CSCI-1680
DNS II + WWW

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, Scott Shenker and John Jannotti

Administrivia

- TCP grading: sign up for a grading meeting
  - Let us know if you don't see any slots
- Final project: you should have received an email about teams
- Project proposal: due by Monday, 12/5
  - Really not much required, just sketch what you want to do and your plan
  - I'll review these daily: submit earlier => earlier feedback!
- There will be a short HW5
- My office hours today: 3-4pm (CIT316, zoom), 5-7pm (location TBA)

More on DNS

DNS Example

```
$ dig cs.brown.edu @10.1.1.10
; <<>> DiG 9.10.6 <<>> cs.brown.edu @10.1.1.10
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8536
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1


;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1220
;; QUESTION SECTION:
;cs.brown.edu. IN A


;; ANSWER SECTION:
cs.brown.edu.            1800      IN      A        128.148.32.12


;; Query time: 69 msec
;; SERVER: 10.1.1.10#53(10.1.1.10)
;; WHEN: Tue Apr 19 09:03:39 EDT 2022
;; MSG SIZE  rcvd: 57
```

DNS record types

RR Type	Purpose	Example
A	IPv4 Address	`128.148.56.2`
AAAA	IPv6 Address	`2001:470:8956:20::1`
CNAME	Specifies an alias ("Canonical name")	`systems.cs.brown.edu. 86400 IN CNAME systems-v3.cs.brown.edu.` `systems-v3.cs.brown.edu. 86400 IN A 128.148.36.51`
MX	Mail servers	MX <priority> <ip> eg. MX 10 1.2.3.4
SOA	Start of authority	Information about who owns a zone
PTR	Reverse IP lookup	`7.34.148.128.in-addr.arpa. 86400 IN PTR quanto.cs.brown.edu.`
SRV	How to reach specific services (eg. host, port)	_minecraft._tcp.example.net 3600 SRV <priority> <weight> <port> <server IP>

More: https://en.wikipedia.org/wiki/List_of_DNS_record_types

“Helpful” ISPs

Some ISPs hijack DNS for “helpful” purposes

- Could rewrite NXDOMAIN responses => search page with ads
  - oogle.com => ISP search page
- Captive portals: When joining public Wifi, respond to all DNS queries with IP of login page
  - Most OSes/browsers have mechanisms to detect this

What can be done?

Some defenses against DNS spoofing/hijacking

What can be done?

Some defenses against DNS spoofing/hijacking

- DNSSEC: protocol to sign/verify hierarchy of DNS lookups
  - Expensive to deploy, hierarchy must support at all levels
  - APNIC DNSSEC monitor: https://stats.labs.apnic.net/dnssec
  - https://www.internetsociety.org/resources/deploy360/2012/nist-ipv6-and-dnssec-statistics-6/

- Tunneling DNS: client uses DNS via more secure protocol
  - DNS over HTTPS
  - DNS over TLS

HTTP: Hypertext Transfer Protocol

HTTP

- "Application protocol for distributed, collaborative hypermedia information systems"
- Fundamental protocol behind "the web"
- Today, HTTP is fundamental of most things we do on the Internet… and thus most modern applications

But what is hypertext?

WIKIPEDIA
The Free Encyclopedia

Hypertext Transfer Protocol

From Wikipedia, the free encyclopedia
(Redirected from HTTP)

The **Hypertext Transfer Protocol** (**HTTP**) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems.[1] HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser.

Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989 and summarized in a simple document describing the behavior of a client and a server using the first HTTP protocol version that was named 0.9.[2]

That first version of HTTP protocol soon evolved into a more elaborated version that was the first draft toward a far future version 1.0.[3]

Development of early HTTP Requests for Comments (RFCs) started a few years later and it was a coordinated effort by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium

Hypertext Transfer Protocol

HTTP

International standard	RFC 1945 HTTP/1.0 RFC 9110 HTTP Semantics RFC 9111 HTTP Caching RFC 9112 HTTP/1.1 RFC 9113 HTTP/2 RFC 7541 HTTP/2:

Hypertext predates HTTP

1945: Vannevar Bush envisions the "Memex":

- *"a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility"*

- Precursors to hypertext
  - "The human mind [...] operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate *web of trails* carried by the cells of the brain"

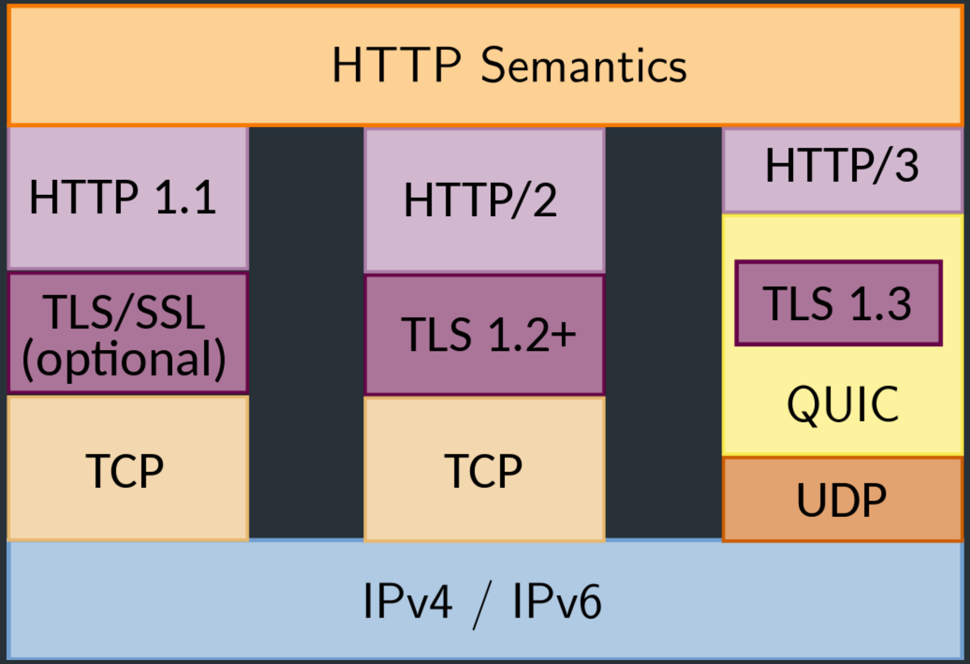
- His essay, "As we may think", is worth reading!

Tim Berners-Lee

- Physicist at CERN, trying to solve real problem
  - Distributed access to data
- WWW: distributed database of pages linked through the Hypertext Transfer Protocol
  - First HTTP implementation: 1990
  - HTTP/0.9 – 1991
    - Simple GET command
  - HTTP/1.0 – 1992
    - Client/server information, simple caching
  - HTTP/1.1 – 1996
    - Extensive caching support
    - Host identification
    - Pipelined, persistent connections, …

The first webserver

- HTTP/2 – 2015
  - Main goal: reduce latency
  - True multiplexing of messages
  - Binary encoding, compression
- HTTP/3 – 2022
  - Same goals as HTTP/2
  - Integrates security via TLS (next class…)
  - Replace transport layer with QUIC
  - Already supported in >70% of browsers



http://httpwg.org/specs/rfc7540.html

Why so successful?

- Ability to self publish
  - Like youtube for video
- But…
  - Mechanism is *easy*
  - Independent, open
  - Free
- Current debate
  - Is it easy enough? Why is facebook so popular, even though it is not open?

Components

- Content
  - Objects (may be static or dynamically generated)
- Clients
  - Send requests / Receive responses
- Servers
  - Receive requests / Send responses
  - Store or generate content
- Proxies/Middleboxes
  - Placed between clients and servers
  - Provide extra functions
    - Caching, anonymization, logging, transcoding, filtering access
  - Explicit or transparent

Ingredients

- HTTP
  - Hypertext Transfer Protocol
- HTML
  - Language for description of content
- Names (mostly URLs)
  - Won’t talk about URIs, URNs

How to find stuff?

- DNS: names for one or more hosts
  - eg. cs.brown.edu

- How do we ask for a specific *resource* from this host?

URL: Uniform Resource Locator

How to find stuff: URLs

protocol://[name@]hostname[:port]/directory/resource?k1=v1&k2=v2#tag

- Name: can identify a client
- *Hostname:* FQDN or IP address
- *Port number: defaults to common protocol port (eg. 80, 22)*
- *Directory:* path to the resource
- *Resource:* name of the object
- After that, various delimiters to specify further, common examples:
  - ?*parameters* are passed to the server for execution
  - #*tag* allows jumps to named tags within document

How to find stuff: URLs

protocol://[name@]hostname[:port]/directory/resource?k1=v1&k2=v2#tag

HTTP

- Client-server protocol
- Protocol (but not data) in ASCII (before HTTP/2)
- Stateless
- Extensible (header fields)
- Server typically listens on port 80 (or 443, with TLS)
- Server sends response, may close connection (client may ask it to say open)

Steps in HTTP$^{(1.0)}$ Request

- Open TCP connection to server
- Send request
- Receive response
- TCP connection terminates
  - How many RTTs for a single request?
- You may also need to do a DNS lookup first!

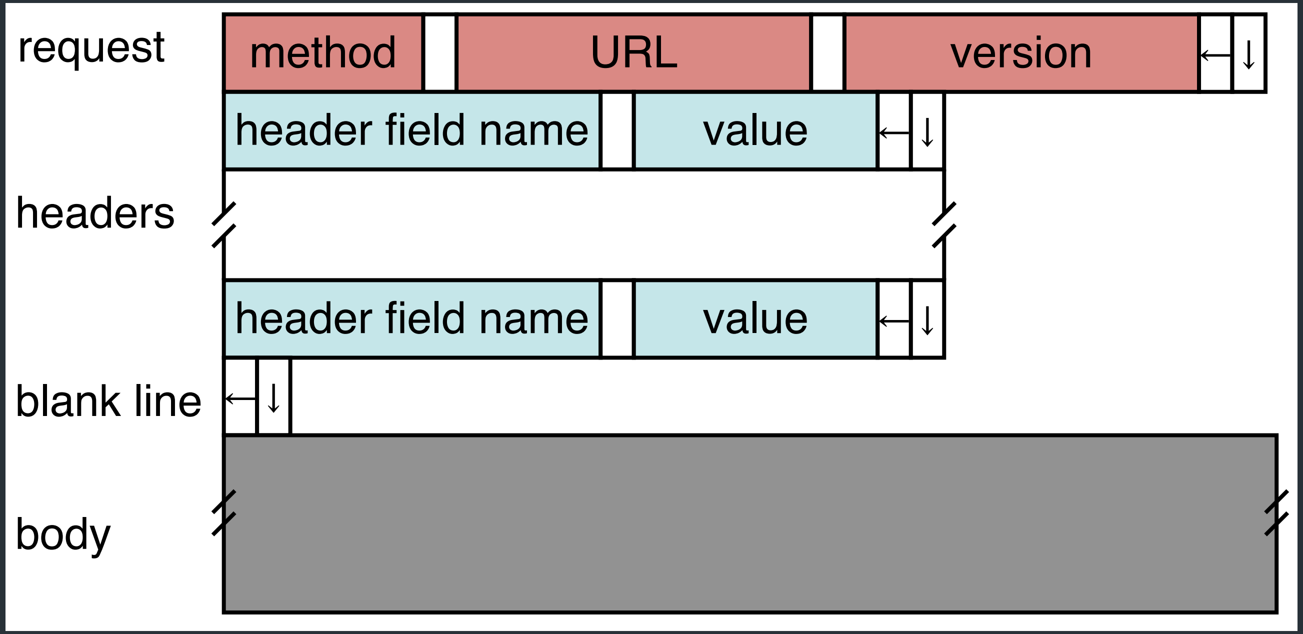
```
> telnet www.cs.brown.edu 80
Trying 128.148.32.110...
Connected to www.cs.brown.edu.
Escape character is '^]'.
GET / HTTP/1.0

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
Last-Modified: Thu, 24 Mar 2011 12:25:27 GMT
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">
```

HTTP Request

- Method:
  - GET: current value of resource, run program
  - HEAD: return metadata associated with a resource
  - POST: update a resource, provide input for a program
- Headers: useful info for proxies or the server
  - E.g., desired language



Sample Browser Request

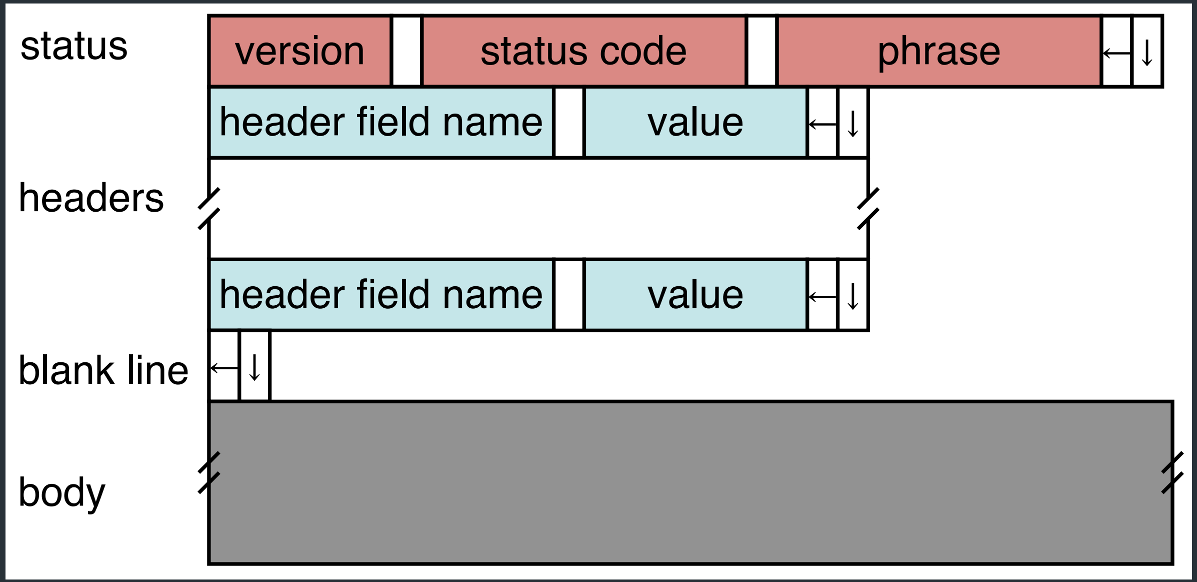
```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
(empty line)
```

In your browser: Inspect element -> Network view

HTTP Response

Status Codes:

- 1xx: Information e.g, 100 Continue
- 2xx: Success e.g., 200 OK
- 3xx: Redirection e.g., 302 Found (elsewhere)
- 4xx: Client Error e.g., 404 Not Found
- 5xx: Server Error e.g, 503 Service Unavailable



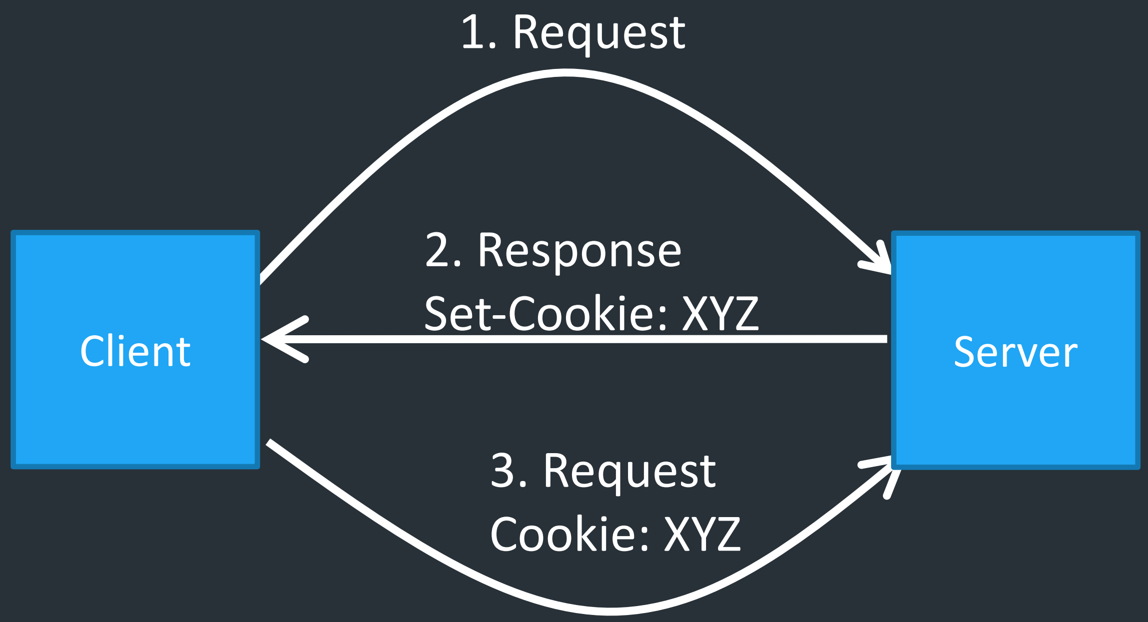
HTTP is Stateless

- Each request/response treated independently
- Servers not required to maintain state
- This is good!
  - Improves server scalability

- This is also bad…
  - Most applications need some persistent state
  - Need to uniquely identify user to customize content
  - E.g., shopping cart, web-mail, usage tracking, (most sites today!)

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



Anatomy of a Web Page

- HTML content
- A number of additional resources
  - Images
  - Scripts
  - Frames
- Browser makes one HTTP request for each object
  - Course web page: 14 objects
  - Modern web pages: hundreds of objects

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=",
  "avatar_url": "https://avatars.githubusercontent.com/u/1191319?v=4",
  "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

```
> telnet www.cs.brown.edu 80
Trying 128.148.32.110...
Connected to www.cs.brown.edu.
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
Last-Modified: Thu, 24 Mar 2011 12:25:27 GMT
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">
```

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)

Browser Request

```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
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
```

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?

HTTP/2

- Adds more options to trade off:
- Multiplexed streams on same connection
  - Plus stream weights, dependencies
- No head of line blocking!
  - But what happens if there is packet loss?

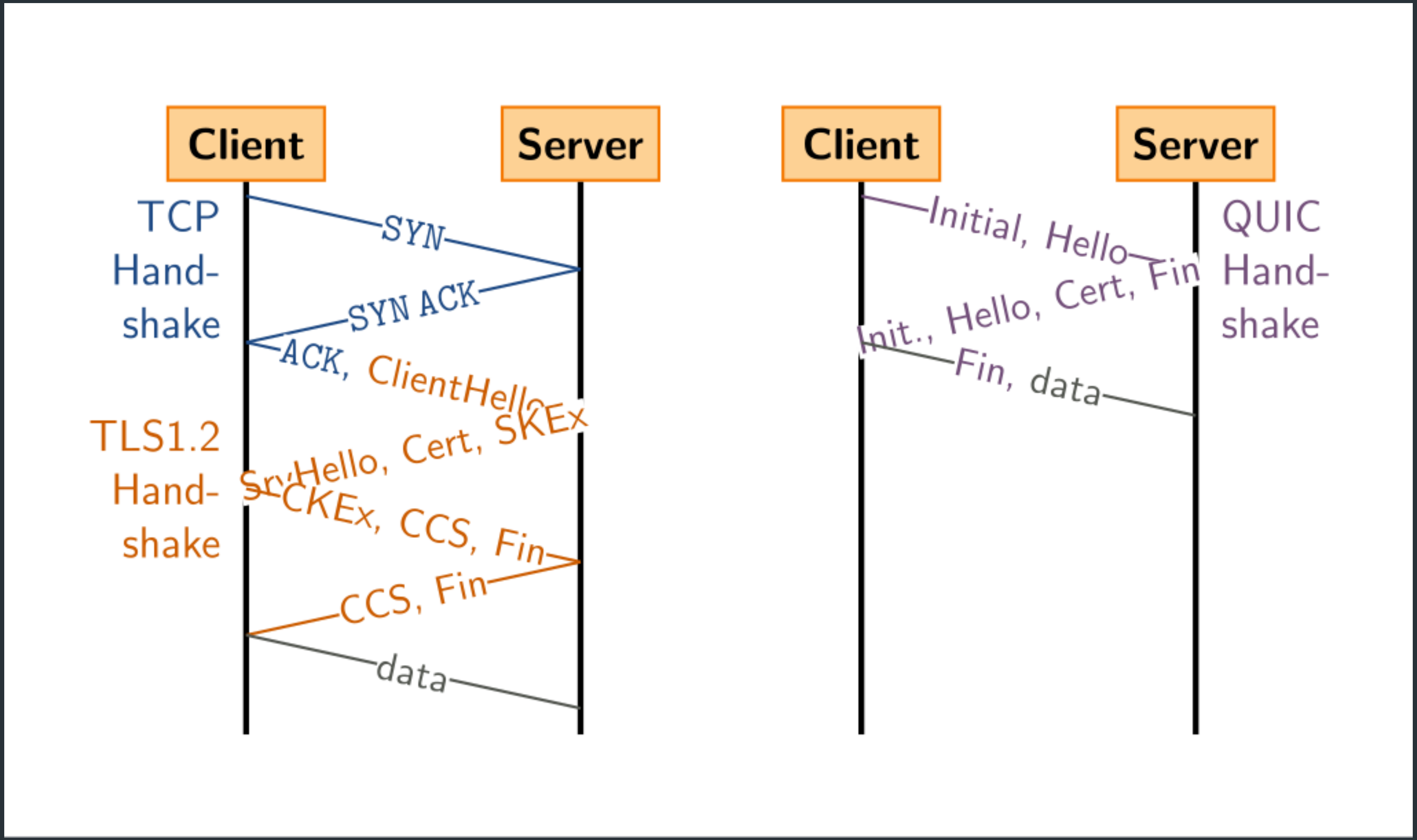
https://www.twilio.com/blog/2017/10/http2-issues.html

HTTP/3

- Mapping of HTTP semantics onto QUIC
  - E.g., QUIC already implements multiple streams, and HTTP doesn't need to do it
- QUIC: Another transport-layer protocol, intended to replace TCP
  - RFC9000
  - Same goals as TCP, but…
  - Integrates security by default (TLS, next class)
  - Supports multiple streams at once
  - Various tricks to reduce message size and latency

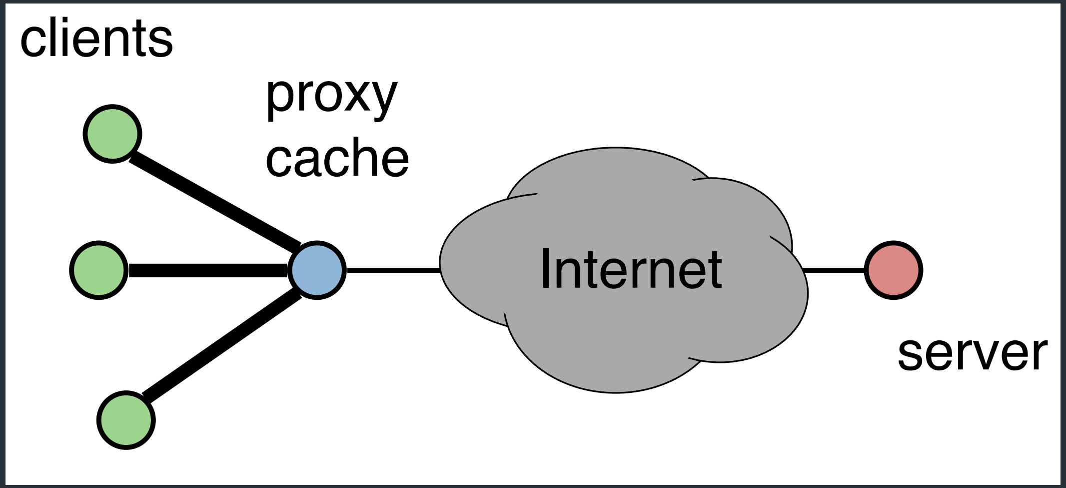
- By moving multiplexing into the transport layer, can do so in a way that benefits HTTP (no head of line blocking!)

Comparison: QUIC's handshake



Larger Objects

- Problem is throughput in bottleneck link
- Solution: HTTP Proxy Caching
  - Also improves latency, and reduces server load

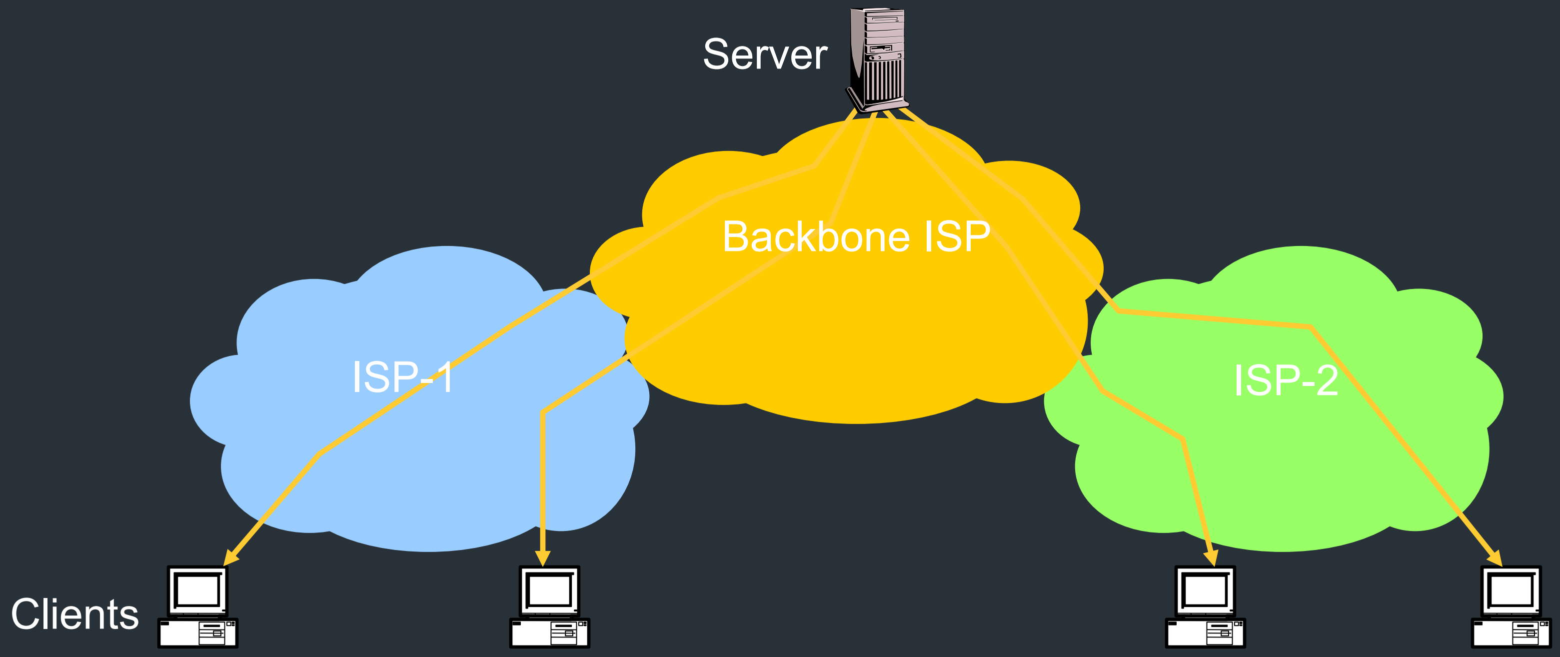


How to Control Caching?

- 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

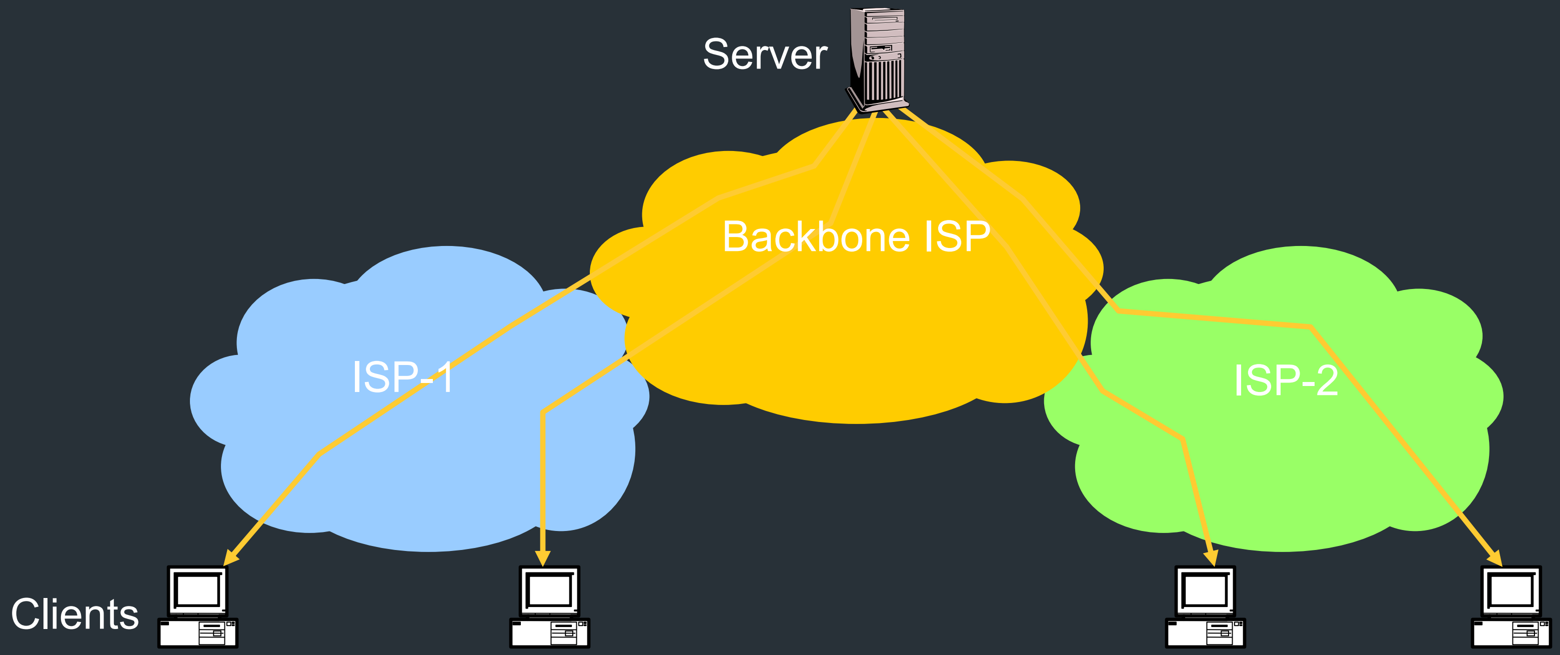
Caching

- Where to cache content?
  - Client (browser): avoid extra network transfers
  - Server: reduce load on the server
  - Service Provider: reduce external traffic



Caching

- Why caching works?
  - Locality of reference:
    - Users tend to request the same object in succession
    - Some objects are popular: requested by many users



How well does caching work?

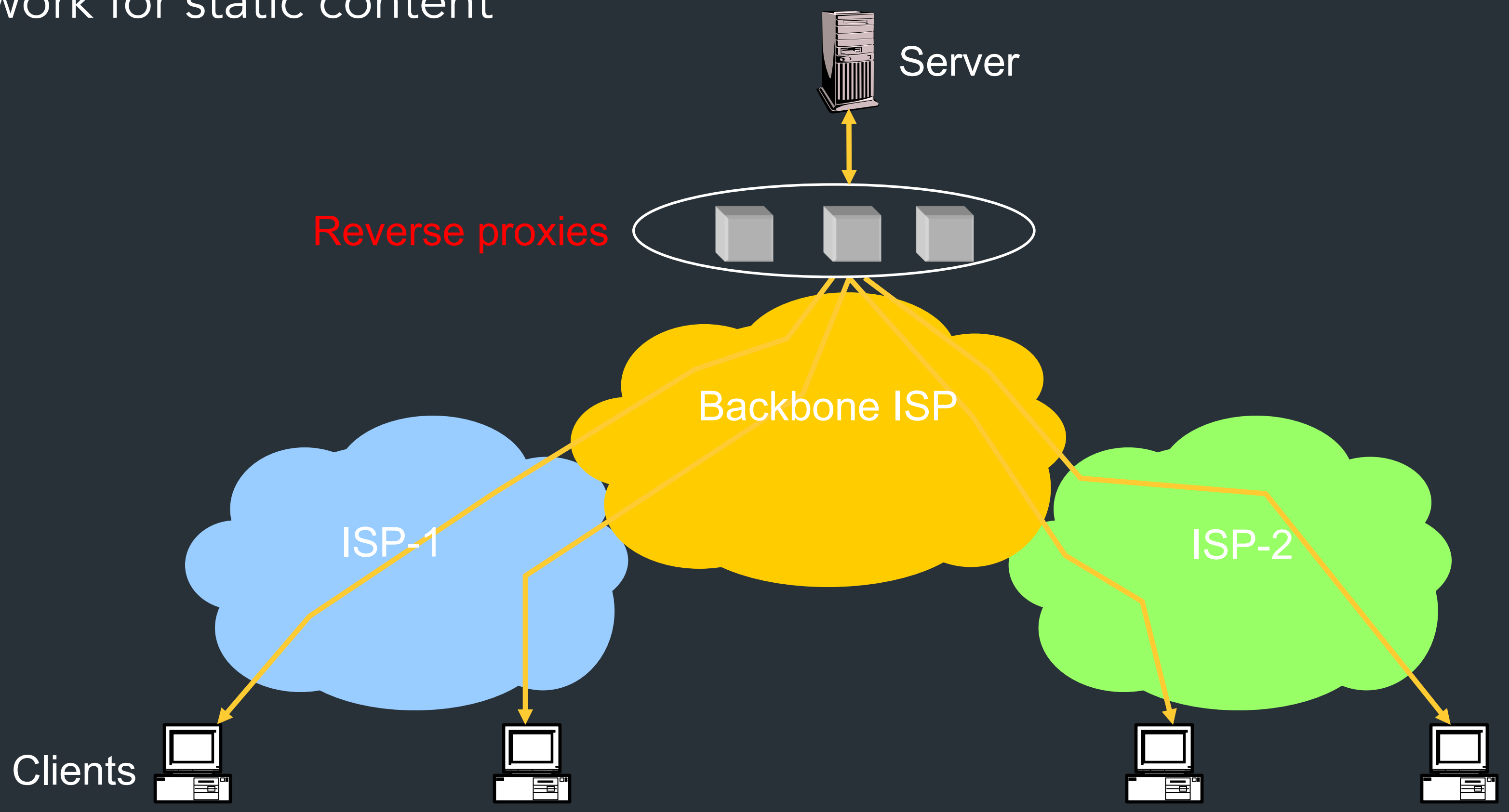
- Very well, up to a point
  - Large overlap in requested objects
  - Objects with one access place upper bound on hit ratio
  - Dynamic objects not cacheable*
- Example: Wikipedia
  - About 400 servers, 100 are HTTP Caches (Squid)
  - 85% Hit ratio for text, 98% for media

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

Reverse Proxies

Close to the server

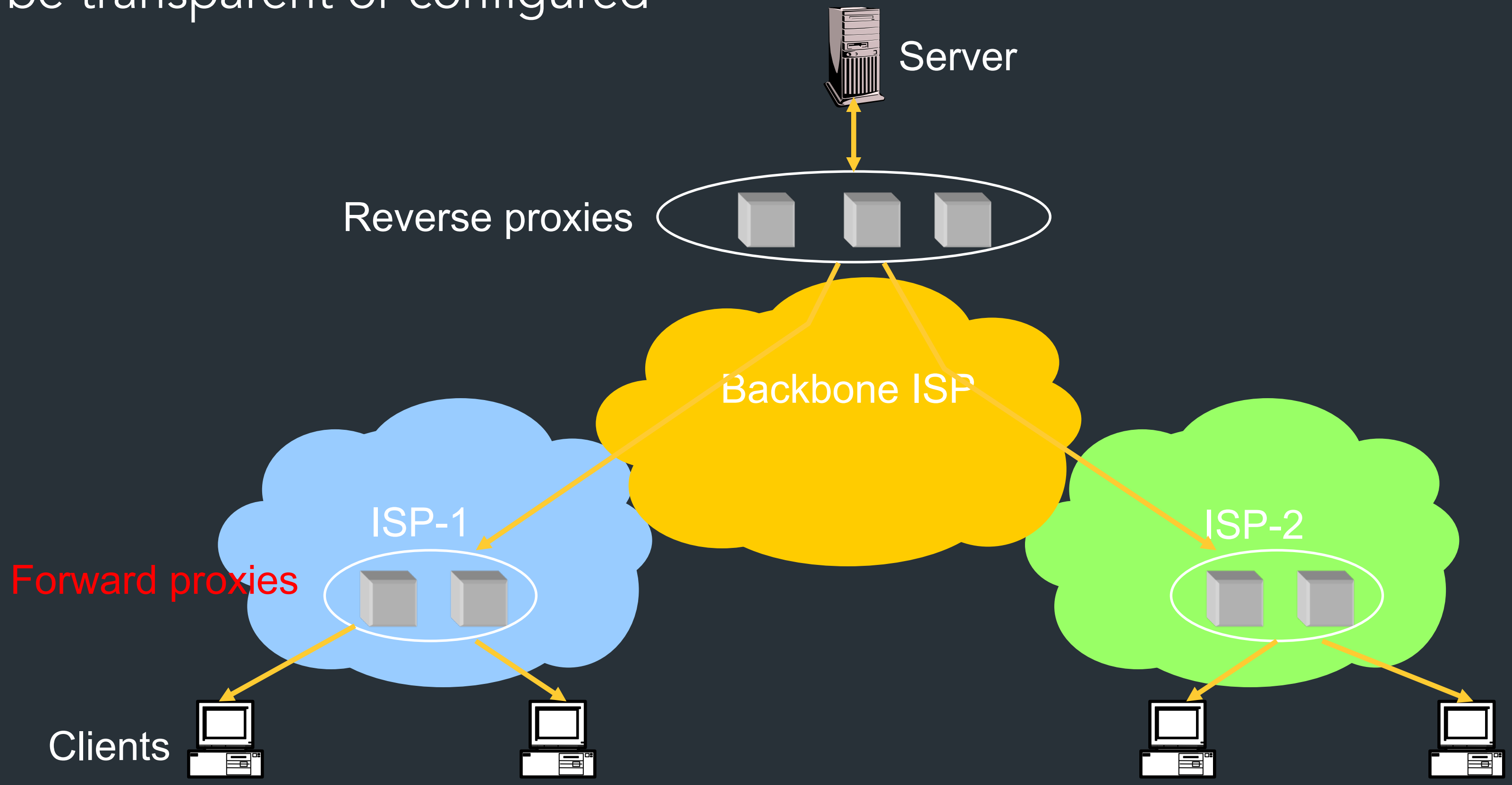
- Also called Accelerators
- Only work for static content



Forward Proxies

Typically done by ISPs or Enterprises

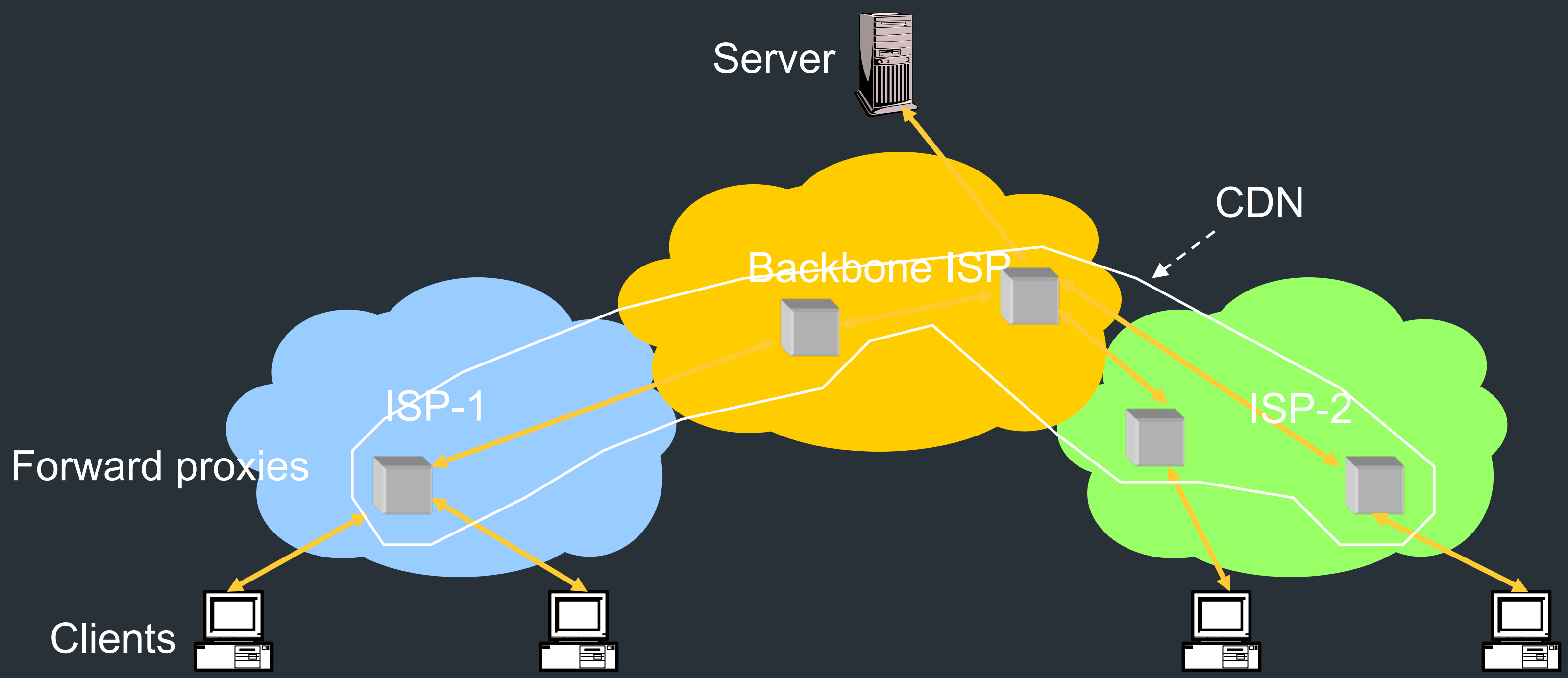
- Reduce network traffic and decrease latency
- May be transparent or configured



Content Distribution Networks

- Integrate forward and reverse caching
  - One network generally administered by one entity
  - E.g. Akamai
- Provide document 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

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)

DNS Resolution

```
dig www.bestbuy.com
;; ANSWER SECTION:
www.bestbuy.com.  3600     IN       CNAME    www.bestbuy.com.edgesuite.net.
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.  3600  IN    CNAME www.bestbuy.com.edgesuite.net.
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

```
dig www.bestbuy.com
;; QUESTION SECTION:
;www.bestbuy.com. IN A

;; ANSWER SECTION:
www.bestbuy.com. 2530 IN CNAME www.bestbuy.com.edgekey.net.
www.bestbuy.com.edgekey.net. 85 IN CNAME e1382.x.akamaiedge.net.
e1382.x.akamaiedge.net. 16 IN A 104.88.86.223

;; Query time: 6 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Thu Nov 16 09:43:11 2017
;; MSG SIZE  rcvd: 123

traceroute to 104.88.86.223 (104.88.86.223), 64 hops max, 52 byte packets
 1  router (192.168.1.1)  2.461 ms  1.647 ms  1.178 ms
 2  138.16.160.253 (138.16.160.253)  1.854 ms  1.509 ms  1.462 ms
 3  10.1.18.5 (10.1.18.5)  1.886 ms  1.705 ms  1.707 ms
 4  10.1.80.5 (10.1.80.5)  4.276 ms  6.444 ms  2.307 ms
 5  lsb-inet-r-230.net.brown.edu (128.148.230.6)  1.804 ms  1.870 ms  1.727 ms
 6  131.109.200.1 (131.109.200.1)  2.841 ms  2.587 ms  2.530 ms
 7  host-198-7-224-105.oshean.org (198.7.224.105)  4.421 ms  4.523 ms  4.496 ms
 8  5-1-4.bear1.boston1.level3.net (4.53.54.21)  4.099 ms  3.974 ms  4.290 ms
 9  * ae-4.r00.bstnma07.us.bb.gin.ntt.net (129.250.66.93)  4.689 ms  4.109 ms
10  ae-6.r24.nycmny01.us.bb.gin.ntt.net (129.250.4.114)  8.863 ms  10.205 ms  10.477 ms
11  ae-1.r08.nycmny01.us.bb.gin.ntt.net (129.250.5.62)  9.298 ms
    ae-1.r07.nycmny01.us.bb.gin.ntt.net (129.250.3.181)  10.008 ms  8.677 ms
12  ae-0.a00.nycmny01.us.bb.gin.ntt.net (129.250.3.94)  8.543 ms  7.935 ms
    ae-1.a00.nycmny01.us.bb.gin.ntt.net (129.250.6.55)  9.836 ms
13  a104-88-86-223.deploy.static.akamaitechnologies.com (104.88.86.223)  9.470 ms  8.483
ms  8.738 ms
```

```
dig www.bestbuy.com @109.69.8.51

e1382.x.akamaiedge.net. 12 IN A 23.60.221.144


traceroute to 23.60.221.144 (23.60.221.144), 64 hops max, 52 byte packets
 1  router (192.168.1.1)  44.072 ms  1.572 ms  1.154 ms
 2  138.16.160.253 (138.16.160.253)  2.460 ms  1.736 ms  2.722 ms
 3  10.1.18.5 (10.1.18.5)  1.841 ms  1.649 ms  3.348 ms
 4  10.1.80.5 (10.1.80.5)  2.304 ms  15.208 ms  2.895 ms
 5  lsb-inet-r-230.net.brown.edu (128.148.230.6)  1.784 ms  4.744 ms  1.566 ms
 6  131.109.200.1 (131.109.200.1)  3.581 ms  5.866 ms  3.238 ms
 7  host-198-7-224-105.oshean.org (198.7.224.105)  4.288 ms  6.218 ms  8.332 ms
 8  5-1-4.bear1.boston1.level3.net (4.53.54.21)  4.209 ms  6.103 ms  5.031 ms
 9  ae-4.r00.bstnma07.us.bb.gin.ntt.net (129.250.66.93)  3.982 ms  5.824 ms  4.514 ms
10  ae-6.r24.nycmny01.us.bb.gin.ntt.net (129.250.4.114)  9.735 ms  12.442 ms  8.689 ms
11  ae-9.r24.londen12.uk.bb.gin.ntt.net (129.250.2.19)  81.098 ms  81.343 ms  81.120 ms
12  ae-6.r01.mdrdsp03.es.bb.gin.ntt.net (129.250.4.138)  102.009 ms  110.595 ms  103.010
ms
13  81.19.109.166 (81.19.109.166)  99.426 ms  93.236 ms  101.168 ms
14  a23-60-221-144.deploy.static.akamaitechnologies.com (23.60.221.144)  94.884 ms  92.779
ms  93.281 ms
```

Other CDNs

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