Distributed Systems

15-440/640

Fall 2021

Lecture 3 – Communication:
The Internet in a Day (ctnd)

Announcements

- Lectures will be recorded by staff going forward
- Project P0 released yesterday (Sept 6th)
  - Due Sept. 16th, 2021
- Recitations
  - This week –to go over the basics of Golang
    - Look at the recitation you are registered for and go to that one
    - We will be recording the Recitations sessions
  - Prepare by going over the Tour of Go

    https://tour.golang.org
  - Installation of Go

    https://golang.org/doc/install

What you learned so far

Network links and LANs

Inter-network Communication

Layering & Protocols

Internet design

Transport protocols

Application design

What you will learn next

Network links and LANs

Inter-network Communication

Layering & Protocols

Internet design

Transport protocols

Application design

Network Service Model

- What is the *service model* for inter-network*?*
  - Defines what promises that the network gives for any transmission
  - Defines what type of failures to expect

- ***best-effort***

  - Ethernet/Internet– packets can get lost, etc.

⇒ Development of “failure models” in DS design

Possible Failure models

- Fail-stop:
  - When something goes wrong, the process stops / crashes / etc.
- Byzantine:
  - Anything that can go wrong, will.
  - Including malicious entities taking over your computers and making them do whatever they want.
- Fail-slow or fail-stutter:
  - Performance may vary on failures

- These models are useful for proving things;
- The real world typically has a bit of everything.

- Deciding which model to use is important!

Fancier Network Service Models

- What if you want more?

  - Performance guarantees (QoS)
  - Reliability
    - Corruption
    - Lost packets
  - Flow and congestion control
  - Fragmentation
  - In-order delivery
  - Etc…

If network provides this ⇒ reuse across applications

How would you implement these?

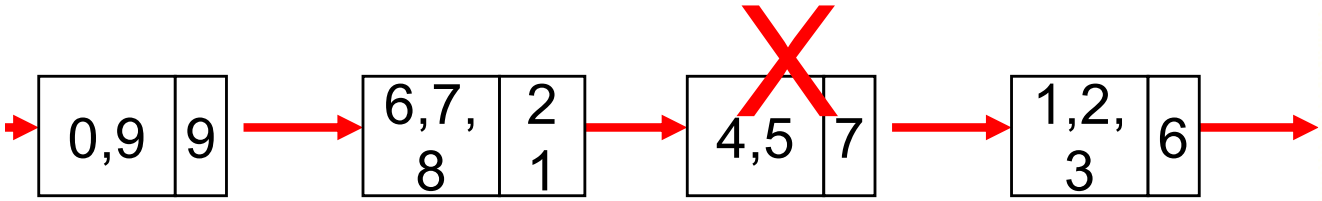
What if the Data gets Corrupted?

Problem: Data Corruption

GET
index.html

GET
inrex.html

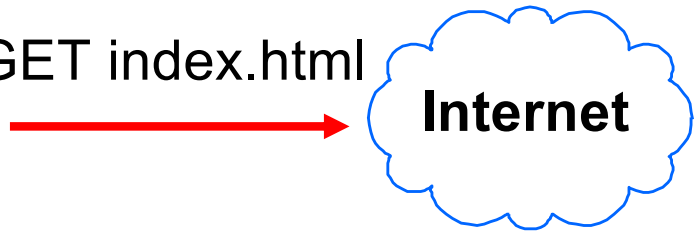
Solution: Add a *checksum*



What if the Data gets Lost?

Problem: Lost Data

GET index.html



Solution: Timeout and Retransmit

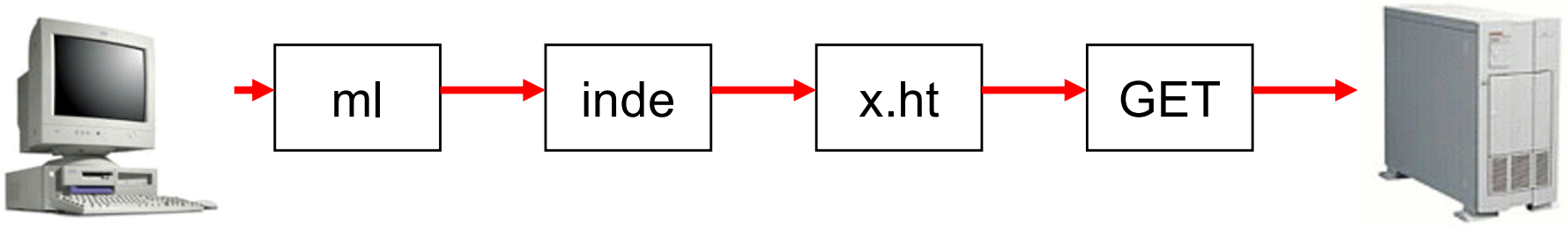
GET index.html

GET index.html

GET index.html

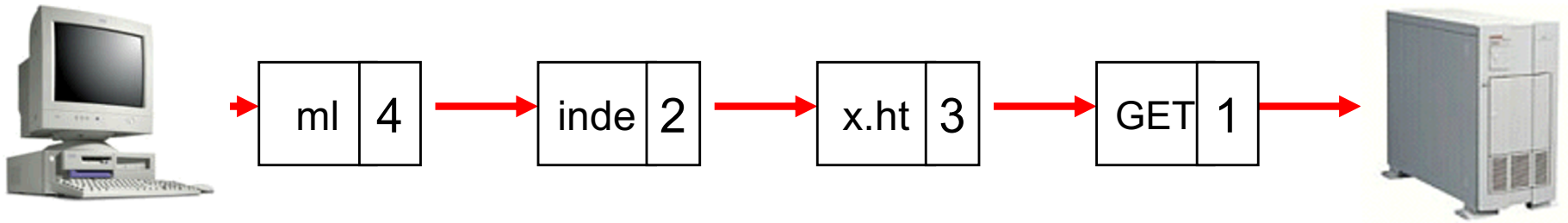
What if the Data is Out of Order?

Problem: Out of Order



GET x.htindeml

Solution: Add Sequence Numbers



GET index.html

Networks [including end points] Implement Many Functions

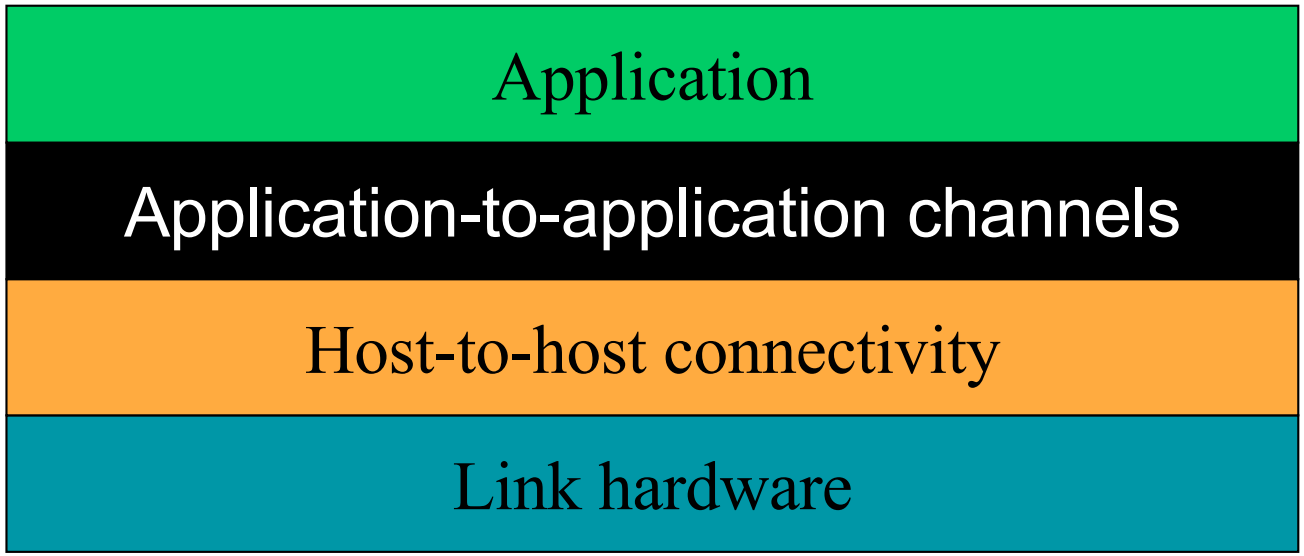
- Link
- Multiplexing
- Routing
- Addressing/naming (locating peers)
- Reliability
- Flow control
- Fragmentation
- Etc.…

But note limitations: these can't turn a byzantine failure model into a fail-stop model!

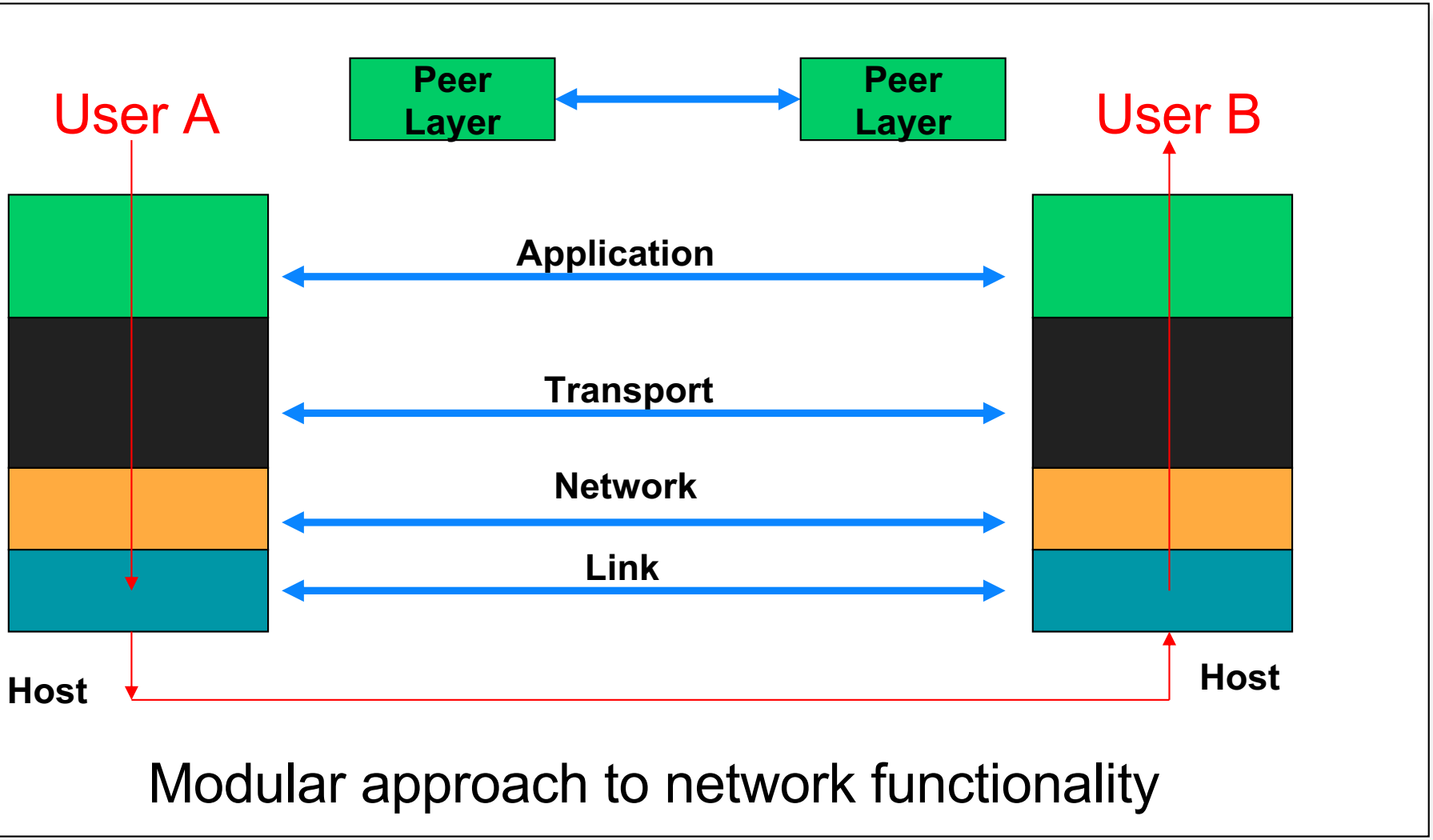
Where would you implement each function?

What is Layering?

- Modular approach to network functionality
- Example:



What is Layering?



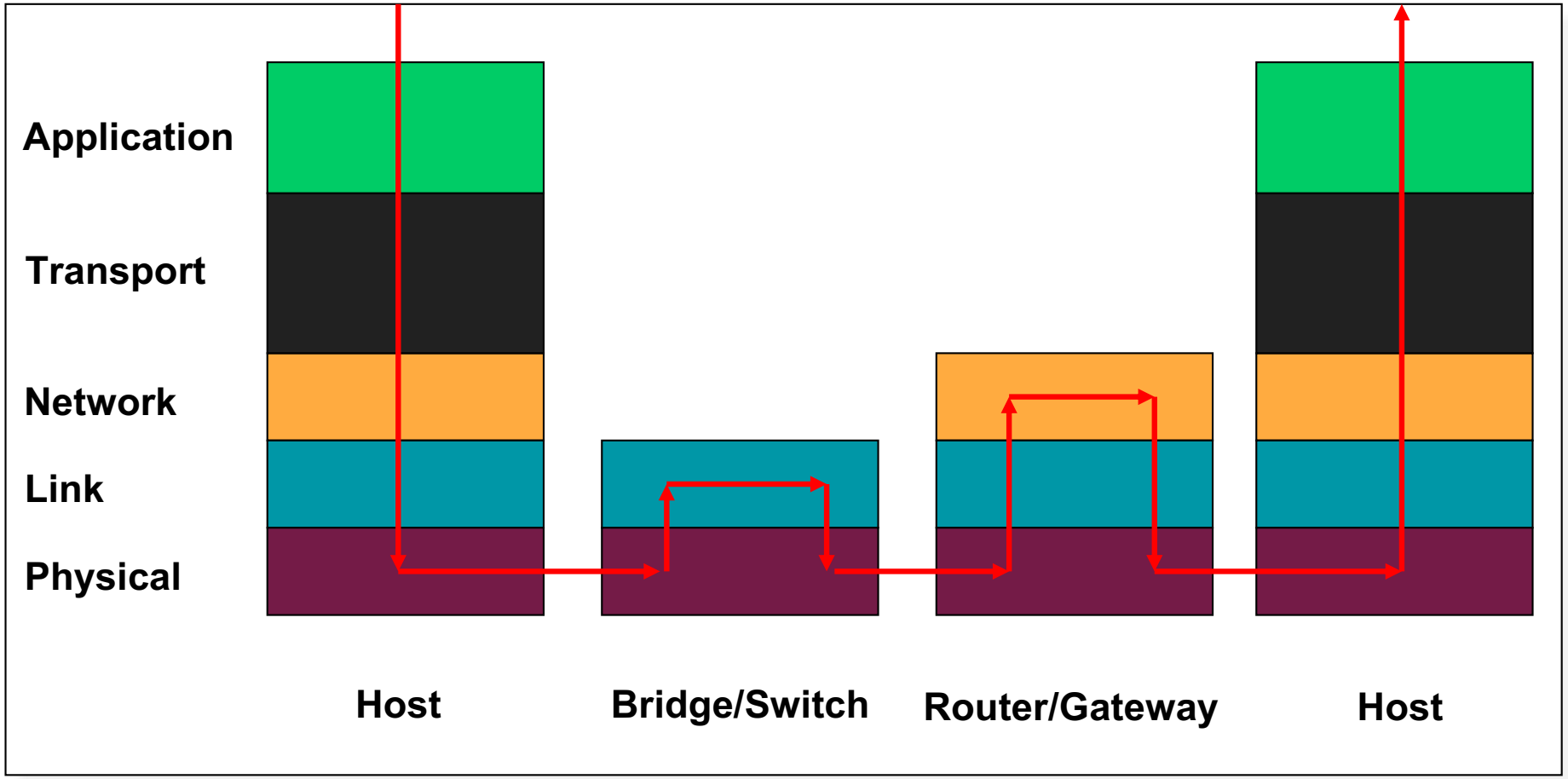
Modular approach to network functionality

Layering Characteristics

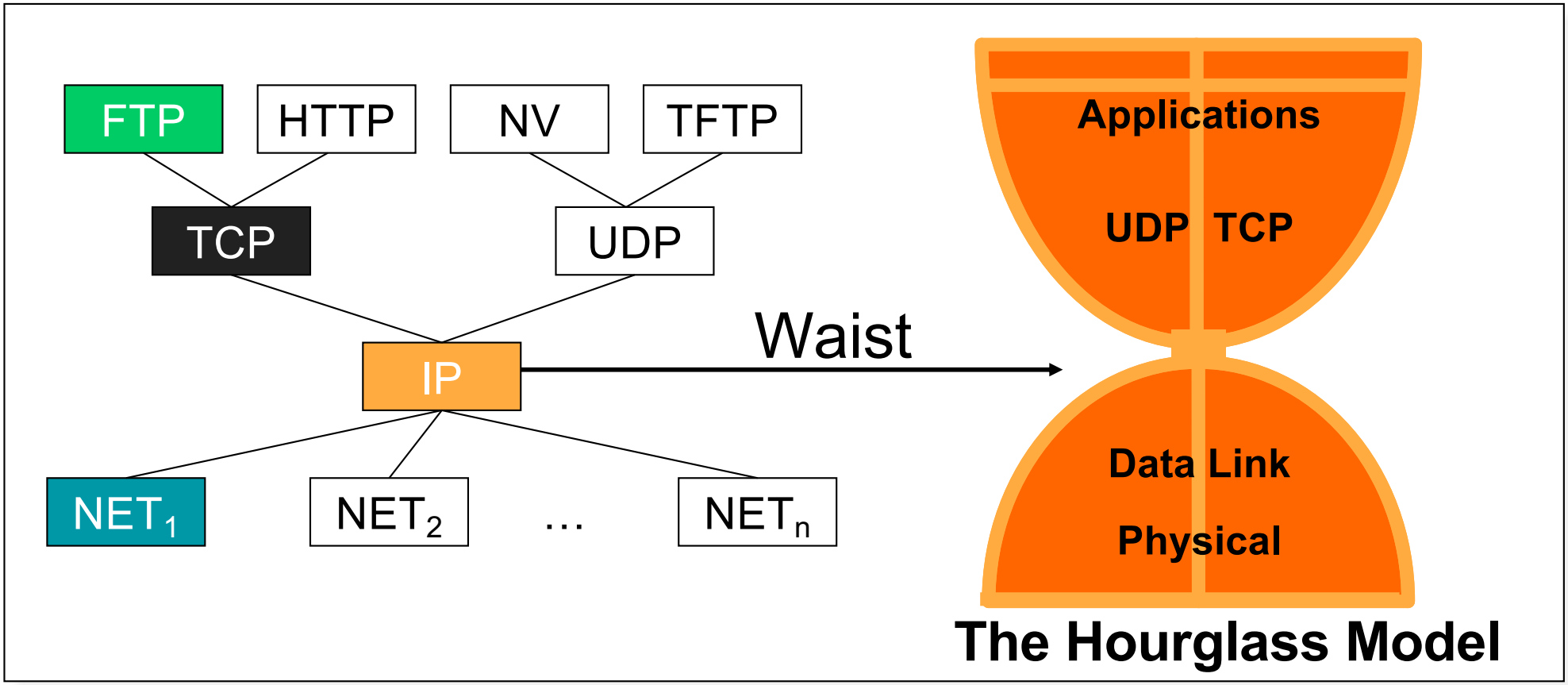
- Each layer relies on services from layer below and exports services to layer above
- Interface defines interaction with peer on other hosts
- **Protocols** define:
  - Interface to higher layers (API)
  - Interface to peer (syntax & semantics)
    - Actions taken on receipt of a messages
    - Format and order of messages
    - Error handling, termination, etc.
- Hides implementation - layers can change without disturbing other layers (black box)

Internet Protocol Layering

- Relatively simple



The Internet Protocol Suite

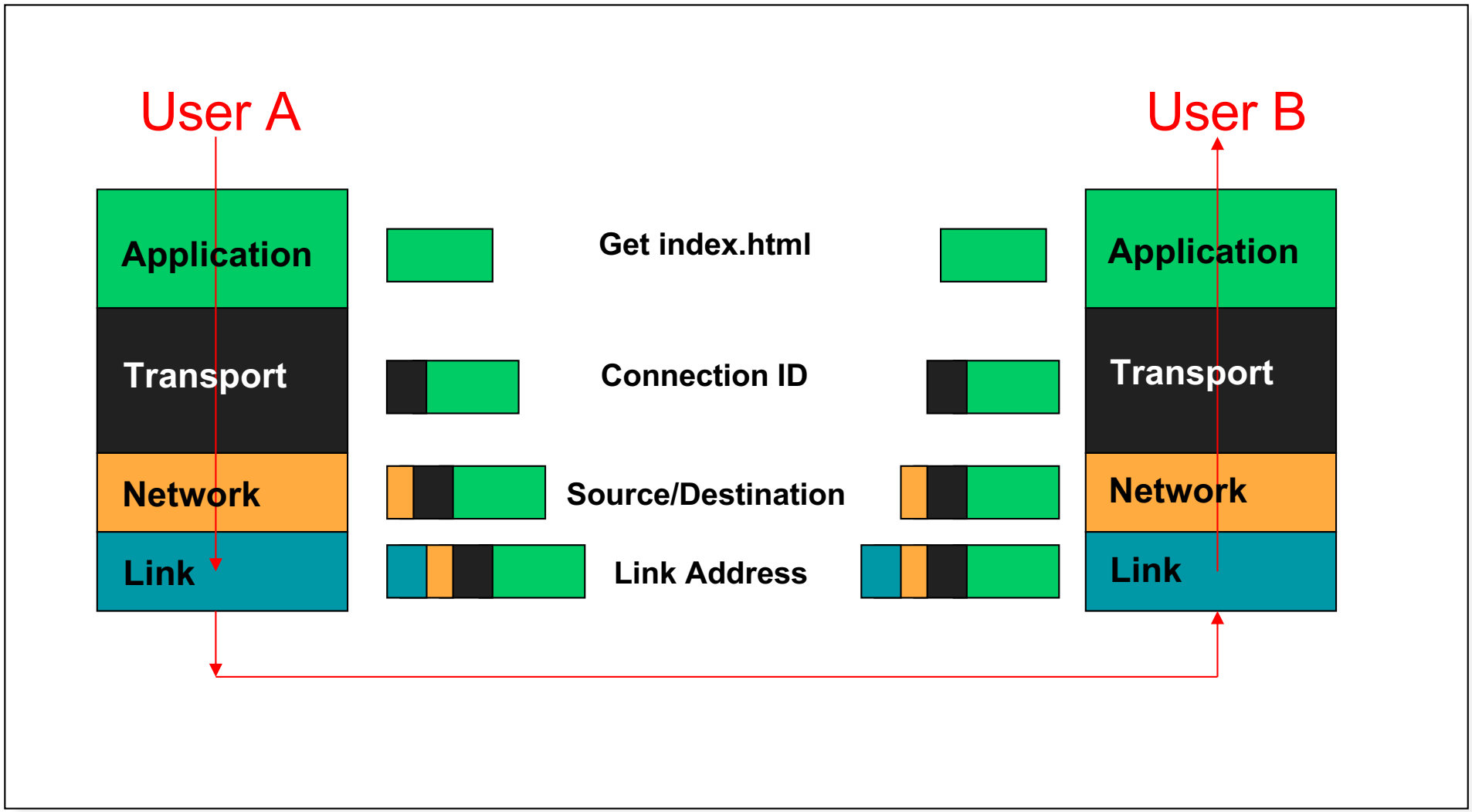


The waist facilitates interoperability:

IP over anything, anything over IP

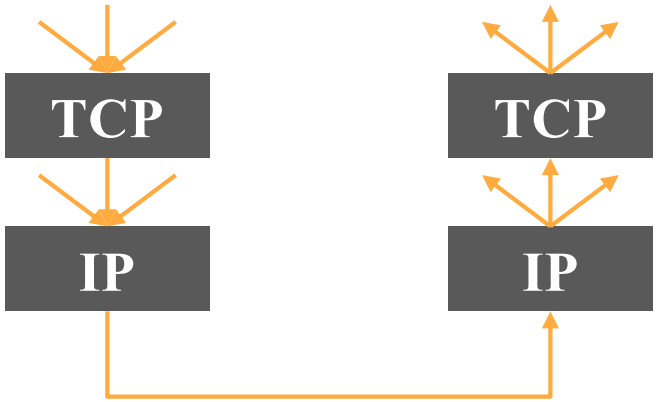
What's the disadvantage of the "IP waist"?

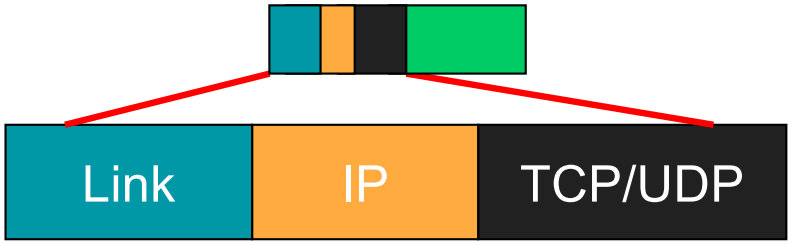
Layer Encapsulation



Multiplexing and Demultiplexing

- There may be multiple implementations of each layer.
  - How does the receiver know what version of a layer to use?
- Each header includes a demultiplexing field that is used to identify the next layer.
  - Filled in by the sender
  - Used by the receiver





Type Field | Protocol Field | Port Number

Recall: IP header

V/HL	TOS	Length
ID		Flags/Offset
TTL	Prot.	H. Checksum
Source IP address		
Destination IP address		
Options..		

Today's Lecture

Network links and LANs

Inter-network Communication

Layering & Protocols

Internet design

Transport protocols

Application design

Goals [Clark88]

0 **Connect existing networks**

1. Survivability

   ensure communication service even in the presence of network and router failures

2. Support multiple types of services
3. Must accommodate a variety of networks
4. Allow distributed management
5. Allow host attachment with a low level of effort
6. Be cost effective
7. Allow resource accountability

Goal 1: Survivability

- If network is disrupted and reconfigured…
  - Communicating entities should not care!
  - No higher-level state reconfiguration
- How to achieve such reliability?
  - Where can communication state be stored?

	Network	Host
Failure handing	Replication	“Fate sharing”
Net Engineering	Tough	Simple
Switches	Maintain state	Stateless
Host trust	Less	More

Fate Sharing

- Lose state information for an entity if and only if the entity itself is lost.
- Examples:
  - OK to lose TCP state if one endpoint crashes
    - NOT okay to lose if an intermediate router reboots
- Tradeoff
  - Survivability: Heterogeneous network → less information available to end hosts and Internet level recovery mechanisms

End-to-End Argument/Reasoning

- Deals with where to place functionality
  - Inside the network (in switching elements)
  - At the edges

- Argument
  - If you have to implement a function end-to-end anyway (e.g., because it requires the knowledge and help of the end-point host or application), **don't implement it inside the communication system**
  - Unless there's a compelling performance enhancement

- Key motivation for split of functionality between TCP,UPD and IP

Further Reading: "End-to-End Arguments in System Design." Saltzer, Reed, and Clark.

Today's Lecture

Network links and LANs

Inter-network Communication

Layering & Protocols

Internet design

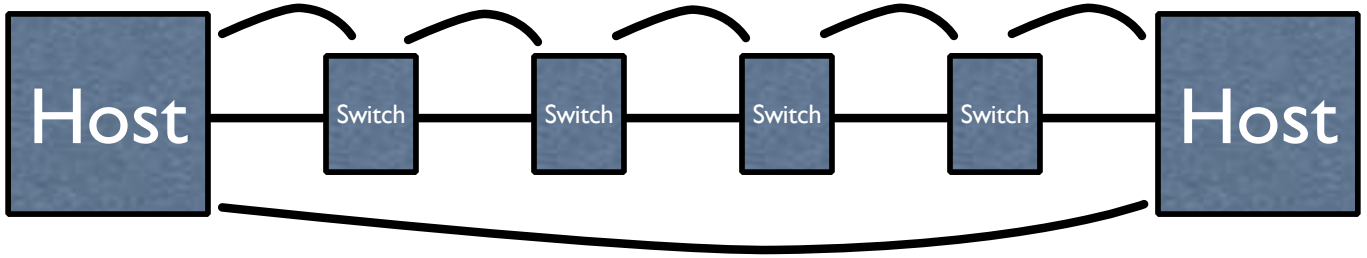
Transport protocols

Application design

Design Question

- If you want reliability, where should you implement it?

Option 1: Hop-by-hop



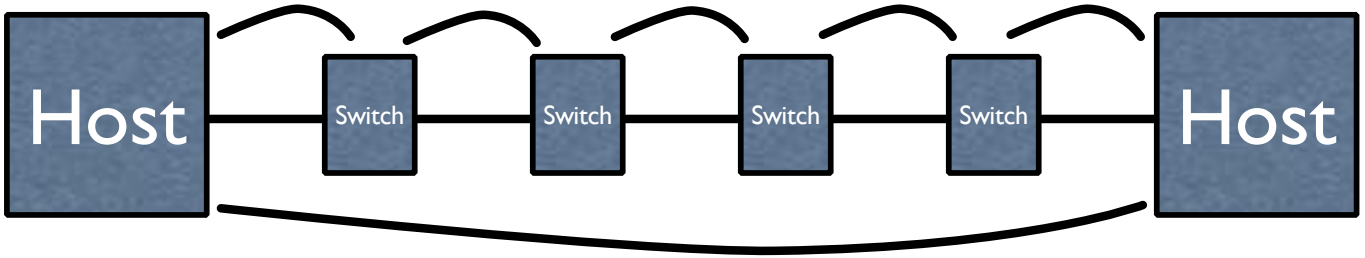
Option 2: end-to-end

Hint: End-to-end argument

Design Question

- If you want reliability, where should you implement it?

Option 1: Hop-by-hop



Option 2: end-to-end

Needs careful thought (decision to be based on performance rather than correctness)

- Sometimes adding functionality at a lower level might be more efficient. Example?
- Sometimes might decrease efficiency since lower level subsystems common to many applications. Example?

Transport Protocols

- Types of Service
  - Elastic apps that need reliability:
    - remote login or email
  - Inelastic, loss-tolerant apps:
    - real-time voice or video
  - Others in between, or with stronger requirements
  - Biggest cause of delay variation: reliable delivery
    - Today's net: ~100ms RTT
    - Reliable delivery can add *seconds*.
- Original Internet model: "TCP/IP" one layer
  - First app was remote login…
  - But then came debugging, voice, etc.
  - These differences caused the layer split, added UDP

Transport Protocols

- UDP provides just basic functionality with demux
- TCP adds…
  - Connection-oriented
  - Reliable
  - Ordered
  - Point-to-point
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled

User Datagram Protocol (UDP): An Analogy

UDP	Postal Mail
• Single socket to receive messages • No guarantee of delivery • Not necessarily in-order delivery • Datagram – independent packets • Must address each packet	• Single mailbox to receive letters • Unreliable ☺ • Not necessarily in-order delivery • Letters sent independently • Must address each letter

Example UDP applications

Multimedia, voice over IP

Transmission Control Protocol (TCP): An Analogy

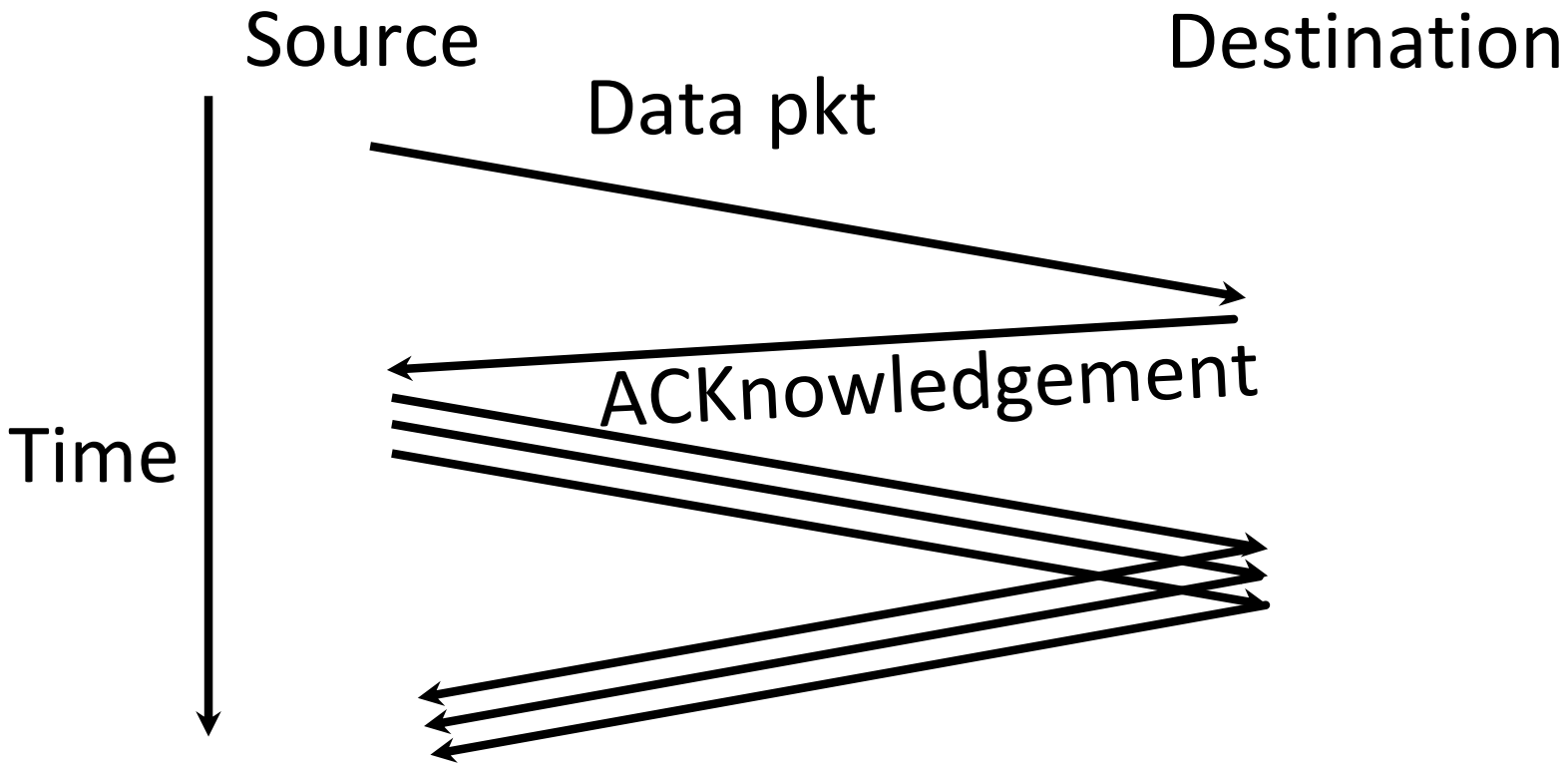
TCP	Telephone Call
• Reliable – guarantee delivery	• Guaranteed delivery
• Byte stream – in-order delivery	• In-order delivery
• Connection-oriented – single socket per connection	• Connection-oriented
• Setup connection followed by data transfer	• Setup connection followed by conversation

Example TCP applications

Web, Email, Telnet

Rough view of TCP

(This is a *very* incomplete view - take 15-441. :)



What TCP does:

1) Figures out which packets got through/lost

2) Figures out how fast to send packets to use all of the unused capacity,

- But not more

- And to share the link approx. equally with other senders

Questions to ponder

- If you have a whole file to transmit, how do you send it over the Internet?
  - You break it into packets (packet-switched medium)
  - TCP, roughly speaking, has the sender tell the receiver “got it!” every time it gets a packet. The sender uses this to make sure that the data’s getting through.
  - If you acknowledge the correct receipt of the entire file... why bother acknowledging the receipt of the individual packets???

- The answer: Imagine the waste if you had to retransmit the entire file because one packet was lost. Ow.

Today's Lecture

Network links and LANs

Inter-network Communication

Layering & Protocols

Internet design

Transport protocols

Application design

Today's Lecture

Network links and LANs

Inter-network Communication

Layering & Protocols

Internet design

Transport protocols

Application design

Client-Server Paradigm

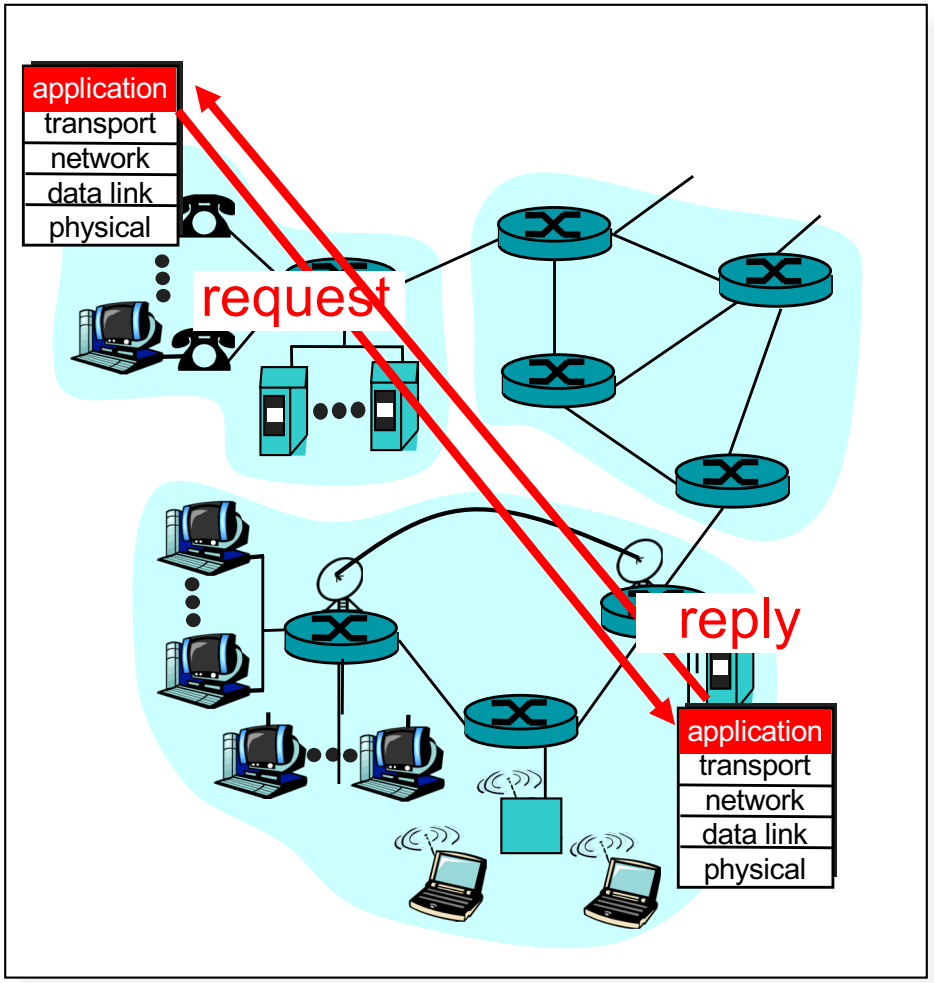
Typical network app has two pieces: *client* and *server*

Client:

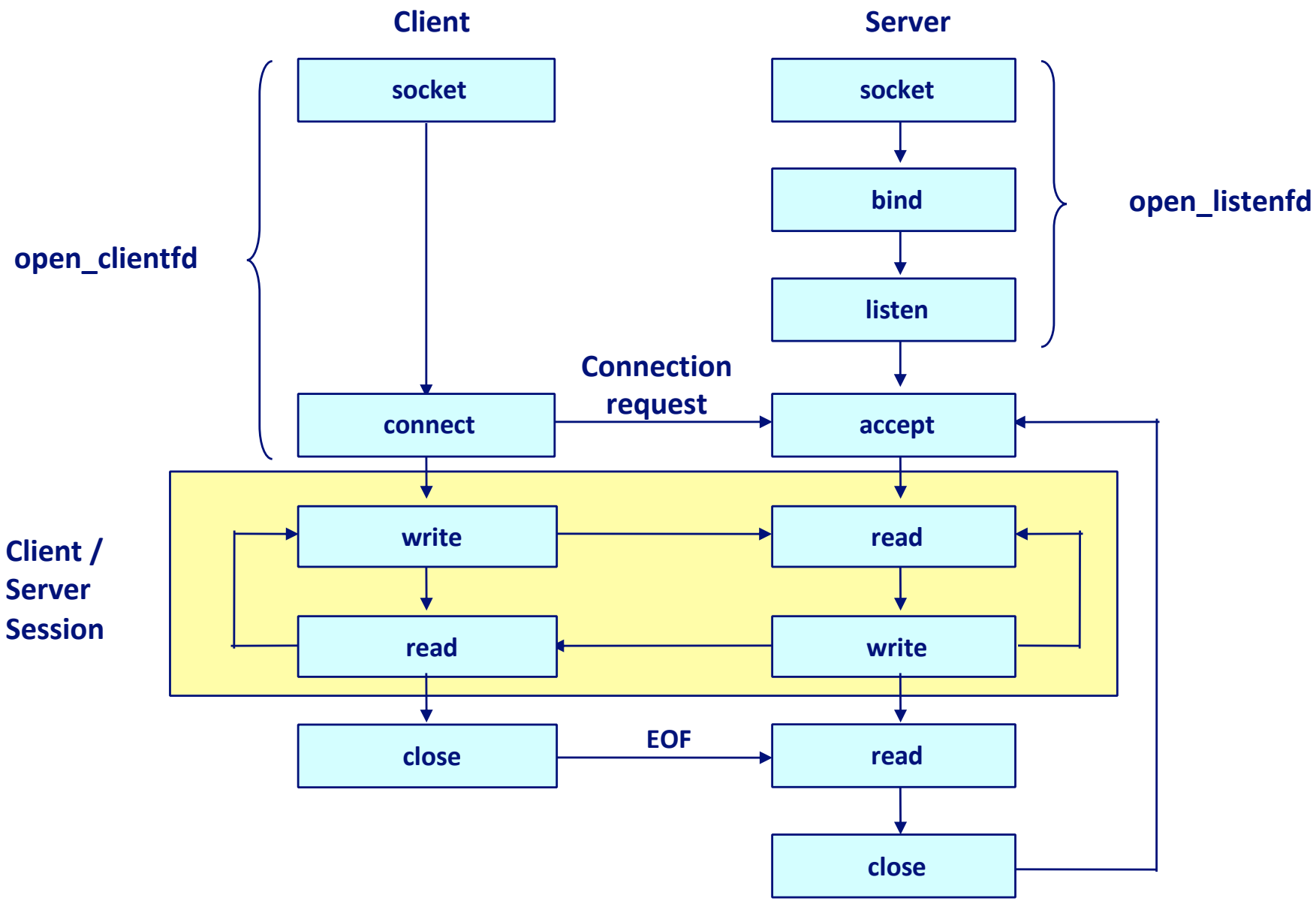
- Initiates contact with server ("speaks first")
- Typically requests service from server,
- For Web, client is implemented in browser; for e-mail, in mail reader

Server:

- Provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail



Socket API Operation Overview



What Service Does an Application 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

Transport Service Requirements of Common Apps

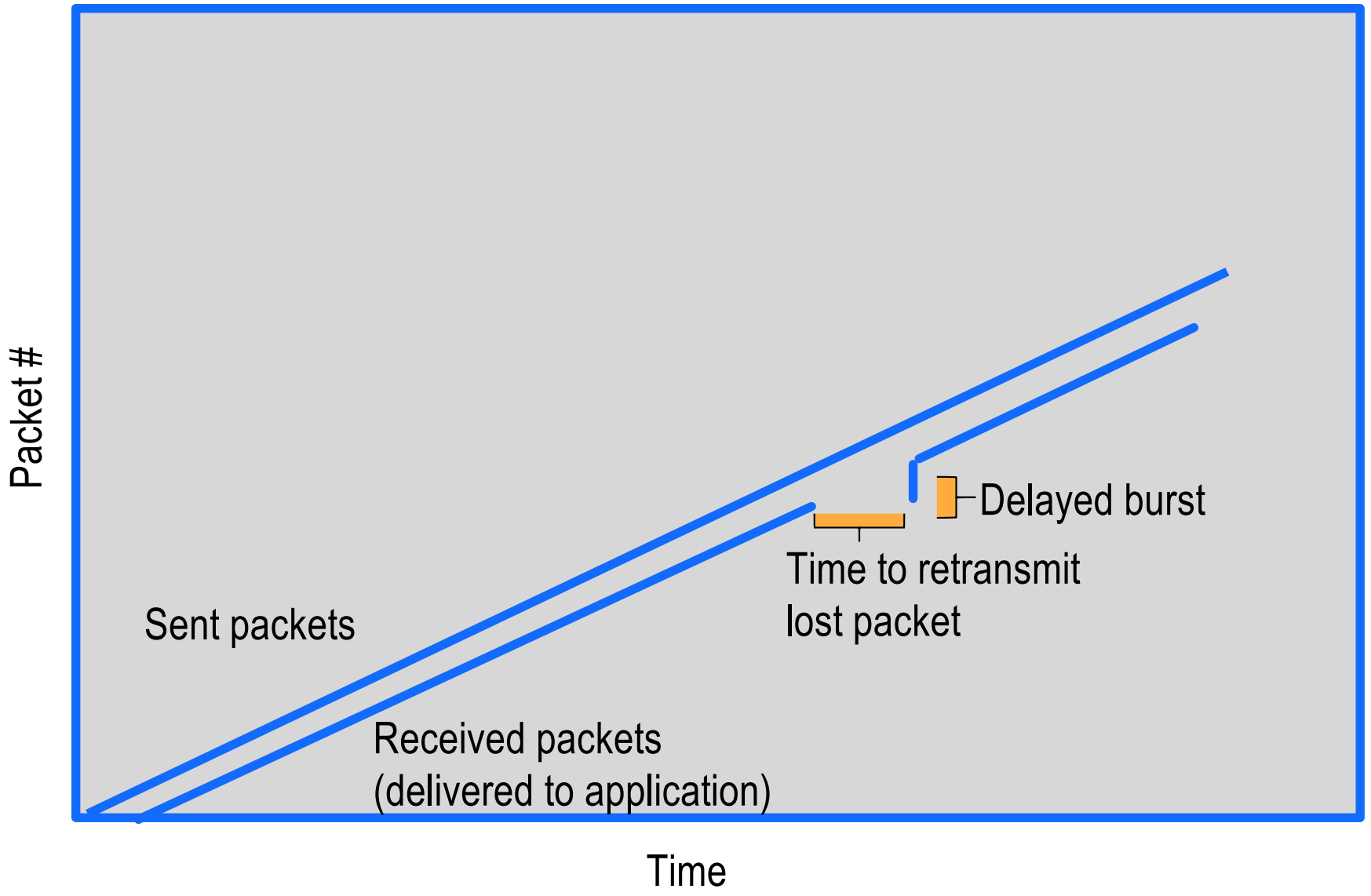
Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
web documents	no loss	elastic	No
interactive audio/video	loss-tolerant (often)	audio: 5Kb-1Mb video:10Kb-5Mb	yes, 100's msec
non-interactive audio/video	loss-tolerant (sometimes)	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps	yes, 100's msec
financial apps	no loss	elastic	yes and no: μs?

Why not always use TCP?

- TCP provides “more” than UDP
- Why not use it for everything??

- A: Nothing comes for free...
  - Connection setup (take on faith) -- TCP requires one round-trip time to setup the connection state before it can chat...
  - How long does it take, using TCP, to fix a lost packet?
    - At minimum, one “round-trip time” (2x the latency of the network)
    - That could be 100+ milliseconds!
  - If I guarantee in-order delivery,
    what happens if I lose one packet in a stream of packets?

One lost packet



Design trade-off

- If you're building an app...

- Do you need everything TCP provides?
- If not:
  - Can you deal with its drawbacks to take advantage of the subset of its features you need?

    OR
  - You're going to have to implement the ones you need on top of UDP
    - Caveat: There are some libraries, protocols, etc., that can help provide a middle ground.
    - Takes some looking around - they're not as standard as UDP and TCP.

Blocking sockets

- What happens if an application write()s to a socket waaaaay faster than the network can send the data?
  - TCP figures out how fast to send the data...
  - And it builds up in the kernel socket buffers at the sender... and builds...
  - until they fill. The next write() call *blocks* (by default).
  - What's blocking? It suspends execution of the blocked thread until enough space frees up...

In contrast to UDP

- UDP doesn't figure out how fast to send data, or make it reliable, etc.
- So if you write() like mad to a UDP socket...
- It often silently disappears. *Maybe* if you're lucky the write() call will return an error. But no promises.

Web Page Retrieval

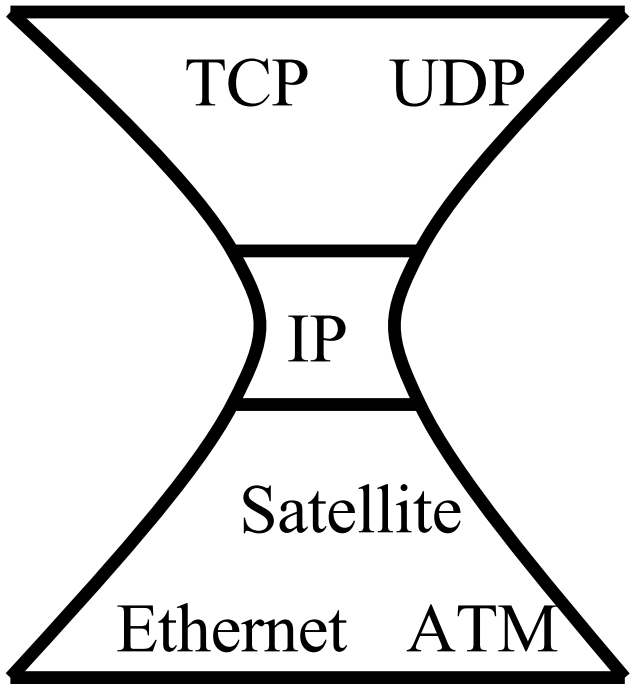
1. Static configuration
   - IP address, DNS server IP address, IP address of routers,
2. ARP for router
3. DNS lookup for web server
   - Several packet exchanges for lookup
4. TCP SYN exchange
5. HTTP Get request
6. HTTP response
   - Slow start, retransmissions, etc.

Caching Helps

1. Static configuration
   - IP address, DNS server IP address, IP address of routers,
2. ARP for router
3. DNS lookup for web server
   - Several packet exchanges for lookup
4. TCP SYN exchange
5. HTTP Get request
6. HTTP response
   - Slow start, retransmissions, etc.

Summary: Internet Architecture

- Packet-switched datagram network
- IP is the “compatibility layer”
  - Hourglass architecture
  - All hosts and routers run IP
- Stateless architecture
  - no per flow state inside network



Summary: Minimalist Approach

- Dumb network
  - IP provide minimal functionalities to support connectivity
    - Addressing, forwarding, routing
- Smart end system
  - Transport layer or application performs more sophisticated functionalities
    - Flow control, error control, congestion control
- Advantages
  - Accommodate heterogeneous technologies (Ethernet, modem, satellite, wireless)
  - Support diverse applications (telnet, ftp, Web, X windows)
  - Decentralized network administration

Example: project 1

- Project 1: Build a bitcoin miner
- Server --- many clients
- Communication:
  - Send job
  - ACK job
  - do some work
  - send result to server
  - (repeat)

- IP communication model:
  - Messages may be lost, re-ordered, corrupted (we'll ignore corruption, mostly, except for some sanity checking)
- Fail-stop node model:
  - You don't need to worry about evil participants faking you out.

Proj 1 and today's material

- You'll use UDP. Why?
  - A1: The course staff is full of sadists who want you to do a lot of work. This is true in part: timeouts and retransmission are a core aspect of using the network.
  - A2: The communication needed is very small, and you have to implement a lot of reliability stuff anyway to ensure that the work gets done...
  - Honestly? This one is a middle ground. You might use TCP for "other" reasons (firewalls that block everything but TCP), or to avoid the need for the "job ack" part of the protocol. Or you might stick with UDP to reduce the overhead at the server.

Networks: Common (Interview) Questions

What are Unicasting, Anycasting, Multi-casting and Broadcasting?

What are network “layers”? List some of them.

What is Stop-and-Wait Protocol?

Differences between Bridge/Switch and Router?

What is DHCP, how does it work?

What is ARP, how does it work?

...