# Distributed Systems

**15-440/640**

Fall 2018

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

# Announcements

- Recitations
  - Tomorrow (9/5) – Wean 7500 to go over the basics of Golang at 6pm and again at 7pm
  - Prepare by going over the Tour of Go

    https://tour.golang.org
  - In general: will be on Wednesday evenings
    - Not every week
    - Will be announced

# Thursday’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

# 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.
- Fail-slow or fail-stutter:
  - Performance may vary on failures as well
- Byzantine:
  - Anything that can go wrong, will.
  - Including malicious entities taking over your computers and making them do whatever they want.
- 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…

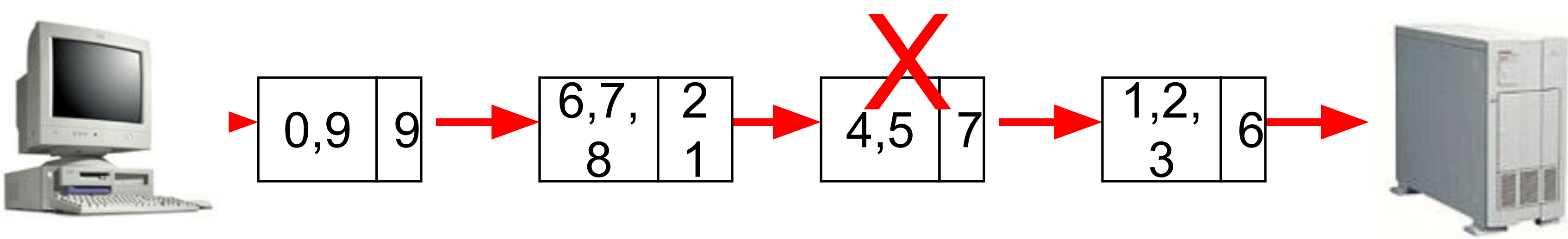
If network provides this ⇒ reuse across applications

How would you implement these?

# What if the Data gets Corrupted?

Problem: Data Corruption


Solution: Add a *checksum*



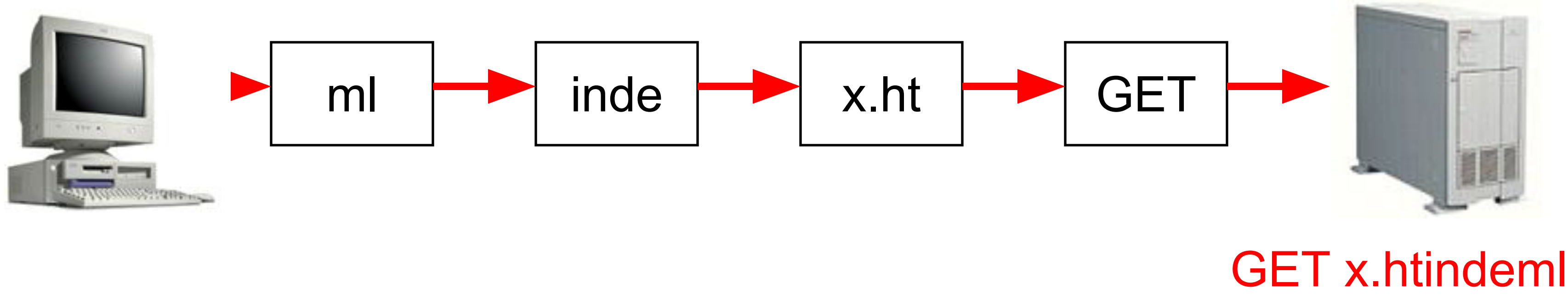
# What if the Data gets Lost?

Problem: Lost Data

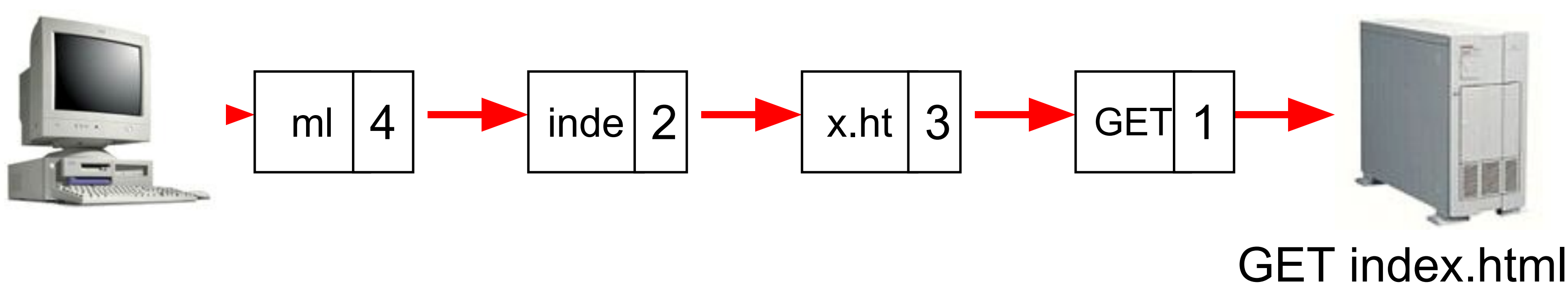

Solution: Timeout and Retransmit

# What if the Data is Out of Order?

Problem: Out of Order




Solution: Add Sequence Numbers



# Networks [including end points] Implement Many Functions

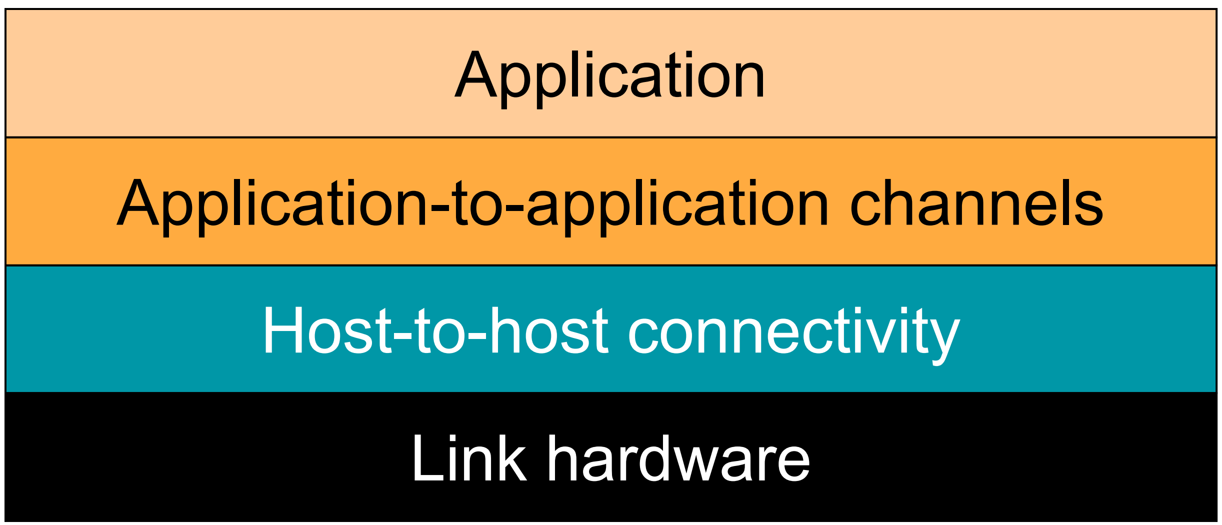
- Link
- Multiplexing
- Routing
- Addressing/naming (locating peers)
- Reliability
- Flow control
- Fragmentation
- Etc….

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

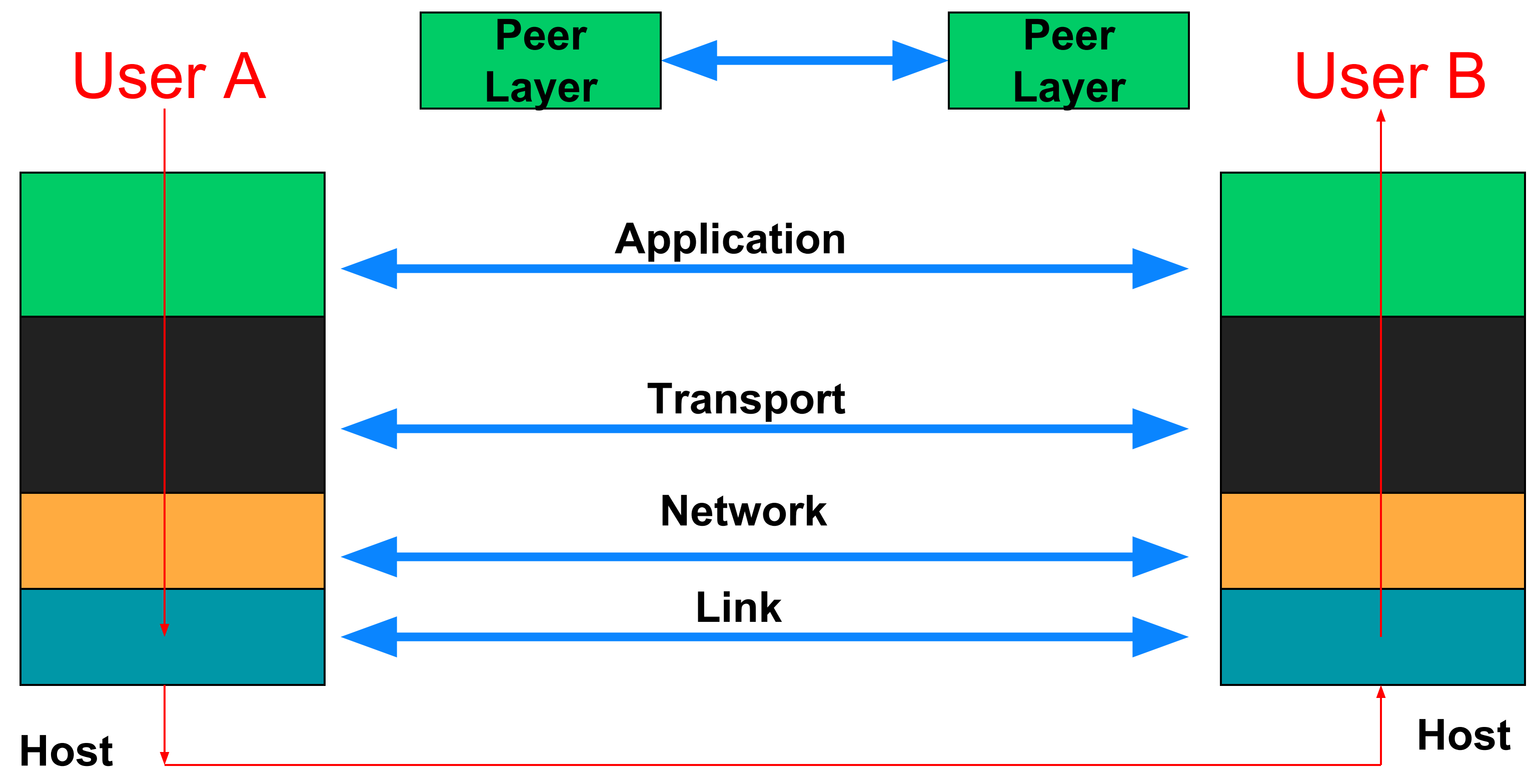
Where would you implement each function?

# 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, ordering of requests, etc.
- Hides implementation - layers can change without disturbing other layers (black box)

# Today's Lecture

Network links and LANs

Inter-network Communication

Layering & Protocols

Internet design

Transport protocols

Application design

# Goals [Clark88]

0 **Connect existing networks**

initially ARPANET and ARPA packet radio network

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

# Gateway Alternatives

- Translation
  - Difficulty in dealing with different features supported by networks
  - Scales poorly with number of network types (N^2 conversions)
- Standardization
  - “IP over everything” (**Design Principle 1**)
  - Minimal assumptions about network
  - Hourglass design

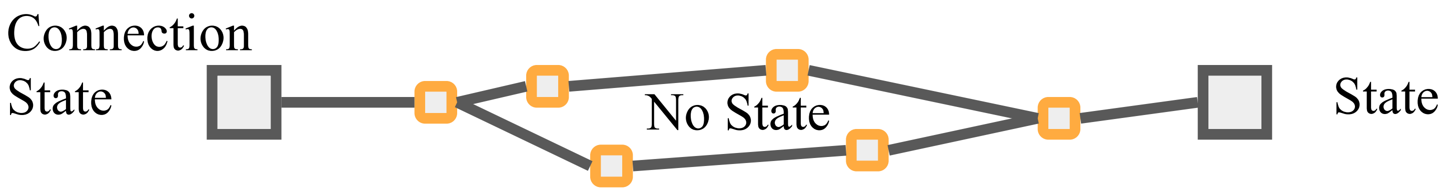
# Goal 1: Survivability

- 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
- Tradeoffs
  - Survivability: Heterogeneous network → less information available to end hosts and Internet level recovery mechanisms
  - Trust: must trust endpoints more

# End-to-End Argument

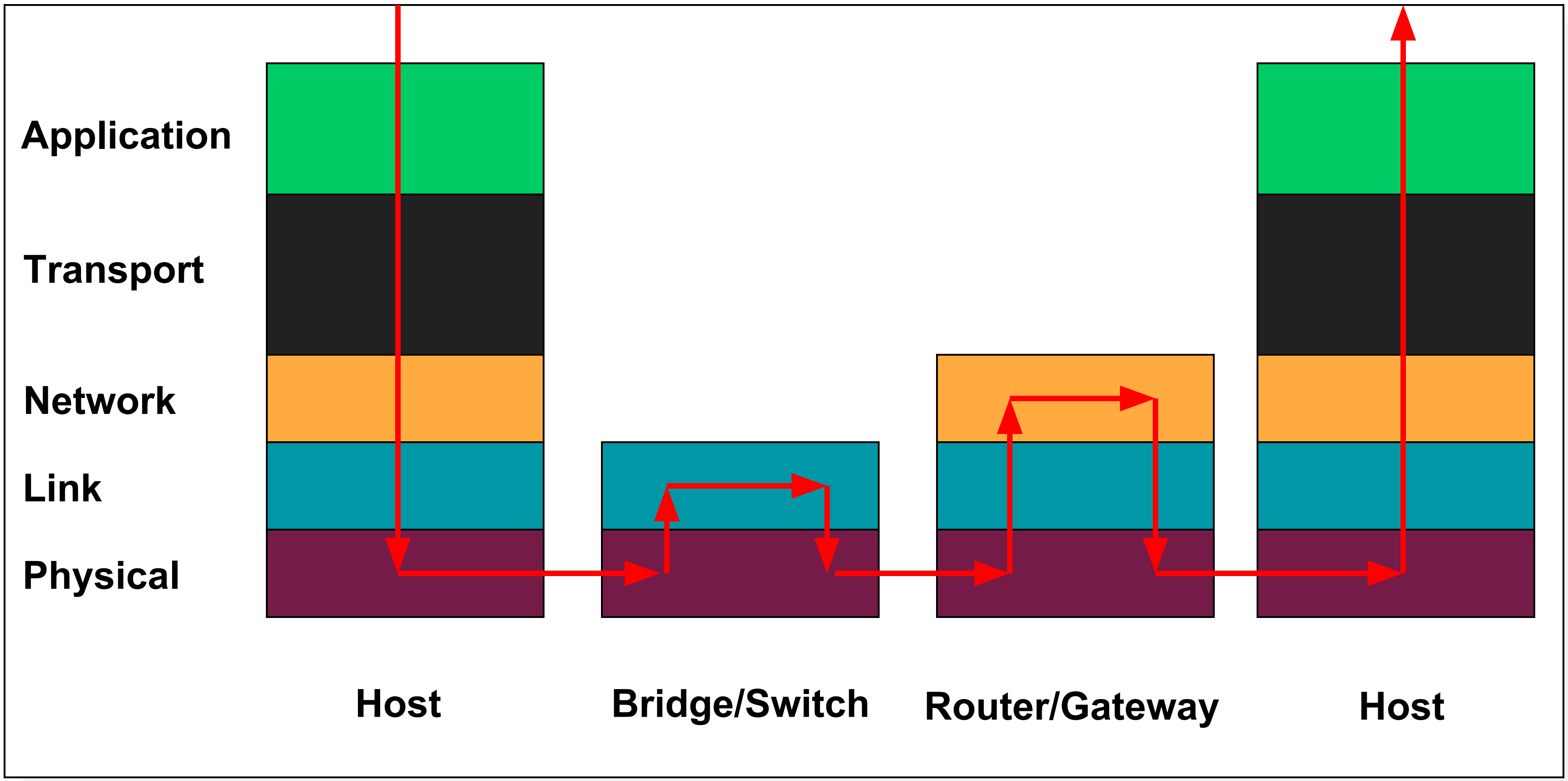
- Deals with where to place functionality
  - Inside the network (in switching elements)
  - At the edges
- Argument
  - If you have to implement a function end-to-end anyway (e.g., because it requires the knowledge and help of the end-point host or application), **don’t implement it inside the communication system**
  - Unless there’s a compelling performance enhancement

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

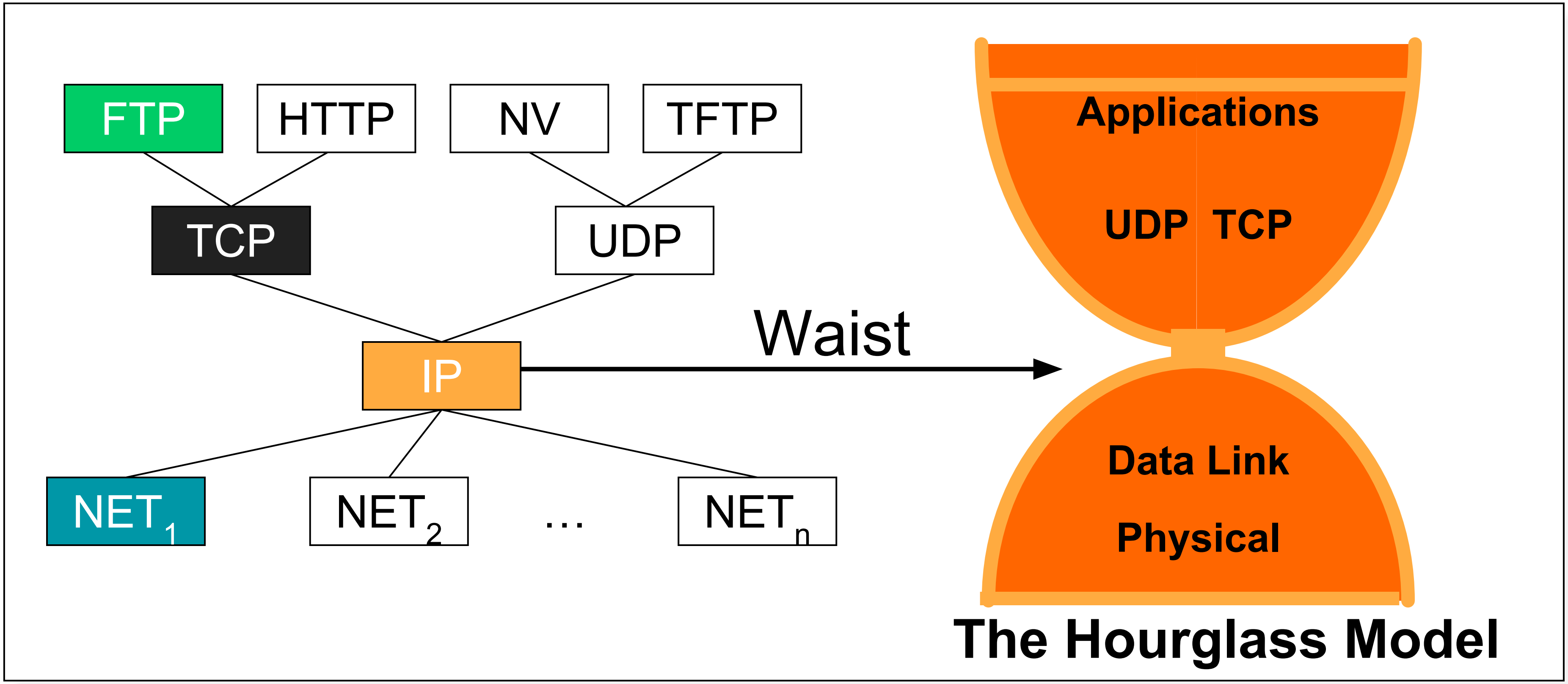
*Further Reading: “End-to-End Arguments in System Design.” Saltzer, Reed, and Clark.*

# IP Layering

- Relatively simple



# The Internet Protocol Suite

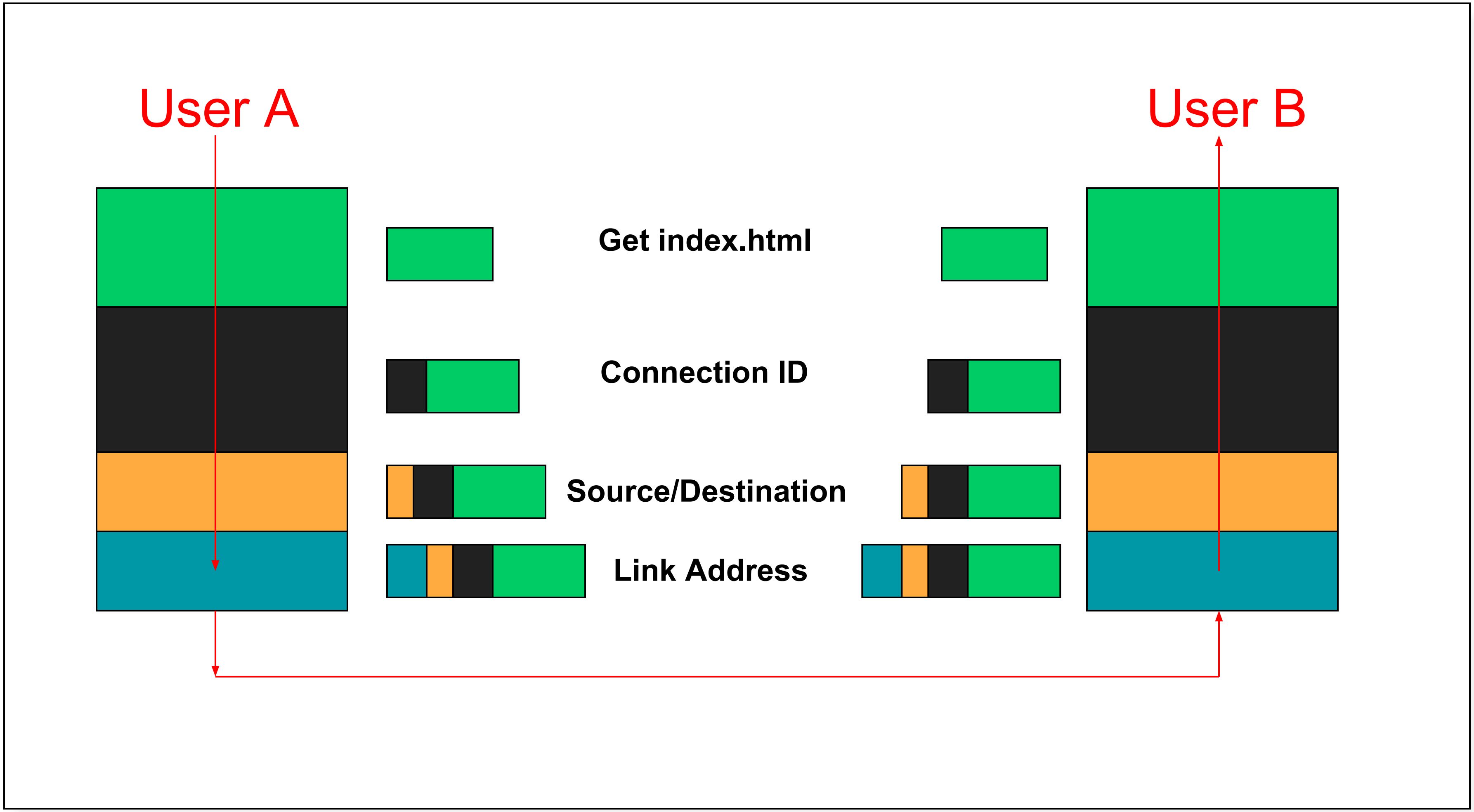



The waist facilitates interoperability

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

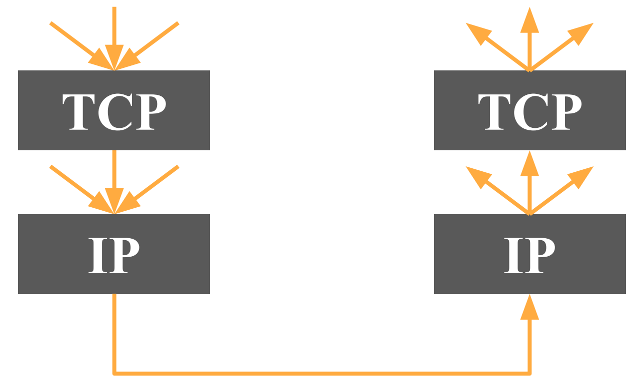
What creates the "edges" in the tree above?

# Layer Encapsulation

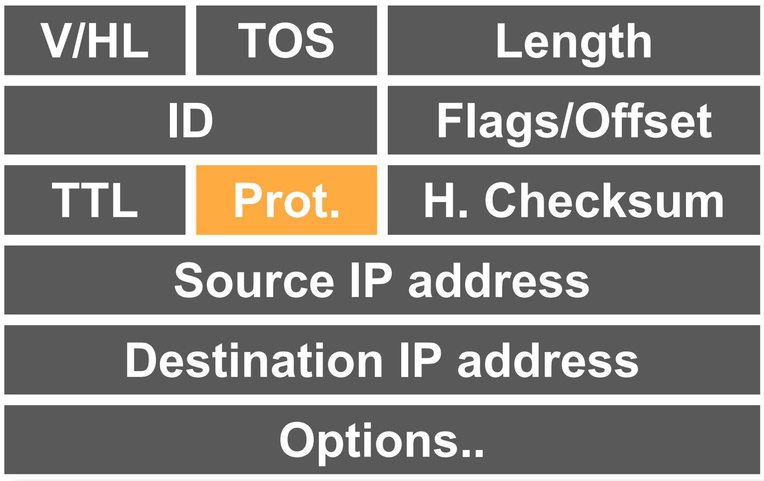


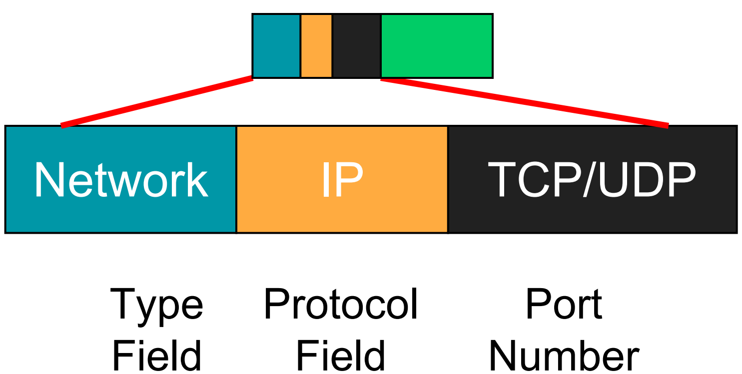
# Multiplexing and Demultiplexing

- 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




Recall: IP header



# Today’s Lecture

Network links and LANs

Inter-network Communication

Layering & Protocols

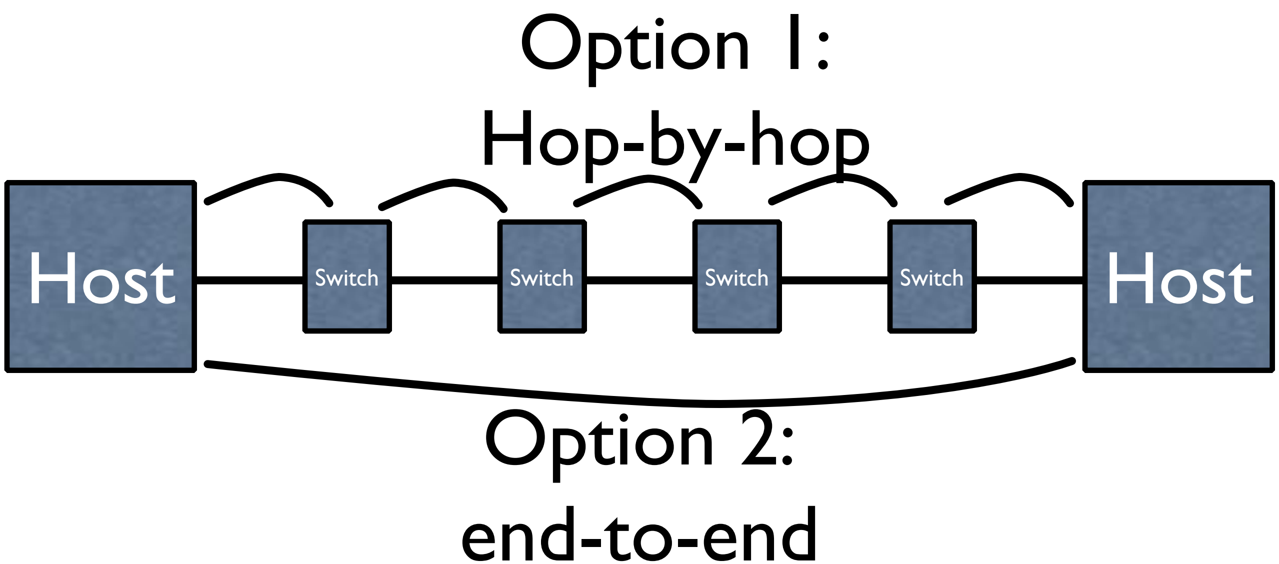
Internet design

Transport protocols

Application design

# Design Question

- If you want reliability, etc.
- Where should you implement it?



# Transport Protocols & Types of Service

- TCP vs. UDP
  - 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 integrity and 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<br>• No guarantee of delivery<br>• Not necessarily in-order delivery<br>• Datagram – independent packets<br>• Must address each packet | • Single mailbox to receive letters<br>• Unreliable ☺<br>• Not necessarily in-order delivery<br>• Letters sent independently<br>• Must address each letter |

Example UDP applications
Multimedia, voice over IP

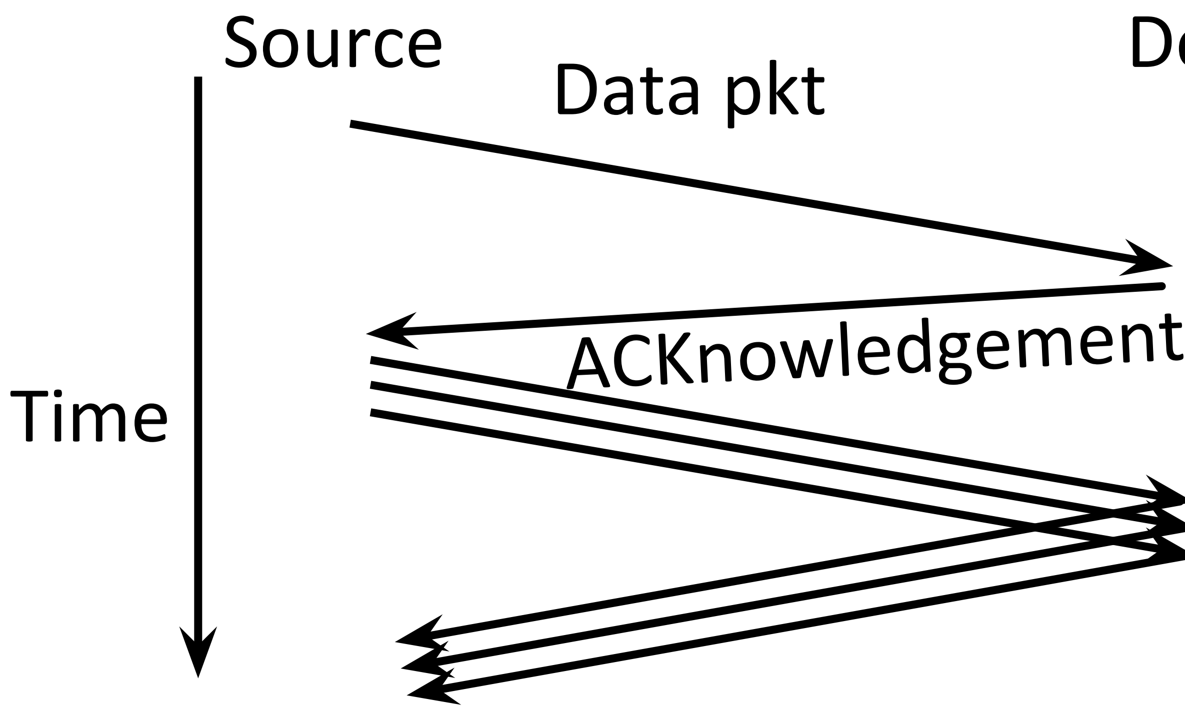
# Transmission Control Protocol (TCP): An Analogy

| TCP | Telephone Call |
| --- | --- |
| • Reliable – guarantee delivery<br>• Byte stream – in-order delivery<br>• Connection-oriented – single socket per connection<br>• Setup connection followed by data transfer | • Guaranteed delivery<br>• In-order delivery<br>• Connection-oriented<br>• 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 (e.g. due to e2e argument)... 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

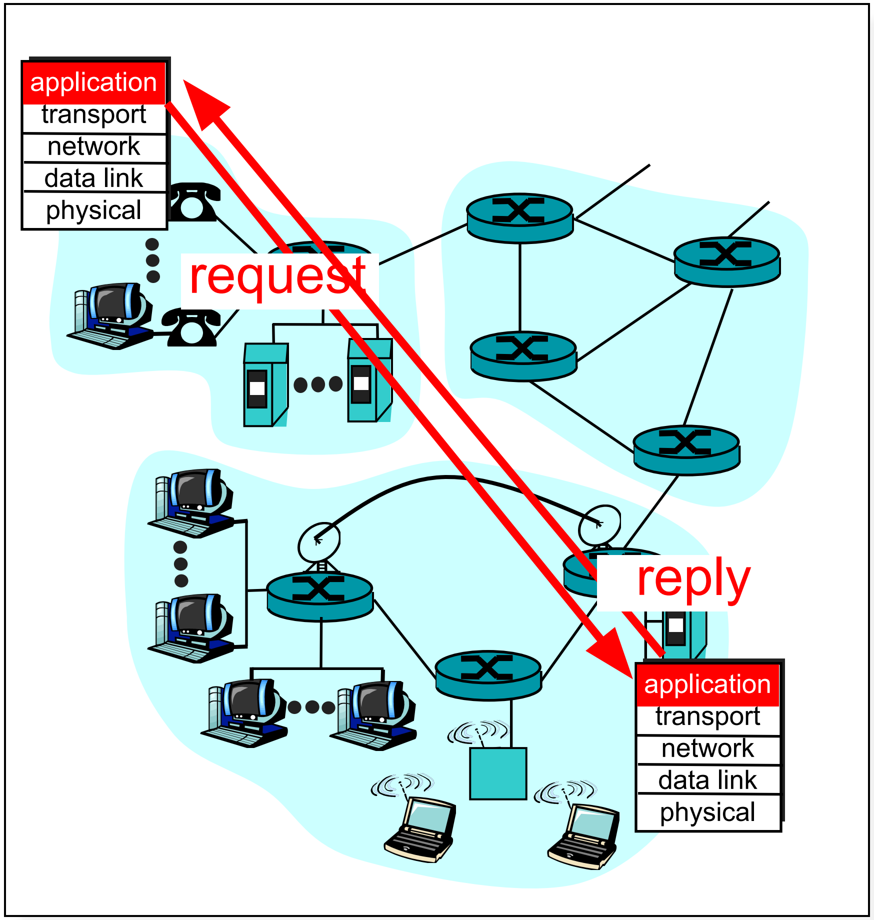
Typical network app has two pieces: *client* and *server*

Client:

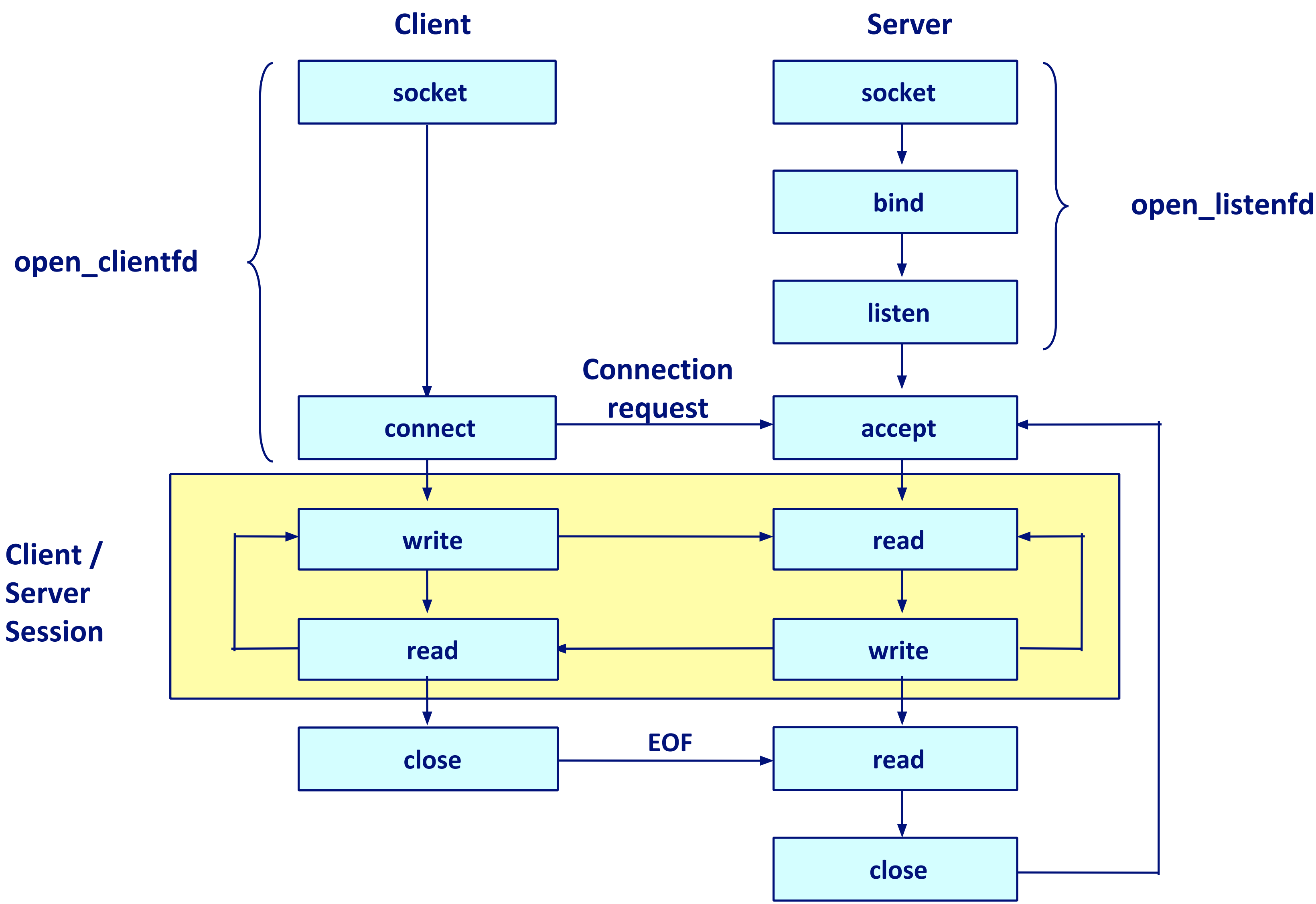
- Initiates contact with server ("speaks first")
- Typically requests service from server,
- For Web, client is implemented in browser; for e-mail, in mail reader

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

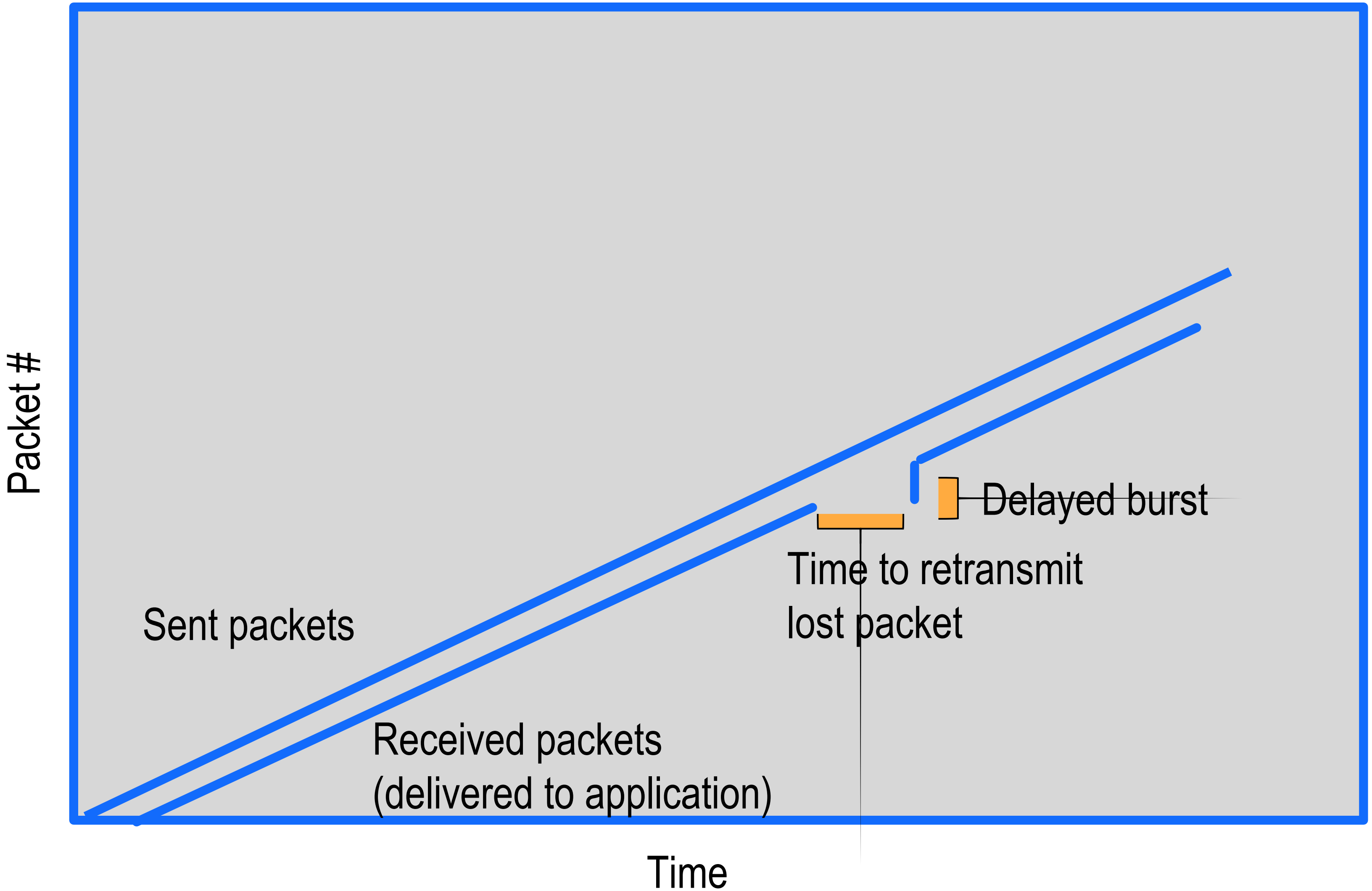
| Application | Data loss | Bandwidth | Time Sensitive |
|---|---|---|---|
| file transfer | no loss | elastic | no |
| e-mail | no loss | elastic | no |
| web documents | no loss | elastic | no |
| interactive audio/video | loss-tolerant (often) | audio: 5Kb-1Mb video:10Kb-5Mb | yes, 100's msec |
| 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

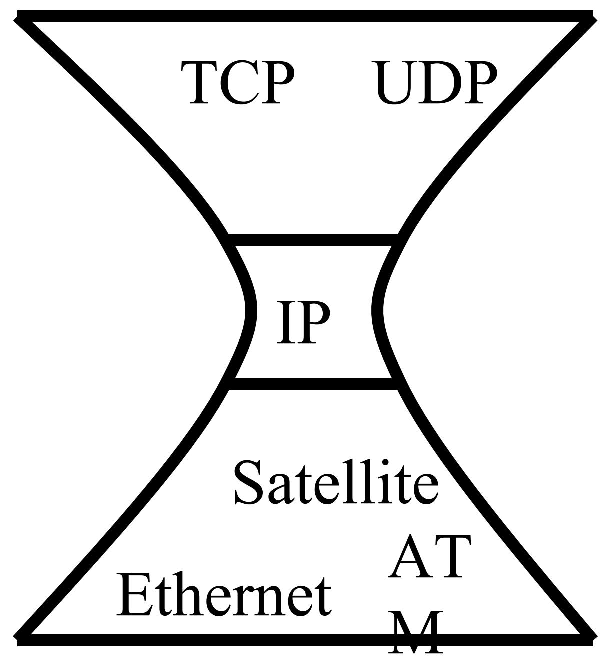
1. Static configuration
   - IP address, DNS server IP address, IP address of routers,
2. ARP for router
3. DNS lookup for web server
   - Several packet exchanges for lookup
4. TCP SYN exchange
5. HTTP Get request
6. HTTP response
   - Slow start, retransmissions, etc.

# 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

# Rehashing all of that...

- TCP is layered on top of IP
  - IP understands only the IP header
  - The IP header has a “protocol” ID that gets set to TCP
  - The TCP at the receiver understands how to parse the TCP information
- IP provides only “best-effort” service
- TCP adds value to IP by adding retransmission, in-order delivery, data checksums, etc., so that programmers don’t have to re-implement the wheel every time. It also helps figure out how fast to send data. This is why TCP sockets can “block” from the app perspective.
- The e2e argument suggests that functionality that must be implemented end-to-end anyway (like retransmission in the case of dead routers) should probably be implemented only there -- unless there’s a compelling perf. optimization