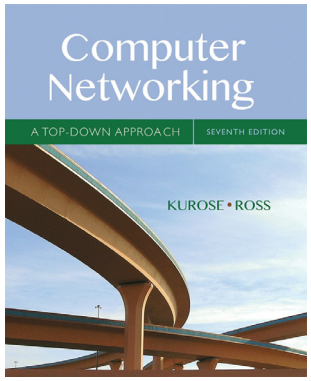
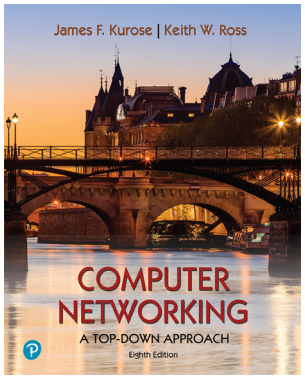
# Chapter 2
# Application Layer

Courtesy to the textbooks' authors and Pearson Addison-Wesley because many slides are adapted from the following textbooks and their associated slides.




Jim Kurose, Keith Ross, "Computer Networking: A Top Down Approach", 7th Edition, Pearson, 2016.




Jim Kurose, Keith Ross, "Computer Networking: A Top Down Approach", 8th Edition, Pearson, 2020.

# Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP
  - reading assignment

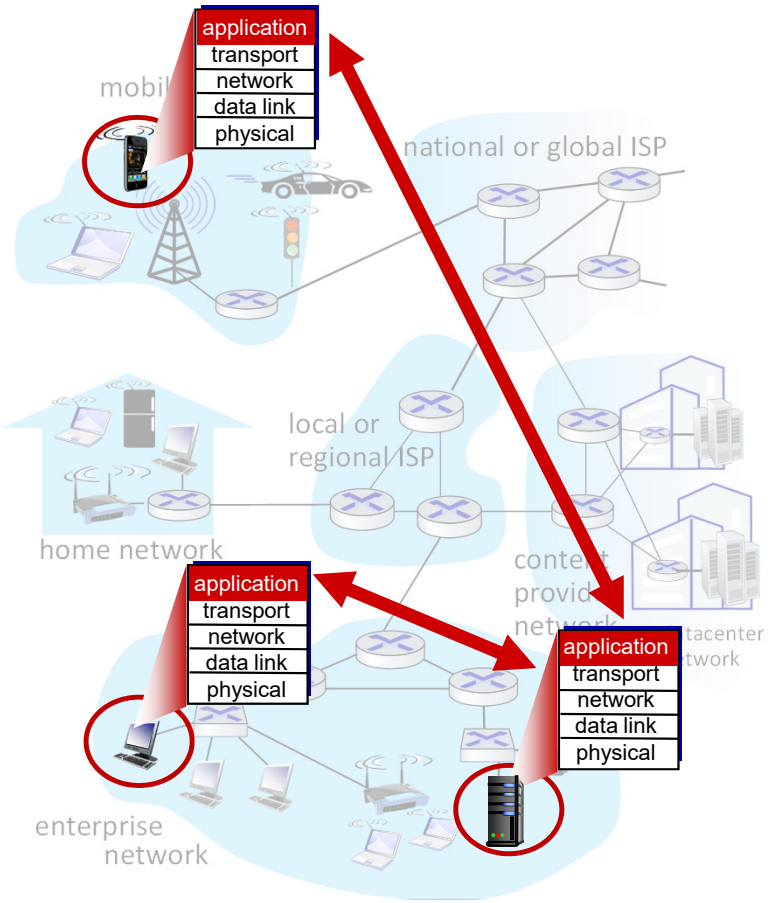
# Where does a network app reside?

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



# Client-server paradigm

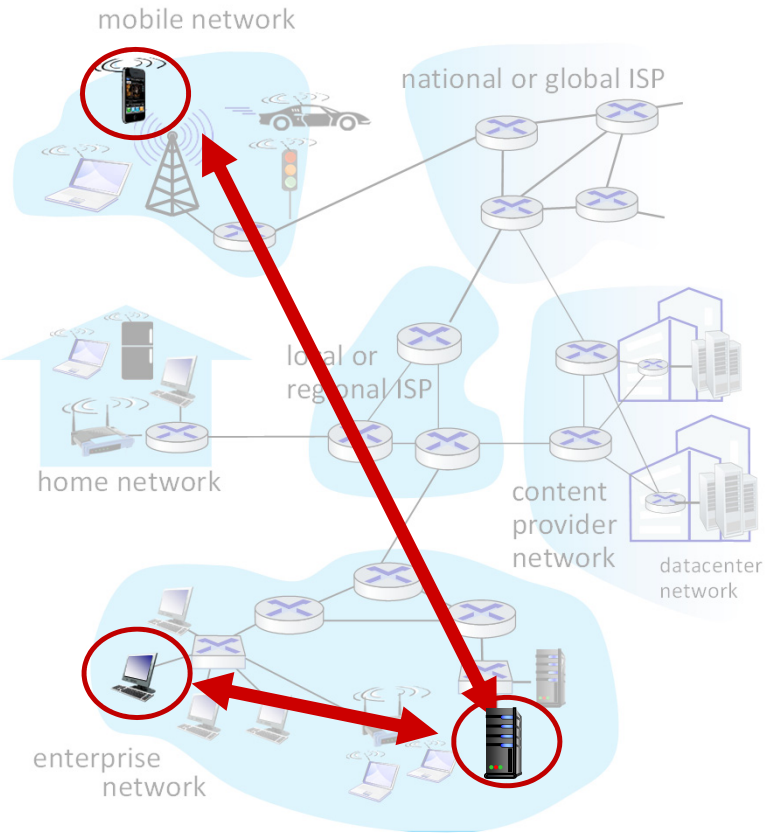
server:

- always-on host
- permanent IP address
- often in data centers, for scaling

clients:

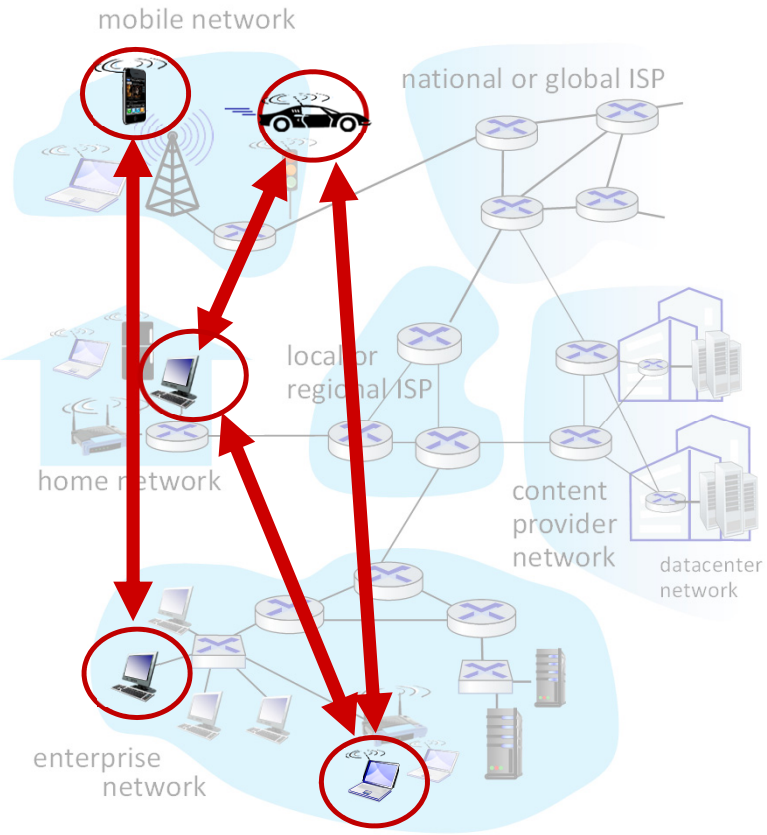
- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

- examples: HTTP, IMAP, FTP



# Peer-peer architecture

- *no* always-on server
- arbitrary end systems directly communicate
- a peer *i)* requests service from other peers and *ii)* provides 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
- example: P2P file sharing



# How does two processes in hosts communicate?

*process:* program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- in different hosts, network app processes communicate by exchanging messages via socket

clients, servers

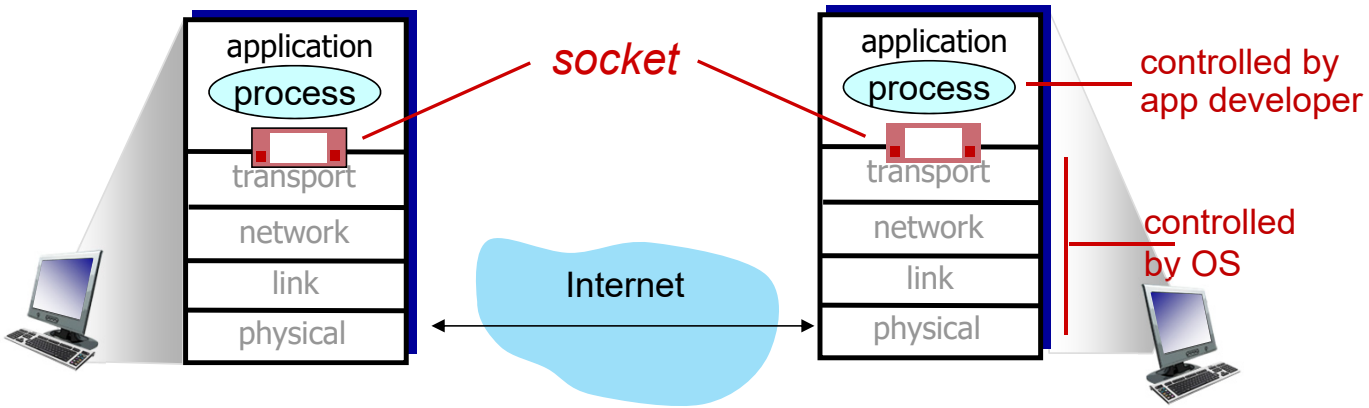
*client process:* process that initiates communication

*server process:* process that waits to be contacted

- note: an application with P2P architectures have both client process & server process

# Sockets

- process sends/receives messages to/from its socket
- socket analogous to door (or mailbox)
  - 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
  - two sockets involved: one on each side



# 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 number associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to www.nthu.edu.tw web server:
  - IP address: 140.114.69.135
  - port number: 80
- more shortly…

# An application-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

# 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
- throughput
  - some apps (e.g., multimedia) require minimum amount of throughput
  - 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/download | 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<br>video: 10Kbps-5Mbps | yes, 100's msec |
| streaming audio/video | loss-tolerant | same as above | yes, few secs |
| interactive games | loss-tolerant | Kbps+ | yes, 100's msec |
| 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 is overloaded
- *connection-oriented:* setup required between client and server processes
- *does not provide:* timing, minimum throughput guarantee, security

*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.

Why is there UDP?

- is no-frills and lightweight
- provides minimal services

# Internet applications, and transport protocols

| application | application layer protocol | transport protocol |
|---|---|---|
| file transfer/download | FTP [RFC 959] | TCP |
| e-mail | SMTP [RFC 5321] | TCP |
| Web documents | HTTP 1.1 [RFC 2616] | TCP |
| Internet telephony | SIP [RFC 3261], RTP [RFC 3550], or proprietary | TCP or UDP |
| streaming audio/video | HTTP [RFC 7320], DASH | TCP |