



# Romain Fontugne

2019 Fall Semester

## Information Network Systems

*The Application Layer*

Last lecture:

### The transport layer

- Multiplexing / Demultiplexing
- UDP
  - Connection-less
  - Best effort
- TCP
  - Connection oriented
  - Reliable communication

### IP Stack

Application
Transport
Network
Link
Physical

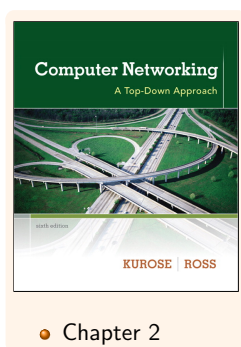
Today's Lecture: Application layer

Creating a Network Application

### 1 Principles of Network Applications

### 2 Protocols

- HTTP
- Email: SMTP, POP3, IMAP
- DNS
- FTP
- P2P Applications

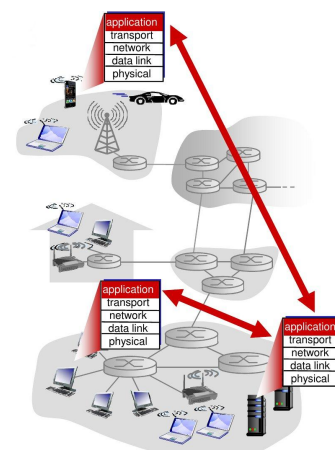


### Application that

- run on (different) end systems
- communicate over network
- e.g., web server software communicates with browser software

### No need to write software for network-core devices

- network-core devices do not run user applications
- data transport is done by lower level protocols



Application Architectures

Client-Server Architecture

### Possible structure of applications:

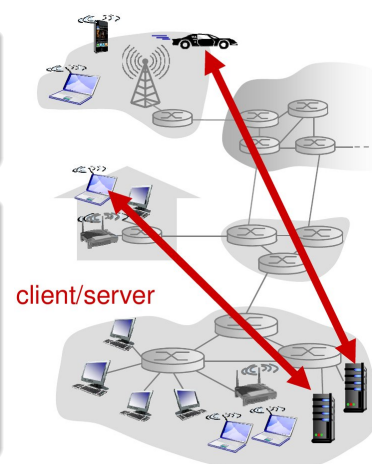
- client-server
- peer-to-peer (P2P)

### Server:

- always-on host
- permanent IP address
- data centers for scaling

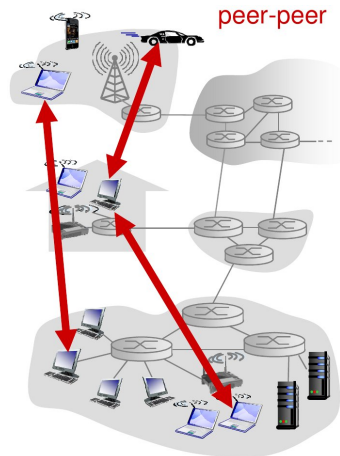
### Clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other



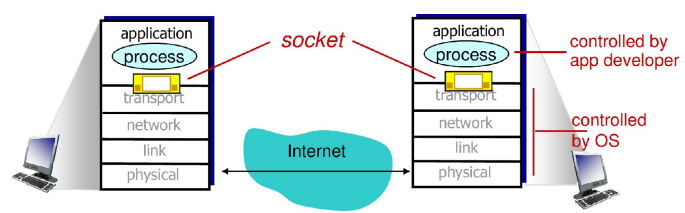
# P2P Architecture

- P2P
- 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
    - complex management



# Sockets

- Two applications communicate via sockets
- Interface between application and network
- Identified by IP address and port number
- The application has little control (and trouble) with the transport-layer



# Application Layer

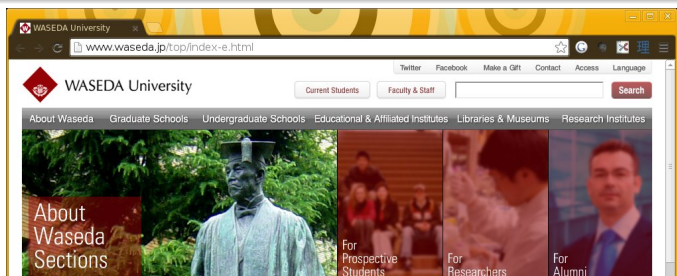
- Application layer protocols define:
- Types of messages exchanged:**
    - e.g., request, response
  - Message syntax:**
    - what fields in messages & how fields are delineated
  - Message semantics:**
    - meaning of information in fields
  - Rules** for when and how processes send & respond to messages
    - Open protocols:**
      - defined in RFCs
      - allows for interoperability
      - e.g., HTTP, SMTP
    - Proprietary protocols:**
      - e.g., Skype

# What transport service does an app need?

- Data integrity**
- Some apps (e.g. file transfer, web transactions) require 100% reliable data transfer
  - Other apps (e.g. audio) can tolerate some loss
- Timing**
- Some apps (e.g. Internet telephony, interactive games) require low delay to be “effective”
- Throughput**
- Some apps (e.g. multimedia) require minimum amount of throughput to be “effective”
  - Other apps make use of whatever throughput they get

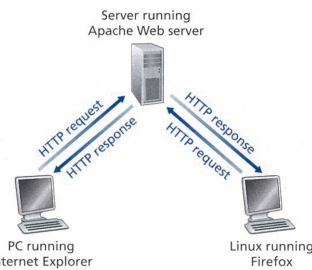
# HTTP: Hyper Text Transfer Protocol

- Webpage and URL
- Webpages consist of a **base HTML file** and several **referenced objects** (e.g. JPEG image, Java applet, video, audio file,...)
  - each object is addressable by a Uniform Resource Locator (**URL**), e.g.,  
http : // www.someschool.edu / someDept/pic.gif  
protocol                      host                      path name



# HTTP overview

- HTTP: Hyper Text Transfer Protocol
- Web's application layer protocol
  - Client/server model:
    - Client:**
      - browser that requests, receives, (using HTTP protocol) and “displays” Web objects
      - e.g. Firefox, Chrome, Safari, Internet Explorer
    - Server:**
      - Web server sends (using HTTP protocol) objects in response to requests
      - e.g. Apache



HTTP uses **TCP**:

- Server is constantly waiting for request on **port 80**
- Client initiates TCP connection (creates socket) to server
- Server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

non-persistent HTTP

- at most one object sent over TCP connection (connection then closed)
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

Non-persistent HTTP

Example:

- User enters URL, `http://www.someSchool.edu/someDept/index.html`
- Webpage contains text and references to 10 jpeg images

- 1 HTTP client initiates connection with server `www.someSchool.edu` on port 80. A socket is created at both client and server
- 2 HTTP client sends an HTTP **request message** to the server indicating that client wants `/someDept/index.html`
- 3 HTTP server receives request message, forms **response message** containing requested object, and sends message
- 4 HTTP server tells TCP to close connection
- 5 HTTP client receives the response message containing html file, display/parse it and finds URLs of 10 images
- 6 For each image, the steps 1-4 are repeated

Persistent HTTP

non-persistent HTTP issues:

- requires **2 RTTs per object** (TCP connection establishment and HTTP request/response)
- OS overhead for each TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP Request Message

HTTP request message

- ASCII text (human-readable format)
- Includes URL and **method type** (GET, POST, HEAD, PUT, DELETE)

```
Request line: [method][sp][URL][sp][Version][cr][lf]
Header lines: [header field name:][sp][value][cr][lf]
Blank line: [cr][lf]
Entity body: [ ]
```

Example: HTTP GET Request

```
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
```

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

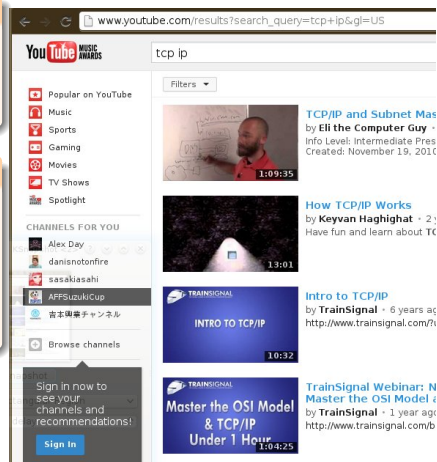
Uploading Form Input

GET method

- Input is added in the requested URL
- e.g. search on youtube

POST method

- Usually used with complex form input (e.g. long text, images, pdf)
- Input is uploaded to server in entity body



Method Types

HTTP/1.1

- GET
  - Obtain object at specified address
- POST
  - Request of a webpage with specific input to form-fields
- HEAD
  - Ask reply with empty message (e.g. debugging)
- PUT
  - Upload file in entity body to path specified in URL field
- DELETE
  - Delete file specified in the URL field

HTTP Response Message

HTTP response message

- ASCII text (human-readable format)
- Includes status code and requested file

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 Code

Some sample codes:

- 200 OK
  - request succeeded, requested object later in this msg
- 301 Moved Permanently
  - requested object moved, new location specified later in the header (Location:)
- 400 Bad Request
  - request msg not understood by server
- 404 Not Found
  - requested document not found on this server
- 505 HTTP Version Not Supported

User-Server State: Cookies

HTTP is a stateless protocol

- HTTP does NOT track states of clients
- Websites implement user sessions on top of HTTP
- Users state is usually kept thanks to cookies



What cookies are used for:

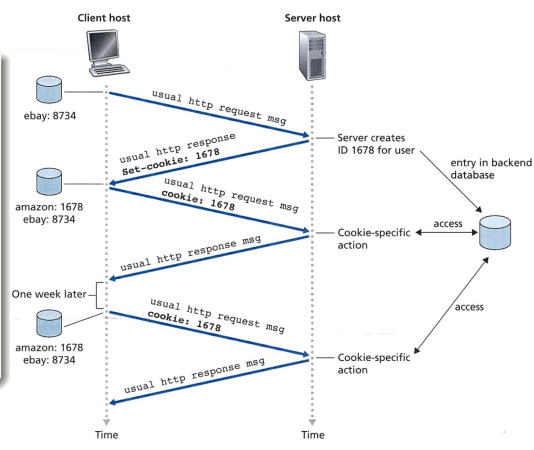
- authorization, shopping carts, recommendations, user session state (Web e-mail), advertising

Cookies consist of four components

- Cookie header line in HTTP response message (Set-cookie: )
- Cookie header line in HTTP request message (Cookie: )
- Cookie file kept on the users end system managed by the browser
- Back-end database at the web site

Cookies: Example

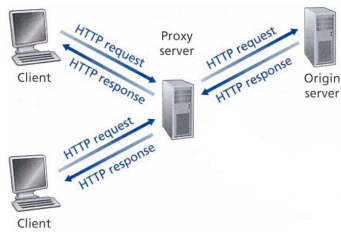
- Client visits amazon for first time
- Amazon creates an ID, attached to the reply
- Client stores the ID in a file with the server host name
- Client provides the ID with all following requests to amazon



Web Caches (Proxy Server)

Goal: Satisfy client request without involving origin server

- User sets browser: Web accesses via cache
- Browser **sends all HTTP requests to cache**
  - Object in cache: cache returns object
  - Else cache requests object from origin server, then returns object to client



Web Caches (Proxy Server)

More about web caching:

- Cache acts as both client and server
- Typically cache is installed by university, company, residential ISP...
- Cache **reduces traffic and response time**

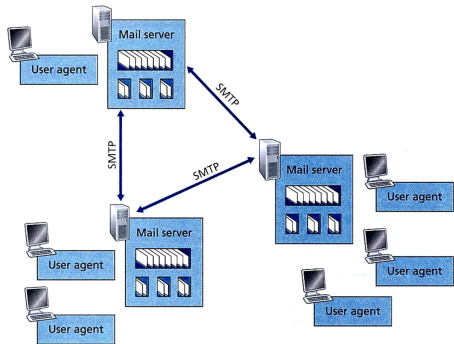
How to know if the cache is up-to-date?

- HTTP defines a **conditional GET** to verify the validity of an object
- Request message header includes "If-Modified-Since: <date>", e.g.
- Server response contains no object if cached copy is up-to-date
- For example, client request:
  - GET /fruit/kiwi.gif HTTP/1.1
  - Host: www.exotiquecuisine.com
  - If-modified-since: Wed, 7 Nov 2013 09:23:24
- Server response:
  - HTTP/1.1 304 Not Modified

Electronic Mail in the Internet

Electronic Mail

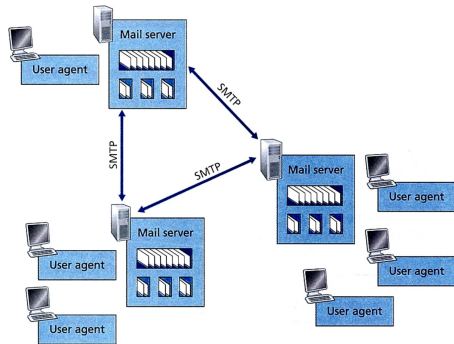
- **Three major components:**
  - User agents
  - Mail servers
  - Simple mail transfer protocol: SMTP



Electronic Mail in the Internet

User Agent

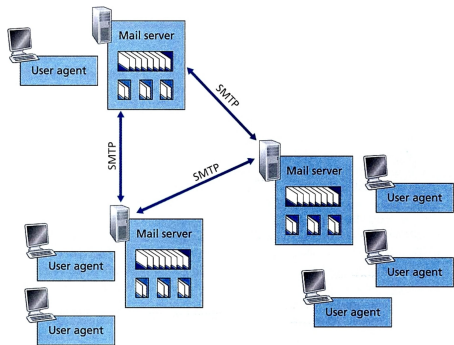
- a.k.a. "mail reader"
- Composing, reading e-mail
- e.g. Outlook, Thunderbird, iPhone mail client...



Electronic Mail in the Internet

Mail Servers:

- **mailbox** contains incoming messages for user
- **message queue** of outgoing (to be sent) mail messages
- **SMTP protocol** between mail servers to send email messages



Electronic Mail: SMTP [RFC 2821]

SMTP: Simple mail transfer protocol

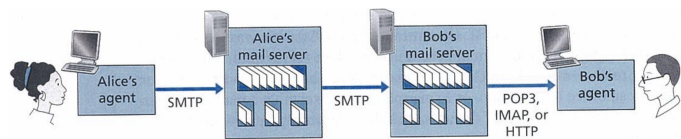
- Uses **TCP** to reliably transfer email message (**port 25**)
- Direct transfer: sending server to receiving server
- **Three phases** of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- **Command/response interaction** (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- Comparison with HTTP:
  - HTTP: pull
  - SMTP: push



Scenario:

Alice sends message to Bob

- Alice uses her user agent to compose a message “to” bob@some school.edu
- Alice’s user agent sends message to her mail server; message placed in message queue
- client side of SMTP opens TCP connection with Bob’s mail server
- SMTP client sends Alice’s message over the TCP connection
- Bob’s mail server places the message in Bob’s mailbox
- Bob invokes his user agent to read message



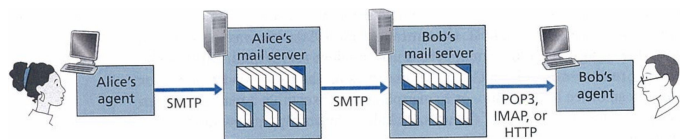
Electronic Mail

SMTP

- Delivery/storage to receiver's server
- Not to the receiver’s computer!

Mail access protocols

- **Mail access protocol:** retrieval from server
  - **POP3:** Post Office Protocol: authorization, download
  - **IMAP:** Internet Mail Access Protocol: more features, including manipulation of stored msgs on server
  - **HTTP:** gmail, Hotmail, Yahoo! Mail, etc.



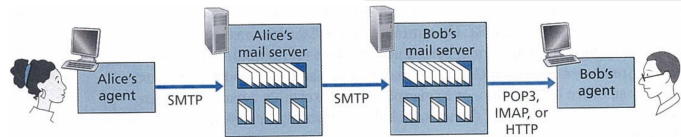
Mail access protocols: POP3 and IMAP

Post Office Protocol ver.3

- Simple mail access protocol
- Authorization and transaction
- 2 transaction modes:
  - “download and delete”: cannot re-read e-mail using another user agent
  - “download and keep”: copies of messages on different user agent
- Stateless across sessions

Internet Mail Access Protocol

- Provides additional features
- Keeps all messages in **one place**: at server
- Allows user to **organize messages in folders**
- **Keeps user state** across sessions:
  - names of folders and mappings between message IDs and folder name



Domain Name System (DNS)

Domain Name System

- Internet hosts can be identified either by their **hostname** or their **IP address**:
  - **Hostname:** www.google.com
  - **IP address:** 74.125.235.144
- **DNS** is a directory service that **maps IP addresses to hostnames**
- **Distributed database** implemented in hierarchy of many **name servers**
- DNS queries use UDP for fast communication (port **53**)

DNS: Services

DNS Services

- **Hostname to IP address** translation
  - www.gmail.com → 74.125.235.86
- **Host aliasing** (alias → canonical names)
  - e.g. www.gmail.com → mail.google.com
- **Mail server aliasing** (alias → mail server)
  - e.g. gmail.com → gmail-smtp-in.l.google.com
- **Load distribution**
  - Replicated Web servers: many IP addresses correspond to one name
  - www.google.com →  $\begin{cases} 74.125.235.83 \\ 74.125.235.80 \\ 74.125.235.82 \\ 74.125.235.81 \\ 74.125.235.84 \end{cases}$

DNS: A Distributed, Hierarchical Database

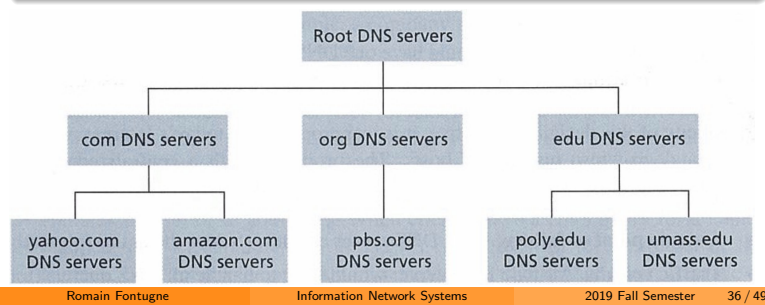
Why not centralized DNS? **doesn't scale!**

- Single point of failure
- Traffic volume
- Distant centralized database
- Maintenance

DNS: A Distributed, Hierarchical Database

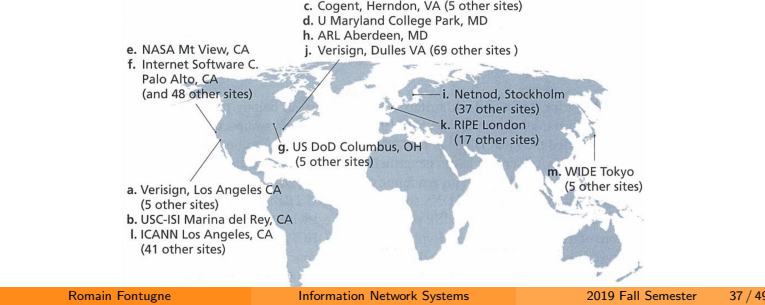
DNS Hierarchy

- DNS uses a **large number of hierarchically organized servers**
- No single DNS server has all of the mappings for all of the hosts in the Internet
- 3 levels: **root**, **top-level-domain (TLD)**, **authoritative servers** (+ local DNS server)



DNS: Root Name Servers

- Contacted by local name server that can not resolve name
- Root name server:
  - Contacts authoritative name server if name mapping not known
  - Gets mapping
  - Returns mapping to local name server
- 13 DNS root “servers” (network of replicated servers, 377 sites as in 2013)



TLD and Authoritative Servers

Top-level-domain (TLD) servers:

- Responsible for .com, .org, .net, .edu, .gov,... and all top-level country domains, e.g.: uk, fr, ca, jp
- e.g. Network Solutions maintains servers for .com TLD
- e.g. 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
- e.g. ns1.google.com, ns2.google.com, ns3.google.com, ns4.google.com, are the authoritative name servers for the domain google.com

Local DNS Name Server

Local DNS (a.k.a. default name server)

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one
- 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

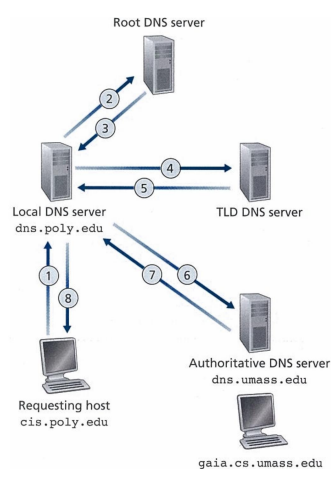
DNS Name Resolution Example

Iterated query:

- Contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”

Example:

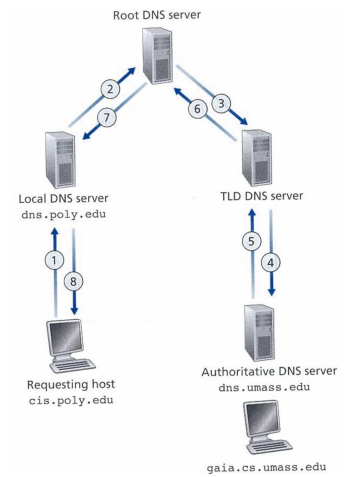
- Host cis.poly.edu wants IP address for gaia.cs.umass.edu
- Root server provides the IP of the TLD server .edu
  - TLD server provides the IP of the Authoritative server responsible for umass.edu
  - Authoritative server provides the IP for gaia.cs.umass.edu



DNS Name Resolution Example

Recursive query:

- Contacted server forward the request
- Put burden on contacted name server
- More load at upper levels of hierarchy



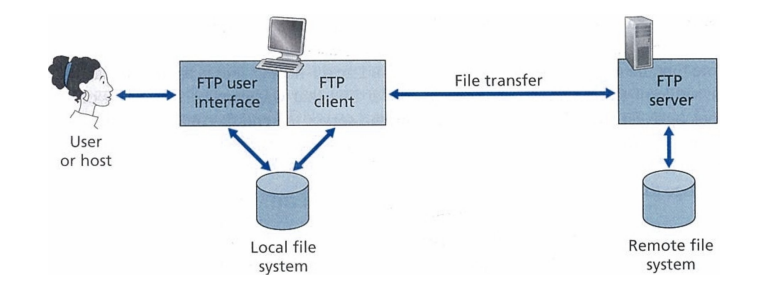
# DNS: Caching

## DNS servers cache responses

- Once (any) name server learns mapping, it **cached** mapping
- Cache entries timeout (disappear) after some time (TTL)
- TLD servers typically cached in local name servers (thus root server are not visited all the time)
- Cached entries may be out-of-date:
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire

→ Selecting TTL value is a trade-off between performance (using cached data/less queries) and up-to-date data

# File Transfer Protocol (FTP)



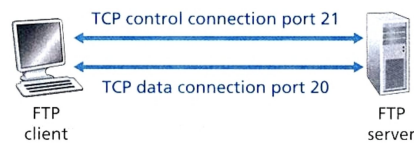
## FTP: File Transfer over TCP

- Transfer file** to/from remote host
- Client/server model
  - Client: side that initiates transfer (either to/from remote)
  - Server: remote host
- FTP server listen on **port 21** (port 20 is used for file transfer)

# File Transfer Protocol (FTP)

## FTP uses two types of connection

- Control connection:** Transmit control information between the two hosts (user authentication, password, commands to change remote directory, put or get files)
- Data connection:** Transfer files



# Peer-to-Peer Architecture

## In P2P networks:

- No always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses

## Examples:

- File distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)

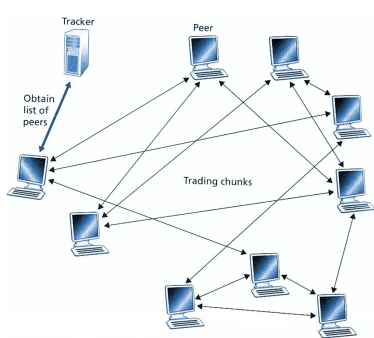
# P2P File Distribution: BitTorrent

## Why P2P for file distribution?

- In client-server model, enormous burden on the server
- In P2P model, distributed peers share the load for distributing the file

## BitTorrent

- File divided into 256Kb **chunks**
- Torrent:** group of peers exchanging chunks of a file
- Tracker:** tracks peers that belongs to the torrent



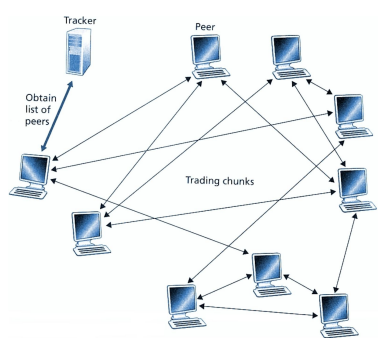
# P2P File Distribution: BitTorrent

## Peer joining torrent

- Has no chunks yet
- Registers with tracker and get a list of peers ("neighbors")

## File sharing

- Downloaded chunks are uploaded to other peers
- Peers may leave so it may connect to new neighbors
- Neighbors periodically ask list of downloaded chunks
- Download rarest chunks first





## The Application Layer: Summary

### Today's lecture covered:

- HTTP
- Email
- DNS
- FTP
- P2P

### Next week: Midterm exam!

- Covers all layers of the protocol stack  
From Physical to Application layer
- Goal: check that you know the basics for each layer
- Less than 20 short questions

### Then we'll see:

Network management:  
Monitoring network performance,  
fault, security...

### Today's important points

- Client/server vs. P2P
- HTTP
- DNS
- Common port numbers