Web and E-Mail

Jens Lechtenbörger

Summer Term 2018

Contents

1 Introduction

1.1 Learning Objectives

- Perform simple HTTP requests via `telnet` or `gnutls-cli`
- Explain the concept of "stateless servers"
- Explain constraints and advantages of caching
- Interpret E-Mail headers
- Discuss alternatives to and weaknesses of e-mail security established by secure channels between MUA and MTA

1.2 Previously on CACS . . .

1.2.1 Communication and Collaboration

- Communication frequently takes place via the **Internet**
  - Telephony
  - Instant messaging

  - E-Mail
  - Social networks
- Collaboration frequently supported by tools using **Internet** technologies
  - All of the above means for communication
  - ERP, CRM, e-learning systems
  - File sharing: Sciebo, etherpad, etc.
  - Programming (which subsumes file sharing): Git, subversion, etc.
- All of the above are instances of **DSs**

1.2.2 Recall: Internet Architecture

- "Hourglass design"

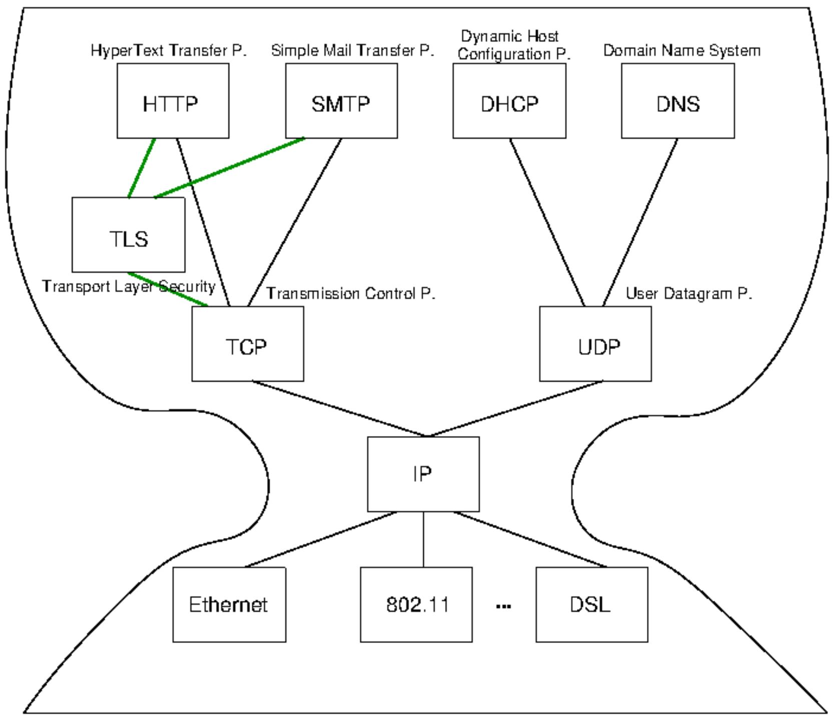

Figure 1: Internet Architecture with narrow waist

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

1.3 Today's Core Questions

- What does your browser do when you enter a URI in the address bar?
- How does e-mail transfer work?

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`
    - `http://groups.google.com/group/alt.hypertext/msg/395f282a67a1916c`

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 browser (client) sends HTTP requests to Web server
  - Based on
    - Internet
    - 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 (as of 2018-07-15)
- 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 telnet

- Original purpose: Login to remote host (plaintext passwords)
  - Nowadays, we use Secure Shell, `ssh`, for that
- Still, `telnet` can establish **arbitrary TCP connections**
  - `telnet www.google.de 80`
    - (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)
  - `telnet wi.uni-muenster.de 25`
    - (Revisited later on)
  - **Beware**: Buggy telnet implementations may stop sending after first line (use Wireshark to verify)

3.2.2 gnutls-cli

- Establish TLS protected TCP connection
  - Alternative to `telnet` on previous slide
  - `gnutls-cli --crlf www.informationelle-selbstbestimmung-im-internet.de`
    - `GET /Anonymes_Surfen_mit_Tor.html HTTP/1.1` [enter]
    - `Host: www.informationelle-selbstbestimmung-im-internet.de` [enter] [enter]
  - `gnutls-cli --crlf --starttls -p 25 wi.uni-muenster.de`
    - Type `ehlo localhost`, then `starttls`; press ctrl-d to enter TLS mode

3.3 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 /newsticker/ HTTP/1.1\r\n
      Host: www.heise.de\r\n
      User-Agent: Mozilla/5.0\r\n
      \r\n
      ```
- Sample HTTP **response** to previous `GET` request

  -
    ```
    HTTP/1.1 200 OK\r\n
    Date: Tue, 02 Nov 2010 13:49:26 GMT\r\n
    Server: Apache\r\n
    Vary: Accept-Encoding,User-Agent\r\n
    Content-Encoding: gzip\r\n
    Content-Length: 20046\r\n
    Connection: close\r\n
    Content-Type: text/html; charset=utf-8\r\n
    \r\n
    gzip'ed HTML code as body
    ```

3.4 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)

3.5 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 /Anonymes_Surfen_mit_Tor.html HTTP/1.1
      Host: www.informationelle-selbstbestimmung-im-internet.de
      If-None-Match: "4fc5-568ed5e21e210"
      ```
  - Response

    -
      ```
      HTTP/1.1 304 Not Modified
      Date: Mon, 16 Jul 2018 08:23:07 GMT
      additional headers
      ```

3.6 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
    - 304: Not Modified
  - 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

3.7 HTTP Connection Management

- Options
  - **Short-lived** connections: Each request on separate TCP connection
  - **Persistent** connections
    - TCP connection reused for multiple HTTP requests
      - HTTP/1.0: Connection: Keep-Alive
      - HTTP/1.1: Persistence by default

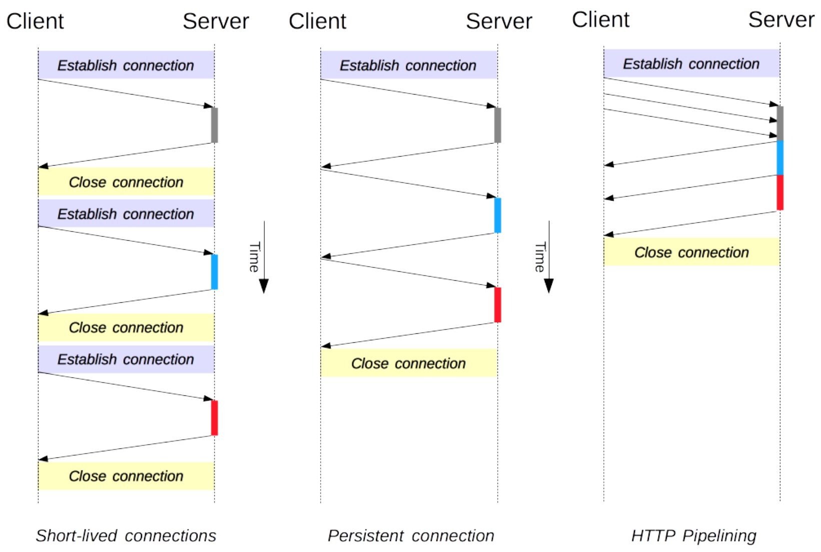

Figure 2: "HTTP/1.x connection management" by Mozilla Contributors under CC BY-SA 2.5; from MDN web docs

- **Pipelined** connections
  - Client may send multiple requests over single TCP connection before receiving a response
  - HTTP/1.1 compliant servers support pipelining
    - Not activated in browsers by default, compatibility and performance problems

3.8 Review Question

- Did you execute GET requests and conditional GET requests on the command line? Any surprises?

4 Server State and Cookies

4.1 State Models

- **Stateless**: Server does not maintain client state
  - Advantages
    - State changes on server do not require client notifications
    - Recovery (restart after server crash) "simple": No client state to restore
  - E.g.: HTTP
    - Web server forgets client after request
    - No session
- **Stateful**: Server maintains client state

  - E.g., file server with table of pairs (Client, File) for caching
    - Keep track which client has current version
    - Performance improvement via locality
  - Recovery requires to restore consistent state

4.2 Stateful Web Applications

- HTTP is stateless
  - Yet, Web applications often maintain client state
- E.g., personalized session after login
  - Virtual shopping cart
  - Shopping history, preferences
  - Exercises in Learnweb

4.3 Session IDs

- Need identifier to keep track of subsequent requests
- Two major variants
  - Session ID embedded in dynamically generated URIs
    - May hinder caching
      - URI does not identify resource any longer
  - Cookies
    - Piece of data, sent by server S, stored by browser
    - Browser includes cookies set by S for every subsequent visit of S
      - Think of automatic ID card
      - You may want to configure your browser to discard cookies upon exit

4.4 Cookies (1/2)

- RFC 6265: HTTP State Management Mechanism
  - Idea
    - Client stores data sent by server
    - Client sends this data with subsequent requests
      - Without understanding that data at all
  - Details
    - Cookie is **named byte string**
    - Server transfers cookie in `Set-Cookie` (2) header in response
      - `Set-Cookie`: Version 0/Netscape and RFC 6265
      - `Set-Cookie2`: Version 1/RFC 2965
      - (JavaScript may create cookie at client)
    - Client sends cookie in `Cookie` header in requests

4.5 Cookies (2/2)

- Cookies have name, value, optional attributes/flags
  - `Expires`, `Max-Age`
    - Determine lifetime of cookie
    - If both missing: "Session" cookie to be deleted when browser exits
  - `Domain`
    - DNS domain of servers to which the cookie should be sent
  - `Path`
    - Restrict sending of cookie based on directory path
  - `Secure`
    - Should only be sent via HTTPS
  - `HttpOnly`
    - Script access restricted; XSS counter-measure

5 Caching

5.1 HTTP Caching

- HTTP caching assumptions

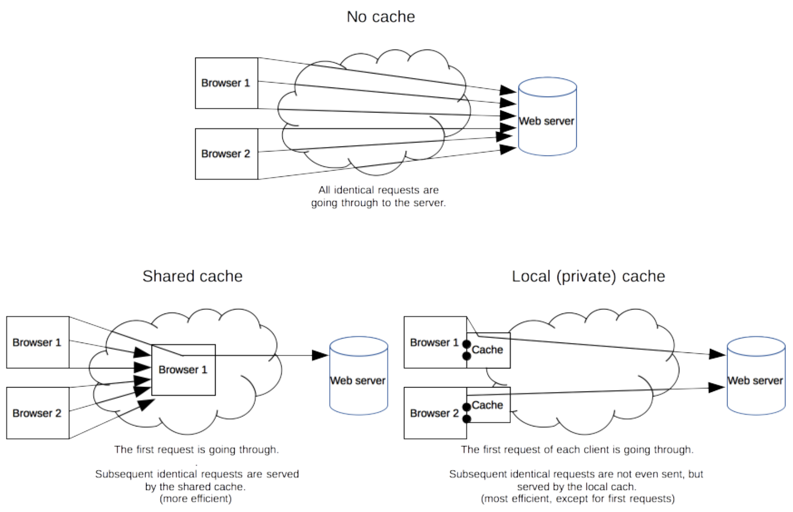
No cache
Browser 1
Browser 2
Web server
All identical requests are going through to the server.
Shared cache
Browser 1
Browser 2
Browser 1
Web server
The first request is going through.
Subsequent identical requests are served by the shared cache.
(more efficient)
Local (private) cache
Browser 1
Cache
Browser 2
Cache
Web server
The first request of each client is going through.
Subsequent identical requests are not even sent, but served by the local cach.
(most efficient, except for first requests)

Figure 3: "HTTP cache types" by Mozilla Contributors under CC BY-SA 2.5; from MDN web docs

  - URI identifies resource, stability, client-independence

- Semantic transparency
  - Caching is not visible to users
  - Response from cache is equivalent to hypothetical one from server

5.2 HTTP Caching Mechanisms

- Expiration
  - Server may indicate expiration date in Expires or Cache-Control header
- Validation
  - After expiration date, cache must check whether resource still usable
  - May return new expiration date
    - Conditional `GET` (“Slow hit”)

5.3 HTTP Caching Rules

- Complex rules, lots of details
  - (Some details on Cache Control header)
- Server may limit caching
  - `no-store`, `no-cache`, `must-revalidate`
- Client may
  - enforce validation
    - `no-cache`
  - forbid caching
    - `no-store`

6 Proxies

6.1 Web Proxies

- Web proxy server is intermediary between client and server
  - Acts as server to client
  - Acts as client to server

6.2 Sample Proxy Applications

- Cache
- Firewall/Content filter
- Anonymizer, e.g., Tor
  - My privacy policy recommends surfing via Tor
- Debugging tool
  - E.g., intercept and analyze app network data
- Surrogate/Reverse proxy, Content Delivery Network (CDN)

  - Replicated contents, inbound messages intercepted and redirected, e.g.:
    * Load balancing
    * Geographical diversity (reduced latency, increased availability)

6.3 Review Questions

- What is a stateless protocol? Given that HTTP is a stateless protocol, how can lots of applications that apparently require state be implemented on top of HTTP?
- Where are HTTP caches typically located? What impact might HTTPS have on caching?

7 E-Mail

7.1 E-Mail Basics

- Among oldest Internet applications
- Message format
  - RFC 822 seen above
  - Extended with Multipurpose Internet Mail Extensions (MIME)
    * `Content-Type` (type of data contained in message)
    * `Content-Transfer-Encoding` (how data in message body is encoded)
- Plaintext messages
  - E-mail is like **postcard**, written with **erasable pencil**
    * Neither confidentiality nor integrity
  - Learn self-defense, use GnuPG if you don't like this
    * SSL/TLS insufficient approach, recall end-to-end security

7.2 Message Transfer

- Terminology
  - **Mail User Agent** (MUA): Your mail reader
    * E.g., browser, Thunderbird, Emacs
  - **Mail Transfer Agent** (MTA): Mail server/daemon
    * E.g., sendmail, exim, postfix
- **Simple Mail Transfer Protocol**, 1982 (SMTP, RFC 821→2821→5321)
  - Outgoing messages, MUA-to-MTA, MTA-to-MTA
    * Plaintext (TCP/IP, port 25)

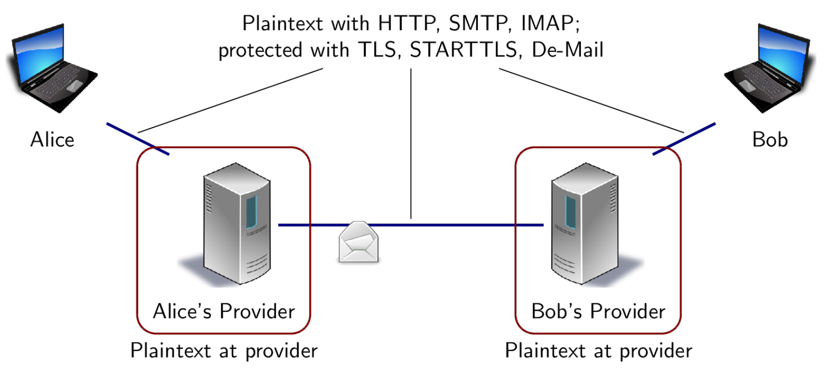

Figure 4: Hop-to-hop security of e-mail

7.3 SMTP

```
telnet wi 25
Trying 128.176.159.139...
Connected to wi.uni-muenster.de.
Escape character is '\^]'.
220 wi-vm700.wi1.uni-muenster.de Microsoft ESMTP MAIL Service ready at Tue, 27 Oct 2009 11
HELO mouse.nix
250 wi-vm700.wi1.uni-muenster.de Hello [128.176.159.107]
MAIL From: micky@mouse.nix
250 2.1.0 Sender OK
RCPT To: lechten@wi.uni-muenster.de
250 2.1.5 Recipient OK
DATA
354 Start mail input; end with <CRLF>.<CRLF>
Received: from mx1.disney.com ([192.195.66.20]) by smtp.mouse.nix Super Duper SMTP Server;
To: 42@universe.com
From: micky@mouse.nuix
Subject: Don't panic

Somebody Else's Problem!  (This is the message body after the empty
line.  Note that headers preceding the empty line have also been
entered manually.  They are ignored by SMTP, but displayed to user.)

.

250 2.6.0 <b13a2a36-f56b-43ec-ad81-41ec44190e6a@wi-vm700.wi1.uni-muenster.de> Queued mail
```

7.4 SMTP MUA Header

```
Microsoft Mail Internet Headers Version 2.0
Received: from wi-vm700.wi1.uni-muenster.de ([128.176.158.92]) by wi-vmail2005.wi1.uni-mue
Received: from mouse.nix (128.176.159.107) by wi-vm700.wi1.uni-muenster.de (128.176.159.13
Received: from mx1.disney.com ([192.195.66.20]) by smtp.mouse.nix Super Duper SMTP Server;
To: 42@universe.com
```

```
From: <micky@mouse.nuix>
Subject: Don't panic
MIME-Version: 1.0
Content-Type: text/plain
Message-ID: <b13a2a36-f56b-43ec-ad81-41ec44190e6a@wi-vm700.wi1.uni-muenster.de>
Return-Path: micky@mouse.nix
Date: Tue, 27 Oct 2009 11:22:28 +0100
X-OriginalArrivalTime: 27 Oct 2009 10:22:35.0473 (UTC) FILETIME=[66C35410:01CA56EF]
```

7.5 Review Questions

- Who will find the previous e-mail in his inbox?
- How do you expect it to look like?
- What parts of header data are trustworthy (to what degree)?

8 Conclusions

8.1 Summary

- Web browsers and servers talk HTTP
  - Simple message format
  - Stateless request/response protocol
    - State via cookies
  - Different connection types
  - Caching for performance
- E-Mail transferred via SMTP

8.2 Outlook

- HTTP used for various applications
  - Web services
    - SOAP messages
  - Ad-hoc request/reply protocols
- REST
  - Representational State Transfer
  - Software architecture for distributed hypermedia systems
    - Generalization of Web
    - Defining constraints
      - Client/Server
      - Stateless
      - Cacheable

- Uniform interface, may use: URIs, MIME types, HTTP methods
- Layered System
- (Code on demand)

License Information