

# Chapter 2: outline

## 2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

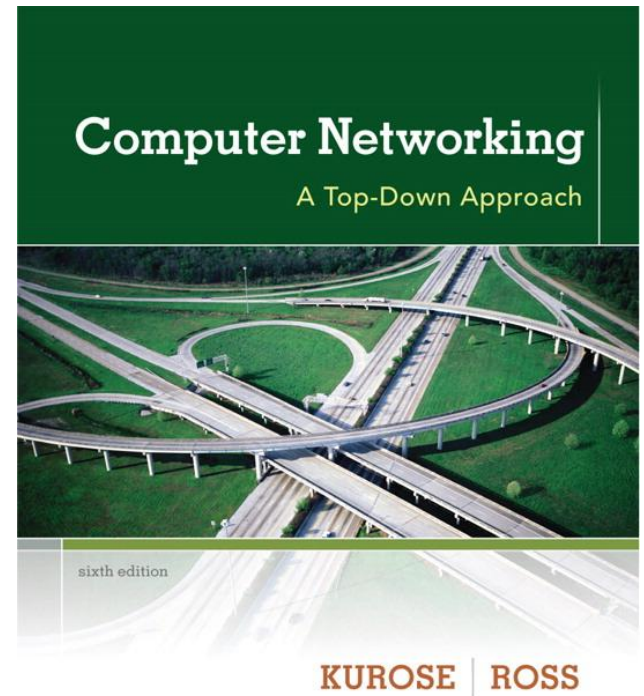
2.5 DNS

2.6 P2P applications

2.7 socket programming  
with UDP and TCP

# The Book

*Start Reading  
Chapter 2 Now*



# Chapter 2: application layer

## our goals:

- ❖ conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- ❖ learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- ❖ creating network applications
  - socket API

# Some network apps

- ❖ e-mail
- ❖ web
- ❖ text messaging
- ❖ remote login
- ❖ P2P file sharing
- ❖ multi-user network games
- ❖ streaming stored video (YouTube, Hulu, Netflix)
- ❖ voice over IP (e.g., Skype)
- ❖ real-time video conferencing
- ❖ social networking
- ❖ search
- ❖ ...
- ❖ ...

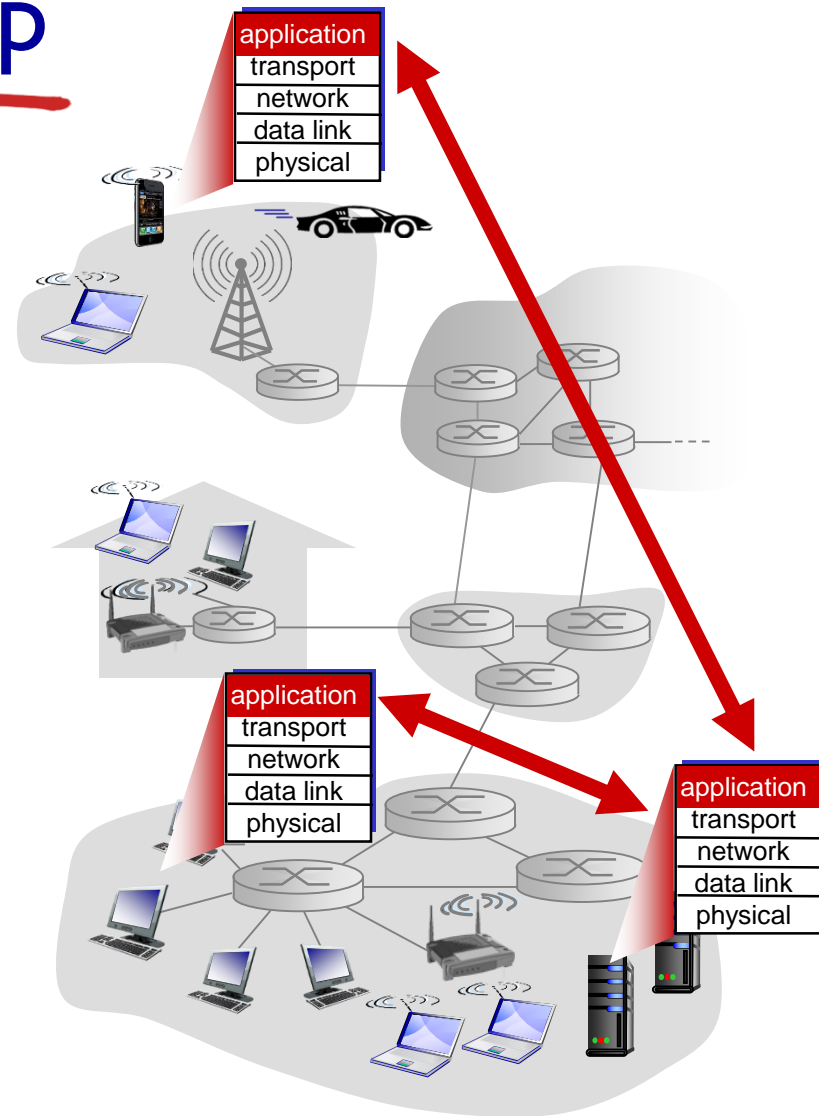
# Creating a network app

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

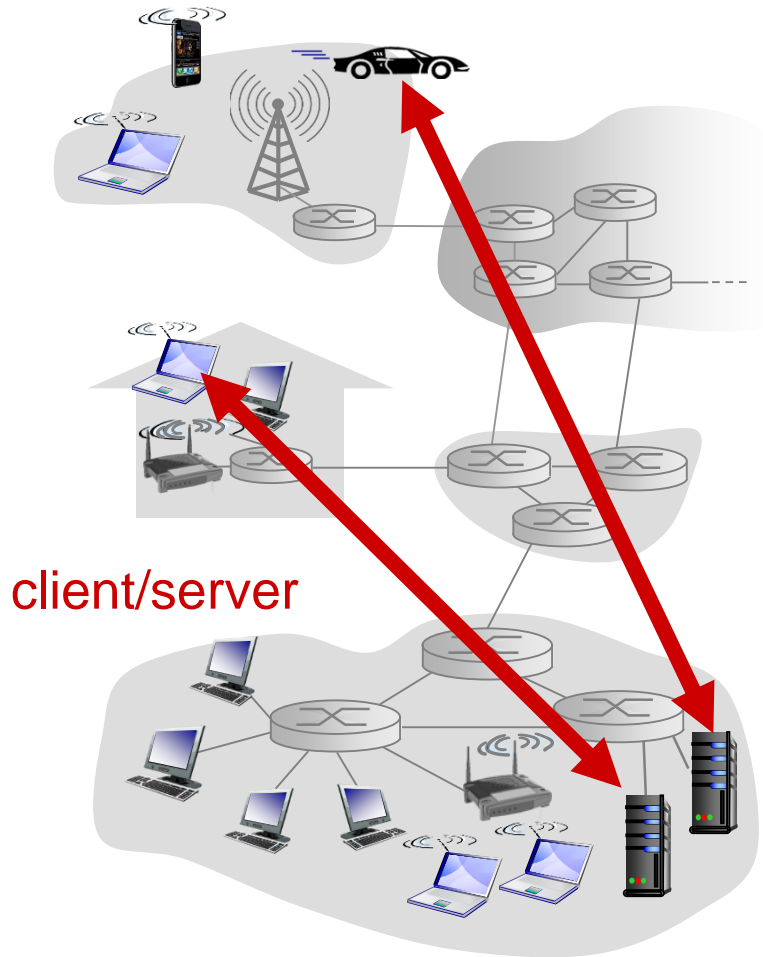


# Application architectures

possible structure of applications:

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

# Client-server architecture



## server:

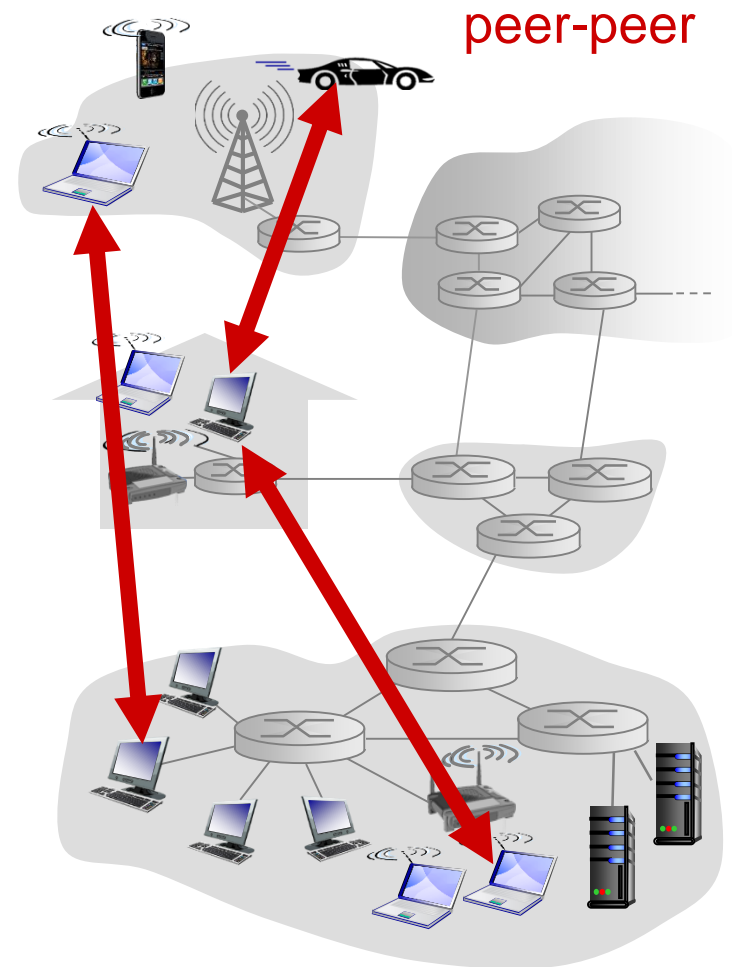
- ❖ always-on host
- ❖ permanent IP address
- ❖ data centers for scaling

## clients:

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

# P2P architecture

- ❖ 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





# Processes communicating

*process*: program running within a host

- ❖ within same host, two processes communicate using *inter-process communication* (defined by OS)
- ❖ processes in different hosts communicate by exchanging *messages*

clients, servers

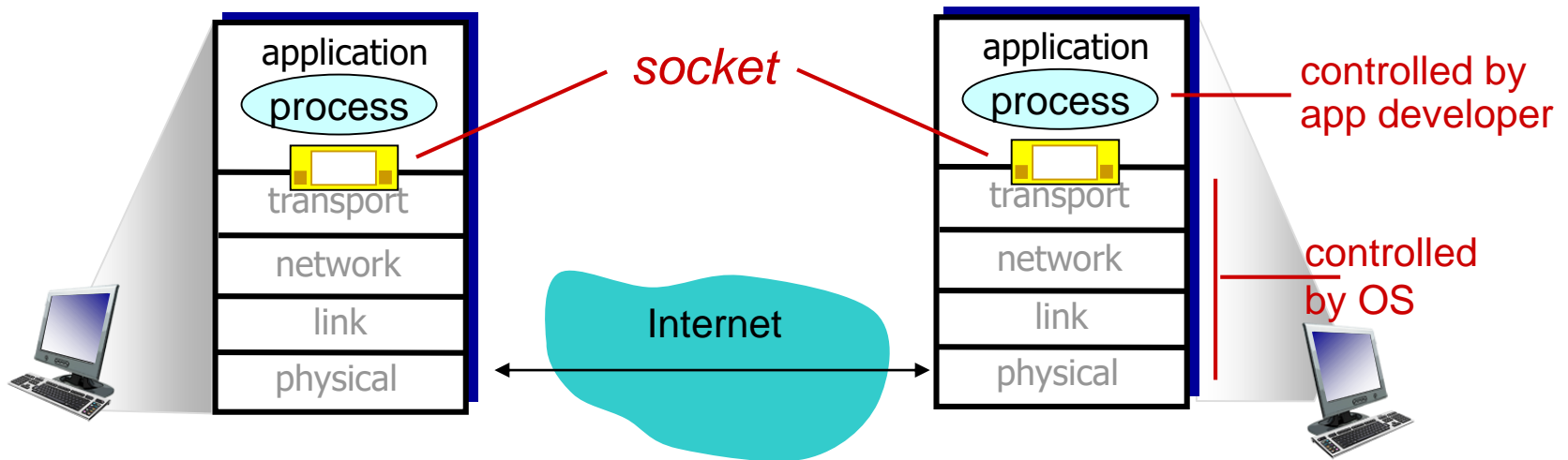
*client process*: process that initiates communication

*server process*: process that waits to be contacted

- ❖ aside: applications with P2P architectures have client processes & 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?
  - 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 gaia.cs.umass.edu web server:
  - **IP address**: 128.119.245.12
  - **port number**: 80
- ❖ more shortly...

# 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

## 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 (“elastic apps”) make use of whatever throughput they get

## security

- ❖ encryption, data integrity,  
...

# Transport service requirements: 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, ms
stored audio/video	loss-tolerant	same as above	yes, s
interactive games	loss-tolerant	few kbps up	yes, ms
text messaging	no loss	elastic	yes and no

# Internet transport protocols services

## TCP service:

- ❖ *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 guarantee, security
- ❖ *connection-oriented*: setup required between client and server processes

## UDP service:

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

Q: why bother? Why is there a UDP? What applications want that?

# Internet apps: application, transport protocols

<b>application</b>	<b>application layer protocol</b>	<b>underlying transport protocol</b>
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)	TCP or UDP



# Securing TCP – be aware as you write apps!

## TCP & UDP

- ❖ no encryption
- ❖ cleartext passwords 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 passwords sent into socket traverse Internet encrypted
- ❖ See Chapter 7

# Chapter 2: outline

## 2.1 principles of network applications

- app architectures
- app requirements

## 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 UDP and TCP