

---

---

# Application Layer

CS5700 Fall 2019

---

---

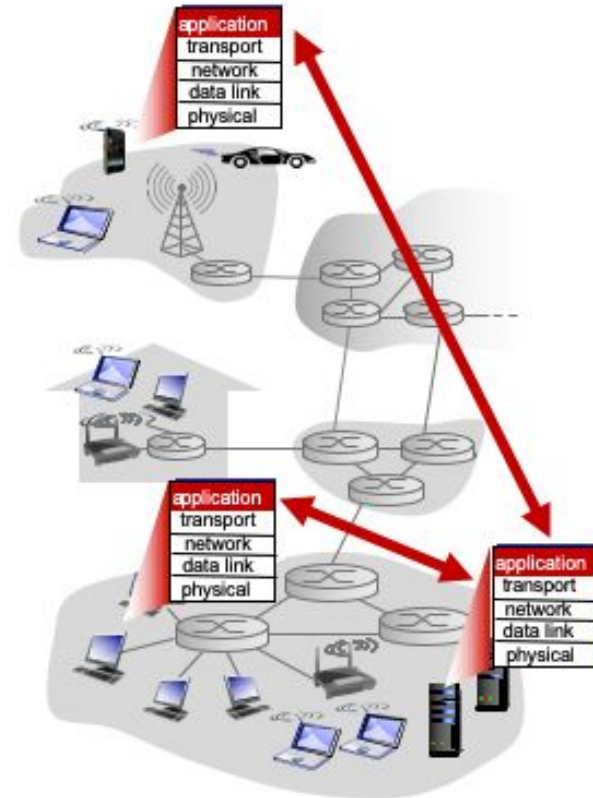
# Agenda

- Principles of network applications
- DNS
- Web and HTTP
- CDN
- SMTP
- DHCP

# Principles of network applications

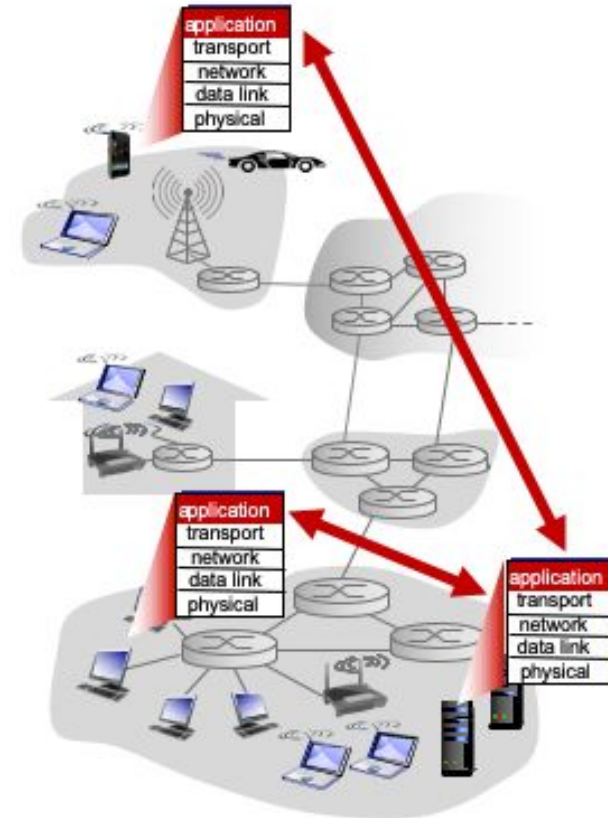
# Principle - intelligence at the edge

- Internet does not provide services. It only provides communication.
- Application programs provide all services.



# Principle - intelligence at the edge

- Write application programs that
  - Run on hosts
  - Communicate over network
- No need to change network core
  - Network core devices do not run user applications



# Principle - intelligence at the edge

- Web
- Email
- Network games
- Streaming videos (Youtube, Netflix, Hulu, etc.)
- Realtime video conferencing
- Social networking
- ...
- All require no change at the network core

# Why is it a good principle?



# How long does it take us to adopt IPv6?

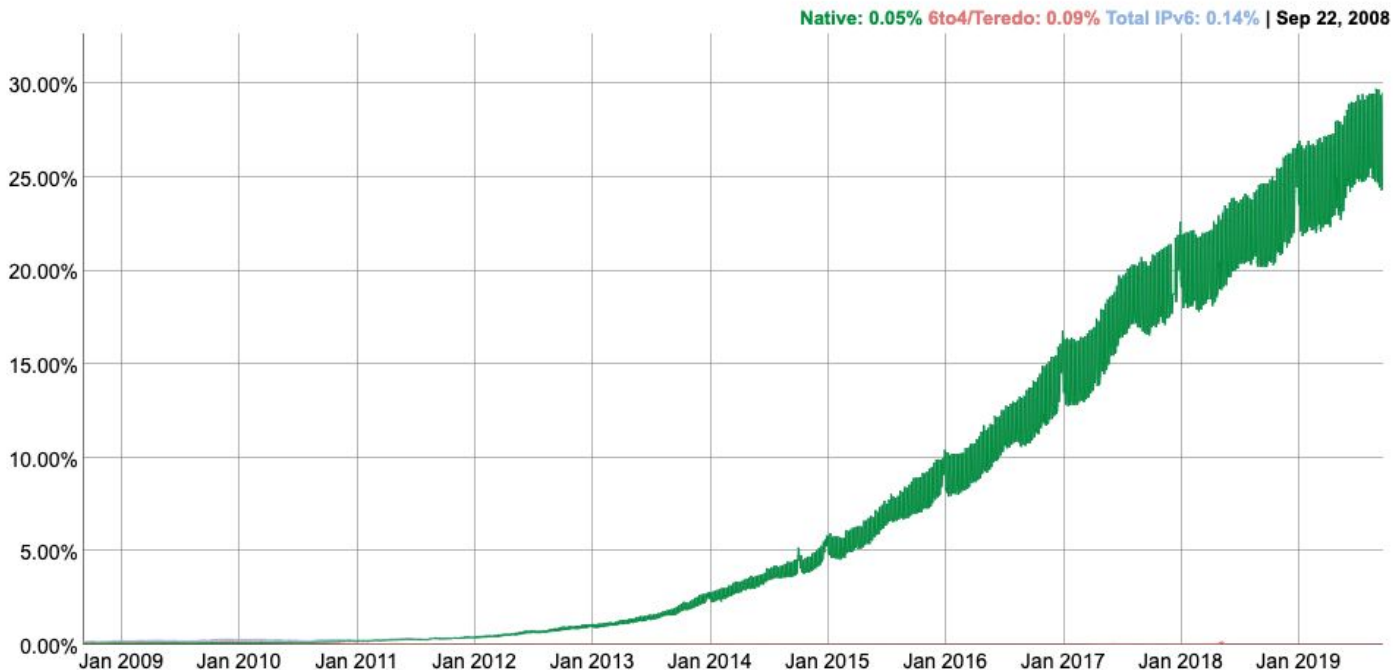




# How long does it take us to adopt IPv6?

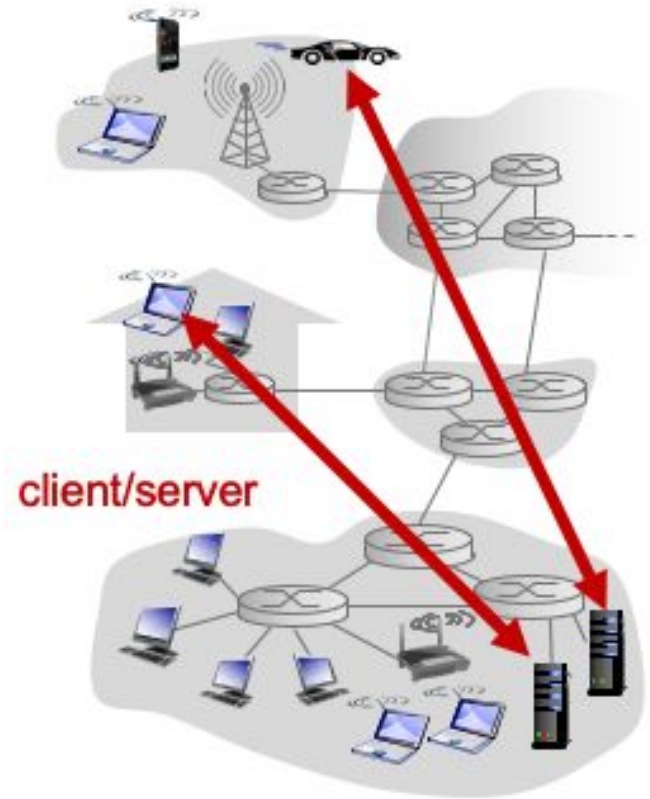
## IPv6 Adoption

We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.



# Client-server architecture

- Server
  - Always-on host
  - Permanent IP address
  - Data center for scaling
- Client
  - Communicate with server
  - May have dynamic IP address
  - Do not communicate directly with each other



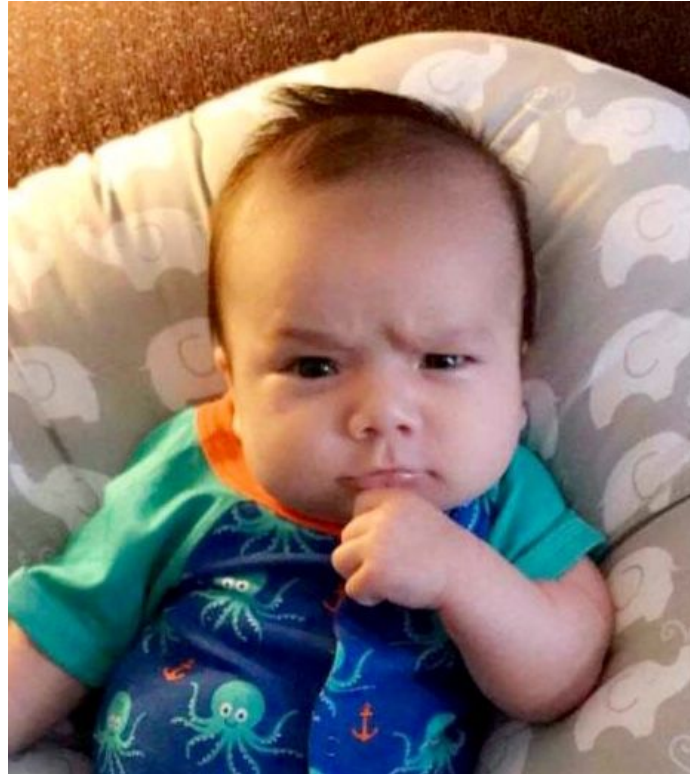
# Transport layer service model - TCP

- Reliable data transfer
  - No loss, in-order
- Flow control: sender won't overwhelm receiver
- Congestion control: throttle sender when network is overloaded
- Connection oriented: setup required between client and server

# Transport layer service model - UDP

- Unreliable data transfer
  - Loss, out-of-order, duplicate
- That's it!

**Any service you'd like transport layer to have?**



# Other important services

- Timing (aka bounded latency)
  - E.g. Internet telephony, interactive games
- Throughput
  - E.g. multimedia
- Security
  - Encryption, data integrity, etc.
- ...
- None of the above is provided in transport layer! :(

# DNS

# DNS - domain name system

- Important piece of Internet infrastructure
- Runs at the application layer
- Translate human-readable names into IP addresses
- Distributed database
  - Centralized DNS doesn't scale!

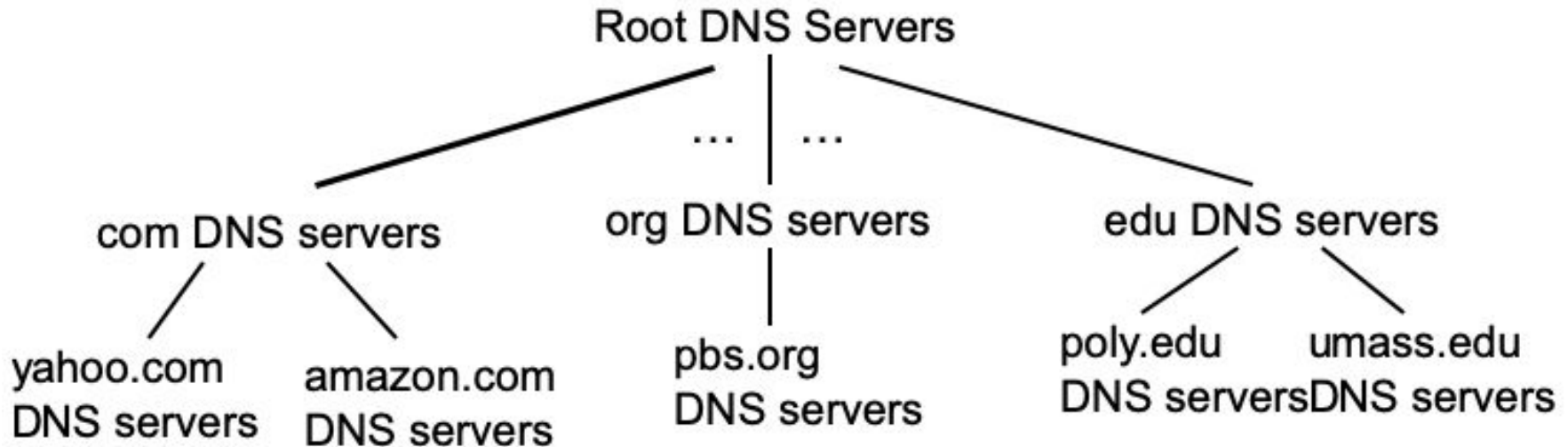


# DNS

- Names are hierarchical
- Each name divided into segments by period char
  - Read as “dot”
- Most significant segment is on the right
- Rightmost segment known as a top-level domain (TLD)
- E.g. neu.edu

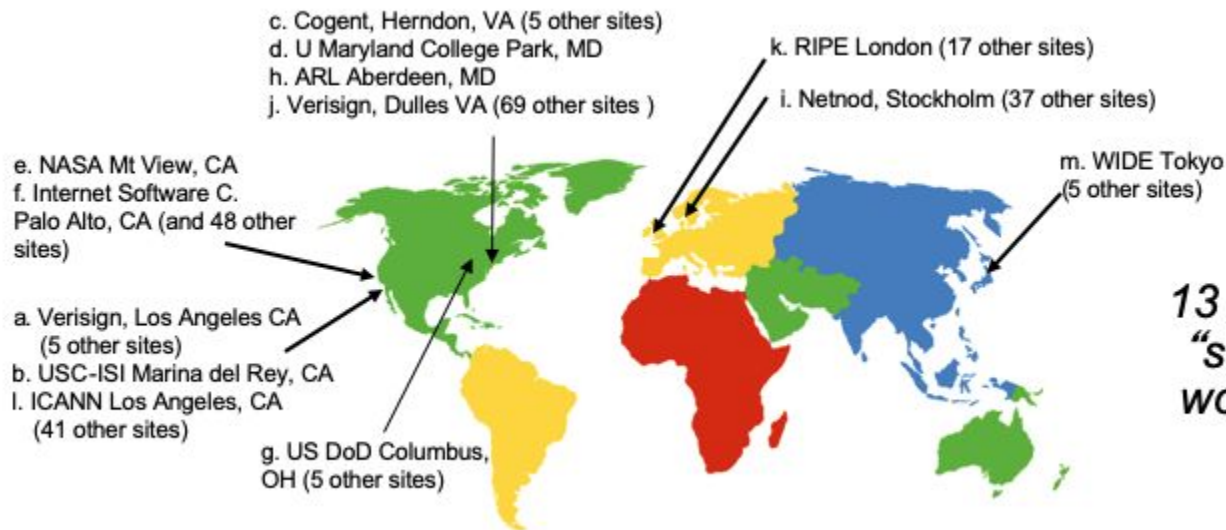
# DNS - hierarchical database

- How do you get IP address for `www.neu.edu`?



# DNS - root name servers

- 13 logical root name servers. ([a-m].root-servers.net)
- Provide which TLD name server to ask next



*13 root name  
“servers”  
worldwide*

# DNS - TLD name servers

- Responsible for com, org, net, edu, ..., and all top-level country domains
- Provide which authoritative name server to ask next

# DNS - authoritative name servers

- Organization's own name servers
- Provide authoritative hostname to IP mappings for organization's named hosts

## Summary so far...

- How many DNS queries you need?
  - 1 for root name server
  - 1 for TLD name server
  - 1 for authoritative name server
- Is there any issue?

**Too slow!!**



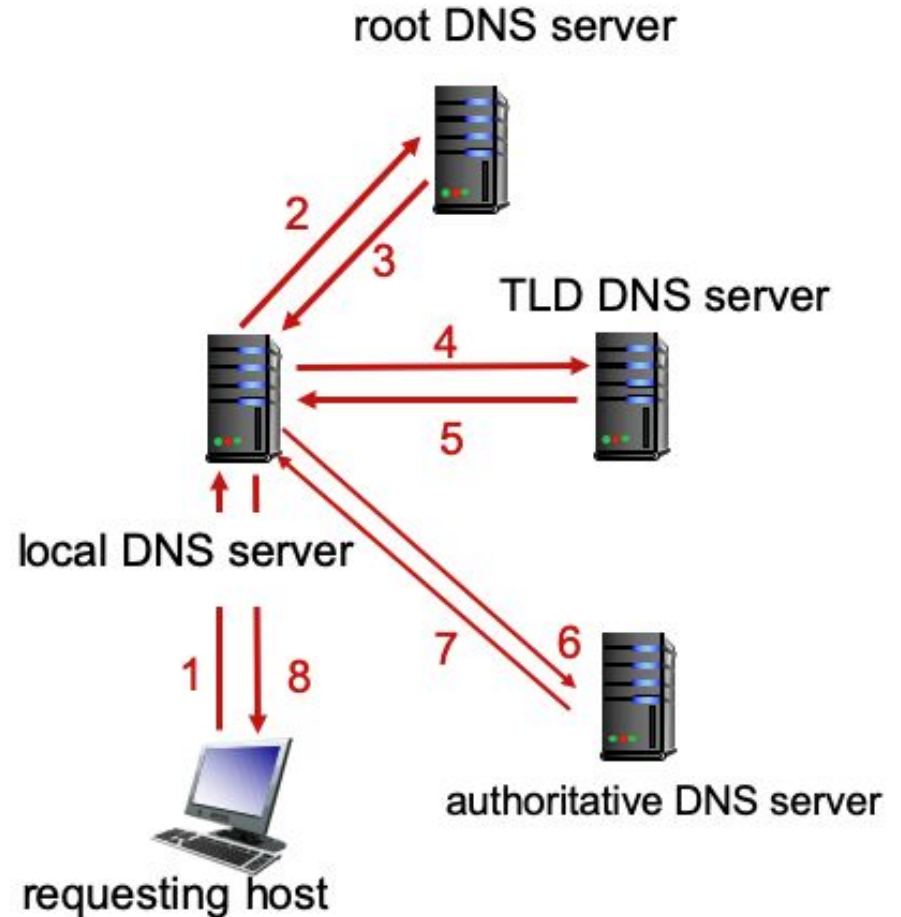
# DNS - local name server

- Does not belong to hierarchy
- Each ISP (residential ISP, company, university) has one
- When host makes DNS query, query is sent to its local name server
  - Acts as proxy, forwards query into hierarchy
  - Has local cache of recent name-to-address map



# Put all together

- Can you see this is more efficient?



# DNS - caching

- Cache entries timeout after TTL
  - What is reasonable TTL? Who decide?
- TLD name servers typically cached in local name servers
  - Thus root name servers not often visited
- Cached entries may be out-of-date

# DNS records

**DNS:** distributed db storing resource records (RR)

RR format: (name, value, type, ttl)

## type=A

- **name** is hostname
- **value** is IP address

## type=NS

- **name** is domain (e.g., foo.com)
- **value** is hostname of authoritative name server for this domain

## type=CNAME

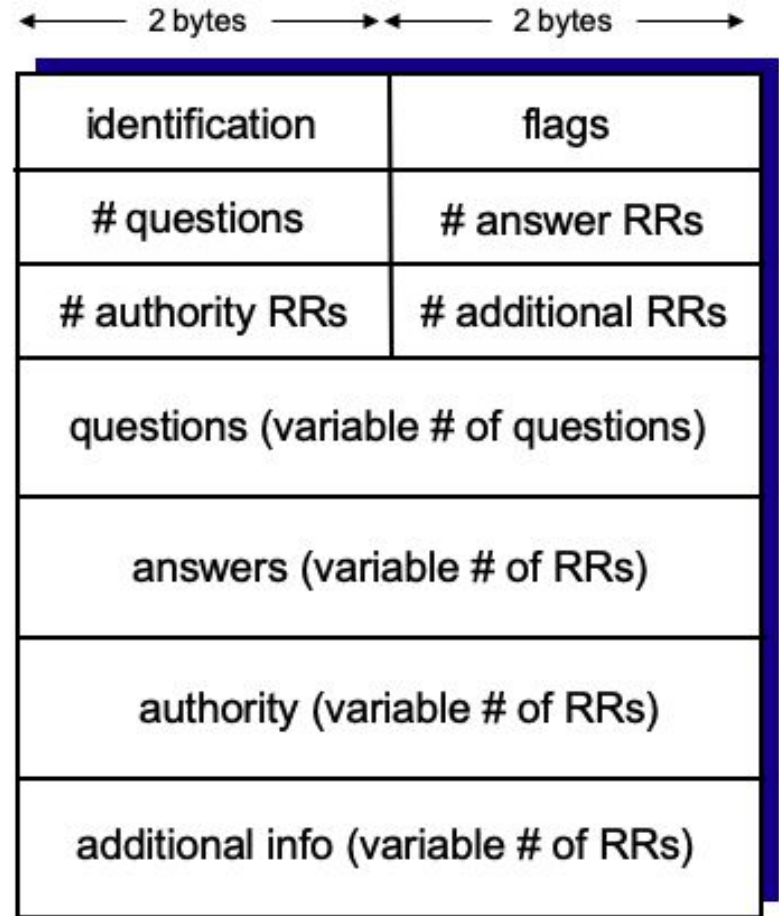
- **name** is alias name for some “canonical” (the real) name
- **www.ibm.com** is really **servereast.backup2.ibm.com**
- **value** is canonical name

## type=MX

- **value** is name of mailserver associated with **name**

# DNS - message format

- Both query and reply messages have the same format
- Flags:
  - Query or reply
  - Recursion desired
  - Recursion available
  - Reply is authoritative



# DNS

- Is DNS using TCP or UDP as transport layer protocol?



# Demo wireshark



# Inserting records into DNS

- New startup “Network Utopia”
- Register name networkutopia.com at DNS registrar
  - Provide names, IP addresses of authoritative name server (primary and secondary)
  - Registrar inserts two RRs into .com TLD name server (networkutopia.com, dns1.networkutopia.com, NS) (dns1.networkutopia.com, 10.1.1.1, A)
- Create type A record for [www.networkutopia.com](http://www.networkutopia.com) in authoritative name server.

# Web and HTTP



# What's in a web page?

- Web page consists of objects
- Object can be HTML file, JPEG image, JS file, etc.
- Web page consists of base HTML file which includes several referenced objects
- Each object is addressable by a URL

`www.someschool.edu/someDept/pic.gif`

host name

path name

# HTTP overview

- HyperText Transfer Protocol
- Client: browser requests, receives, and display Web objects
- Server: Web server sends objects in response to requests



# HTTP overview

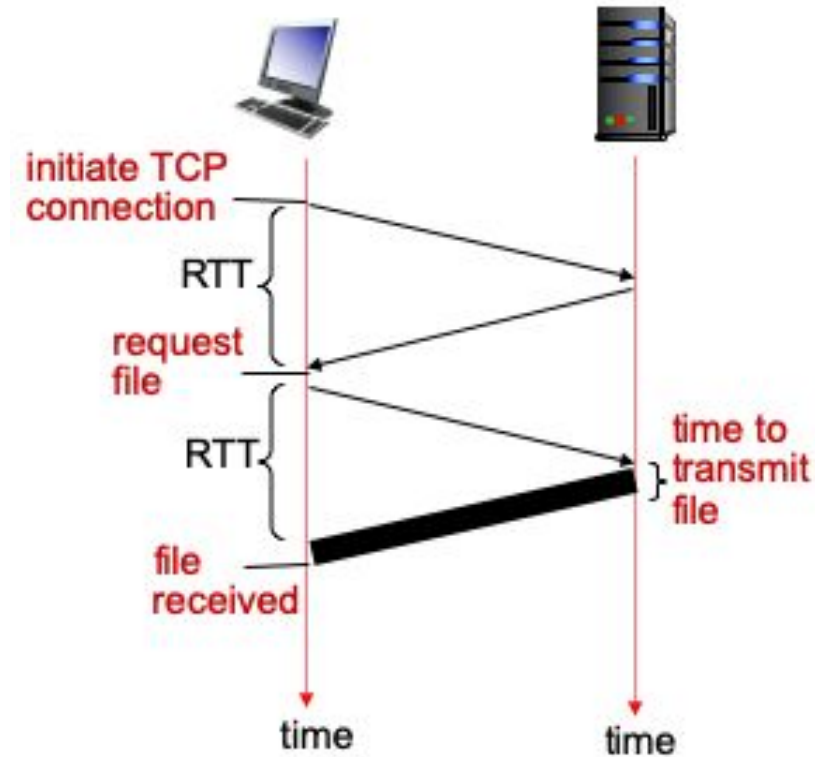
- Uses TCP as transport layer protocol
- Server uses well-known port 80
- HTTP is “stateless”
  - Server maintains no information about past client requests

# HTTP connections

- Non-persistent HTTP
  - At most one object sent over a TCP connection
    - Connection then closed
  - Downloading multiple objects required multiple connections
- Persistent HTTP
  - Multiple objects can be sent over single TCP connection

# HTTP response time

- One RTT to initiate TCP connection
- One RTT for HTTP request and first few bytes of HTTP response to return
- File transmission time



# What's the response time using persistent HTTP?



# HTTP request message

- ASCII (human-readable format)

The diagram illustrates the structure of an HTTP request message in ASCII format. It consists of a request line followed by header lines, each terminated by a carriage return and line feed character sequence (\r\n).

**request line (GET, POST, HEAD commands)**

**header lines**

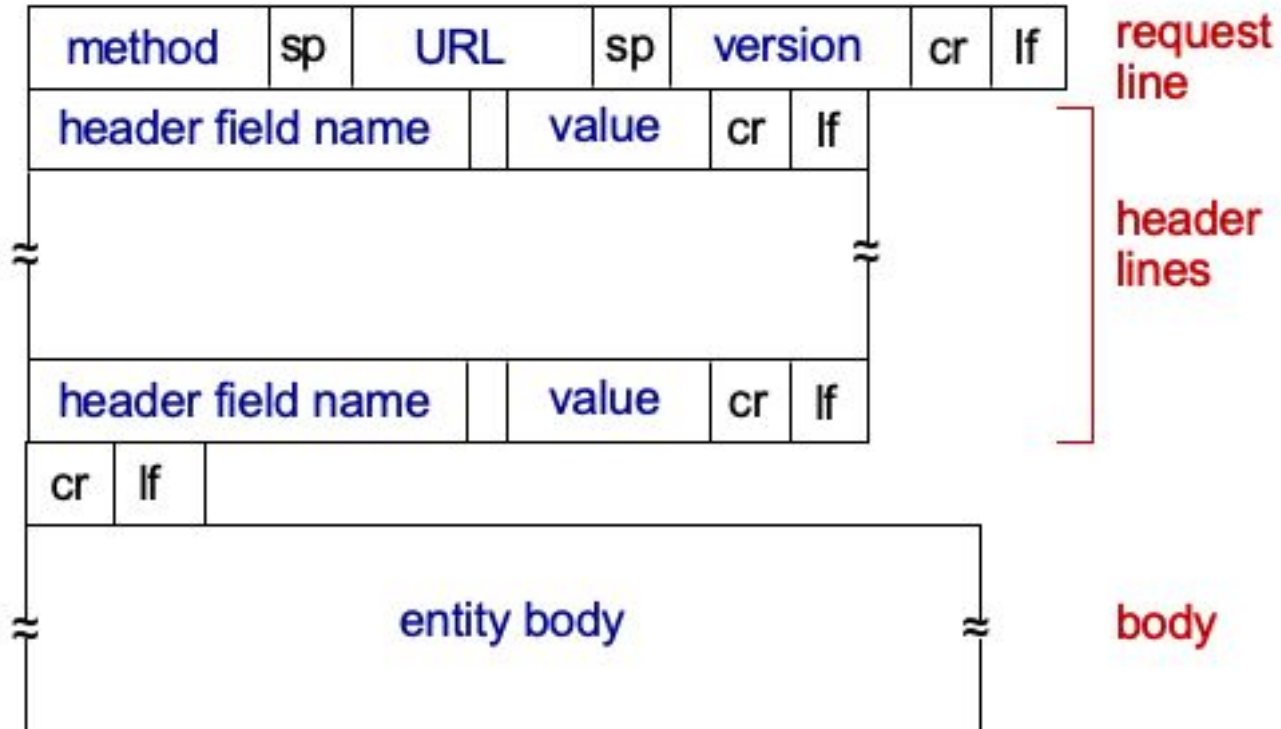
**carriage return, line feed at start of line indicates end of header lines**

**carriage return character**

**line-feed character**

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

# HTTP request message





# Method types

- **GET:** request a document; server responds by sending status information followed by a copy of the document
- **HEAD:** request status information; server responds by sending status information, but not the document
- **POST:** sends data to a server; the server appends the data to a specified item
- **PUT:** sends data to a server; the server uses the data to completely replace the specified item

# HTTP response message

status line  
(protocol  
status code  
status phrase)

header  
lines

data, e.g.,  
requested  
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
1\r\n
\r\n
data data data data data ...
```

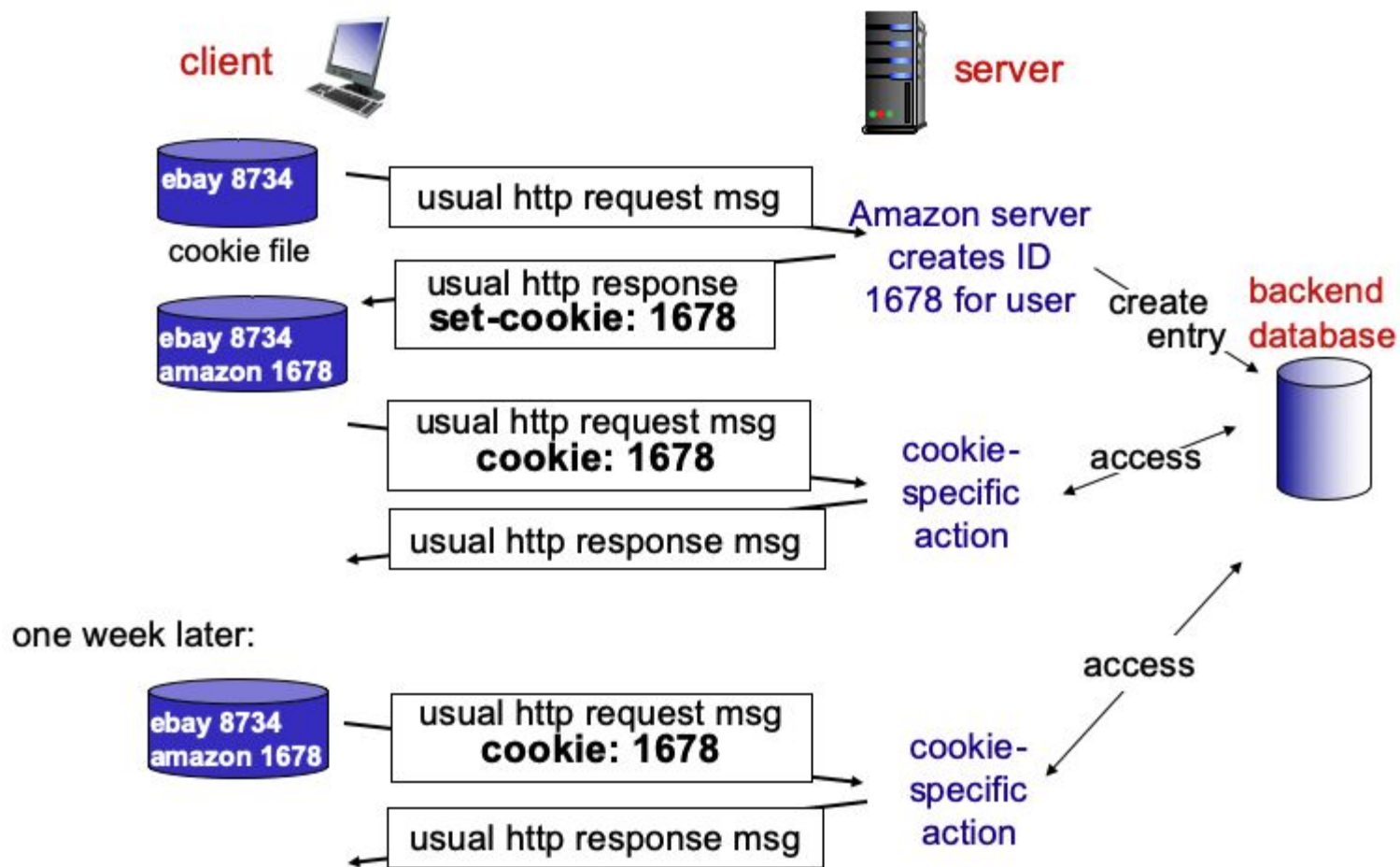
# HTTP response status codes

- Appears in the first line
- Some sample codes:
  - 200 OK
  - 301 Moved Permanently
  - 400 Bad Request
  - 404 Not Found
  - 505 HTTP Version Not Supported

# Cookies

- Cookie header line of HTTP response message
- Cookie header line in next HTTP request message
- Cookie file kept on user's host, managed by user's browser
- Back-end database at server side

# Cookies

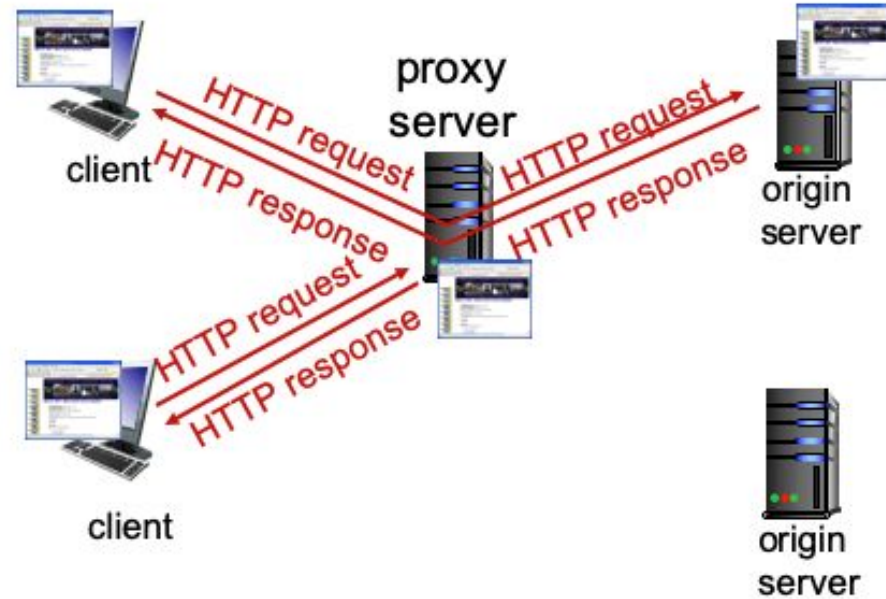


# Cookies

- What cookies can be used for
  - Authorization
  - Shopping carts
  - Recommendations
  - Ads
- Privacy
  - Cookies permit sites to learn a lot about you
  - You may supply name and email to sites

# Web proxy server

- Goal: satisfy client request without involving origin server
- Browser sends all HTTP requests to proxy
  - Hit, return object.
  - Miss, request from origin server



# Web proxy server

- Proxy acts as both client and server
- Typically proxy is installed by ISP (university, company, etc.)
- What are the benefits?



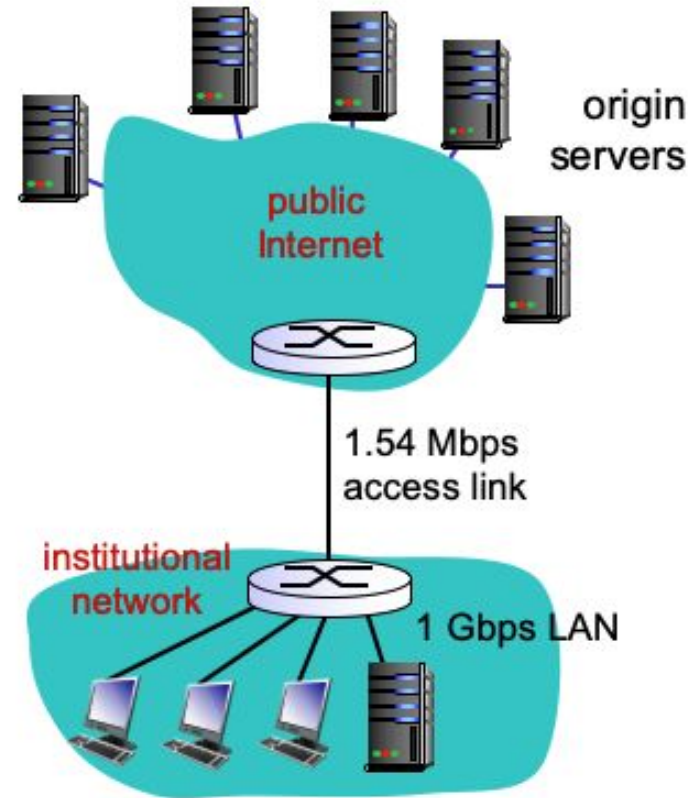


# Web proxy server

- Benefits
  - Reduce response time for client request
  - Reduce traffic on an institution's access link
  - Better user experience and save money

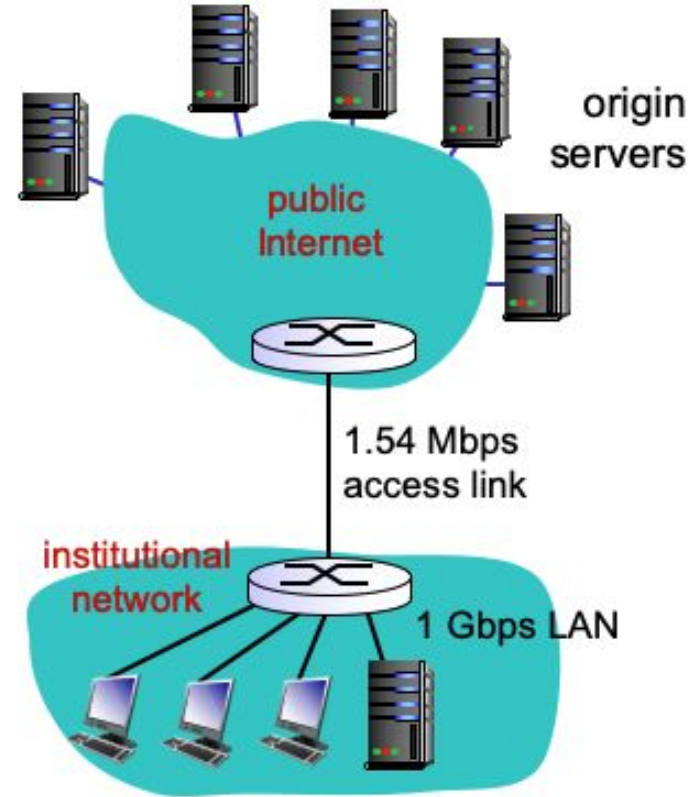
# Some calculation

- Assumptions
  - Avg object size: 100K bits
  - Avg request rate from browsers to origin server: 15/sec
  - Avg data rate:  $100\text{K bits} * 15/\text{sec} = 1.50 \text{ Mbps}$
  - RTT from institutional router to origin server: 2 sec
  - Access link data rate: 1.54 Mbps
  - Local network data rate: 1 Gbps



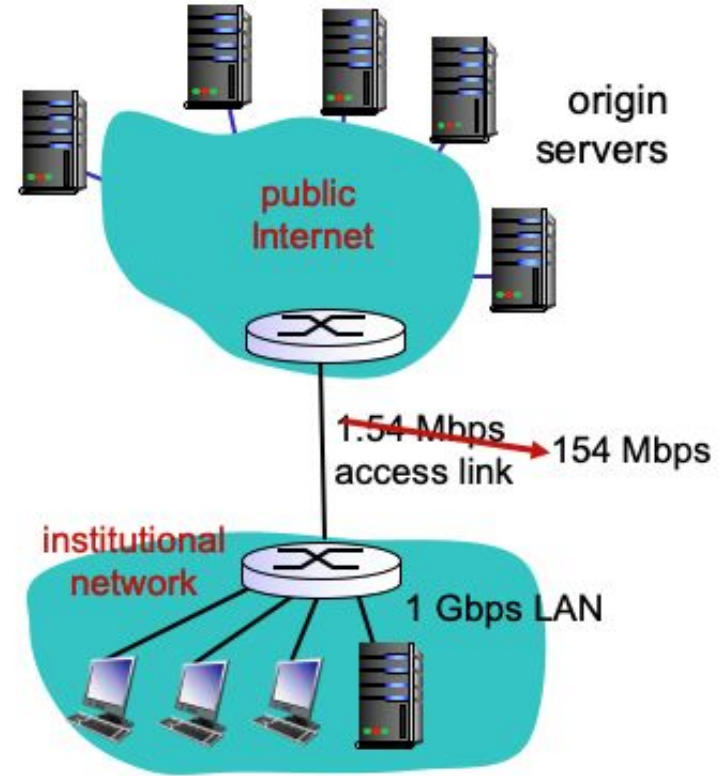
# Some calculation

- Consequences
  - LAN utilization  
 $1.5 \text{ Mbps} / 1 \text{ Gbps} = 15\%$
  - Access link utilization  
 $1.5 \text{ Mbps} / 1.54 \text{ Mbps} = 97\%$ 
    - What happens in this case?
  - What is the total delay? Is it good user experience?



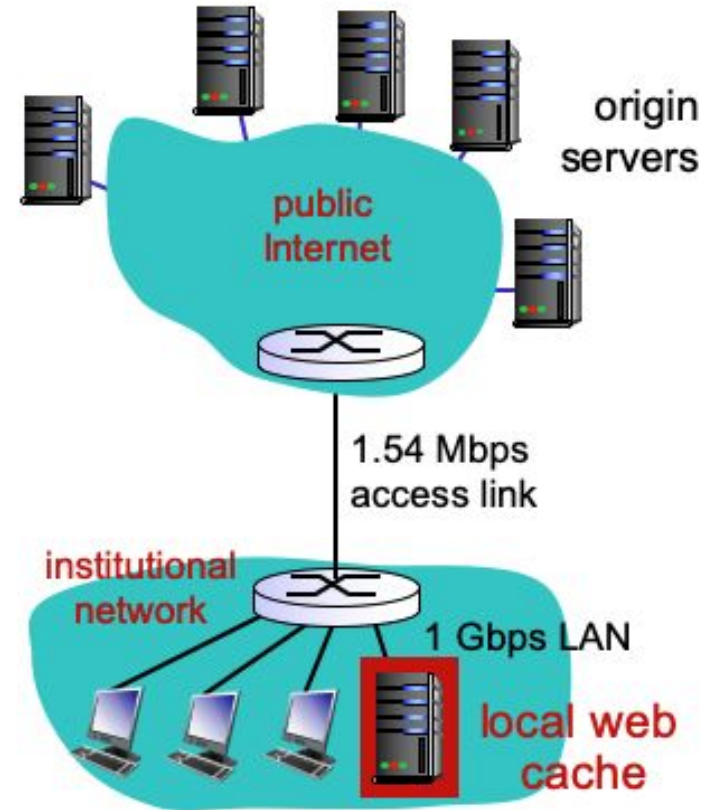
# Solution I

- Fatter access link
- Access link utilization:  
 $1.5 \text{ Mbps} / 154 \text{ Mbps} = 9.7\%$
- What's the total delay? Is it good user experience?
- Money solves everything?



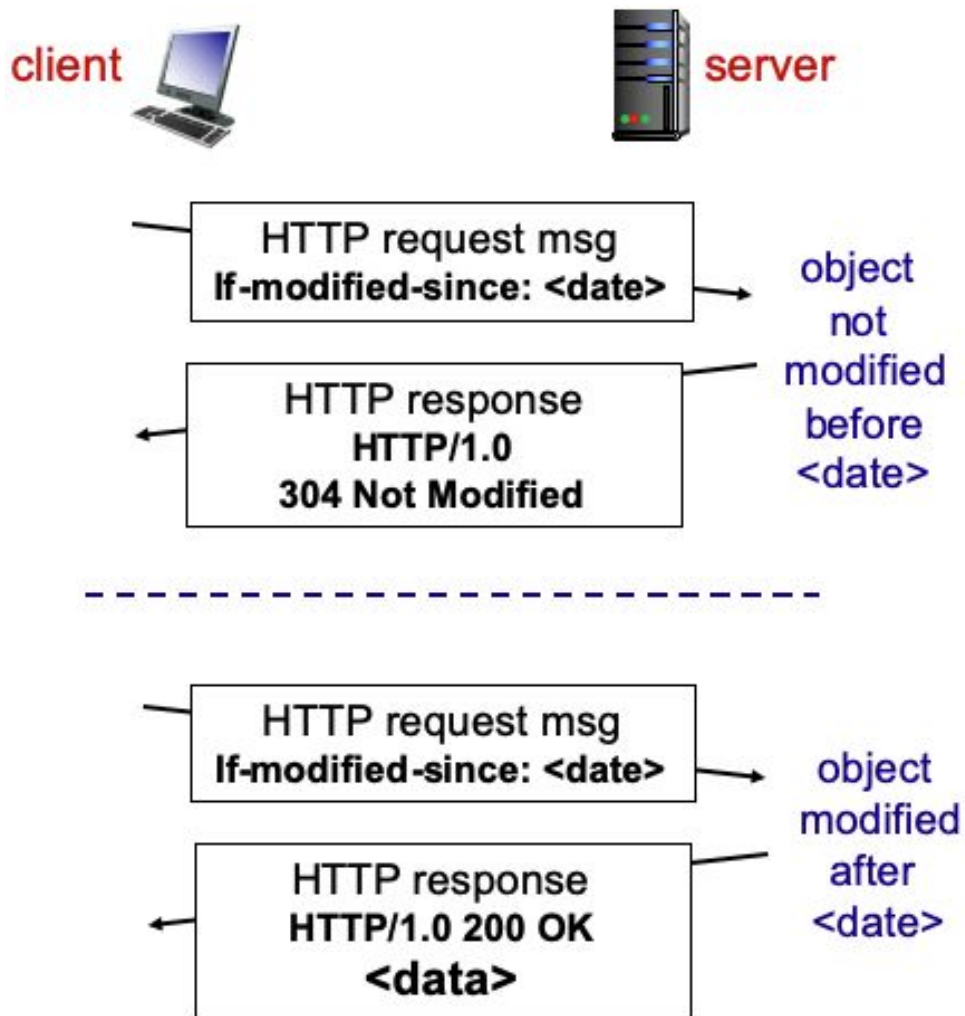
# Solution II

- Assume hit rate is 40%
- Access link utilization:  
 $1.5 \text{ Mbps} * 60\% / 1.54 \text{ Mbps} = 58\%$
- What's the total delay?  
 $60\% * (\text{delay from origin server}) + 40\% * (\text{delay from proxy})$



# Conditional GET

- Goal: don't send object if cache has up-to-date version

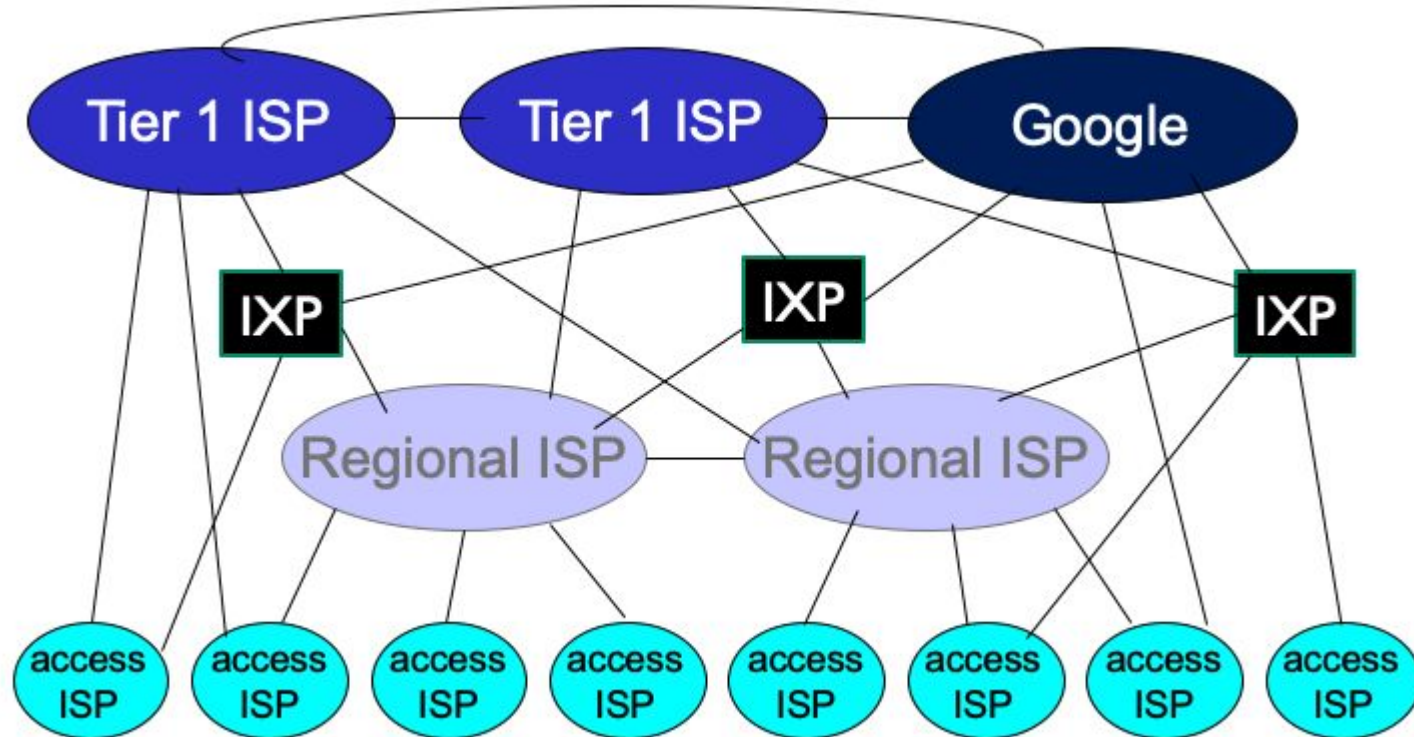




**CDN**

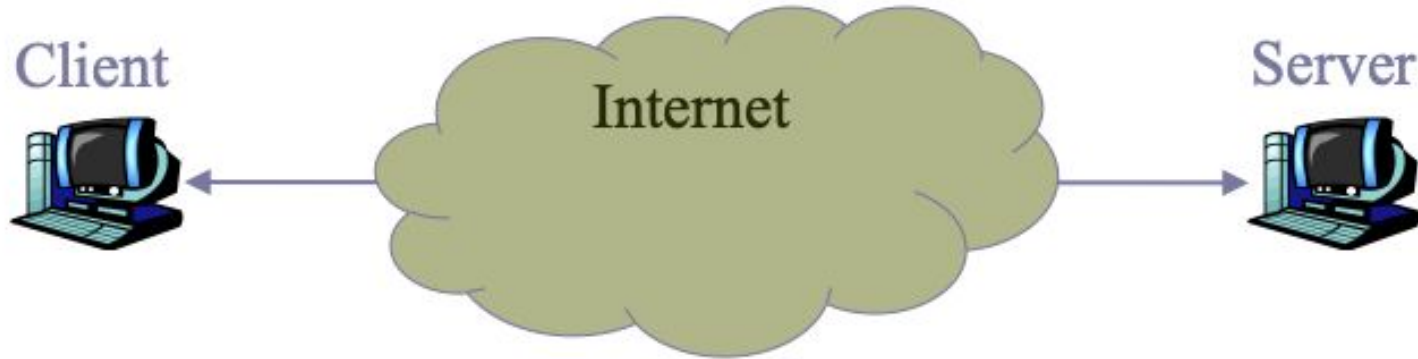


# Review structure of the Internet



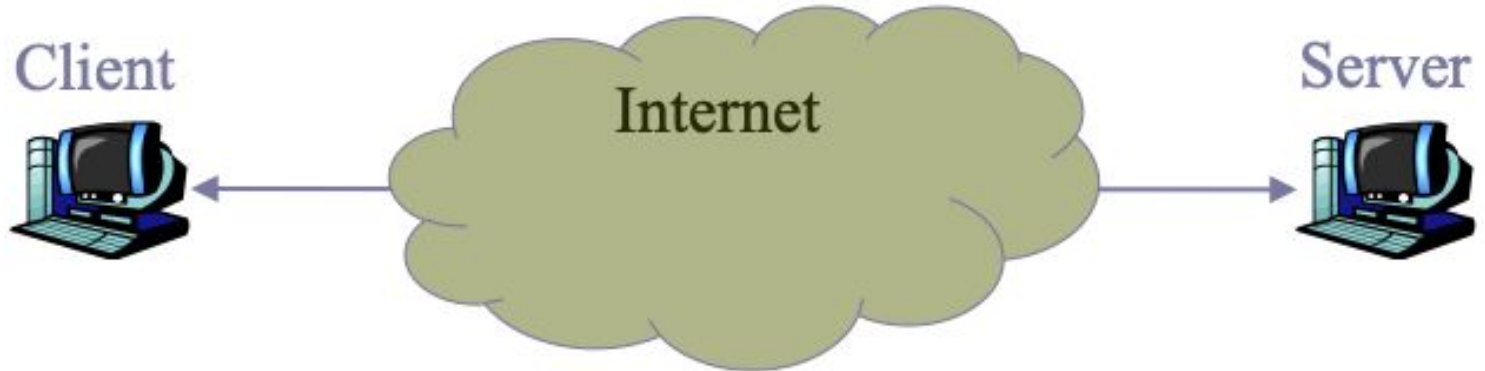
# Serve web content

- Why not a single server for web content? (e.g. Google search or Youtube)



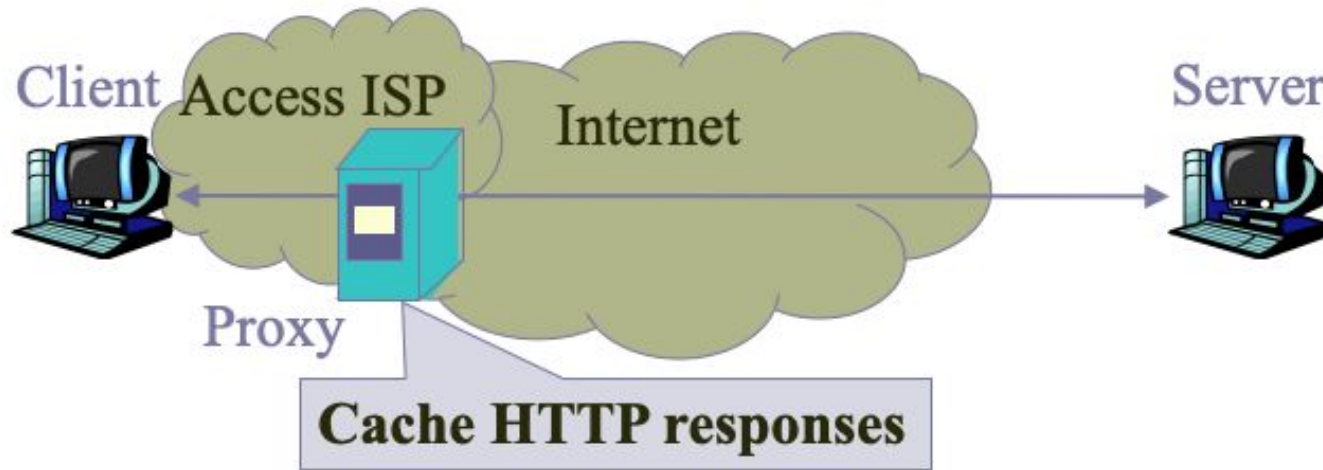
# Single web server

- Single point of failure
- Easier to be overloaded
- Long latency
- etc.



# What about ISP proxy caching?

- Does this solve the single server issues once for all?



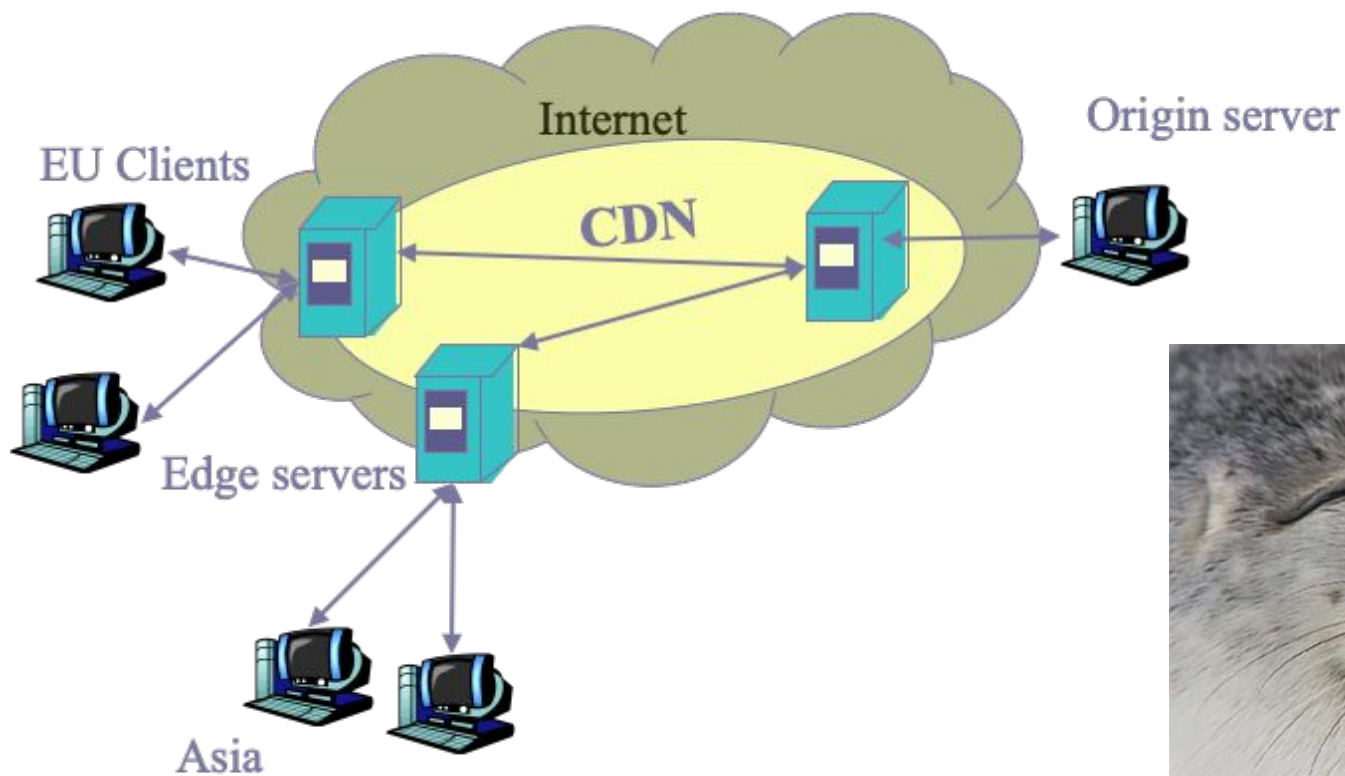
# What about ISP proxy caching?

- Pros
  - Reduced latency for cached contents
- Cons
  - Security/authentication
  - Fine-grained control on when and where to cache content
  - Cold start

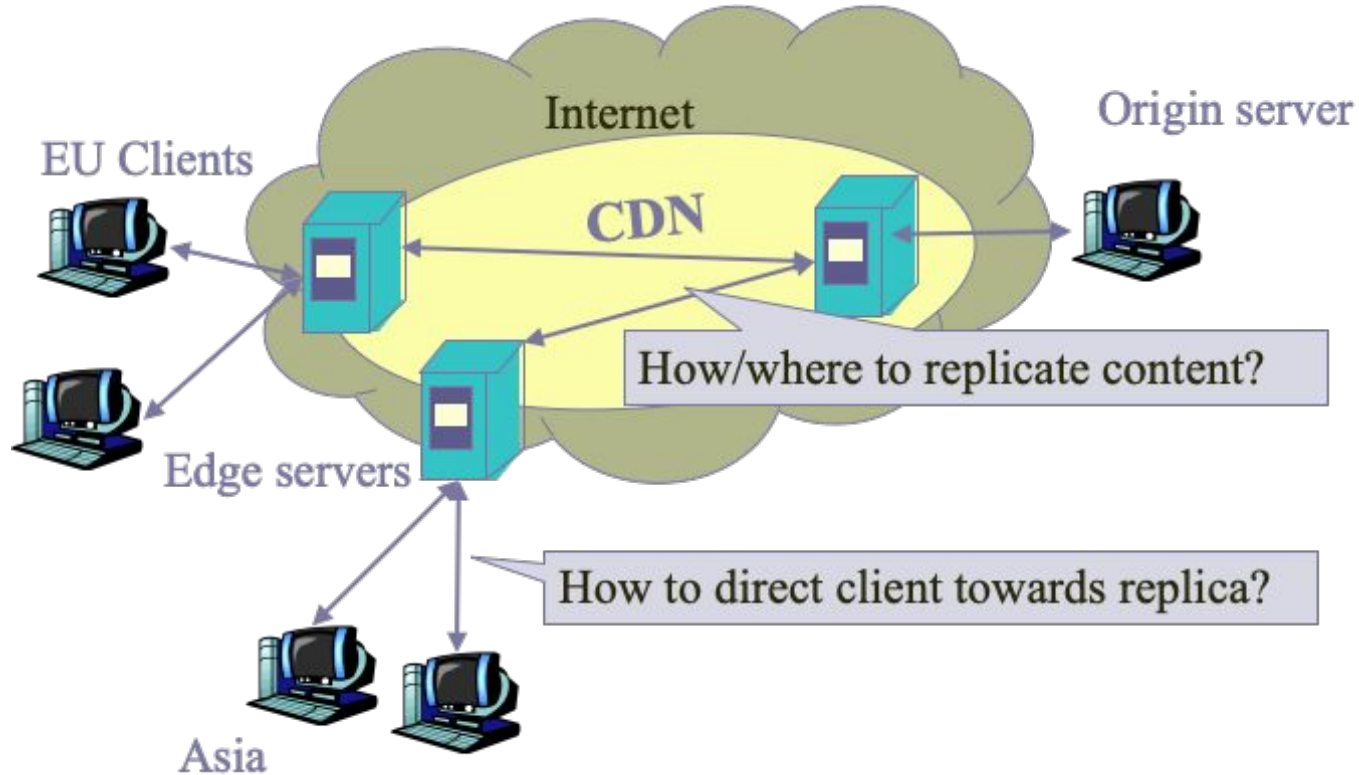
# CDN

- Content delivery networks
- Content providers contract with CDN companies
- CDN companies have servers in lots of networks
- Better coordination between replicas (controlled refresh and removal)
- Proactively content replication on edge servers
- Win-win for both content providers and ISPs

# CDN



But,...





# Akamai

- Distributed servers
  - Over 250,000 servers
  - 137 countries
  - ~1,600 networks
- Scale
  - Over 50 Tbps
  - 15-30% of global traffic

# Who is using Akamai?

- 55 percent of the Fortune Global 500



# How Akamai works

- Clients delegate domain to Akamai

```
mit.edu.      1506      IN        NS        eur5.akam.net.  
mit.edu.      1506      IN        NS        ns1-37.akam.net.  
mit.edu.      1506      IN        NS        use5.akam.net.  
mit.edu.      1506      IN        NS        usw2.akam.net.  
mit.edu.      1506      IN        NS        asial.akam.net.
```

- CNAME chaining

```
www.mit.edu.      1799      IN        CNAME     www.mit.edu.edgekey.net.  
www.mit.edu.edgekey.net. 58      IN        CNAME     e9566.dscb.akamaiedge.net.  
e9566.dscb.akamaiedge.net. 18      IN        A         184.51.176.128
```

# How Akamai works

