small fast.html file (~100 bytes)
  echo "<html><body><h1>Fast Response</h1></body></html>" > fast.html


  # Create a large file (~20 MB) for slow1.dat
  dd if=/dev/urandom of=slow1.dat bs=1M count=20


  # Create a large file (~30 MB) for slow2.dat
  dd if=/dev/urandom of=slow2.dat bs=1M count=30
  ```

Step 2 - Introduce Network Impairments

Introduce network impairments on the loopback interface using tc (traffic control):

```
sudo tc qdisc add dev lo root netem delay 100ms loss 15%
```

Step 3 – Download multiple files in parallel over HTTP/2 using curl

Run the following commands:

```
   sudo curl --http2 -k --parallel --parallel-immediate \
     --output slow1_downloaded.txt \
     --output slow2_downloaded.txt \
     --output fast_downloaded.txt \
     --trace-ascii trace_http2.log \
     --trace-time \
     https://localhost/slow1.dat \
     https://localhost/slow2.dat \
```

```
  https://localhost/fast.html
```

```
grep -E 'GET|HTTP/2' trace_http2.log
```

Question 7:

i. **Explain these commands.**
ii. **Analyze the output and explain why does the small file (fast.html) might arrive last, even though HTTP/2 allows multiplexed streams? How is this affected by TCP's design? Attach screenshots.**

Step 4 –HTTP/2 vs. HTTP/3 – Head-of-Line Blocking

In this step, you will experimentally observe the differences between HTTP/2 and HTTP/3 in terms of Head-of-Line blocking and understand how HTTP/3 solves a major limitation of TCP-based HTTP/2.

Follow the instructions carefully:

- Make sure your Caddy server is correctly set up to serve all three files from the previous step.
- Download the files over HTTP/3:

```
sudo curl --http3 -k --parallel --parallel-immediate \
  --output slow1_downloaded.txt \
  --output slow2_downloaded.txt \
  --output fast_downloaded.txt \
  --trace-ascii trace_http3.log \
  --trace-time \
  https://localhost/slow1.dat \
  https://localhost/slow2.dat \
  https://localhost/fast.html
```

Question 8: Compare the output log here to the output log from the previous step. Can you conclude that there is no TCP-Level HOL blocking? Add screenshots.

Part 4 – SCTP

Step 1 – iperf Server Setup

In this part you will run Docker containers for iperf server and client to test SCTP communication.

In a terminal window, run the following command to start a Docker container with iperf server:

```
sudo docker run --rm -t --net=host --name iperf-server
sofianinho/iperf3:3.6-ubuntu18.04 -s --sctp
```

Step 2 – Test SCTP communication

1. Open Wireshark and start capture.
2. Apply filter: `sctp`.
3. In a new terminal window, run the following command:

   ```
   sudo docker run --rm -t --name iperf-client sofianinho/iperf3:3.6-
   ubuntu18.04 -c <server-ip> --sctp
   ```

   *server-ip can be obtained by running hostname -I (this is the host IP address).

4. Wait for the command to finish execution and then stop the capture.

Question 9: Compare the SCTP handshake to the TCP handshake that you are already familiar with and is used for instance in HTTP/2. What differences do you observe in terms of the number of steps? Which one includes cookie-based verification? Explain and add screenshots.

Question 10: Why do you not see HEARTBEAT packets in your capture?

Question 11: Compare SCTP to QUIC (used in HTTP/3). How does SCTP's multistreaming compare to QUIC's multiplexing?

Good Luck!

Appendix – Install curl with HTTP/3 Support

Follow these steps on Ubuntu to compile and install curl with HTTP/3 (QUIC) support alongside HTTP/1.1 and HTTP/2.

Step 1: Install Build Dependencies

```
sudo apt update
sudo apt install -y \
  git build-essential autoconf libtool pkg-config \
  libev-dev libssl-dev zlib1g-dev \
  cmake curl libbrotli-dev libnghttp2-dev \
  python3 python3-venv python3-pip
```

Step 2: Build and Install OpenSSL v3.5+

```
cd ~/Desktop
git clone --depth 1 -b openssl-3.5.0 https://github.com/openssl/openssl
cd openssl
./config --prefix=$HOME/opt/openssl --libdir=lib
make -j$(nproc)
make install
```

Step 3: Build and Install nghttp3

```
cd ~/Desktop
git clone -b v1.3.0 https://github.com/ngtcp2/nghttp3
cd nghttp3
git submodule update --init
autoreconf -fi
./configure --prefix=$HOME/opt/nghttp3 --enable-lib-only
make -j$(nproc)
make install
```

Step 4: Build and Install ngtcp2

```
cd ~/Desktop

git clone -b v1.3.0 https://github.com/ngtcp2/ngtcp2

cd ngtcp2

autoreconf -fi

./configure \

      PKG_CONFIG_PATH="$HOME/opt/openssl/lib/pkgconfig:$HOME/opt/nghttp3/
      lib/pkgconfig" \

      LDFLAGS="-Wl,-rpath,$HOME/opt/openssl/lib" \

      --prefix=$HOME/opt/ngtcp2 \

      --enable-lib-only \

      --with-openssl

make -j$(nproc)

make install
```

Step 5: Build and Install curl with HTTP/3 Support

```
cd ~/Desktop

git clone https://github.com/curl/curl

cd curl

autoreconf -fi

./configure \

  --with-ssl=$HOME/opt/openssl \

  --with-nghttp2 \

  --with-nghttp3=$HOME/opt/nghttp3 \

  --with-ngtcp2=$HOME/opt/ngtcp2 \

  --prefix=$HOME/opt/curl-http3 \

  LDFLAGS="-Wl,-rpath,$HOME/opt/openssl/lib"

make -j$(nproc)

make install
```

Step 6: Use the Compiled Version

Add it to your PATH:

```
export PATH=$HOME/opt/curl-http3/bin:$PATH
```

Verify:

```
curl --version
```

Expected output should include:

```
Features: ... HTTP1.1 HTTP2 HTTP3 ...
```