416 Distributed Systems

Networks review

Sep 11, 2018

Distributed Systems vs. Networks

- Low level (c/go)
- Run forever
- Support others
- Adversarial environment
- Distributed & concurrent
- Resources matter

- And have it implemented/run by vast numbers of different people with different goals/skills

Keep an eye out for…

- Modularity, Layering, and Decomposition:
  - Techniques for dividing the work of building systems
  - Hiding the complexity of components from each other
  - Hiding implementation details to deal with heterogeneity
- Naming/lookup/routing
- Resource sharing and isolation
- Models and assumptions about the environment and components

Today's Lecture (317 review-ish)

- Network links and LANs
- Layering and protocols
- Internet design

Basic Building Block: Links

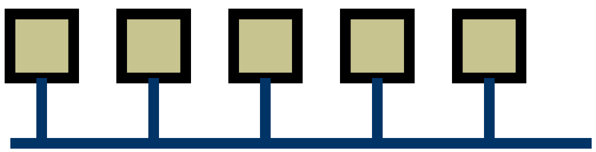
- Electrical questions
  - Voltage, frequency, …
  - Wired or wireless?
- Link-layer issues: How to send data?
  - When to talk – can either side talk at once?
  - What/how to say – low-level format?

Model of a communication channel

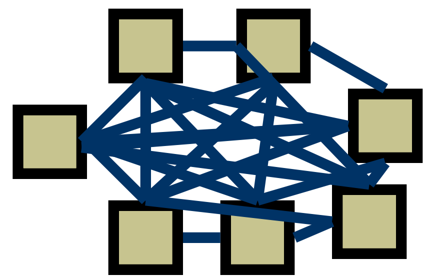
- Latency - how long does it take for the first bit to reach destination
- Jitter - how much variation in latency?
- Capacity - how many bits/sec can we push through? (often termed “bandwidth”)
- Loss / Reliability - can the channel drop packets?
- Reordering

Basic Building Block: Links

- … But what if we want more hosts?



One wire

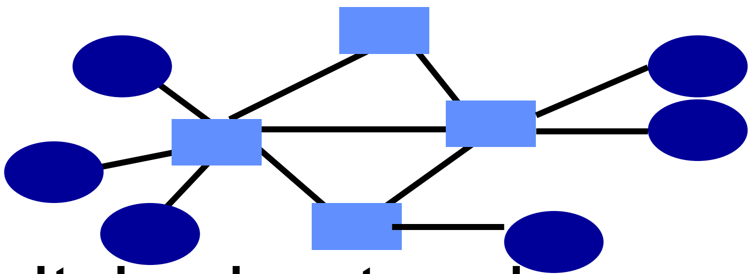


Wires for everybody!

- Scalability?!

Multiplexing

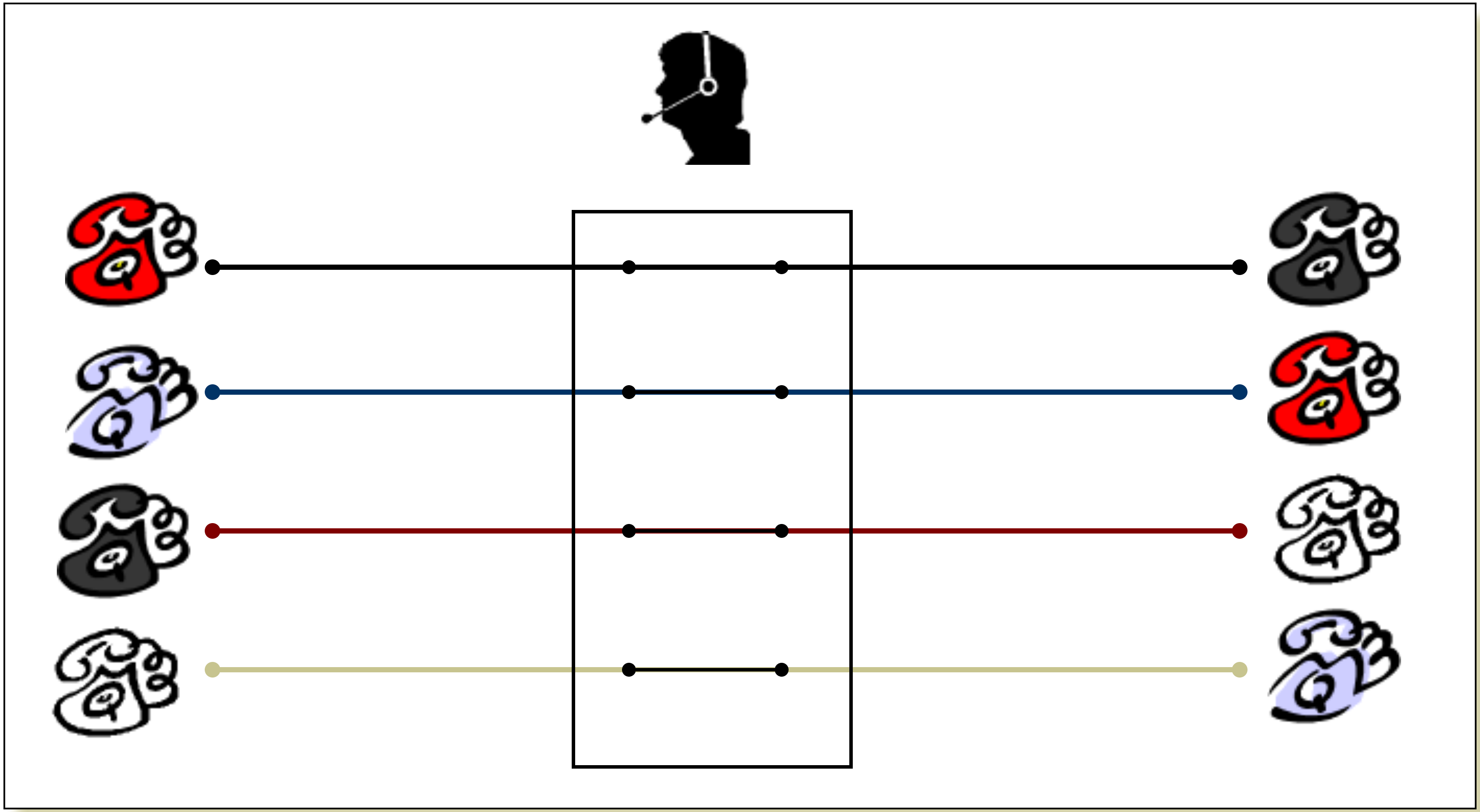
- Need to share network resources



- How? Switched network
  - Party "A" gets resources sometimes
  - Party "B" gets them sometimes
- Interior nodes act as "Switches"

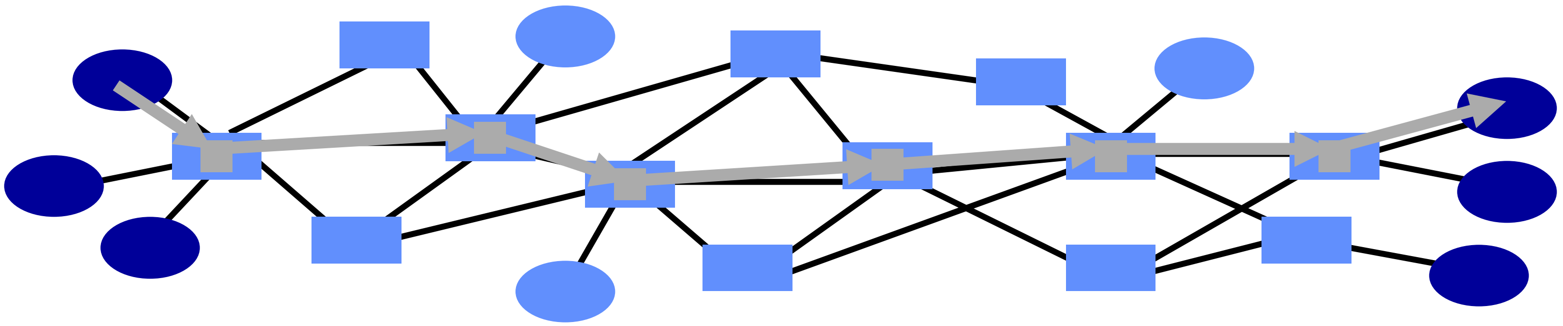
- What mechanisms to share resources?

In the Old Days…Circuit Switching

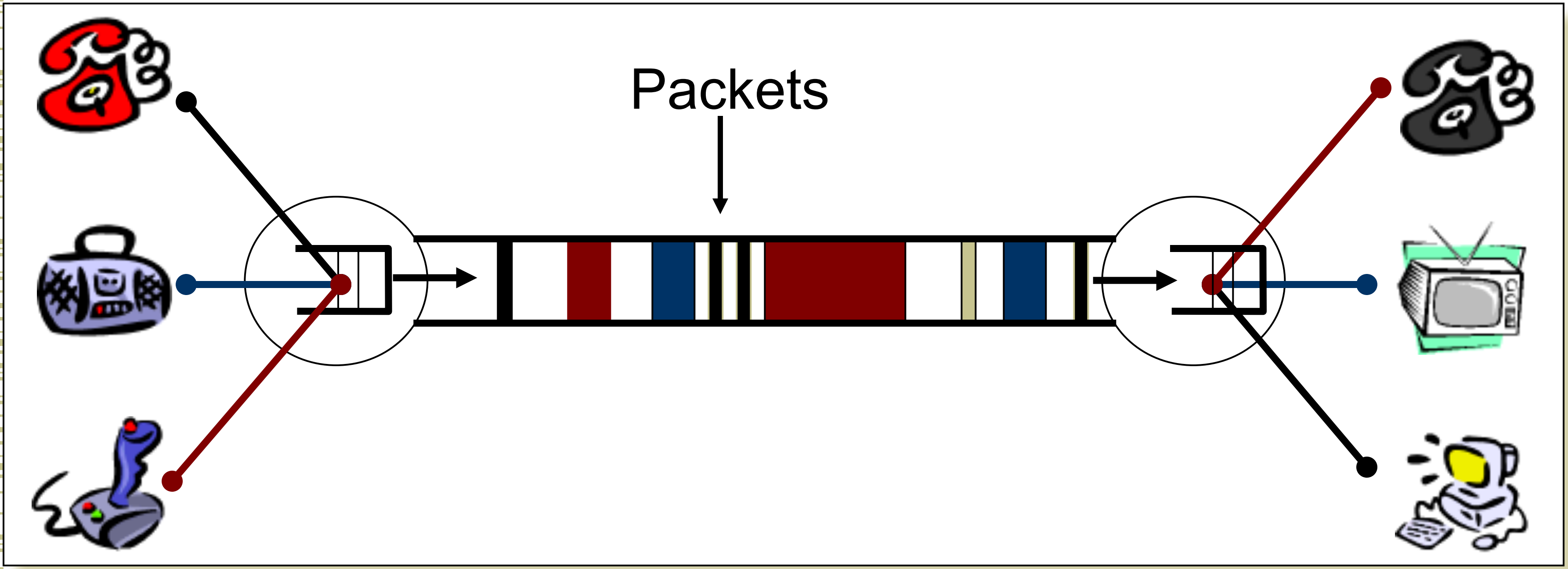


Packet Switching

- Source sends information as self-contained packets that have an address.
  - Source may have to break up single message in multiple
- Each packet travels independently to the destination host.
  - Switches use the *address* in the packet to determine how to forward the packets
  - Store and forward
- Analogy: a letter in surface mail (snail mail)



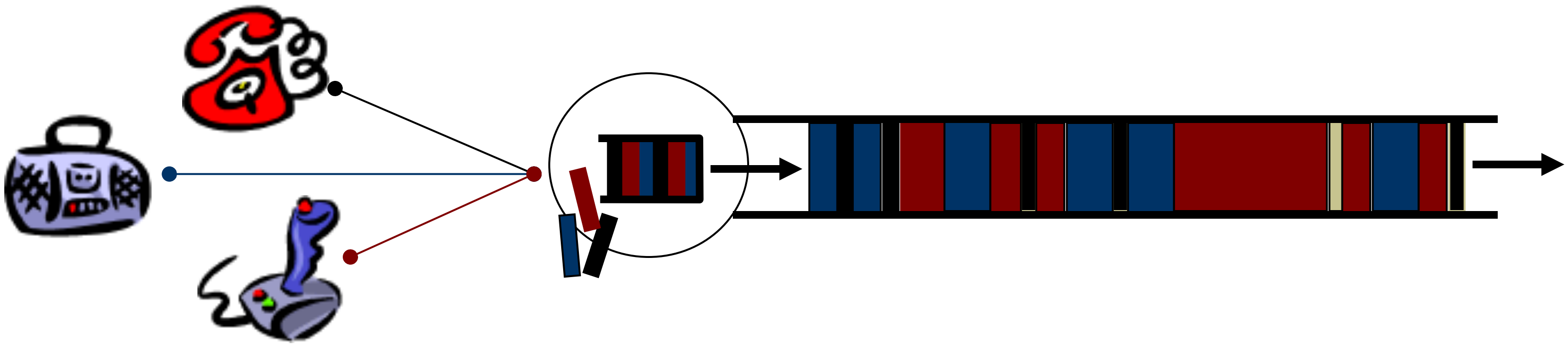
Packet Switching – Statistical Multiplexing



- Switches arbitrate between inputs
- Can send from *any* input that's ready
    - Links never idle when traffic to send
    - (Efficiency!)

What if Network is Overloaded?

Problem: Network Overload

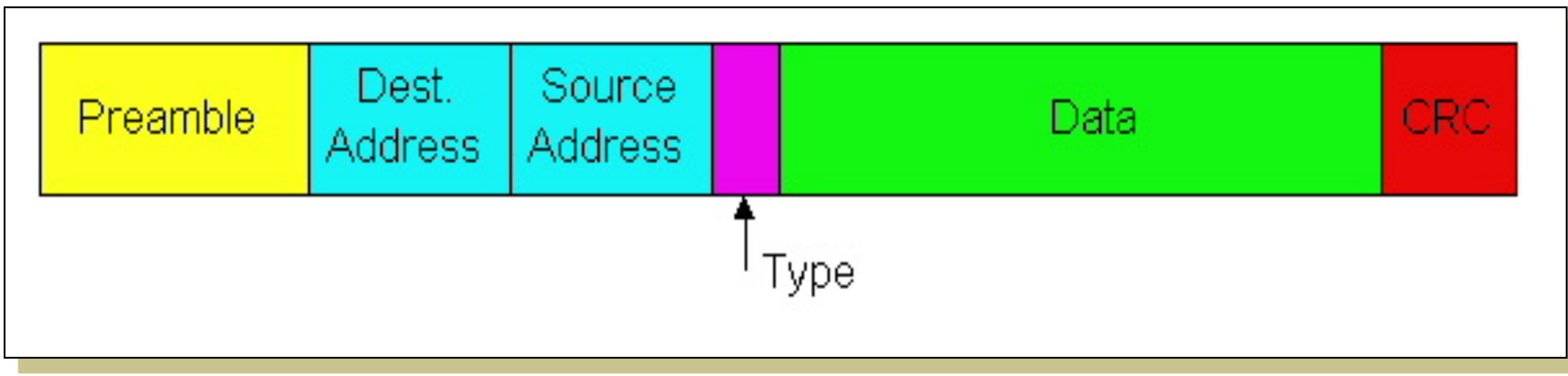


Solution: Buffering and Congestion Control

- Short bursts: buffer
- What if buffer overflows?
  - Packets dropped
  - Sender adjusts rate until load = resources → "congestion control"

Example: Ethernet Packet

- Sending adapter **encapsulates** IP datagram (or other network layer protocol packet) in Ethernet frame



Ethernet Frame Structure

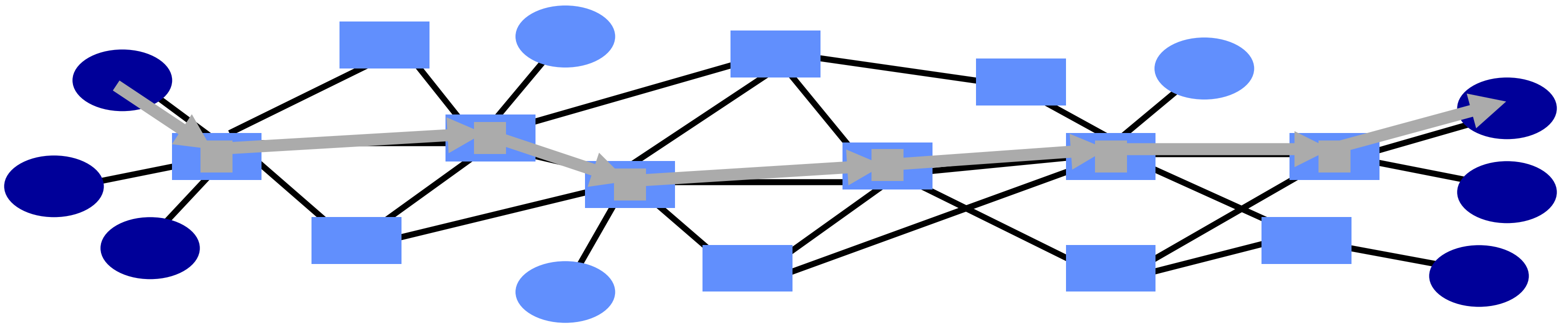
- Each protocol layer needs to provide some hooks to upper layer protocols
  - Demultiplexing: identify which upper layer protocol packet belongs to
  - E.g., port numbers allow TCP/UDP to identify target application
  - Ethernet uses Type field
- Type: 2 bytes
  - Indicates the higher layer protocol, mostly IP but others may be supported such as Novell IPX and AppleTalk

Ethernet Frame Structure (cont.)

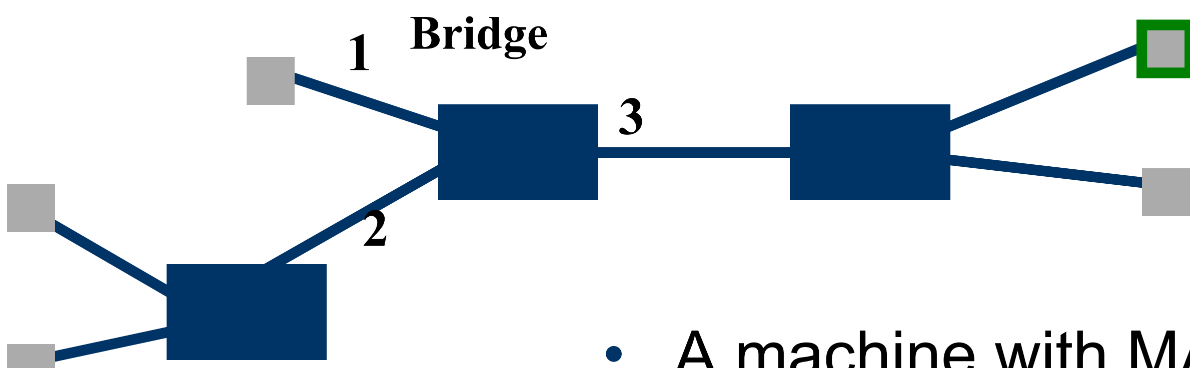
- Addresses: 6 bytes
  - Each adapter is given a globally unique address at manufacturing time
    - Address space is allocated to manufacturers
      - 24 bits identify manufacturer
      - E.g., 0:0:15:* → 3com adapter
    - Frame is received by all adapters on a LAN and dropped if address does not match
  - Special addresses
    - Broadcast – FF:FF:FF:FF:FF:FF is “everybody”
    - Range of addresses allocated to multicast
      - Adapter maintains list of multicast groups node is interested in

Packet Switching

- Source sends information as self-contained packets that have an address.
  - Source may have to break up single message in multiple
- Each packet travels independently to the destination host.
  - Switches use the address in the packet to determine how to forward the packets
  - Store and forward
- Analogy: a letter in surface mail.



Frame Forwarding

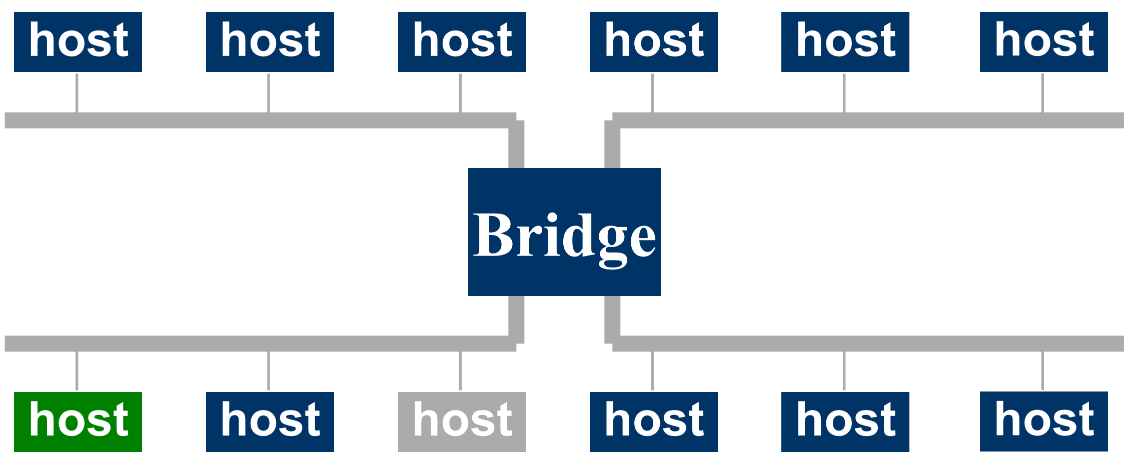


MAC Address	Port	Age
A21032C9A591	1	36
99A323C90842	2	01
8711C98900AA	2	15
301B2369011C	2	16
695519001190	3	11

- A machine with MAC Address lies in the direction of number port of the bridge
- For every packet, the bridge "looks up" the entry for the packet's destination MAC address and forwards the packet on that port.
  - Other packets are broadcast – why?
- Timer is used to flush old entries

Learning Bridges

- Manually filling in bridge tables?
  - Time consuming, error-prone
- Keep track of source address of packets arriving on every link, showing what segment hosts are on
  - Fill in the forwarding table based on this information

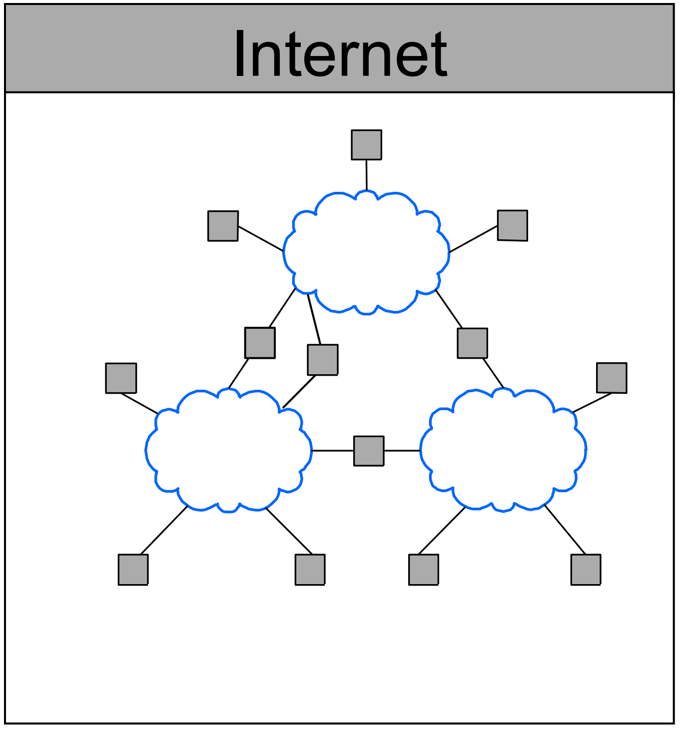


Today's Lecture

- Network links and LANs
- Layering and protocols
- Internet design

Internet

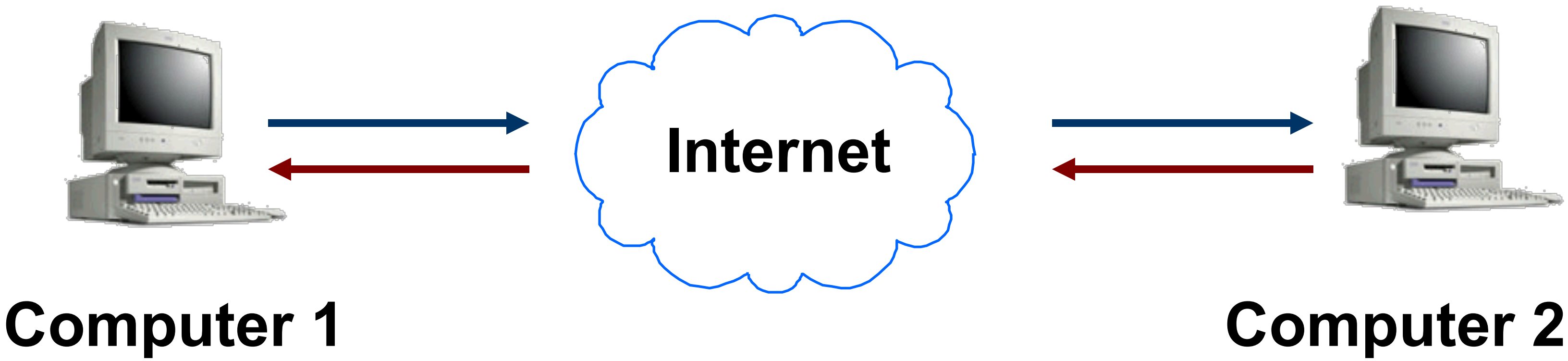
- An inter-net: a network of networks.
  - Networks are connected using routers that support communication in a hierarchical fashion
  - Often need other special devices at the boundaries for security, accounting, ..
- The Internet: the interconnected set of networks of the Internet Service Providers (ISPs)
  - About 17,000 different networks make up the Internet



Challenges of an internet

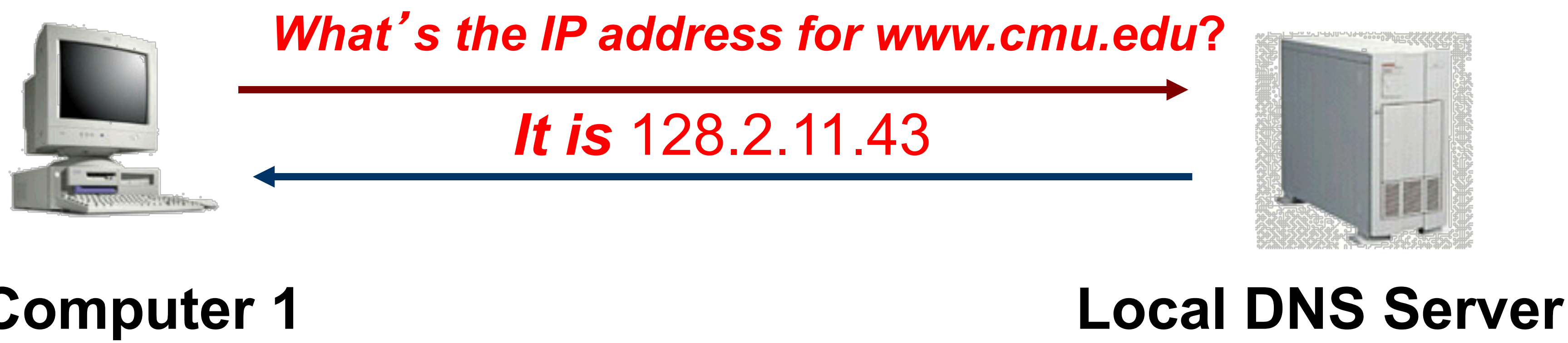
- Heterogeneity
  - Address formats
  - Performance – bandwidth/latency
  - Packet size
  - Loss rate/pattern/handling
  - Routing
  - Diverse network technologies → satellite links, cellular links, carrier pigeons
  - In-order delivery

How To Find Nodes?



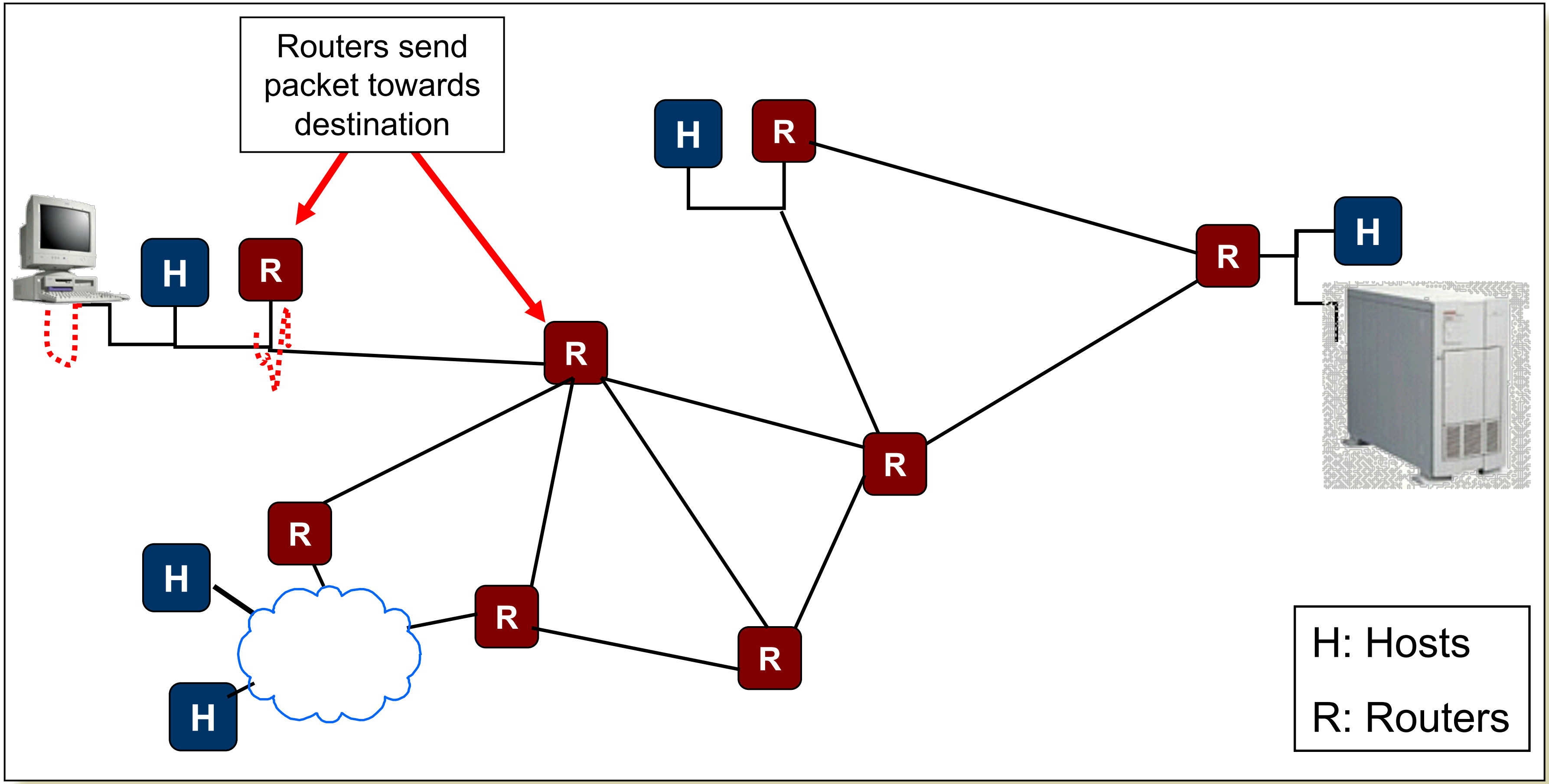
Need naming and routing

Naming



Translates human readable names to logical endpoints

Routing


Routers send packet towards destination
H
R
H
R
R
H
R
R
R
H
R
R
H
H: Hosts
R: Routers

Network Service Model

- What is the base (IP) *service model?*
  - Ethernet/Internet: *best-effort* – packets can get lost, etc.
- What if you want more?
  - Performance guarantees (QoS)
  - Reliability
    - Corruption
    - Lost packets
  - Flow and congestion control
  - Fragmentation
  - In-order delivery
  - Etc…

Aside: 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

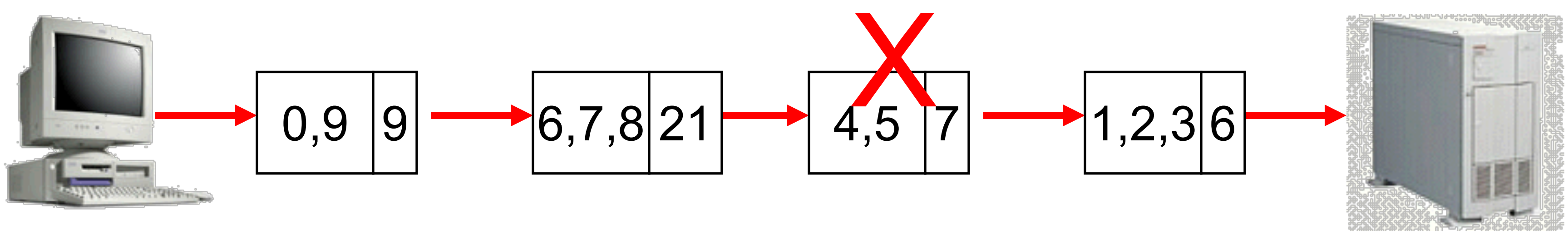
- What if network had reliable, in-order, mostly no-corruption, stream-oriented communication (i.e. TCP)

- Programmers don't have to implement these features in every application
- But note limitations: this can't turn a byzantine failure model into a fail-stop model...

What if the Data gets Corrupted?

Problem: Data Corruption

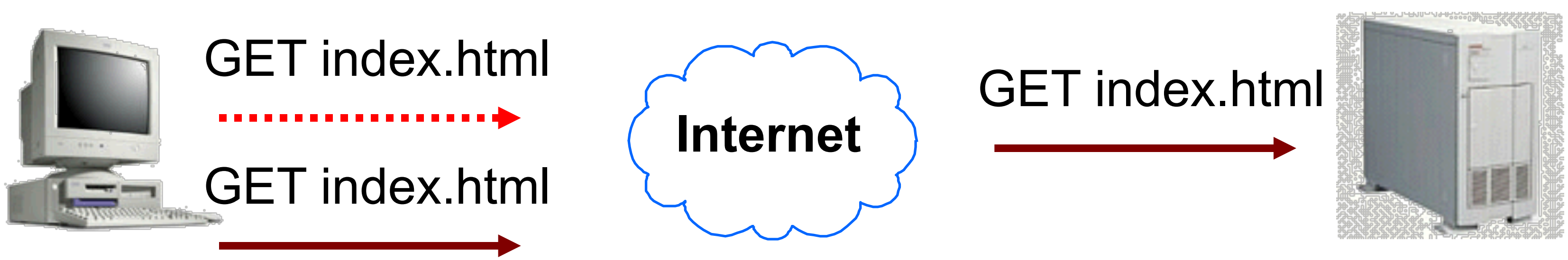
Solution: Add a *checksum*



What if the Data gets Lost?

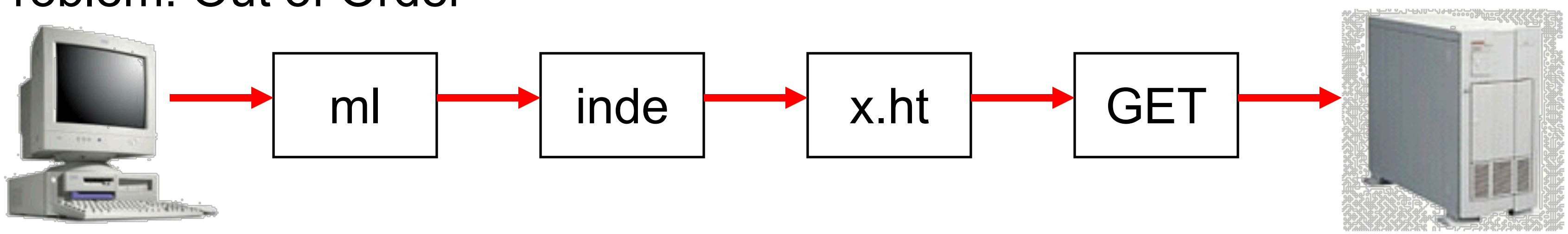
Problem: Lost Data

Solution: Timeout and Retransmit



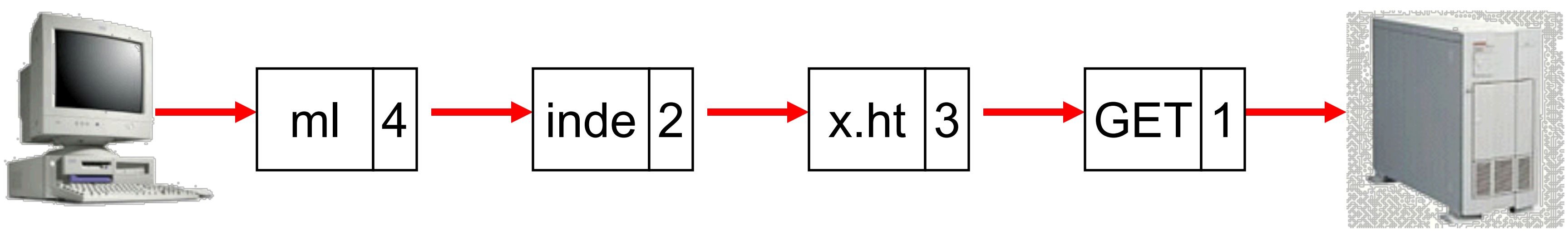
What if the Data is Out of Order?

Problem: Out of Order



GET x.htindeml

Solution: Add Sequence Numbers



GET index.html

Networks [including end points] Implement Many Functions

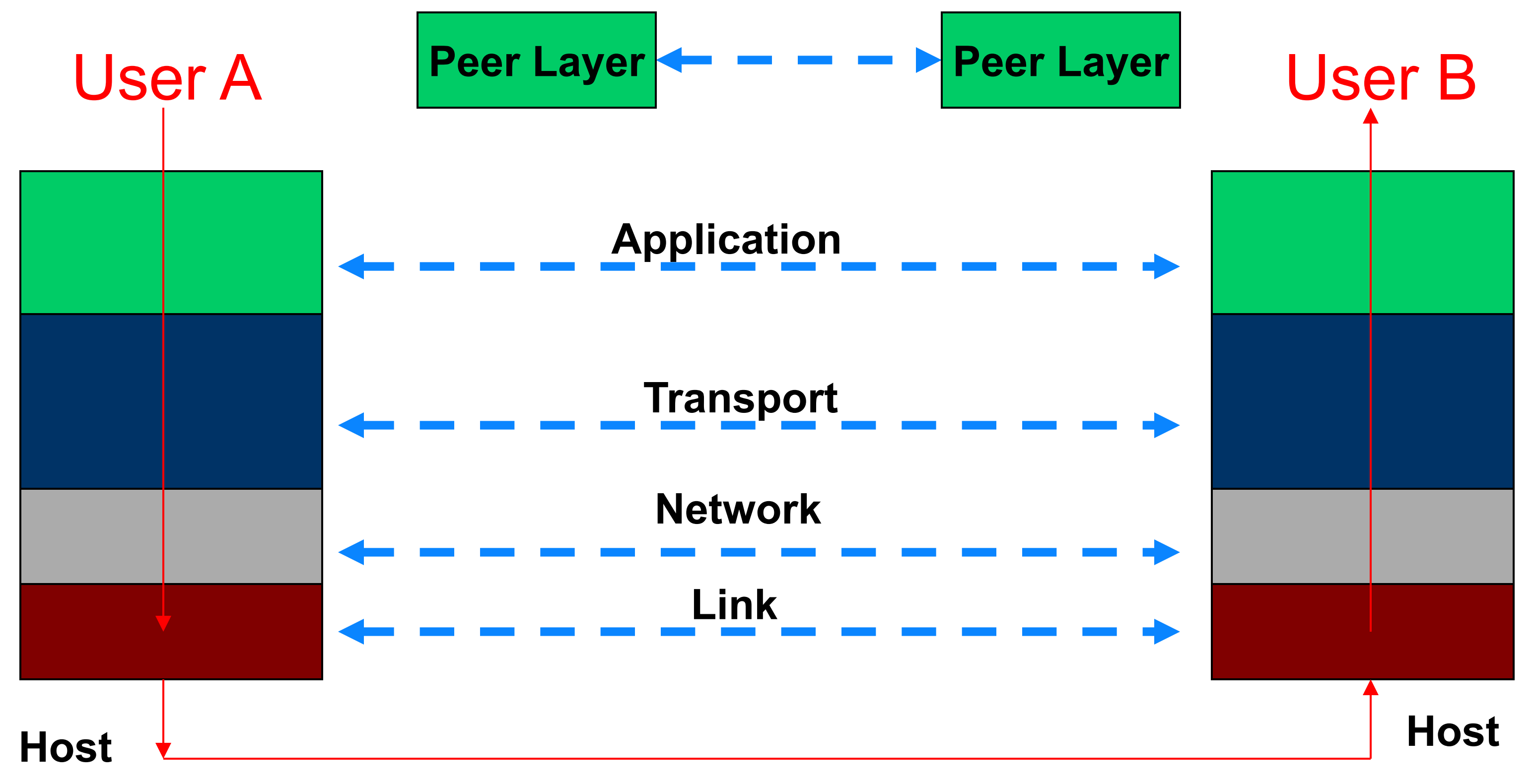
- Link
- Multiplexing
- Routing
- Addressing/naming (locating peers)
- Reliability
- Flow control
- Fragmentation
- Etc….

What is Layering?

- Modular approach to network functionality
- Example:

Application
Application-to-application channels
Host-to-host connectivity
Link hardware

What is Layering?



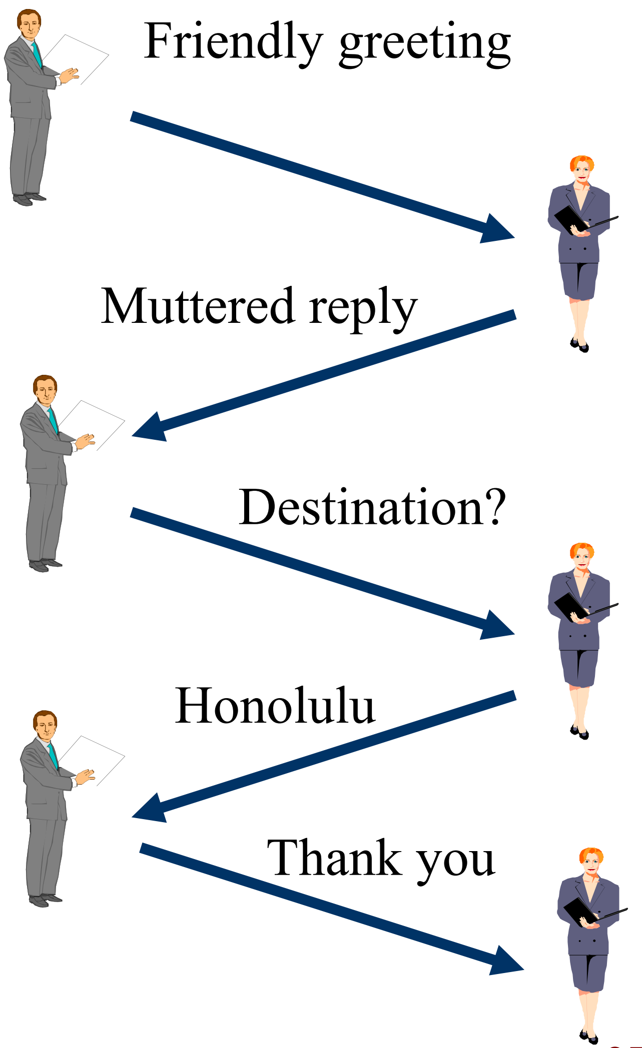
Modular approach to network functionality

Layering Characteristics

- Each layer relies on services from layer below and exports services to layer above
- Interface defines interaction with peer on other hosts
- Hides implementation - layers can change without disturbing other layers (black box)

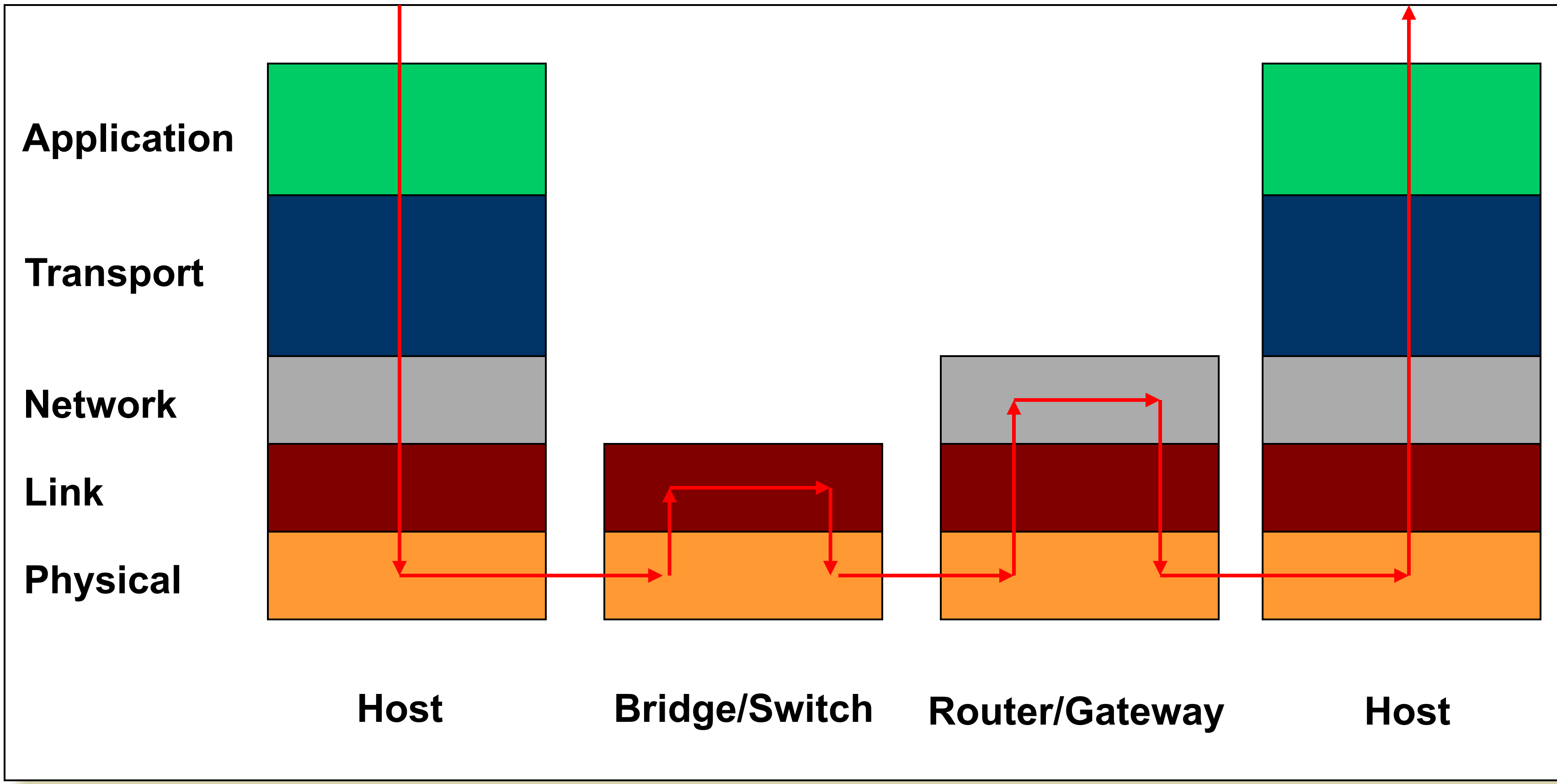
What are Protocols?

- An agreement between parties on how communication should take place
- Module in layered structure
- 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.
- Example: Buying airline ticket

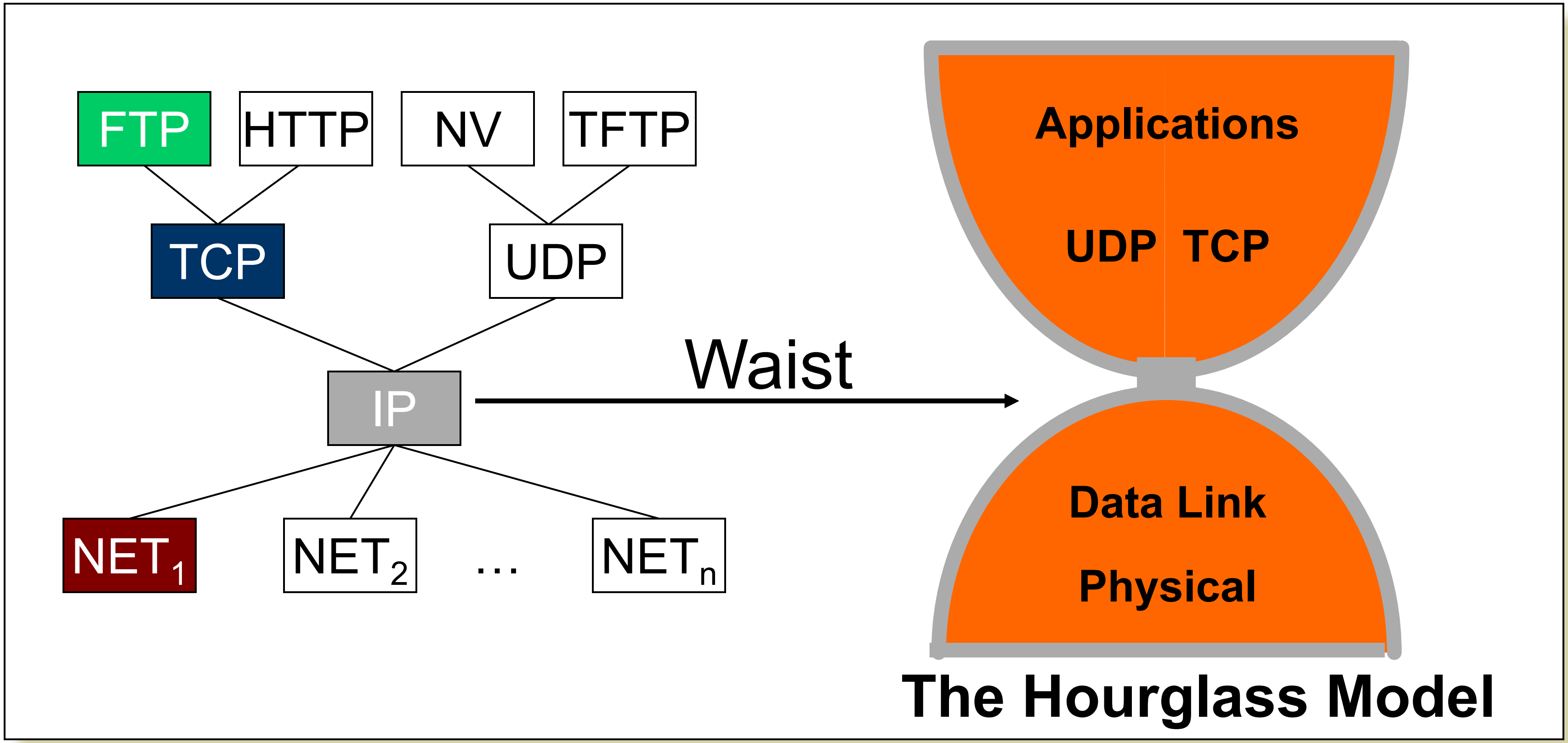


IP Layering (control flow)

- Relatively simple

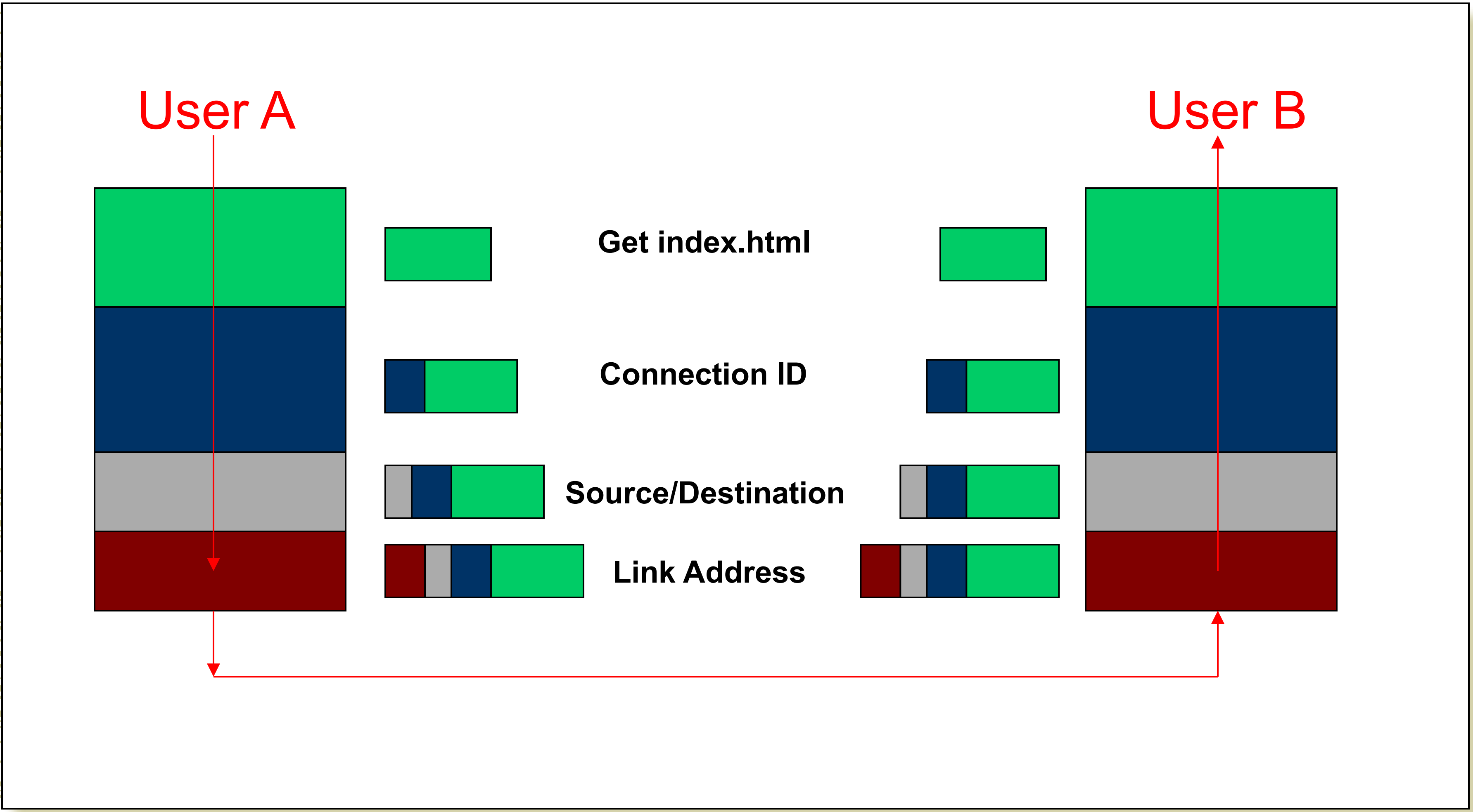


The Internet Protocol Suite



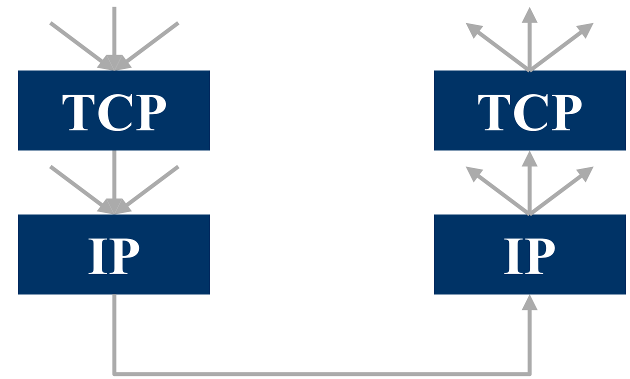
The “thin” waist facilitates interoperability

Layer Encapsulation (data flow)



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
- Multiplexing occurs at multiple layers. E.g., IP, TCP, …



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

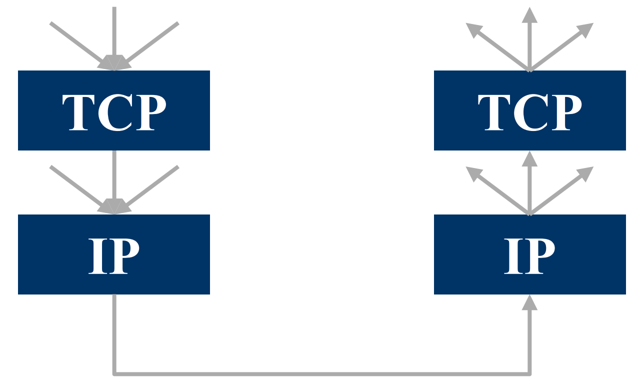
Multiplexing and Demultiplexing

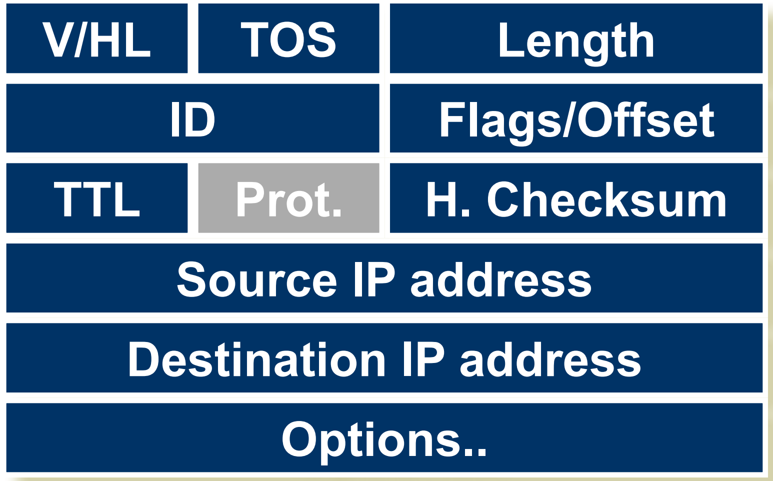
List of IP protocol numbers

From Wikipedia, the free encyclopedia

This is a list of IP numbers **used in the *Protocol* field of the** IPv4 hea

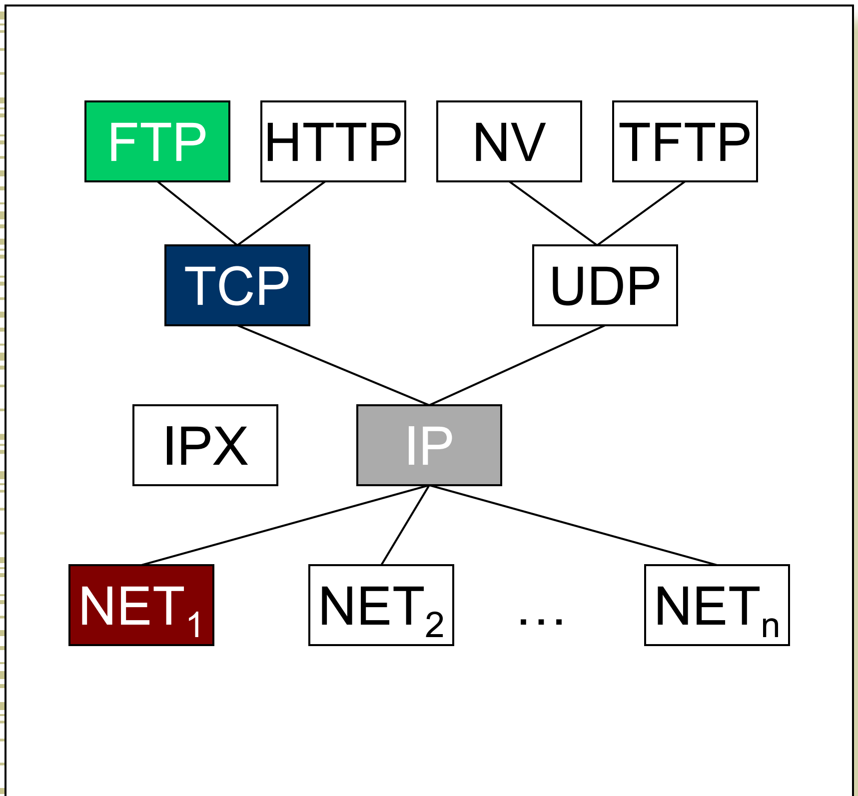
Decimal	Hex	Keyword	
0	0x00	HOPOPT	IPv6 Hop-by-Hop Option
1	0x01	ICMP	Internet Control Message Prot
2	0x02	IGMP	Internet Group Management F
3	0x03	GGP	Gateway-to-Gateway Protocol
4	0x04	IP-in-IP	IP in IP (encapsulation)
5	0x05	ST	Internet Stream Protocol
6	0x06	TCP	Transmission Control Protocol
7	0x07	CBT	Core-based trees
8	0x08	EGP	Exterior Gateway Protocol
9	0x09	IGP	Interior Gateway Protocol (any their IGRP))
10	0x0A	BBN-RCC-MON	BBN RCC Monitoring
11	0x0B	NVP-II	Network Voice Protocol
12	0x0C	PUP	Xerox PUP
13	0x0D	ARGUS	ARGUS

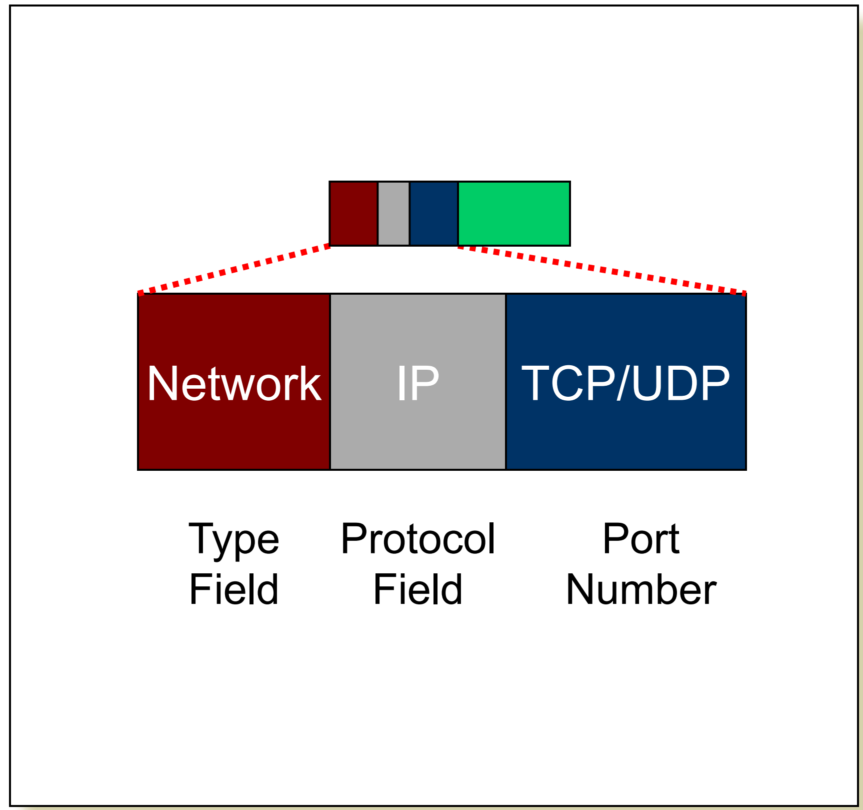




Protocol Demultiplexing

- Multiple choices at each layer





Today's Lecture

- Network links and LANs
- Layering and protocols
- Internet 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

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?

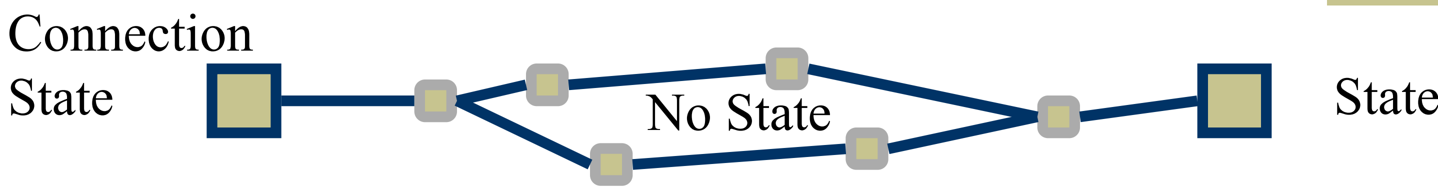
	State in Network	State in Host
Failure handing	Replication	“Fate sharing”
Net Engineering	Tough	Simple
Routing state	Maintain state	Stateless
Host trust	Less	More

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?

	State in Network	State in Host
Failure handing	Replication	“Fate sharing”
Net Engineering	Tough	Simple
Routing state	Pkts on same path: complex	Pkts on indep. paths: simple
Host trust	Less	More

Fate Sharing



- Definition*: 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
  - Is this still true in today’s network?
    - NATs and firewalls
- Tradeoffs
  - Less information available to the network
  - Must trust endpoints more

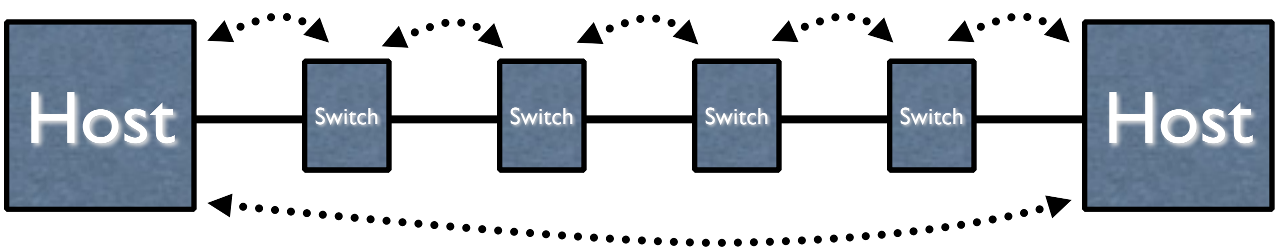
Networks [including end points] Implement Many Functions

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

Design Question

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

Option 1: Hop-by-hop (at switches)



Option 2: end-to-end (at end-hosts)

Options

- Hop-by-hop: Have each switch/router along the path ensure that the packet gets to the next hop
- End-to-end: Have just the end-hosts ensure that the packet made it through
- What do we have to think about to make this decision??

A question

- Is hop-by-hop enough?
  - [hint: What happens if a switch crashes? What if it’s buggy and goofs up a packet?]

End-to-End Argument

- Deals with where to place functionality
  - Inside the network (in switching elements)
  - At the edges
- Guideline not a law
- 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

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

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.
  - But by e2e, if you have to acknowledge the correct receipt of the entire file... why bother acknowledging the receipt of the individual packets???

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.
  - But by e2e, if you have to acknowledge the correct receipt of the entire file... why bother acknowledging the receipt of the individual packets???

- The answer: if you want performance, then you better do it this way (a mixture of e2e and in-network); imagine the waste if you had to retransmit the entire file because one packet was lost!

Internet Design: Types of Service

- **Principle**: network layer provides one simple service: best effort datagram (packet) delivery
  - All packets are treated the same

- Relatively simple core network elements
- Building block from which other services (such as reliable data stream) can be built
- Contributes to scalability of network

- No QoS support assumed from below
  - In fact, some underlying nets (e.g., link/physical layer) only provide reliable delivery (not best effort)
    - This made Internet datagram service *less* useful!
  - Hard to implement QoS without network support
  - QoS is an ongoing debate…

User Datagram Protocol (UDP): An Analogy

UDP

- Single socket to receive messages
- No guarantee of delivery
- Not necessarily in-order delivery
- Datagram – independent packets
- Must address each packet

Postal Mail

- 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

- Reliable – guarantee delivery
- Byte stream – in-order delivery
- Connection-oriented – single socket per connection
- Setup connection followed by data transfer

Telephone Call

- Guaranteed delivery
- In-order delivery
- Connection-oriented
- Setup connection followed by conversation

Example TCP applications
Web, Email, Telnet

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?
  - Has semantics that may be too strong for the app (e.g., Netflix streaming)

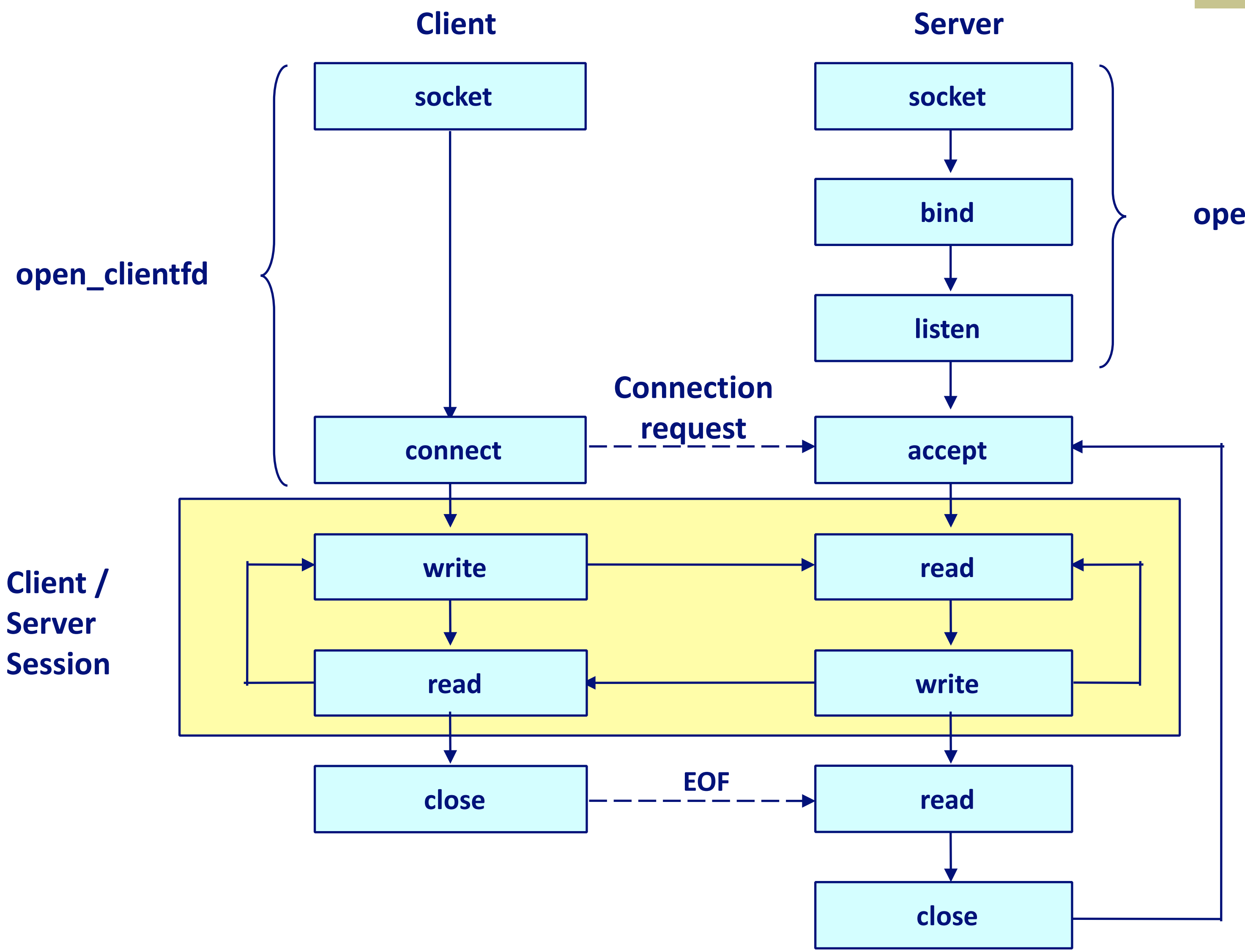
Design trade-off

- 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

Socket API Operation Overview



Blocking sockets

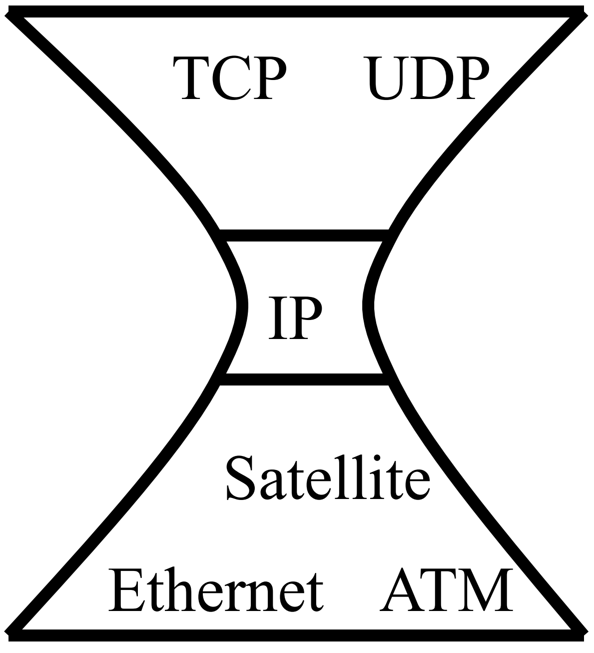
- What happens if an application write()s to a socket waaaaay faster than the network can send the data?
  - TCP figures out how fast to send the data...
  - And it builds up in the kernel socket buffers at the sender... and builds...
  - until they fill. The next write() call *blocks* (by default).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

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