Administrivia

• TCP is due next Tuesday

Will announce some final project info, grading feedback soon
Warmup

Browser wants to fetch: http://example.com/page.html

Assuming no caching, what is the minimum number of packets the browser needs to wait for?
It gets worse

Modern web traffic almost always uses HTTPS: https://example.com/page.html
=> Creates a secure transport layer to prevent eavesdropping, etc
(more on this later)
How does a browser load a page?

- Click a link, type in URL => browser fetches main page
How does a browser load a page?

- Click a link, type in URL => browser fetches main page
- Main page has links to more resources => need to fetch these too!
  - Images, CSS, Javascript, etc.
How does a browser load a page?

• Click a link, type in URL => browser fetches main page
• Main page has links to more resources => need to fetch these too!
  – Images, CSS, Javascript, etc.

• New resources might load yet more resources…

Recursive process with many dependencies!
Early websites: not many dependencies, usually served by one server
Now???
On a modern webpage…
On a modern webpage...

• Huge number of dependencies, external resources
  – ... from many different locations, not just one server!

• Lots of asynchronous operations => loading new resources as you are using the page

• Lots of dynamic content => generated by the server specifically for you (your feed, ad data, ...)

How to make this fast?
How to make this fast?

What’s important for performance?
Observation: lots of small requests

Latency is a problem! Need many RTTs just to fetch one resource!
Observation: lots of small requests

Latency is a problem! Need many RTTs just to fetch one resource!

HTTP/1.0: One TCP connection per request!
Can we do better?

HTTP/1.1: Persistent connections

=> Reuse TCP connection to for multiple requests
Can we do better?

HTTP/1.1: Persistent connections

=> Reuse TCP connection to for multiple requests
Can we do better?

HTTP/1.1 (1996): Persistent connections

- Reuse TCP connection for multiple requests

Problems?
- One big request blocks others \( \Rightarrow \) head of line blocking
- Same if connection has packet loss
- Doesn’t help when fetching from multiple locations
What can be done?
HTTP/1.1 Request

GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh ... 
Accept: text/xml,application/xml ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7;*,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
What can be done?

Pipelining: have multiple “in-flight” requests at once

Two methods
• Multiple TCP connections in parallel

• Change the HTTP protocol: multiple requests per connection
What can be done?

Pipelining: have multiple “in-flight” requests at once

Two methods

• Multiple TCP connections in parallel
  => Browsers often do this (up to a limit)

• Change the HTTP protocol: multiple requests per connection
  => Newer HTTP versions: HTTP/2, HTTP/3
HTTP/2 (2015)

Adds support for **multiplexed streams on one connection**

What happens if a packet gets dropped?

https://www.twilio.com/blog/2017/10/http2-issues.html
HTTP/2 (2015)

Adds support for **multiplexed streams on one connection**

TCP provides a single, ordered byte stream

=> doesn’t know about multiple connections!

https://www.twilio.com/blog/2017/10/http2-issues.html
HTTP/2 (2015)

Adds support for multiplexed streams on one connection

TCP provides a single, ordered byte stream
=> doesn’t know about multiple connections!

Encumbered by TCP’s semantics:
If a packet is lost, all streams suffer! 😭 😭 😭

https://www.twilio.com/blog/2017/10/http2-issues.html
HTTP/2 (2015)

Adds support for **multiplexed streams on one connection**

TCP provides a single, ordered byte stream

=> doesn’t know about multiple connections!

Encumbered by TCP’s semantics:
If a packet is lost, all streams suffer! 😢😢😢

=> Head of line blocking

https://www.twilio.com/blog/2017/10/http2-issues.html
HTTP/3 (2022): HTTP + QUIC

QUIC (RFC9000): Newer transport-layer protocol, same goals as TCP
  – Supports multiple streams at once
  – Various tricks to reduce message size and latency
  – Integrates security by default (TLS)

• By moving multiplexing into the transport layer, can do so in a way that benefits HTTP (no head of line blocking!)
Comparison: QUIC’s handshake
What else can we do?
Caching

Place caches throughout network
- Use locality: closer to clients => lower latency
- Improve throughput by avoiding bottleneck links
How to Control Caching?

• Server sets options
  – *Expires* header
  – No-Cache header

• Client can do a conditional request:
  – Header option: if-modified-since
  – Server can reply with 304 NOT MODIFIED
Where to cache content?
Where to cache content?

- Client (browser): avoid extra network transfers
- Server: reduce load on the server
- Service Provider: reduce external traffic
How well does caching work?

• Very well, up to a point
  – Large overlap in requested objects
  – Objects with one access place upper bound on hit ratio
  – Dynamic objects not cacheable*

• Example: Wikipedia
  – About 400 servers, 100 are HTTP Caches (Squid)
  – 85% Hit ratio for text, 98% for media

* But can cache portions and run special code on edges to reconstruct
Reverse Proxies

=> Cache close to server
Reverse Proxies

- Also called Accelerators
- Can distribute load within datacenter

=> Cache close to server
Forward Proxies

- Server
- Backbone ISP
- Reverse proxies
- ISP-1
- ISP-2
- Clients

Forward proxies
Forward Proxies

Typically done by ISPs or Enterprises

- Reduce network traffic and decrease latency
- May be transparent or configured
Content Distribution Networks (CDNs)

Companies that specialize in providing caching services (among other things)

=> Akamai, Cloudflare, …
Content Distribution Networks (CDNs)

Companies that specialize in providing caching services (among other things)
=> Akamai, Cloudflare, ...

• Provide both forward and reverse caching

• Can also do some processing
Content Distribution Networks (CDNs)

Companies that specialize in providing caching services (among other things)
=> Akamai, Cloudflare, …

• Provide both forward and reverse caching
  – Pull: result from client requests
  – Push: expectation of high access rates to some objects
• Can also do some processing
  – Deploy code to handle some dynamic requests
  – Can do other things, such as transcoding
An Example CDN
How Akamai works

Akamai has cache servers deployed close to clients
  – Co-located with many ISPs

• Challenge: make same domain name resolve to a proxy close to the client
• Lots of DNS tricks. BestBuy is a customer
  – Delegate name resolution to Akamai (via a CNAME)
Other CDNs

- Akamai, Limelight, Cloudflare
- Amazon, Facebook, Google, Microsoft
- Netflix
- Where to place content?
- Which content to place? Pre-fetch or cache?
DNS Resolution

```
dig www.bestbuy.com
;; ANSWER SECTION:
www.bestbuy.com.edgesuite.net. 21600 IN CNAME a1105.b.akamai.net.
a1105.b.akamai.net. 20 IN A 198.7.236.235
a1105.b.akamai.net. 20 IN A 198.7.236.240
;; AUTHORITY SECTION:
b.akamai.net. 1101 IN NS n1b.akamai.net.
b.akamai.net. 1101 IN NS n0b.akamai.net.
;; ADDITIONAL SECTION:
n0b.akamai.net. 1267 IN A 24.143.194.45
n1b.akamai.net. 2196 IN A 198.7.236.236
```

- `n1b.akamai.net` finds an edge server close to the client’s local resolver
  - Uses knowledge of network: BGP feeds, traceroutes. *Their secret sauce...*
Example

From Brown

dig www.bestbuy.com

;; ANSWER SECTION:

www.bestbuy.com.edgesuite.net. 21600 IN CNAME a1105.b.akamai.net.
a1105.b.akamai.net. 20 IN A 198.7.236.235
a1105.b.akamai.net. 20 IN A 198.7.236.240
  - Ping time: 2.53ms

From Berkeley, CA

a1105.b.akamai.net. 20 IN A 198.189.255.200
a1105.b.akamai.net. 20 IN A 198.189.255.207
  - Ping time: 3.20ms
Welcome!
> telnet www.cs.brown.edu 80
Trying 128.148.32.110...
Escape character is '^^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 24 Mar 2011 12:58:46 GMT
Server: Apache/2.2.9 (Debian) mod_ssl/2.2.9 OpenSSL/0.9.8g
ETag: "840a88b-236c-49f3992853bc0"
Accept-Ranges: bytes
Content-Length: 9068
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
...
HTTP Cookies

• Client-side state maintenance
  – Client stores small state on behalf of server
  – Sends request in future requests to the server
  – Cookie value is meaningful to the server (e.g., session id)

• Can provide authentication
Modern web pages and HTTP

• Web APIs: HTTP response/requests are a standard way to ask for anything
• Modern web pages: use Javascript to make lots of requests without reloading page
  – And can use APIs for all kinds of other stuff
Example: Github public API

```bash
$ curl https://api.github.com/users/ndemarinis
{
  "login": "ndemarinis",
  "id": 1191319,
  "node_id": "MDQ6VXNlcjExOTEzMTk=",
  "gravatar_id": "",
  "url": "https://api.github.com/users/ndemarinis",
  "type": "User",
  "site_admin": false,
  "name": "Nick DeMarinis",
  "blog": "https://vty.sh",
  "twitter_username": null,
  "public_repos": 10,
  ...
}
```
Modern web pages and HTTP

• Web APIs: HTTP response/requests are a standard way to ask for anything
• Modern web pages: use Javascript to make lots of requests without reloading page
  – And can use APIs for all kinds of other stuff
Example: Github public API

$ curl https://api.github.com/users/ndemarinis
{
    "login": "ndemarinis",
    "id": 1191319,
    "node_id": "MDQ6VXNlcjExOTEzMTk=[",
    "gravatar_id": "",
    "url": "https://api.github.com/users/ndemarinis",
    "type": "User",
    "site_admin": false,
    "name": "Nick DeMarinis",
    "blog": "https://vty.sh",
    "twitter_username": null,
    "public_repos": 10,
    ...
}
HTTP: What matters for performance?

Depends on type of request
- Lots of small requests (objects in a page)
- Some big requests (large download or video)
Small Requests

- Latency matters
- RTT dominates
- Major steps:
  - DNS lookup (if not cached)
  - Opening a TCP connection
  - Setting up TLS (optional, but now common)
  - Actually sending the request and receiving response
How can we reduce the number of connection setups?

• Keep the connection open and request all objects serially
  – Works for all objects coming from the same server
  – Which also means you don’t have to “open” the window each time

Persistent connections (HTTP/1.1)
Small Requests (cont)

• Second problem is that requests are serialized
  – Similar to stop-and-wait protocols!

• Two solutions
  – Pipelined requests (similar to sliding windows)
  – Parallel Connections
    • Browsers implement this differently—see “Inspect element”
  – How are these two approaches different?