HTTP

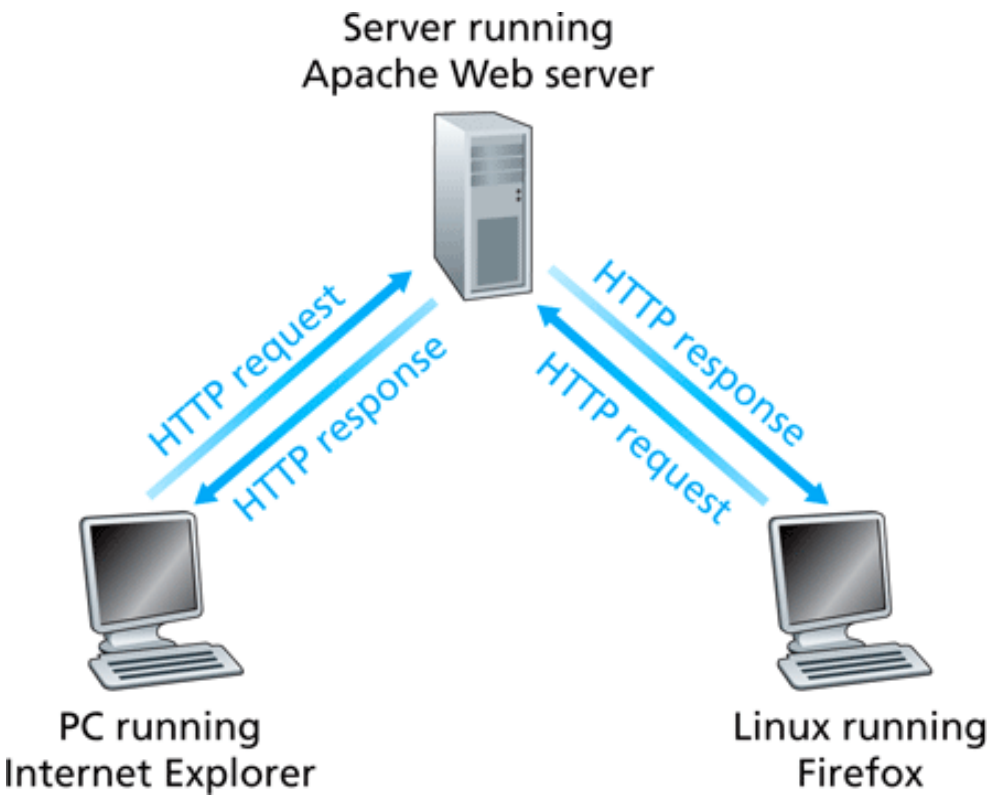
Daniel Zappala

CS 360 Internet Programming
Brigham Young University

HTTP Objects

Requesting Objects

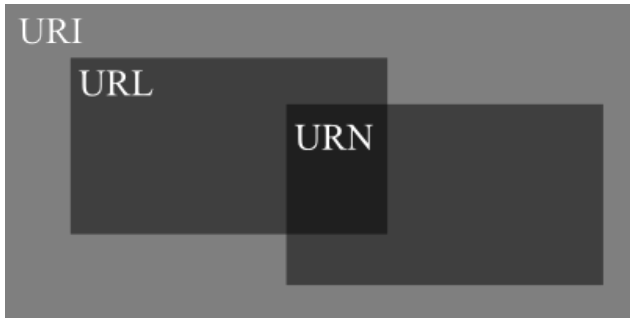
- clients request objects from servers using the HTTP protocol
  - client sends an HTTP request
  - server sends an HTTP response
- does not necessarily have a GUI
  - collecting hourly reports on competitor's prices
  - mining Twitter data



Web Objects

- object names: Uniform Resource Identifier (URI)
  - a name that refers to a resource
  - a Uniform Resource Locator (URL) is one type of URI
  - popular URL schemes: http, ftp, gopher, mailto
- object delivery: Hypertext Transfer Protocol (HTTP)
  - IETF standard
  - defines message format for making requests and receiving responses
- object format: Hypertext Markup Language (HTML)
  - representation of documents in ASCII format
  - many other formats - XHTML, XML, PNG, JPG, PDF, etc.

URIs, URLs, and URNs

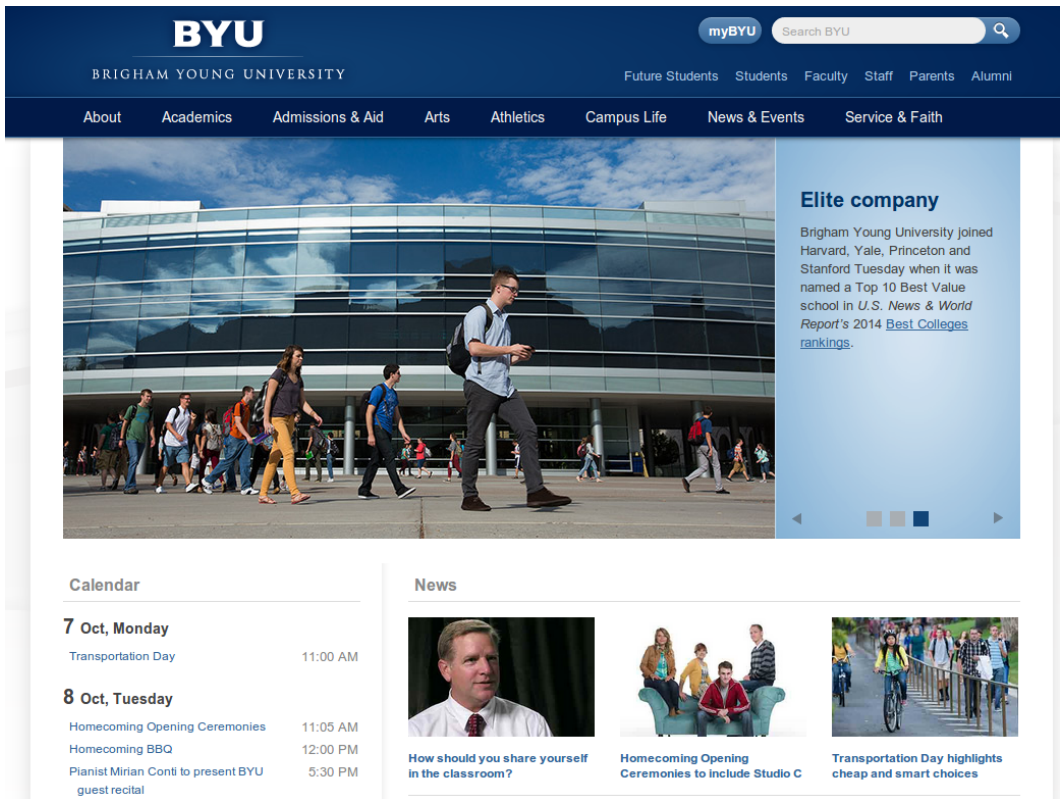


- URI: Uniform Resource Identifier
  - The generic set of all names that refer to resources
- URL: Uniform Resource Locator
  - An informal term (no longer used in technical specifications) associated with popular URI schemes: http, ftp, gopher, mailto, etc.
- URN : Uniform Resource Name
  - A URI that has an institutional commitment to persistence, availability, etc. May also be a URL: see PURLs.
  - persistent, location-independent resource identifiers, urn: specified by RFC 2141

▸ http://www.w3.org/Addressing/

Container Objects

- a web page consists of a container object, which may link to other objects
- fetching a web page consists of requesting the container object and then requesting any linked objects (images, CSS, Javascript, etc.)



Handling Objects

- determines how responses are handled
  - appearance (fonts)
  - content transformations (language)
  - whether to accept cookies
  - whether to allow javascript, popups
  - MIME types and handlers
- see Chrome advanced settings

HTTP Message Formats

HTTP Standards

- HTTP 1.0
  - RFC 1945: ▸ http://www.ietf.org/rfc/rfc1945.txt
  - very basic protocol, documenting what earliest servers and browsers used
- HTTP 1.1
  - RFC 2616: ▸ http://www.ietf.org/rfc/rfc2616.txt
  - backward compatibility with HTTP/1.0, plus many improvements and features
  - what all modern servers and browsers uses

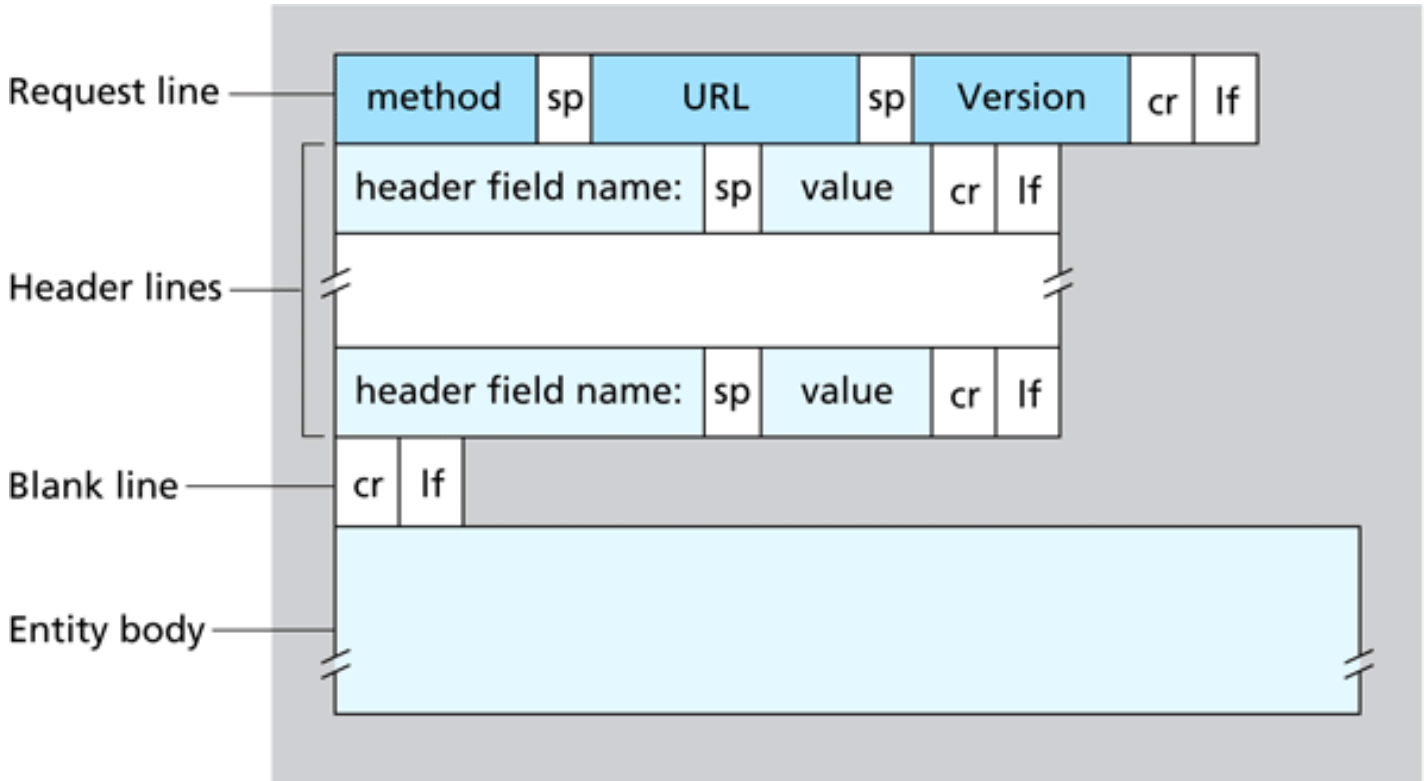
Standardization History

- HTTP/1.1 standardization took 4 years
- after two years RFC 2068 captured the early state of the HTTP/1.1 specification *process*
- browsers started implementing the "HTTP/1.1 standard" before it was officially a standard
- HTTP/1.1 needed to be backward-compatible with many browsers or else many sites would not deploy it
- as a result, RFC 2616 contains some idiosyncrasies, never fully adopted as an official standard

Specification Language

- specification language is precise
  - `MUST`: absolutely essential - if you don't implement this feature you are not compliant
  - `SHOULD`: recommendation - you are compliant if you don't implement this feature, but you should implement it if at all possible
  - `MAY`: optional - not considered necessary
  - there are two obvious counterparts: `MUST NOT`, `SHOULD NOT`
- see RFC 2119

HTTP Request Format

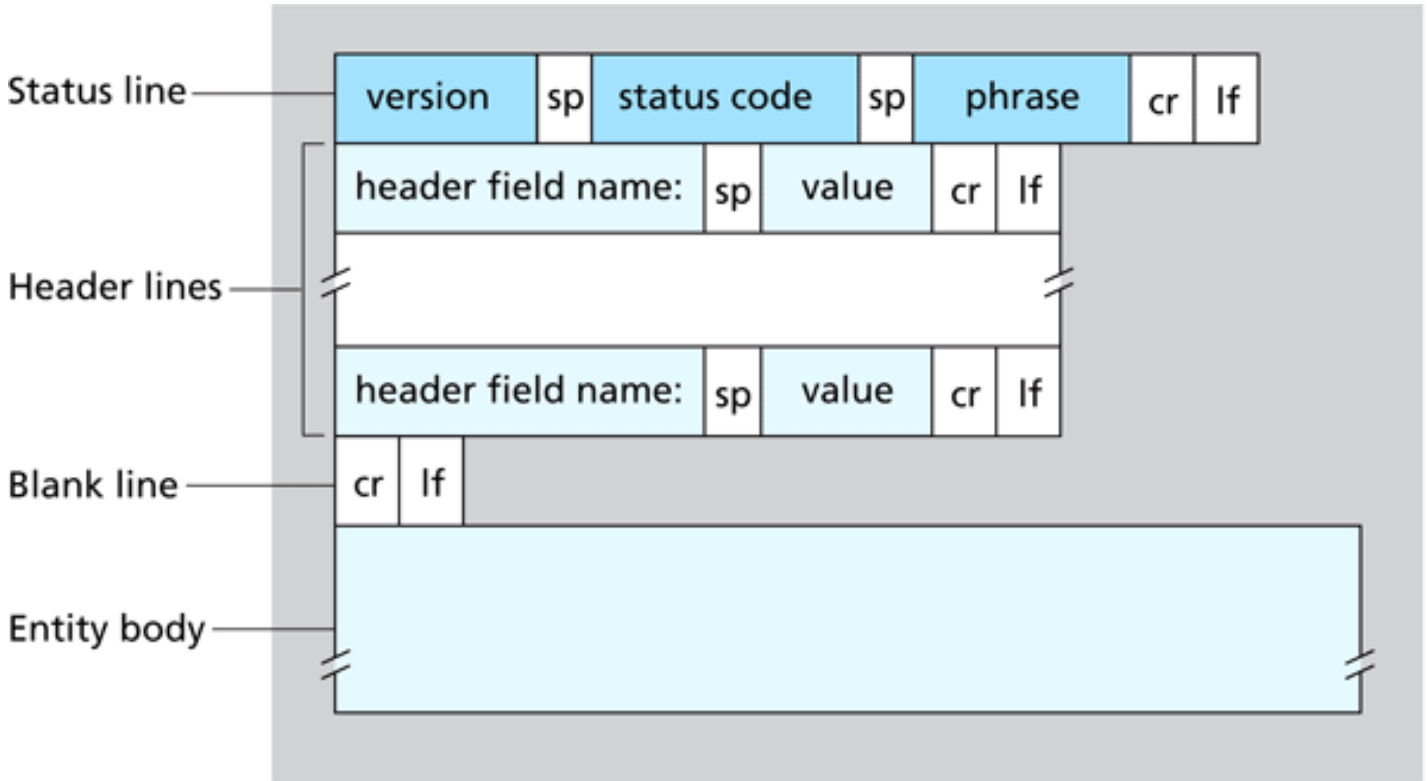


- entity body is optional, with a header that indicates the length of the body in bytes

Example HTTP Request

```
1 GET /index.html HTTP/1.1
2 Host: cs360.byu.edu
3 Accept: text/html,application/xhtml+xml,application/xml
4 Accept-Encoding: gzip,deflate,sdch
5 Accept-Language: en-US,en;q=0.8
6 Connection:keep-alive
7 User-Agent:Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
8 /537.36 (KHTML, like Gecko) Chrome/29.0.1547.76 Safari/537.36
```

HTTP Response Format



- entity body is optional, with a header that indicates the length of the body in bytes

Example HTTP Response

```
1  HTTP/1.1 200 OK
2  Connection:Keep-Alive
3  Date:Mon, 07 Oct 2013 16:31:00 GMT
4  ETag:"c01e0-c4f-4e6c0444f4e40"
5  Keep-Alive:timeout=5, max=100
6  Server:Apache/2.2.22 (Ubuntu)
7  Vary:Accept-Encoding
8
9  <!DOCTYPE html>
10 <html lang="en">
11   <head>
12     <title>CS 360 Internet Programming</title>
13 ...
```

- use telnet cs360.byu.edu 80 to experiment

Request Methods

GET, HEAD

- GET
  - see RFC 1945, section 8.1
  - MAY encode arguments to a script
  - `GET /book.cgi?lastname=Stevens`
- HEAD
  - see RFC 1945, section 8.2
  - MUST NOT return entity body

POST

- see RFC 1945, section 8.3
- MUST have a `Content-Length` to determine length of entity body
- actions taken depend on the URI

PUT, DELETE, LINK, UNLINK

- see RFC 1945, Appendix D
- PUT and DELETE considered dangerous because the client is in charge
- not usually supported by web servers
- when coupled with authentication, can create an API for third-party clients to interact with web services

Headers

Header Syntax

```
1   HTTP-header = field-name ":" [ field-value ] CRLF
```

- arbitrary length
- order is not significant, but recommended that you send General, Request, Response, and Entity Headers in that order
- see RFC 1945, section 4.2

General Headers

- `Date`
  - RFC 1945, Section 10.6
  - date and time at which message originated
  - specifies format of date and time
  - MUST be in GMT

Request Headers

- `If-Modified-Since`
  - RFC 1945, Section 10.9
  - defines a conditional GET - retrieve resource only if it hasn't been changed since the given date
  - uses the same format as the `Date` header
- `From`
  - RFC 1945, Section 10.8
  - used to give user's email address
  - server should parse and log
- `User-Agent`
  - RFC 1945, Section 10.15
  - information about user's browser
  - server should parse and log

Response Headers

- `Server`
  - RFC 1945, Section 10.14
  - identifies server software
  - many servers include this in all responses

Entity Headers

- `Content-Length`
  - RFC 1945, Section 10.4
  - size of the entity body in bytes
  - MUST be included for any message containing an entity body
  - server should send an error if it is missing, see Section 7.2.2
- `Last-Modified`
  - RFC 1945, Section 10.10
  - date and time at which resource was last modified
  - MUST NOT send a time later than message origination
- `Content-Type`
  - RFC 1945, Section 10.5
  - indicates media type of entity body
  - server SHOULD include this header if the message has an entity body

Response Codes

Response Classes

- HTTP groups response codes together into classes
- 100: informational class
- 200: success class
- 300: redirection class
- 400: client error class
- 500: server error class

Notable Error Codes

- 200 OK
- 304 Not Modified
- 400 Bad Request
- 403 Forbidden
- 404 Not Found
- 501 Not Implemented

Features

Multiple Web Sites

- HTTP/1.1 allows many web sites per server
- server MUST include `Host` header in HTTP/1.1

```
1 GET /index.html HTTP/1.1
2 Host: cs360.byu.edu
```

Content Negotiation

- client can ask for different versions
- server-driven
  - client sends hints about user's preference using `Accept-Language`, `Accept-Charset`, `Accept-Encoding`
  - server chooses the best match
- agent-driven
  - server responds with `300 Multiple Choices` that includes list of available representations
  - client makes a new request with the chosen variant
- server-driven is widely used, reduces latency

Persistent Connections and Pipelining

- TCP works best for long-lived connections
  - starts with a slow rate
  - increases rate as segments are delivered successfully
  - decreases rate when loss occurs
- HTTP allows browser to use a single connection for many requests, to keep the rate high
  - useful for fetching images, CSS, Javascript
  - can also use to fetch subsequent pages and their objects
- Connection header
  - by default, connections stay open
  - server sends `Connection:close` to notify client it will close the connection
  - connections may be closed at any time
- many clients use parallel connections to further increase speed
- Chrome uses prefetching to guess which page you will view next

Downloading a Part of a File

- user may want only part of a resource
  - continuing after an aborted transfer
  - fetching in parallel from multiple servers
- `Range` header
  - specify one or more ranges of contiguous bytes
  - `Range: bytes=0-1000`
  - if server's response contains a range, it uses `206 Partial Content`

Compression

- 1997 study showed that compression could save 40% of the bytes sent via HTTP
- `Content-Encoding`: indicates end-to-end content coding
- `Transfer-Encoding`: indicates hop-by-hop transfer-codings (applied by proxies)
- `Accept-Encoding`: indicates the type of encodings a client can accept
- `TE`: indicates the type of transfer-codings a client prefers
- IANA registers transfer-coding token values, including chunked, identity, gzip, compress, and deflate

Chunked Transfers

- chunked transfers are used by a web server for dynamic content, when it doesn't know ahead of time the length of the entire body of the object
- does **not** use Content-Length header
- sends the body as a series of chunks, each having the following format:
  - length of the chunk, in hexadecimal, followed by a CRLF (`\r\n`)
  - a sequence of bytes, whose length is given by the previous line
  - a CRLF
- to end the body, the server sends a valid last chunk with a length of 0 bytes

Cookies

Motivation

- HTTP does not keep track of state, but servers need it
- web sites would like to
  - greet you personally
  - provide an account
  - track your shopping cart
  - provide product recommendations
  - store state about your email account
  - invade your privacy – track what you like to search for, what you like to buy, what you like to read, etc.

How a Cookie Works: RFC 2109

1. user sends usual HTTP request message
2. server creates a unique ID for this user
3. server returns usual HTTP response, but with a cookie header
   - `Set-Cookie: session-id=1678`
4. when user sends subsequent HTTP requests, they also include a cookie header
   - `Cookie: session-id=1678`
5. server can customize its response based on cookie value

Security Implications

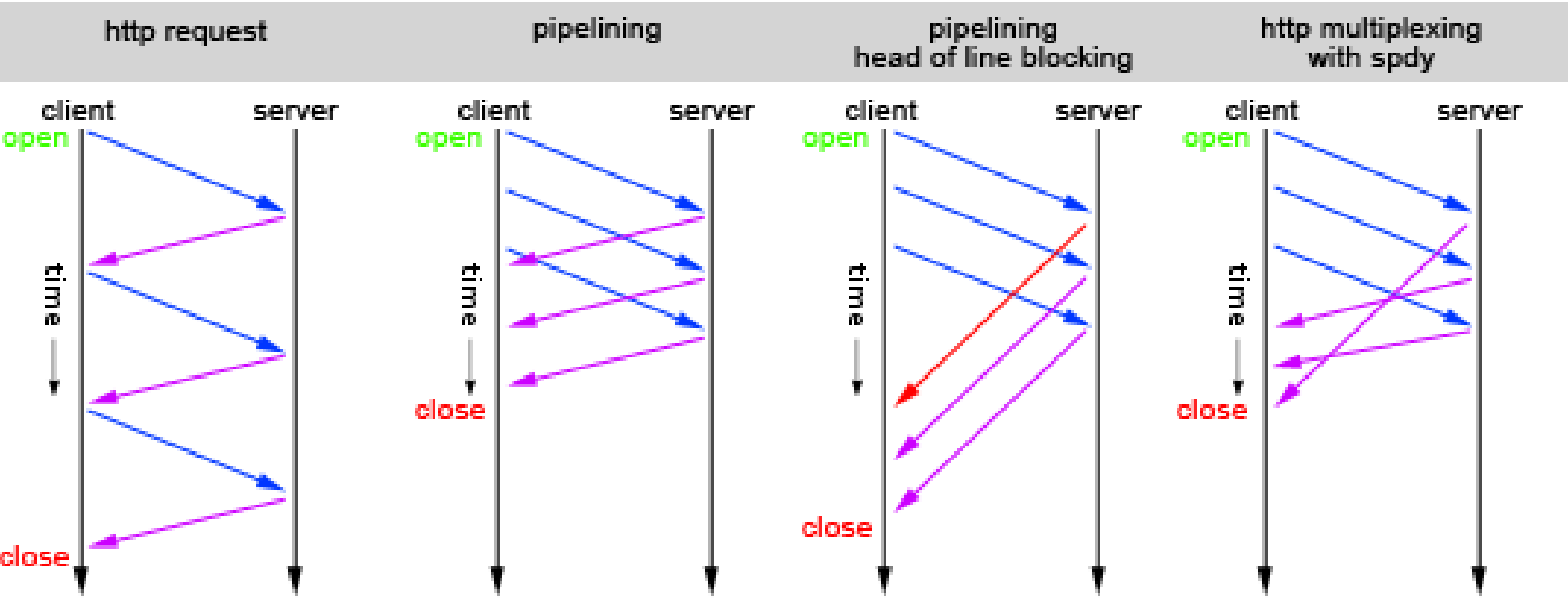
- session hijacking
  1. user supplies name and password
  2. server sets a cookie, which user passes to server for each transaction
  3. intruder snoops on your cookie (*ethereal* on a shared LAN) or guesses its value
  4. intruder supplies cookie to web server and learns your state (e.g. a credit card)
- precautions
  - use SSL to authenticate the user each time
  - send the cookie over the encrypted connection so it can't be snooped
  - use a cookie that expires once the session is finished so it can't be replayed

SPDY, HTTP/2, and QUIC

SPDY

- developed by Google (if in Chrome, will work with any Google web server)
- multiplexing: streams of requests and responses can be interleaved on a single connection (HTTP requires sequential/pipelined requests)
- header compression: avoid repeated headers in multiple requests/responses
- request prioritization: client can mark high priority requests
- server hint: can indidate high priority resources
- server push: can push additional resources (e.g. CSS, JavaScript) in response to a request (e.g. index.html)

SPDY Multiplexing



- for speedup results:
  https://www.chromium.org/spdy/spdy-whitepaper

HTTP/2

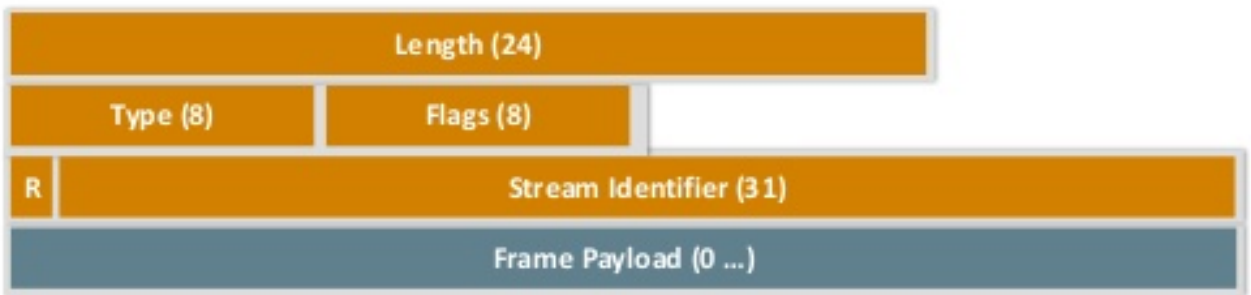
- IETF adopted SPDY improvements
- server push requires a subscription from the client, rather than unilateral action by the server
- binary framing

HTTP/2 Binary Framing

HTTP/2 Binary Framing

Enabled by dumping newline delimited ASCII

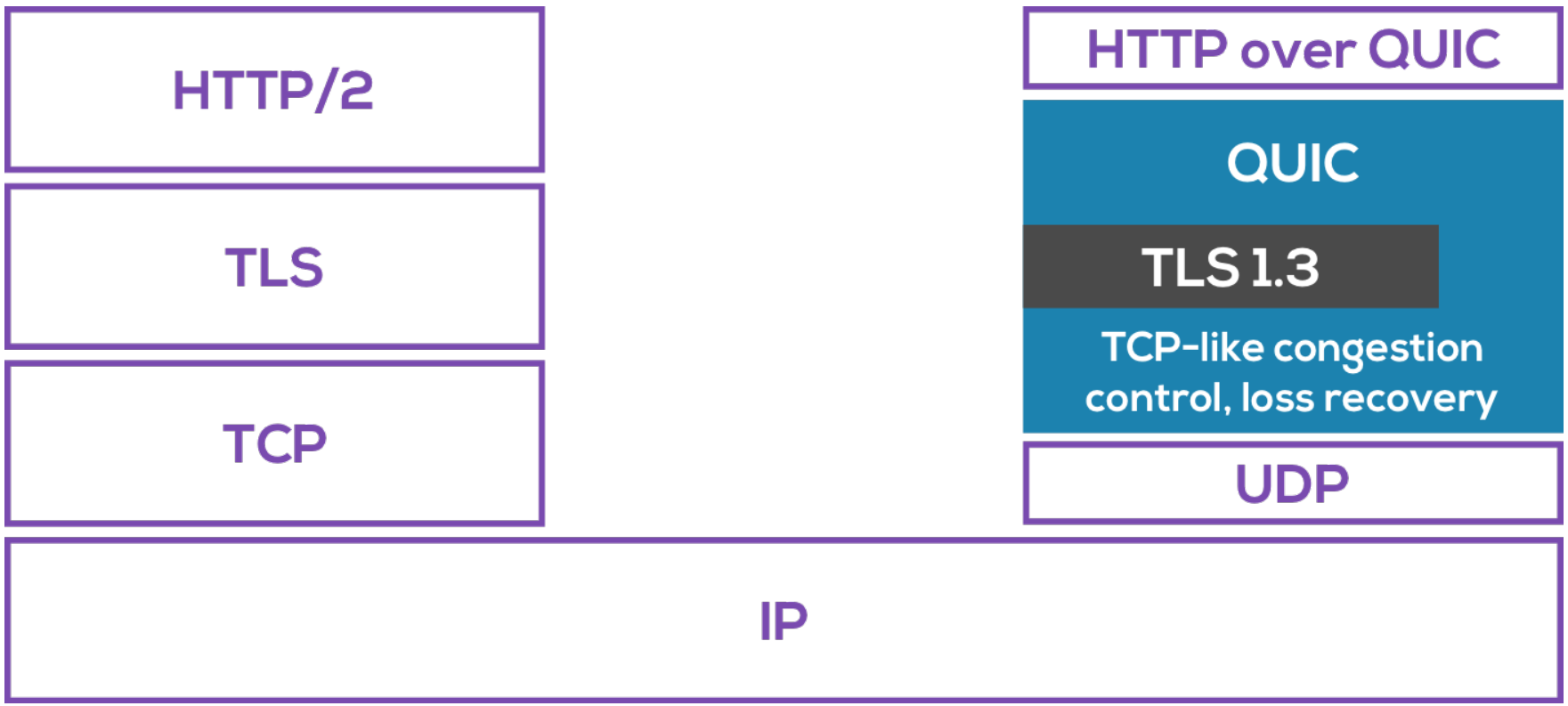
- Frame Header



- Types
  - HEADERS, DATA, PRIORITY, RST_STREAM, SETTINGS, PUSH_PROMISE, PING, GOAWAY, WINDOW_UPDATE, CONTINUATION

Copyright © 2014, Oracle and/or its affiliates. All rights reserved. |

QUIC


HTTP/2
TLS
TCP
HTTP over QUIC
QUIC
TLS 1.3
TCP-like congestion control, loss recovery
UDP
IP

QUIC

- low latency connection establishment
  - cache TLS credentials to resume TLS sessions
- more rapidly change transport layer (implemented in application)
  - loss recovery
  - congestion control
  - FEC
- connection migration