Distributed Systems

2. Application Layer

Werner Nutt

Network Applications: Examples

- E-mail
- Web
- Instant messaging
- Remote login
- P2P file sharing
- Multi-user network games
- Streaming stored video clips
- Social networks
- Voice over IP
- Real-time video conferencing
- Grid computing

Creating a Network App

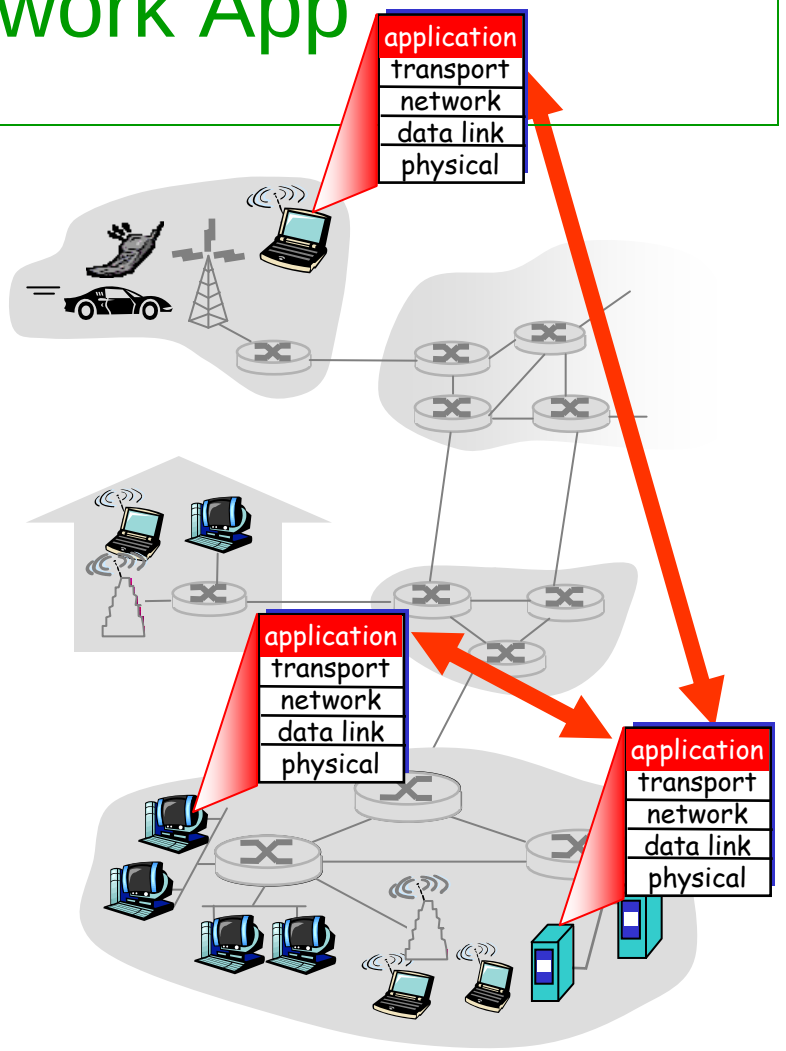
Write programs that

- 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

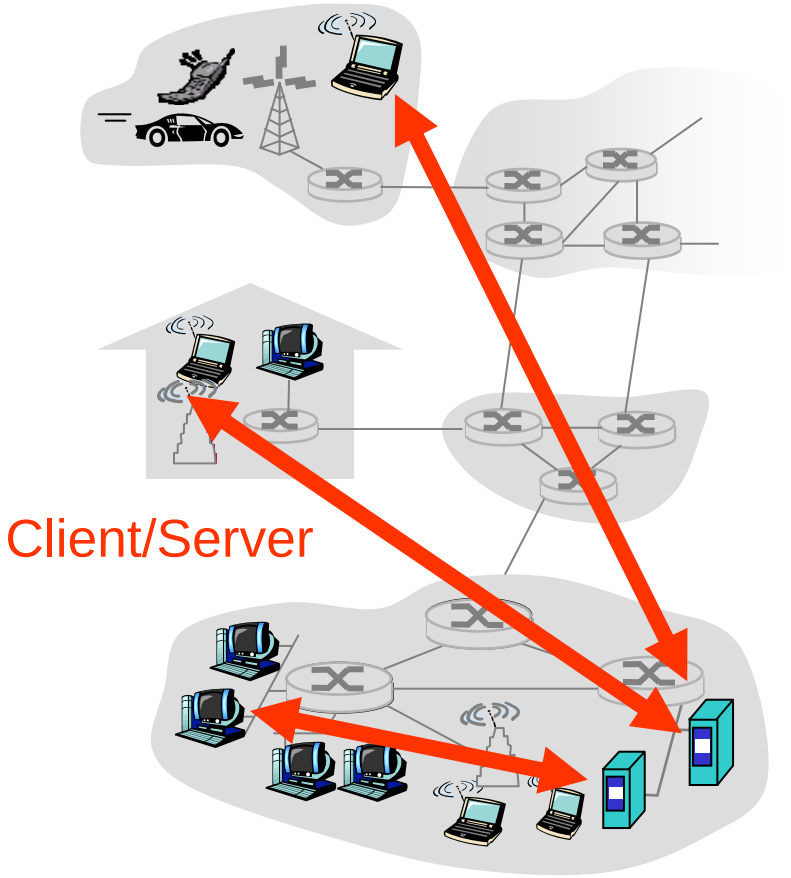
- Network-core devices do not run user applications
- applications on end systems allows for rapid application development, propagation



Application Architectures

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

Client-server Architecture



Server:

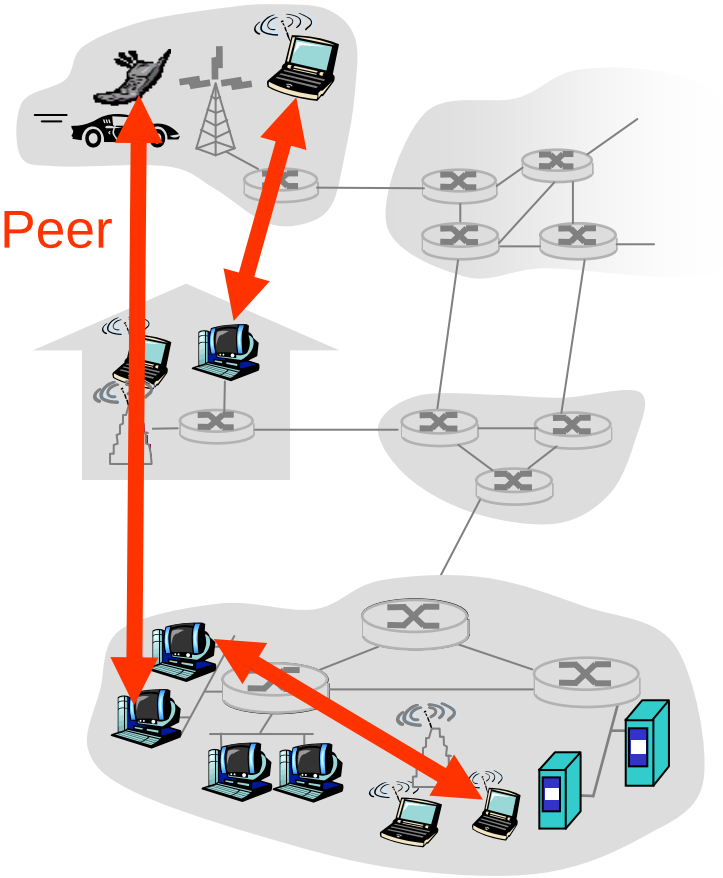
- always-on host
- permanent IP address
- server farms for scaling

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



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: 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

Communication Between Processes

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

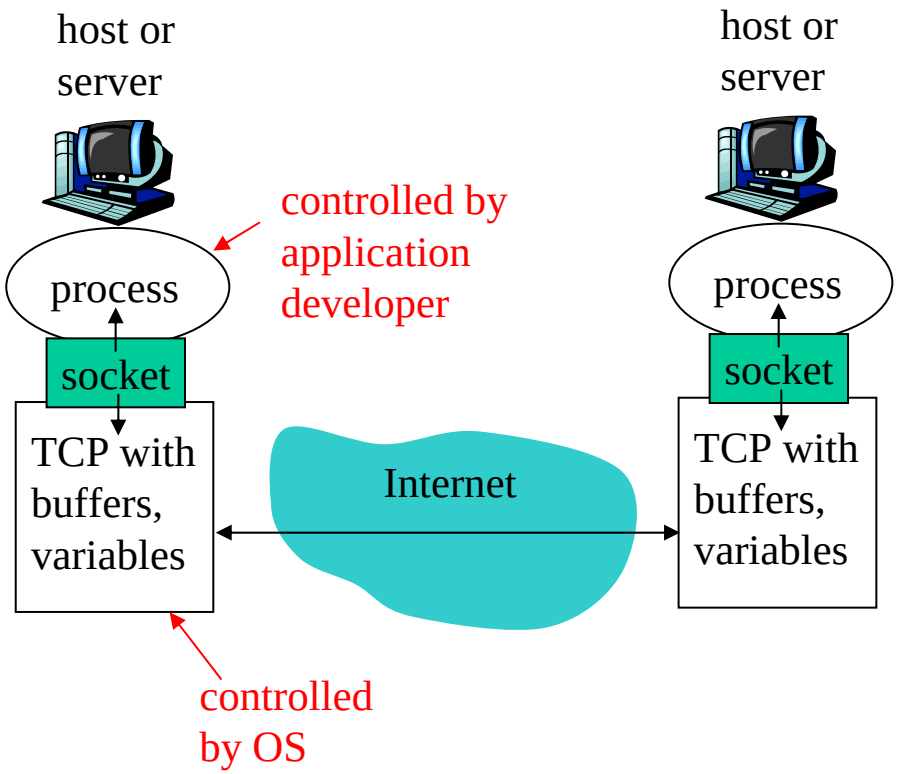
Client process: process that *initiates* communication

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 allows: (1) choice of transport protocol

(2) ability to fix a few parameters (more on this later)

Addressing Processes

- To receive messages, a process must have an *identifier*
- A host device has a unique 32-bit *IP address*
- Exercise: Find out the IP address of your laptop/desktop

- Does the IP address of the host on which a process runs suffice for identifying the process?

  . . .
- *Identifier* includes both IP address and port numbers associated with processes on the host
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25

Assigned Port Numbers

FTP Data	20
FTP Control	21
SSH	22
Telnet	23
SMTP	25
Domain Name Server	42
Whois	43
HTTP	80
POP3	110
IMAP4	143
BGP	179
HTTPS	443
IMAP4 over SSL	993

Assigned by IANA
(= Internet Assigned
Numbers Authority)

Numbers between 0 and 1023 are "well-known" ports — opening a port for such numbers requires privileges

can be found
- on the Web
- in "/etc/services"
under Linux and MAC/OS

Application Layer Protocols Define …

- Types of messages exchanged,
  - e.g., request, response
- Message syntax:
  - fields in messages and 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, BitTorrent

Proprietary protocols:

- e.g., Skype

What Transport Service Does an Application Need?

Data loss

- Some loss can be tolerated: audio, video
- 100% reliability needed: file transfer, telnet

Timing

- Delays can be tolerated: file transfer
- Low delays needed: internet telephony, interactive games

Data Throughput

- Ineffective with throughput below minimum: multimedia
- Run with whatever is offered: “elastic applications”

Security

- Encryption, data integrity, …

Transport Service Requirements of Common Applications

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 Protocol 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

Why bother?

Why is there a UDP?

Applications and Transport Protocols

Application	Application-layer protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Streaming multimedia	HTTP, RTP	TCP or UDP
Internet telephony	SIP, proprietary	typically UDP
Network management	SNMP	typically UDP
Routing protocol	RIP, OSPF	typically UDP
Name translation	DNS	typically UDP