Computer Networks

Mohammed El-Hajj

Jacobs University Bremen

October 10, 2021

JACOBS
UNIVERSITY

Course Content

1. Introduction
2. Fundamental Networking Concepts
3. Local Area Networks (IEEE 802)
4. Internet Network Layer (IPv4, IPv6)
5. Internet Routing (RIP, OSPF, BGP)
6. Internet Transport Layer (UDP, TCP)
7. Firewalls and Network Address Translators
8. Domain Name System (DNS)
9. Document Access and Transfer (HTTP, FTP)

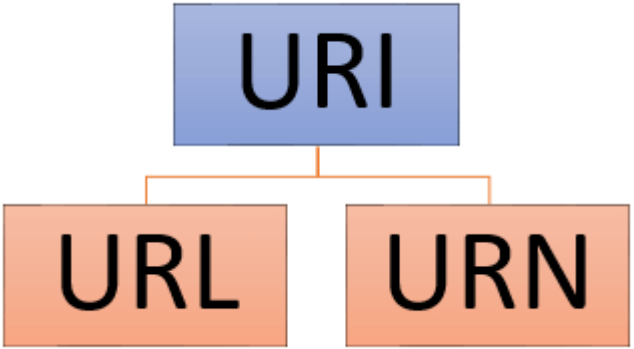
Part 9: Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP)

- HTTP version 1.0 was published in 1996 in RFC 1945.
- HTTP version 1.1 was originally defined in 1997 in RFC 2068 and later revised in 1999 in RFC 2616. It became one of the core building blocks of the World Wide Web. The latest revision is published in RFCs 7230-7235.
- HTTP was designed to retrieve and manipulate documents (resources) maintained on HTTP servers. The protocol runs on top of TCP and it uses the well known port number 80. It uses MIME conventions to distinguish different media types.
- RFC 2817 describes how to use TLS with HTTP 1.1. The secure version of HTTP uses the well known port number 443.
- HTTP version 2.0 was published in 2015 in RFC 7540. It supports multiple streams multiplexed over a single connection, it uses a binary encoding, and it enables servers to push data to clients.

Part 9: Hypertext Transfer Protocol (HTTP)

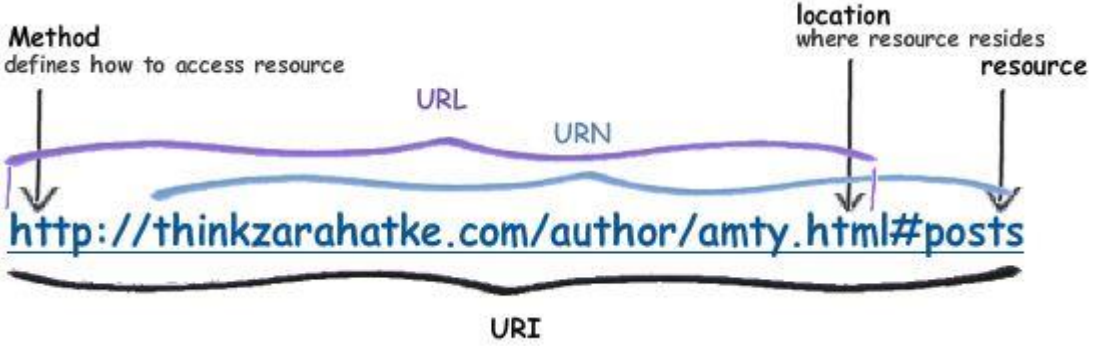
URL, URI, URN, IRI, . . .



URL, URI, URN, IRI, . . .

- *Uniform Resource Identifier* (URI)
  A URI is a sequence of characters from the US-ASCII for identifying an abstract or physical resource.
- *Uniform Resource Locator* (URL)
  The term URL refers to the subset of URIs that provide a means of locating the resource by describing its primary access mechanism (e.g., its network ”location”).
- *Uniform Resource Name* (URN)
  The term URN refer to the subset of URIs that identify resources by means of a globally unique and persistent name.
- *Internationalized Resource Identifier* (IRI)
  An IRI is a sequence of characters from the Universal Character Set for identifying an abstract or physical resource.

URL, URI, URN, IRI, . . .



URL and URN Examples

1. http://www.example.com/
2. http://www.example.com:80/
3. http://www.example.com/thesis#part3
4. http://www.example.com/student?name=bart
5. http://www.example.com/escape%20me
6. mailto:j.looser@example.com
7. tel:+1-201-555-0123
8. file:///tmp/useless.txt

1. urn:isbn:3-8273-7019-1
2. urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6

URI Syntax in ABNF

```
URI           = scheme ":" hier-part [ "?" query ] [ "#" fragment ]

hier-part     = "//" authority path-abempty
              / path-absolute
              / path-rootless
              / path-empty

URI-reference = URI / relative-ref

absolute-URI  = scheme ":" hier-part [ "?" query ]

relative-ref  = relative-part [ "?" query ] [ "#" fragment ]

relative-part = "//" authority path-abempty
              / path-absolute
              / path-noscheme
              / path-empty

scheme        = ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )
```

URI Syntax in ABNF

```
authority     = [ userinfo "@" ] host [ ":" port ]
userinfo      = *( unreserved / pct-encoded / sub-delims / ":" )
host          = IP-literal / IPv4address / reg-name
port          = *DIGIT

IP-literal    = "[" ( IPv6address / IPvFuture  ) "]"

IPvFuture     = "v" 1*HEXDIG "." 1*( unreserved / sub-delims / ":" )

IPv6address   =                            6( h16 ":" ) ls32
              /                       "::" 5( h16 ":" ) ls32
              / [               h16 ] "::" 4( h16 ":" ) ls32
              / [ *1( h16 ":" ) h16 ] "::" 3( h16 ":" ) ls32
              / [ *2( h16 ":" ) h16 ] "::" 2( h16 ":" ) ls32
              / [ *3( h16 ":" ) h16 ] "::"    h16 ":"   ls32
              / [ *4( h16 ":" ) h16 ] "::"              ls32
              / [ *5( h16 ":" ) h16 ] "::"              h16
              / [ *6( h16 ":" ) h16 ] "::"
```

URI Syntax in ABNF

```
h16           = 1*4HEXDIG
ls32          = ( h16 ":" h16 ) / IPv4address

IPv4address   = dec-octet "." dec-octet "." dec-octet "." dec-octet

dec-octet     = DIGIT                 ; 0-9
              / %x31-39 DIGIT         ; 10-99
              / "1" 2DIGIT            ; 100-199
              / "2" %x30-34 DIGIT     ; 200-249
              / "25" %x30-35          ; 250-255

reg-name      = *( unreserved / pct-encoded / sub-delims )

path          = path-abempty    ; begins with "/" or is empty
              / path-absolute   ; begins with "/" but not "//"
              / path-noscheme   ; begins with a non-colon segment
              / path-rootless   ; begins with a segment
              / path-empty      ; zero characters
```

Section 45: HTTP 1.1 Methods

HTTP 1.1 Methods

Method	Description
GET	Retrieve a resource identified by a URI
HEAD	Retrieve meta-information of a resource identified by a URI
POST	Annotate an existing resource by passing information to a resource
PUT	Store information under the supplied URI (may create a new resource)
DELETE	Delete the resource identified by the URI
OPTIONS	Request information about methods supported for a URI
TRACE	Application-layer loopback of request messages for testing purposes
CONNECT	Initiate a tunnel such as a TLS or SSL tunnel

- A client invokes a *method* on a *resource* by sending a *request message*
- A server returns a *response message*, which may include a *representation* of the accessed *resource*

Safe and Idempotent Methods

- Safe methods:
  - Safe methods are intended only for information retrieval and should not change the state of the server
  - Safe methods should not have side effects, beyond relatively harmless effects such as logging
  - The methods GET, HEAD, OPTIONS and TRACEare defined to be safe

- Idempotent methods:
  - Idempotent methods can be executed multiple times without producing results that are different form a single execution of the method
  - The methods PUT and DELETEare idempotent.
  - Safe methods should be idempotent as well

Method Properties

Method	Req Body	Res Body	Safe	Idempotent	Cacheable
GET	No	Yes	Yes	Yes	Yes
HEAD	No	No	Yes	Yes	Yes
POST	Yes	Yes	No	No	Yes
PUT	Yes	Yes	No	Yes	No
DELETE	No	Yes	No	Yes	No
OPTIONS	Opt	Yes	Yes	Yes	No
CONNECT	Yes	Yes	No	No	No
TRACE	Yes	Yes	Yes	Yes	No

- Supporting caches well is a fundamental goal of HTTP

HTTP 1.1 ABNF

```
Message      =  Request / Response

Request      =  Request-Line *(message-header CRLF) CRLF [ message-body ]
Response     =  Status-Line  *(message-header CRLF) CRLF [ message-body ]

Request-Line =  Method SP Request-URI SP HTTP-Version CRLF

Status-Line  =  HTTP-Version SP Status-Code SP Reason-Phrase CRLF

Method       =  "OPTIONS" / "GET" / "HEAD" / "POST"
Method       =/ "PUT" / "DELETE" / "TRACE" / "CONNECT"
Method       =/ Extension-Method

Extension-Method = token
```

HTTP 1.1 ABNF (cont.)

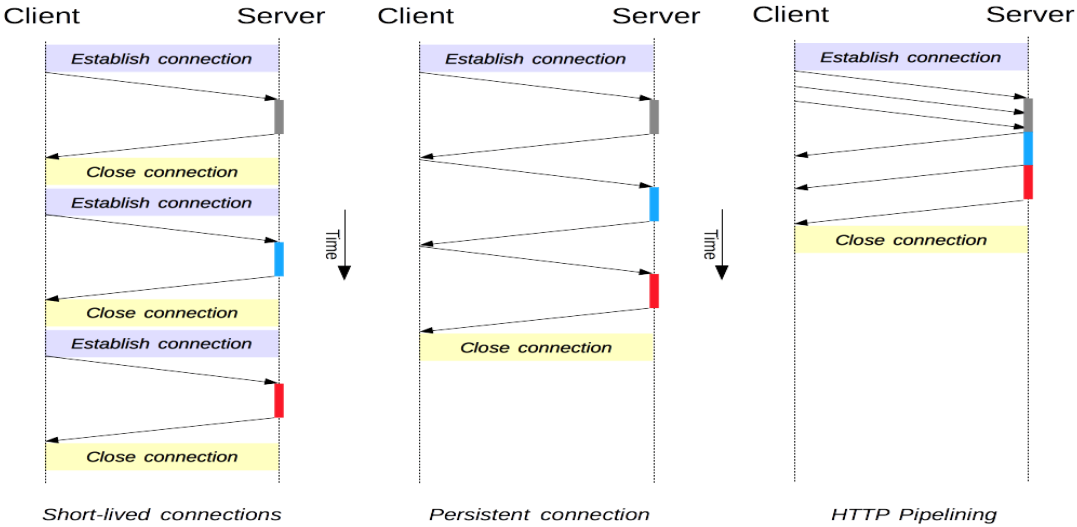
```
Request-URI    = "*" | absoluteURI | abs_path | authority
HTTP-Version   = "HTTP" "/" 1*DIGIT "." 1*DIGIT
Status-Code    = 3DIGIT
Reason-Phrase  = *<TEXT, excluding CR, LF>

message-header = field-name ":" [ field-value ]
field-name     = token
field-value    = *( field-content / LWS )
field-content  = <the OCTETs making up the field-value
                 and consisting of either *TEXT or combinations
                 of token, separators, and quoted-string>
```

Section 46: HTTP 1.1 Features

Persistent Connections and Pipelining

- A client can establish a persistent connection to a server and use it to send multiple Request messages.
- HTTP relies on the MIME Content-Length header field to detect the end of a message body (document).
- Early HTTP versions allowed only a single Request/Response exchange over a single TCP connection, which is of course rather expensive.
- HTTP 1.1 also allows clients to make multiple requests without waiting for each response (pipelining), which can significantly reduce latency.

Persistent Connections and Pipelining



Chunked Transfer Encoding

- Supports streaming of data where the content length is initially not known
- Uses the Transfer-Encoding HTTP header in place of the Content-Length header
- Content is send in small chunks.
- The size of each chunk is sent right before the chunk itself.
- The end of the data transfer is indicated by a final chunk of length zero.

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked
5\r\n
Media\r\n
8\r\n
Services\r\n
4\r\n
Live\r\n
0\r\n
\r\n
```

Caching and Proxies

- Probably the most interesting and also most complex part of HTTP is its support for proxies and caching.
- Proxies are entities that exist between the client and the server and which basically relay requests and responses.
- Some proxies and clients maintain caches where copies of documents are stored in local storage space to speedup future accesses to these cached documents.
- The HTTP protocol allows a client to interrogate the server to determine whether the document has changed or not.
- Not all problems related to HTTP proxies and caches have been solved. A good list of issues can be found in RFC 3143.

Negotiation

- Negotation can be used to select different document formats, different transfer encodings, different languages or different character sets.
- Server-driven negotiation begins with a request from a client (a browser). The client indicates a list of its preferences. The server then decides how to best respond to the request.
- Client-driven negotiation requires two requests. The client first asks the server what is available and then decides which concrete request to send to the server.
- In most cases, server-driven negotiation is used since this is much more efficient.

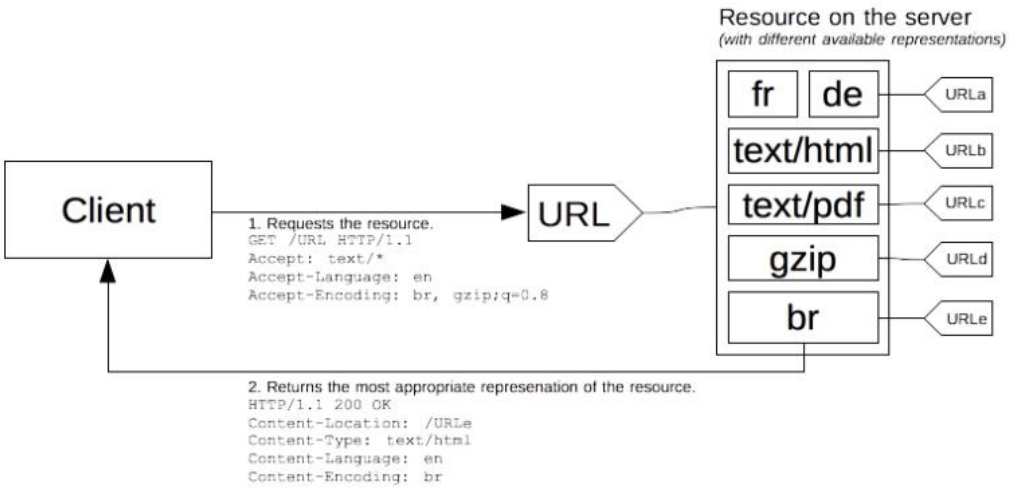
Negotiation Example

- The following is an example of typical negotiation header lines:

  Accept: text/xml, application/xml, text/html;q=0.9, text/plain;q=0.8 Accept-Language: de, en;q=0.5
  Accept-Encoding: gzip, deflate, compress;q=0.9
  Accept-Charset: ISO-8859-1, utf8;q=0.66, *;q=0.33

- The first line indicates that the client accepts text/xml, application/xml, and text/plain and that it prefers text/html over text/plain.
- The last line indicates that the client prefers ISO-8859-1 encoding (with preference 1), UTF8 encoding with preference 0.66 and any other encoding with preference 0.33.

Negotiation Example



Conditional Requests

- Clients can make conditional requests by including headers that qualify the conditions under which the request should be honored.
- Conditional requests can be used to avoid unnecessary requests (e.g., to validate cached data).
- Example:

  If-Modified-Since: Wed, 26 Nov 2003 23:21:08 +0100

- The server checks whether the document was changed after the date indicated by the header line and only processes the request if this is the case.

Entity Tags

- An entity tag (ETag) is an opaque identifier assigned by a web server to a specific version of a resource found at a URL.
- If the resource content at that URL ever changes, a new and different ETag is assigned.
- ETags can be quickly compared to determine whether two versions of a resource are the same.
- A client can send a conditional request using the If-None-Match header.
- Example:

  If-None-Match: "686897696a7c876b7e"

- ETags can be used like cookies for tracking users…

Some Extensions

- Patch Method (RFC 5789)
  - A method to apply partial modifications to a resource.
- Delta Encoding (RFC 3229)
  - A mechanism to request only the document changes relative to a specific version.
- Web Distributed Authoring (RFC 2518, RFC 3253)
  - An extension to the HTTP/1.1 protocol that allows clients to perform remote web content authoring operations.
- HTTP as a Substrate (RFC 3205)
  - HTTP is sometimes used as a substrate for other application protocols (e.g., the Internet Printing Protocol).
  - There are some pitfalls with this approach as documented in RFC 3205.

Section 47: HTTP 2.0

HTTP Evolution

- HTTP/1.0
  - separate TCP connection for every resource request
  - published as RFC 1945 in May 1996
- HTTP/1.1
  - persistent connections and pipelining
  - conditional requests
  - published in RFC 2068 in Jan 1997, revised as RFC 2616 in Jun 1999
  - modularized version published as RFCs 7230-7235 in Jun 2014
- HTTP/2
  - header compression (binary encoding)
  - multiplexing (avoiding head-of-line blocking)
  - server push into client caches
  - published as RFC 7540 in May 2015

Referenc

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee.
Hypertext Transfer Protocol – HTTP/1.1.
RFC 2616, UC Irvine, Compaq/W3C, Compaq, W3C/MIT, Xerox, Microsoft, W3C/MIT, June 1999.

R. Khare and S. Lawrence.
Upgrading to TLS Within HTTP/1.1.
RFC 2817, 4K Associates / UC Irvine, Agranat Systems, May 2001.

E. Rescorla.
HTTP Over TLS.
RFC 2818, RTFM, May 2000.

D. Robinson and K. Coar.
The Common Gateway Interface (CGI) Version 1.1. RFC
3875, Apache Software Foundation, October 2004.

J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein.
Delta encoding in HTTP.
RFC 3229, Compaq WRL, AT&T, Univ. of Saarbruecken, Marimba, ERS/USDA, January 2002.

Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen.
HTTP Extensions for Distributed Authoring – WEBDAV.
RFC 2518, Microsoft, UC Irvine, Netscape, Novell, February 1999.

Reference

G. Clemm, J. Amsden, T. Ellison, C. Kaler, and J. Whitehead.
Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning).
RFC 3253, Rational Software, IBM, Microsoft, U.C. Santa Cruz, March 2002.

K. Moore.
On the use of HTTP as a Substrate.
RFC 3205, University of Tennessee, February 2002.

R. Fielding and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing.
RFC 7230, Adobe, greenbytes, June 2014.

R. Fielding and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.
RFC 7231, Adobe, greenbytes, June 2014.

R. Fielding and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests.
RFC 7232, Adobe, greenbytes, June 2014.

R. Fielding, Y. Lafon, and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1): Range Requests.
RFC 7233, Adobe, W3C, greenbytes, June 2014.

Reference

R. Fielding, M. Nottingham, and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1): Caching.
RFC 7234, Adobe, Akamai, greenbytes, June 2014.

R. Fielding and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1): Authentication.
RFC 7235, Adobe, greenbytes, June 2014.

M. Belshe, R. Peon, and M. Thomson.
Hypertext Transfer Protocol Version 2 (HTTP/2).
RFC 7540, BitGo, Google, Mozilla, May 2015.