Lab3: Peeking under the Web - Solutions

COM-208: Computer Networks

The main goal of this lab is to "peek under the web", i.e., get a sense of how web browsers and web servers communicate: we will sniff and look inside **HTTP messages**, then play around with **caching** and **cookies**. We will close with a glance at an application that you use all the time: **streaming**.

Inside HTTP

HTTP (Hypertext Transfer Protocol) is the communication protocol used by the web application: it specifies the messages that may be exchanged between web clients (also called web browsers) and web servers.

The most common exchange is one where a web browser requests a resource, and a web server sends it; you will now initiate such an exchange and look inside the resulting messages. For this, you will use the **telnet** utility, which allows you to type in messages and send them to a remote process, i.e., a process running on a remote computer. Remember: there is only one process, behind each network interface, that is associated with a given port number; and all processes in the world that are associated with port number 80 are web-server processes.

Open a terminal and type:

```
telnet example.com 80
```

By typing this, you indicated that you want to communicate with the process that has name "example.com, 80" (network interface example.com, port number 80). As a result, the TCP code running on your computer generated a TCP connection setup request and sent it to the TCP code running on example.com; if that code accepted your request and sent a response, you got back a prompt, where you can type in messages that will be sent to the web-server process running on example.com.

Type:

```
GET / HTTP/1.1
host: example.com
[press <return> twice]
```

You just manually created an HTTP request. You specified that:

- the type of your message is GET,
- the resource you want is "/" (the home page of example.com),
- the version of the HTTP protocol you are using is 1.1,
- and the origin web server who owns the resource you want is example.com.

By pressing twice, you told the `telnet` utility that your message was ready to be sent.

What you just did manually is what your web browser does (among other things) under the covers when you type in a URL.

If all went as it should, you received an HTTP response from the web server process, which contains the base file of the target resource. If the HTTP request had been sent by your web browser (not by you via `telnet`), it is your web browser that would have received the response; it would have processed it, retrieved the URLs of all the referenced resources, and sent a new HTTP GET request for each resource.

- What is the content type and size of the HTTP response?

```
$ telnet example.com 80
GET / HTTP/1.1
host: example.com
```

The response is:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 205380
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Sun, 27 Sep 2020 17:35:05 GMT
Etag: "3147526947"
Expires: Sun, 04 Oct 2020 17:35:05 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (dcb/7EC6)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256

<!doctype html>
```

```
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=
    ↪ utf-8" />
    <meta name="viewport" content="width=device-width, initial-
    ↪ scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont
    ↪ , "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica,
    ↪ Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in
    ↪ documents. You may use this
    domain in literature without prior coordination or asking for
    ↪  permission.</p>
```

```
    <p><a href="https://www.iana.org/domains/example">More
    ↪ information...</a></p>
</div>
</body>
</html>
```

The content type is `Content-Type: text/html; charset=UTF-8` and the content length is `Content-Length: 1256`.

In other words, the HTTP response contains 1256 bytes of html (as UTF-8 characters).

- Use the same approach to get the same file, but make your request of type HEAD (instead of GET). How does the HTTP response differ?

```
$ telnet example.com 80
HEAD / HTTP/1.1
host: example.com
```

The response is:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 205385
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Sun, 27 Sep 2020 17:35:10 GMT
Etag: "3147526947"
Expires: Sun, 04 Oct 2020 17:35:10 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (dcb/7EC6)
X-Cache: HIT
Content-Length: 1256
```

The response differs from the previous one in that it does not include any message-body (any data), only the HTTP header. Some times you may also see a different Content-Length: the standard specifies that for a HEAD request the server *should* return the Content-Length it would return in case of a GET request, but it's *not a requirement*.

- Send a GET request like the first one, but for resource `index.html` (instead of `/`).

```
$ telnet example.com 80
GET /index.html HTTP/1.1
host: example.com
```

The response is:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 279238
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Sun, 27 Sep 2020 17:38:48 GMT
Etag: "3147526947+gzip"
Expires: Sun, 04 Oct 2020 17:38:48 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (dcb/7F83)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=
    ↪ utf-8" />
    <meta name="viewport" content="width=device-width, initial-
    ↪ scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont
    ↪ , "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica,
    ↪ Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
```

```
        @media (max-width: 700px) {
            div {
                margin: 0 auto;
                width: auto;
            }
        }
        </style>
    </head>

    <body>
    <div>
        <h1>Example Domain</h1>
        <p>This domain is for use in illustrative examples in
        ↪ documents. You may use this
        domain in literature without prior coordination or asking for
        ↪  permission.</p>
        <p><a href="https://www.iana.org/domains/example">More
        ↪ information...</a></p>
    </div>
    </body>
    </html>
```

Caching at the web browser

Web browsers **cache** resources, so that they don't need to download them again if the user requests them again. You will now experience the difference this browser behavior can make.

Use a Firefox web browser, if you can. It comes with a nice tool, the web-developer network console (≡/Web Developer/Network), which visualizes each HTTP request that the browser makes, as well as the correspoding HTTP response that the browser receives. If you click on an HTTP request from the list on the left, you will see all the relevant information in the panel on the right.

Get ready to capture web traffic:

- Open your web browser and clear the cache. To do so in Firefox: ≡/`Settings/Privacy & Security/Cookies and Site Data/Clear Data...`
- In Firefox, open the web-developer network console.
- Open Wireshark and start a new traffic capture.

Answer the following questions, using the web-developer network console, or Wireshark, or (ideally) both. Using Wireshark is a bit harder this time, but we will guide you:

- Visit Welcome to Rio. Where is this resource downloaded from?

> From an EPFL web server: From Wireshark, we see that the IP address of the web server is 128.178.32.48 (e.g., check the first SSL packet of type `Client Hello`). By running `host 128.178.32.48` we see that 128.178.32.48 maps to DNS name `moodle.epfl.ch`.

- How long did it take to download it?

> From the network console, we see that it took 129ms.

The web server where this resource is downloaded from uses a secure version of the HTTP protocol called HTTPs, which is, essentially, HTTP on top of SSL (the Secure Sockets Layer that we mentioned in class). This makes using Wireshark a bit harder: HTTP messages are encrypted within SSL packets, so Wireshark cannot simply display them. You need to:

- Apply the `ssl` filter to see all the SSL packets sent or received by your computer.
- Identify one of the packets sent by you to the web server or vice versa.
- Click on it, then go to Analyze/Follow/TCP Stream. Ignore/close the window that pops up. Now you should see only the packets that belong to the same TCP connection as the packet you chose.
- Look for the last SSL packet carrying encrypted `Application Data`. This is the packet that carried the resource from the web server to your computer.
- Look for an SSL packet of type `Client Hello`. This is the first packet that your computer sent to the web server to initiate their communication.
- Restart your web browser. Visit Welcome to Rio again. How long did it take to load it this time? What explains the difference?

> This time the image loaded faster. The reason is that the web browser had cached it, and it did not have to retrieve it again (just to check that its cached copy was fresh).

- Open a second tab in your web browser and visit Welcome to Rio II. Where is this resource downloaded from? How long did it take to download it?

From a web server in the US: From Wireshark, we see that the IP address of the web server is 91.198.174.208 (e.g., check the first SSL packet of type `Client Hello`). By using an IP geolocation tool (e.g., https://tools.keycdn.com/geo) we see that the web server with IP 91.198.174.208 is in the US.

From the network console, we see that it took 22ms.

- When you visited `Welcome to Rio II`, your web browser had already cached `Welcome to Rio`, which is essentially the same image. Do you think your browser served `Welcome to Rio II` from the cache, or it downloaded it from its origin web server? Why do you think your browser behaved this way?

The web browser downloaded the image from its origin web server. Web browsers identify images (and web objects in general) by URL. Given that the URLs of the two images are different, the web browser has no way of knowing that the content of the two images is the same, hence does not serve the second image from the cache.

Caching at a proxy web server

It is not only web browsers that cache resources; **proxy web servers** are web servers that act as **intermediaries**: they cache resources that are originally stored in other web servers (called **origin web servers**) and serve them to nearby web clients.

Before you start, clear your browser cache and find the proxy settings of your web browser. In Firefox, navigate to `about:preferences`, then `General/Network Settings/Settings`. Setup a proxy web server using the following settings: `Check "Manual proxy configuration", HTTP Proxy: 18.197.97.162, Port: 9300, Check "Also use this proxy for HTTPS"`. (you could use any proxy web server from https://free-proxy-list.net/ that does HTTPS caching).

Visit the same two resources that you visited before.

- Where were the resources downloaded from?

Both resources were downloaded from the proxy web server: From Wireshark, we see that the IP address of the web server is 18.197.97.162 (e.g., check the SSL packets of type `Client Hello`).

- How long did it take to download each resource this time? Why did the download time change?

From the network console, we see that it took 7335ms and 858ms to download `Welcome to Rio` and `Welcome to Rio II`, respectively. These times are longer compared to not using the proxy. The reason is the choice of the proxy: choosing a proxy which is further away and/or more busy than the origin web server may result in longer delays.

- What will happen to your web browser if the proxy web server that you specified fails? Will your browser be able to load any web pages? Can you think of a way to verify your answer?

Regarding new web pages, any attempt to load such a page will fail as traffic goes through the proxy. Regarding web pages that the browser has cached, the ability of the browser to load such a page depends on the freshness of the page (calculated based on several fields of the HTTP header): if the page is fresh, the browser will load it; if the page is stale, the browser will not be able to load it as the browser has to check with the proxy if the page has been modified (and does not receive a reply from the failed proxy). A good reference about HTTP caching is https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching.

To verify the above, you can start by specifying an invalid IP address as the proxy web server (e.g., 1.2.3.4).

IMPORTANT: Restore your original proxy settings.

Cookies

Cookies enable a web server to **link subsequent HTTP requests** to the same web browser: if you send 10 HTTP GET requests, for 10 different resources, to the same web server, the web server can use cookies to figure out that these 10 requests came from the

same web browser, even if you did not explicitly provide any identification information (e.g., you did not login).

Before you start, figure out how to control cookie settings in your browser. In Firefox:

- To allow or disallow your browser to exchange cookies with web servers: ≡`/Settings/Privacy & Security/Enhanced Tracking Protection/Custom`, and then select String or Standard.
- To view or delete the cookies that have been stored on your computer: ≡`/Settings/Privacy & Security/Cookies and Site Data/`, and then `Manage Data` or `Clear Data...`
- You can also view the cookies that your computer sends along with an HTTP request, or receives along with the corresponding response, through the web developer network console: select an HTTP request from the list of requests on the left, then select the `Cookies` menu from the panel on the right.

First, see cookies in action:

- Allow your browser to exchange cookies. Delete existing cookies. Visit MeteoSuisse. Did the MeteoSuisse web server send you any cookies?

> Yes, there is now one cookie for domain `www.meteosuisse.admin.ch` (and in some cases one from `player.vimeo.com`).

- By default, MeteoSuisse shows you the weather for Geneva. Choose another location for which you want to see the weather. Restart your web browser and re-visit MeteoSuisse. Do you get the weather for Geneva as before? Explain your browser's behavior.

> No, we get the weather for the last-visited location (e.g., Lausanne). This happens because, along with the current HTTP request, our web browser sent the cookie for `www.meteosuisse.admin.ch`, which contains information about our preferred (last-visited) location. This way the server "remembers" our preferences and offers a personalized website experience.

- Delete existing cookies. Restart your web browser and re-visit MeteoSuisse. Do you get the weather for Geneva or for your chosen location? Explain your browser's behavior.

We get the weather for the default location (Geneva): since we deleted the cookies, it is as if we visited the website for the first time.

Now think about **cookies as state**, as information about the user that can be exchanged between third parties:

- Visit Google. Once the resource has finished loading, view the cookies that have been stored on your computer. How many are they? Notice that each set of cookies is associated with a "site" or "domain", e.g., `google.ch`, or `youtube.com`. Which web server sent each of these cookies to your web browser?

Overall, 9 cookies were installed: 2 cookies for domain `google.com`, 5 cookies for `google.ch`, 1 cookie for `consent.google.ch`, and 1 cookie for `youtube.com`.

We see that additionally to cookies for the `google.ch` domain, our computer has installed **third-party** cookies, e.g., for `youtube.com`. These cookies were sent to our web browser upon visiting a web server of the respective domain (a YouTube server in the case of `youtube.com`) to retrieve components that are referenced in the page we have originally requested (www.google.ch).

Note: Your results may differ if Google changes the page.

- Visit YouTube. Did your web browser send along any cookie when it contacted the YouTube web server? Which one?

Yes, it sent cookies for `google.com`, `google.ch`, `static.doubleclick.net`, `googleads.g.doubleclick.net' and`youtube.com‘.

- View again the cookies that have been stored on your computer. How do you think your web browser decides which cookie(s) to send along with each HTTP request?

Based on the domains it communicates with and the paths or sub-paths requested from these domains.

Think about your web browser's communications. Did the Google and YouTube web servers just exchange information about you without talking to each other directly?

> Yes, they communicated through your computer's local storage.

IMPORTANT: Restore your original cookie settings.

Back to layers, headers, encapsulation...

Open this webpage in Firefox and start playing the music (it might play automatically): https://soundcloud.com/relaxdaily/instrumental-music-to-relax

Open the web-developer network console and find the audio stream (you can filter "Media" streams from the button bar). Start a Wireshark capture.

- Which application-layer protocol carries the audio messages?

> HTTP 2: In the network console, click on an HTTP request. This will open a panel on the right. On this panel, go to the "Headers" tab where `Version HTTP/2`.

- What is the content type and content size?

> From the "Response Headers" we can see that `Content-Type: audio/mpeg` and `Content-Length: 159661` (bytes) (varies per response).

- Which transport-layer protocol encapsulates the audio messages?

> TCP. We cannot find out from the network console, only from Wireshark.