Computer Networks and Applications

COMP 3331/COMP 9331

Week 2

Application Layer (Principles)

Chapter 2, Section 2.1

2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

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
  - SMTP / POP3 / IMAP
  - DNS
- creating network applications
  - socket API

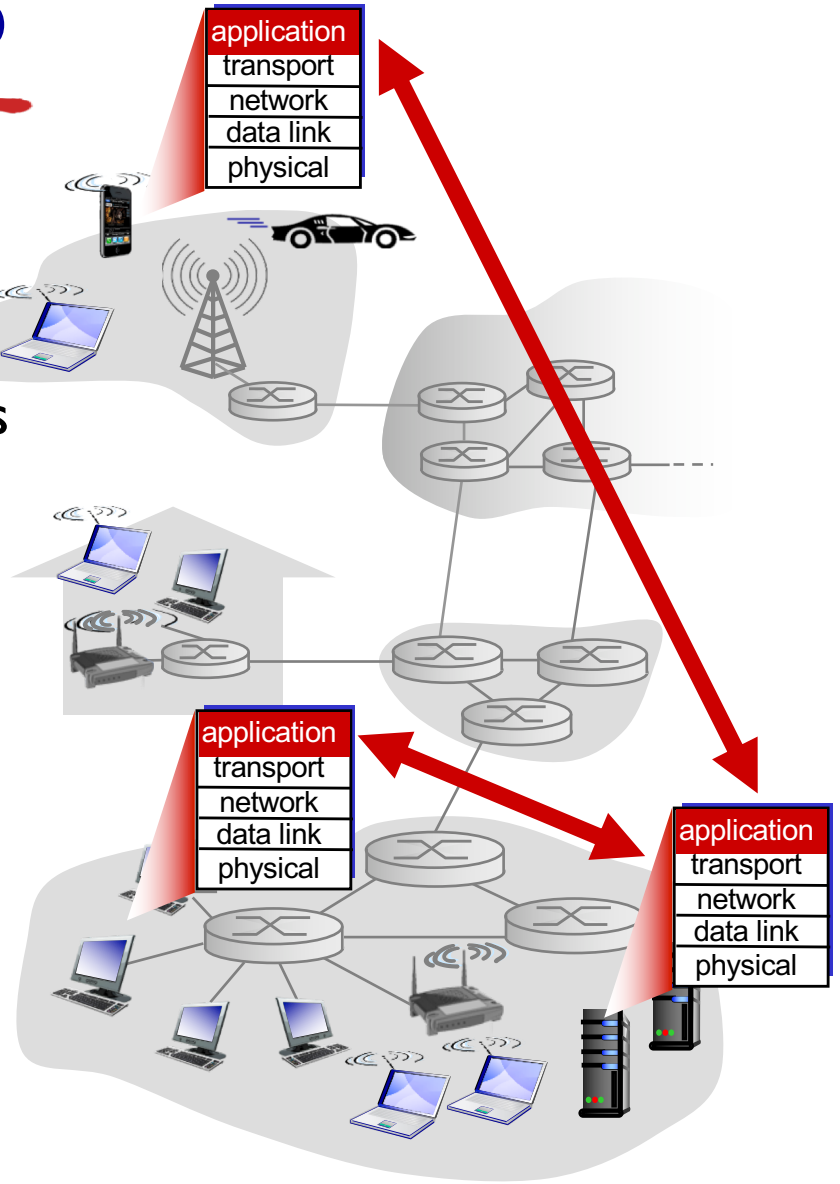
Creating a network app

Write programs that:

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

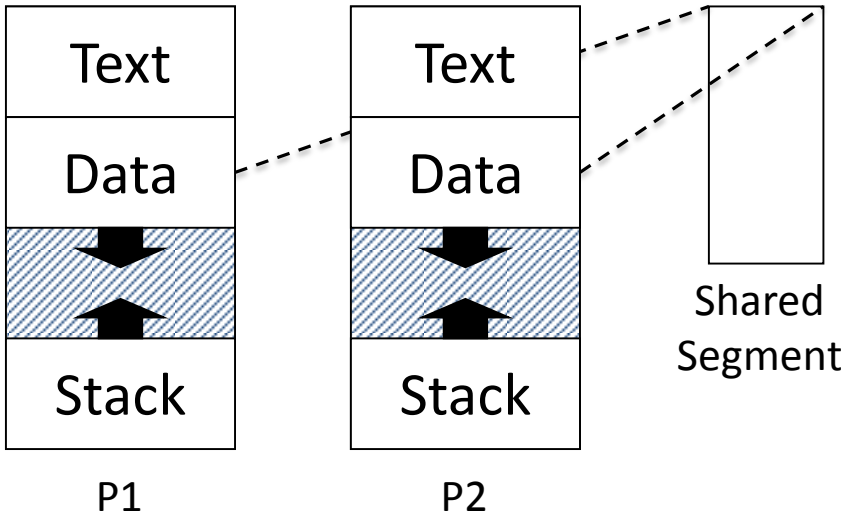
No need to write software for network-core devices

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



Interprocess Communication (IPC)

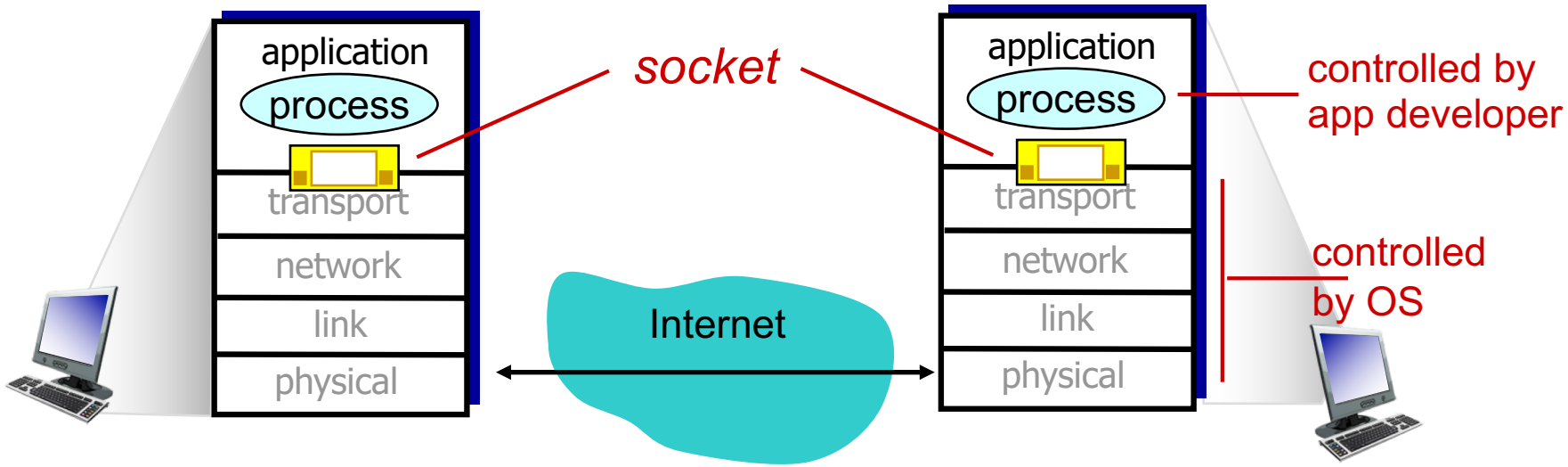
- Processes talk to each other through Inter-process communication (IPC)
- On a single machine:
  - Shared memory



- Across machines:
  - We need other abstractions (message passing)

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
- Application has a few options, OS handles the details

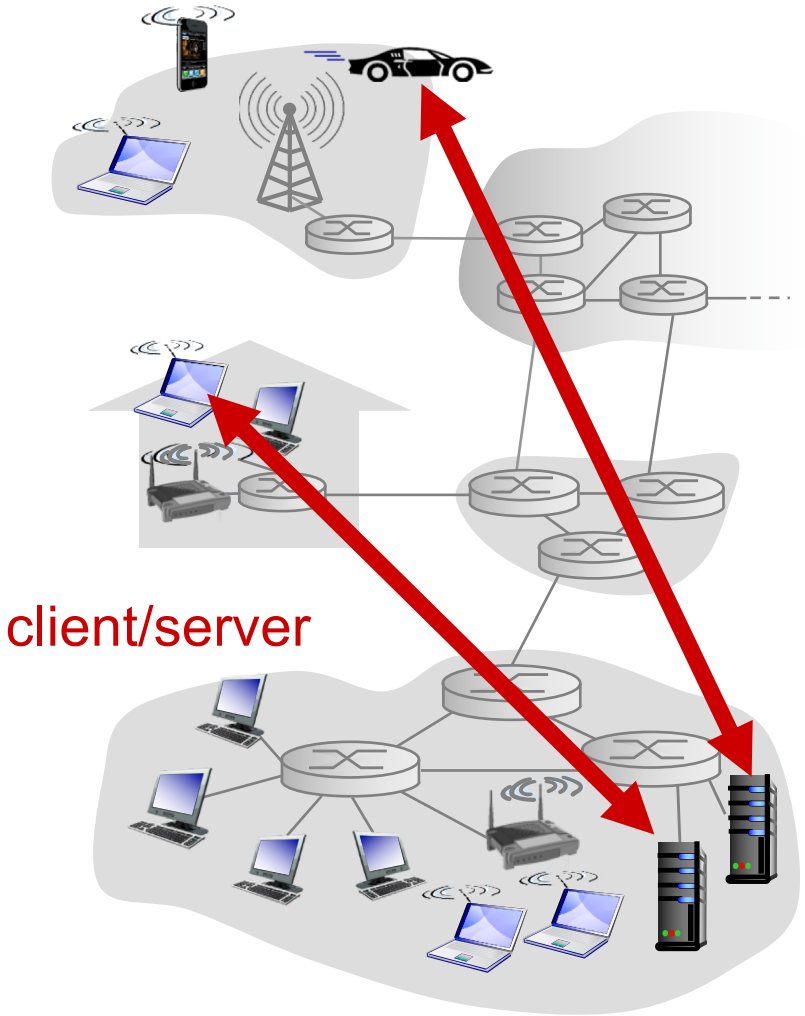


Addressing processes

- 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 cse.unsw.edu.au web server:
  - IP address: 129.94.242.51
  - port number: 80

Client-server architecture



server:

- Exports well-defined request/response interface
- long-lived process that waits for requests
- Upon receiving request, carries it out

clients:

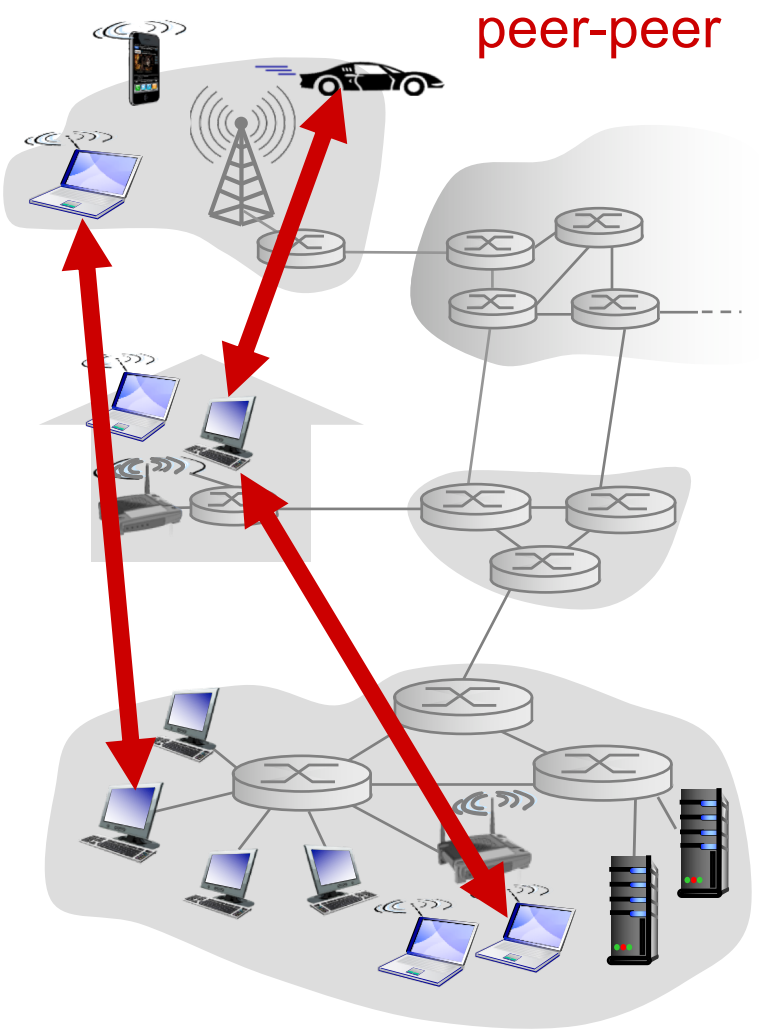
- Short-lived process that makes requests
- "User-side" of application
- Initiates the communication

Client versus Server

- Server
  - Always-on host
  - Permanent IP address (rendezvous location)
  - Static port conventions (http: 80, email: 25, ssh:22)
  - Data centres for scaling
  - May communicate with other servers to respond
- Client
  - May be intermittently connected
  - May have dynamic IP addresses
  - Do not communicate directly with each other

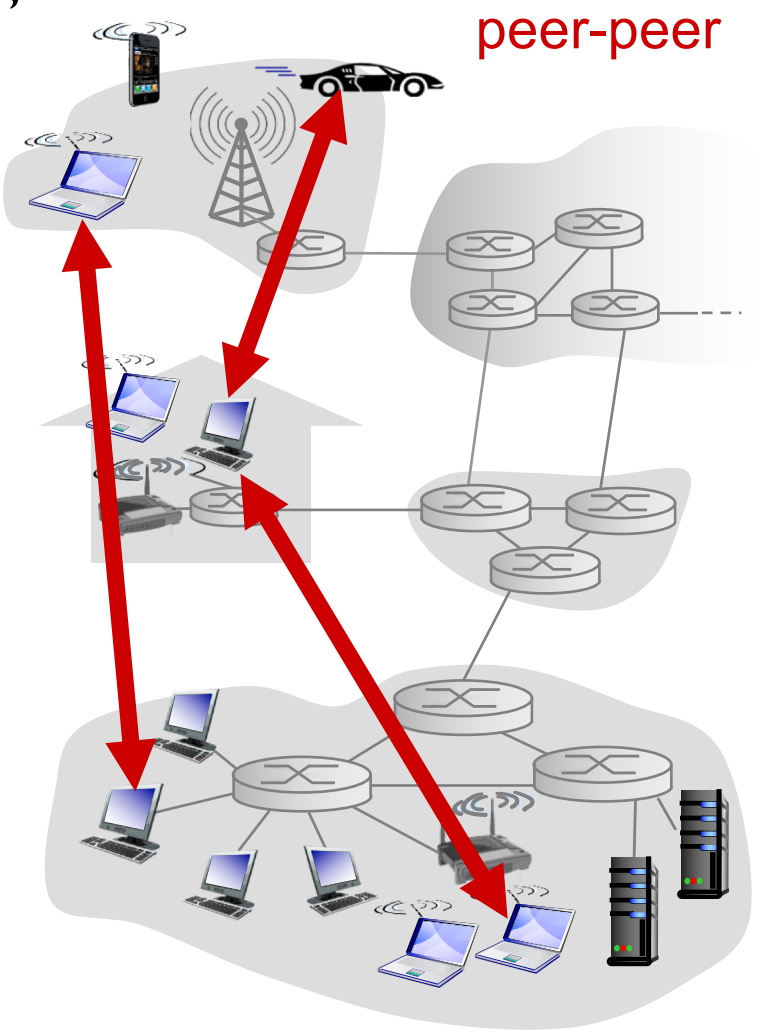
P2P architecture

- *no* always-on server
  - No permanent rendezvous involved
- arbitrary end systems (peers) directly communicate
- Symmetric responsibility (unlike client/server)
- Often used for:
  - File sharing (BitTorrent)
  - Games
  - Video distribution, video chat
  - In general: "distributed systems"



P2P architecture: Pros and Cons

+ 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
+ Speed: parallelism, less contention
+ Reliability: redundancy, fault tolerance
+ Geographic distribution



- Fundamental problems of decentralized control
  - State uncertainty: no shared memory or clock
  - Action uncertainty: mutually conflicting decisions
- Distributed algorithms are complex

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: 50kbps-1Mbps video:100kbps-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
Chat/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, orconnection setup,

Q: why bother? Why is there a UDP?

NOTE: More on transport later on

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)	TCP or UDP