CMSC 322 Computer Networks

Applications and End-To-End

Professor Doug Szajda

Review

- In the previous two slide sets, we discussed the fundamentals of networking at a high level
    - *hosts* communicate over *networks* using *protocols*
    - *layered* designs use *encapsulation* to provide an abstraction to network devices, e.g., *routers*, which relay *packets* across the *physical media* that makes up the *Internet*
    - Network *delay* is broken down into *nodal processing*, *queueing*, *transmission delay*, and *propagation delay*.
    - *Security* is difficult - we need to think about it at every layer.

System Design

- *End-To-End Arguments in System Design*
  - Saltzer, Reed, Clark
- Asks the question:

  Where should we place functionality?
- What do we mean by "functionality"?
  - e.g.: reliable data transmission
- What do we mean by "where"?
  - Recall the concept of network layers and the devices that interact with the layers

Design Principle

- The Principle:

  "The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)" -- Saltzer et al.

  - What does this mean in layman's terms?

Design Principle

- The Principle:

"The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)" -- Saltzer et al.

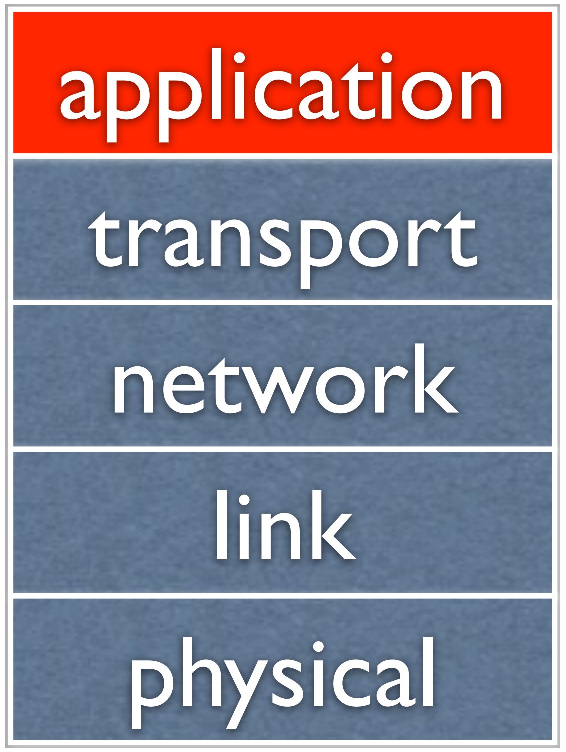
  - What does this mean in layman's terms?
    - Put functionality at the lowest layer at which it can be correctly and completely implemented. The exception is that sometimes functionality may be duplicated at a lower layer for performance reasons.

An Example

- Reliable data transmission
  - Consider a file transferred across a network
    - It is all a matter of *context*
  - What happens to performance if we strictly adhere to the principle?
- Other examples:
  - guaranteed delivery, secure transmission,
    duplicate message suppression,
    in-order delivery
  - We will discuss these concepts in more depth in the coming weeks

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 file sharing



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
- programming network applications
  - socket API

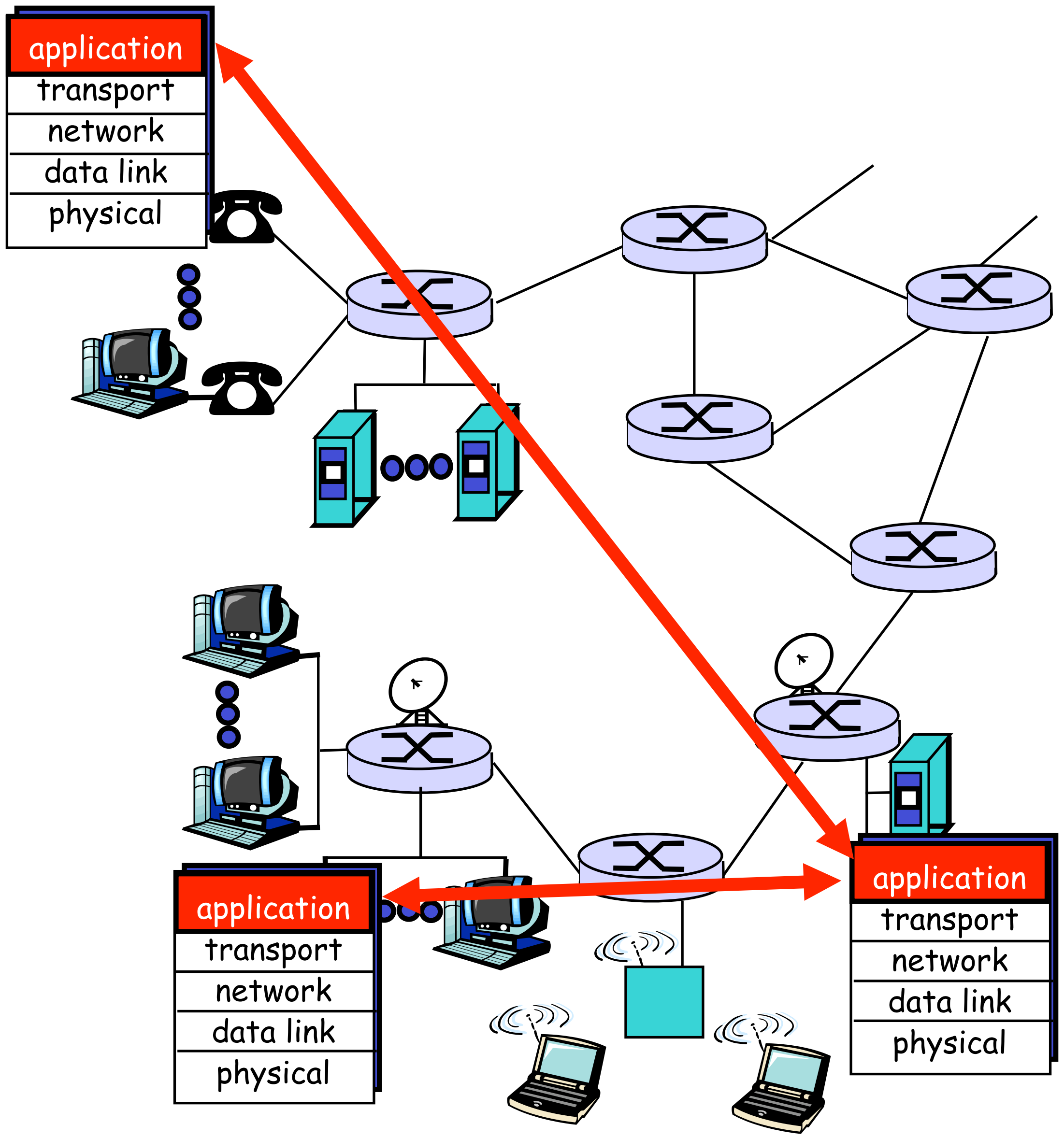
Some network apps

- E-mail
- Web
- Instant messaging
- Remote login
- P2P file sharing (e.g., KaZaA)
- Multi-user network games
- Streaming stored video clips
- Internet telephone
- Real-time video conferencing
- Massive parallel computing

Creating a network app

Write programs that

- run on different end systems and
- communicate over a network.
- e.g., Web: Web server software communicates with browser software

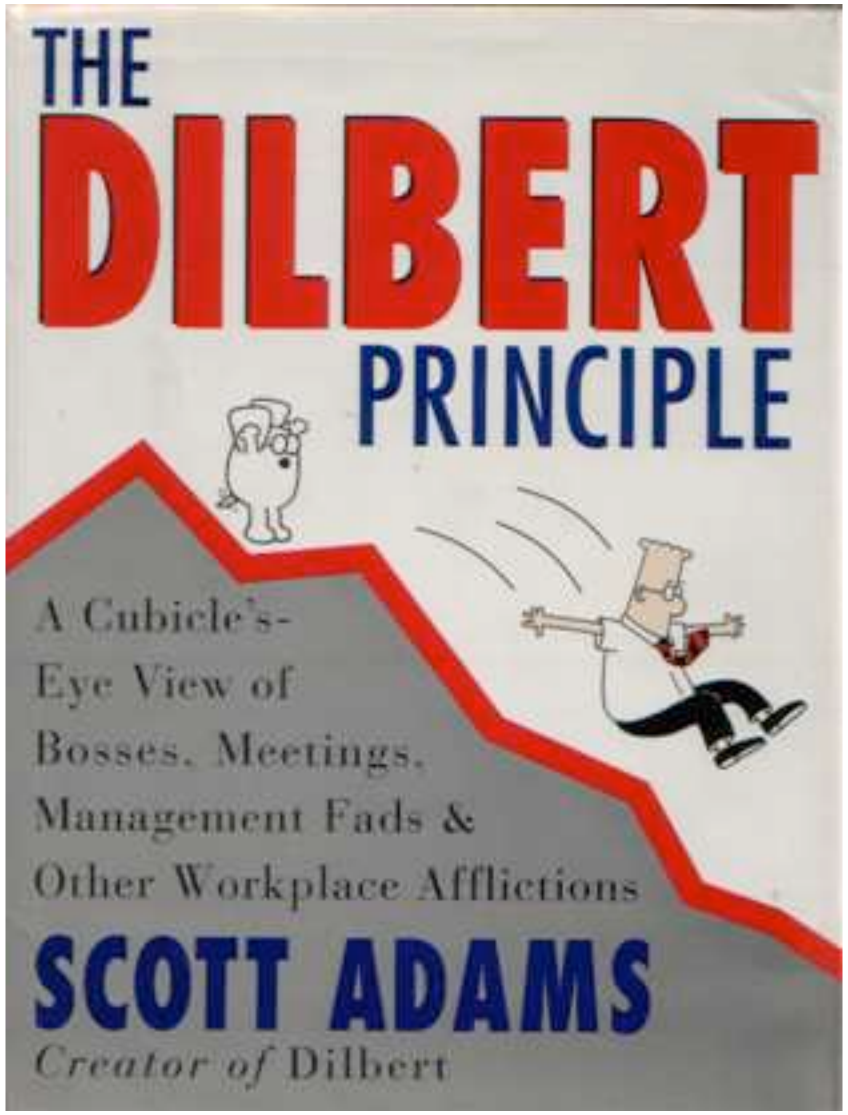


little software written for devices in network core

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

Chapter 2: Application layer

- 2.1 Principles of Network Applications
- 2.2 Web and HTTP
- 2.3 File Transfer: FTP
- 2.4 Electronic Mail in the Internet
- 2.5 DNS - Internet Directory Service
- 2.6 P2P Applications
- 2.7-2.8 Socket Programming



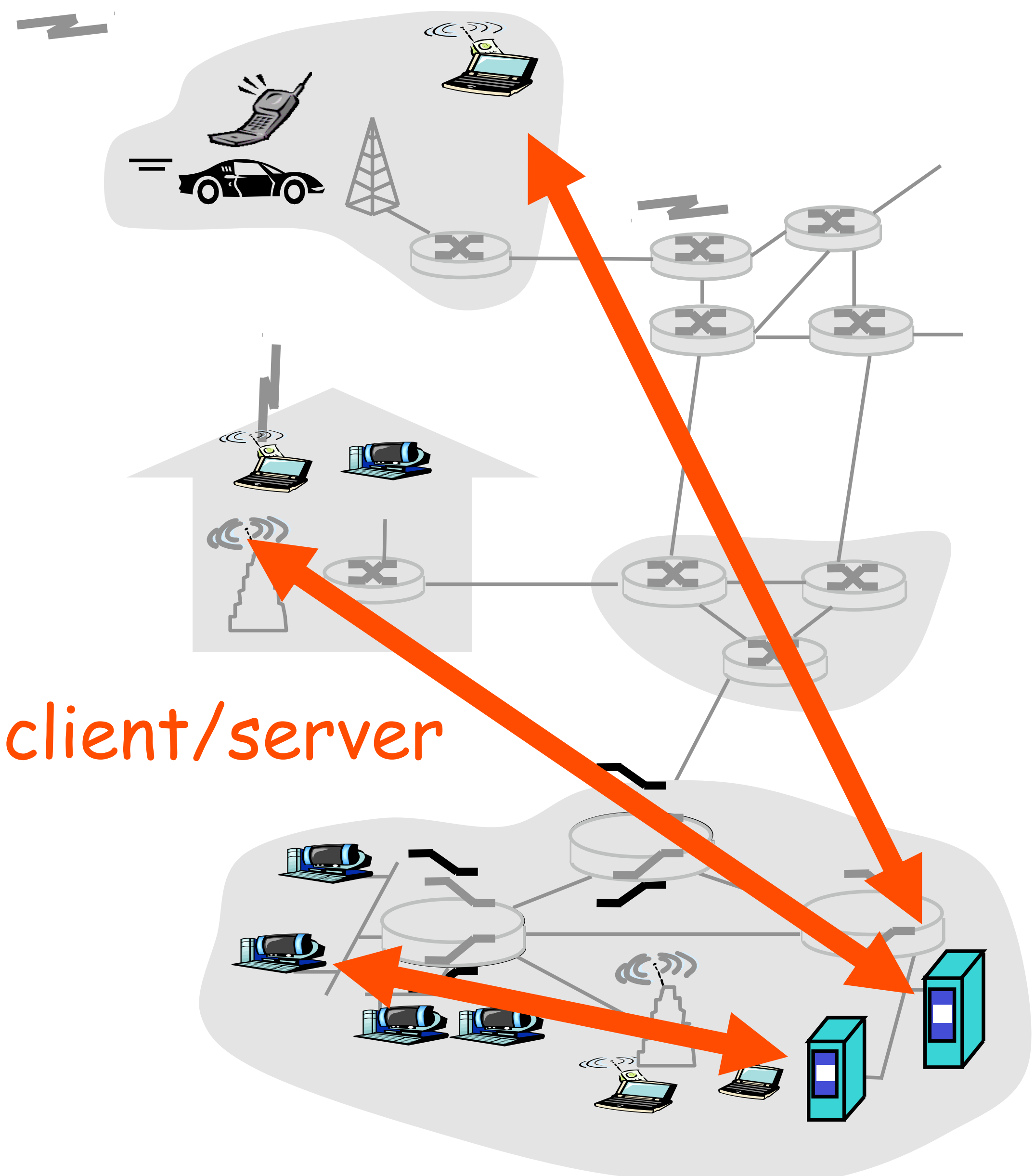
Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server architecture



server:

- always-on host
- permanent IP address
- server farms for scaling

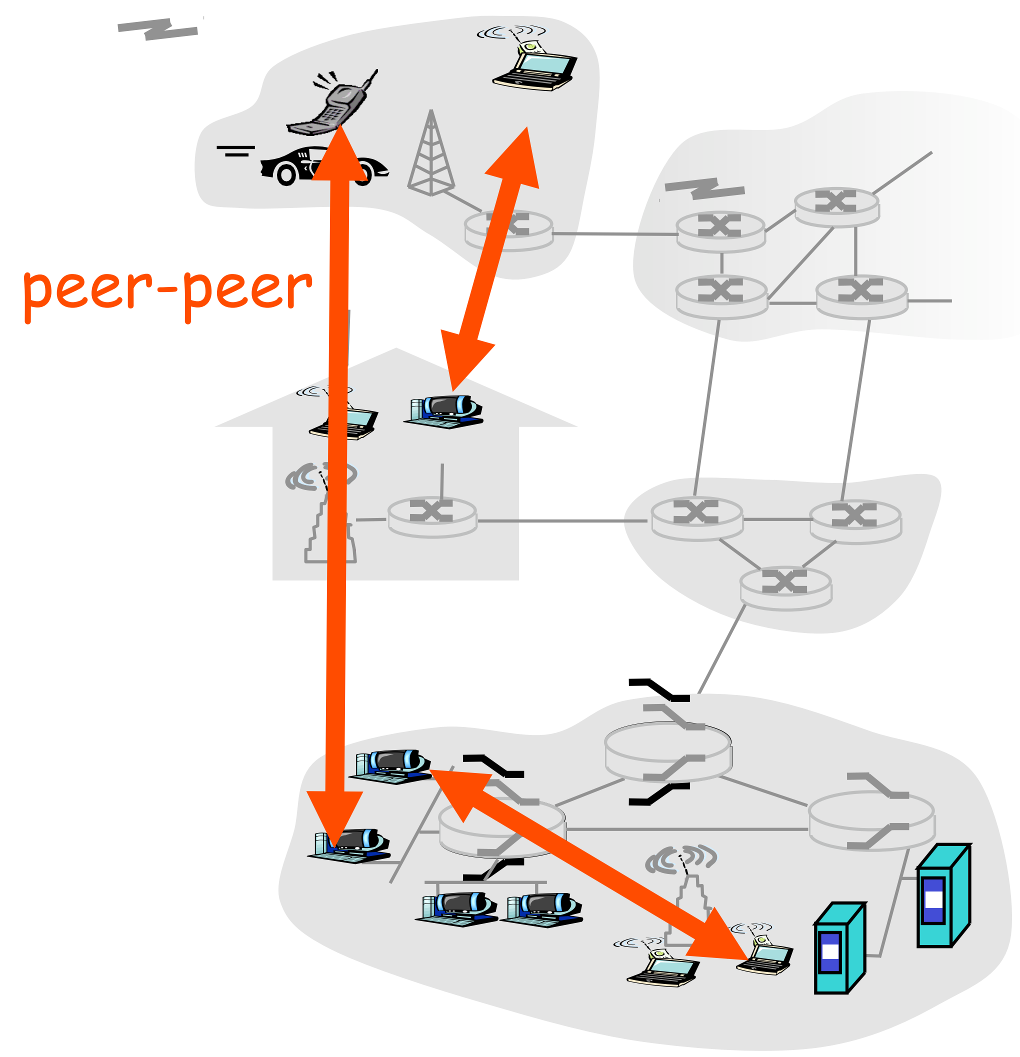
clients:

- 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
- example: Gnutella, KaZaa
  - Compare to Napster

Highly scalable but difficult to manage



Hybrid of client-server and P2P

Skype

- Internet telephony app
- Finding address of remote party: centralized server(s)
- Client-client connection is direct (not through server)
- Zoom, on the other hand, is client/server
  - Why?

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
  - 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 within a host.

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

Client process:

process that initiates communication

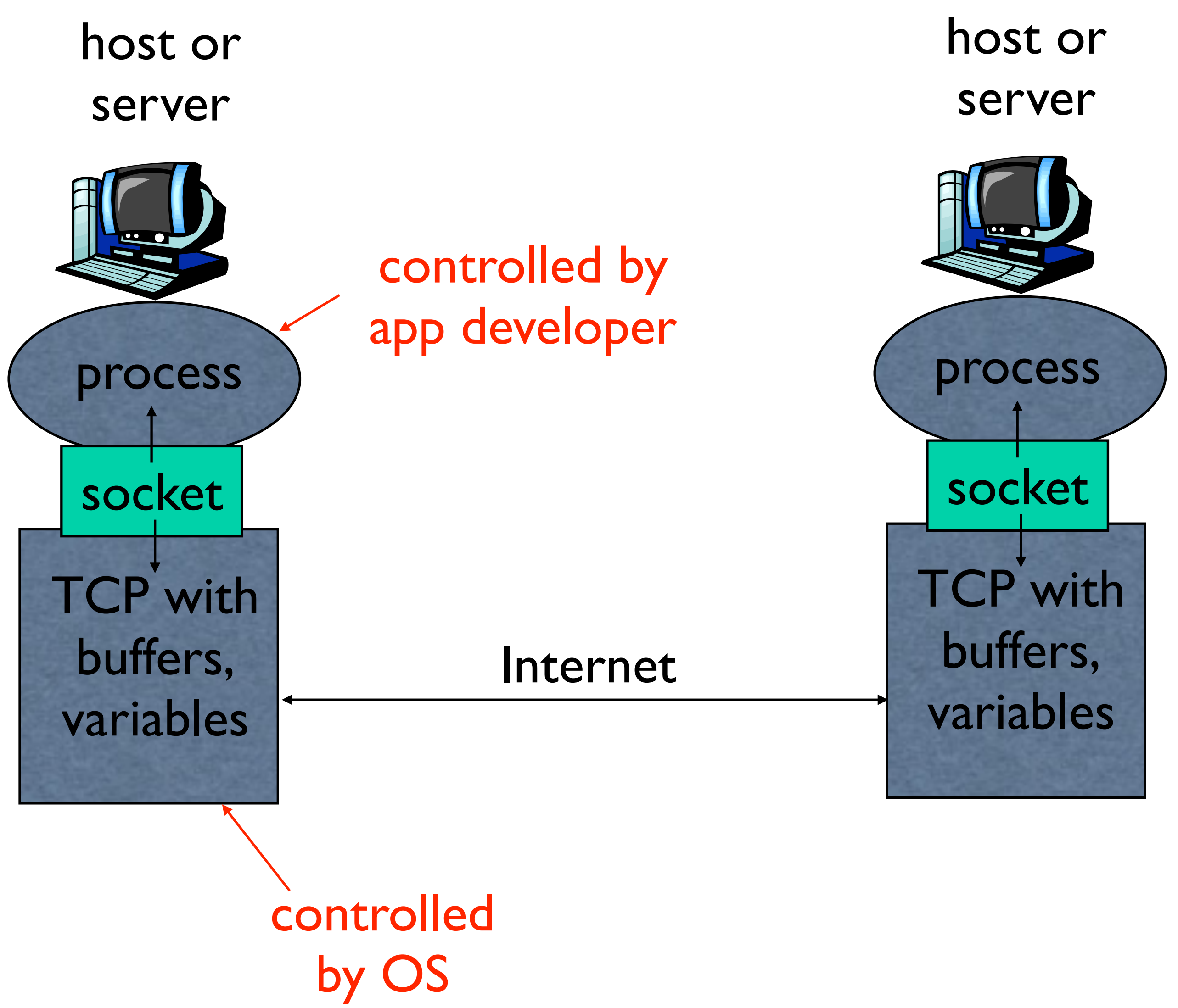
Server process:

process that waits to be contacted

- Note: 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 which brings message to socket at receiving process



- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this in pocket socket guide)

Addressing processes

- to receive messages, process must have identifier
- host device has unique 32-bit (or 128-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 (or 128-bit) IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  - Answer: 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 www.cse.psu.edu web server:
  - IP address: 130.203.4.2
  - 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

Public-domain protocols:

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

Proprietary protocols:

- e.g., Skype

I E T F®

What transport service does an app need?

Data loss

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

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Bandwidth

- some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- other apps ("elastic apps") make use of whatever bandwidth they get

Security

- Encryption, integrity?

Question:

- Assume you have strong encryption and integrity mechanisms…
  - If you want to keep information private, where in the protocol stack do you perform encryption? And on what?
  - Do you perform encryption end-to-end or on a hop-by-hop basis? What are the pros and cons of each?

Transport service requirements of common apps

Application			
file transfer e-mail Web documents real-time audio/video stored audio/video interactive games instant messaging	Data loss: No loss or loss tolerant	Bandwidth: Elastic or has strict requirements	Time Sensitive?: Yes or no?

Transport service requirements of common apps

Application	Data loss	Bandwidth	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
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

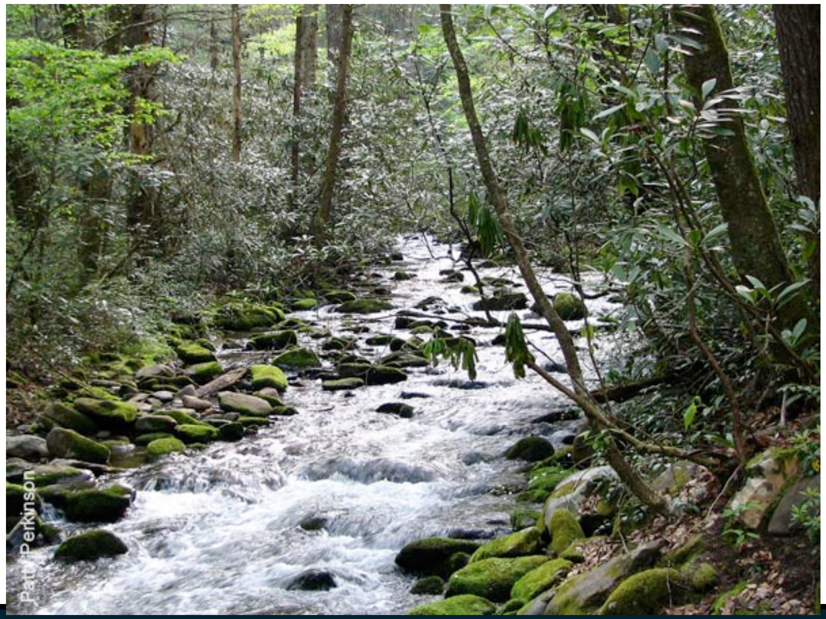
TCP service:

- connection-oriented: setup required between client and server processes
- 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 bandwidth guarantees

UDP service:

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

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	Telnet [RFC 854]	TCP
Web access	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary	TCP or UDP
Internet telephony	proprietary	typically UDP