HTTP & CDNs

Spring 2024
cs168.io

Rob Shakir

Thanks to Iuniana Oprescu for some of the content!

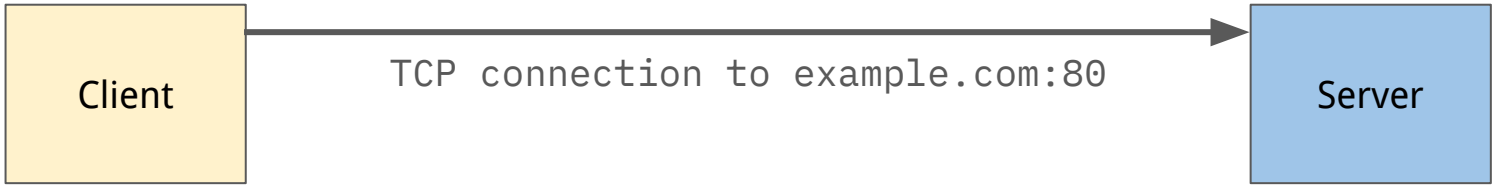
Today

- Think about another application that runs on the Internet.
- What is Hypertext Transfer Protocol - HTTP?
- How do we make HTTP services perform well?
- What are Content Delivery Networks (CDNs)?
- Evolving HTTP.

HTTP

- Development initiated by Tim Berners-Lee at CERN in 1989.
  - Published a specification that was developed to eventually become the first version.
- Driven by a need to have information shared between scientists.
  - Developed the first website – recreated by CERN at https://info.cern.ch/.
- Needed a mechanism to transfer these “hypertext” pages between computers.
  - And hence invented a protocol for it – **H**yper**T**ext **T**ransfer **P**rotocol

HTTP is a TCP-based Request/Response Protocol



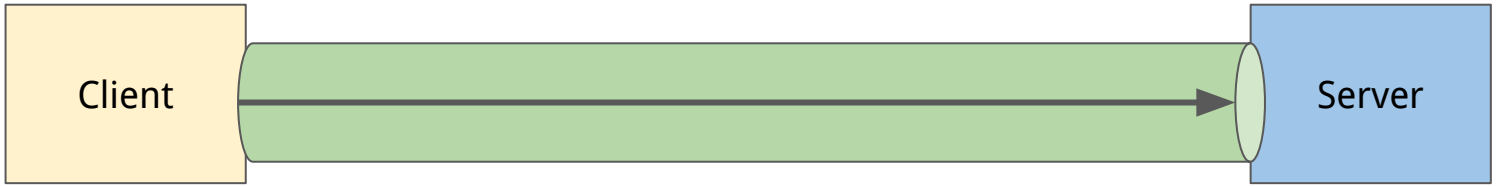
- HTTP runs on a well-known TCP port, 80.
  - We will discuss secure HTTP later, which runs on tcp/443.
- TCP allows for reliable transport of the bytes that make up content.

HTTP is a TCP-based Request/Response Protocol

- An HTTP client generates requests which ask for specific content from the server.

- To start with we are going to think about HTTP/1.1.
  - Initial specification was HTTP/0.9 in 1991.
  - HTTP/1.0 was standardised in 1996.
  - HTTP/1.1 was standardised in 1997.

HTTP is a TCP-based Request/Response Protocol



- HTTP client sends a **request** which the server responds to with a **response**.
- Requests are a fixed format - ended by a carriage return and linefeed (`\r\n`):

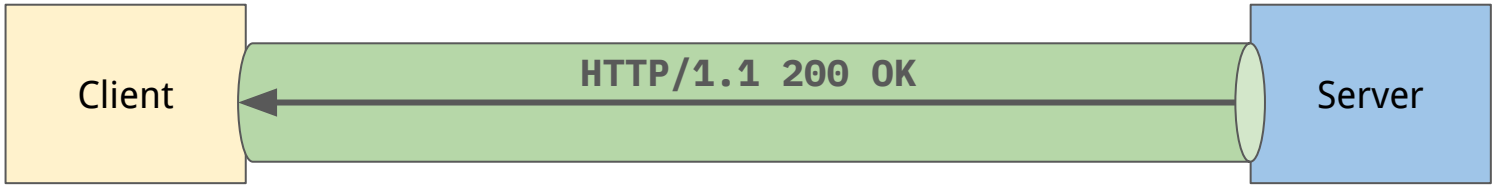
```
<method> <requested URL> <version>
```

HTTP is a TCP-based Request/Response Protocol

- Initially, HTTP had only one method – `GET`.
  - Allowed a client to retrieve a specific URL (page) from the server.

- Clients can include **Headers** which allow additional information to be propagated to the server.

HTTP is a TCP-based Request/Response Protocol



- Following a client request, the server provides a HTTP response.
- Responses are in the format:

```
<version> <status code> <optional message>
<content>
```

HTTP Request Messages

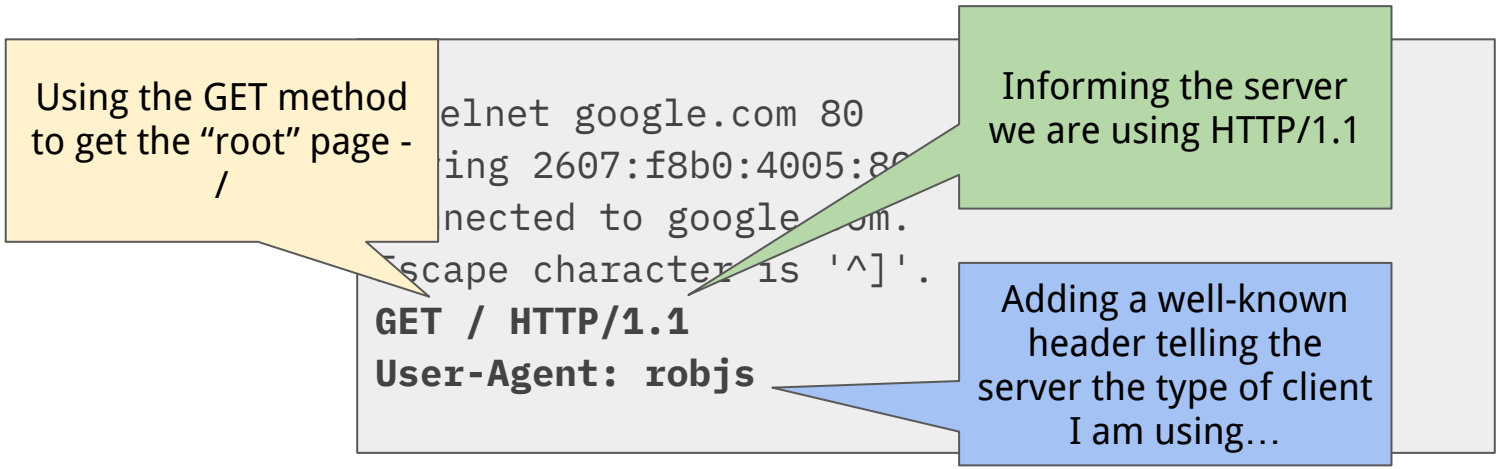
- Simple text-based protocol that has been in the same form for >20 years.

- You can implement this protocol by connecting to a remote server on port 80 and just typing messages…

```
▸ telnet google.com 80
Trying 2607:f8b0:4005:802::200e...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.1
User-Agent: robjs
```

HTTP Request Messages

- Simple text-based protocol that has been in the same form for >20 years.
- You can implement this protocol by connecting to a remote server on port 80 and just typing messages…



HTTP Response Messages

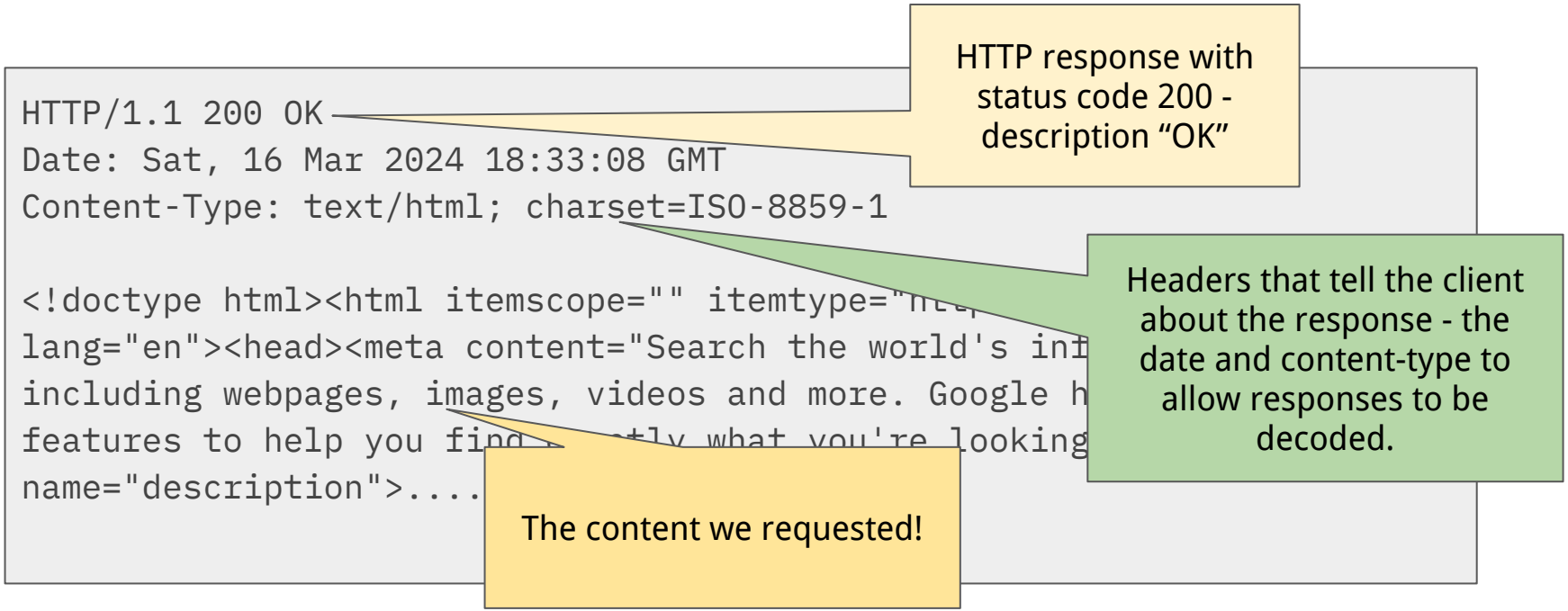
- The server responds back using the same TCP connection with a response…

```
HTTP/1.1 200 OK
Date: Sat, 16 Mar 2024 18:33:08 GMT
Content-Type: text/html; charset=ISO-8859-1

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage"
lang="en"><head><meta content="Search the world's information,
including webpages, images, videos and more. Google has many special
features to help you find exactly what you're looking for."
name="description">....
```

HTTP Response Messages

- The server responds back using the same TCP connection with a response…



Questions?

Types of HTTP Methods

- GET is not the only method that we can use – although it is very common.
- HTTP was extended to add other methods.
- **HEAD**
  - receive the headers of the content that is being requested, but not the content itself.
- **POST**
  - supplying content from the client to the server at the specified URL.
- **PUT, CONNECT, DELETE, OPTIONS, PATCH, TRACE.**
  - Makes HTTP a mechanism for manipulating content – not just receiving it.
  - Client can make changes to content on the server, or retrieve it.

Format of HTTP Requests

```
GET /test.html HTTP/1.1
User-Agent: robjs
```

```
GET <URL> HTTP/1.1
<Headers>
```

- URL allows for the content location on the server to be specified.
- Headers allow for additional information about the client to be propagated to the server.

Format of HTTP Requests

```
GET /test.html HTTP/1.1
User-Agent: robjs
```

```
GET <URL> HTTP/1.1
<Headers>
```

```
POST /test HTTP/1.1
User-Agent: robjs

field1=val1&field2=val2
```

```
POST <URL> HTTP/1.1
<Headers>

<Contents supplied by client>
```

- The URL lets the server know how to parse the information that is received in the body of the request.

Format of HTTP Requests

```
GET /test.html HTTP/1.1
User-Agent: robjs
```

```
GET <URL> HTTP/1.1
<Headers>
```

```
POST /test HTTP/1.1
User-Agent: robjs

field1=val1&field2=val2
```

```
POST <URL> HTTP/1.1
<Headers>

<Contents supplied by client>
```

```
PUT /test.html HTTP/1.1
User-Agent: robjs

<p>Some File</p>
```

```
PUT <URL> HTTP/1.1
<Headers>

<Contents supplied by client>
```

Format of HTTP Responses

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

<html><head>...
```

```
HTTP/1.1 <Status Code> <Description>
<Headers>

<Contents>
```

```
HTTP/1.1 201 Created
Location: foo.html
```

```
HTTP/1.1 <Status Code> <Description>
<Headers>
```

```
HTTP/1.1 201 Created
Content-Location: test.html
```

```
HTTP/1.1 <Status Code> <Description>
<Headers>
```

HTTP Status Codes

- Status codes are used by the server to propagate information about the result of the request to the client.

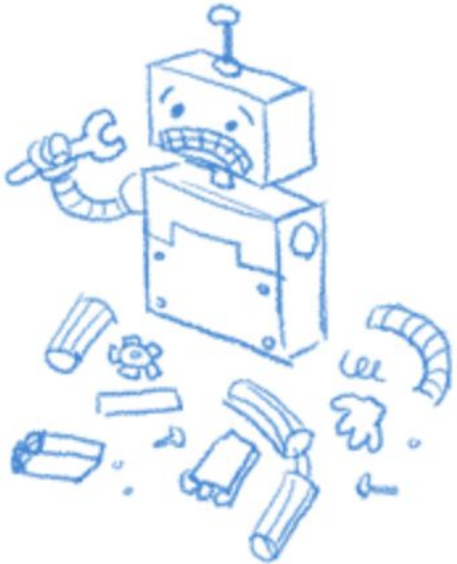
- Classified into various categories – according to numeric value:
  - 100 - Informational responses
  - 200 - Successful responses
  - 300 - Redirection messages
  - 400 - Client error responses
  - 500 - Server error responses

- Some are very recognisable – 404 (File Not Found), 503 (Service Unavailable)
  - You'll probably run into these errors just through your browser.

Google

404. That's an error.

The requested URL /doesnotexist was not found on this server. That's all we know.



google.com/doesnotexist

Common Successful HTTP Status Codes

- `200 – OK`
  - Request was successful.
  - Definition of success depends on the HTTP method that was being used.

- `201 - Created`
  - Request succeeded and some new resource was created.
  - Seen generally in POST or PUT requests.

Common Redirection HTTP Status Codes

- Used when a server is telling a client that they should go and look for the resource (specified by the URL) somewhere else.

- `301 – Moved Permanently`
  - This resource has moved somewhere else!
  - Includes a header – `Location:` `https://some.other.site/newpage.html`

- `302 – Found`
  - This resource has moved somewhere else, but temporarily.
  - Includes a header – `Location:` `https://some.other.site/temppage.html`

- Headers are required to give client additional context.
- Status code lets the client determine future behaviour.
  - e.g., temporarily redirected – client should come back to this URL to check in the future, permanently redirected – client can always go to the new location.

Common Error HTTP Status Codes

- `401 - Unauthorized`
  - Client is not allowed to access this content and must authenticate to do so.
- `403 - Forbidden`
  - Client has authenticated, and the server knows its identity, but access is forbidden.
- `404 – File Not Found`
  - Client is requesting a file that doesn't exist.
- `500 - Internal Server Error`
  - The server hit an error processing the request and can't respond.
- `503 - Service Unavailable`
  - The server cannot respond at the current time.

203

Non-Authoritative Information

httpstatusdogs.com/203

HTTP Error Codes

- There can be some ambiguity as to the status code to be used…

```
▸ telnet google.com 80
Trying 2607:f8b0:4005:80c::200e...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/0.9

HTTP/1.0 400 Bad Request
Content-Type: text/html; charset=UTF-8
Referrer-Policy: no-referrer
Content-Length: 1555
Date: Sat, 16 Mar 2024 19:17:01 GMT
```

Status code could be `505 (HTTP version not supported)`, but rather `400 (Bad Request)` used.

Generally, category of error is the most important (400 or 500 ⇒ error)

HTTP Headers

- In some types of messages, Headers are optional information.
  - e.g., `User-Agent` allows some metadata about the client browser or program to be provided to the server.
  - *Could* result in different processing of the request.

- In other types, Headers are critical information.
  - e.g., `Content-Type` tells the client how to parse the body that is enclosed.
  - e.g., `Host` tells a server that has multiple different web sites hosted on it, which is being addressed.

- However, in HTTP/1.1 no header is mandatory.

Classes of HTTP Headers - Request

- Request Headers
  - Pass information about the client to the server.
- `Accept`
  - Allows the client to determine what encoding of the response should be.
  - e.g., `Accept: text/html`
  - e.g., `Accept: application/json`
  - e.g., `Accept: image/*`
- `Host`
  - Allows the client to specify which host specifically they are aiming to access.
  - e.g., `Host: google.com:80`
- `Referer` [sic], `User-Agent` ...

Classes of HTTP Header - Response

- Used in the response of the message - but does not relate to content.
- `Content-Encoding` - how the server encoded the content to be carried over HTTP.
  - e.g., `Content-Encoding: gzip` says that the server compressed the contents.
- `Date` - when the server generated the response.

Classes of HTTP Header - Representation

- Used in HTTP requests and responses to describe how the content is represented.

- `Content-Type` specifies the document type of the content.
  - e.g., `Content-Type: text/html`
  - e.g., `Content-Type: image/png`

- Representation headers allow us to carry different types of content over HTTP!
  - We can now request an image as well as an HTML page over HTTP!

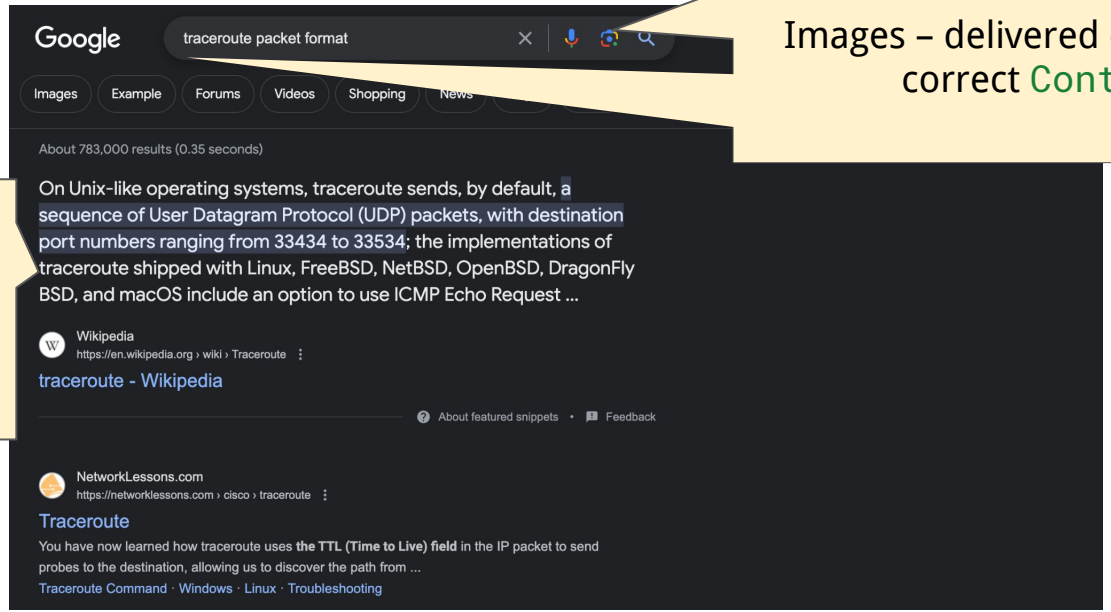
Questions?

HTTP for More than Just HTML Pages

- HTTP is flexible to carry many different types of content.

Images – delivered over HTTP with the correct `Content-Type`.

HTML page delivered as `text/html`.

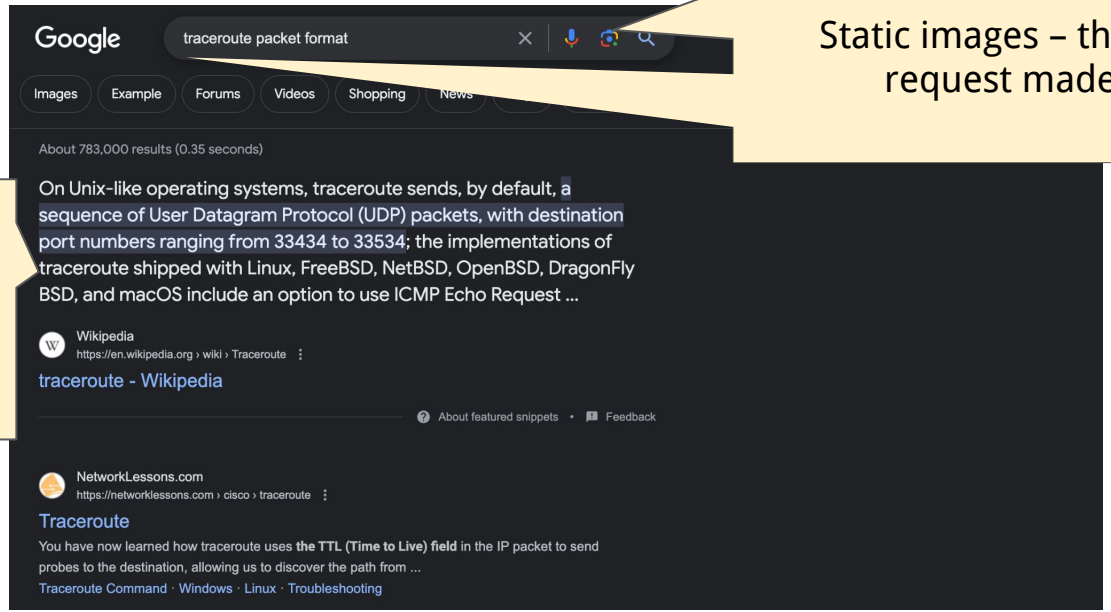


Dynamic vs. Static Content

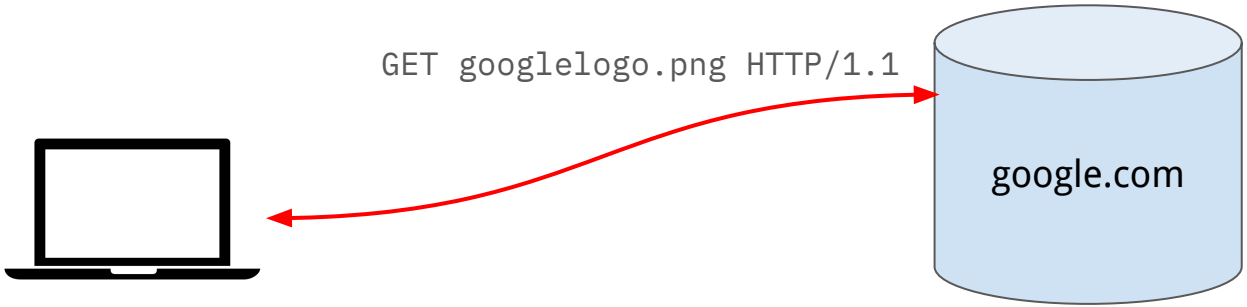
- Whilst dynamic pages might have their content change often, other “resources” (specified by a URL) are static.

Static images – the same for every request made to the page.

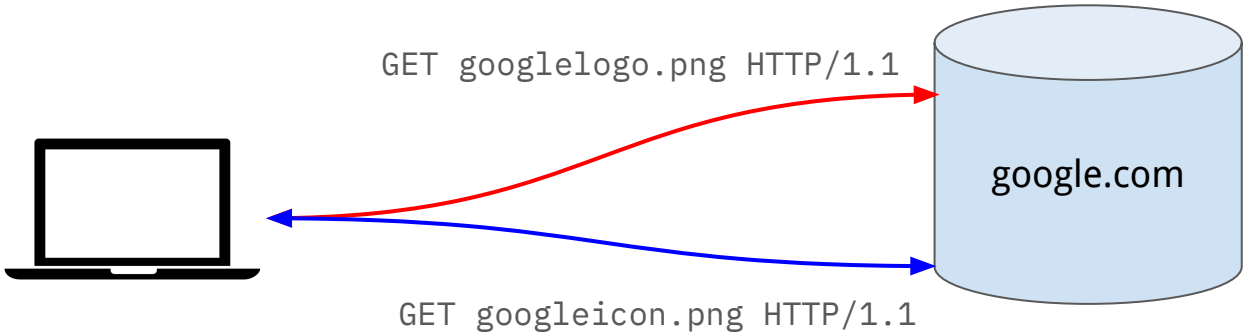
Dynamic content - generated for each different search request.



Improving Web Performance

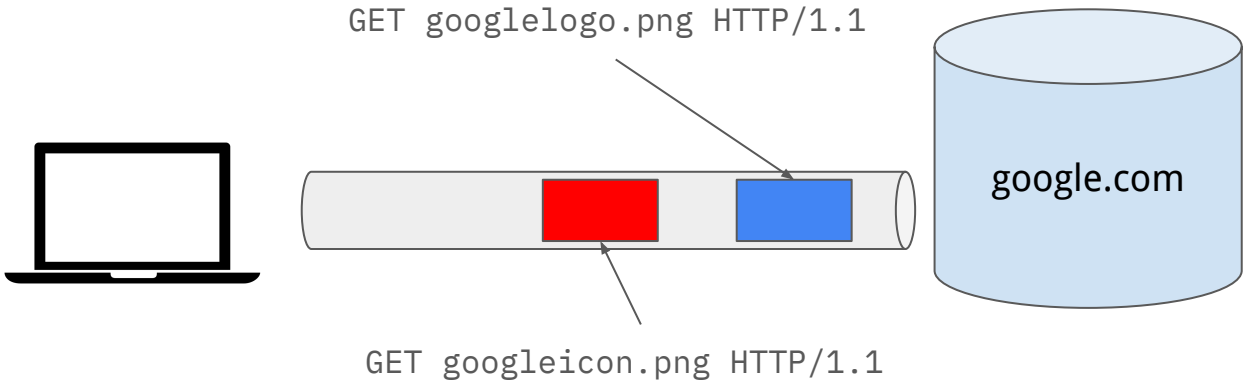


Improving Web Performance



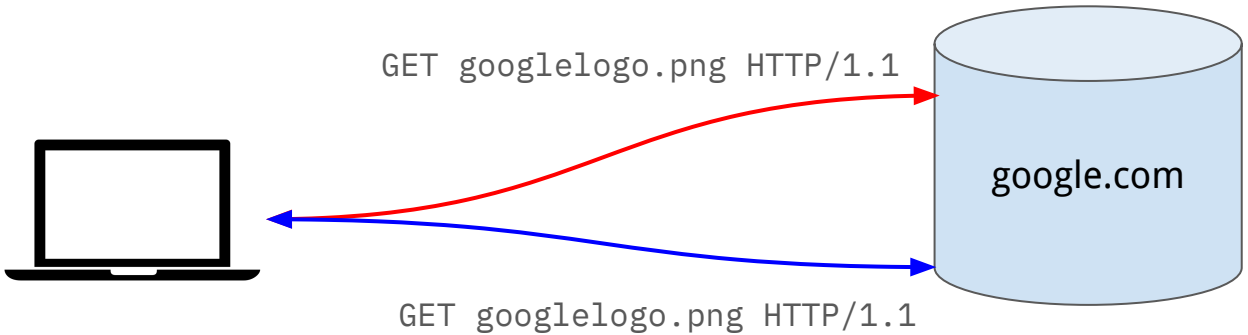
Improving Web Performance

Improving Web Performance - Pipelining



- Rather than requiring a new TCP connection per HTTP request, allow for multiple requests to be “pipelined” over the same connection.
- Often need to load a lot of objects together!
  - e.g., youtube.com HTML page, and then each image for each video.
  - Server must maintain more open connections.

Improving Web Performance



- Rather than requiring a client to load the same content on every request – can we have them cache the content if it won't change?
- Need some way to carry metadata about the content that we returned ⇒ Headers!

HTTP – Headers indicating content validity

- The server can use response headers to indicate when content is valid until.

```
GET http://www.google.com/images/branding/googlelogo/2x/googlelogo_color_150x54dp.png HTTP/1.1

HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Type: image/png
Date: Sat, 16 Mar 2024 19:40:24 GMT
Expires: Sat, 16 Mar 2024 19:40:24 GMT
Cache-Control: private, max-age=31536000

?PNG
IHDR,?R???IDATx??
```

HTTP – Headers indicating content validity

- The server can use response headers to indicate when content is valid until.

```
GET http://www.google.com/images/branding/googlelogo/2x/googlelogo_color_150x54dp.png
HTTP/1.1

HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Type: image/png
Date: Sat, 16 Mar 2024 19:40:24 GMT
Expires: Sat, 16 Mar 2024 19:40:24 GMT
Cache-Control: private, max-age=31536000

?PNG
IHDR,?R???IDATx??
```

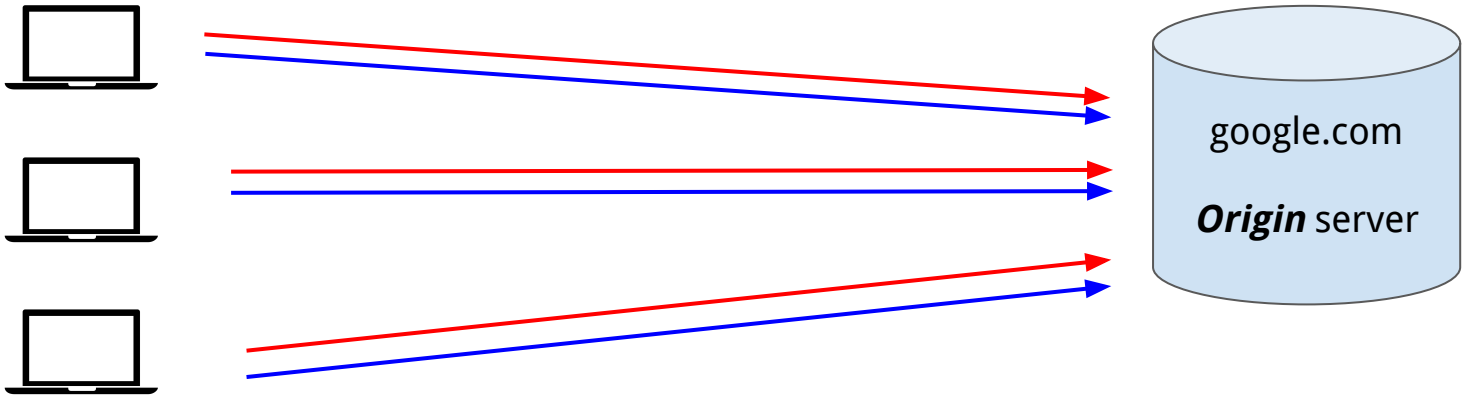
Legacy header used in HTTP/1.0, obsoleted in HTTP/1.1

`Cache-Control` header allows the server to inform the client how to cache the resource.

Types of HTTP Cache

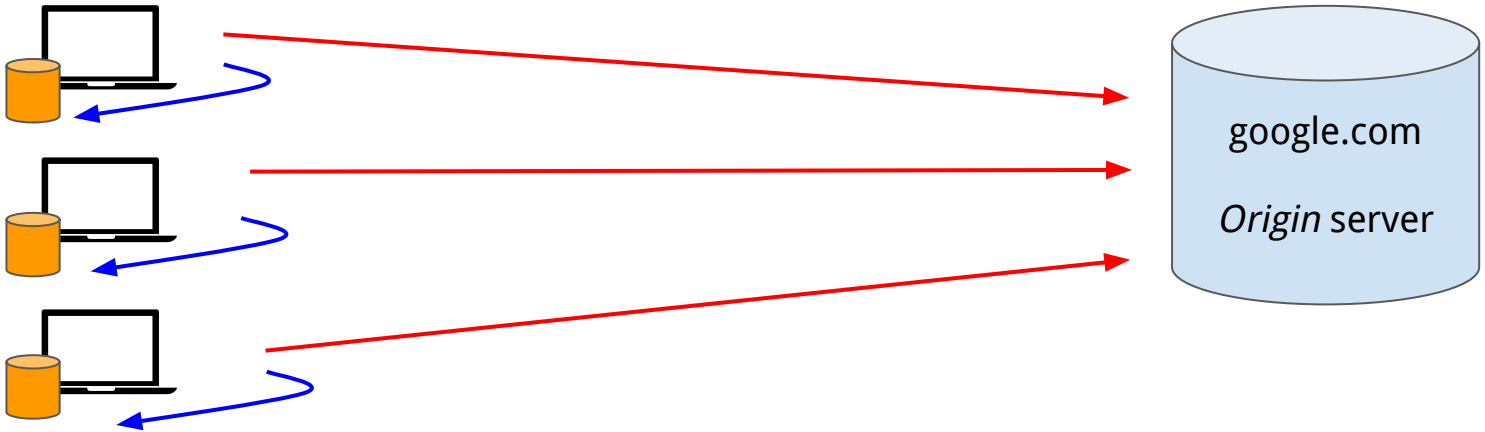
- There are different types of HTTP cache.
- **Private** – tied to a specific end client that is connecting to the server.
  - e.g., a browser's cache.
- **Proxy** – not run by the application provider, but exists in the network to reduce network bandwidth.
- **Managed** – run by the application provider, but is not the original server that generated content.

Operation with No Caching



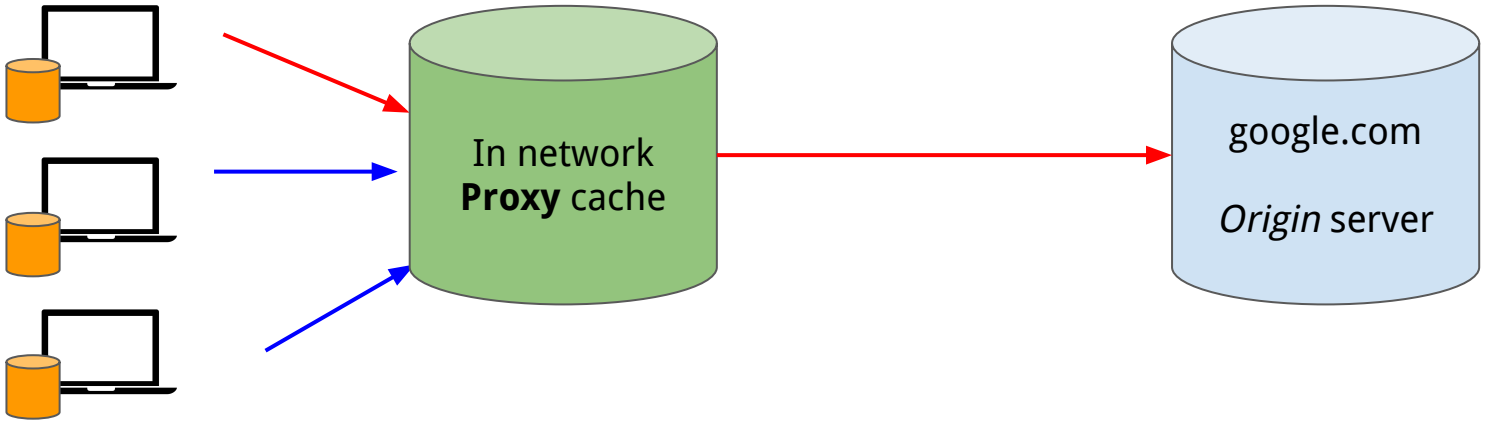
- Every request results in a new request between a client and the origin server.

Private Caches



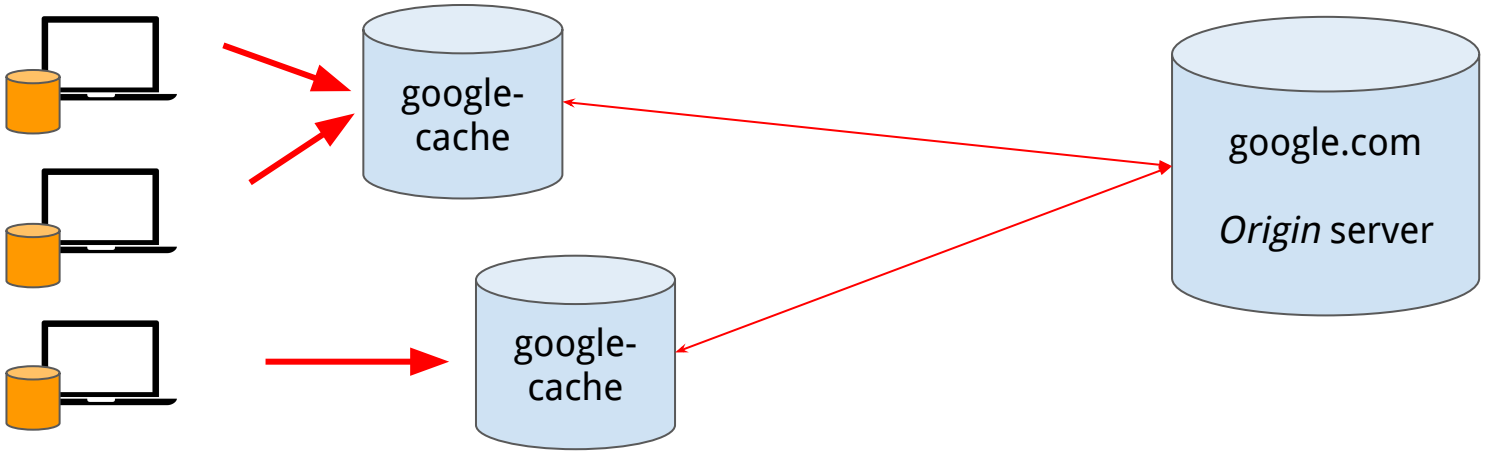
- Introducing private caches at each client means that *cacheable* content will not be retrieved every request.

Proxy Caches



- Introducing a **proxy** cache can reduce the bandwidth needed between a network and the origin server.
  - Most useful where there is low bandwidth out of a particular network.
  - Requires clients to be redirected to the proxy cache.

Managed Caches



- Managed caches allow the application provider to have more control.
  - Achieved by having some redirection mechanism (e.g., different DNS name – `static.foo.com`)
- Improves performance for clients by reducing latency.
  - Faster object retrieval for small content.
  - Higher throughput for large content.

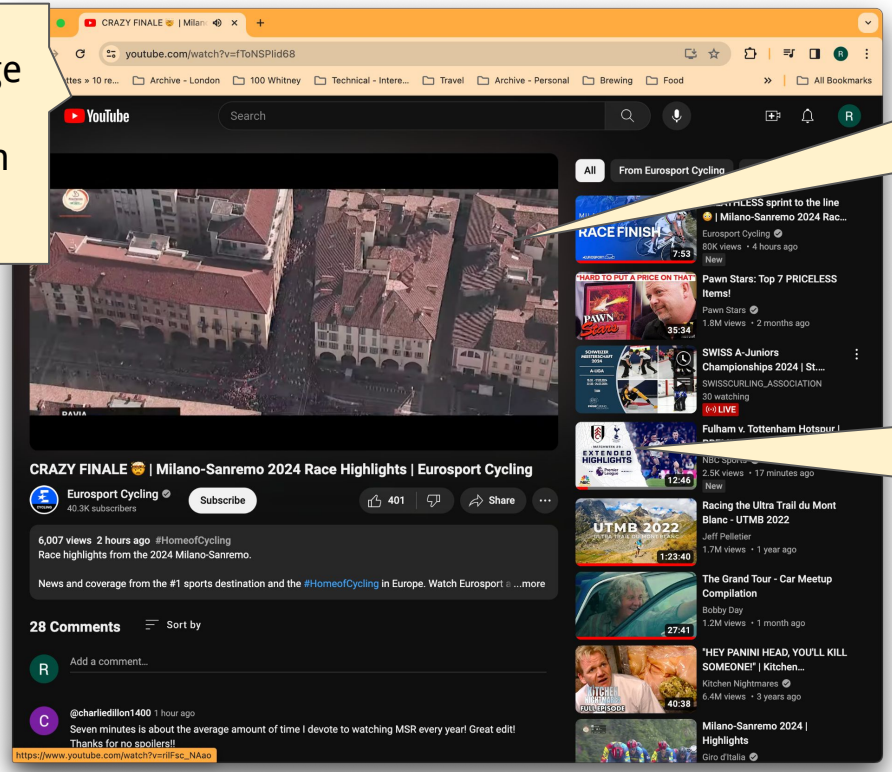
Questions?

`Cache-Control` Header

- How do we control how long these caches hold on to some resource?
- Use a header that specifies the type of cache and what the required behaviours are for such caches.
  - No contract though – really a request!
- `Cache-Control: private,max-age=86400`
  - `private` allows us to specify that this applies to a private (browser) cache.
  - `max-age` allows the owner of the content to specify how long to store the contents before invalidating the cache.
- `Cache-Control: no-store`
  - Client/proxy is not allowed to cache the content.
- More complex policies possible.
  - E.g. “revalidate before using cache” (using `HEAD` etc.).

Loading a Complex HTTP Application

Dynamic HTML page generated in response to search or user.



Video delivered over HTTP - large amount of data!

Images delivered over HTTP – more data than HTML of web page.

Improving Application Performance

- Want to achieve the best TCP throughput we can for our application.
  - Especially important for the larger objects on the page – i.e., video and images.
  - TCP throughput $\propto$ 1/RTT
- Conveniently, the larger objects are static.
  - Image and video content does not change based on user.
- So, can we find a way to use proxies to be able to improve our load time?
  - Only go to the origin server for the dynamic content (HTML page), and have all the static objects loaded from a proxy.

Using Caches for Content Delivery

- Private caches – implies the user accesses the same content multiple times.
  - Some performance improvements, but only on the second access.

- Proxy caches – must be installed by the network operator.
  - And need some changes to the client to know to access them.
  - May not obey the rules specified in the `Cache-Control` header.

- **Managed caches**
  - Can be controlled by the application provider.
  - Can be placed “close” to end users.
  - Redirects can be achieved by the application provider.

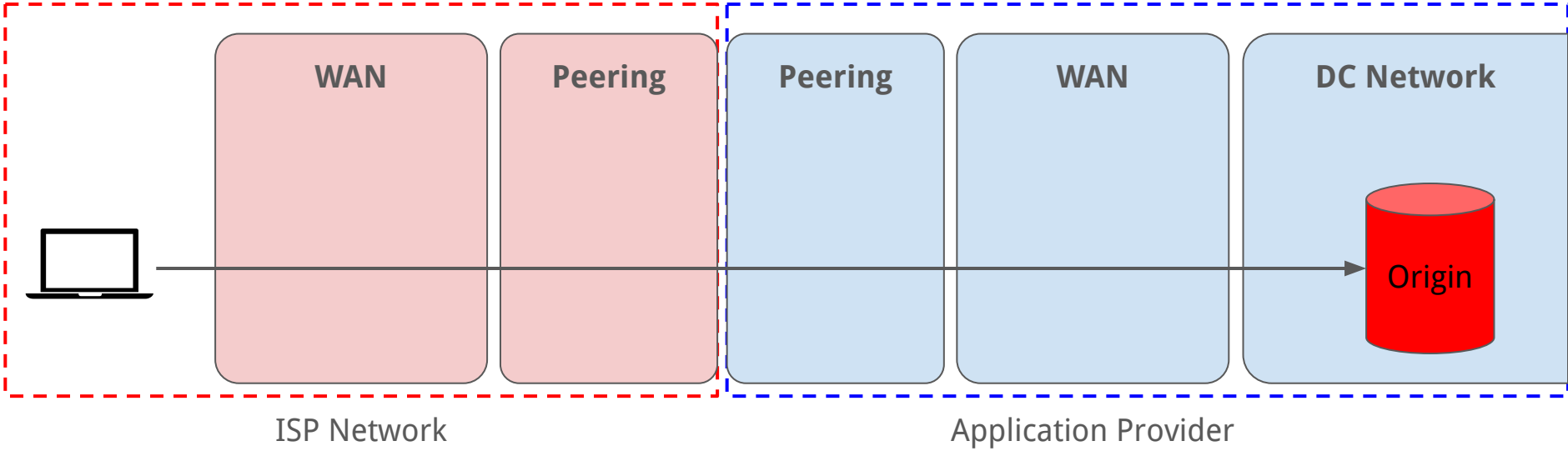
Content Delivery Networks (CDNs)

- Deployments of servers that can serve content (HTTP resources!)

- “Close” to end users.
  - Geographically.
  - From a network perspective.

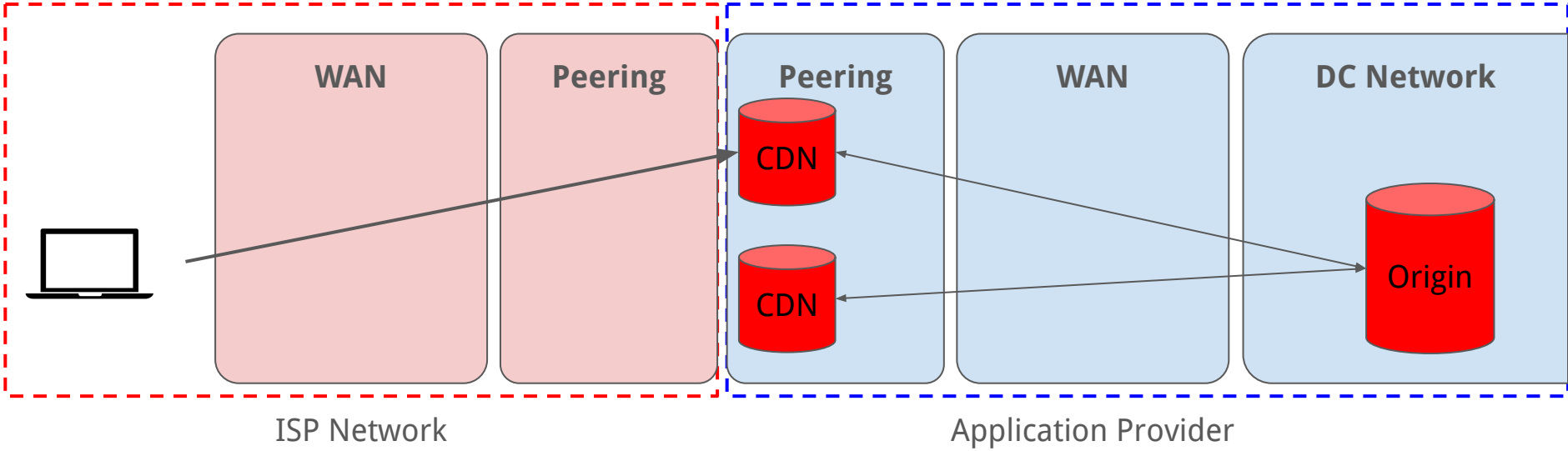
- Allow for:
  - Higher-performance delivery of content (low-latency access to a close server)
  - Significant reductions in the bandwidth needed in the network.
  - Reduces the scaling needed for server infrastructure.
  - Allows for new modes of handling failures.

CDN Deployments.



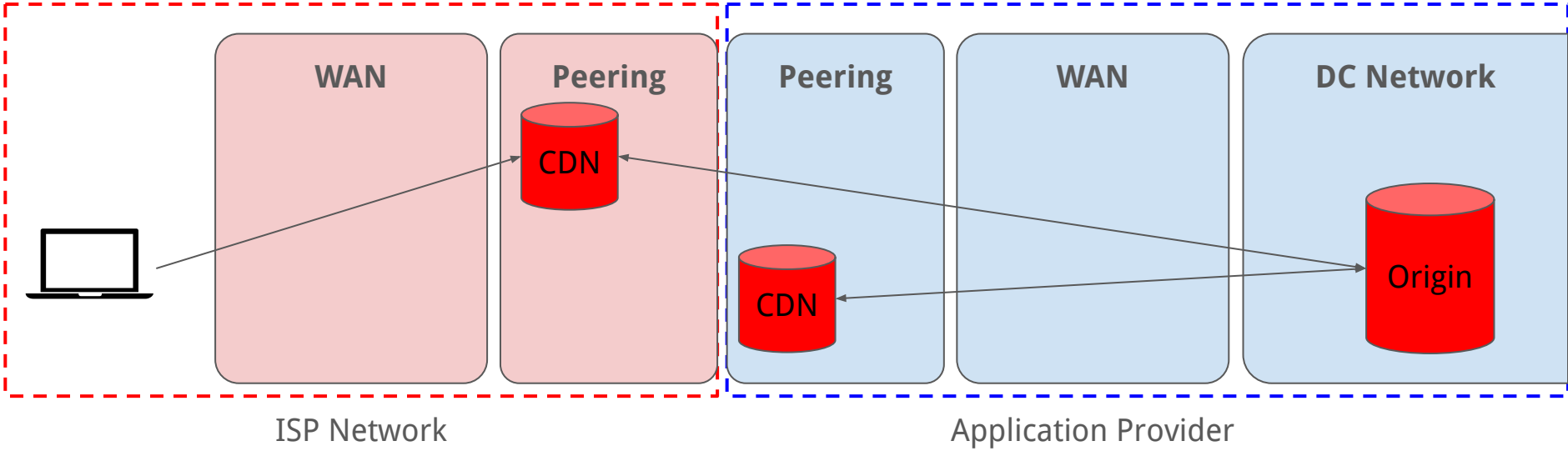
- Clients going to the origin:
  - Maximum latency ⇒ lowest performance.
  - Maximum amount of “backbone” network traversed ⇒ highest cost.
  - Scale must be supported on the origin server.

CDN Deployments.



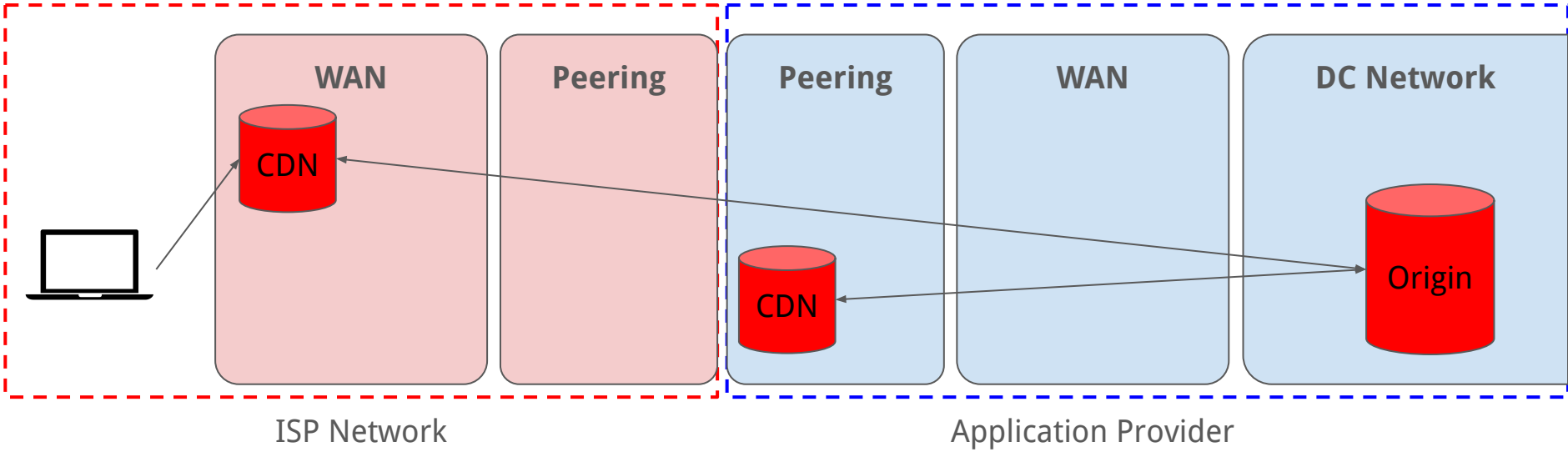
- CDN infrastructure can be deployed in the application provider.
  - Smaller sets of servers at the “edge” of the application provider’s networks.
  - Reduces the volume of backbone traffic for the application provider.
  - Reduces scale per deployment.

CDN Deployments.



- Can push caching “deeper” into the network.
  - Deploy in the ISP’s network – improves performance and reduces cost.
  - ISP reduces their backbone network cost.

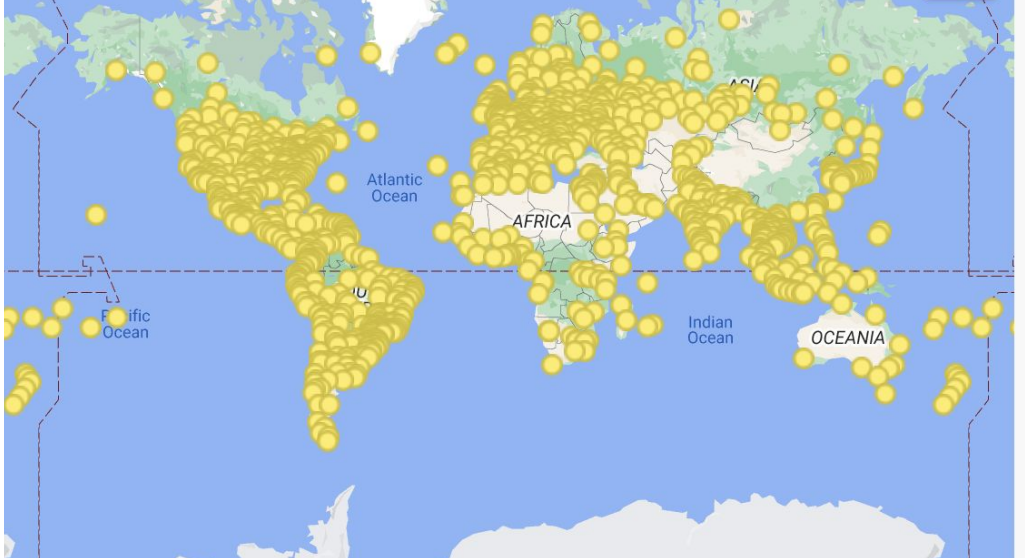
CDN Deployments.



- Deployment depth is limited by efficiency.
  - Need multiple users to be accessing the same content.
  - Cost savings are only worth it if the cost of the additional server infrastructure is less than the network capacity.

Large Global CDNs

- Specific CDN providers.
  - Akamai, Cloudflare, Edgio.
- Large application providers.
  - Netflix, Google, Amazon, Meta.
- Deployments either in their own networks, or directly into ISP networks.



https://peering.google.com/#/infrastructure

https://www.cloudflare.com/network/

CDNs in ISP Networks

- Often ISPs have their own content.
  - Video-on-Demand, or Live TV content as part of TV+Internet bundles.
- CDN server infrastructure is also deployed by these ISPs.
- Often a need for both third-party caches and ISP's own infrastructure.
  - Sandvine report
    - Netflix - 15% of Internet traffic,
    - YouTube - 11.4% of traffic,
    - Disney+ - 4.5% of traffic.
- Deploying caches can mean reducing ~25% of network capacity!

Questions?

Mapping Clients to Caches - recall.

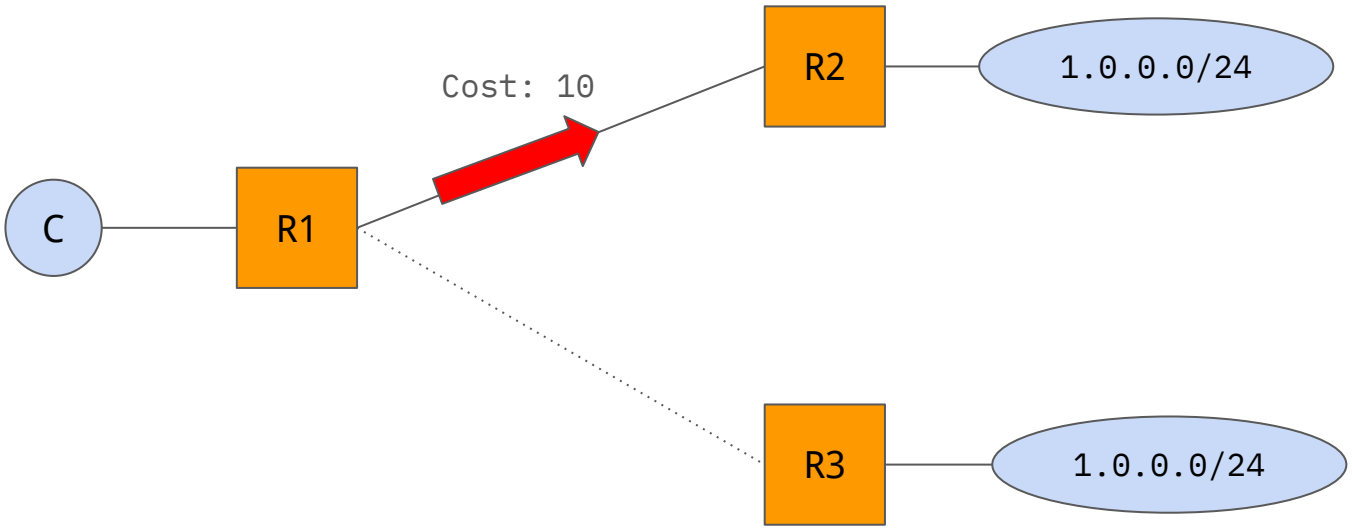
Recall from our DNS lecture:

- **Anycast** – advertise the same IP prefix from multiple locations, allow *least-cost routing* to choose the best location.

- **DNS-based load-balancing** – use the resolver/client's address to be able to choose what response to give.

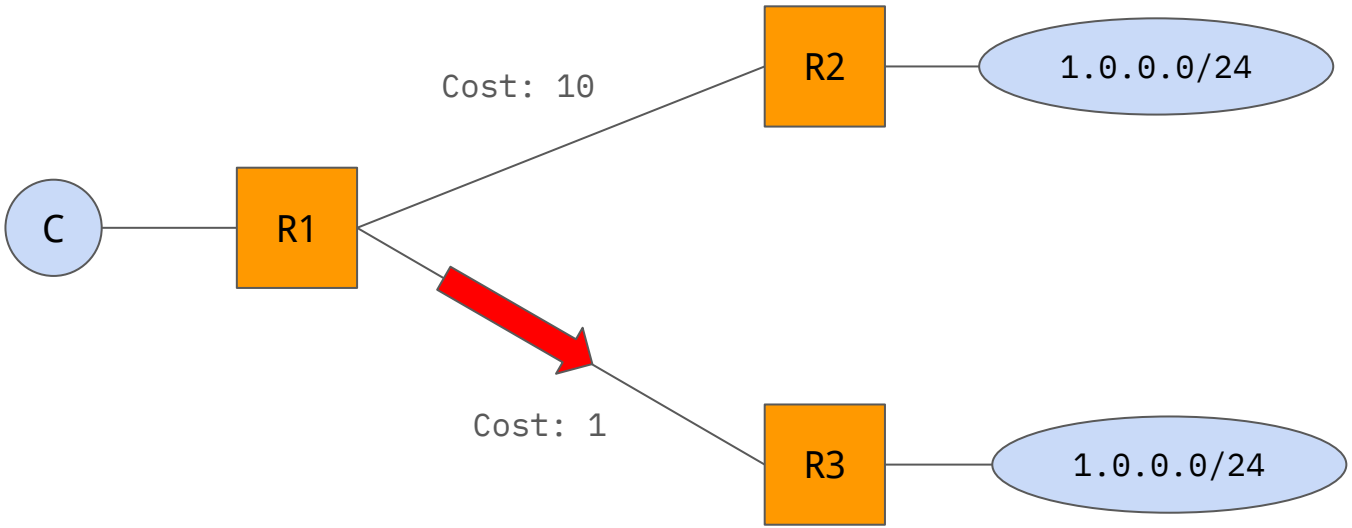
Mapping Clients to Caches

- Anycast-based mapping – may have problems with long-lived connections.
  - Routing can change!



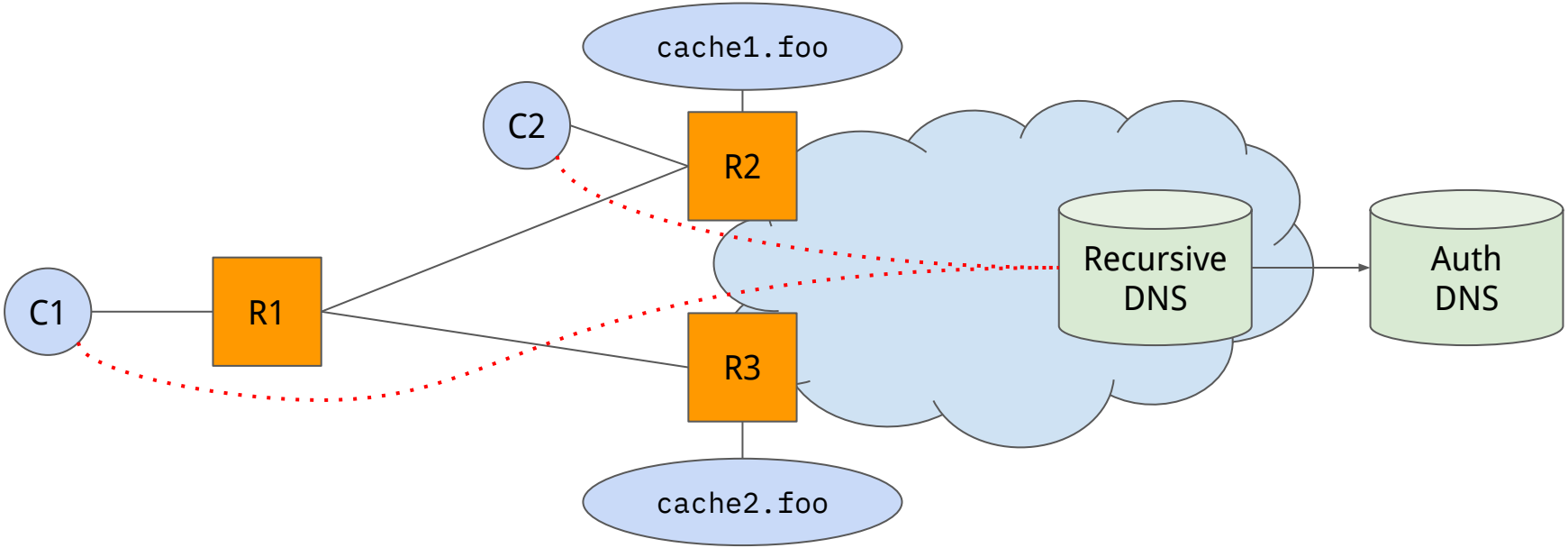
Mapping Clients to Caches

- Anycast-based mapping – may have problems with long-lived connections.
  - Routing can change!



DNS-Based Mapping

- Allows stable mapping – but only at the granularity of client address.
  - May be at the resolver level.
  - EDNS extensions for client information may not be available



Application-Level Mapping

- Application can determine for a specific client where to map a user.
  - If the client is in Berkeley, give the DNS name of a cache in San Francisco.
- Application servers know the remote client's address.
- Still need to understand the "closest" cache to a client.
  - And what the right strategy for failures is.
- Allows for mapping at per-content item granularity.
  - e.g., Cat videos are served at cache in Berkeley
  - e.g., Niche content is served from cache close to the origin.

Caching Server Deployments

- Highly optimised for content delivery and storage.

Flash appliance focus areas

- 2U for rack efficiency (no deeper than 29 inches)
- Enough low cost NAND to reach 24GB/s of throughput (<0.3 DWPD)
- Connect at up to 2X100G LAG
- 2 and 4 post racking
- AC or DC power
- Single processor

Storage appliance focus areas

- Large storage capacity
- 2U for rack efficiency (no deeper than 29 inches)
- Enough low cost NAND to reach 10GB/s of throughput (<0.3 DWPD)
- Network flexibility to connect at 6x10G LAG or up to 2x100GE
- 2 and 4 post racking
- AC or DC power
- Single processor

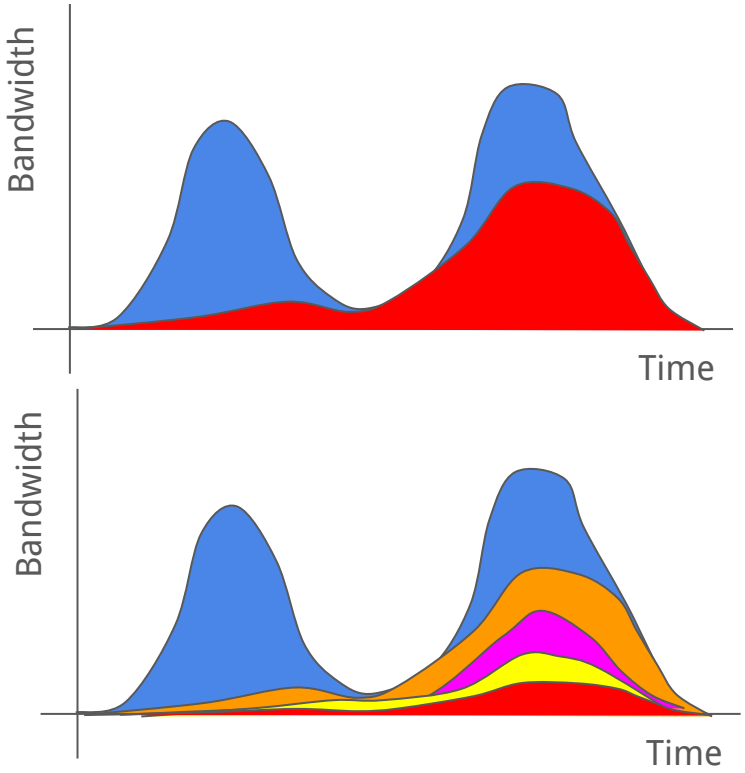
https://openconnect.netflix.com/en/appliances/

Commercial Model

- Mutually beneficial!
  - Content provider gets better application performance ✅
  - ISP gets lower bandwidth costs ✅
- Cooperative commercial model:
  - Content provider usually provides the servers for free.
  - ISP usually pays the fees for hosting them.
- In some cases, commercial negotiations required.
  - Cost of power/space might be more “deeper” into the network.
- Becomes more difficult as there are more caching providers.

Commercial Challenges - Fragmentation

- Cache deployment makes sense if there are small numbers of large content providers.
- Long-tail of content providers.
  - [Sandvine, 2023] Disney+ 4.5% of traffic, Amazon Prime 2.8%.
- Idea: can we have shared caching infrastructure?
  - CDN Interconnect (CDNI) - IETF
  - OpenCaching
- Challenging!
  - Who ensures quality?
  - How are resources shared?



Questions?

Beyond HTTP/1.1

- Lots of applications are over HTTP!

- Security of HTTP is a concern.
  - HTTPS introduces security – Transport Layer Security (TLS) handshakes for certificate exchange.
  - Subsequent communications are encrypted.
  - Majority of traffic on the Internet is now HTTPS.
    - [W3Techs] 85.4% of sites are default HTTPS.

Beyond HTTP/1.1 - HTTP/2.0

- Introduced in 2015 (first new revision since 1997!).

- Aimed to improve performance:
  - Decrease latency and improve page load speed.
    - Data compression of headers.
    - Server-side pushing (server can send objects the client will need!)
    - Prioritisation of requests.
    - Better multiplexing of requests over an HTTP connection.

- Widely adopted across client software (browsers, RPC software) and CDNs.

Beyond HTTP/1.1 - HTTP/3.0

- Introduced in 2022.
- Semantics are the same as HTTP/2.0, but adopts a new underlying transport – QUIC.
- QUIC:
  - Quick **UDP** Connections.
  - Designed at Google, standardised in IETF.
- Avoids some of the impact of TCP reliability mechanisms on HTTP performance.

Recap

- HTTP is a protocol used to transfer data between a client and server – originally designed for HTML web pages.
- HTTP consists of request and response messages with headers in them – allowing for different types of content to be carried over it.
- Performance of HTTP can be improved through *caching* static content – HTTP provides means to control how this caching is used.
- Content Delivery Networks (CDNs) provide infrastructure to allow for this caching to be implemented to improve application performance.