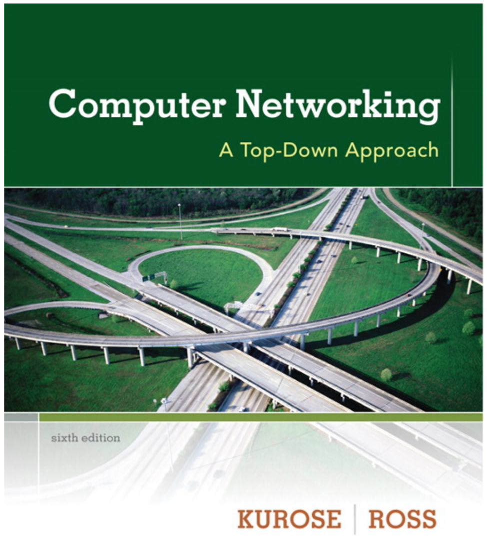
Chapter 2
Application Layer



Chapter 2: Application layer

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 Socket programming with TCP

2.8 Socket programming with UDP

Chapter 2: Application Layer

- Conceptual + implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - socket API
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS

Some network apps

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Netflix, Hulu)
- online social networks
- voice over IP
- real-time video conferencing
- cloud computing
- mobile apps
- …

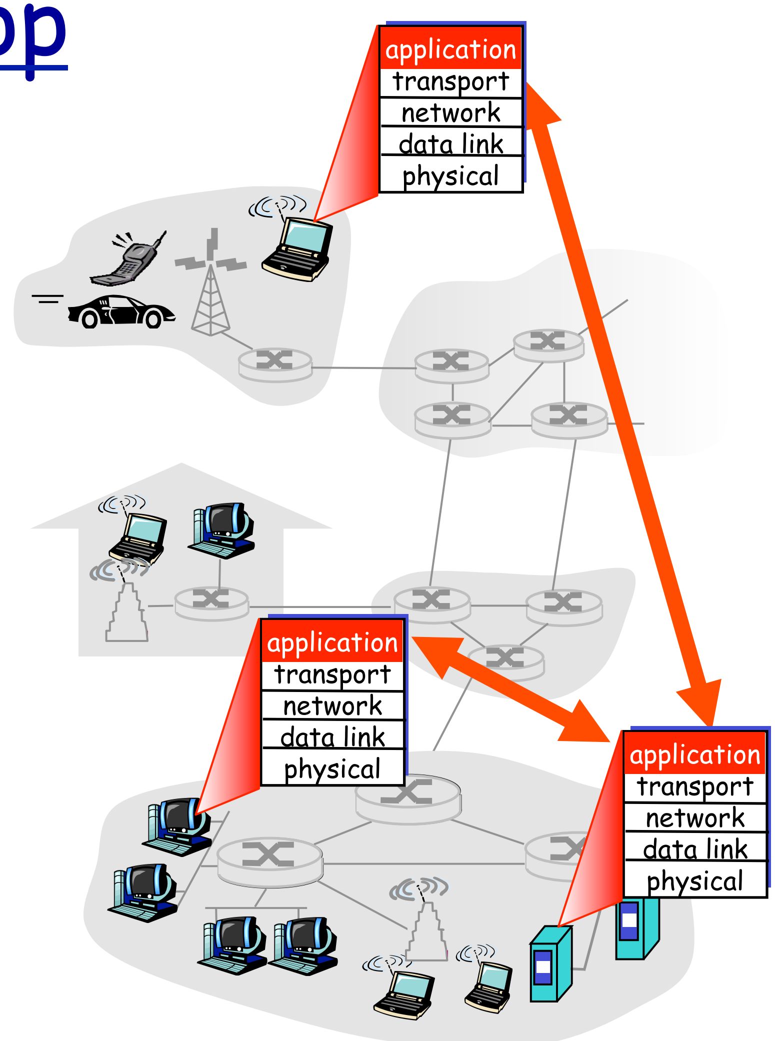
Creating a network app

write programs 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

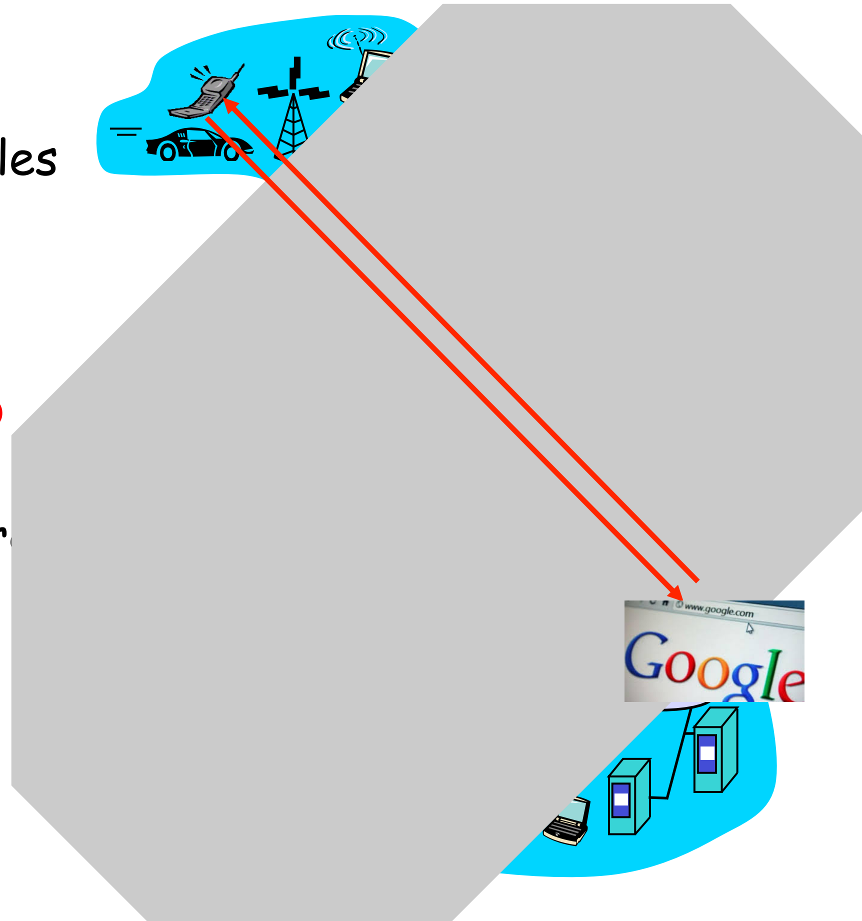
- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



Interner "service" view - revisited

From an application developer's point-of-view, it seems like an API.

- communication infrastructure enables distributed applications:
  - Web, VoIP, email, games, e-commerce, file sharing
- communication services provided to applications:
  - reliable data delivery from sour to destination
  - "best effort" (unreliable) data delivery



- Analogy: Postal Service.

Chapter 2: Application layer

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

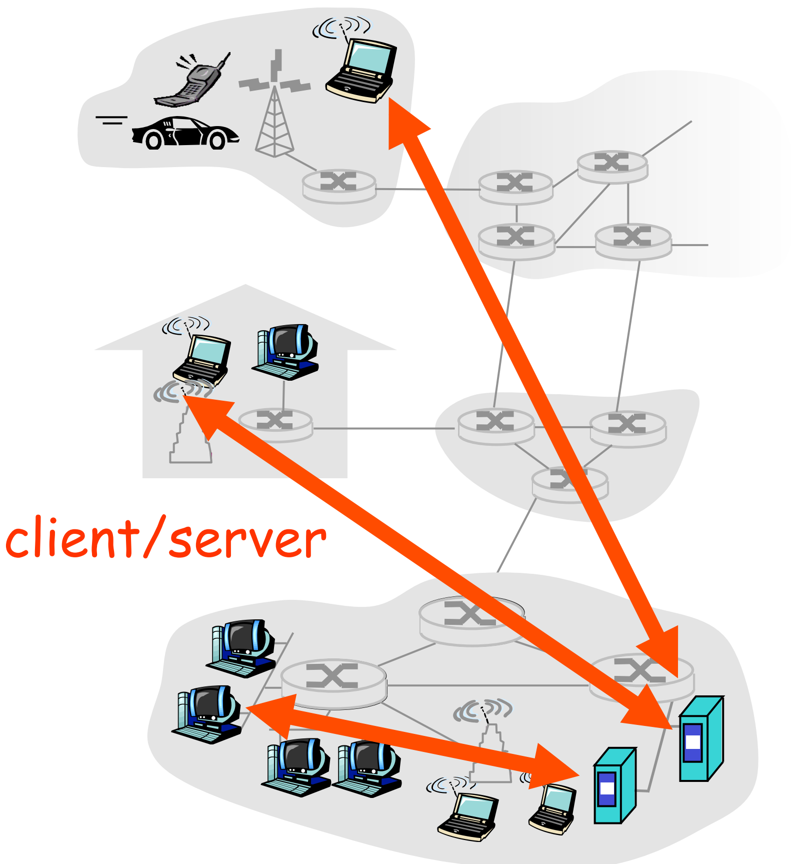
2.7 Socket programming with TCP

2.8 Socket programming with UDP

Application architectures

- client-server
- peer-to-peer (P2P)
- hybrid of client-server and P2P

Client-server architecture



server:

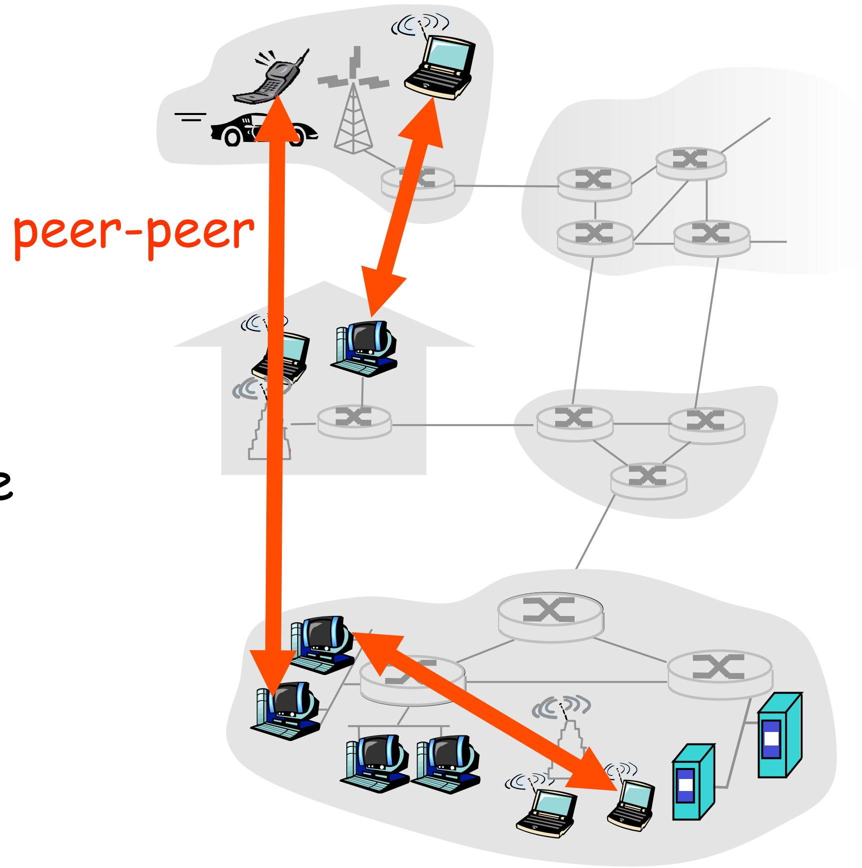
- always-on host
- permanent IP address
- data centers for scaling

clients:

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

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses
- peers request service from other peers, provide service in return to other peers
  - *self scalability – why?*
  - A: new peers bring new service capacity, as well as new service demands



Hybrid of client-server and P2P

Skype

- voice-over-IP P2P application
- centralized server: finding address of remote party
- Peer-peer connection: direct (not through server)

Instant messaging

- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

Processes communicating

process: program running at a host.

- within the same host, two processes/threads communicate using inter-process communication (IPC defined by OS).
  - E.g. methods for message passing, shared memory, synchronization
- processes in different hosts communicate by exchanging messages

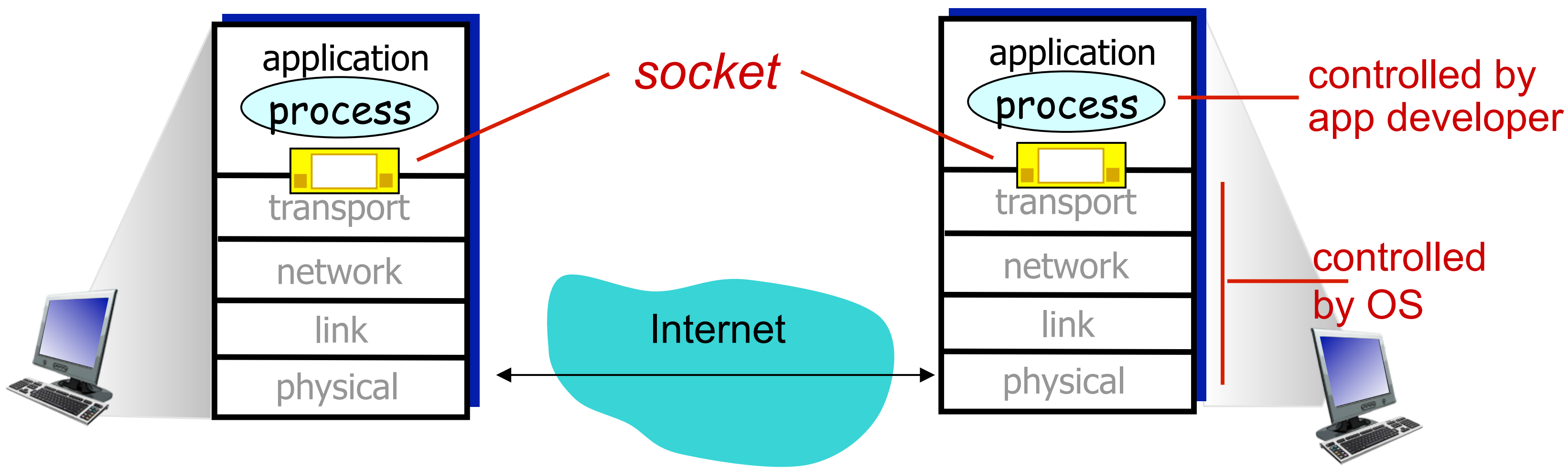
client process: process that initiates communication

server process: process that waits to be contacted

- BTW, applications with P2P architectures have client & server processes

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* No, *many* processes can be running on same host
- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to odysseas.calit2.uci.edu web server:
  - IP address: 128.195.185.112
  - Port number: 80

Analogy to postal service…

App-layer protocol defines

- 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

public-domain protocols:

- defined in RFCs
- allow for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

What transport service does an app need?

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, web transactions) require 100% reliable data transfer

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 ("elastic apps") make use of whatever throughput they get

Security

- encryption, data integrity, …

Transport service requirements of common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- connection-oriented: setup/ teardown required between client and server processes
- full-duplex
- reliable transport between sending and receiving process
- flow control: sender won't overwhelm receiver
- congestion control: throttle sender when network overloaded
- does not provide: timing, minimum throughput guarantees, security

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Securing TCP

TCP & UDP

- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer

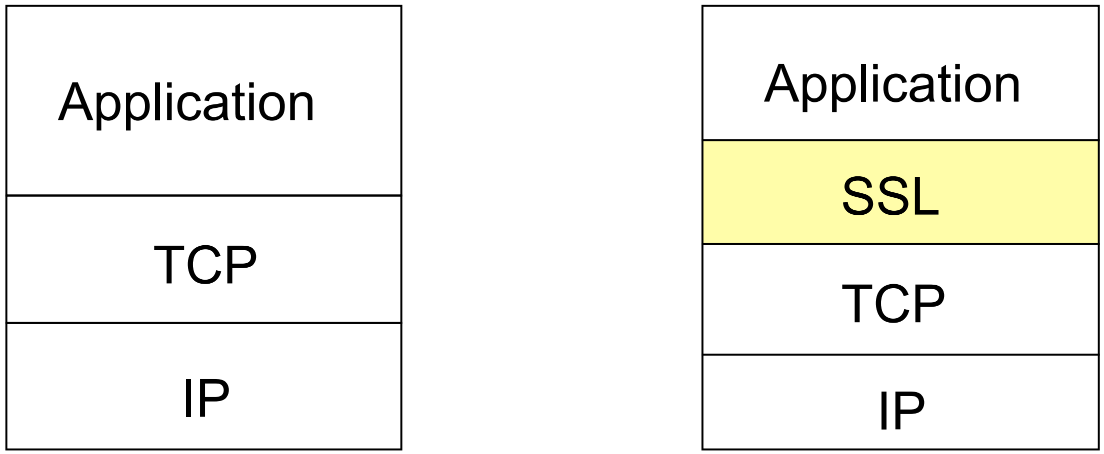
- Apps use SSL libraries, which "talk" to TCP

SSL socket API

- cleartext passwds sent into socket traverse Internet encrypted

- TLS
  - SSL v3

SSL vs. TCP/IP (Ch8.6)



normal application *application with SSL*

- SSL provides API to applications
- C and Java SSL libraries/classes readily available