



CSCI 446 Introduction to Computer Networks

Zhaohong Wang
Department of EECE
California State University, Chico

Topic

Principles of Network Applications

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Application Layer

our goals:

- conceptual, implementation aspects of network application protocols
 - transport-layer service models
 - client-server paradigm
 - peer-to-peer paradigm
 - content distribution networks
- 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
- ...
- ...

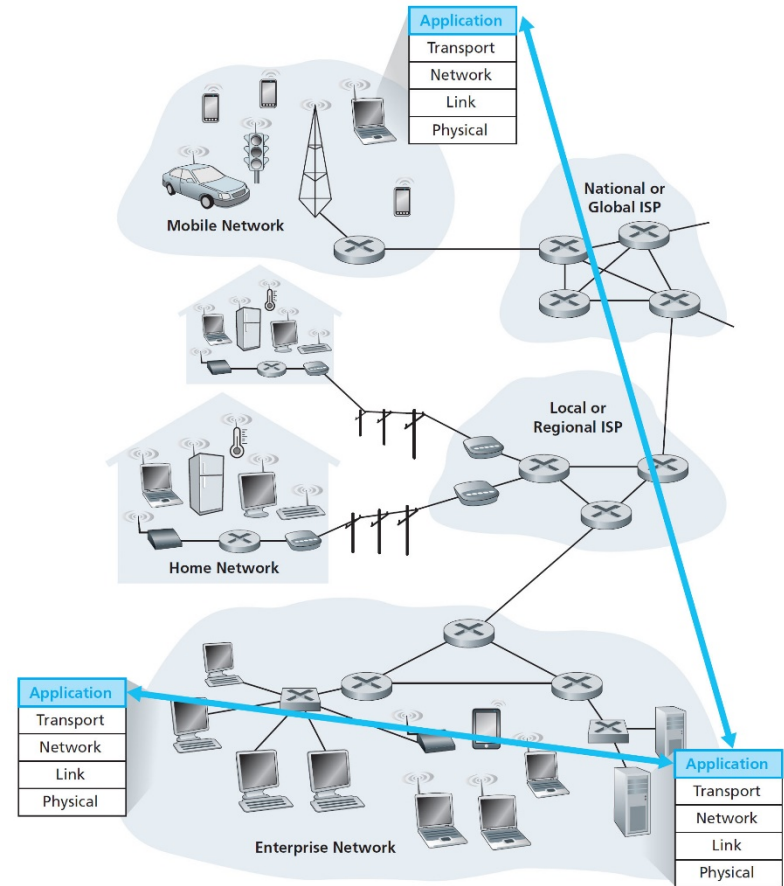
Creating a Network App

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)

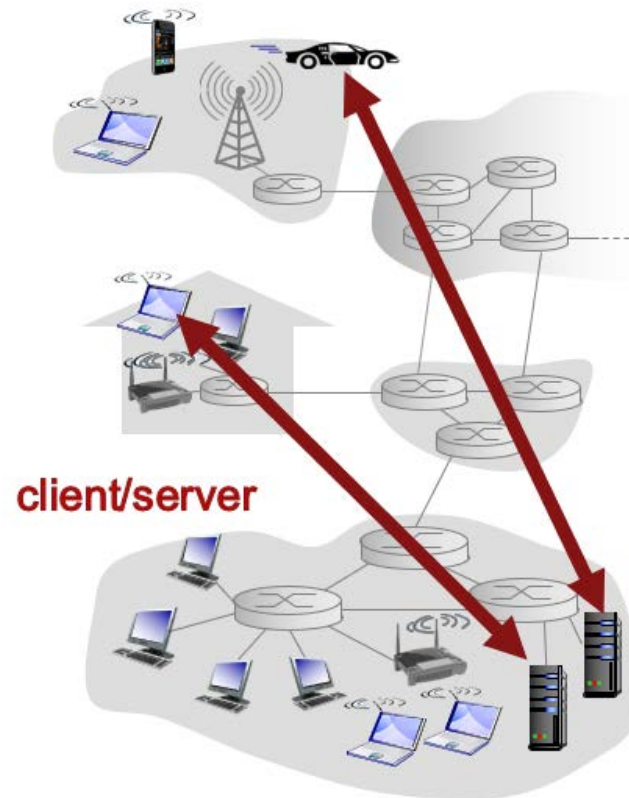
Client-Server Architecture

server:

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

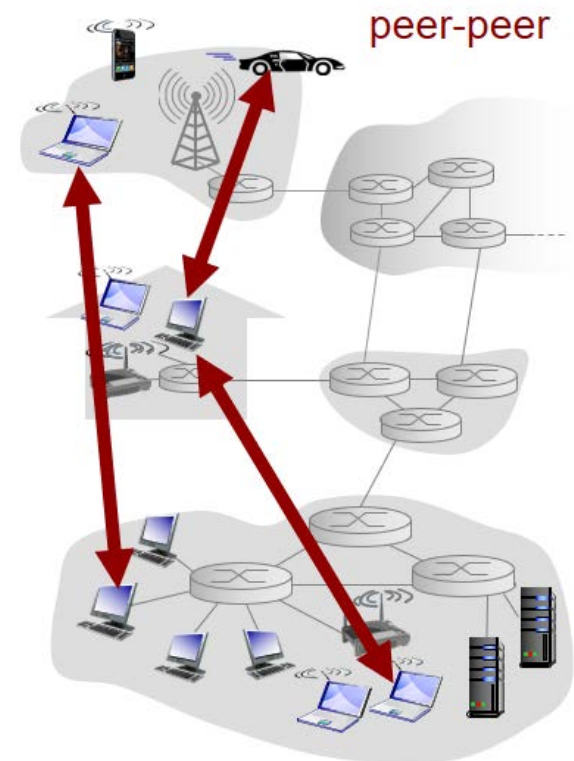
clients:

- 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

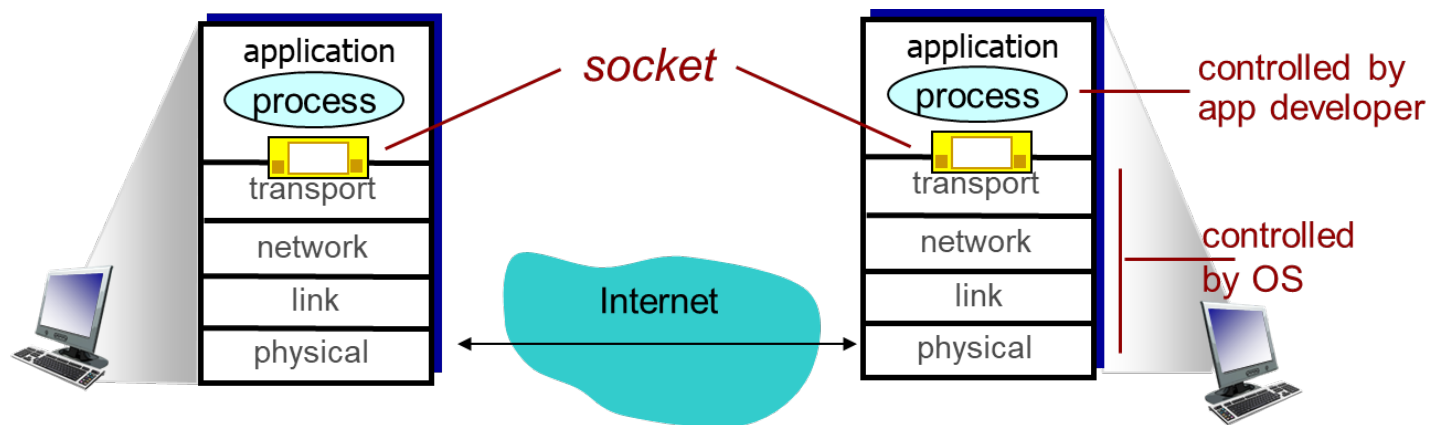
client process: process that initiates communication

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, 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:

- **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?

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

Securing TCP

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, that “talk” to TCP

SSL socket API

- cleartext passwords sent into socket traverse Internet encrypted
- see Chapter 8



Thank you!