

CSCI 491-01

Topics: Internet Programming

Fall 2008

## Application Layer

Derek Leonard  
Hendrix College

September 8, 2008

Original slides copyright © 1996-2007 J.F Kurose and K.W. Ross

# Chapter 2: Roadmap

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

2.7 Socket programming with TCP

2.8 Socket programming with UDP

2.9 Building a Web server

Application (5)

Transport (4)

Network (3)

Data-link (2)

Physical (1)

# Chapter 2: Application Layer

## Chapter goals:

- Conceptual, implementation aspects of network application protocols
- Cover 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 Applications

- E-mail
- Web
- Instant messaging
- Remote login
- P2P file sharing
- Multi-user network games
- Streaming stored video clips
- Internet telephone
- Real-time video conferencing
- Massively parallel computing

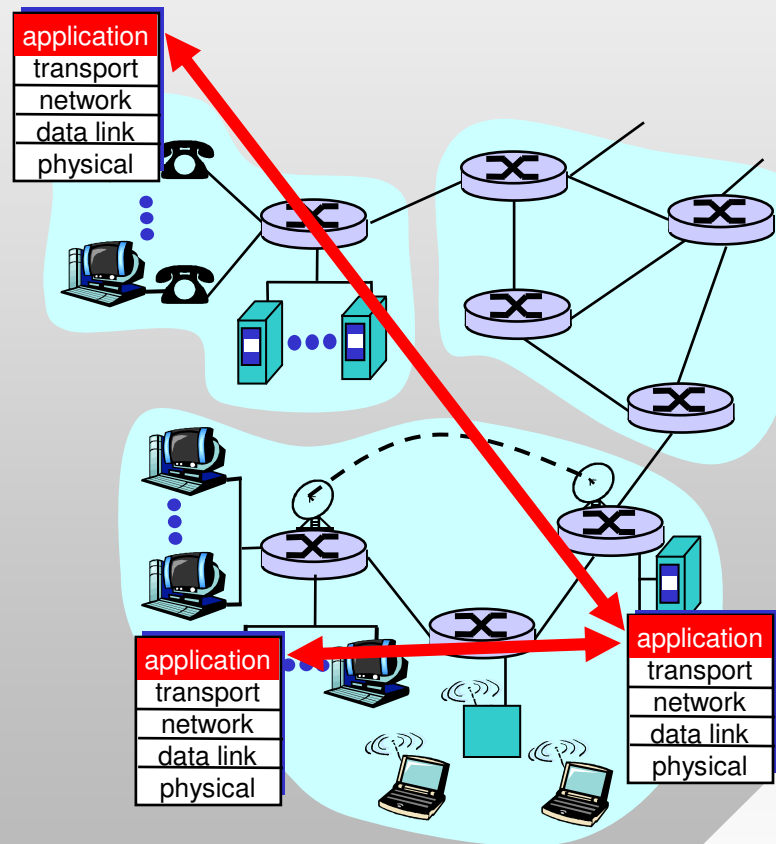
# Creating a Network Application

## Programs that

- Usually interact with user
- Communicate over a network
- E.g., Web server software communicates with browser software

## No software written for devices in network core

- Network core devices do not function at app layer
- This design allows for rapid application development



# Chapter 2: Roadmap

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

2.7 Socket programming with TCP

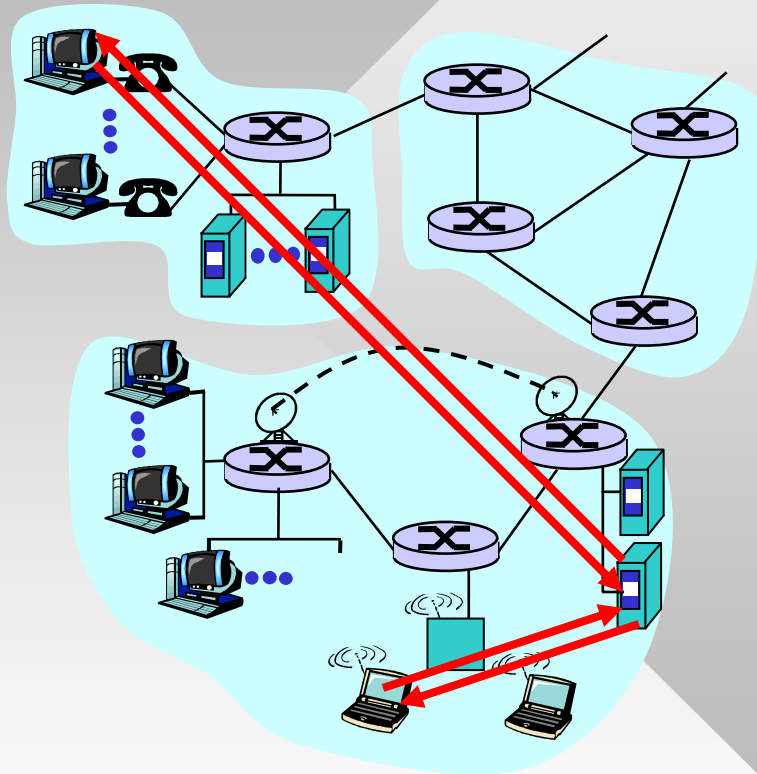
2.8 Socket programming with UDP

2.9 Building a Web server

# Application Architectures

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

# Client-Server Architecture



## Server:

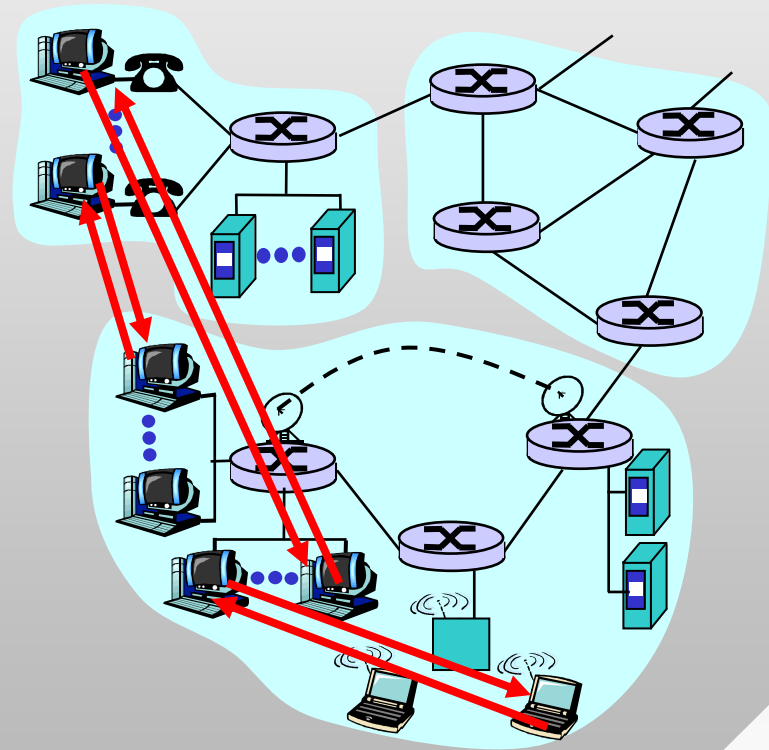
- An always-on host
- Permanent IP address
- Server farms for scaling

## 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
  - Note: search and download are pure P2P, bootstrapping is not
- **Highly scalable**
- **But difficult to manage**



# Hybrid of Client-Server and P2P

## Skype

- Voice-over-IP P2P application
- Centralized server: finding address of remote party
- Client-client connection for talking is direct (not through server)

## Instant messaging

- Chatting between two users is P2P
- Centralized service: client presence detection/location
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies

# Process Communication

**Process:** program running within a host

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

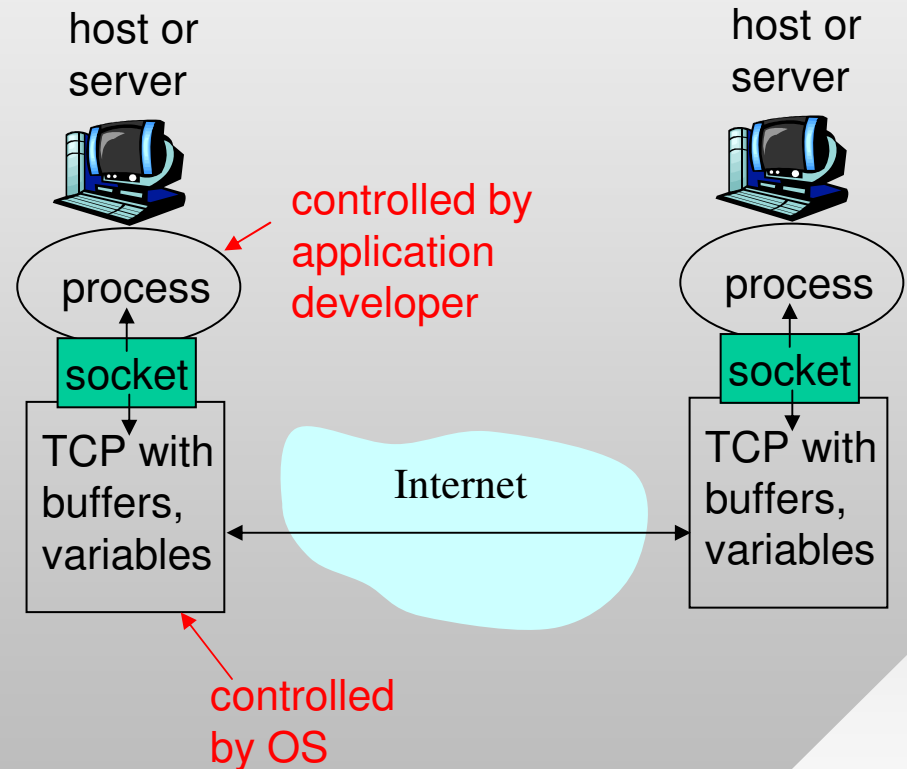
**Client process:** process that initiates communication

**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



# Addressing Processes

- For a process to receive messages, it must have an identifier
- Identifier includes both the **IP address** (32-bit) and **port number** (16-bit) associated with the process on the host
- Well-known port numbers:
  - HTTP server: 80
  - FTP: 21
  - Telnet: 23
  - Mail server: 25
- See <http://www.iana.org/assignments/port-numbers>

## 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

# Transport Services and Application Needs

## Data loss

- Some apps (e.g., audio) can tolerate certain 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

- When building an application, one must know what service to ask from the layer below

## Transport service requirements of common apps

<u>Application</u>	<u>Data loss</u>	<u>Throughput</u>	<u>Time Sensitive</u>
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 (Layer 4)

## 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

# Internet apps: application, transport protocols

<u>Application</u>	<u>Application layer protocol</u>	<u>Underlying transport protocol</u>
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)	typically UDP

## Chapter 2: Roadmap

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

2.7 Socket programming with TCP

2.8 Socket programming with UDP

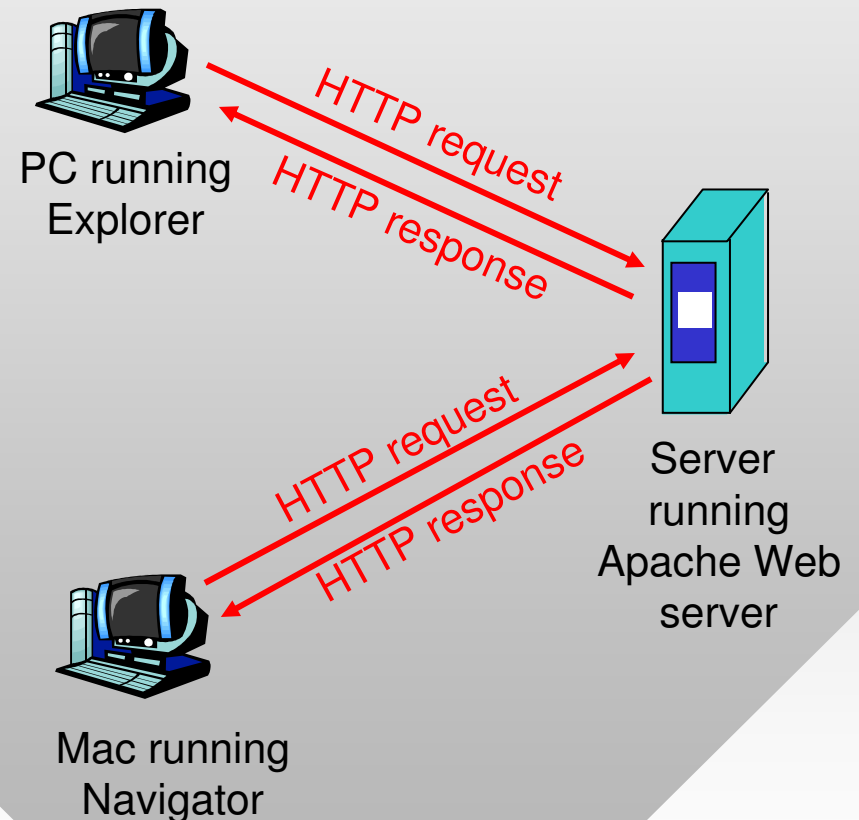
2.9 Building a Web server



# HTTP Overview

## HTTP: HyperText Transfer Protocol

- Web's application layer protocol
- Client/server model
  - **Client:** browser that requests, receives, "displays" Web objects
  - **Server:** Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



# HTTP overview (continued)

## Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

## HTTP is “stateless”

- server maintains no information about past client requests

## Protocols that maintain “state” are complex!

- past history (state) must be maintained
- if server/client crashes, their views of “state” may be inconsistent, must be reconciled

# HTTP Connections

## Nonpersistent HTTP

- At most one object is sent over a TCP connection
- HTTP/1.0 uses nonpersistent HTTP

## Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server
- HTTP/1.1 uses persistent connections in default mode

# Nonpersistent HTTP

Suppose user enters URL

`www.someSchool.edu/someDepartment/home.html`

(contains text,  
references to 10  
jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at `www.someSchool.edu` on port 80

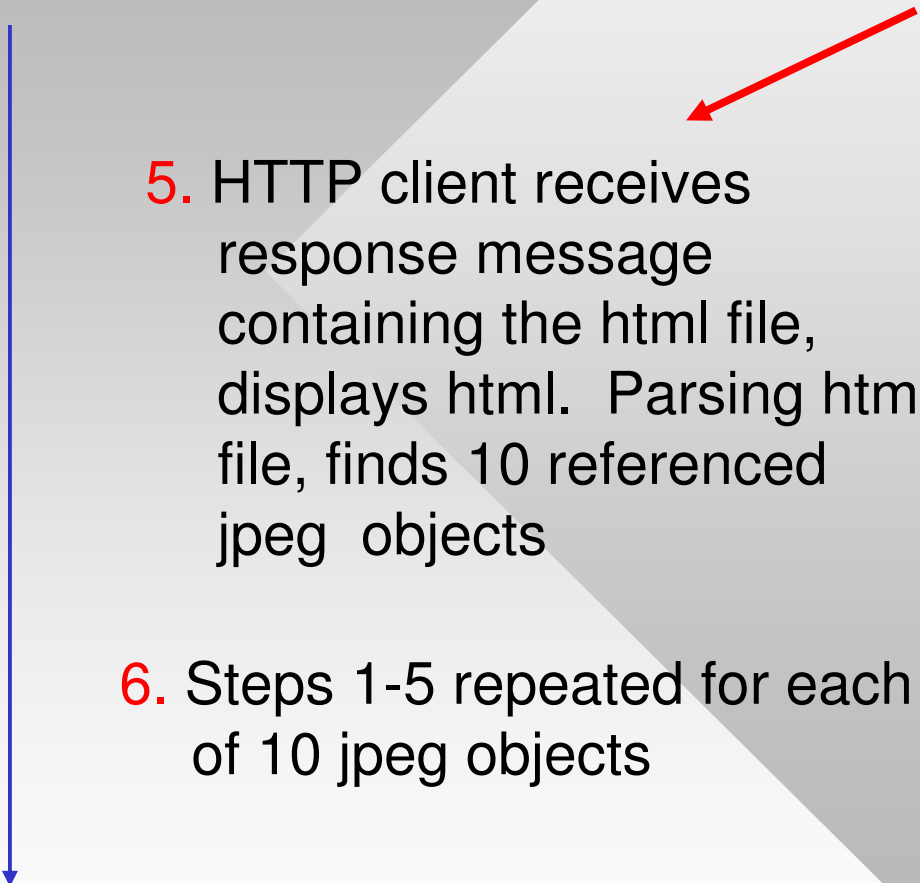
1b. HTTP server at host `www.someSchool.edu` waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `/someDepartment/home.html`

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

## Nonpersistent HTTP (Cont.)

4. HTTP server closes TCP connection.
  5. HTTP client receives response message containing the html file, displays html. Parsing html file, finds 10 referenced jpeg objects
  6. Steps 1-5 repeated for each of 10 jpeg objects
- 

# Response Time Modeling

**Definition of RTT:** time to send a small packet to travel from client to server and back

## Response time:

- One RTT to initiate TCP connection
- One RTT for HTTP request and first few bytes of HTTP response to return
- File transmission time

**total =  $2RTT + \text{transmit time}$**

