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 applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

Chapter 2: Application Layer

Our goals:

- conceptual, implementation aspects of network application protocols
  - client-server paradigm
  - peer-to-peer paradigm
  - transport-layer service models
- programming network applications
  - socket API
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS

Some network apps

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video clips
- voice over IP
- real-time video conferencing
- grid computing
-
-
-

Application architectures

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

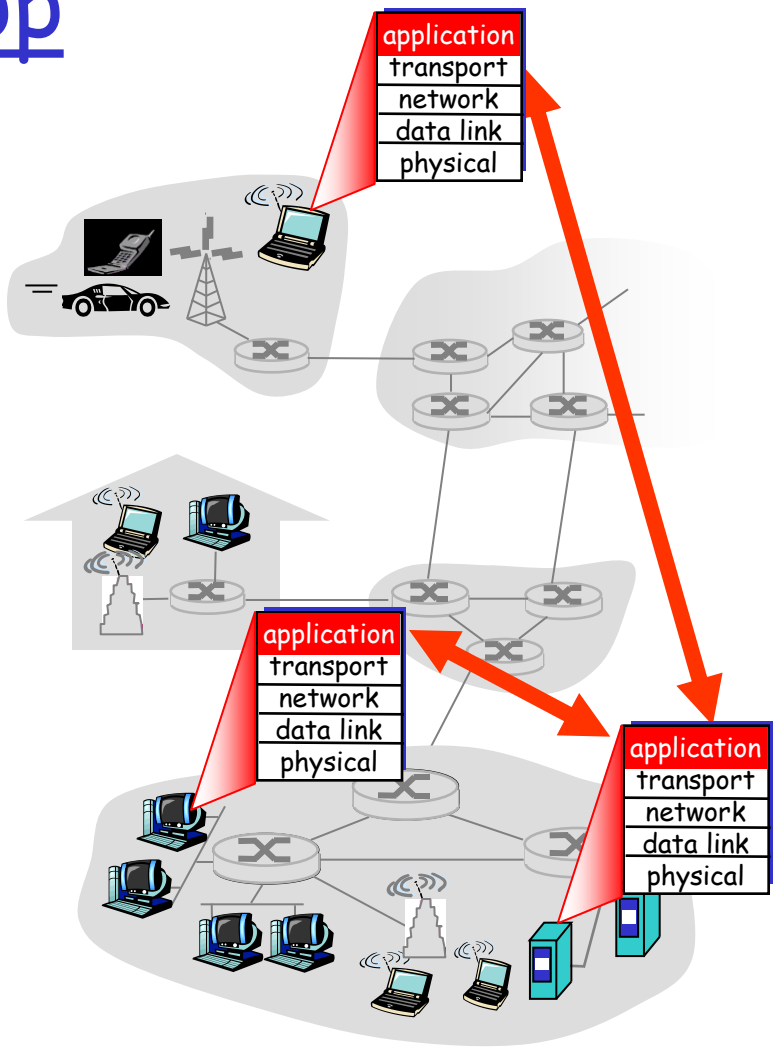
Creating a network app

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

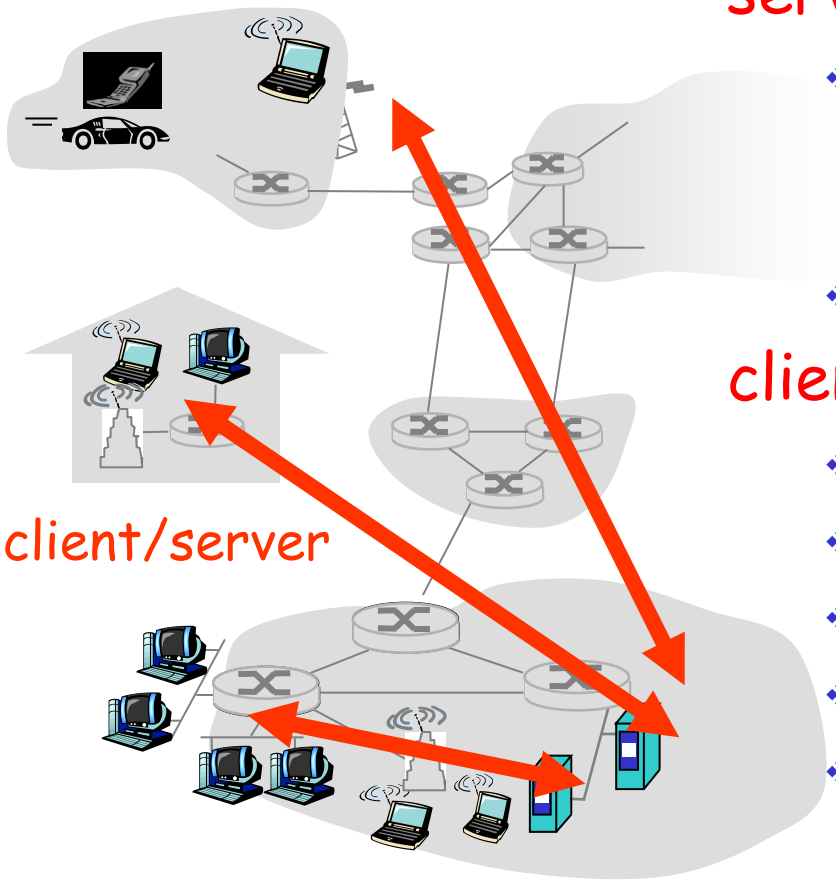
- Network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



Chapter 2: Application layer

- 2.1 Client/ Server network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server

Client-server architecture (on different machines)



server:

- always-on waiting for a request from client

  permanent IP address
- provides some service

clients:

- interested in the server's service
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Client-Server Model

- Typical scenario
  - The server process starts on some computer system
    - Initializes itself, then goes to sleep waiting for a client request
  - A client process starts, either on the same system or on some other system
    - Sends a request to the server
    - When the server process has finished providing its service to the client and the server goes back to sleep, waiting for the next client request to arrive
  - The process repeats

Types of Servers

- Two types of servers:
  - Iterative servers
    - When the server knows in advance how long it takes to handle each request. There is a single copy of the server which provides service in an iterative manner
  - Concurrent servers
    - Used when the service time is unpredictable and may be large. The server creates a copy of itself to cater to a client's request in a dedicated fashion. As many copies of server as there are client requests.

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 achieved by sockets

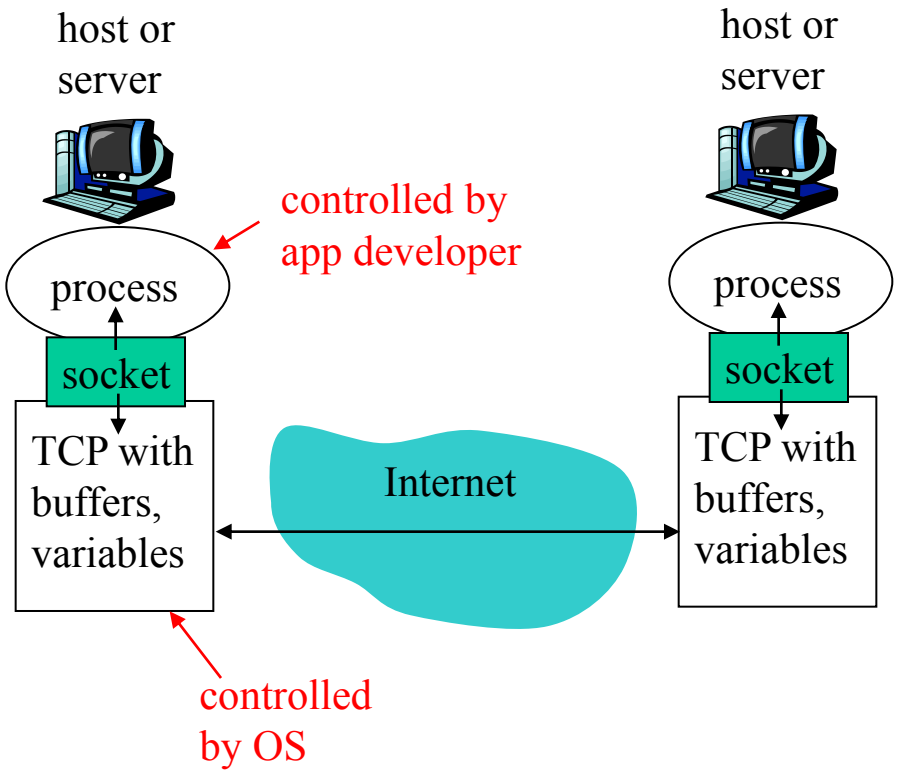
Socket is an association consisting of:

- Protocol, local IP address, local port number
- Protocol, remote IP address, remote port number

- Note: applications with P2P architectures have client processes & server processes

Sockets (from connection)

- 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

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"

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 of 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
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 throughput guarantees, security

UDP service:

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

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 (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Using TCP or UDP

- Before start of communication, a connection has to be established between the two hosts
- Five components in a connection
  - Protocol used
  - Source IP address
  - Source port number
  - Destination IP address
  - Destination port number

Network Programming in Java

- Network programs have to access external data to accomplish their goals
- Java provides a number of ways for accessing external data
  - Handled in a very uniform way
  - An object from which we can read a sequence of bytes is called an input stream
  - An object to which we can write a sequence of bytes is called an output stream

- Input and output streams are implemented in Java as part of the abstract classes InputStream and OutputStream
  - Concept of input stream can be used to abstract almost any kind of input: keyboard, file, network socket etc
  - Similarly, an output stream can be the screen, file, network socket etc
- Java provides a large number of concrete subclasses of InputStream and OutputStream to handle a wide variety of input-output options

Using DataInputStream

- Many applications need to read in an entire line of text at a time
  - DataInputStream class and its readLine() method can be used
  - The readLine() method reads in a line of ASCII text and converts it into a Unicode string

```
DataInputStream inp = new DataInputStream (new
        FileInputStream("student.dat"));
String line = inp.readLine();
```

Network Programming Features

- Java can be used easily to develop network applications
  - It comes with a very powerful class library for networking, as part of java.net package.
  - It supports both the TCP and UDP protocol families
- A simple example is shown next
  - (client) Connects to a specified host (server) over a specified port, and prints whatever is returned

Client Program

Notes

- All networking code are enclosed in the try … catch block
- Most of the network-related methods throw IOException whenever some error occurs

Implementing Server

- This server;
  - Sends a welcome message to the client after it connects
  - Expects some data from the client and echos the client data as long as client is sending the data

Server Program

Some Points

- Once the accept() call returns a Socket object newsock, the getInputStream() and getOutputStream() methods are used to get an input stream and an output stream respectively from that socket
- Everything that the server program sends to the output stream becomes the input of the client program
- Outputs of the client program become the input stream of the server

How to Test Server?

- Alternative 1
  - Write a client program to connect to the server on the particular port
- Alternative 2
  - Use telnet command
    - telnet 127.0.0.1 7500

Writing Concurrent Servers

- What is a concurrent server?
  - Can handle multiple client requests at the same time
- Java threads can be used
  - Every time a client establishes a connection with the server, the accept() call will return
  - At this time a new thread is created to handle the client connection
  - The main thread will go back and wait for the next connection

Disclaimer

Java Network Programming slides are adopted from the Lecture Series on Internet Technologies by Prof. I. Sengupta, Department of Computer Science Engineering, IIT Kharagpur.