HTTP

Godmar Back

Virginia Tech

December 3, 2024

VIRGINIA TECH.

Outline

- Goal: obtain working knowledge of the Hypertext Transfer Protocol HTTP, the primary protocol underlying the world-wide web as well as modern web-based applications
- Started in 1991 with what's now referred to a HTTP 0.9 by Tim Berners-Lee at CERN
- HTTP/1.0 and HTTP/1.1 in 1996, 1997 (updated by RFC 7230-7235 and in 2022 by RFC 9110-9112)
- HTTP/2.0 (2015, RFC 7540) and HTTP/3.0 are mainly transport/security enhancements that preserve semantics
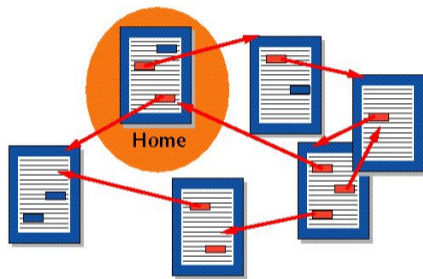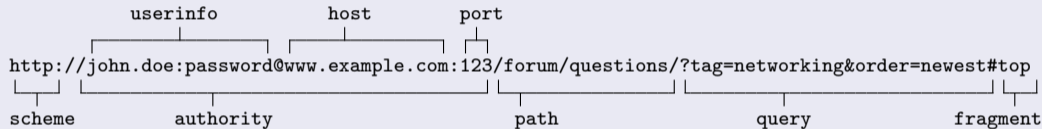

Figure 1: Documents connected by hyperlinks. Source: WikiPedia

VIRGINIA TECH.

HTTP - Basics

- Request/response protocol where user agents request arbitrary objects from a server
  - Very simple syntax; requests and responses use a similar format
- Request: request an arbitrary object/resource denoted by a path derived from a uniform resource locator, or URL
  - Few request types ("methods" GET, POST, PUT, DELETE, HEAD, OPTIONS, PATCH)
- Response: returns objects (comprised of octets/bytes), along with some metadata
  - anything from HTML, text, images, video, audio
  - metadata to interpret it (content type + encoding)
  - also supports streaming of larger objects

VIRGINIA TECH.

Uniform Resource Locators - URLs

Structure

```
              userinfo          host          port
          ┌──────┴──────┐    ┌───┴───┐      ┌─┴─┐
http://john.doe:password@www.example.com:123/forum/questions/?tag=networking&order=newest#top
└─┬─┘  └───────────┬───────────────────────┘ └────────┬────────┘ └────────────┬────────────┘ └┬┘
scheme          authority                           path                     query          fragment
```

- Common schemes: `http`, `https`, `file`
- Default port depends on scheme
- Query string is a sequence of `key=value` pairs

HTTP Transaction Example 1

https://duckduckgo.com/html?q=http

```
GET /html?q=http HTTP/1.1
Host: duckduckgo.com
User-Agent: curl/7.58.0
Accept: */*
```

```
HTTP/1.1 200 OK
Server: nginx
Date: Sun, 26 Apr 2020 01:02:56 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Expires: Sun, 26 Apr 2020 01:02:57 GMT
Cache-Control: max-age=1
X-DuckDuckGo-Locale: en_US
```

HTTP Transaction Example 1

https://duckduckgo.com/html?q=http

```
GET /html?q=http HTTP/1.1    <- method path?query version
Host: duckduckgo.com         <- domain (in case of virtual hosts)
User-Agent: curl/7.58.0      <- User agent is curl
Accept: */*                  <- Accept any content type
```

```
HTTP/1.1 200 OK        <- HTTP version + status + message
Server: nginx          <- various headers
Date: Sun, 26 Apr 2020 01:02:56 GMT
Content-Type: text/html; charset=UTF-8  <- body is utf8-encoded HTML
Transfer-Encoding: chunked  <- body will be sent in chunks
Expires: Sun, 26 Apr 2020 01:02:57 GMT
Cache-Control: max-age=1
X-DuckDuckGo-Locale: en_US
```

HTTP Transaction Example 2

http://optiplex:10000/api/login

```
POST /api/login HTTP/1.1
Host: optiplex:10000
User-Agent: curl/7.58.0
Accept: */*
Content-Type: application/json
Content-Length: 45

{"username":"user0","password":"thepassword"}
```

```
HTTP/1.1 200 OK
Server: CS3214-Personal-Server
Set-Cookie: auth_token=eyJhbGc ... 81Phms; Path=/
Content-Type: application/json
Content-Length: 49

{"exp":1588010897,"iat":1587924497,"sub":"user0"}
```

HTTP Transaction Example 2

http://optiplex:10000/api/login

```
POST /api/login HTTP/1.1
Host: optiplex:10000
User-Agent: curl/7.58.0
Accept: */*
Content-Type: application/json  <- request body type
Content-Length: 45              <- request body length

{"username":"user0","password":"thepassword"}
```

```
HTTP/1.1 200 OK
Server: CS3214-Personal-Server
Set-Cookie: auth_token=eyJhbGc ... 81Phms; Path=/
Content-Type: application/json
Content-Length: 49

{"exp":1588010897,"iat":1587924497,"sub":"user0"}
```

HTTP Transaction Example 3

http://optiplex:10000/private/secret.txt

```
GET /private/secret.txt HTTP/1.1
Host: optiplex:10000
User-Agent: curl/7.58.0
Accept: */*
Cookie: auth_token=eyJhb .... fv24M9Ijl1ePpM81Phms
```

```
HTTP/1.1 200 OK
Server: CS3214-Personal-Server
Content-Length: 12
Content-Type: text/plain

Secret File.
```

HTTP Transaction Example 4

http://optiplex:10000/private/secret.txt

```
GET /private/secret.txt HTTP/1.1
Host: optiplex:10000
User-Agent: curl/7.58.0
Accept: */*
```

```
HTTP/1.1 403 Permission Denied
Server: CS3214-Personal-Server
Content-Length: 18

Permission denied.
```

HTTP is a stateless protocol. Servers do not maintain state across requests as part of the protocol. Clients must present information that allows servers to recognize them as part of an earlier interaction or ongoing session. Usually cookies or bearer tokens are used.

HTTP Request Methods

GET Request transfer of target resource in selected representation

HEAD Like GET, except only metadata, no body

POST Process data sent in request, possibly creating a new resource

PUT Update target resource

DELETE Delete target resource

CONNECT Create a tunnel to target resource

OPTIONS Inquire about server options or policies

TRACE, PATCH ...

Common HTTP Response Status Codes

200 OK	Success
301 Moved Permanently	Follow direction in Location:
304 Not Modified	Resource hasn't changed, use cached copy
400 Bad Request	Client send ill-formed request
401 Unauthorized	Unauthenticated
403 Forbidden	Authenticated, but unauthorized
404 Not Found	Resource doesn't exist
418 I'm a Teapot	Attempt to brew coffee in a teapot [1]
500 Internal Server Error	Something went wrong that shouldn't have
502 Bad Gateway	Can't connect to upstream

HTTP and the Transport Layer

- Fetching a typical webpage involves multiple HTTP transactions to (often) different servers to retrieve JavaScript, style sheets, fonts, images, etc. etc.
- As of 2020, HTTP is most often run over TCP, directly or indirectly (via Transport Layer Security/TLS)
- As such, transport layer properties have strong influence on HTTP performance
- Let's look at how as we trace through the versions of HTTP

HTTP 1.0

- HTTP 1.0 created a new TCP connection for each resource it requested
- The request can be sent with the 3rd leg of the handshake
- Time needed: $2 \times \text{RTT} + T_t$ where RTT is the round-trip time for a small packet and T_t is the transmit time for the object
- Inefficient, especially for small resources (relative to bandwidth) and high propagation delay (e.g. WAN) connections
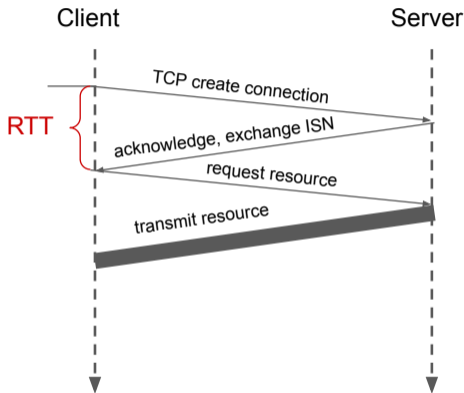

Figure 2: HTTP 1.0

HTTP 1.1 - Persistent Connections

- Reuses one connection for multiple transfers
- Common scenario:
  - User fetches say www.vt.edu/index.html, finds numerous references to objects on multiple servers `www.vt.edu`, `www.assets.cms.vt.edu`
  - what's the fastest way to fetch them?
- Q1: How many connections should be established to each server?
  - Servers want this number to be low (protection from overload). Networks want this number to be low (fairness, congestion control).
- Q2: Which requests should be sent over which connection and when?
  - Clients want to use the connection that is served quickest by a server
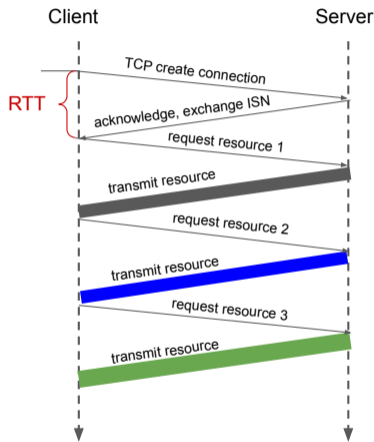
Figure 3: HTTP 1.1

HTTP 1.1 Pipelining

- Sending requests for next resources as soon as known can, in theory, allow a server to send resources back-to-back

- Leads to HOL (head of line) blocking: resource 3 cannot be sent until after resource 2 is ready to be sent

- Plus, servers aren't implemented to fetch resources in parallel on the same connection – though servers can almost always handle multiple connections in parallel

- Result: clients use multiple connections and did not use pipelining. How many?
  - RFC 2616: 2 per server.
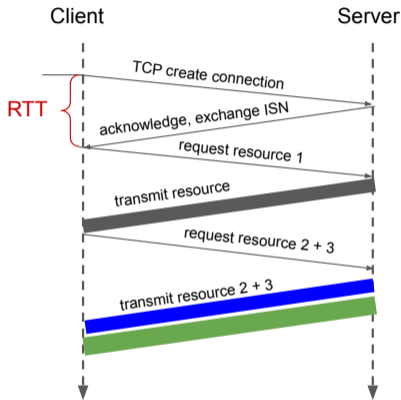 - RFC 7230: Clients should be *"conservative when opening multiple connections."*

Figure 4: HTTP 1.1 with pipelining

HTTP 2.0

- No longer first-come, first-serve: client can specify transmission order preference
- Server can send objects back in any order, in fact, they can be sent divided into frames (parts) which can be mixed and are reassembled
- Server push: server may sent objects before they are requested[1]
- Header compression: no longer ASCII
- Goal: reduce incentive for clients to open multiple connections
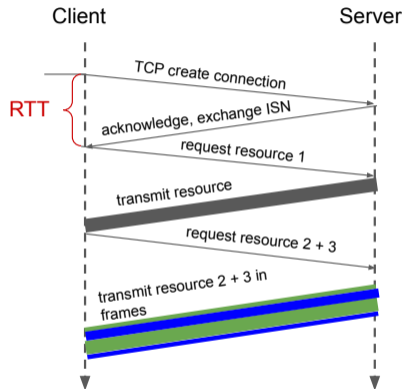 - Still, packet loss on the underlying TCP transport can temporarily stall connection

Figure 5: HTTP 2.0

---

[1]Interestingly, seems to not work out: Chrome removed support for it as of 2022

Transport Layer Security

- Recent years have seen a trend to use `https` - HTTP over a secure transport, e.g. RFC 7258
  - Can use any of a family of TLS protocols
  - Fully transparent to HTTP layer
  - Generally, provides (a) encryption and (b) server authentication via certificates and a public key infrastructure
- From a performance perspective, we now have an additional TLS handshake following the TCP handshake in which suitable TLS protocols are negotiated
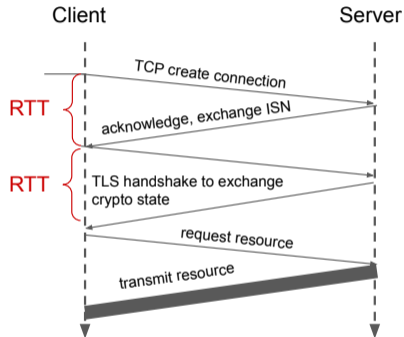

Figure 6: HTTP over TLS

HTTP/3 over QUIC: Quick UDP Internet Connections

- QUIC: Proposal by Google:
  https://www.chromium.org/quic
- HTTP/3 uses QUIC
- Uses UDP instead of TCP to avoid blocking entire stream on packet loss
  - Reimplements reliability on a per-stream basis
  - Reimplements congestion control
  - Combines connection establishment handshakes and crypto handshakes (0 RTT if crypto state can be reused)
  - Forward error correction and connection migration
- Already widely used in mobile video

Figure 7: QUIC Logo

References

[1] Larry M Masinter.
Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0).
RFC 2324, April 1998.

VIRGINIA TECH.