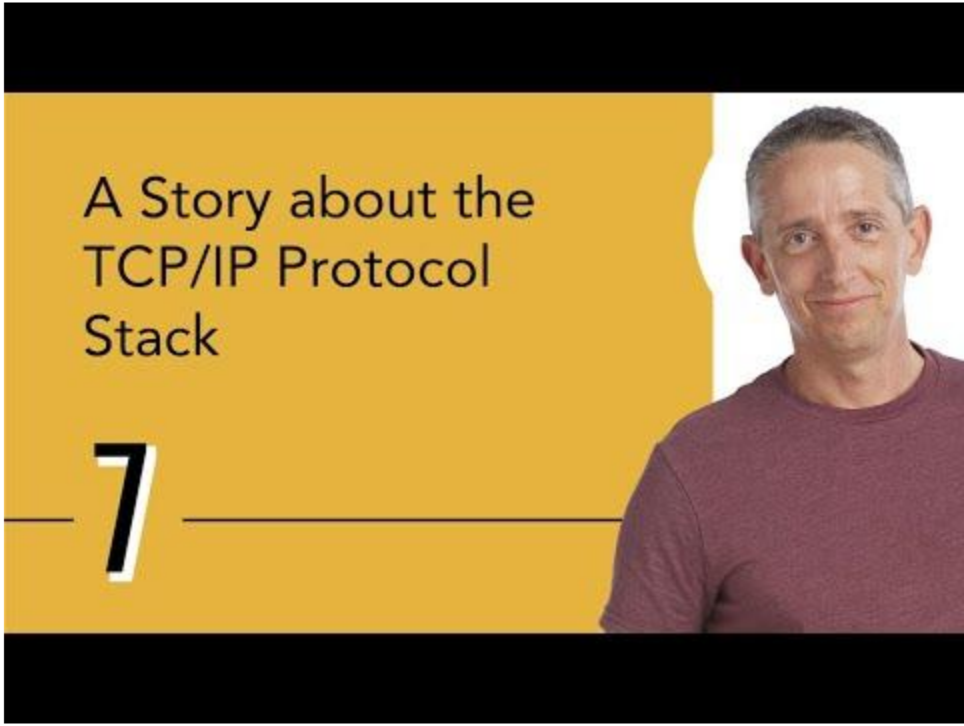
Computer Networks

Service Model and Protocol Architecture

Jianping Pan
Fall 2022



* interactive undersea cable map https://www.submarinecablemap.com/

First things first

- Busy Friday!
  - A1 due today 5pm through brightspace
  - P1 released already
    - Simple Web Server (SWS)
  - T2 today: spec-go-thru, Q&A, simple design
  - W1 released already
  - A2 will be released today
    - L2 (next *Monday*/Tuesday/Wednesday): HTTP

* no in-person labs on next monday sept 19; video provided for drop-in catch-up

Last few lectures

- So far, “nuts and bolts” views of the 'Net
  - Internet access technologies
    - over phone/cable/power/fiber lines
    - Ethernet
    - wireless
  - Internet backbone technologies
    - fiber, satellite
  - Internet evolution and state-of-the-art
  - UVicNet, BCNET, CA*Net4

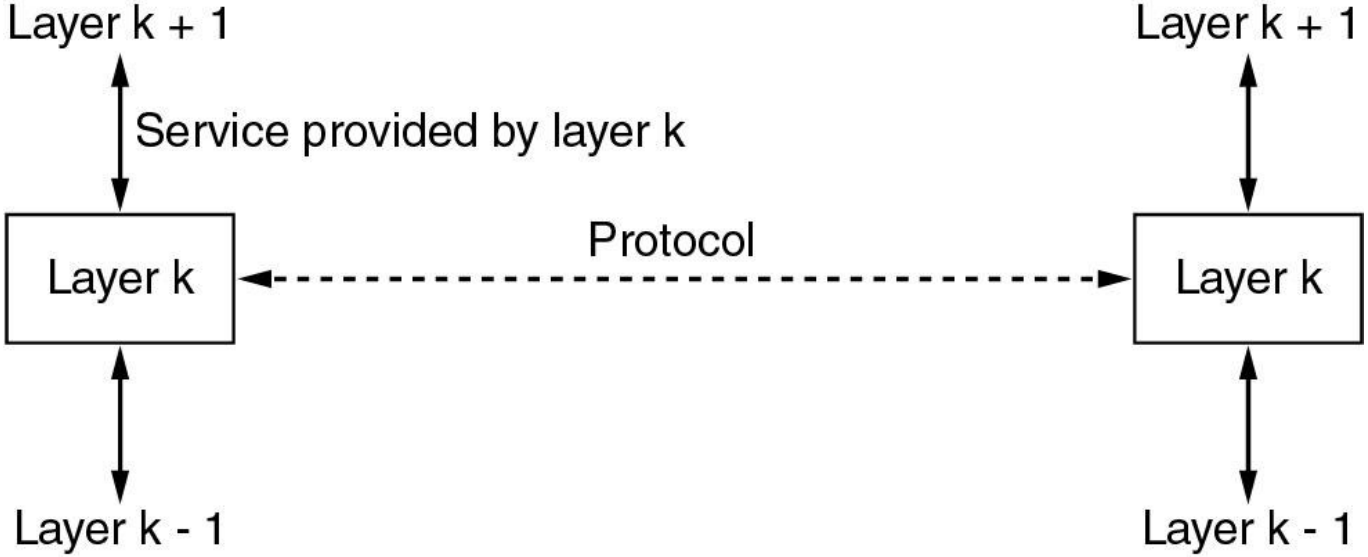
* anyone using starlink?

Today's topic

- How does the Internet really work?
  - network architectures
    - layered structures
  - network services
    - between adjacent layers in the same node
  - network protocols
    - between the same layer in different nodes
- Internet service models
  - client-server model
  - client-server programming

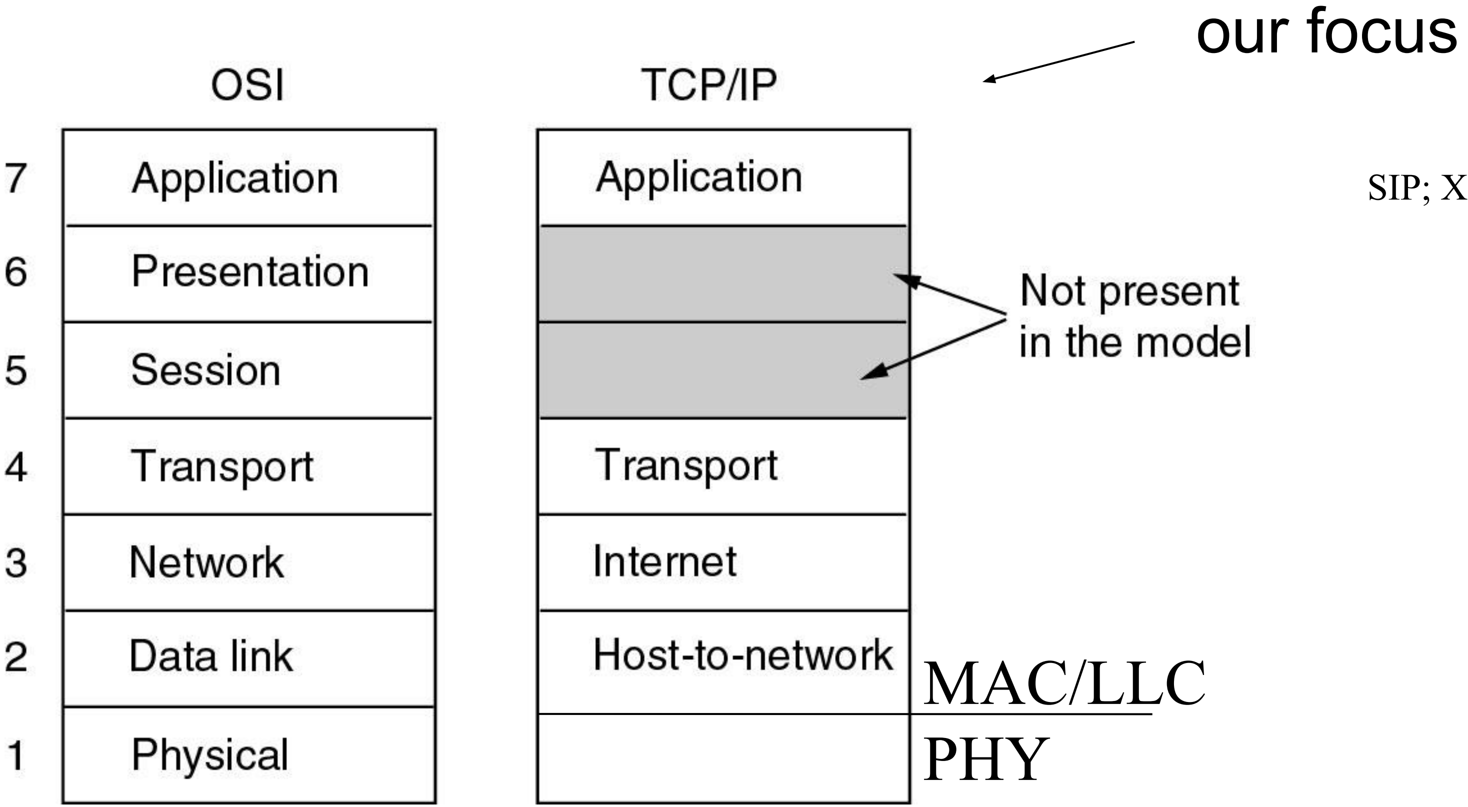
Network architectures

- Layered architecture (Q: why layered?)
  - service vs protocol



* layering in operating systems (CSc360) vs that in computer networks (CSc361)

OSI and TCP/IP models



	OSI	TCP/IP
7	Application	Application
6	Presentation	
5	Session	
4	Transport	Transport
3	Network	Internet
2	Data link	Host-to-network
1	Physical	

our focus

Not present in the model

SIP; XDR(RPC); etc

MAC/LLC

PHY

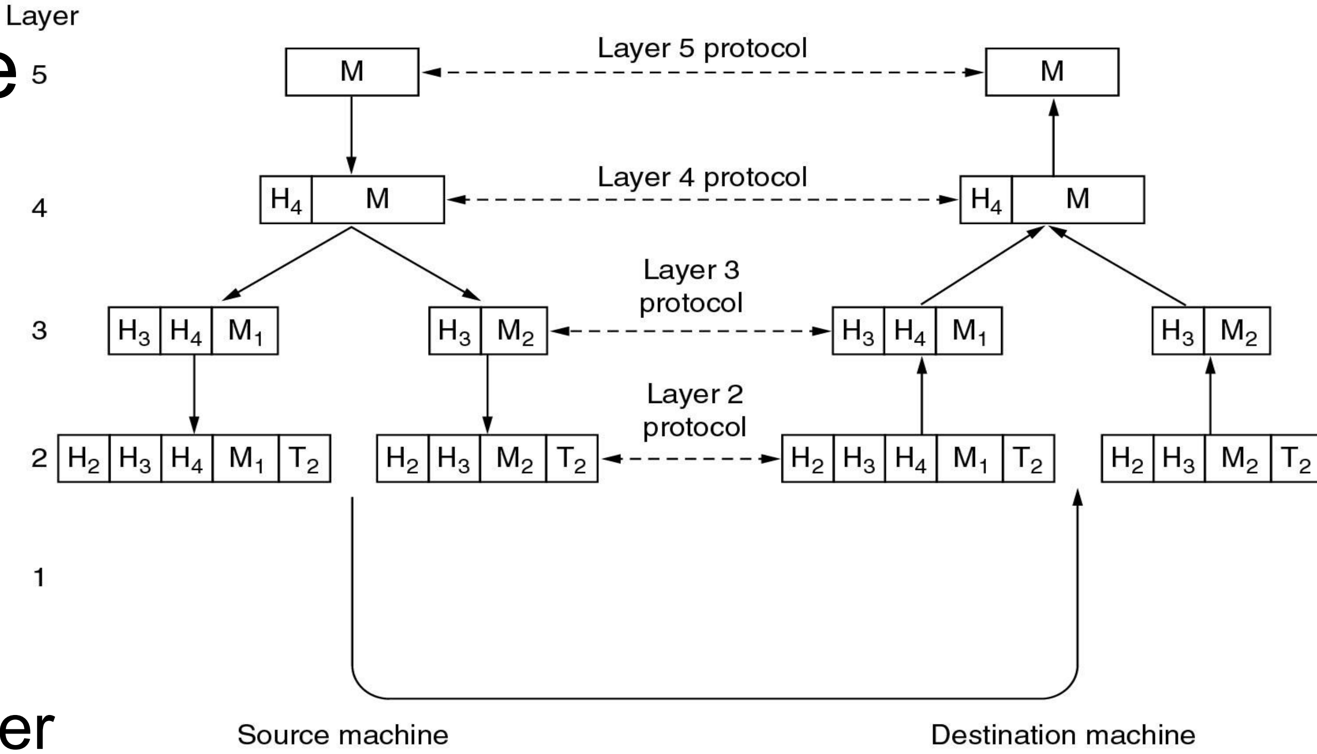
* arpanet’s imp and 1822?

Q: where're the missing layers?

Protocol hierarchies example

- HTTP message
- TCP segment
- IP packet
  - $M_1+M_2=M$
- Ethernet frame
- Bit stream

  H: header; T: trailer



* header vs trailer?

Network services

- Connection-oriented vs connectionless
  - connection establishment
  - data transfer
  - connection release
- Reliable vs unreliable
  - error checking
  - error correction
  - error recovery

Q: reliable services always connection-oriented?

discussion on blackboard

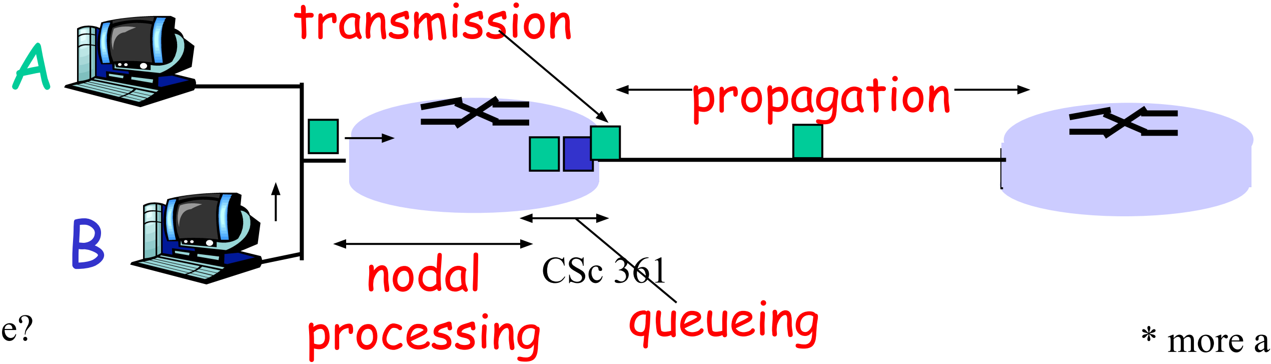
Switching technologies

- Circuit switching
  - e.g., telephone network
- Packet switching
  - virtual circuit
    - e.g., ATM
  - datagram
    - e.g., the Internet

Q: IP/ATM/SONET/WDM?

Link characteristics

- Speed (bandwidth)*: bit-per-second
- Delay: millisecond
  - transmission delay: packet length / link speed
  - propagation delay: travel distance / signal speed
  - processing delay
  - queuing delay: the most complicated one



* the tollbooth example?

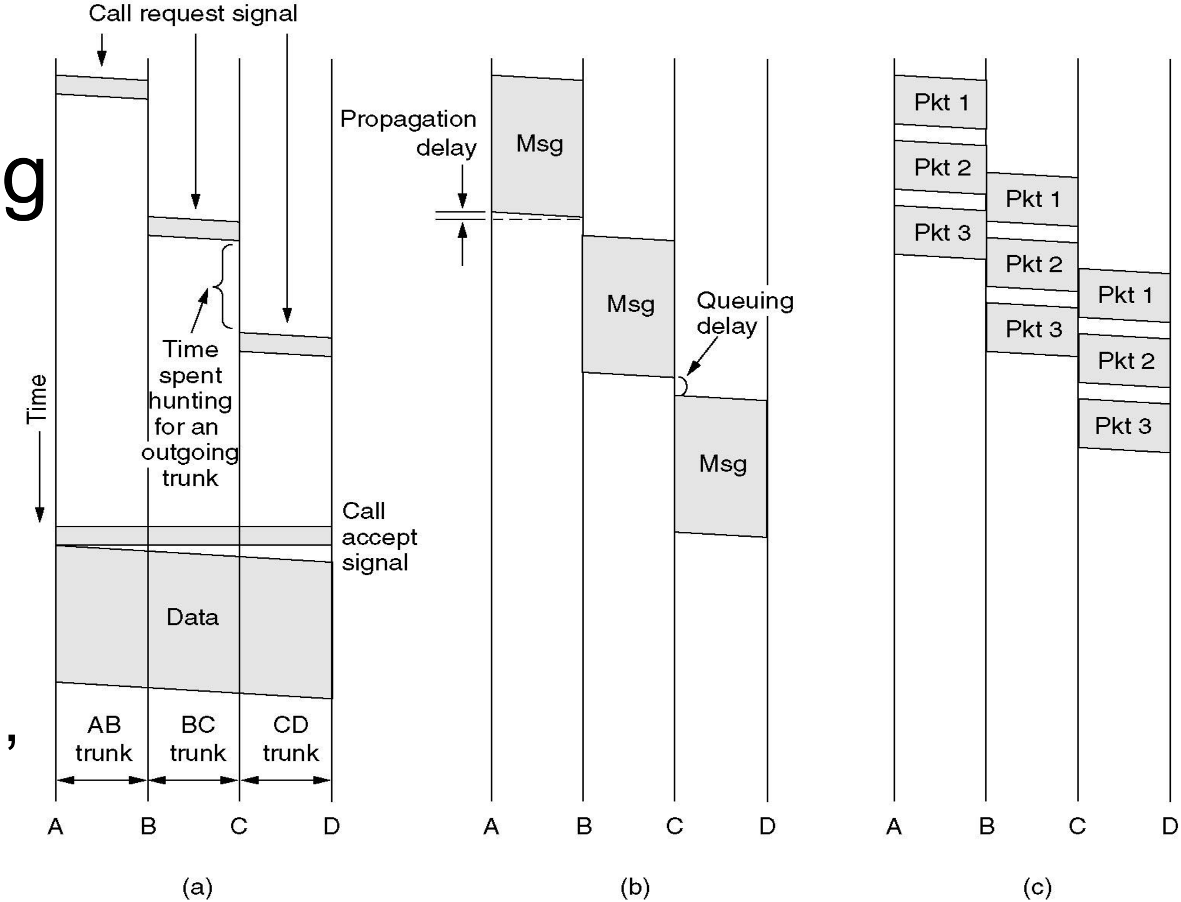
* more accurately, data rate

More on link characteristics

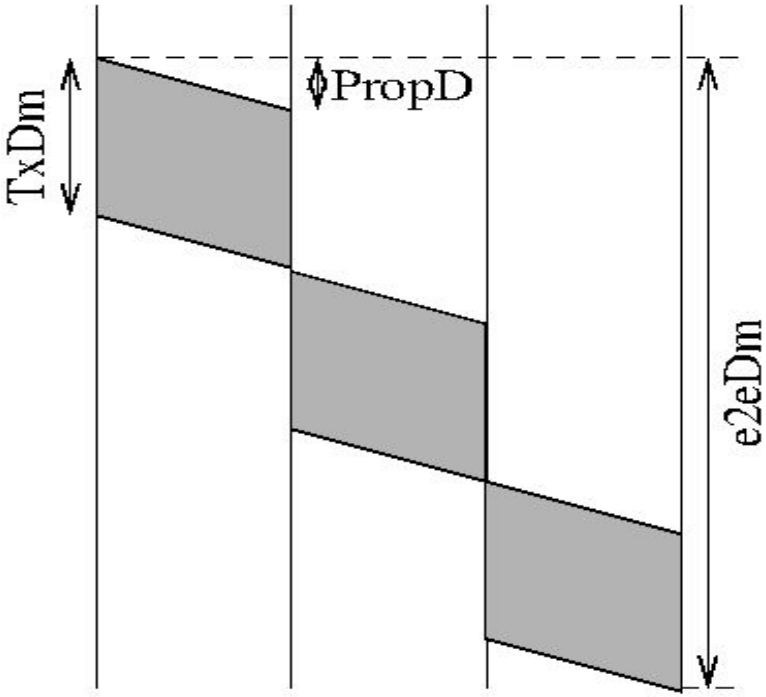
- Loss: percentage
  - transmission error
  - congestion loss
    - router buffer
      - packets enqueue when output is busy
      - packet dequeue when output is idle
    - if buffer is full
      - some packets have to be dropped

- Circuit switching
- Message switching
- Packet switching
  - Internet: store-and-forward packet switching

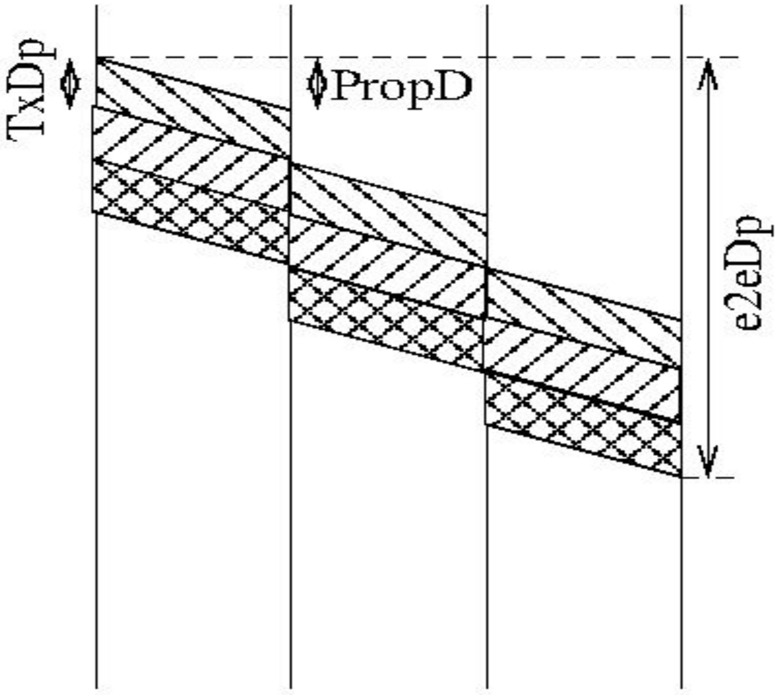
Q: transmission, propagation, processing, queuing delay?



calculation on next slide



e2eDm=#hops*(TxDm+PropD)
=3*(TxDm+PropD)
=3*TxDm+3*PropD



e2eDp=#hops*(TxDp+PropD)
+(#pkts−1)*TxDp
=3*(TxDp+PropD)+2*TxDp
=5*TxDp+3*PropD
=~1.66*TxDm+3*PropD

* NOTE: processing delay ignored in the calculation above!

Internet Protocol Suite

- "Hourglass" model
  - application: telnet, ftp, email, Web, VoIP, ...
    - Web/HTTP: a client-server application layer protocol
  - transport: TCP, UDP, RTP, SCTP
  - network: IP
  - subnetwork: Ethernet, ATM, FDDI, PPP, FR, ...
- "Everything over IP"
- "IP over everything"

everything over HTTP?

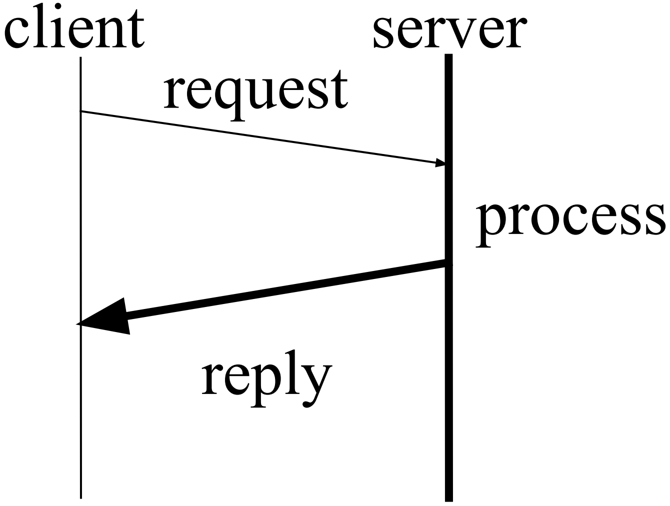
Service models

- Client-server model
  - server: services at well-known socket (WKS)
  - client: request services from anywhere!
  - client-server: request-reply transactions
- Later, client-*intermediary*-server model
  - web caching and content distribution
- In CSc466, peer-to-peer model
  - client/server-server/client

DDoS?

Client-server model

- Server
  - a process (running program)
  - on a (server) computer
  - (hosted in a server farm)
  - waiting for incoming requests
    - process and reply



Q: many clients?

- Client
  - a process on a client computer making requests

Client-server programming

- E.g., socket API (system calls)
  - Client
    - socket()
    - connect()
    - send()
    - recv()
    - close()
  - Server
    - socket()
    - bind()
    - listen()
    - accept()
    - recv()
    - send()
    - close()

* CSc360: system call vs API

details in tutorial 1 (t1) on Python API

Socket API in C

- int socket(int *domain*, int *type*, int *protocol*);
  - domain
    - PF_INET (Internet protocol family), and others
  - type
    - **SOCK_STREAM (supported by TCP)**
    - SOCK_DGRAM (supported by UDP)
    - and others …
  - protocol
    - normally implied by socket type

in Python:
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

* https://docs.python.org/3/library/socket.html

Service offered by TCP

- Service offered by TCP
  - reliable
  - in-sequence
  - stream-like
  - data transfer
- TCP protocol mechanisms (stay tuned!)
  - connection management
  - flow, error, congestion control

Socket, IP address, port number

- int **bind** (int *sockfd*, struct sockaddr *my_addr*, socklen_t *addrlen*);
  - struct sockaddr_in {short int ***sin_family***; unsigned short int ***sin_port***; // 16-bit port# struct in_addr ***sin_addr***; // 32-bit IP address unsigned char ***sin_zero***[8];};
  - struct in_addr {unsigned long *s_addr*;};
- /etc/services, /etc/hosts, DNS

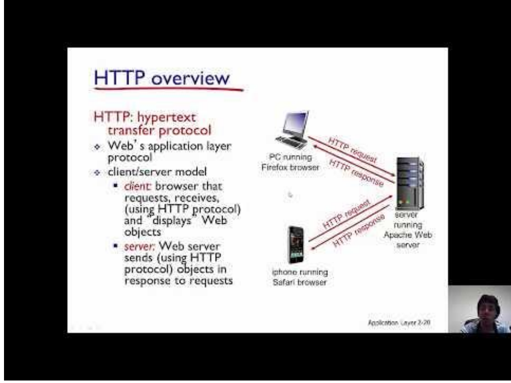
in Python, much simplified/limited:

```
HOST = ''
PORT = 50007
s.bind((HOST, PORT))
```

This lecture

- Internet architecture
  - architecture, services and protocols
- Internet service models
  - client-server model
  - introduction to socket API
- T2 today 1:30pm
  - P1 spec go-through
  - Q&A, and a simple design

Next lectures



HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

PC running Firefox browser
HTTP request
HTTP response
server running Apache Web server
HTTP request
HTTP response
iphone running Safari browser

Application Layer 2-20

- HTTP
  - read K&R4: Computer Networking
    - Chapter 2

- Explore further
  - socket programming tutorial
    - http://beej.us/guide/bgnet/
  - Python socket tutorial: non-blocking socket
    - https://docs.python.org/3/library/socket.html