Computer Networks

Wenzhong Li, Chen Tian

Nanjing University

Material with thanks to James F. Kurose, Mosharaf Chowdhury, and other colleagues.

Internet Applications

- WWW and HTTP
- Content Distribution Networks (CDNs)

WWW and HTTP

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

蒂姆·伯纳斯-李爵士
Sir Tim Berners-Lee

出生 1955年6月8日（61歲）[1]
英格兰伦敦
机构 万维网联盟
南安普敦大学
Plessey
麻省理工学院
知名于 发明万维网
麻省理工学院计算机科学及人工智能实验室创办主席

2016 Turing Award

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 Resource Locator

- A unique identifier for an object on WWW
- URL format

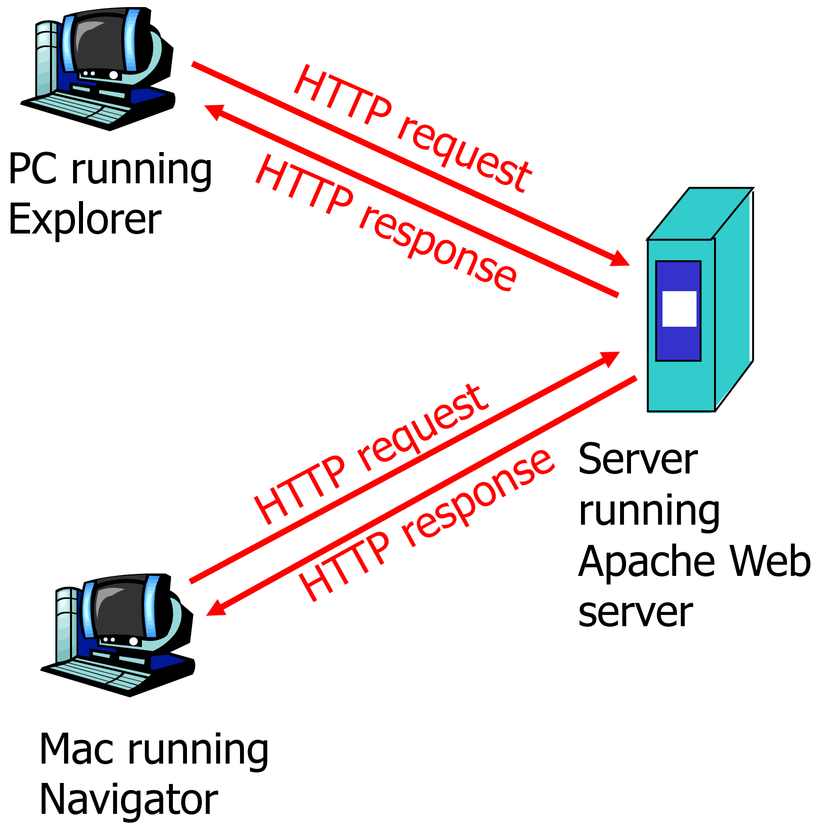
  <protocol>://<host>:<port>/<path>?query_string

  - Protocol: method for transmission or interpretation of the object, e.g. http, ftp, Gopher
  - Host: DNS name or IP address of the host where object resides
  - Path: pathname of the file that contains the object
  - Query_string: name/value pairs sent to app on the server
- An example

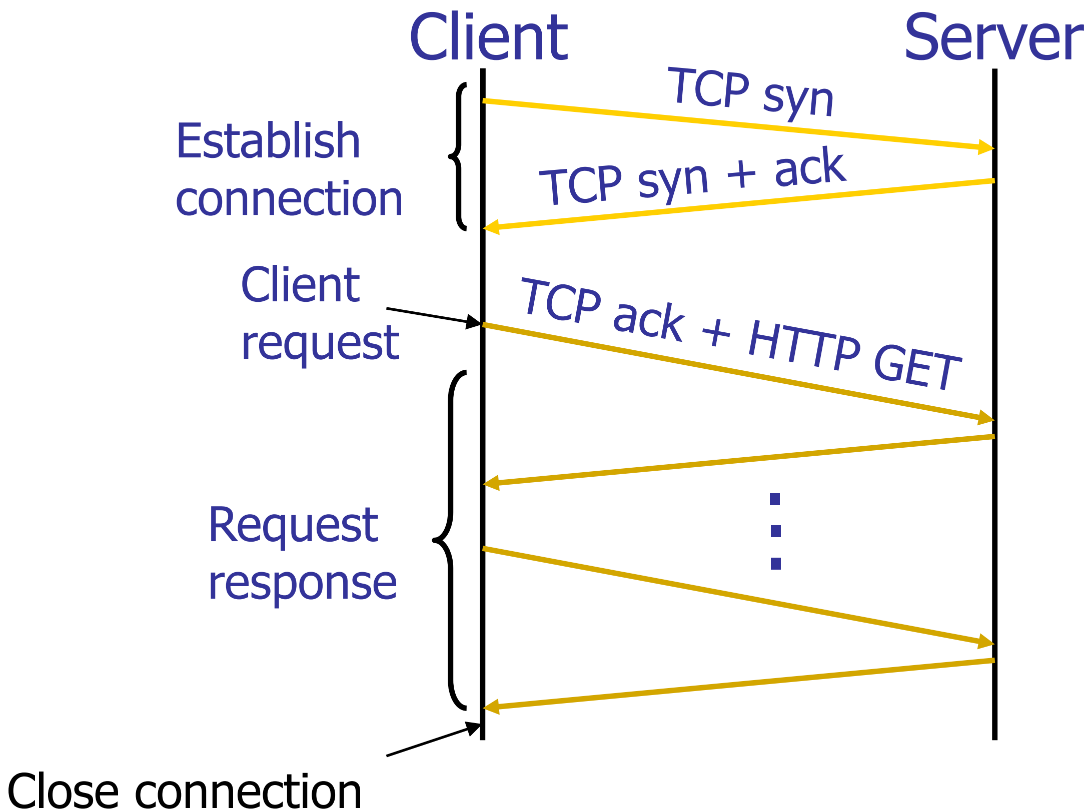
  http://www.nju.edu.cn:8080/somedir/page.htm

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

header lines

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
(blank line)
```

carriage return line feed indicates end of message

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

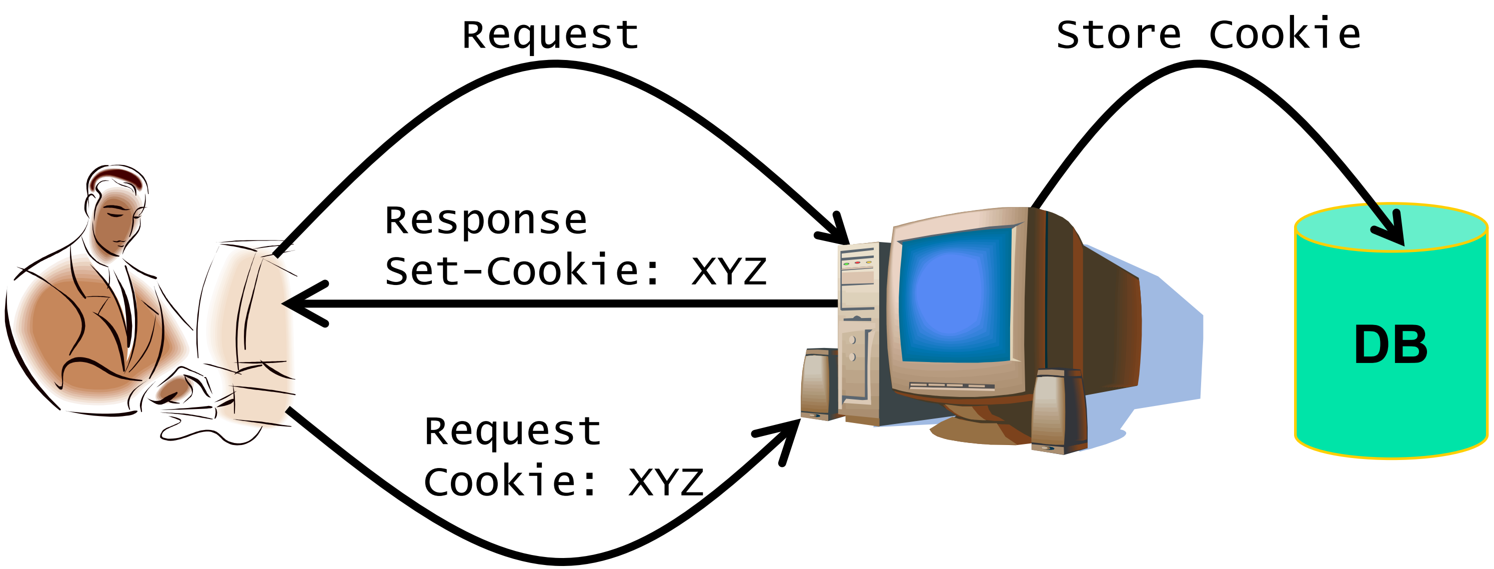
- Each request-response treated independently
  - Servers not required to retain state
- Good: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn't matter
- Bad: Some applications need persistent state
  - Need to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

Question

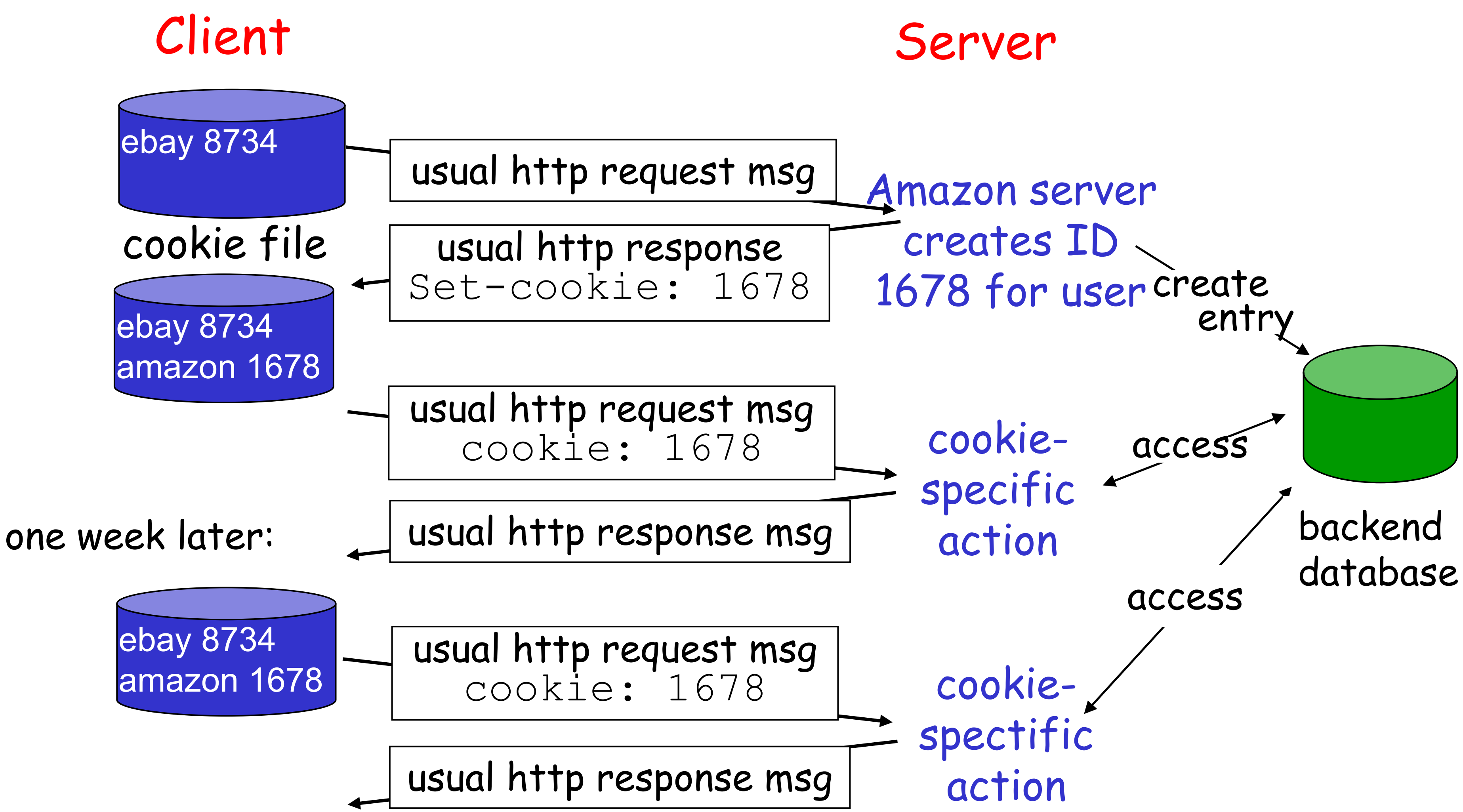
- How does a stateless protocol keep state?

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



A Cookies Example



Application of Cookies

What cookies can bring

- Authorization
- Shopping carts
- Recommendations
- User session state (Web Email)

aside

Cookies and privacy

- Cookies permit servers to learn a lot about user
- User may supply name and Email to servers
- Search engines may use cookies to obtain info across sites
- Hacked browser may do bad things with cookies

Web 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 … atency communication!)
  - High availability
- Content provider
  - Happy users (hence, above)
  - Cost-effective infrastructure
- Network (secondary)
  - Avoid overload

Solutions?

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

Improve networking protocols including HTTP, TCP, etc.

Caching and replication

Solutions?

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

Improve networking protocols including HTTP, TCP, etc.

Caching and replication

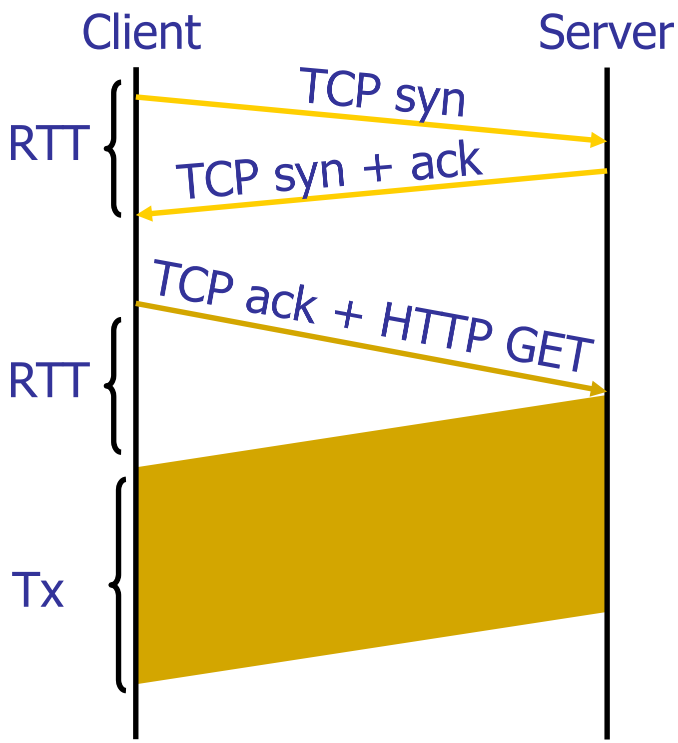
Exploit economies of scale; e.g., webhosting, CDNs, datacenters

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

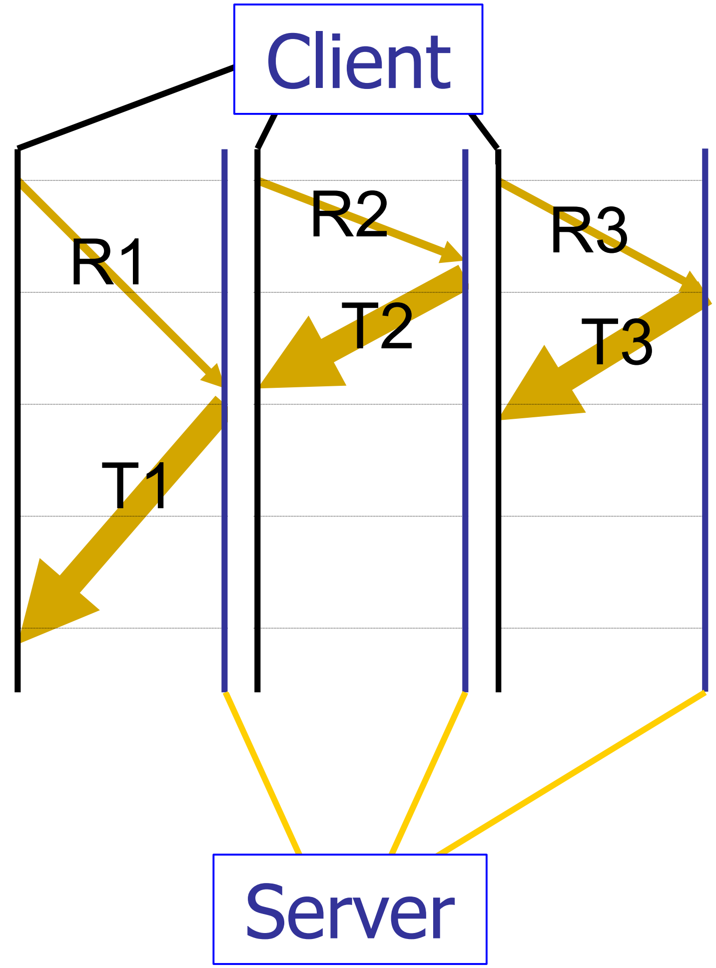


Non-persistent connections

- Default in HTTP/1.0
- 2RTT+△ for each object in the HTML file!
  - One more 2RTT+△ 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?

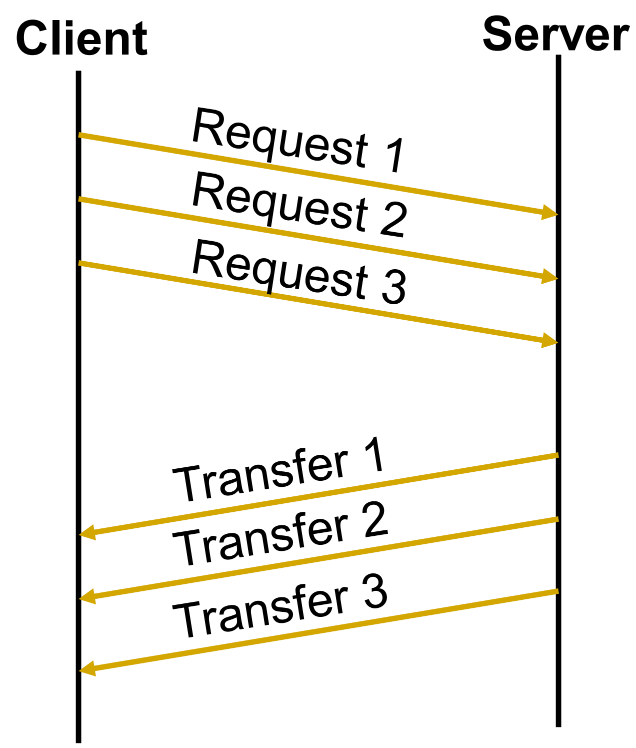
Persistent connections

- 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: ~2n RTT
- m concurrent: ~2[n/m] RTT
- Persistent: ~ (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: ~ nF/B
- m concurrent: ~ [n/m] F/B
  - Assuming shared with large population of users and each TCP connection gets the same bandwidth
- Pipelined and/or persistent: ~ 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

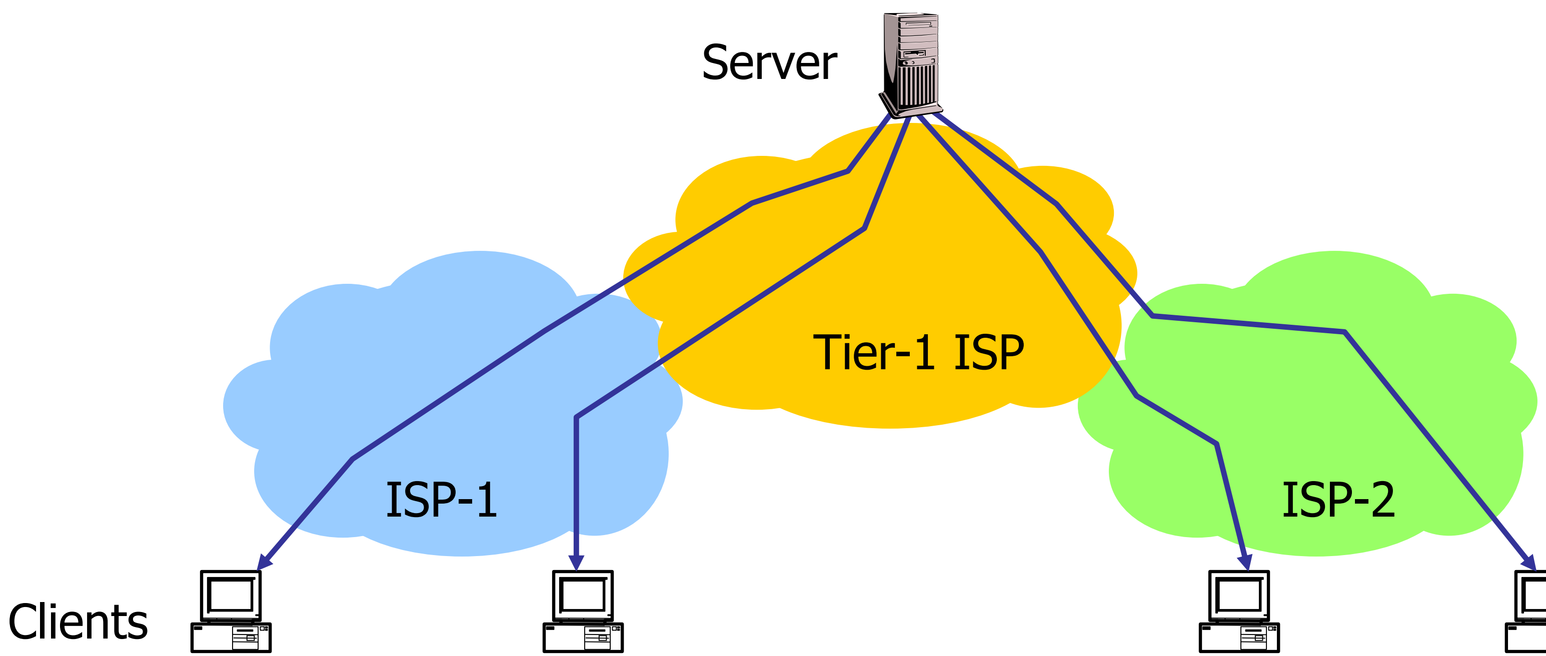
- Modifier to GET requests:
  - `If-modified-since` – returns "not modified" if resource not modified since specified time
- Response header:
  - `Expires` – how long it's safe to cache the resource
  - `No-cache` – ignore all caches; always get resource directly from server

Caching: Where?

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

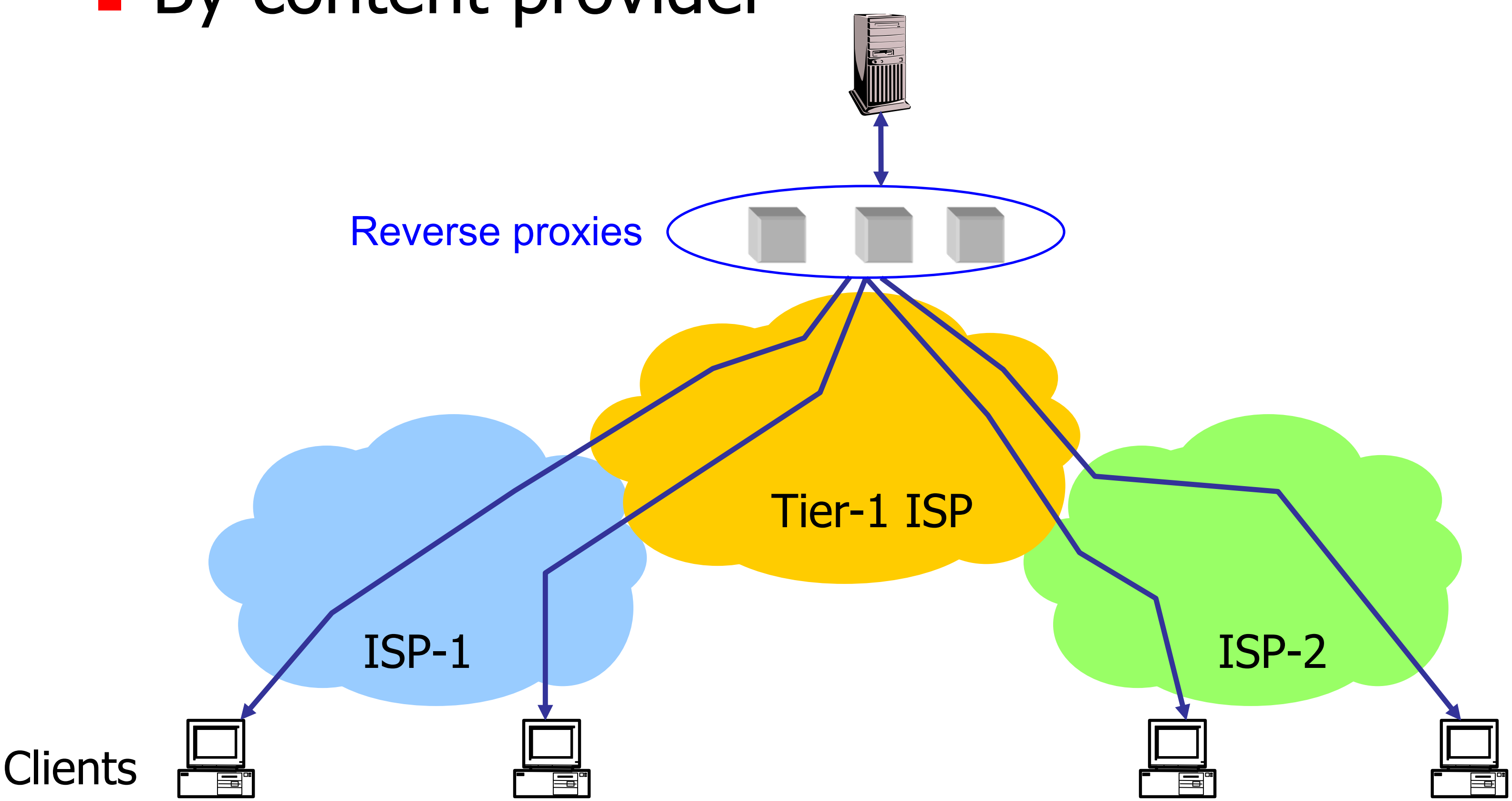
Caching: Where?

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



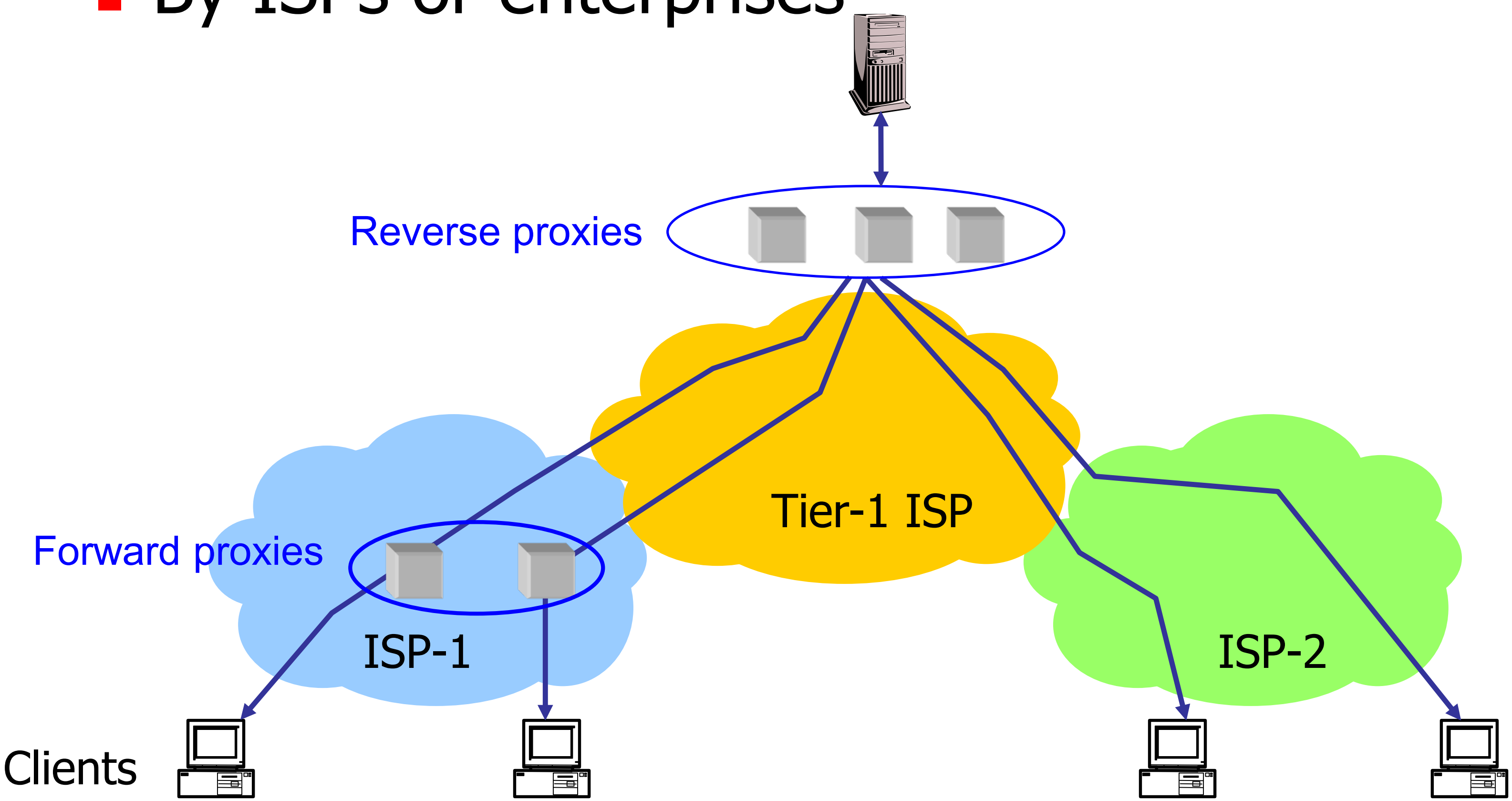
Caching with Reverse Proxies

- 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

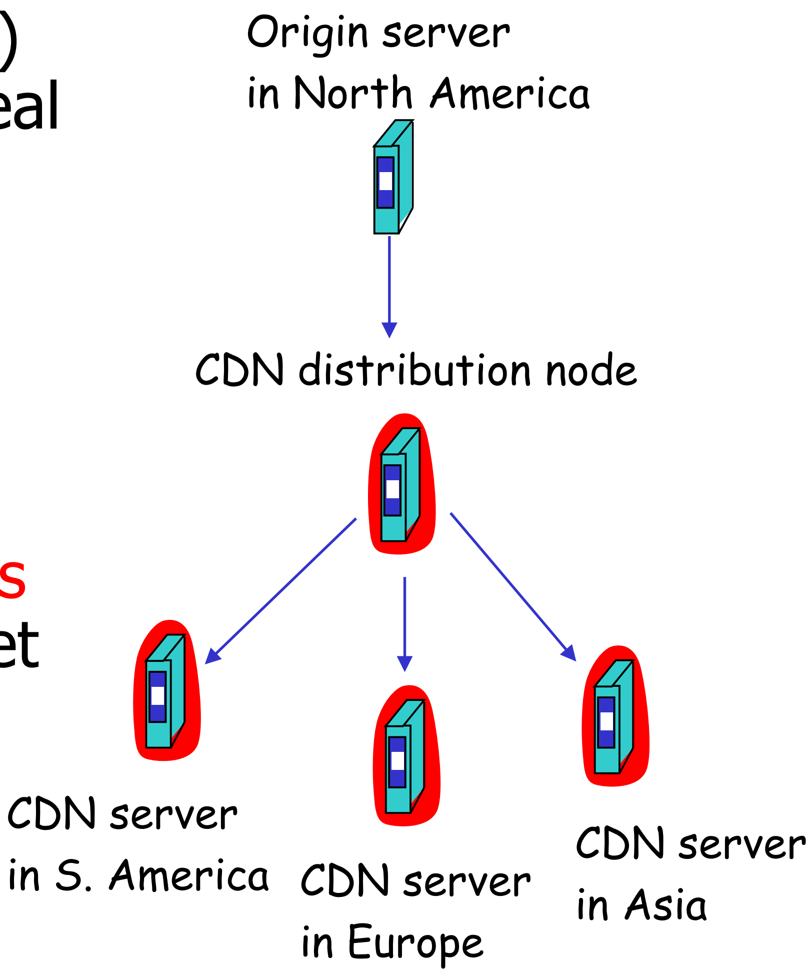


- HTTP/1.1
  - Text-based protocol
  - Being replaced by binary HTTP/2 protocol
- Many ways to improve performance
  - Pipelining and batching
  - Caching in proxies and CDNs
  - Datacenters

Content Distribution Networks (CDNs)

Content Distribution Networks (CDNs)

- Challenge
  - Stream large files (e.g. video) from single origin server in real time
  - Protect origin server from DDOS attacks
- Solution
  - Replicate content at hundreds of servers throughout Internet
  - CDN distribution node coordinate the content distribution
  - Placing content close to user



Content Replication

- Content provider (origin server) is CDN customer

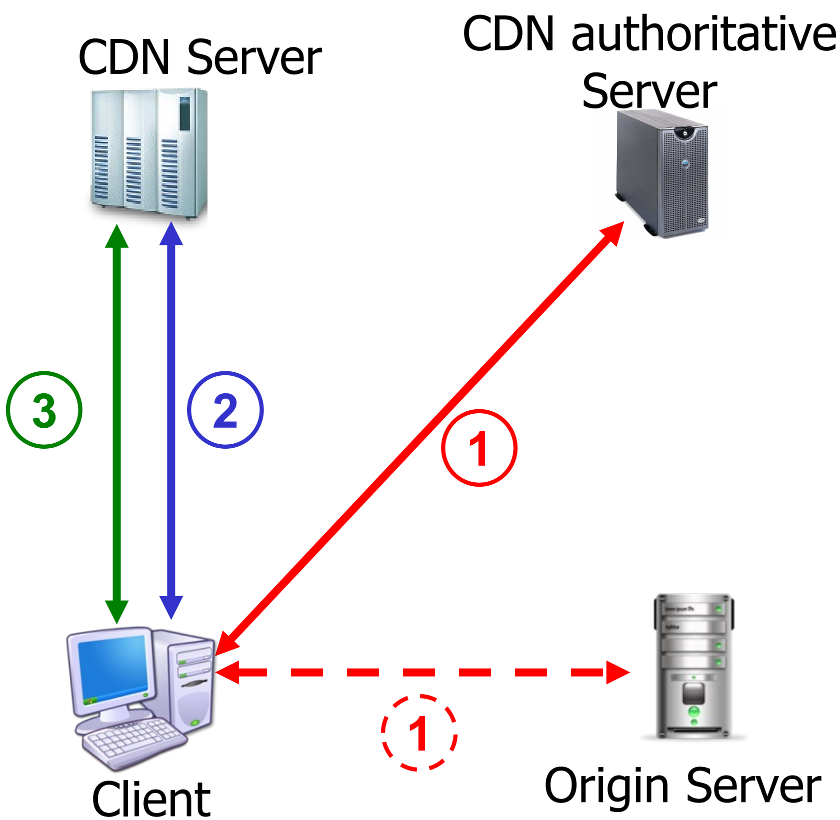
- CDN replicates customers' content in CDN servers
- When provider updates content, CDN updates its servers

- Use authoritative DNS server to redirect requests

Supporting Techniques

- DNS
  - One name maps onto many addresses
- Routing
  - Content-based routing (to nearest CDN server)
- URL Rewriting
  - Replaces "http://www.sina.com/sports/tennis.mov" with "http://www.cdn.com/www.sina.com/sports/tennis.mov"
- Redirection strategy
  - Load balancing, network delay, cache/content locality

CDN Operation



1′ URL rewriting – get authoritative server

1. Get near CDN server IP address

2. Warm up CDN cache

3. Retrieve pages/media from CDN Server

Redirection

- CDN creates a "map", indicating distances from leaf ISPs and CDN servers
- When query arrives at authoritative DNS server
  - Server determines ISP from which query originates
  - Uses "map" to determine best CDN server
- CDN servers create an application-layer overlay network

Summary of Web App

- Conceptual, implementation aspects of network application protocols
  - Client-Server vs. Peer-to-Peer
  - Data presentation formatting

- Examining popular application-level protocols
  - DNS, SNMP / MIB
  - HTTP, FTP, SMTP / POP3 / MIME
  - Content distribution networks (CDNs)

Summary

- Internet Applications
  - WWW and HTTP
  - Content Distribution Networks (CDNs)