The Web [*]

Jens Lechtenbörger

VM Neuland im Internet 2021

Contents

1 Introduction

1.1 Learning Objectives

- Explain message format and `GET` requests of HTTP as application protocol
- Perform simple HTTP requests via `telnet` or `gnutls-cli`

1.1.1 Recall: Internet Architecture

- “Hourglass design”

[*]This PDF document is an inferior version of an OER HTML page; free/libre Org mode source repository.

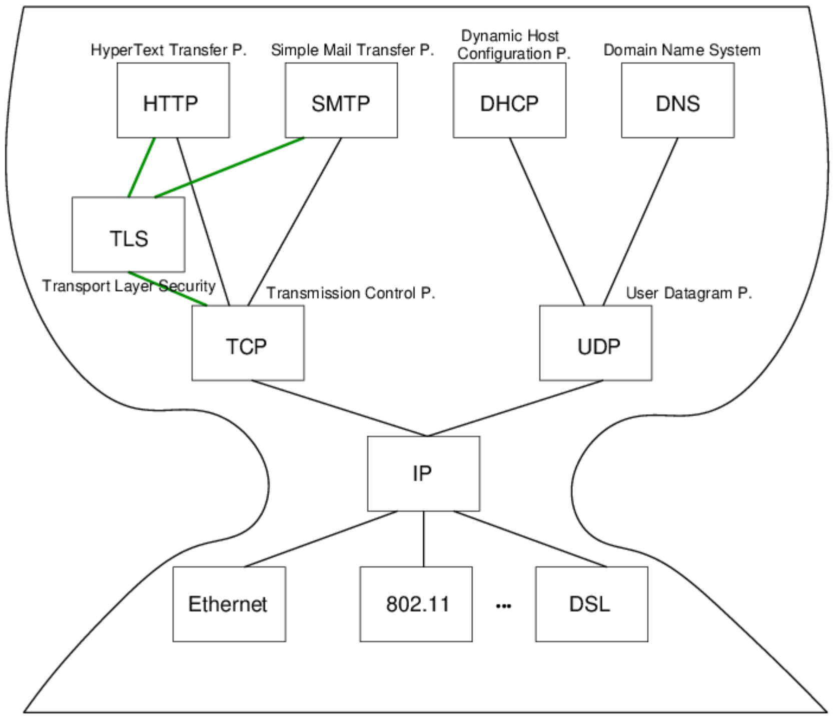

Figure 1: Internet Architecture with narrow waist

- IP is focal point
  - "Narrow waist"
  - Application independent!
    - Everything over IP
  - Network independent!
    - IP over everything
- Now: **HTTP** at application layer

1.2 Today's Core Question

- What does your browser do when you enter a URI in the address bar?

2 Web

2.1 History of the Web (1/2)

- 1945, Vannevar Bush: As we may think
  - Memex for information storage
  - Associative indexing (Hyperlinks)
- 1989, article by Tim Berners-Lee

  - Distributed hypertext system, “»web« of notes with links”
  - Initially for cooperation among physicists at CERN
- May 1991
  - Distributed information system based on HTML, HTTP, and client software at CERN
- August 1991
  - Availability of CERN files announced in `alt.hypertext`

2.2 History of the Web (2/2)

- 1992, NCSA **Web Server** available
  - National Center for Supercomputing Applications, University of Illinois, Urbana-Champaigne
- 1993, Mosaic **browser** created at NCSA
- 1994, World Wide Web Consortium (W3C) founded by Tim Berners-Lee
  - Publication of technical reports and “recommendations”
- Now
  - Web 2.0, Semantic Web, cloud computing, browser as access device

2.3 WWW/Web

- Standards
  - W3C (HTML 4 Specification)
    - “The World Wide Web (Web) is a network of information resources.”
  - HTTP/1.1 Specification (RFC 7230)
    - “The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems.”
- Distributed information system
  - Client-Server architecture
    - Web clients (browsers) and servers exchange HTTP messages based on Internet standards
  - Sample Web standards (application layer of Internet architecture)
    - URIs (Uniform Resource Identifiers, generalize URLs and URNs)
    - HTTP (now)
    - ((X)HTML)

3 HTTP

3.1 HTTP

- Hypertext Transfer Protocol
  - HTTP/1.1, RFC 7230
    - Plain text messages, discussed subsequently
  - HTTP/2, RFC 7540
    - Adds frame format with compression
    - Adoption increasing, from 15% in July 2017 to 28% in July 2018, to 33.4% in July 2019, to 50% in January 2021
  - HTTP/3 also upcoming
- Request/response protocol
  - Specific message format
  - Several access methods
  - Requires reliable transport protocol
    - Typically TCP/IP, port 80 (or port 443 for HTTPS)

3.2 Excursion: Manual Connections

- HTTP (before HTTP/2) and SMTP are plain text protocols
  - With encrypted variants HTTPS and SMTPS (or STARTTLS)
- Enables experiments on the command line
  - Type (or copy&paste) request, see server response
  - For unencrypted connections, `telnet` can be used (preinstalled or available for lots of OSs)
  - For encrypted connections, `gnutls-cli` can be used (part of GnuTLS, which is free software)
    - TLS = Transport Layer Security
      - · Successor to SSL
      - · Layer between application layer and TCP, recall Internet architecture
      - · Secure channels based on asymmetric cryptography

3.2.1 Warnings

- Next two slides demonstrate how to type HTTP commands (for an improved understanding of the protocol)
  - Subsequent examples with `www.informationelle-selbstbestimmung-im-internet.de` **require** GnuTLS
    - Server redirects from port 80 to port 443

    - If your manual typing is too slow, connections may **time out** (e.g., “Peer has closed the GnuTLS connection”)
    - Also, use of backspace or cursor keys may destroy connections
- Suggestion: Type in text editor and copy&paste into command line

3.2.2 telnet

- Original `telnet` purpose: Login to remote host
    - **Insecure** plaintext passwords
    - Nowadays, remote login performed with Secure Shell, `ssh`
- Establish **TCP connection** to destination port
    - `telnet www.google.de 80` (port 80 for HTTP)
        * (For variants without visual feedback possibly followed by `ctrl-+` or `ctrl-]`, `set localecho` [enter] [enter])
        * `GET / HTTP/1.1` [enter]
        * `Host: www.google.de` [enter] [enter]
        * (Context for above lines soon)
    - **Beware**: Buggy telnet implementations may stop sending after first line (use Wireshark to verify)

Here, you see a sample use of `telnet` to open a TCP connection to port 80 on a Google server. You could try out any other number to check on what ports the server is prepared to talk with you. Port 80 is reserved for HTTP, which is slowly phased out in favor of the cryptographically secured variant HTTPS on port 443.

Anyways, once a TCP connection is established successfully, you can send data to the server by typing it. When typing, you need to “speak” the protocol that is expected by the server, here HTTP, and the lines starting with `GET` as well as with `Host` are both part of the HTTP protocol, which is explained on later slides.

Note that you cannot use `telnet` with encrypted connections as you would need to type bytes that setup and use cryptographic protocols then. Thus, while you can *open* a TCP connection to port 443 with `telnet`, it is unlikely that you can *use* that connection by typing the necessary bytes for cryptographic protocols afterwards.

For cryptographically secured connections, you may want to use the GnuTLS client as shown on the next slide.

An aside: On the slide, “ctrl-+” means: Press the `ctrl` key and `+` simultaneously. Similarly for other keys.

3.2.3 gnutls-cli

- Establish TLS protected TCP connection with GnuTLS
    - Alternative to `telnet` on previous slide
    - `gnutls-cli --crlf www.informationelle-selbstbestimmung-im-internet.de`
        * (HTTPS on port 443 by default)
        * `GET /chaosreader.html HTTP/1.1` [enter]
        * `Host: www.informationelle-selbstbestimmung-im-internet.de` [enter] [enter]
    - SMTP for e-mail, port 587 as alternative to 25

            * `gnutls-cli --crlf --starttls -p 587 secmail.uni-muenster.de`
                - (Type `ehlo localhost`, then `starttls`; press `ctrl-d` to enter TLS mode; needs authentication)

The cryptographic protocol suite TLS is used in two major variants.

1. A special port, e.g., 443, is reserved for cryptographically secured connections. The connecting client (here, GnuTLS) must immediately "talk" a cryptographic protocol.
2. A single port, e.g., 25, supports plaintext as well as cryptographically secured connections. Here, the client starts with plaintext (as with `telnet`), but can issue a specific command (here, `starttls` followed by `ctrl-d`) to switch to a cryptographically secured connection.
    - "ctrl-d" means: Press the `ctrl` key and `d` simultaneously.

In any case, application data is transmitted through secure channels.

3.3 Excursion: Browser Tools

- Modern browsers offer developer tools
    - E.g., press `ctrl-shift-I` with Firefox
    - Tools to inspect HTML, CSS, Javascript
    - Tools to inspect HTTP traffic (Network tab)
        * Live view on browser requests and server responses
            - With details on timing, caching, headers
    - Console with error messages
    - And much more

3.4 HTTP Messages

- Requests and responses
    - Generic message format of RFC 822, 1982 (822→2822→5322)
        * Originally for e-mail, extensions for binary data
            - Lines end with CRLF, `\r\n` below
    - Messages consist of
        * Headers
            - In HTTP always a distinguished start-line (request or status)
            - Then zero or more headers
        * Empty line
        * Optional message body
    - Sample `GET` **request** (does not have a body)

        *
          ```
          GET /chaosreader.html HTTP/1.1\r\n
          Host: www.informationelle-selbstbestimmung-im-internet.de\r\n
          \r\n
          ```
- Excerpt of sample HTTP **response** to previous `GET` request

  -
    ```
    HTTP/1.1 200 OK\r\n
    Date: Wed, 08 Apr 2020 13:30:10 GMT\r\n
    Server: Apache\r\n
    Last-Modified: Wed, 24 Jul 2019 12:25:46 GMT\r\n
    ETag: "2cd1-58e6c6898dce2"\r\n
    Content-Length: 11473\r\n
    more headers omitted
    Content-type: text/html; charset=utf-8\r\n
    \r\n
    HTML code as body
    ```

3.5 HTTP Methods

- Case-sensitive (capital letters)
  - `GET` (Request for resource, see section 4.3.1)
  - `HEAD` (Request information on resource, see section 4.3.2)
  - `POST` (Transfers entity, see section 4.3.3)
    * Annotations, postings, forms, database extensions
  - `PUT` (Creates new resource on server, see section 4.3.4)
  - `DELETE` (Deletes resource from server, see section 4.3.5)
  - `CONNECT` (Establish tunnel with proxy, see section 4.3.6)
  - `OPTIONS` (Asks for server capabilities, see section 4.3.7)
  - `TRACE` (Tracing of messages through proxies, see section 4.3.8)

3.6 Conditional GET

- `GET` under conditions
  - Requires (case-insensitive) request header
    * (Can be used by browser to check if cached version still fresh)
    * `If-Modified-Since`
    * `If-Match`
    * `If-None-Match`
- Example
  - Request

    *
      ```
      GET /chaosreader.html HTTP/1.1
      Host: www.informationelle-selbstbestimmung-im-internet.de
      If-None-Match: "2cd1-58e6c6898dce2"
      ```
  - Response

    *
      ```
      HTTP/1.1 304 Not Modified
      Date: Wed, 08 Apr 2020 14:07:31 GMT
      additional headers
      ```

Please revisit the response for an earlier HTTP request. Note that the response contains a `Last-Modified` date and an `ETag`. Both pieces can be used for conditional gets. While the date is probably self-explanatory, the ETag is some version identifier provided by the server. Changed page contents are reflected in changed ETag values (but not necessarily the other way round).

On this slide, you see a conditional GET with the ETag value `"2cd1-58e6c6898dce2"` from the previous response. As the server's ETag value did not change, it responds with status code 304, indicating that no modification took place. Hence, a cached result would still be fresh and usable, saving bandwidth and reducing transmission delays.

3.7 Sample Status Codes

- Three digits, first one for class of response
  - 1xx: Informational - Request received, continuing process
    - 100: Continue - Client may continue with request body
  - 2xx: Successful - Request successfully received, understood, and accepted
    - 200: OK
  - 3xx: Redirection - Further action necessary to complete request
    - 302: Found (temporarily under different URI)
    - 303: See Other (redirect to different URI in `Location` header)
    - 304: Not Modified (previous slide)
  - 4xx: Client Error - Request with bad syntax or cannot be fulfilled
    - 403: Forbidden
    - 404: Not Found
  - 5xx: Server Error - Server failed for apparently valid request

4 Conclusions

4.1 Summary

- Web browsers and servers talk HTTP
  - Simple message format
  - More details in CACS

License Information

This document is part of a larger course. Source code and source files are available on GitLab under free licenses.

Except where otherwise noted, the work "The Web", © 2018-2021 Jens Lechtenbörger, is published under the Creative Commons license CC BY-SA 4.0.