

Lecture 03: Application Layer Intro

CS 326E Elements of Networking

Mikyung Han

mhan@cs.utexas.edu

Example Protocols

FTP, HTTP, SMTP

Application

TCP, UDP

Transport

IP

Network

Ethernet, WiFi

Link

802.3 PHY

Physical

Responsible for

application specific needs

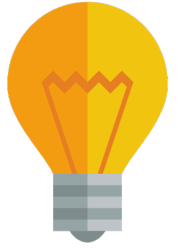
process to process data transfer

host to host data transfer across different network

data transfer between physically adjacent nodes

bit-by-bit or symbol-by-symbol delivery

Internet Reference Model



Outline



1. Design point of view: End-to-end argument
2. Architecture point of view: Server/client vs peer-to-peer
3. Maintenance point of view: Stateless protocol vs Stateful protocol
4. OS point of view: Network application as a process

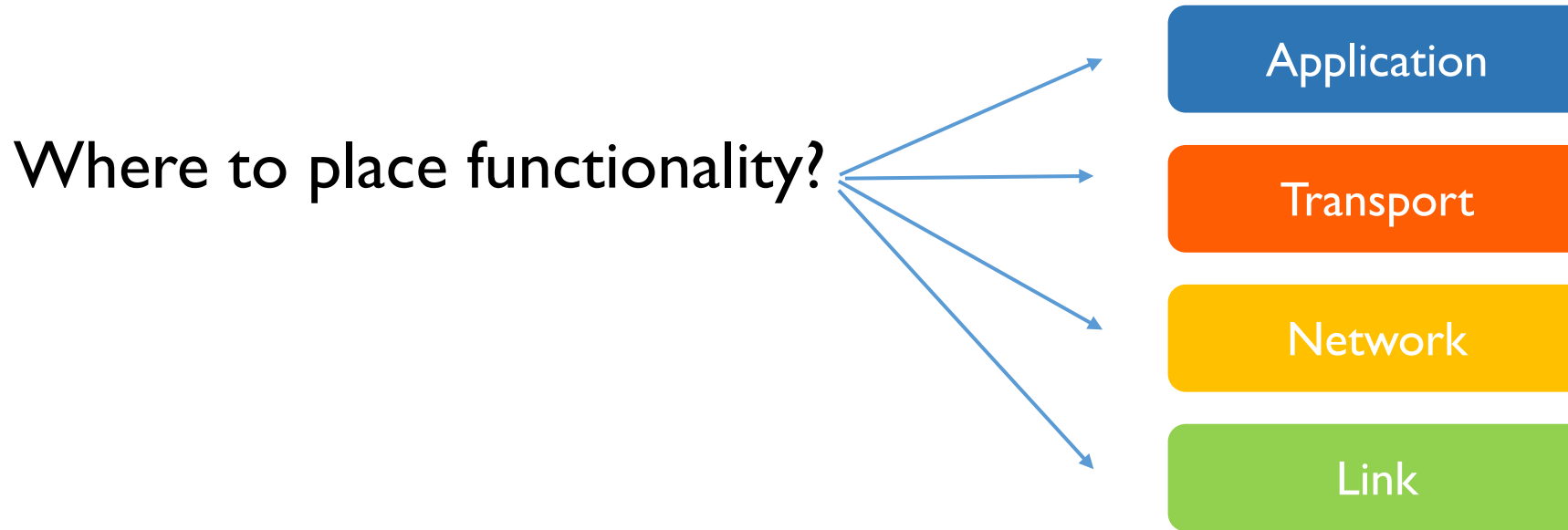
Imagine yourself as one of the system designers of the Internet



Liba Svobodova (left)
David D. Clark (mid)
Jerome H. Saltzer (right)
David P. Reed (below)



According to end-to-end argument: Not at the Core But at the Edges!



Saltzer, Reed, Clark advocated for dumb network and intelligent endpoints

- “The application knows best.”
- “Functionality should be implemented at a lower layer if and only if it can be correctly and completely implemented there”
 - Avoid at lower level if redundant with higher level
 - Performance optimizations are not a violation

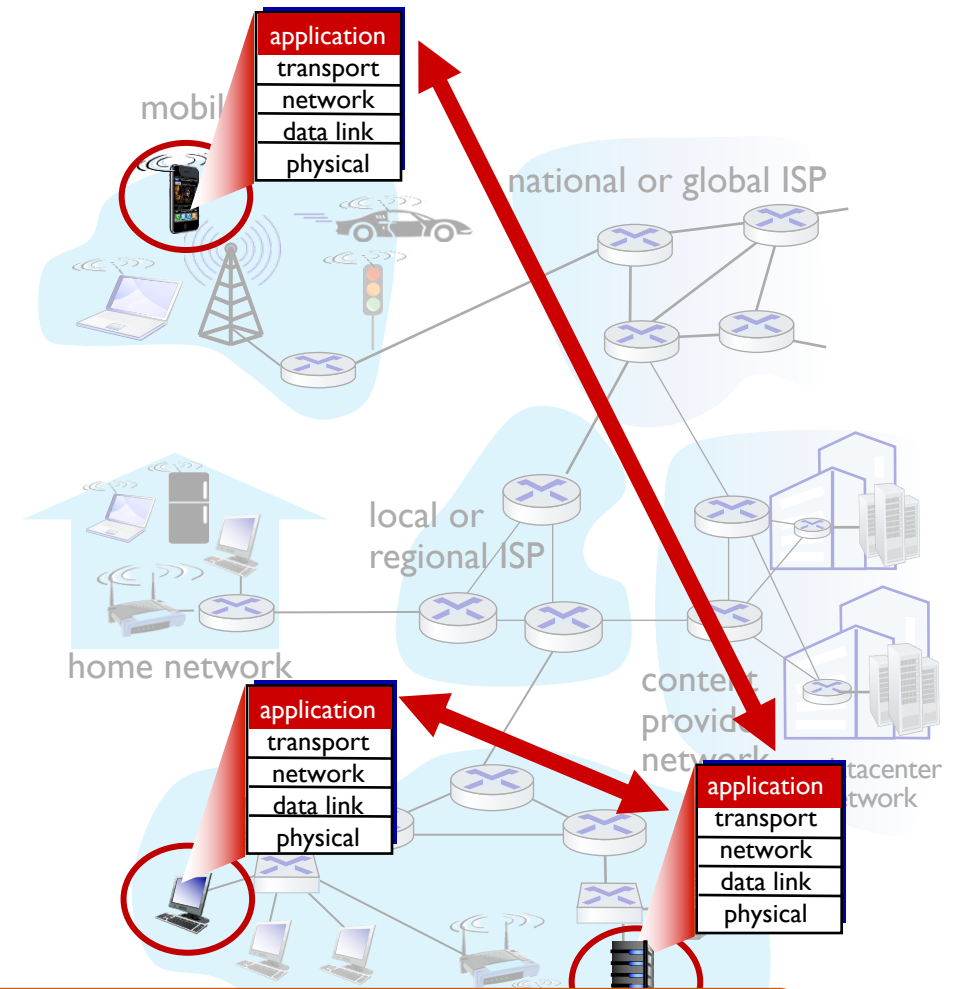
Reliable File Transfer:

What can go wrong when sending a file over a network?

- Disk can introduce bit errors
- Host I/O buses can introduce bit errors
- Packets can get garbled, dropped, mis-ordered at any hop
- Checking correctness at each step/hop is redundant
- **Solution: integrity check on file by application**

Applications only run on the endpoints!

- Network core devices do NOT run user applications
 - No code to write for these 😊
- When developing an app, we only need to consider the two ends
 - server/client or peers



This allowed rapid app development and propagation

Outline

1. Design point of view: End-to-end argument
-  2. Architecture point of view: Server/client vs peer-to-peer
3. Maintenance point of view: Stateless protocol vs Stateful protocol
4. OS point of view: Network application as a process

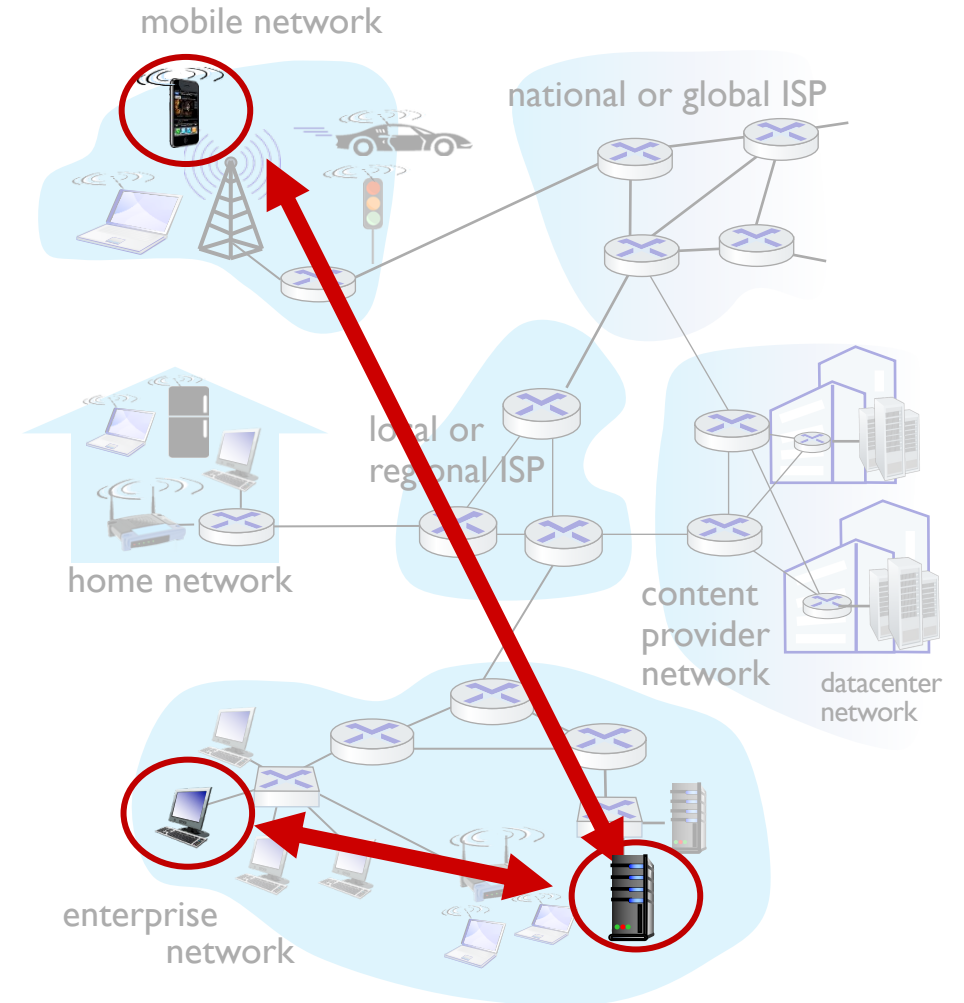
Client-server model

server:

- always-on host
- permanent IP address
- often in data centers, for scaling

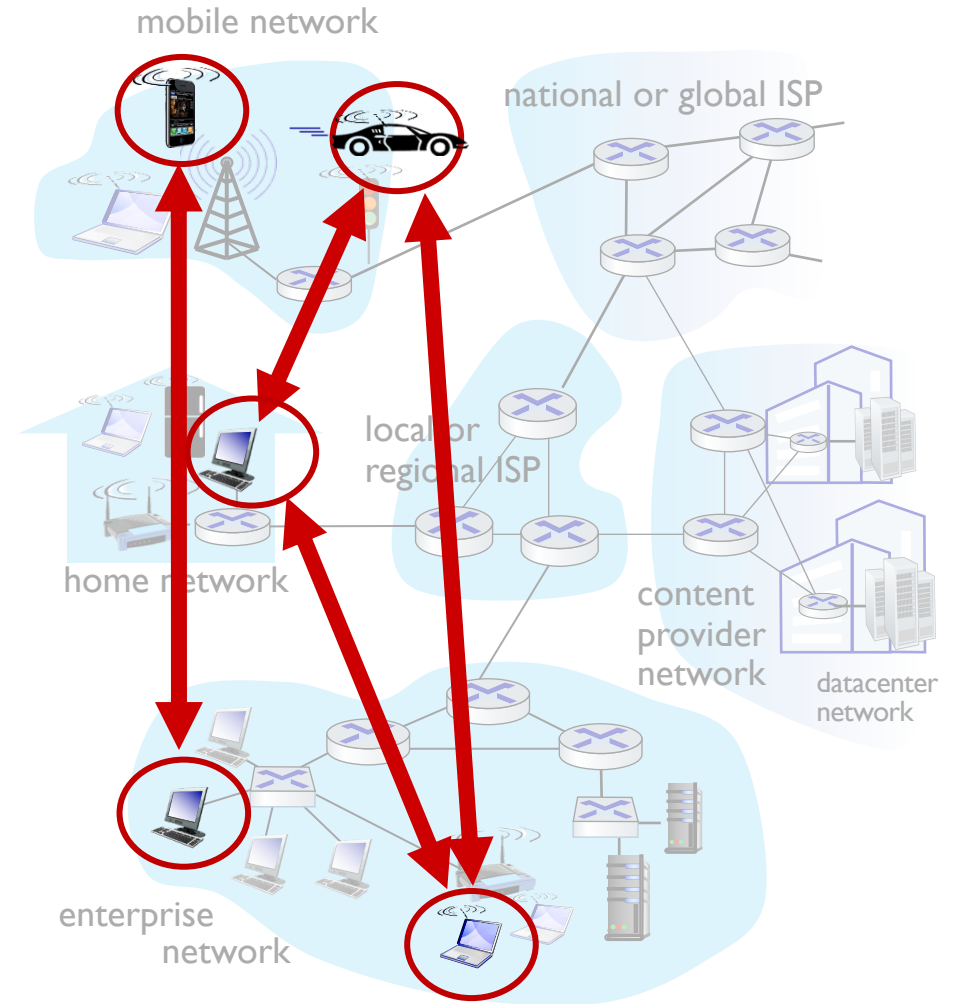
clients:

- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
- examples: HTTP, IMAP, FTP



Peer-to-peer model

- no always-on server
- arbitrary end systems directly communicate
- **Self scalability** - new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
- example: Gnutella, BitTorrent



Outline

1. Design point of view: End-to-end argument
2. Architecture point of view: Server/client vs peer-to-peer
-  3. Maintenance point of view: Stateless protocol vs Stateful protocol
4. OS point of view: Network application as a process

A **stateless** protocol does not store any “state”

- No session information is retained by the server or the client (or peers)
- Does not track “state” of each other
- Each request/response pair is independent of each other
- No need to do recovery from a partially-completed transaction
- Ex) HTTP, IP, UDP

Wait, is HTTP a really stateless protocol?

A **stateful** protocol does store and maintain “states”

- Typically, the server keeps track of session info for each client (Or peers keep track of session info of others)
- The request has to be understood within a context based on previous history
- When one crash, need to handle the recovery from partially completed session

Can stateless protocol be used on top of stateful one?

- Vice-versa?

Yes! Mix-n-match is possible!

- HTTP – stateless
- TCP – stateful
- IP – stateless
- 802.11 – stateful



Encapsulation of layering enables it!

Outline

1. Design point of view: End-to-end argument
2. Architecture point of view: Server/client vs peer-to-peer
3. Management point of view: Stateless protocol vs Stateful protocol
-  4. OS point of view: Network application as a process

Net applications are two processes communicating over network by exchanging messages

Process? A program running within a host

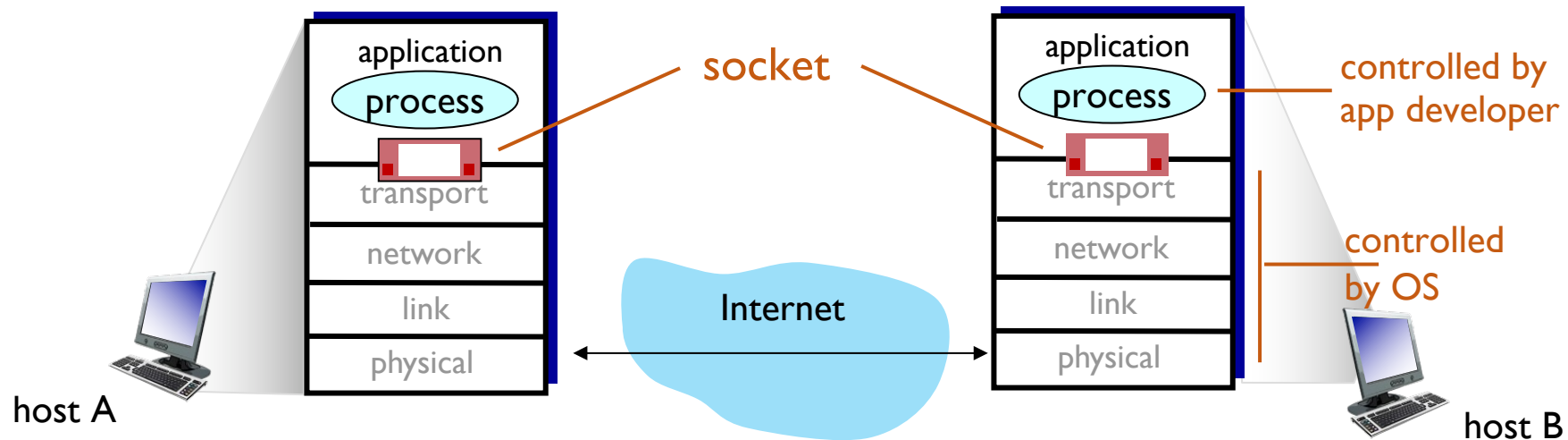
- processes within the same hosts communicate using **inter-process communication** (defined by OS)
- processes in different hosts communicate by exchanging **messages across network**

client process: process that initiates communication

server process: process that waits to be contacted

What is a Socket?

- process sends/receives messages to/from its **socket**
- socket analogous to a “door”
 - sending process shoves message out the door
 - sending process relies on transport layer to deliver message to socket at receiving process
 - two sockets involved: one on each end



Since many processes run on the same host thus, socket identifier must include **IP and port number**

- example port numbers:
 - HTTP server: 80
 - mail server: 25
- to send HTTP message to `www.cs.utexas.edu` web server:
 - **IP address:** 128.83.120.48
 - **port number:** 80

In summary, network application vs socket vs port

- Network app is a process that runs on an end-host
- Network app sends/recvs messages to/from transport layer via socket
 - Sockets are the two endpoints of transport layer
- Can one network application may have multiple sockets?
 - Why or why not?
- End-host can be identified by a IP address
 - Can one host have many IP addresses?
- Sockets are identified by IP + port number
 - More detail to come!

Resources

- [Socket identification in TCP/HTTP](#)

Resources

Acknowledgements

Slides are adopted from Kurose' Computer Networking Slides