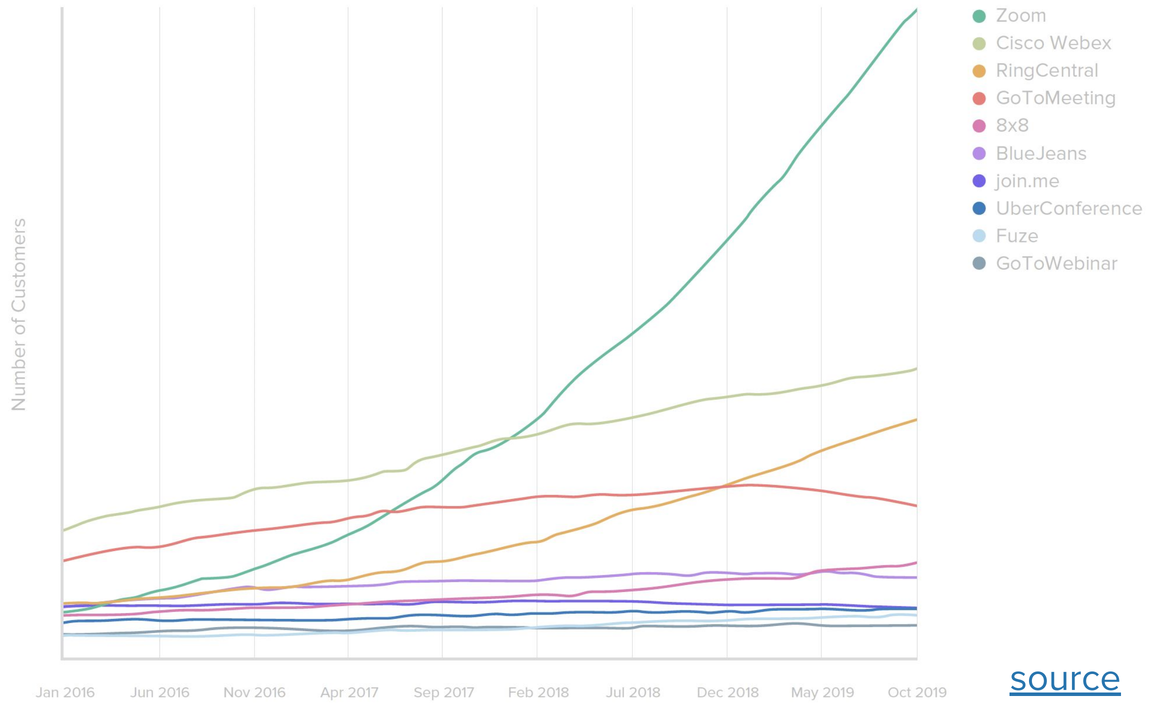
CS188
Scalable Internet Services

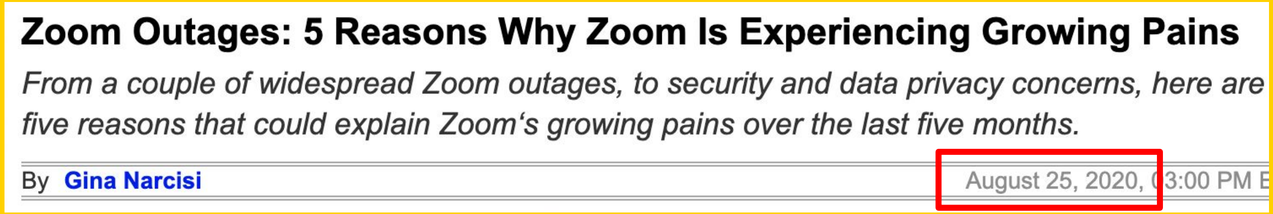
John Rothfels, 10/6/20

Motivation

Most Popular Video Conferencing Apps



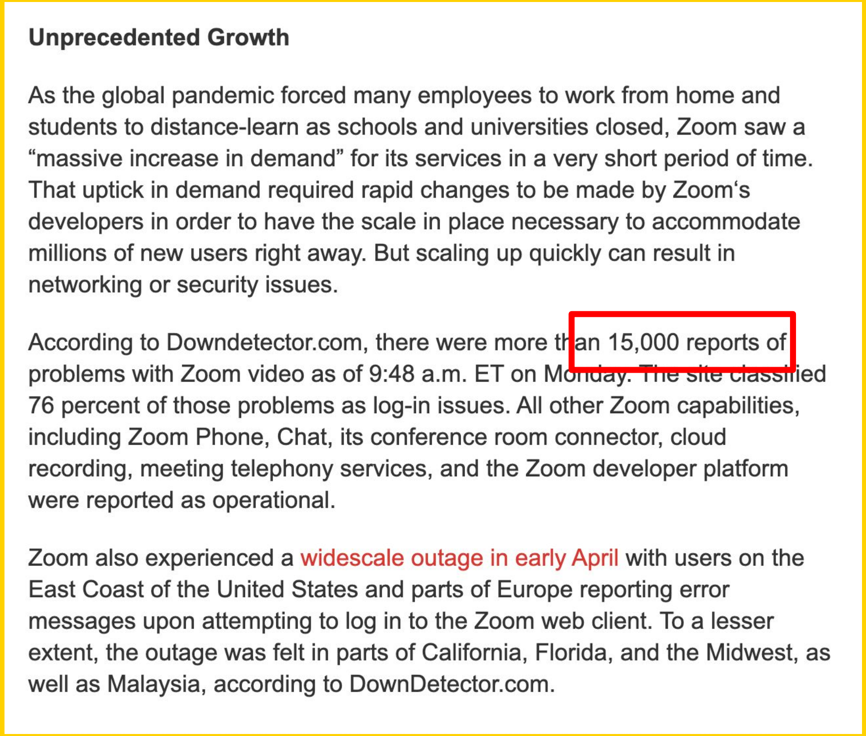
source



Zoom Outages: 5 Reasons Why Zoom Is Experiencing Growing Pains

From a couple of widespread Zoom outages, to security and data privacy concerns, here are five reasons that could explain Zoom's growing pains over the last five months.

By **Gina Narcisi** August 25, 2020, 03:00 PM E



Unprecedented Growth

As the global pandemic forced many employees to work from home and students to distance-learn as schools and universities closed, Zoom saw a "massive increase in demand" for its services in a very short period of time. That uptick in demand required rapid changes to be made by Zoom's developers in order to have the scale in place necessary to accommodate millions of new users right away. But scaling up quickly can result in networking or security issues.

According to Downdetector.com, there were more than 15,000 reports of problems with Zoom video as of 9:48 a.m. ET on Monday. The site classified 76 percent of those problems as log-in issues. All other Zoom capabilities, including Zoom Phone, Chat, its conference room connector, cloud recording, meeting telephony services, and the Zoom developer platform were reported as operational.

Zoom also experienced a widescale outage in early April with users on the East Coast of the United States and parts of Europe reporting error messages upon attempting to log in to the Zoom web client. To a lesser extent, the outage was felt in parts of California, Florida, and the Midwest, as well as Malaysia, according to DownDetector.com.

source

Motivation

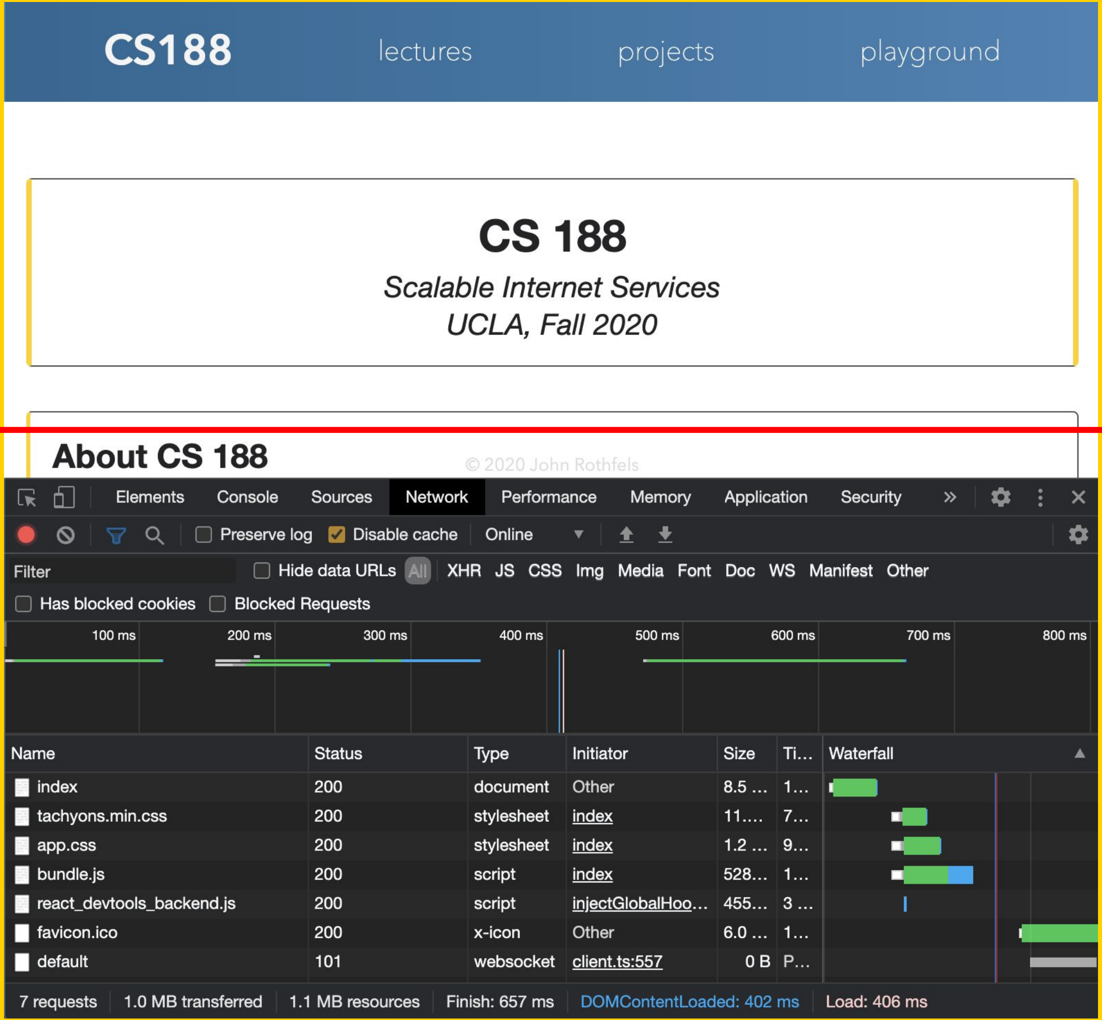
http://highscalability.com/blog/2020/5/14/a-short-on-how-zoom-works.html

- Zoom in extra depth
- Interesting reading, not required 😉

Today

The web's technology stack:

- Browsers / Servers
- HTTP
- HTML / CSS / JavaScript
- Demo: course website under the hood, building a feature



CS188 lectures projects playground

CS 188

Scalable Internet Services
UCLA, Fall 2020

About CS 188

© 2020 John Rothfels

Elements Console Sources Network Performance Memory Application Security
Preserve log
Disable cache
Online
Filter
Hide data URLs
All XHR JS CSS Img Media Font Doc WS Manifest Other
Has blocked cookies
Blocked Requests
100 ms 200 ms 300 ms 400 ms 500 ms 600 ms 700 ms 800 ms

Name	Status	Type	Initiator	Size	Ti…	Waterfall
index	200	document	Other	8.5 …	1…	
tachyons.min.css	200	stylesheet	index	11.…	7…	
app.css	200	stylesheet	index	1.2 …	9…	
bundle.js	200	script	index	528…	1…	
react_devtools_backend.js	200	script	injectGlobalHoo…	455…	3 …	
favicon.ico	200	x-icon	Other	6.0 …	1…	
default	101	websocket	client.ts:557	0 B	P…	

7 requests | 1.0 MB transferred | 1.1 MB resources | Finish: 657 ms | DOMContentLoaded: 402 ms | Load: 406 ms

Browser

What is a **browser**?

Browser

What is a **browser**?

- A process
- Runs on an operating system
- Renders graphics (HTML, CSS)
- Responds to user input
- Uses the network
- Executes JavaScript inside an embedded VM

Server

What is a **(web) server**?

Server

What is a **(web) server**?

- A process
- Runs on an operating system
- Responds to client requests
- Interacts with the network
- Interacts with the filesystem

HTTP

What is **HTTP**?

HTTP

What is **HTTP**?

- Simple ASCII protocol
  - Client opens a TCP socket (standard port: 80)
  - Client sends ASCII text request to server
  - Server sends response to client
  - Client closes the TCP socket

- Originally designed to exchange HTML, extended to send anything.
- Originally designed for web browser to server interaction. Today also very widely used for server to server APIs.

HTTP

In 1990 the internet has existed for 20 years, email for 8 years.

Tim Berners-Lee is at CERN, and believes that two already-existing technologies could be combined to good effect: the **internet** and **HyperText**.

HyperText is the idea of linking documents together with **HyperLinks**.

```
    <a href=”http://mycoolsite.com/page2” />
```

Berners-Lee creates the first versions of **HTTP** and **HTML**.

HTTP

In 1993, Mosaic is created at the NCSA center at UIUC

In 1994, Marc Andreessen leaves UIUC and founds Netscape with Jim Clark.

In 1998, AOL acquires Netscape.

HTTP

In 1993, Mosaic is created at the NCSA center at UIUC.

In 1994, Marc Andreessen leaves UIUC and founds Netscape with Jim Clark.

In 1998, AOL acquires Netscape for $4.2B.

Moral of the story?

HTTP

In 1993, Mosaic is created at the NCSA center at UIUC.

In 1994, Marc Andreessen leaves UIUC and founds Netscape with Jim Clark.

In 1998, AOL acquires Netscape for $4.2B.

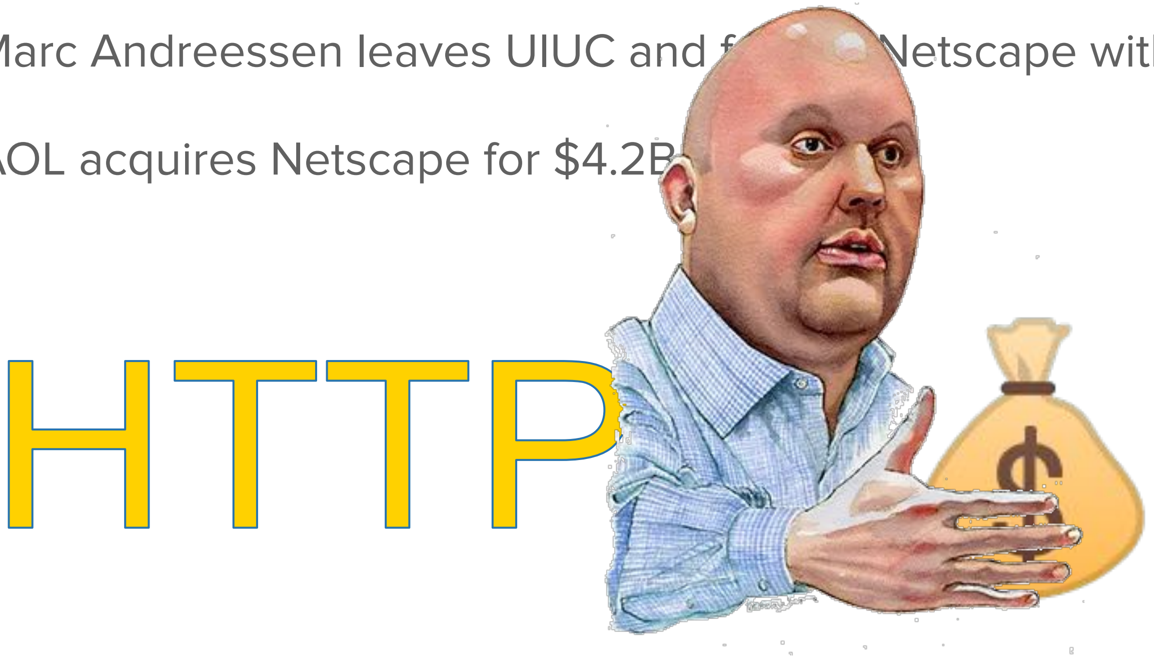
HTTP = 💰

HTTP

In 1993, Mosaic is created at the NCSA center at UIUC.

In 1994, Marc Andreessen leaves UIUC and f Netscape with Jim Clark.

In 1998, AOL acquires Netscape for $4.2B



HTTP request

What’s in an **HTTP request**?

HTTP request

What’s in an **HTTP request**?

- **verb**: what do you want to do? **GET** something? **POST** something?
- **resource**: what is the path of the resource you want to act on?
- **version**: specified to aid in compatibility
- **headers**: standard ways to request optional behavior or convey metadata
- **body**: a data payload (e.g. JSON, text) possibly empty

HTTP response

What’s in an **HTTP response**?

HTTP response

What’s in an **HTTP response**?

- ~~**verb**: what do you want to do? **GET** something? **POST** something?~~
- ~~**resource**: what is the path of the resource you want to act on?~~
- **version**: specified to aid in compatibility
- **headers**: standard ways to request optional behavior or convey metadata
- **body**: a data payload (e.g. JSON, text) possibly empty
- 🆕 **status**: success? failure? Other?
  - Also a header

HTTP request / response

Request:

```
GET /about/ HTTP/1.1
Accept: text/html
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html class="example" lang="en">
```

Verb
Resource
Version
Headers
Status
Body

(some) HTTP verbs

- **`GET`**
  - fetches a copy of a resource
  - assumed to have no side-effects *and* idempotent
  - e.g. **`GET /users/:id`**, *not* **`GET /users/sign_out`**
- **`POST`**
  - sends data to a server, used for creating a new resource, commonly used for form submission
  - assumed to have side-effects but *not* idempotent (“do you want to post your form again?”)
  - e.g. **`POST /users/create`**
- **`DELETE`**
  - destroys a resource, assumed to have side-effects *and* idempotent
  - e.g. **`DELETE /session/:id`**
- **`PUT`**
  - send data to a server, commonly used to modify an existing resource
  - assumed to have side-effects *and* idempotent

HTTP verbs

⚠️ The usage of HTTP verbs is governed by convention.

For example, **`GET`** *should* have no side-effects… but it’s up to you to follow the convention when implementing your server.

ℹ️ Different protocols may use different conventions.

For example, GraphQL requests are conventionally all **`POST`** requests (even the ones that just fetch data).

(other) HTTP verbs

- `HEAD`
  - fetches just the headers for a resource
  - ⁉️ why would this be useful?
- `OPTIONS`
  - requests the “options” available for making a request from a different origin

Documentation:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

⁉️ What problems could happen through the misuse of HTTP verbs?

HTTP version

The **HTTP version** is included with each request/response for ease of protocol development.

Everyone today uses **HTTP 1.1**, with growing adoption for **HTTP 2.0**.

- 1991 · **HTTP 0.9**: Single line protocol. No headers.
- 1996 · **HTTP 1.0**: Headers added, more than just HyperText!
- 1999 · **HTTP 1.1**: Connection keep-alive, more caching mechanisms, everything else we know and love today.
- 2015 · **HTTP 2.0**: *<future lecture>*

HTTP headers

HTTP headers are plentiful, there are many more than we'll cover here.

- **`Accept`**: indicates the preferred format for a resource
  - e.g. **`accept: text/html,text/json`**
- **`Accept-Encoding`**: indicates the client accepts compressed data
  - e.g. **`accept-encoding: bzip2,gzip`**
- **`Host`**: indicates the DNS host the requestor is trying to reach
  - ⁉️ Why isn't this important? Why isn't the host implied?
- **`Accept-Language`**: indicates the client's preferred language, in order
  - ⁉️ Why is this a header instead of added to request body/payload?
- **`User-Agent`**: indicates what type of browser or device is connecting
  - Lmao, good luck deciphering.
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

HTTP headers

HTTP headers are plentiful, there are many more than we'll cover here.

Some specifically useful for scaling / performance which we'll cover later:

- **`ETag`**, **`Date`**, **`Last-Modified`**, **`Cache-Control`**, **`Age`**

Headers that begin with **`X-`** are not part of the specification, but may be commonly used and even "standardized".

You may include your own **`X-`** headers in any request.

HTTP status code

The **HTTP status code** indicates the outcome of the request. The first digit of the code classifies it:

- **`1xx`**: informational
- **`2xx`**: successful
- **`3xx`**: redirecting
- **`4xx`**: client error
- **`5xx`**: server error

⁉️ If I make a request to a non-existent resource, which category of error is it?

HTTP status code

The **HTTP status code** is commonly:

- **`200 OK`**: the normal success case
- **`301 Moved Permanently`**: the client should use the new provided URL moving forward. ⁉️ Where would the new URL be? ⚠️ Use with caution.
- **`302 Found`**: you should go to a different URL to get this resource, but it hasn't moved there permanently
- **`403 Forbidden`**: the request is not authorized
- **`404 Not Found`**: the resource could not be found
- **`500 Internal Server Error`**: something crashed or error thrown
- **`503 Service Unavailable`**: temporary failure, e.g. deployment downtime

HTTP resource

The **resource** specifies the thing you are referring to via a logical hierarchy.

Examples:

- Good: http://www.amazon.com/**gp/product/1565925092/**
- Bad: http://www.cocacola.com/**index.jsp**?page_id=4251

Anything past the question mark is called a **query string**.

- Intended to assist in locating the resource
- Parameters are assigned using equals, concatenated using ampersand
- E.g. http://www.amazon.com/search**?term=dogs&new=1**

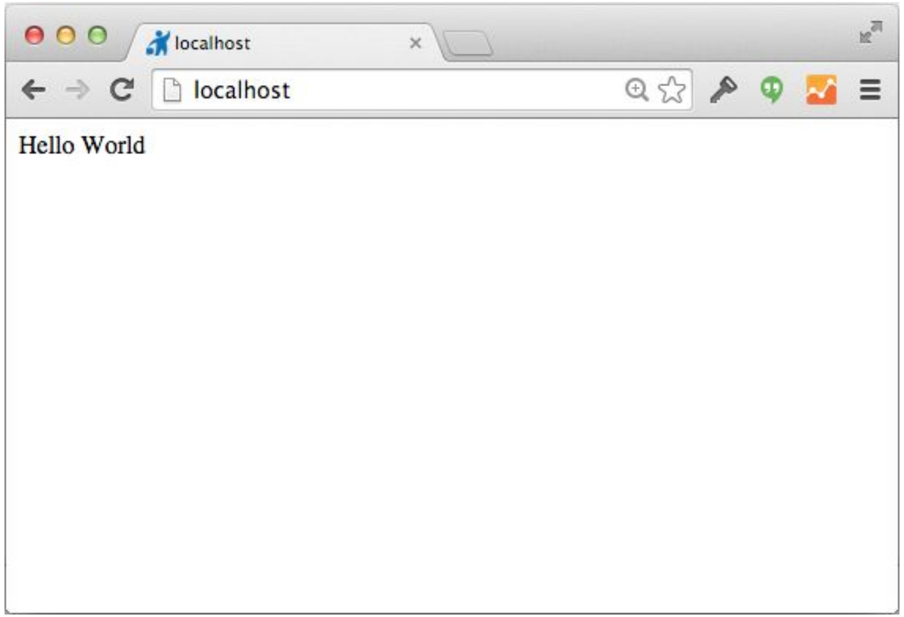
HTTP protocol: demo

```
sudo nc -l 80 # runs a TCP listener on port 80
```

```
GET / HTTP/1.1
Host: localhost
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/37.0.2062.124 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
```

HTTP protocol: demo

```
echo "HTTP/1.1 200
OK\nContent-Type:
text/html\n\n<html><body>Hello
World</body></html>" | sudo nc -l
80
```



Hello World

HTTP protocol: demo

```
echo "GET /about/ HTTP/1.1\nAccept:
text/html \n\n" | nc www.google.com
80
```

```
HTTP/1.1 301 Moved Permanently
Location: https://about.google/
Content-Type: text/html;
charset=UTF-8
X-Content-Type-Options: nosniff
Date: Mon, 05 Oct 2020 22:50:08 GMT
Expires: Mon, 05 Oct 2020 23:20:08
GMT
Cache-Control: public, max-age=1800
Server: sffe
Content-Length: 218
X-XSS-Protection: 0

<HTML><HEAD><meta
http-equiv="content-type"
content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD>
...
```

HTTP: beyond one connection

Thinking of HTTP as one request per TCP connection can be a useful simplification for reasoning about your application.

⁉️ In practice, this is inefficient. Why?

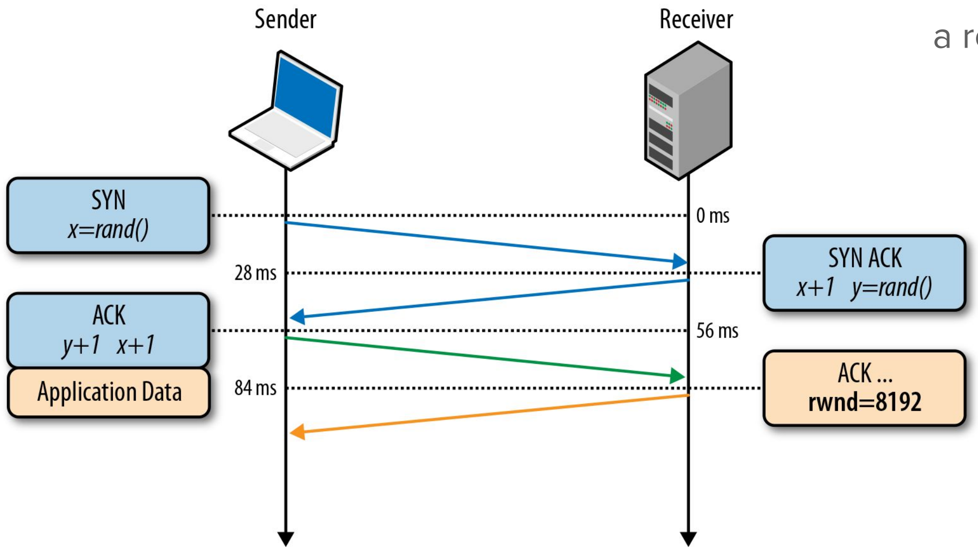
HTTP: beyond one connection

Thinking of HTTP as one request per TCP connection can be a useful simplification for reasoning about your application.

⁉️ In practice, this is inefficient. Why?

- Initial round trips to setup connection
- TCP connections start out low bandwidth

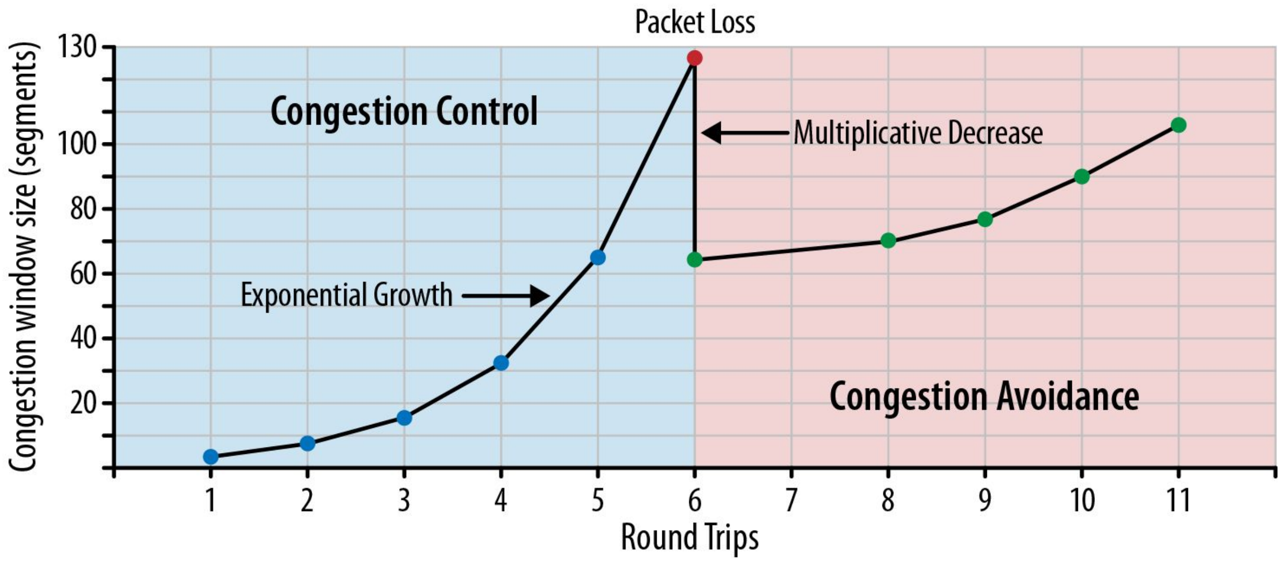
HTTP: beyond one connection



Each TCP connection that is established requires a round trip for setup.

HTTP: beyond one connection

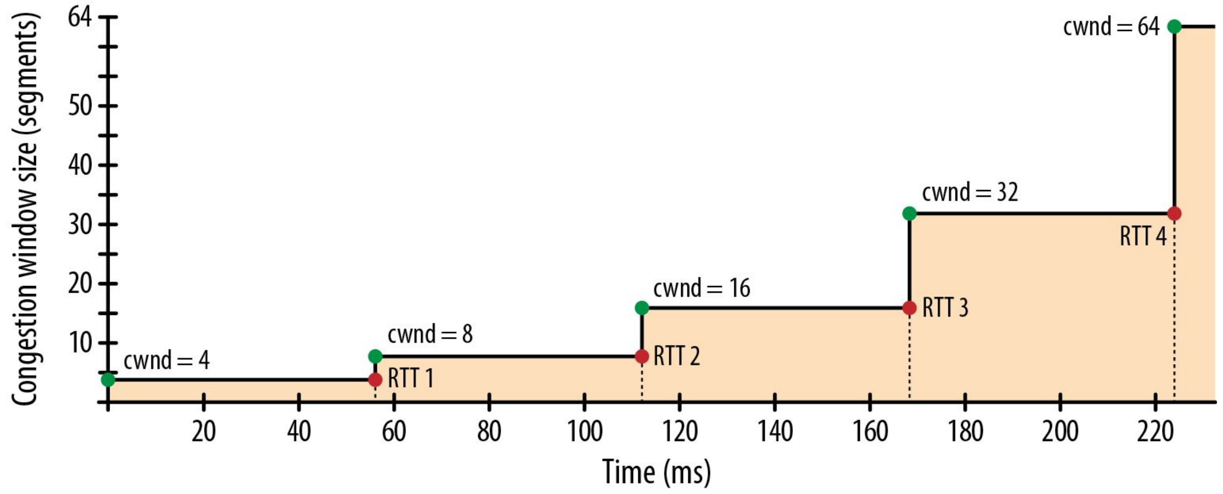
The early phases of a TCP connection are bandwidth constrained.



HTTP: beyond one connection

The early phases of a TCP connection are bandwidth constrained.

By repeatedly reopening TCP connections, we only use the most constrained part.



HTTP: beyond one connection

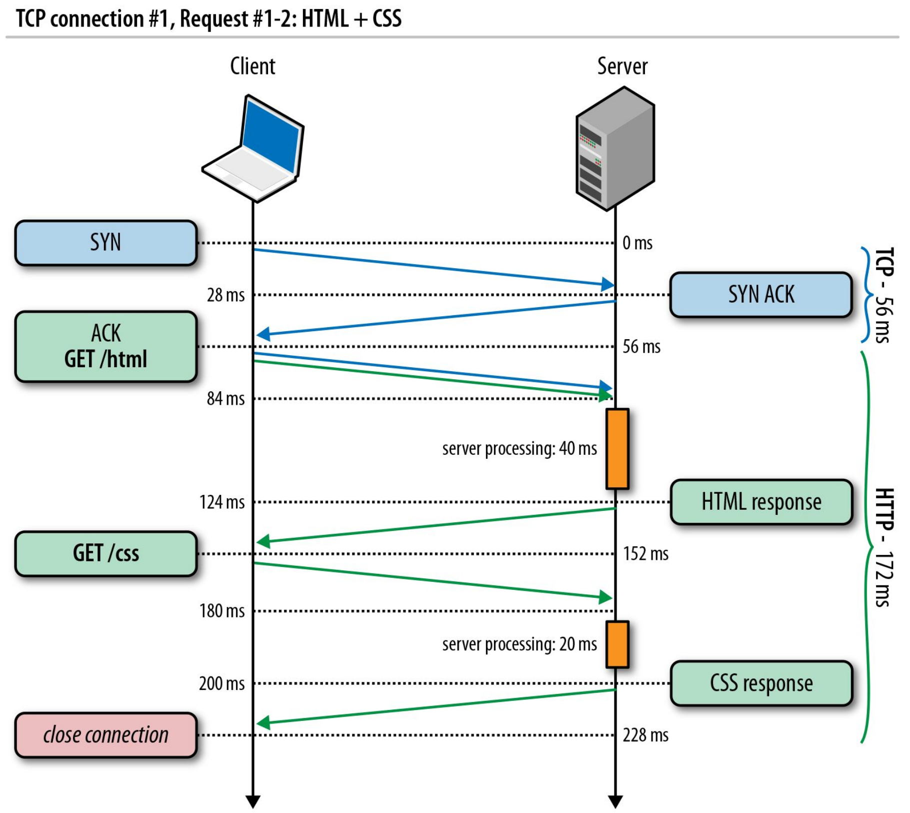
HTTP 1.1 introduced **connection keep-alive** to address this.

It is the default behavior as of 1.1, but you can also set it manually via header:

```
Connection: keep-alive
```

- Tells the server not to close the socket. Server will expect multiple requests to arrive on the same socket.
- After receiving a response, client can send a new request.

HTTP: beyond one connection



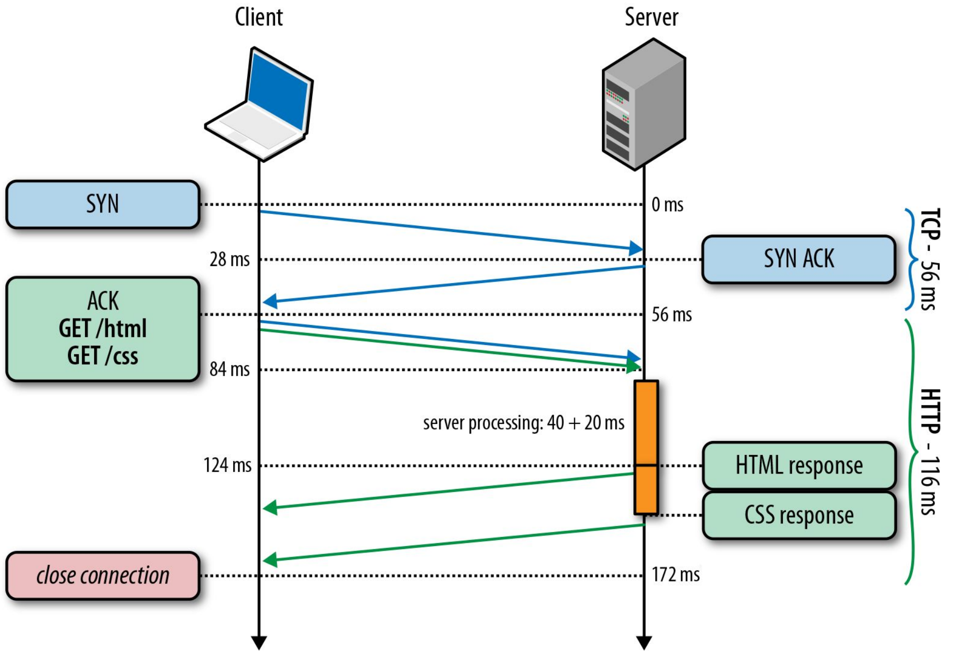
Multiple requests sent over a single TCP socket, using **`connection: keep-alive`**.

HTTP: beyond one connection

Taking it a step further: **HTTP pipelining**

- Frequently we need many resources from the same server
- Why do we wait for responses before sending subsequent requests?
- Can’t we save time on round trips if we send our requests without waiting for each response?

HTTP: beyond one connection



Multiple requests sent over a single TCP message using **HTTP pipelining**.

HTTP: beyond one connection

Taking it a step further: **HTTP pipelining**

- It “works” but it has a lot of problems
  - Head of line blocking
  - Failure mechanics
- Due to problems, while support for it exists widely, it is generally not turned on
- Expect to see this make a comeback in **HTTP 2.0**

HTTP: beyond one connection

A browser that needs N resources can establish anywhere between 1 and N TCP connections to get these.

⁉️ How many should it use?

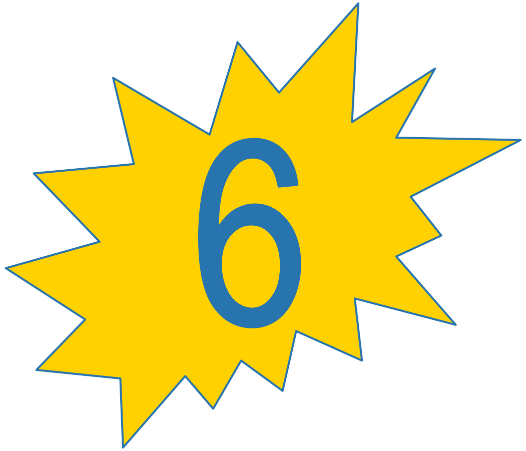
- Too many and you can DoS the site
- Too few and you prohibit any parallelism

HTTP: beyond one connection

A browser that needs N resources can establish anywhere between 1 and N TCP connections to get these.

⁉️ How many should it use?

- Too many and you can DoS the site
- Too few and you prohibit any parallelism



HTTP: beyond one connection

A browser that needs N resources can establish anywhere between 1 and N TCP connections to get these.

⁉️ How many should it use?

- Six connections per server is today's "standard" (not part of an actual standard)
- If you want more, use **domain sharding**
  - Set up www1.foo.com, www2.foo.com, ... to all point to same server
  - Browser treats each as a separate server, opens six per subdomain

HTML

Hyper**T**ext **M**arkup **L**anguage takes flat text and annotates it to add structure.

Areas of text are marked up using **tags**, e.g. **`<div>hello world</div>`**

Tags are nestable.

Tags have key-value **attributes**, e.g. `<div hidden="true">goodbye</div>`

ℹ️ Markup languages aren’t programming “languages”, they’re structured data.

HTML

HTML documents consist of an all-encompassing **`<html>`** tag. The logical tree of tags found inside this is referred to as the **D**ocument **O**bject **M**odel (**DOM**).

Within this, the document is divided into the **`<head>`** and the **`<body>`**:

```
<html>
<head>
   <title>My HTML Page</title>
</head>
<body>Hello World</body>
</html>
```

HTML

The HTML **`<head>`** generally contains information about the page that isn't directly rendered:

- CSS
- JavaScript
- Metadata (`title`, `refresh`)

HTML

The **`<body>`** includes the content that is seen by the user of the browser. Tags can be used for formatting and styling, but this is increasingly the role of CSS.

- **`h1`**: Header, 1st level
- **`p`**: Paragraph
- **`ul`**, **`ol`**, **`li`**: Unordered and ordered lists
- **`table`**, **`tr`**, **`td`**: Tabular information
- **`span`**: An inline grouping mechanism
- **`div`**: A block-level grouping mechanism

HTML

The **`<body>`** also includes **anchor tags**. The **`<a>`** tag is one of the original, central innovations of the world wide web.

```
You can search <a href=”http://www.google.com”>here</a>
```

The links that are created by anchor tags make HyperText.

HTML

The **`<body>`** also includes **forms**. The **`<form>`** tag wraps user inputs. The inputs collect data, and the form submits that data to the specified action (resource).

```
<form action="/communities" method="post">
  <label for="cn">Name</label><br>
  <input type="text" name="community" id="cn">
  <input type="submit" name="commit" value="Submit">
</form>
```

ℹ️ Method doesn’t have to be **`POST`**

ℹ️ Encoding defaults to **`application/x-www-form-urlencoded`**

HTML

HTML elements have some important **attributes**:

- **`id`**: A unique identifier can be assigned to a single DOM element
- **`class`** - Multiple classes can be assigned to DOM elements. There is a many-to-many relationship between classes and DOM elements.

```
<span class=”alert,loud” id=”flash_message”>Error.</span>
```

ℹ️ CSS and JavaScript can refer to DOM elements by their **`id`** or **`class`**.

CSS

Inside the HTML `<head>` (or `<body>`), we can define CSS using the `<style>` tag.

```
<html>
<head>
<style>
    h1 {
        color: blue;
    }
</style>
</head>
<body>
    <h1> Hello World </h1>
</body>
</html>
```

CSS

We can refer to elements in multiple ways.

```
<html>
<head>
<style>
    #header {
        color: blue;
    }
</style>
</head>
<body>
    <h1 id=”header”> Hello World </h1>
</body>
</html>
```

CSS

We can refer to elements in multiple ways.

```
<html>
<head>
<style>
    .header { font-size: 32px; }
    .blue { color: blue; }
</style>
</head>
<body>
    <h1 class=”header blue”> Hello World </h1>
    <span class=”blue”> I’m feeling blue today </span>
</body>
</html>
```

CSS

We can apply styling inline.

```
<html>
<head>
</head>
<body>
    <h1 style=”color:blue;”> Hello World </h1>
</body>
</html>
```

CSS

We can apply styling by loading a CSS file.

```
<html>
<head>
    <link rel="stylesheet" href="/assets/app.css" >
</head>
<body>
    <h1> Hello World </h1>
</body>
</html>
```

CSS

CSS is hard. There’s lots of rules about precedence & cascading. This is not a class about CSS. 🙏

Scaling CSS & design systems is an interesting topic that we may have a guest lecture about.

For this class, you need to know:

- How to refer to an element by its CSS selector (`#id`, `.class`)
- How to add style to an element (see previous slides)
- How to use Google: https://developer.mozilla.org/en-US/docs/Web/CSS
- You’re encouraged to use CSS libraries (e.g. Bootstrap) instead of writing your own. 🎉

For next time

Assigned reading for next lecture [on course website].

Find your project group! You will need a group by the start of this week’s lab. Use Piazza to look for a project group.