# 应用层

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

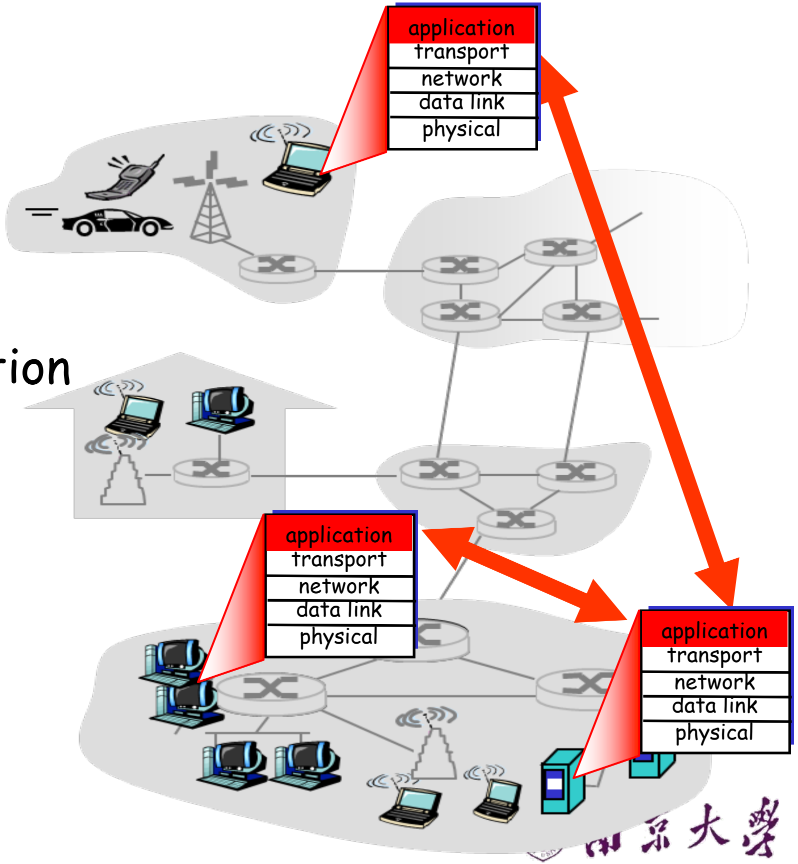
# Internet Applications

- Internet Applications Overview
- Domain Name Service (DNS)
- Electronic Mail
- File Transfer Protocol (FTP)
- WWW and HTTP
- Content Distribution Networks (CDNs)

# Internet Applications Overview

Application: communicating, distributed processes

- e.g., Email, Web, P2P file sharing, instant messaging
- Running in end systems (hosts)
- Exchange messages to implement application




Application-layer protocols

- One "piece" (agent) of an app
- Define messages exchanged by apps and actions taken
- Use communication services provided by lower layer protocols (TCP, UDP, RTP)

# Typical Internet Applications

| Application | App-Layer Protocol | Underlying Transport Protocol |
| --- | --- | --- |
| Email | SMTP [RFC 2821] | TCP |
| Remote terminal access | Telnet [RFC 854] | TCP |
| Web | HTTP [RFC 2616] | TCP |
| File transfer | FTP [RFC 959] | TCP |
| Streaming multimedia | Proprietary<br>e.g. RealNetworks | RTP, RTSP<br>TCP or UDP |
| Internet telephony | Proprietary<br>e.g. Dialpad | SIP on UDP |

# Jargons of Internet Applications

- Process: program running within a host
  - Within same host, 2 processes communicate using inter-process communication (defined by OS)
  - Processes running in different hosts communicate with an app-layer protocol
- User agent: interfaces with app "above" and network "below"
  - Implements user interface & app-layer protocol, e.g.
  - Web: browser, web server
  - Email: mail reader, mail server
  - Streaming audio/video: media player, media server

# App-Layer Protocols

- Types of messages exchanged
  - e.g. request & response messages
- Syntax of message types
  - What fields in messages & how fields are delineated
- Semantics of the fields
  - Meaning of information in fields
- Rules for when and how processes send & respond to messages

# Application Architectures

possible structure of applications:

- client-server (CS)
- peer-to-peer (P2P)

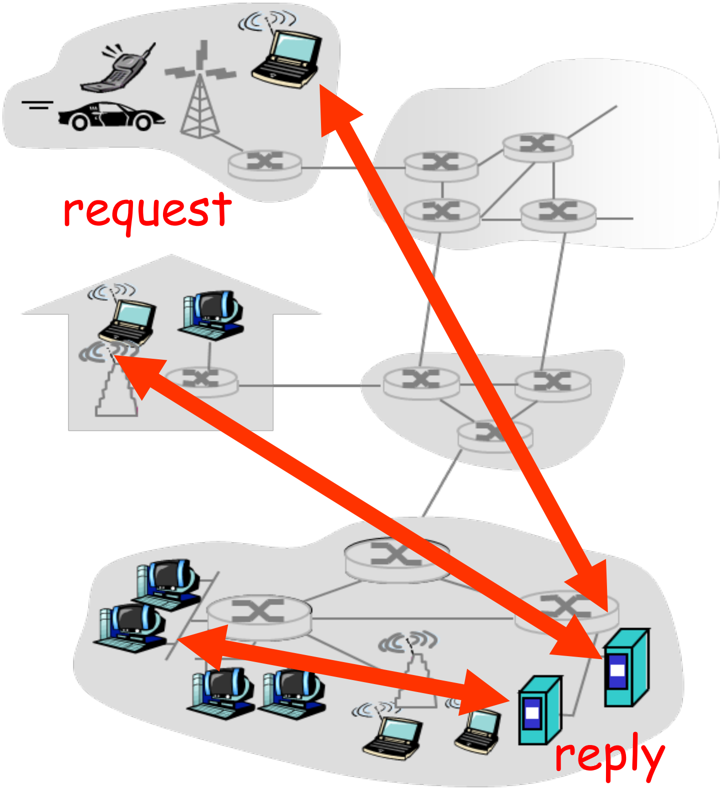
# Client-Server Paradigm

Client:

- Start as required
- Initiates contact with server, "speaks first"
- Host may have dynamic IP addresses
- e.g. Web: client implemented in browser; Email: in mail reader

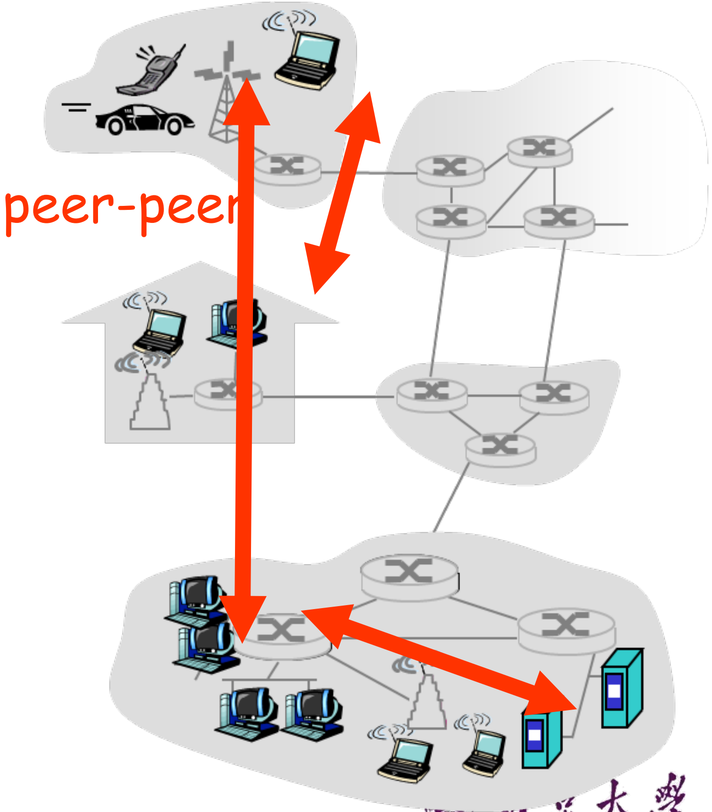
Server:

- Run as daemon (always-on)
- Provides requested service to Client
- Host has permanent IP address
- e.g. Web server sends requested Web page, mail server delivers Email



# Peer-to-Peer Paradigm

- No always-on server
- Arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - self scalability – new peers bring new service capacity, as well as new service demands
- Peers are intermittently connected and change IP addresses
  - Highly scalable but difficult to manage
- Examples: Gnutella, BitTorrent, Skype



# Client-Server and P2P

Skype

- Voice-over-IP P2P application
- Centralized server: finding address of remote party
- Direct client-client connection

Instant messaging

- Chatting between two users is P2P
- Centralized service: user presence detection/location
- User registers its IP address with central server when it comes online
- User contacts central server to find IP addresses of parties

# Typical Applications

- DNS
- Email
- FTP
- Web and HTTP
- CDN
- P2P Applications

# Internet Applications

- Internet Applications Overview
- Domain Name Service (DNS)
- Electronic Mail
- File Transfer Protocol (FTP)
- WWW and HTTP
- Content Distribution Networks (CDNs)

# Domain Name Service (DNS)

- Function
  - Map “domain names” into IP addresses
  - e.g. www.baidu.com → 119.75.217.109
- Domain Name System
  - Distributed database implemented in hierarchy of many name servers
  - App-layer protocol host and name servers to communicate to resolve “domain names”
  - Load balancing: set of IP addresses for one server name

Q: why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn’t scale!

# Goals

- Uniqueness: no naming conflicts
- Scalable
  - Many names and frequent updates (secondary)
- Distributed, autonomous administration
  - Ability to update my own (machines') names
  - Don't have to track everybody's updates
- Highly available
- Lookups are fast
- Perfect consistency is a non-goal

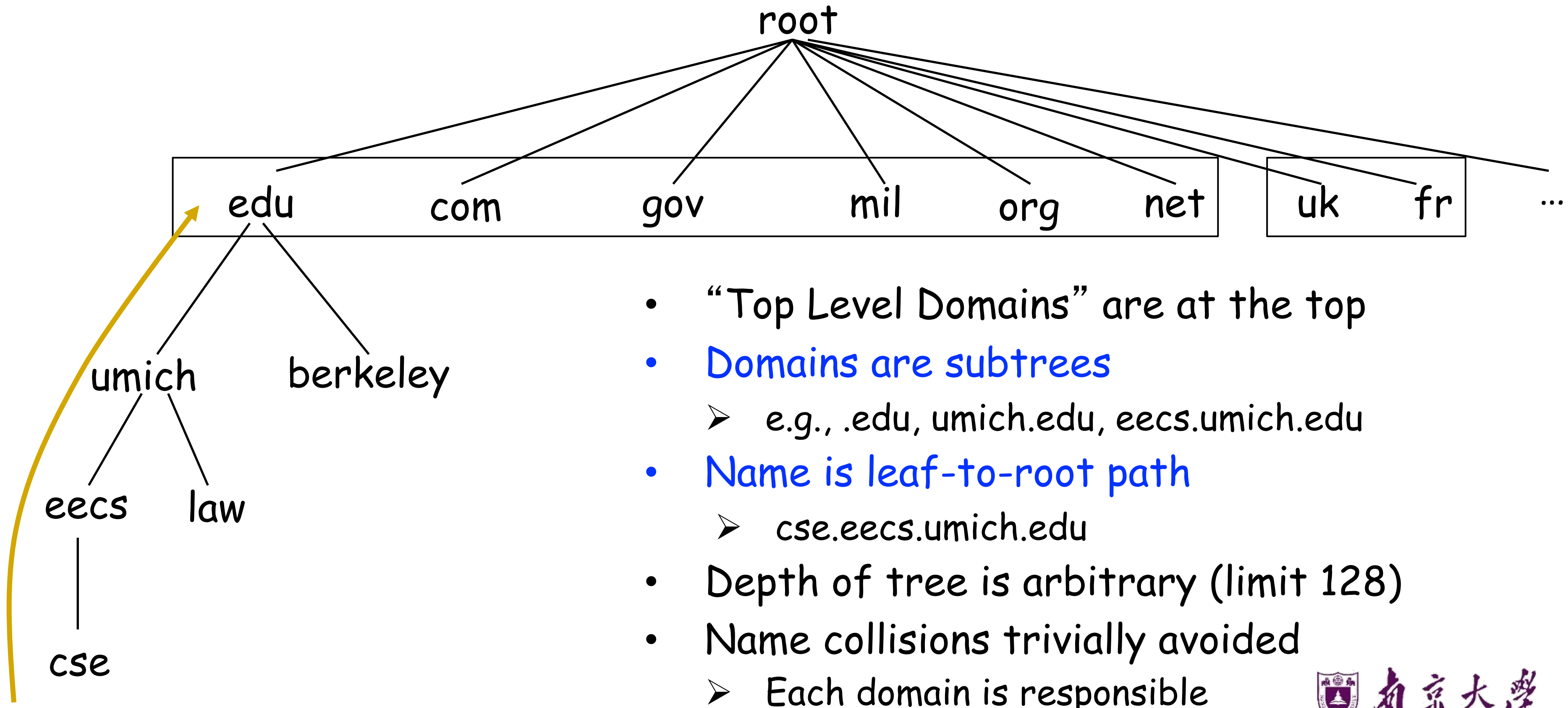
# How?

- Partition the namespace
- Distribute administration of each partition
  - Autonomy to update my own (machines') names
  - Don't have to track everybody's updates
- Distribute name resolution for each partition
- How should we partition things?

# Key idea: Hierarchy

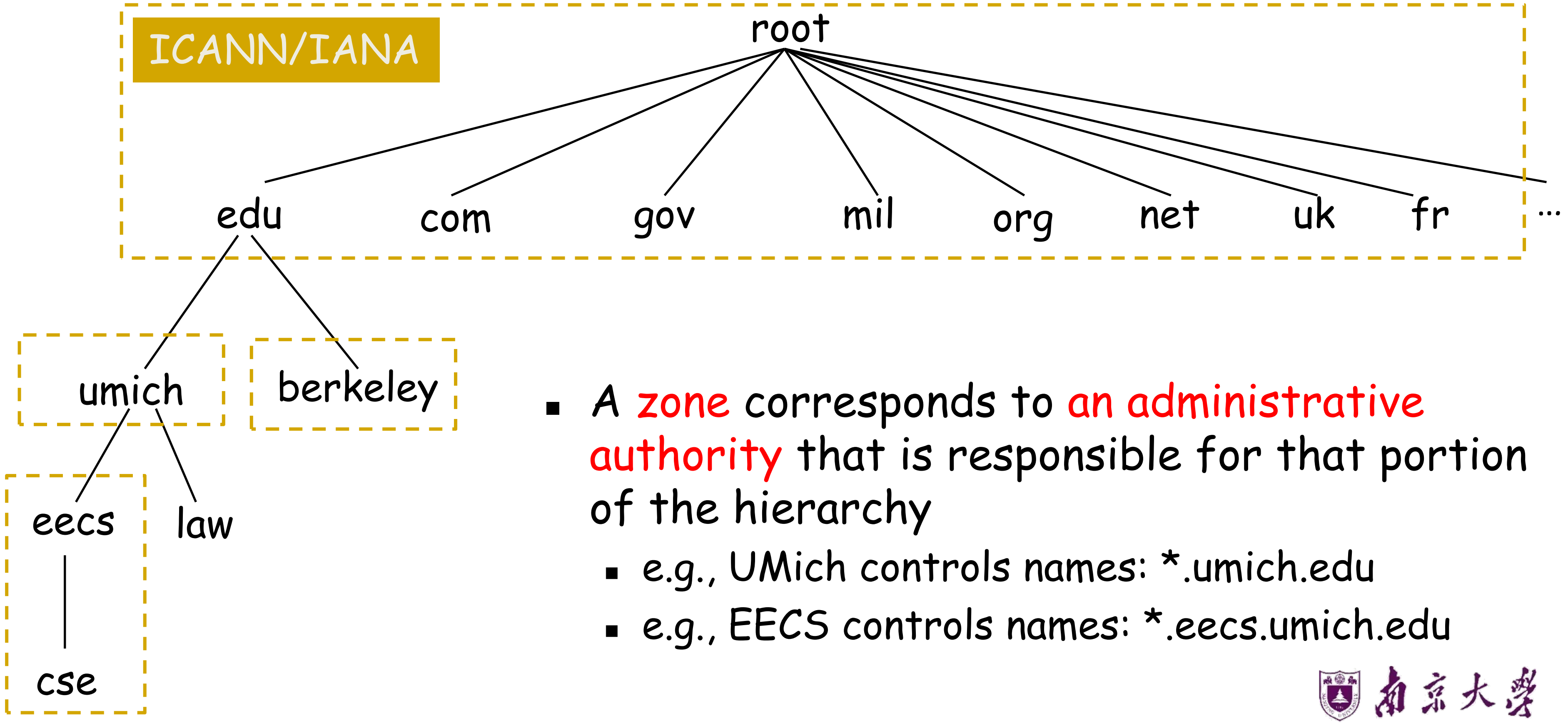
- Three intertwined hierarchies
  - ➢ Hierarchical namespace
    - ✓ As opposed to original flat namespace
  - ➢ Hierarchically administered
    - ✓ As opposed to centralized
  - ➢ (Distributed) hierarchy of servers
    - ✓ As opposed to centralized storage

# Hierarchical namespace




- "Top Level Domains" are at the top
- Domains are subtrees
  - e.g., .edu, umich.edu, eecs.umich.edu
- Name is leaf-to-root path
  - cse.eecs.umich.edu
- Depth of tree is arbitrary (limit 128)
- Name collisions trivially avoided
  - Each domain is responsible

# Hierarchical administration




- A zone corresponds to an administrative authority that is responsible for that portion of the hierarchy
  - e.g., UMich controls names: *.umich.edu
  - e.g., EECS controls names: *.eecs.umich.edu

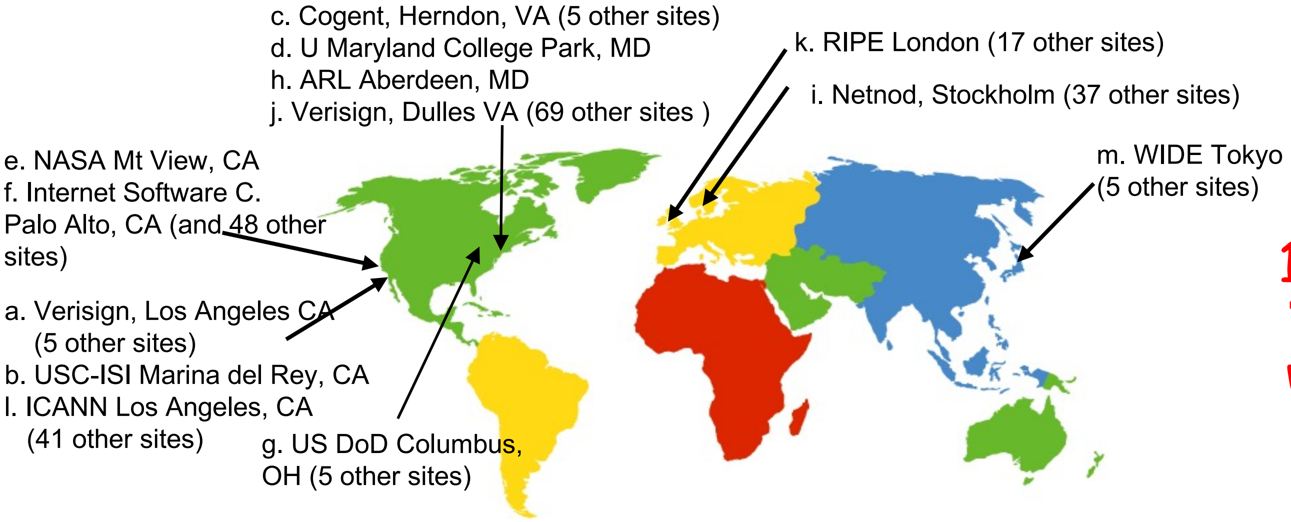
# Hierarchy of DNS Servers

- Root name servers
  - Contacted by local name server that can not resolve name
- Top-level domain servers
  - Responsible for com, org, net, edu, etc, and all top-level country domains, e.g. cn, uk, fr
- Authoritative DNS servers
  - Organization's DNS servers, providing authoritative hostname to IP mappings
- Local Name Servers
  - Maintained by each residential ISP, company, university
  - When host makes DNS query, query is sent to its local DNS server

- Each server stores a (small!) subset of the total DNS database
- An authoritative DNS server stores "resource records" for all DNS names in the domain that it has authority for
- Each server needs to know other servers that are responsible for the other portions of the hierarchy
  - Every server knows the root
  - Root server knows about all top-level domains

# DNS: root name servers

- root name server:
  - returns IP mappings of TLD servers




13 root name "servers" worldwide

# TLD, authoritative servers

- Top-level domain (TLD) servers:
  - responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
  - Network Solutions maintains servers for .com TLD
  - Educause for .edu TLD

- Authoritative DNS servers:
  - organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
  - can be maintained by organization or service provider

# Local DNS name server

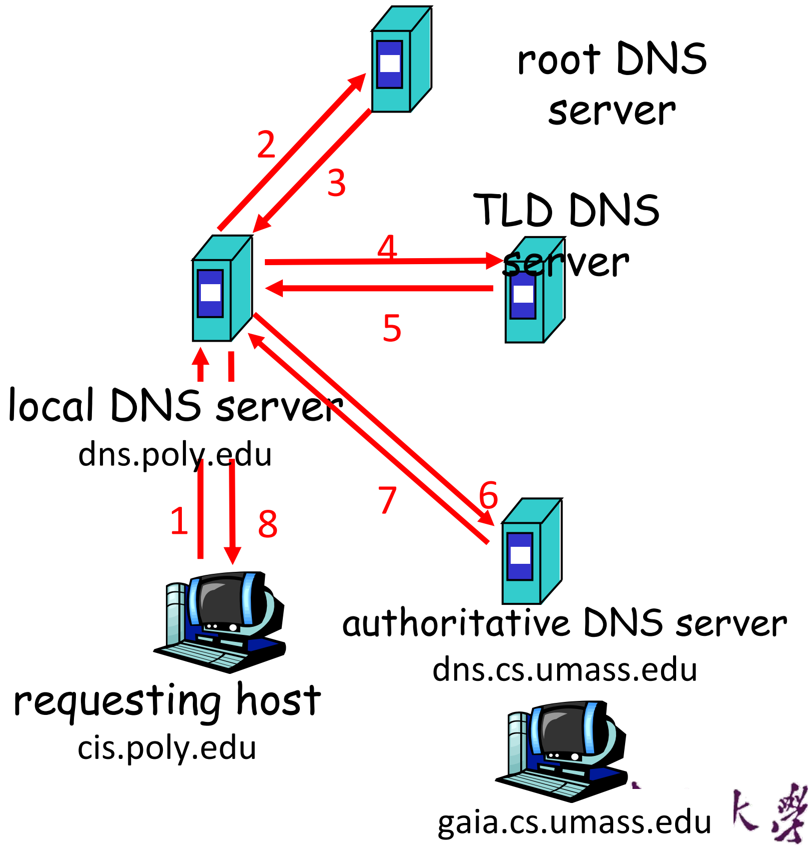
- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one
  - also called “default name server”
- When host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

# DNS Name Resolution Example

- Bob at cis.poly.edu wants IP address for Alice at gaia.cs.umass.edu

- Iterated query:
- Contacted server replies with name of next server to contact

- Host-Server: recursive query
- Server-Server: iterative query



# DNS Records

- A DNS resource record (RR)

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

- "Name" is the domain name, "type" denotes how "value" is explained
  - e.g. Name Server records (NS), Mail Exchangers (MX), Host IP Address (A), Canonical name (CNAME)
- Examples
  - (networkutopia.com, dns1.networkutopia.com, NS, 32768)
  - (dns1.networkutopia.com, 212.212.212.1, A, 5600)

# DNS protocol

- Query and Reply messages; both with the same message format
  - Header: identifier, flags, etc.
  - Plus resource records
  - See text/section for details
- Client-server interaction on UDP Port 53
  - Spec supports TCP too, but not always implemented

# Goals: Are we there yet?

- Uniqueness: No naming conflicts
- Scalable
- Distributed, autonomous administration
- Highly available?

# Reliability

- Replicated DNS servers (primary/secondary)
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- Usually, UDP used for queries
  - Reliability, if needed, must be implemented on UDP
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don't care which server responds

# Goals: Are we there yet?

- Uniqueness: No naming conflicts
- Scalable
- Distributed, autonomous administration
- Highly available
- Fast lookups?

# DNS caching

- Performing all these queries takes time
  - Up to 1-second latency before starting download
- Caching can greatly reduce overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., www.cnn.com) visited often
  - Local DNS server often has the information cached
- How DNS caching works
  - DNS servers cache responses to queries
  - Responses include a "time to live" (TTL) field
  - Server deletes cached entry after TTL expires

# Internet Applications

- Internet Applications Overview
- Domain Name Service (DNS)
- Electronic Mail
- File Transfer Protocol (FTP)
- WWW and HTTP
- Content Distribution Networks (CDNs)

# Electronic Mail

- One of most heavily used apps on Internet
- SMTP: Simple Mail Transfer Protocol
  - Delivery of simple text messages
- MIME: Multi-purpose Internet Mail Extension
  - Delivery of other types of data, e.g. voice, images, video clips
- POP: Post Office Protocol
  - Msg retrieval from server, including authorization and download
- IMAP: Internet Mail Access Protocol
  - Manipulation of stored msgs on server

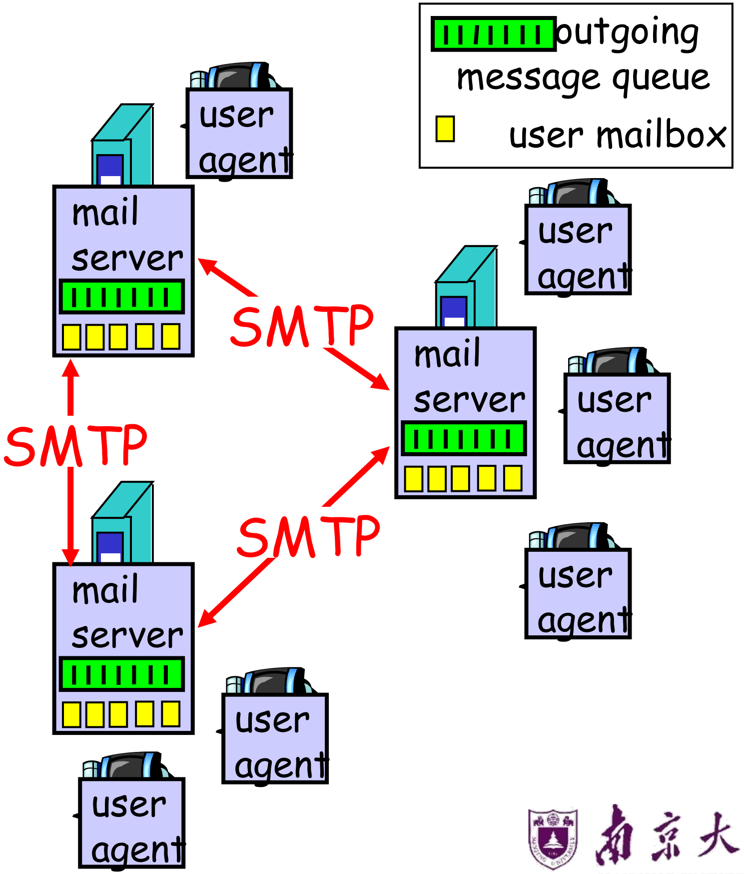
# Components of Email System

## User Agent

- Composing, editing, reading mail messages
- e.g. Eudora, Outlook, Foxmail, Netscape Messenger
- Outgoing, incoming mail messages stored on server

## Mail Servers (Host)

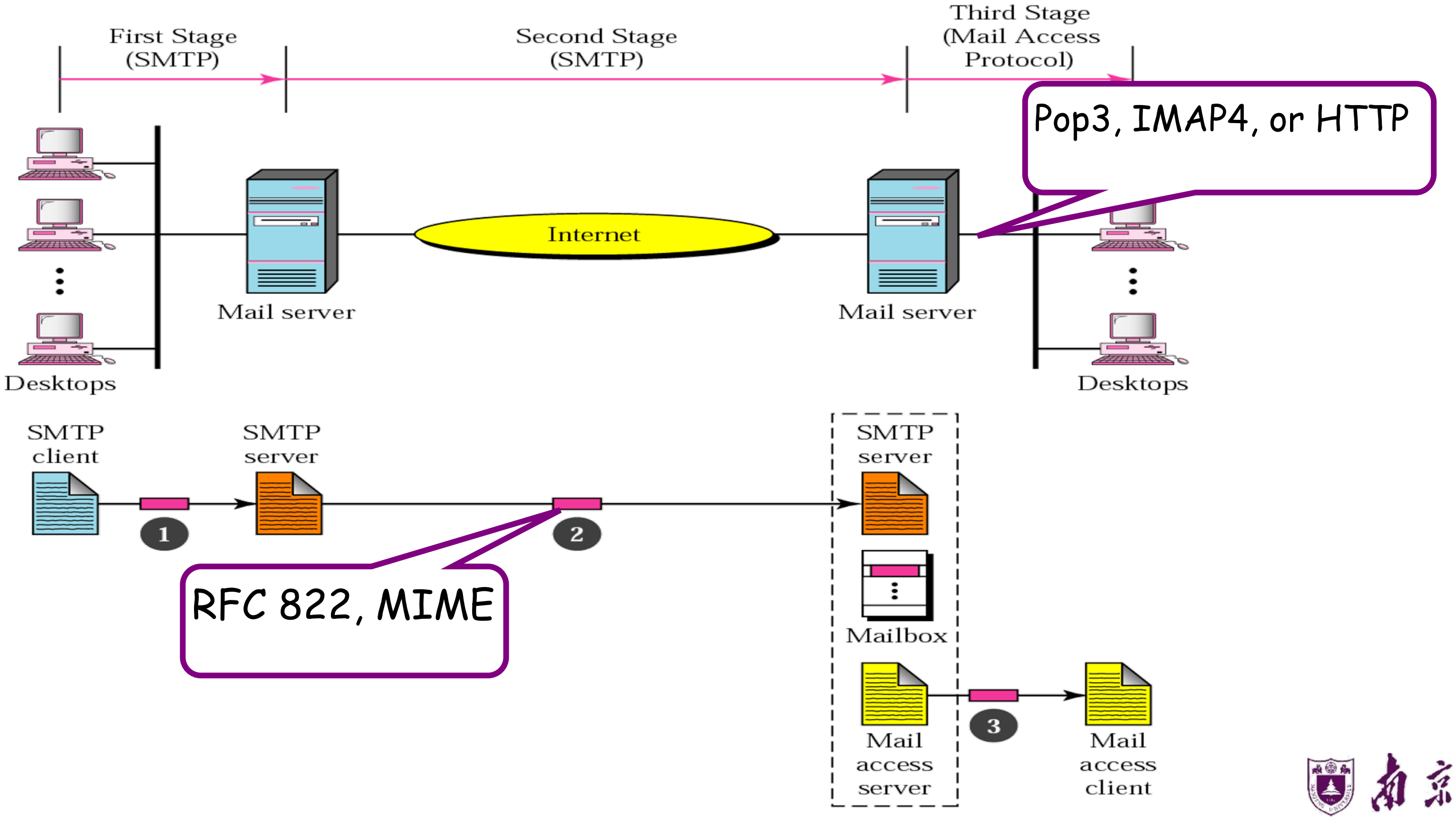
- Mailbox contains incoming mail messages for user
- Message queue of outgoing mail messages
- SMTP protocol between mail servers to send mail messages



# 3 Stages of Mail Delivery

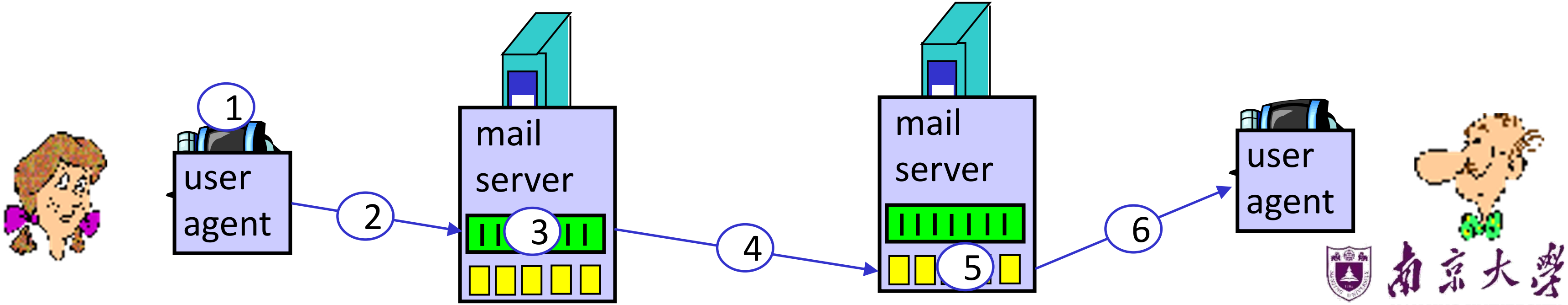
- 1st Stage
  - Email goes from local user agent to the local SMTP server
  - User agent acts as SMTP client
  - Local server acts as SMTP server
- 2nd Stage
  - Email is relayed by the local server to the remote SMTP server
  - Local server acts as SMTP client now
- 3rd Stage
  - The remote user agent uses a mail access protocol to access the mailbox on remote server
  - POP3 or IMAP4

# Illustration of Mail Delivery



# A Mail Delivery Scenario

- 1) Alice uses UA to compose a mail message and to bob@someschool.edu
- 2) Alice's UA sends mail to her mail server using SMTP, mail placed in message queue
- 3) Client side of SMTP opens TCP connection with Bob's mail server
- 4) SMTP client sends Alice's mail over the TCP connection
- 5) Bob's mail server places the mail in Bob's mailbox
- 6) Bob invokes his UA to read the mail, e.g. by Pop3



# SMTP

- RFC 821:
  - Uses TCP, port 25
  - Direct transfer: transfer Email message from client to server
  - Needs info written on envelope of a mail (i.e. message header)
  - May add log info to message header to show the path taken
- Does not cover format of mail messages or data
  - Defined in RFC 822 or MIME
  - Messages must be in 7-bit ASCII

# SMTP Transaction

3 phases of transfer

- Handshaking (greeting)
- Transfer of one or more mails data
- Close connection

Command/response interaction

- Commands: ASCII text
- Response: status code and phrase

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr … Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu … Recipient ok
C: RCPT TO: <Johm@hamburger.edu>
S: 550 No such user here
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C:    How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

# Try SMTP interaction for yourself:

- **telnet servername 25**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

# Reliability of SMTP

- Transfer mails from sender to receiver over TCP connection
  - Rely on TCP to provide reliable service

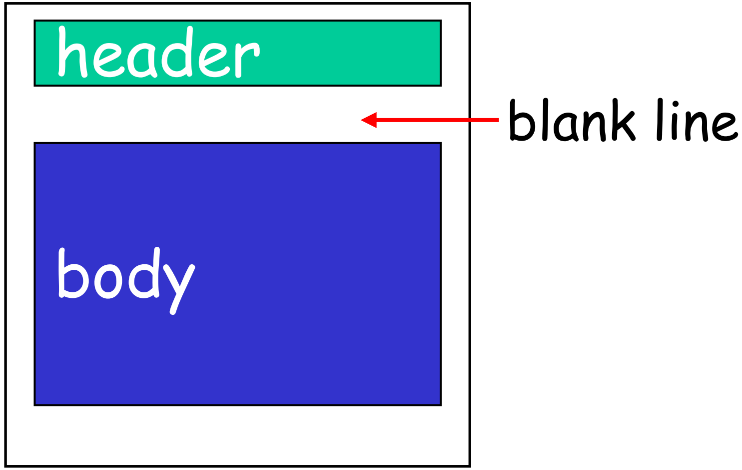
- No guarantee to recover lost mails
- No end to end acknowledgement to originator (user)

- Error indication delivery not guaranteed
  - Indicates mail has arrived at host, but not user

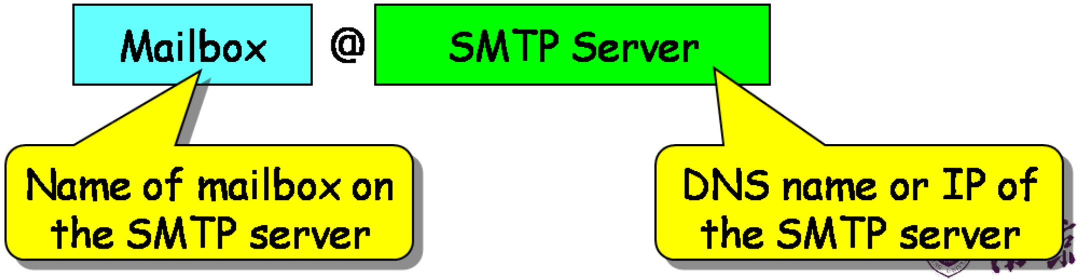
- Generally considered reliable

# An Email Message

- Header lines, e.g.
  - To: Alice@sina.com
  - From: Bob@gmail.com
  - Subject: Dinner tonight
- Body
  - Mail contents, ASCII characters only

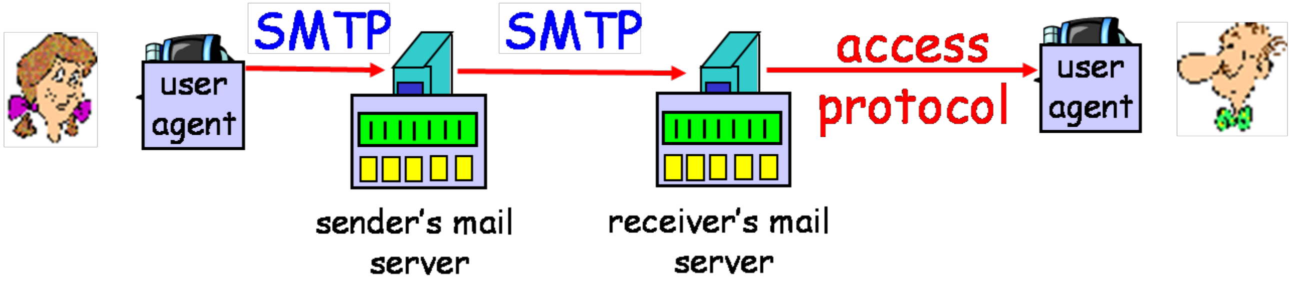



- Mail destinations



# Mail Access Protocols

- SMTP: delivery/storage to receiver's server
- Mail access protocol: mail retrieval from server
- POP: Post Office Protocol [RFC 1939]
  - Authorization (agent <-->server) and download
- IMAP: Internet Mail Access Protocol [RFC 1730]
  - more features, including manipulation of stored mails on server
- HTTP: gmail, Hotmail, Yahoo!, etc.



# POP3 Protocol

## Authorization phase

- Client commands
  - user: declare username
  - pass: password
- Server responses
  - +OK
  - -ERR

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

## Transaction phase, by client

- list: list mail numbers
- retr: retrieve mail by number
- dele: delete
- quit

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

# POP3 (more) and IMAP

## more about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

## IMAP

- Internet Mail Access Protocol, RFC 1730
- keeps all messages in one place: at server
  - A complicated use case
    - Bob reads emails at his office while his wife is simultaneously reading from same mailbox at home
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name
  - Keeps track of mail states (read, replied, deleted)

# RFC 822 - Format for Text Mails

- Simple 2-part format
  - Header (envelope) includes transmit and delivery info
  - Lines of text in format keyword: information value
  - Body (contents) carries text of message
  - Header and body separated by a blank line
- Mail is a sequence of lines of text
  - Ends with two <CRLF>

```
From: John@hamburger.edu
To: Alice@crepes.fr
Cc: bob@hamburger.edu
Date: Wed, 4 Sep 2003 10:21:22 EST
Subject: Lunch with me

Alice,
   Can we get together for lunch when
you visit next week? I'm free on
Tuesday or Wednesday. Let me know
which day you would prefer.

John
```

# MIME

- Multipurpose Internet Mail Extension
  - Extends and automates encoding mechanisms
  - Allows inclusion of separate components in a single mail
    - e.g. programs, pictures, audio clips, videos
- Features
  - Compatible with existing mail systems
    - Everything encoded as 7-bit ASCII
    - Headers and separators ignored by non-MIME mail systems
  - MIME is extensible
    - As long as sender and receiver agree on encoding scheme

# Overview of MIME

- 5 new mail header fields
  - MIME version
  - Content type
  - Content transfer encoding
  - Content Id
  - Content Description

- Number of content formats defined
- Transfer encoding defined

# A MIME Mail Example

MIME version

Method used to encode data

Type of data

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
```

# A Multi-Part Example

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="StartOfNextPart"

--StartOfNextPart
Dear Bob, Please find a picture of a crepe.
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
.........................
......base64 encoded data
--StartOfNextPart
Do you want the recipe?
--StartOfNextPart--
```

# Internet Applications

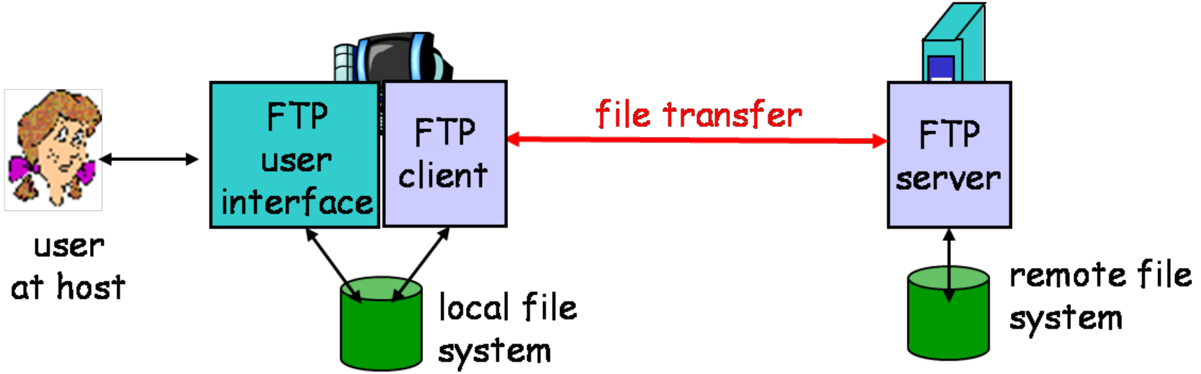
- Internet Applications Overview
- Domain Name Service (DNS)
- Electronic Mail
- File Transfer Protocol (FTP)
- WWW and HTTP
- Content Distribution Networks (CDNs)

# File Transfer Protocol (FTP)

- RFC 959, use TCP, port 21/20
- Transfer file to/from remote host

- Client/Server model, client side initiates file transfer (either to/from remote)

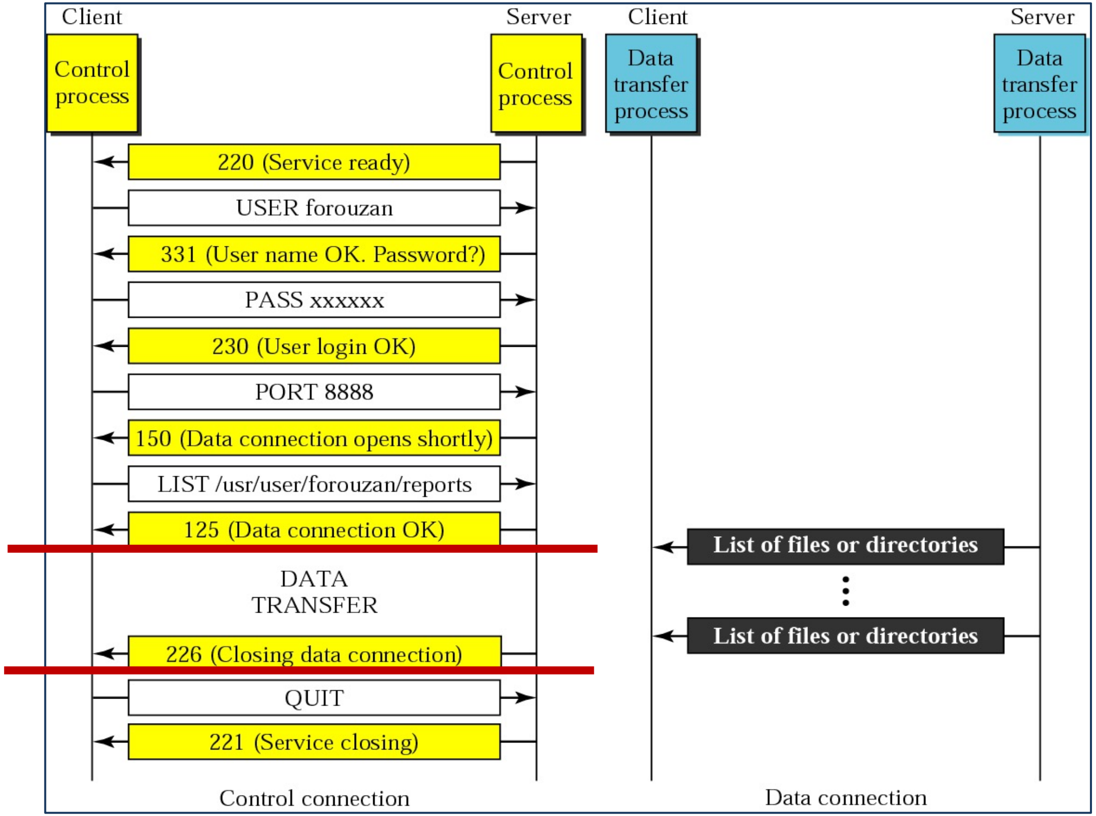
- Deals with heterogeneous OS and file systems
- Needs access control on remote file system



# Control and Data Connections

- FTP client contacts FTP server at port 21, opens a control connection
- Client authorized over control connection
- Client browses remote directory by sending commands over control connection
- When server receives file transfer command, server opens 2nd TCP data connection (for file) to client
  - One connection for each file transferred
- After transferring one file, server closes data connection
- Control connection stays "out of band"
- FTP server maintains "user state": current directory, earlier authentication

# Illustration of FTP Session



# FTP Commands and Responses

Sample commands:

- Sent as ASCII text over control channel
- USER username
- PASS password
- LIST return list of file in current directory
- RETR filename retrieves (gets) file
- STOR filename stores (puts) file onto remote server

Sample return codes:

- Status code and phrase (as in HTTP)
- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can't open data connection
- 452 Error writing file

# BitTorrent

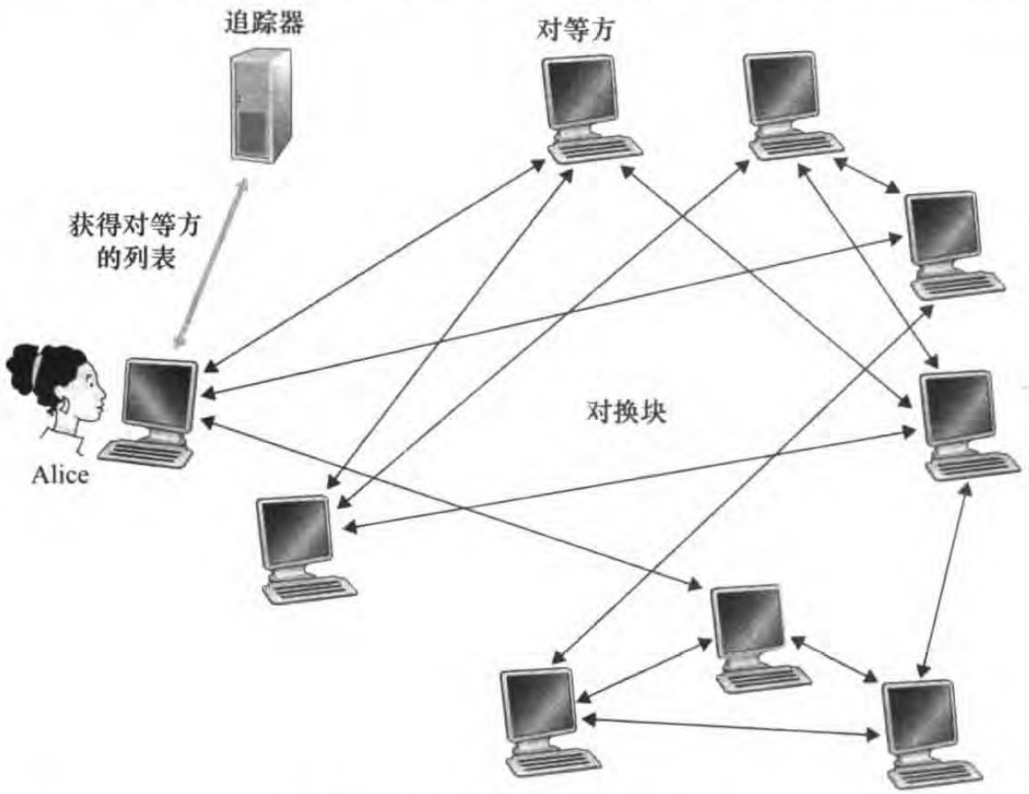



图 2-23　用 BitTorrent 分发文件

# Internet Applications

- Internet Applications Overview
- Domain Name Service (DNS)
- Electronic Mail
- File Transfer Protocol (FTP)
- WWW and HTTP
- Content Distribution Networks (CDNs)

# 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]<br>英格兰伦敦 |
| **机构** | 万维网联盟<br>南安普敦大学<br>Plessey<br>麻省理工学院 |
| **知名于** | 发明万维网<br>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

NANJING UNIVERSITY

# The Web: History

- 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

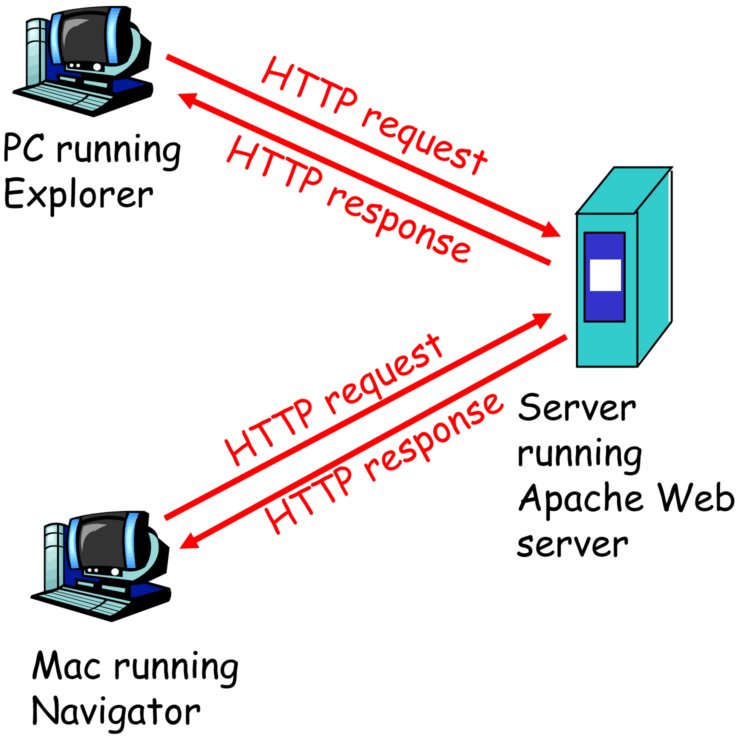
- Protocol for exchanging information: HTTP

# URL - Uniform Resource Locator

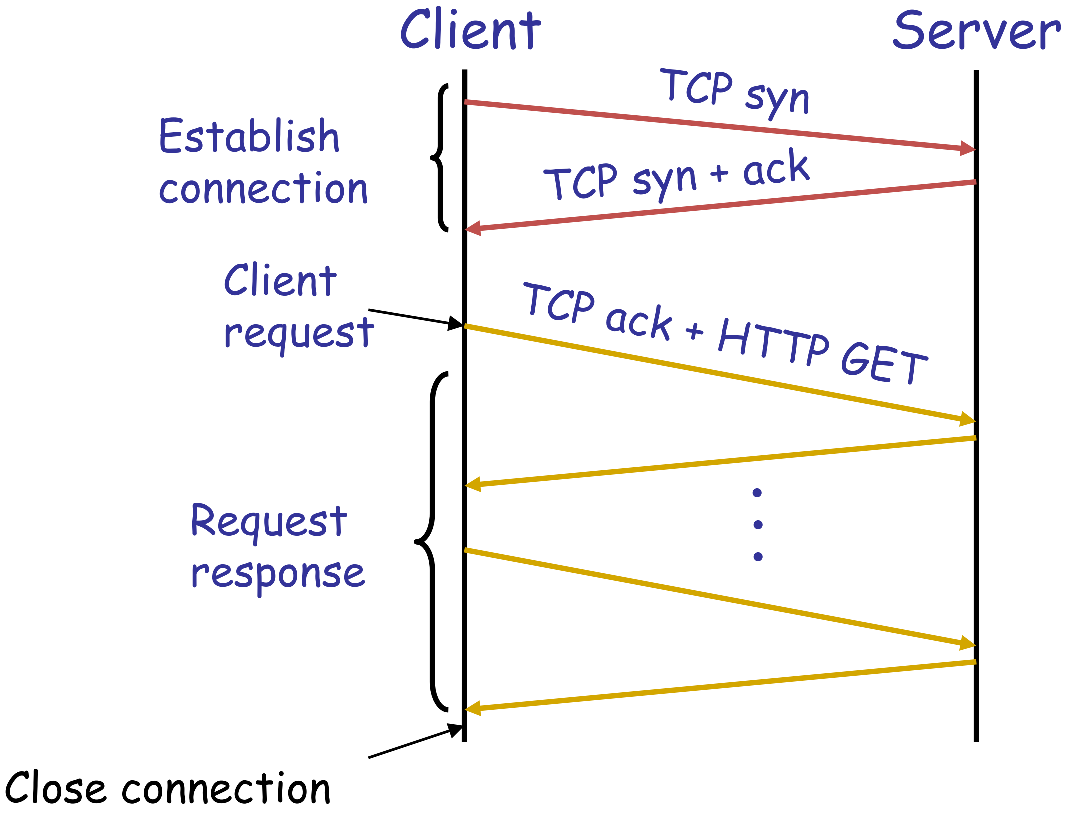
- A unique identifier for an object on WWW
- URL format
  <protocol>://<host>:<port>/<path>?query_string
  - Protocol: method for transmission or interpretation of the object, e.g. http, ftp, Gopher
  - Host: DNS name or IP address of the host where object resides
  - Path: pathname of the file that 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

carriage return line feed indicates end of message

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

# 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

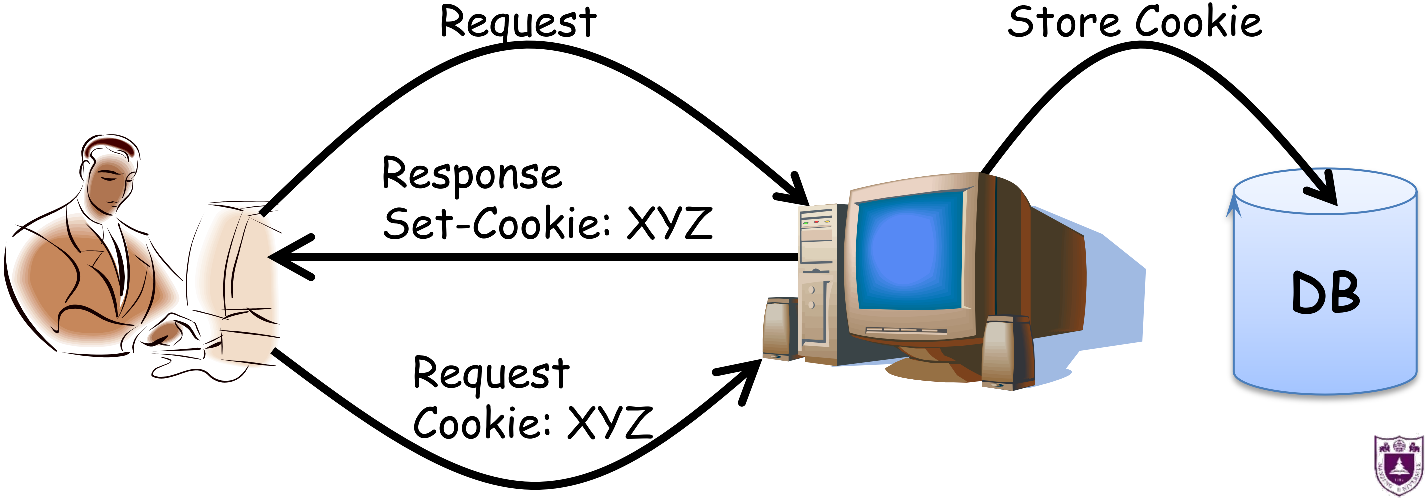
- Each request-response treated independently
  - Servers not required to retain state
- Good: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn't matter
- Bad: Some applications need persistent state
  - Need to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

# Question

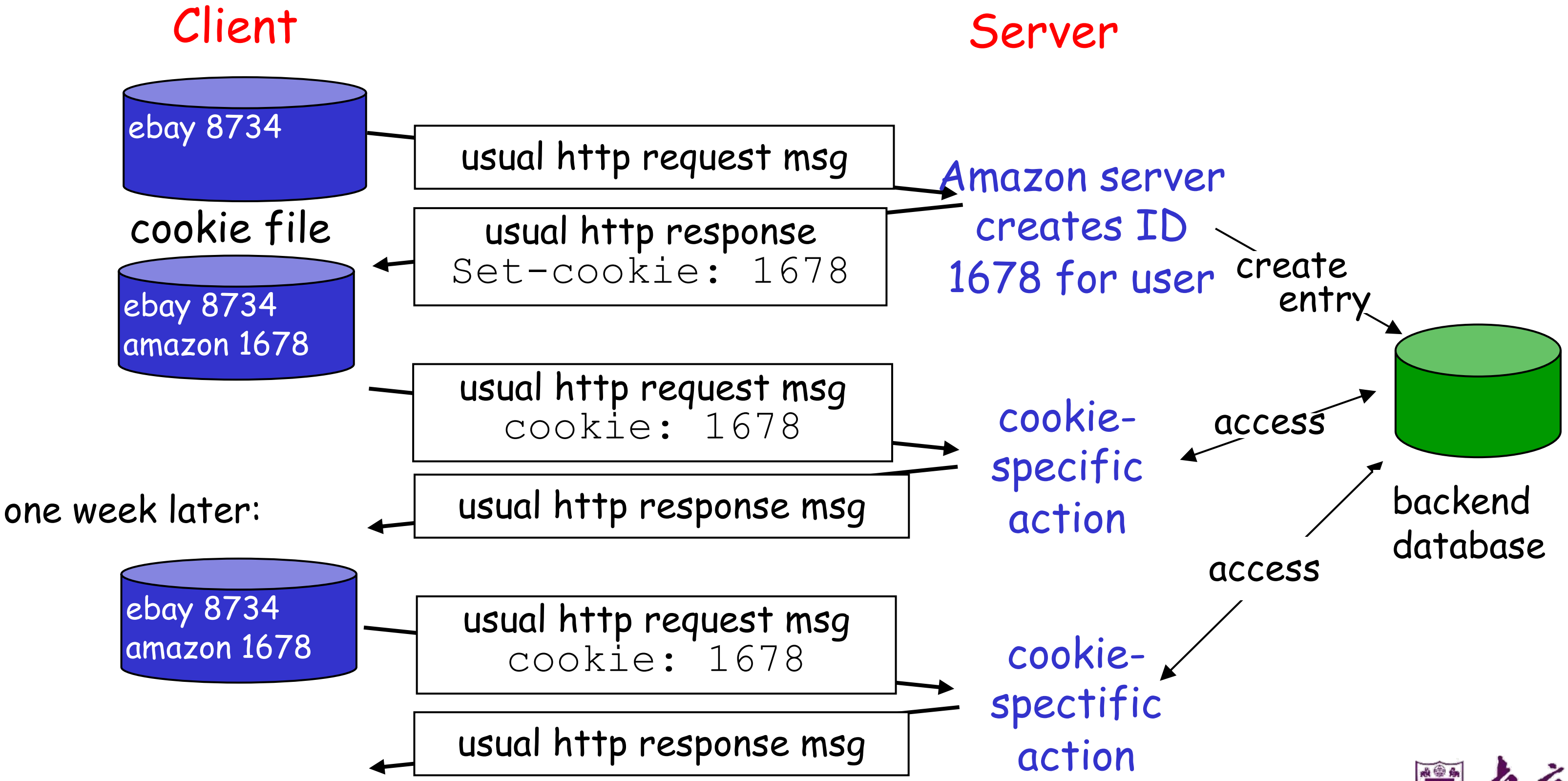
- How does a stateless protocol keep state?

# 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)

**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 identical to low-latency communication!)
  - High availability
- Content provider
  - Happy users (hence, above)
  - Cost-effective infrastructure
- Network (secondary)
  - Avoid overload

# Solutions?

- 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?

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

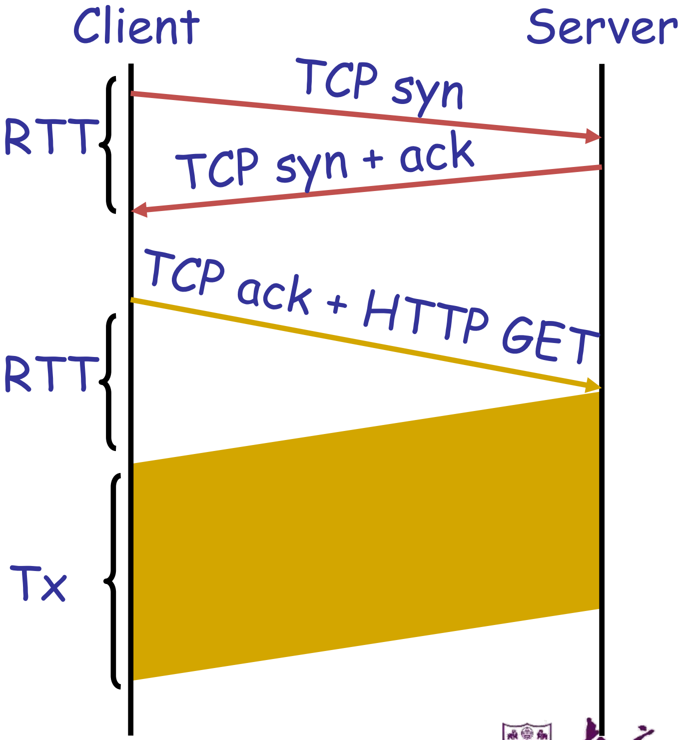
Exploit economies of scale; e.g., webhosting, CDNs, datacenters

# 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+Δ for each object in the HTML file!
  - One more 2RTT+Δ for the HTML file itself
- Doing the same thing over and over again
  - Inefficient

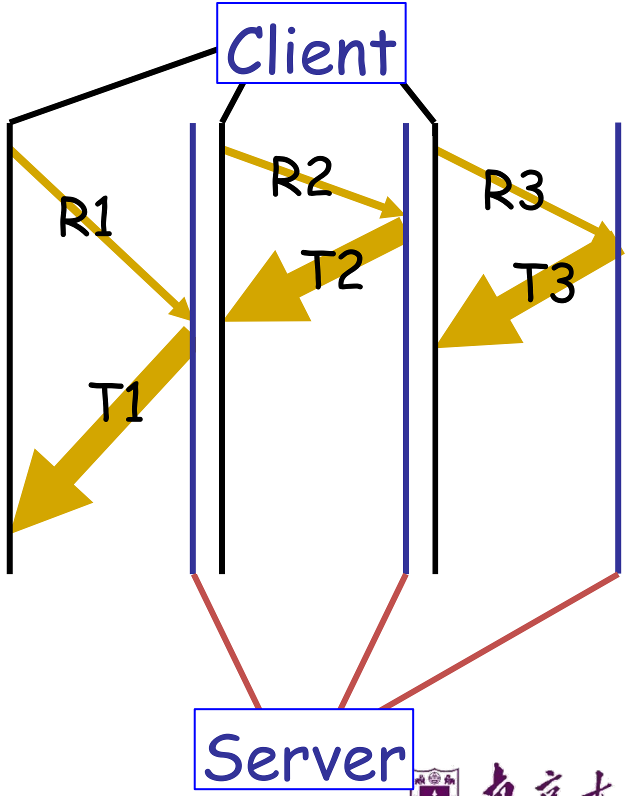
# Concurrent requests and responses

- Use multiple connections in parallel
- Does not necessarily maintain order of responses

➢ Client = ☺

➢ Content provider = ☺

➢ Network = ☹ Why?

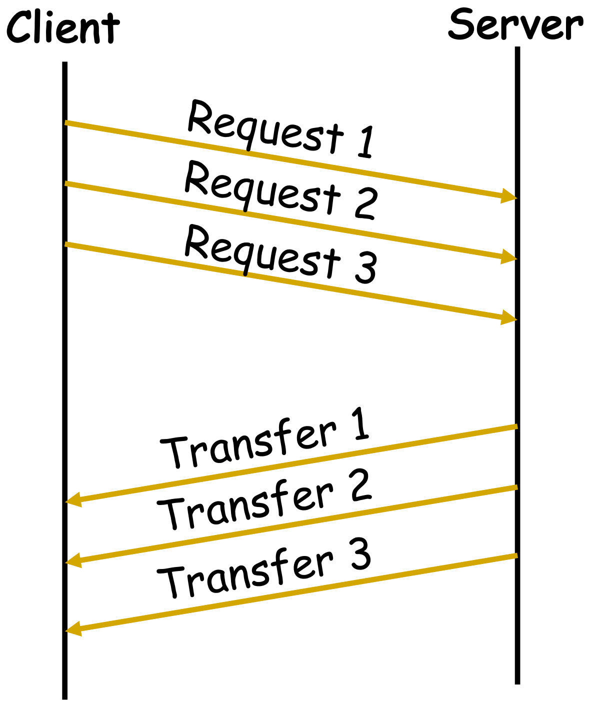


# Persistent connections

- 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

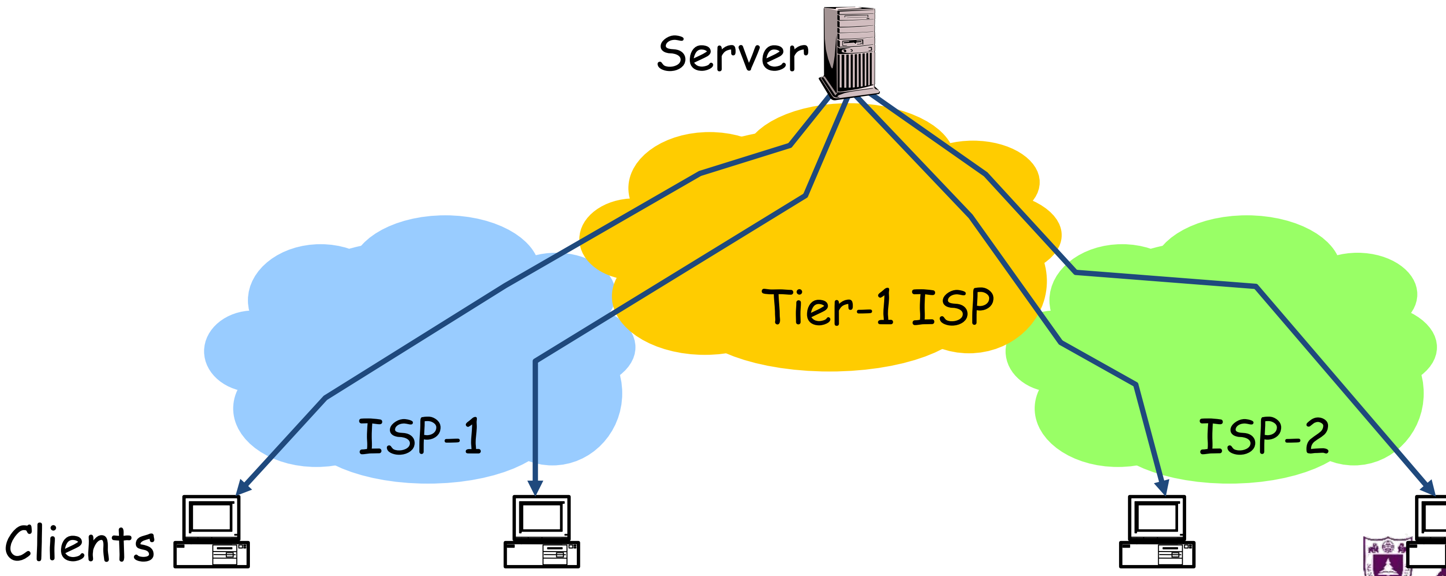
- Modifier to GET requests:
  - If-modified-since – returns “not modified” if resource not modified since specified time
- Response header:
  - Expires – how long it’s safe to cache the resource
  - No-cache – ignore all caches; always get resource directly from server

# Caching: Where?

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

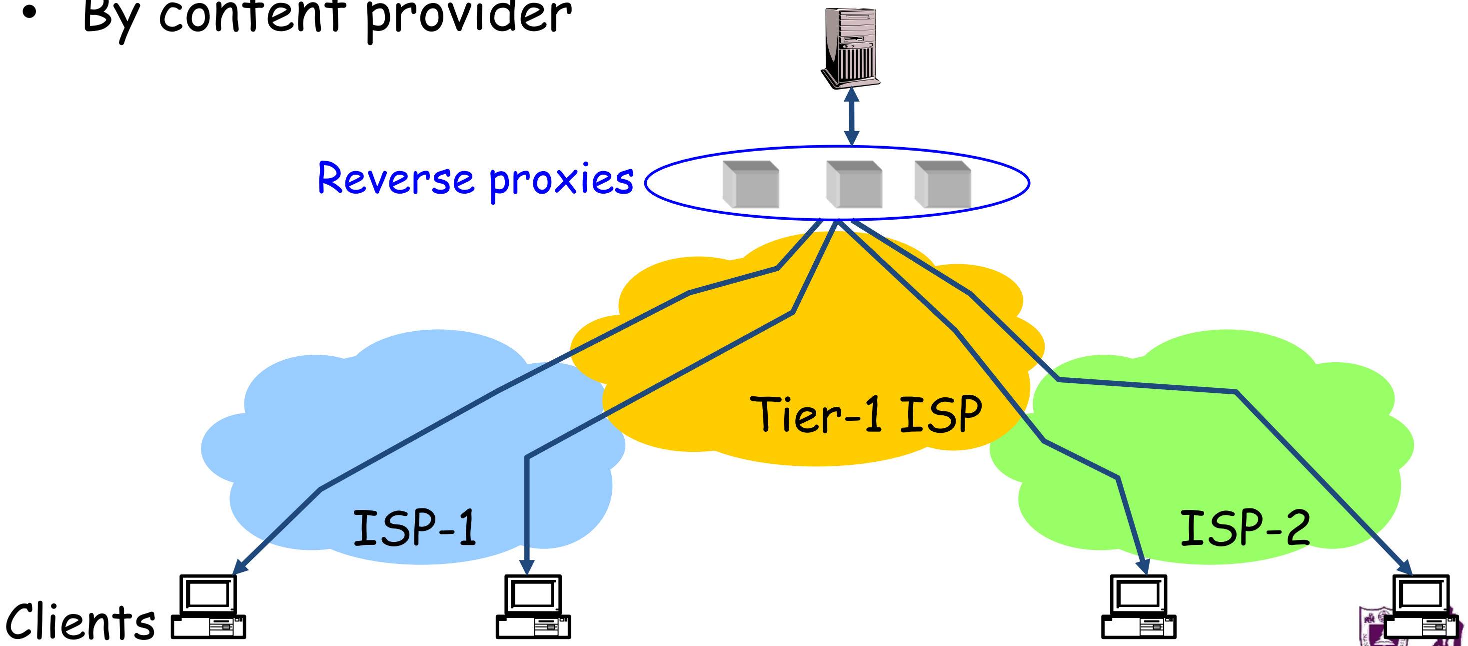
# Caching: Where?

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



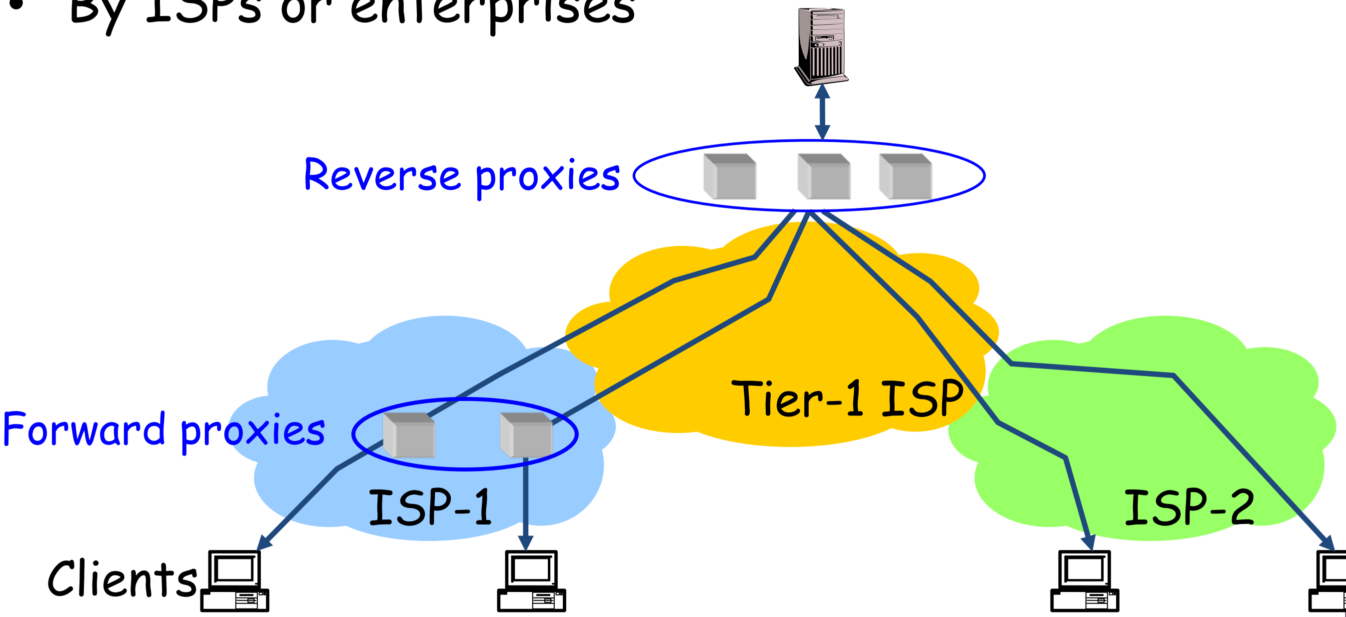
# Caching with Reverse Proxies

- 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

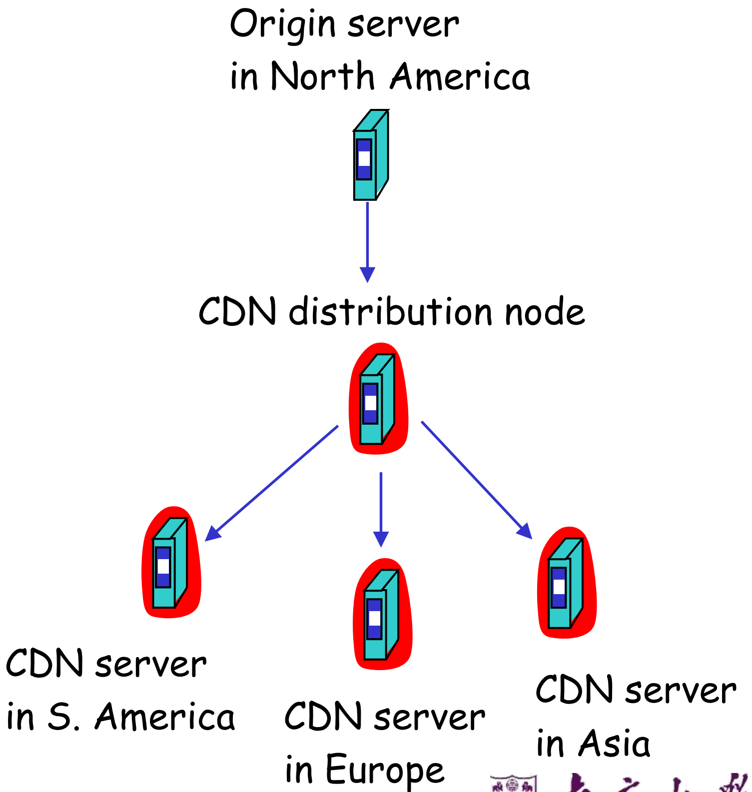
# Internet Applications

- Internet Applications Overview
- Domain Name Service (DNS)
- Electronic Mail
- File Transfer Protocol (FTP)
- WWW and HTTP
- 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



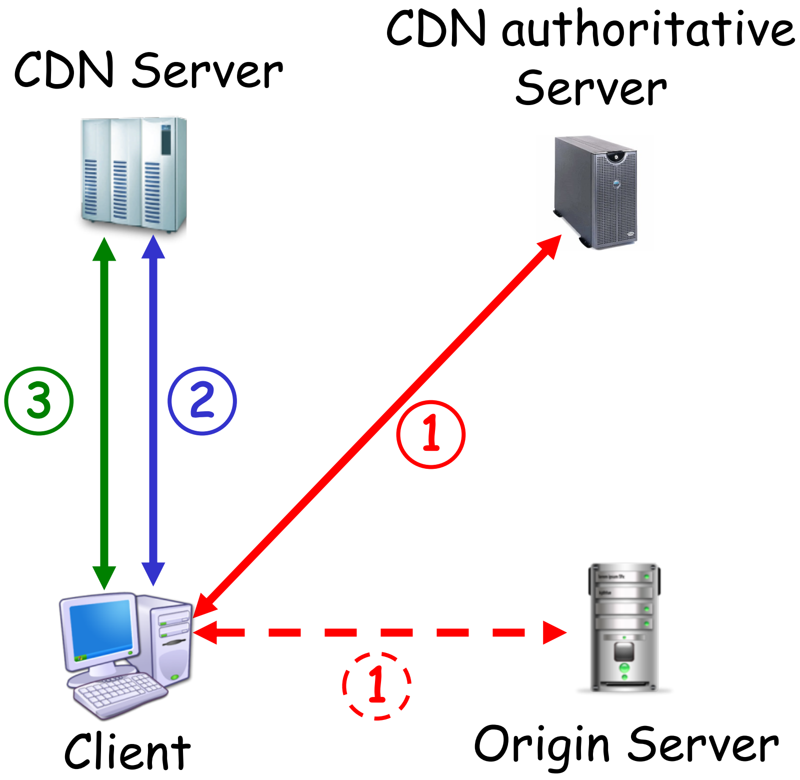
# Content Replication

- 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

# 提问

# Q & A

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/