

Computer Networks

Jiaqi Zheng

Material with thanks Mosharaf Chowdhury, and many other colleagues.

Agenda

- HTTP and the Web
- Improving HTTP Performance

The Web: History

- World Wide Web (WWW): a distributed database of “pages” linked through Hypertext Transport Protocol (HTTP)
 - First HTTP implementation – 1990
 - » [Tim Berners-Lee](#) at CERN
 - HTTP/0.9 – 1991
 - » Simple GET command for the Web
 - HTTP/1.0 – 1992
 - » Client/server information, simple caching

The Web: History (cont'd)

- World Wide Web (WWW): a distributed database of “pages” linked through Hypertext Transport Protocol (HTTP)
 - HTTP/1.1 – 1996
 - » Performance and security optimizations
 - HTTP/2 – 2015
 - » Latency optimizations via request multiplexing over single TCP connection
 - » Binary protocol instead of text
 - » Server push

Web components

- Infrastructure:
 - Clients
 - Servers (DNS, CDN, Datacenters)
- Content:
 - URL: naming content
 - HTML: formatting content
- Protocol for exchanging information: HTTP

URL: Uniform Record Locator

- `protocol://host-name[:port]/directory-path/resource`
- Extend the idea of hierarchical hostnames to include anything in a file system
 - `https://github.com/mosharaf/eecs489/blob/master/Slides/010417.pptx`
- Extend to program executions as well...
 - `http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b`
 - Server side processing can be included in the name

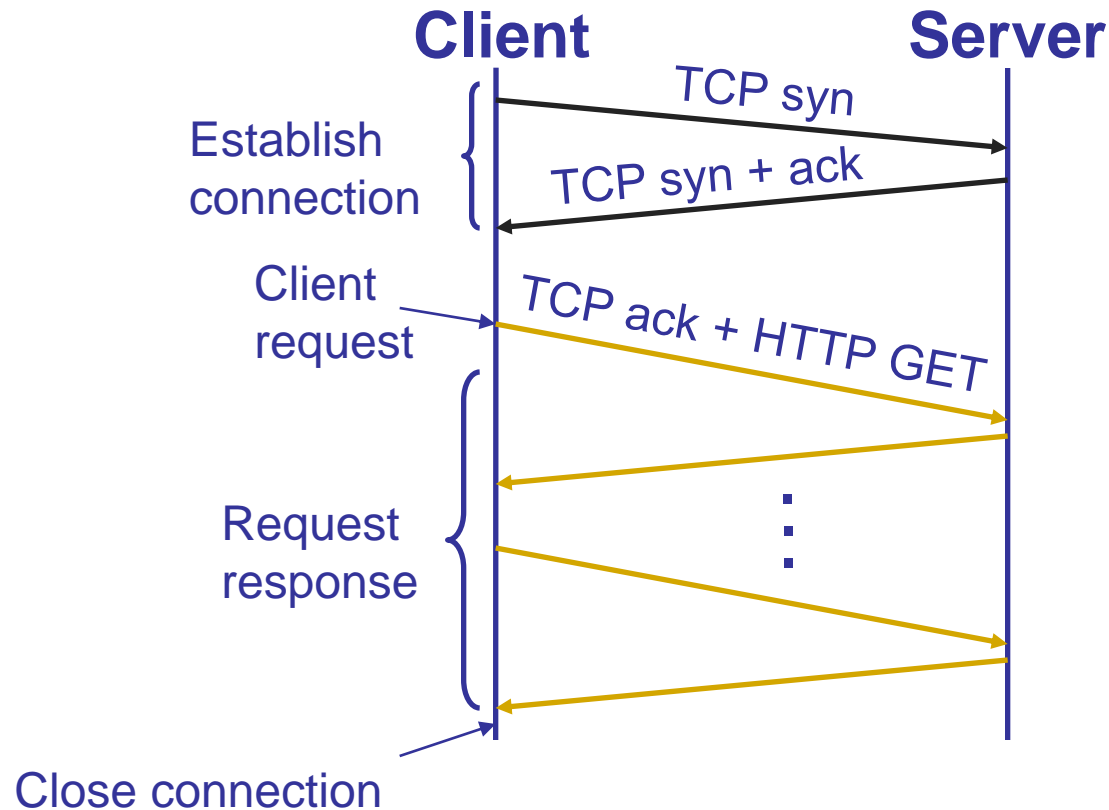
URL: Uniform Record Locator

- `protocol://host-name[:port]/directory-path/resource`
 - `protocol`: http, ftp, https, smtp, rtsp, *etc.*
 - `hostname`: DNS name, IP address
 - `port`: defaults to protocol's standard port
 - » *e.g.*, http: 80, https: 443
 - `directory path`: hierarchical, reflecting file system
 - `resource`: Identifies the desired resource

Hyper Text Transfer Protocol (HTTP)

- Client-server architecture
 - Server is “always on” and “well known”
 - Clients initiate contact to server
- Synchronous request/reply protocol
 - Runs over TCP, Port 80
- Stateless
- ASCII format
 - Before HTTP/2

Steps in HTTP request/response



Method types (HTTP 1.1)

- GET, HEAD
- POST
 - Send information (e.g., web forms)
- PUT
 - Uploads file in entity body to path specified in URL field
- DELETE
 - Deletes file specified in the URL field

Client-to-server communication

- HTTP Request Message

- Request line: method, resource, and protocol version

request line → **GET /somedir/page.html HTTP/1.1**

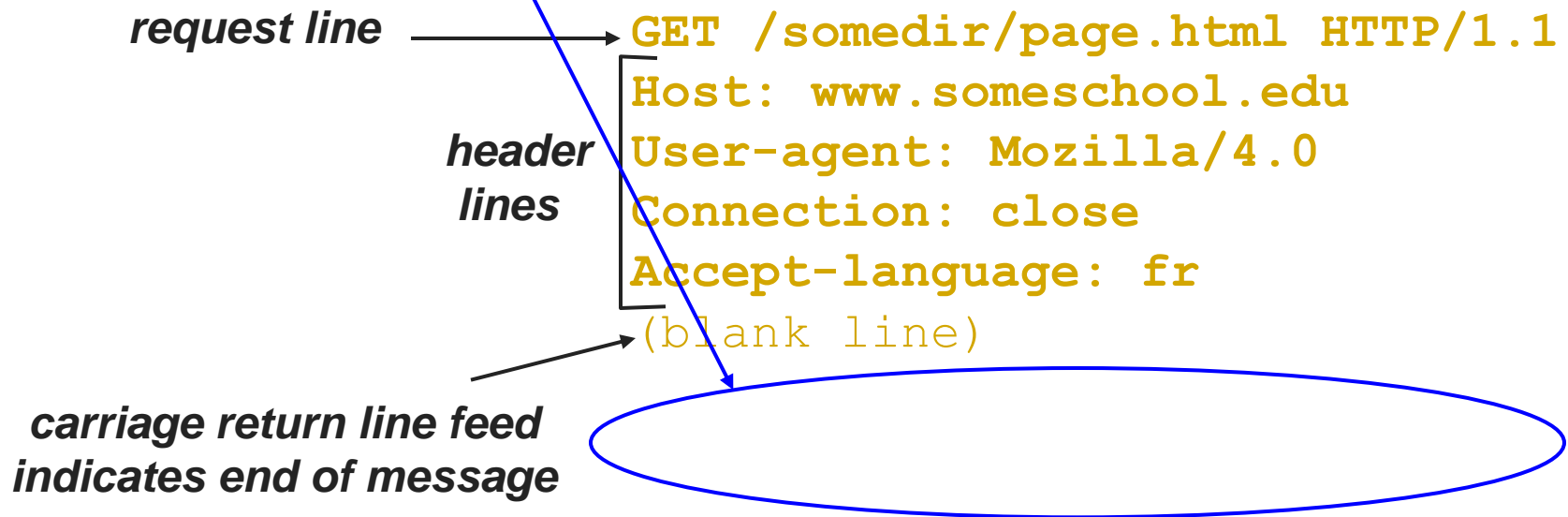
header lines →
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
(blank line)

*carriage return line feed
indicates end of message*

Client-to-server communication

■ HTTP Request Message

- Request line: method, resource, and protocol version
- Request headers: provide info or modify request
- Body: optional data (e.g., to “POST” data to server)



Server-to-client communication

■ HTTP Response Message

- Status line: protocol version, status code, status phrase
- Response headers: provide information
- Body: optional data

status line

(protocol, status code, status phrase)

header lines

data

e.g., requested HTML file

HTTP/1.1 200 OK

Connection close

Date: Thu, 06 Jan 2017 12:00:15 GMT

Server: Apache/1.3.0 (Unix)

Last-Modified: Mon, 22 Jun 2006 ...

Content-Length: 6821

Content-Type: text/html

(blank line)

data data data data data ...

HTTP is stateless

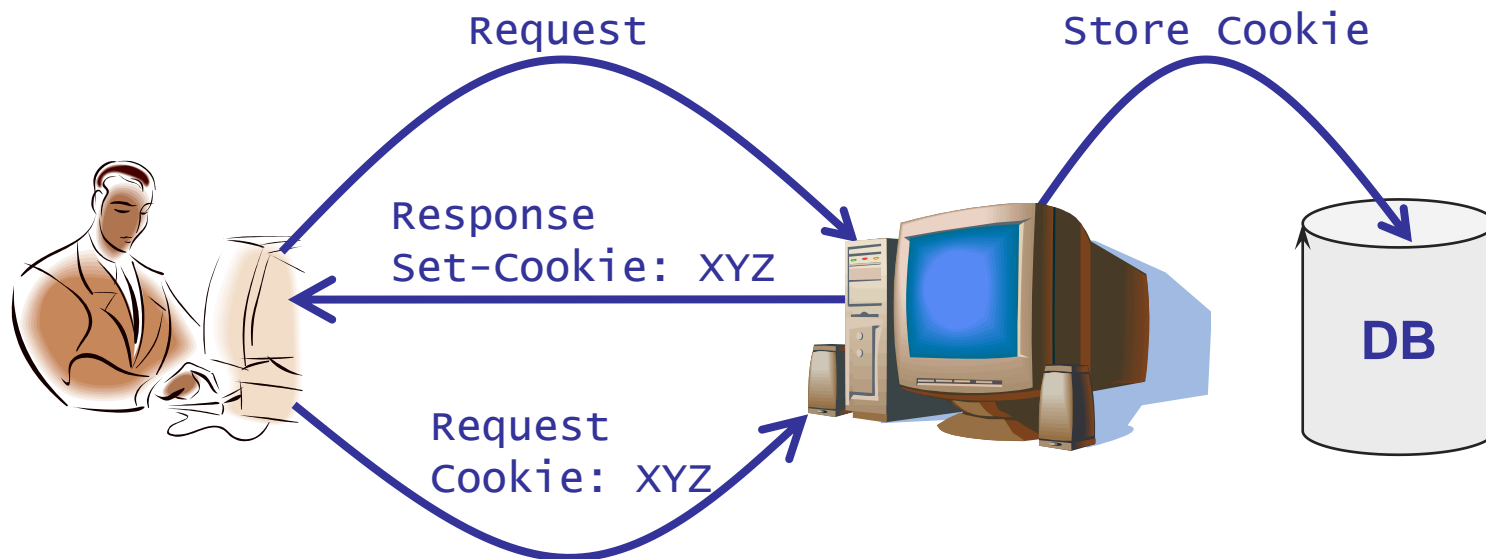
- Each request-response treated independently
 - Servers not required to retain state
- **Good:** Improves scalability on the server-side
 - Failure handling is easier
 - Can handle higher rate of requests
 - Order of requests doesn't matter
- **Bad:** Some applications need persistent state
 - Need to uniquely identify user or store temporary info
 - e.g., Shopping cart, user profiles, usage tracking, ...

Question

- How does a stateless protocol keep state?

State in a stateless protocol: Cookies

- Client-side state maintenance
 - Client stores small state on behalf of server
 - Client sends state in future requests to the server
- Can provide authentication



“Abuse” of cookies


- Excellent marketing opportunities and concerns for privacy
 - Cookies permit sites to learn a lot about you
 - You may unknowingly supply personal info to sites
 - Advertising companies tracks your preferences and viewing history across sites

Performance goals

- User
 - Fast downloads (not identical to low-latency communication!)
 - High availability
- Content provider
 - Happy users (hence, above)
 - Cost-effective infrastructure
- Network (secondary)
 - Avoid overload

Solutions?

Improve networking
protocols including HTTP,
TCP, etc.



- User
 - Fast downloads (not identical to low-latency communication!)
 - High availability
- Content provider
 - Happy users (hence, above)
 - Cost-effective infrastructure
- Network (secondary)
 - Avoid overload

Solutions?

Improve networking
protocols including HTTP,
TCP, etc.

- User
 - Fast downloads (not identical to low-latency communication!)
 - High availability
- Content provider
 - Happy users (hence, above)
 - Cost-effective infrastructure
- Network (secondary)
 - Avoid overload

Caching and replication

Solutions?

Improve networking protocols including HTTP, TCP, etc.

- User
 - Fast downloads (not identical to low-latency communication!)
 - High availability
- Content provider
 - Happy users (hence, above)
 - Cost-effective infrastructure
- Network (secondary)
 - Avoid overload

Caching and replication

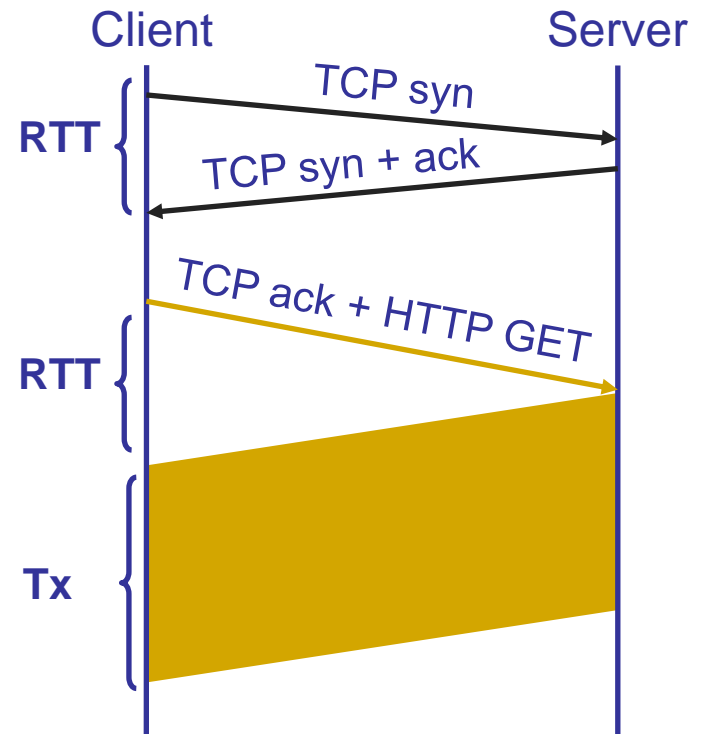
Exploit economies of scale; e.g., webhosting, CDNs, datacenters

HTTP performance

- Most Web pages have multiple objects
 - e.g., HTML file and a bunch of embedded images
- How do you retrieve those objects (naively)?
 - One item at a time
- New TCP connection per (small) object!

Object request response time

- RTT (round-trip time)
 - Time for a small packet to travel from client to server and back
- Response time
 - 1 RTT for TCP setup
 - 1 RTT for HTTP request and first few bytes
 - Transmission time
 - **Total** = 2RTT + Transmission Time

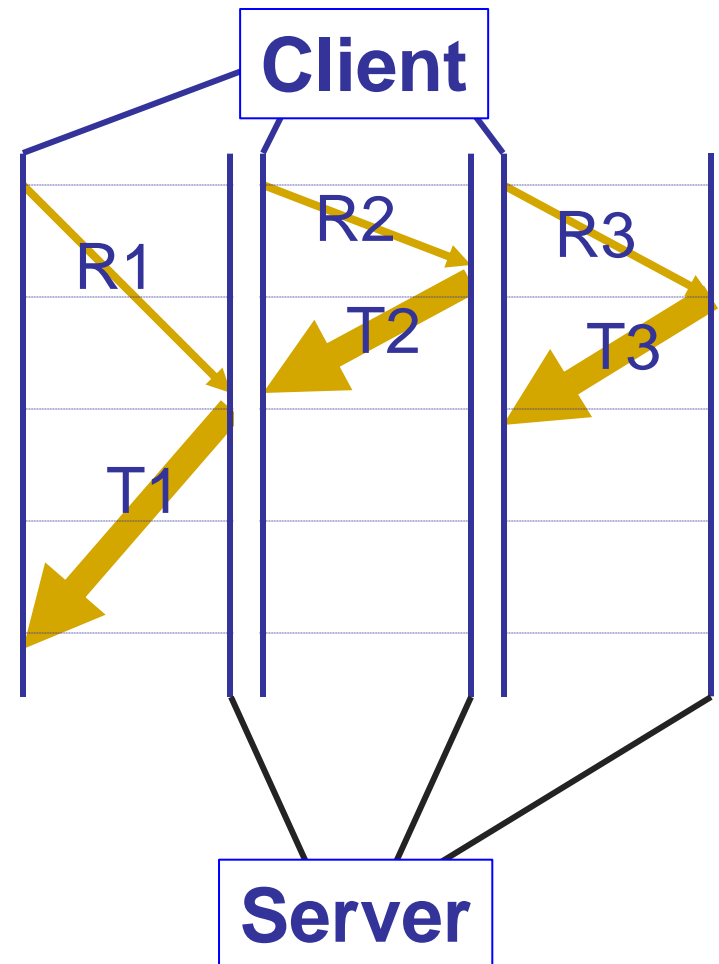


Non-persistent connections

- Default in HTTP/1.0
- $2RTT + \Delta$ for each object in the HTML file!
 - One more $2RTT + \Delta$ for the HTML file itself
- Doing the same thing over and over again
 - Inefficient

Concurrent requests and responses

- Use multiple connections in parallel
 - Does not necessarily maintain order of responses
-
- Client = 😊
 - Content provider = 😊
 - Network = 😞 Why?

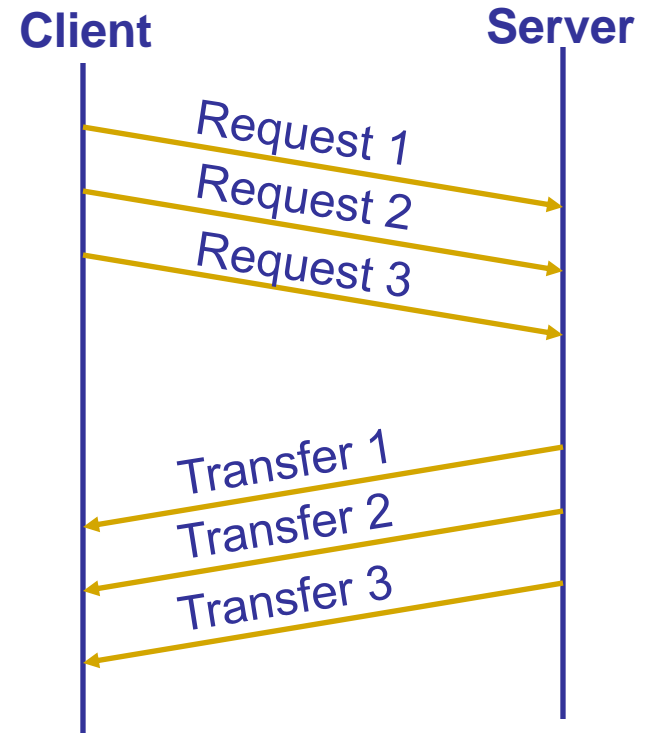


Persistent connections

- Maintain TCP connection across multiple requests
 - Including transfers subsequent to current page
 - Client or server can tear down connection
- Advantages
 - Avoid overhead of connection set-up and tear-down
 - Allow underlying layers (e.g., TCP) to learn about RTT and bandwidth characteristics
- Default in HTTP/1.1

Pipelined requests & responses

- Batch requests and responses to reduce the number of packets
- Multiple requests can be contained in one TCP segment



Scorecard: Getting n small objects

- Time dominated by latency
- One-at-a-time: $\sim 2n$ RTT
- m concurrent: $\sim 2\lceil n/m \rceil$ RTT
- Persistent: $\sim (n+1)$ RTT
- Pipelined: ~ 2 RTT
- Pipelined/Persistent: ~ 2 RTT first time, RTT later

Scorecard: Getting n large objects each of size F

- Time dominated by bandwidth
- One-at-a-time: $\sim nF/B$
- m concurrent: $\sim [n/m] F/B$
 - Assuming shared with large population of users and each TCP connection gets the same bandwidth
- Pipelined and/or persistent: $\sim nF/B$
 - The only thing that helps is getting more bandwidth

Caching

- Why does caching work?
 - Exploits locality of reference
- How well does caching work?
 - Very well, up to a limit
 - Large overlap in content
 - But many unique requests
 - » A universal story!
 - » Effectiveness of caching grows logarithmically with size

Caching: How

- Modifier to GET requests:
 - **If-modified-since** – returns “not modified” if resource not modified since specified time

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
If-modified-since: Wed, 18 Jan 2017 10:25:50 GMT
(blank line)
```

Caching: How

- Modifier to GET requests:
 - `If-modified-since` – returns “not modified” if resource not modified since specified time
- Client specifies “`if-modified-since`” time in request
- Server compares this against “last modified” time of resource
- Server returns “Not Modified” if resource has not changed
- or a “OK” with the latest version otherwise

Caching: How

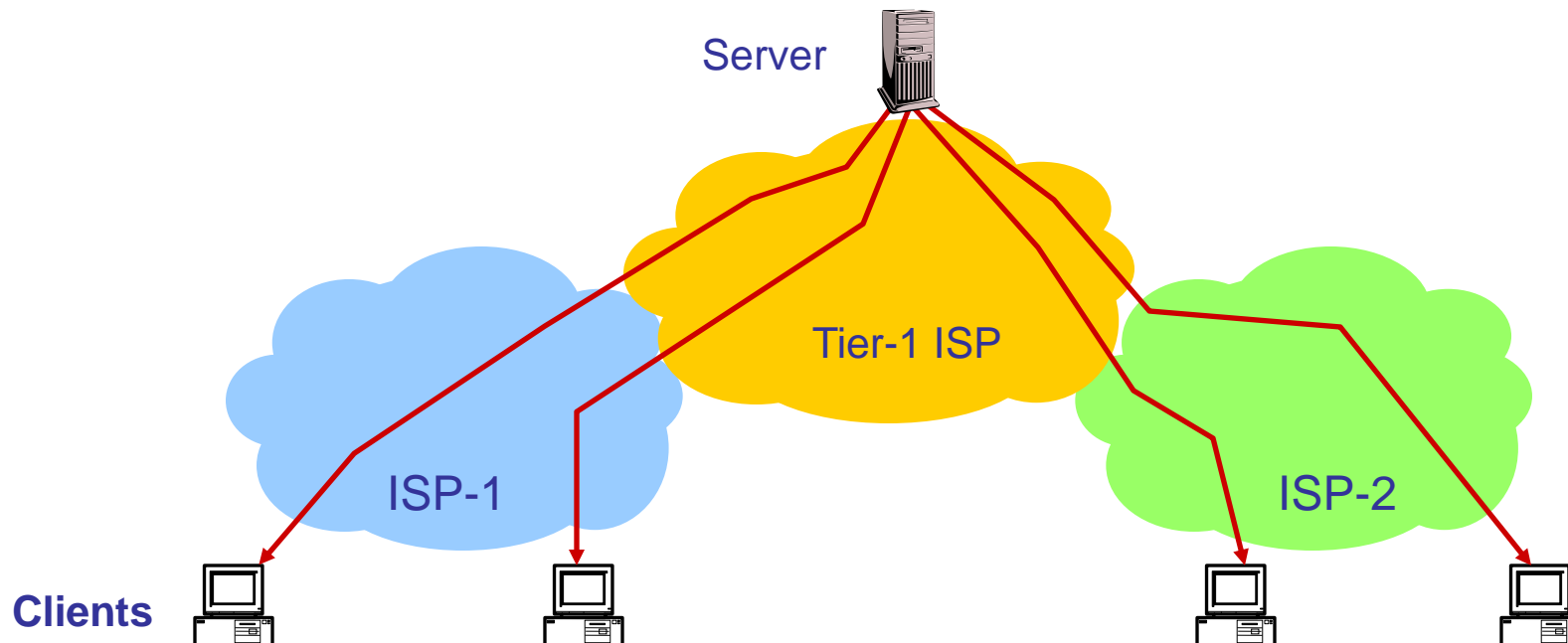
- Modifier to GET requests:
 - **If-modified-since** – returns “not modified” if resource not modified since specified time
- Response header:
 - **Expires** – how long it's safe to cache the resource
 - **No-cache** – ignore all caches; always get resource directly from server

Caching: Where?

- Options
 - Client (browser)
 - Forward proxies
 - Reverse proxies
 - Content Distribution Network

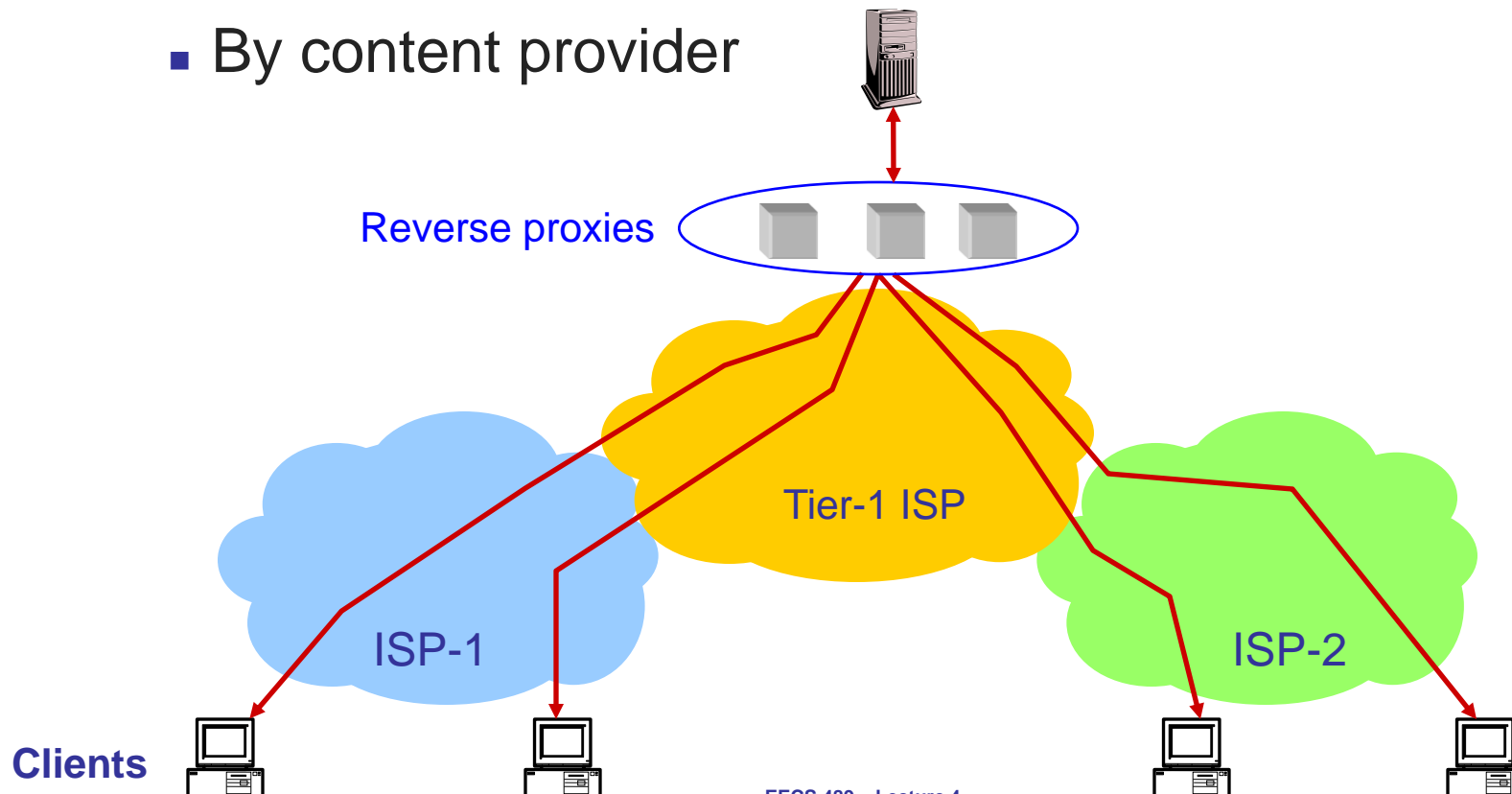
Caching: Where?

- Many clients transfer same information
 - Generate unnecessary server and network load
 - Clients experience unnecessary latency



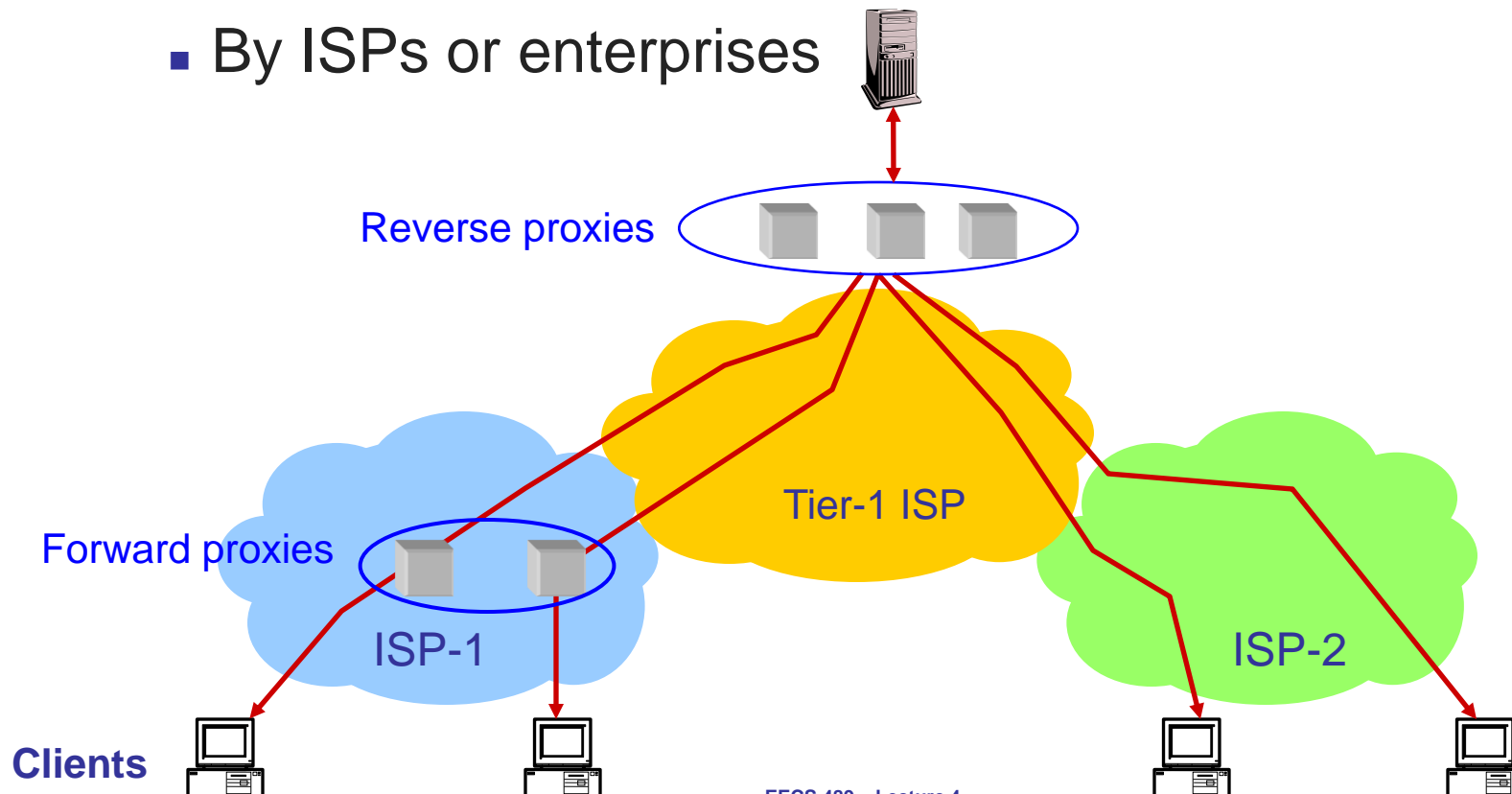
Caching with Reverse Proxies

- Cache documents close to server
 - Decrease server load
 - By content provider



Caching with Forward Proxies

- Cache documents close to clients
 - Reduce network traffic and decrease latency
 - By ISPs or enterprises



Summary

- HTTP/1.1
 - Text-based protocol
 - Being replaced by binary HTTP/2 protocol
- Many ways to improve performance
 - Pipelining and batching
 - Caching in proxies and CDNs
 - Datacenters