CS162
Operating Systems and Systems Programming
Lecture 21

Networking

April 8, 2010

Ion Stoica

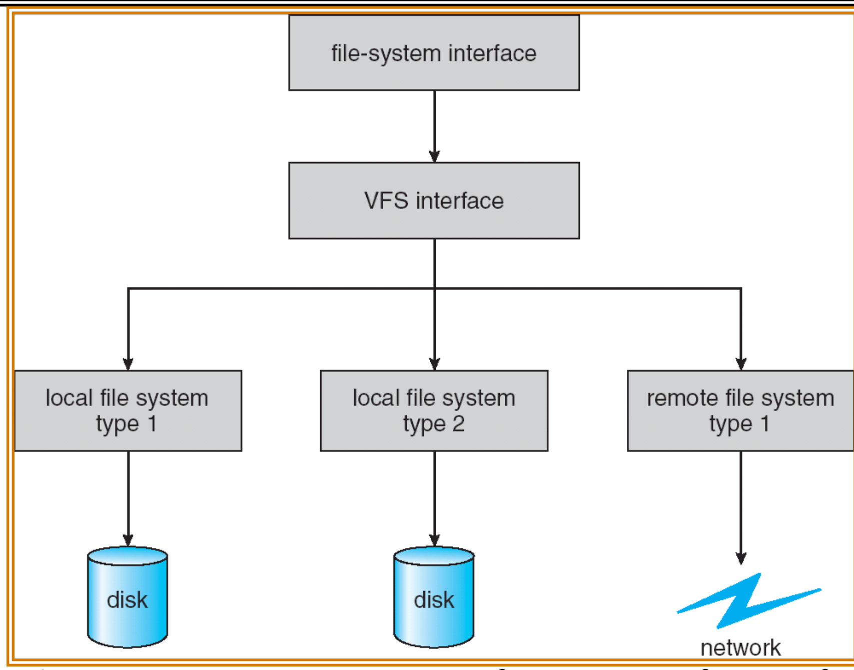
http://inst.eecs.berkeley.edu/~cs162

Goals for Today

- Distributed file systems
- Authorization
- Networking
  - Broadcast
  - Point-to-Point Networking
  - Routing
  - Internet Protocol (IP)

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Many slides generated from lecture notes by Kubiatowicz.

Remote File Systems: Virtual File System (VFS)

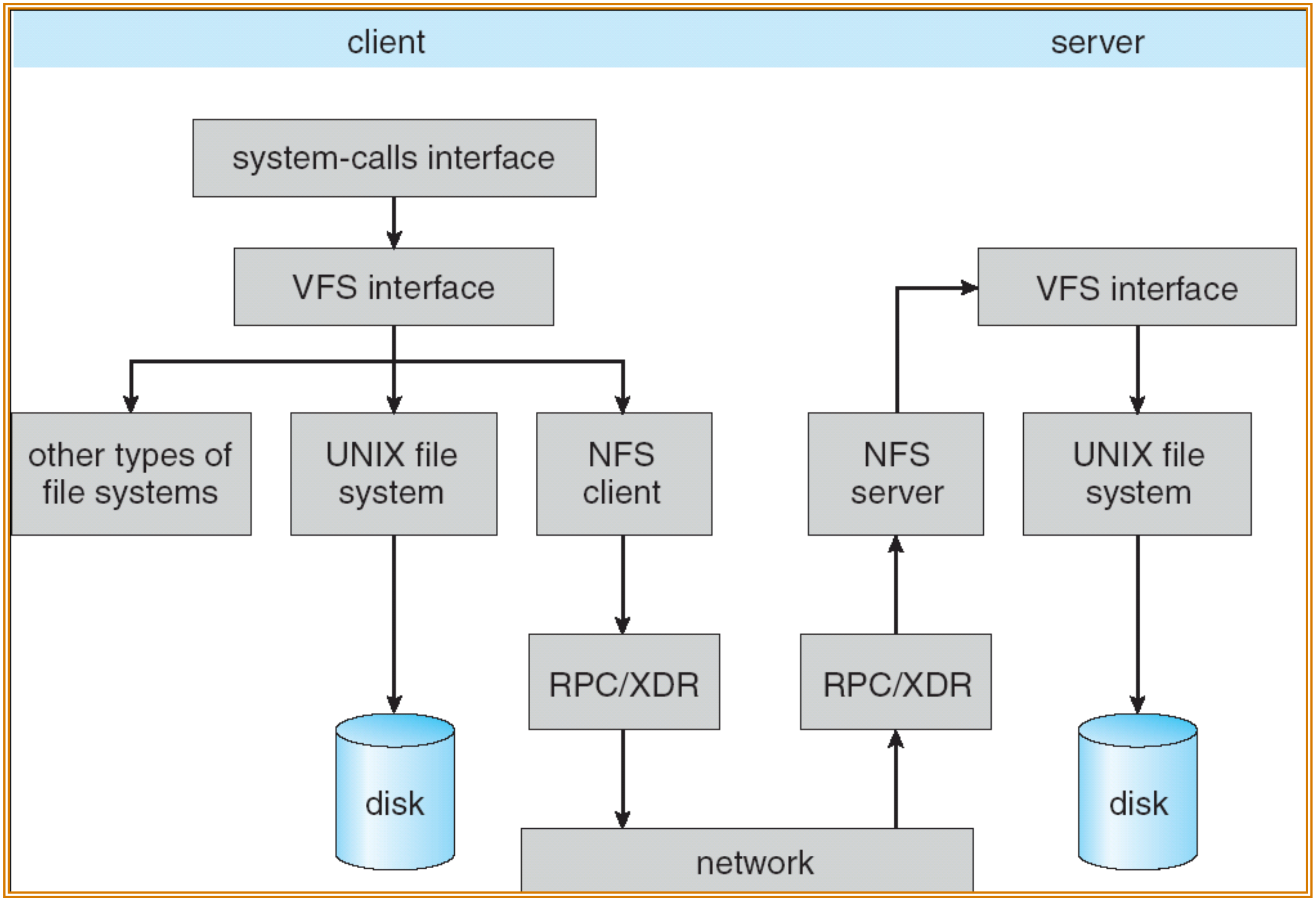


- VFS: Virtual abstraction similar to local file system
  - Instead of "inodes" has "vnodes"
  - Compatible with a variety of local and remote file systems
    - » provides object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - The API is to the VFS interface, rather than any specific type of file system

Network File System (NFS)

- Three Layers for NFS system
  - UNIX file-system interface: open, read, write, close calls + file descriptors
  - VFS layer: distinguishes local from remote files
    - » Calls the NFS protocol procedures for remote requests
  - NFS service layer: bottom layer of the architecture
    - » Implements the NFS protocol
- NFS Protocol: remote procedure calls (RPC) for file operations on server
  - Reading/searching a directory
  - manipulating links and directories
  - accessing file attributes/reading and writing files
- NFS servers are stateless; each request provides all arguments require for execution
- Modified data must be committed to the server's disk before results are returned to the client
  - lose some of the advantages of caching
  - Can lead to weird results: write file on one client, read on other, get old data

Schematic View of NFS Architecture



Authorization: Who Can Do What?

- How do we decide who is authorized to do actions in the system?
- **Access Control Matrix:** contains all permissions in the system
  - Resources across top
    - » Files, Devices, etc…
  - Domains in columns
    - » A domain might be a user or a group of users
    - » E.g. above: User D3 can read F2 or execute F3
  - In practice, table would be huge and sparse!

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Authorization: Two Implementation Choices

- **Access Control Lists:** store permissions with object
  - Still might be lots of users!
  - UNIX limits each file to: r,w,x for owner, group, world
  - More recent systems allow definition of groups of users and permissions for each group
  - ACLs allow easy changing of an object's permissions
    - » Example: add Users C, D, and F with rw permissions
- **Capability List:** each process tracks which objects has permission to touch
  - Popular in the past, idea out of favor today
  - Consider page table: Each process has list of pages it has access to, not each page has list of processes …
  - Capability lists allow easy changing of a domain's permissions
    - » Example: you are promoted to system administrator and should be given access to all system files

Authorization: Combination Approach

- Users have capabilities, called "groups" or "roles"
  - Everyone with particular group access is "equivalent" when accessing group resource
  - Like passport (which gives access to country of origin)

- Objects have ACLs
  - ACLs can refer to users or groups
  - Change object permissions object by modifying ACL
  - Change broad user permissions via changes in group membership
  - Possessors of proper credentials get access

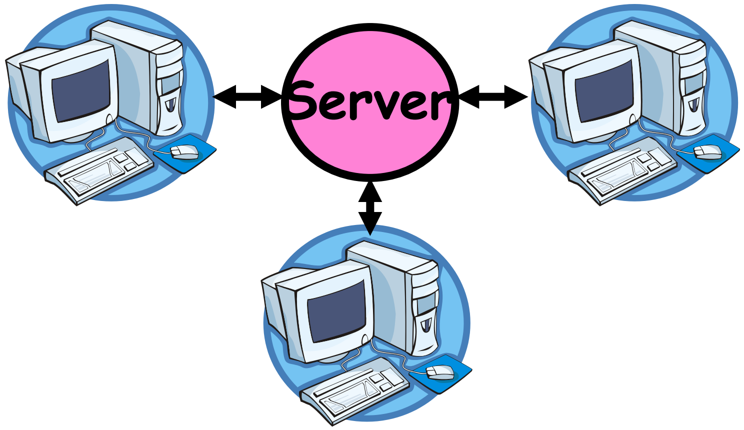
Authorization: How to Revoke?

- How does one revoke someone's access rights to a particular object?
  - Easy with ACLs: just remove entry from the list
  - Takes effect immediately since the ACL is checked on each object access
- Harder to do with capabilities since they aren't stored with the object being controlled:
  - Not so bad in a single machine: could keep all capability lists in a well-known place (e.g., the OS capability table).
  - Very hard in distributed system, where remote hosts may have crashed or may not cooperate (more in a future lecture)

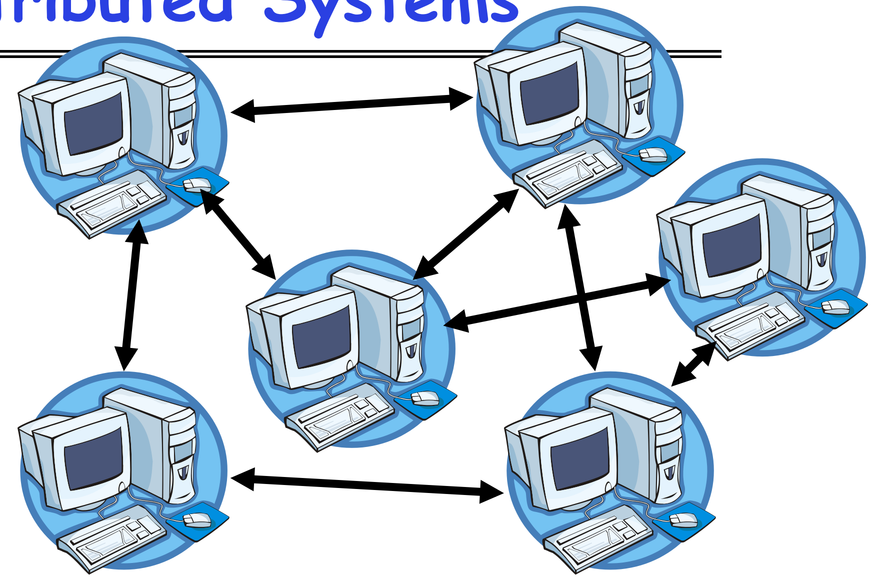
Revoking Capabilities

- Various approaches to revoking capabilities:
  - Put expiration dates on capabilities and force reacquisition
  - Put epoch numbers on capabilities and revoke all capabilities by bumping the epoch number (which gets checked on each access attempt)
  - Maintain back pointers to all capabilities that have been handed out (Tough if capabilities can be copied)
  - Maintain a revocation list that gets checked on every access attempt

Centralized vs Distributed Systems



Client/Server Model



Peer-to-Peer Model

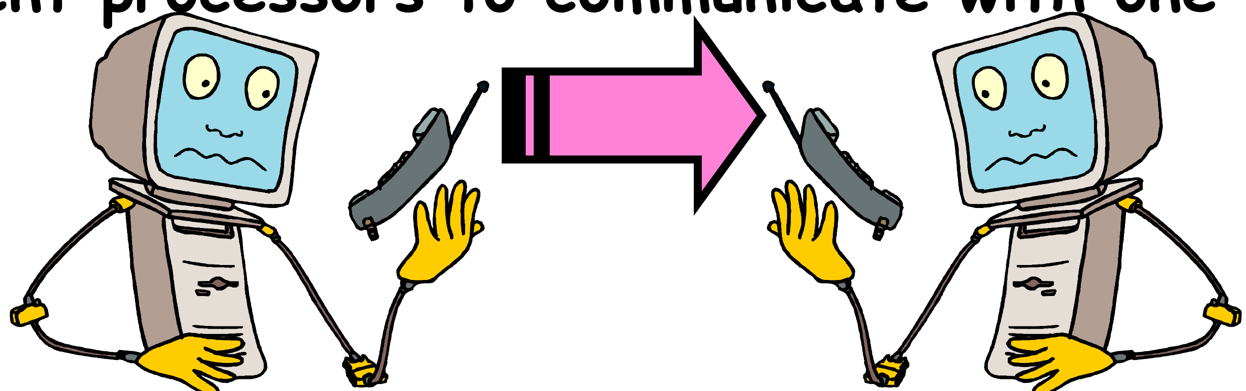
- **Centralized System:** System in which major functions are performed by a single physical computer
  - Originally, everything on single computer
  - Later: client/server model
- **Distributed System:** physically separate computers working together on some task
  - Early model: multiple servers working together
    - » Probably in the same room or building
    - » Often called a "cluster"
  - Later models: peer-to-peer/wide-spread collaboration

Distributed Systems: Motivation/Issues

- Why do we want distributed systems?
  - Cheaper and easier to build lots of simple computers
  - Easier to add power incrementally
  - Users can have complete control over some components
  - Collaboration: Much easier for users to collaborate through network resources (such as network file systems)
- The *promise* of distributed systems:
  - Higher availability: one machine goes down, use another
  - Better durability: store data in multiple locations
  - More security: each piece easier to make secure
- Reality has been disappointing
  - Worse availability: depend on every machine being up
    - » Lamport: "a distributed system is one where I can't do work because some machine I've never heard of isn't working!"
  - Worse reliability: can lose data if any machine crashes
  - Worse security: anyone in world can break into system
- Coordination is more difficult
  - Must coordinate multiple copies of shared state information (using only a network)
  - What would be easy in a centralized system becomes a lot more difficult

Distributed Systems: Goals/Requirements

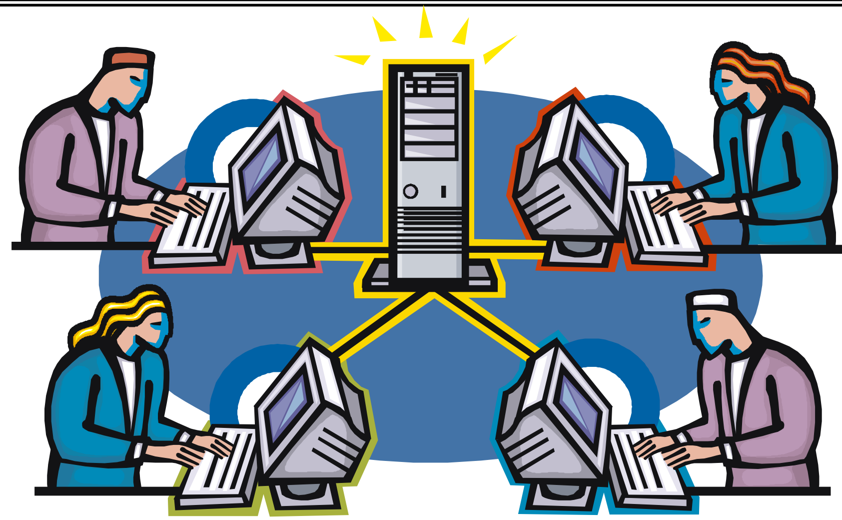
- **Transparency:** the ability of the system to mask its complexity behind a simple interface
- Possible transparencies:
  - **Location:** Can't tell where resources are located
  - **Migration:** Resources may move without the user knowing
  - **Replication:** Can't tell how many copies of resource exist
  - **Concurrency:** Can't tell how many users there are
  - **Parallelism:** System may speed up large jobs by spliting them into smaller pieces
  - **Fault Tolerance:** System may hide varoius things that go wrong in the system
- Transparency and collaboration require some way for different processors to communicate with one another



Administrivia

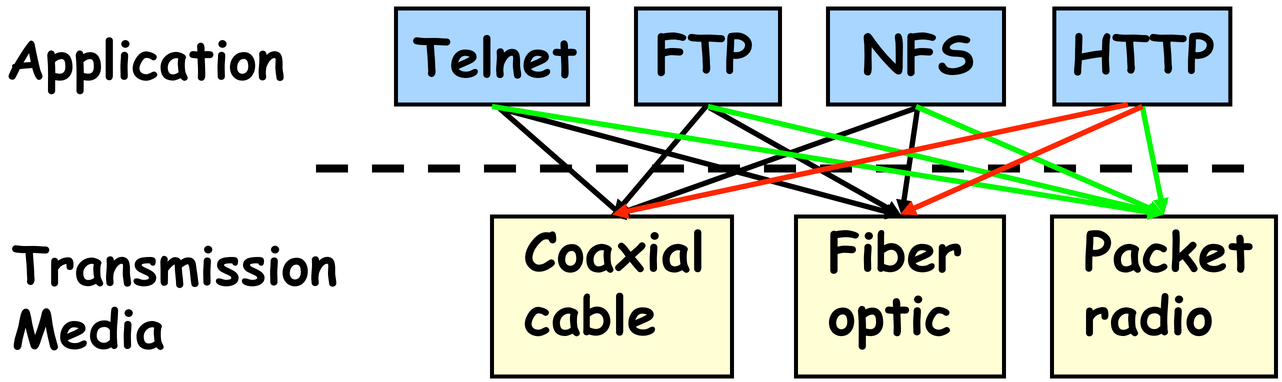
- 3rd project due Monday, April 12
- I'll be away next Wednesday-Friday (Eurosys)
  - Lecture will be taught by Ben Hindman
  - No office hour on Thursday, April 15
- Matei and Andy will be away as well next week
  - Ben will teach the discussion sections of both Matei and Andy
  - No office hours for Andy and Matei next week
- Project 4
  - Initial design, Wednesday (4/21), will give you two discussion sections before deadline
  - Code deadline, Wednesday (5/5), two weeks later

Networking Definitions



- **Network:** physical connection and set of protocols that allows two computers to communicate
- **Packet (frame):** unit of transfer, sequence of bits carried over the network
  - Network carries packets from one CPU to another
  - Destination gets interrupt when packet arrives
- **Protocol:** agreement between two parties as to how information is to be transmitted
- **Layering:** architecture for networking functionality

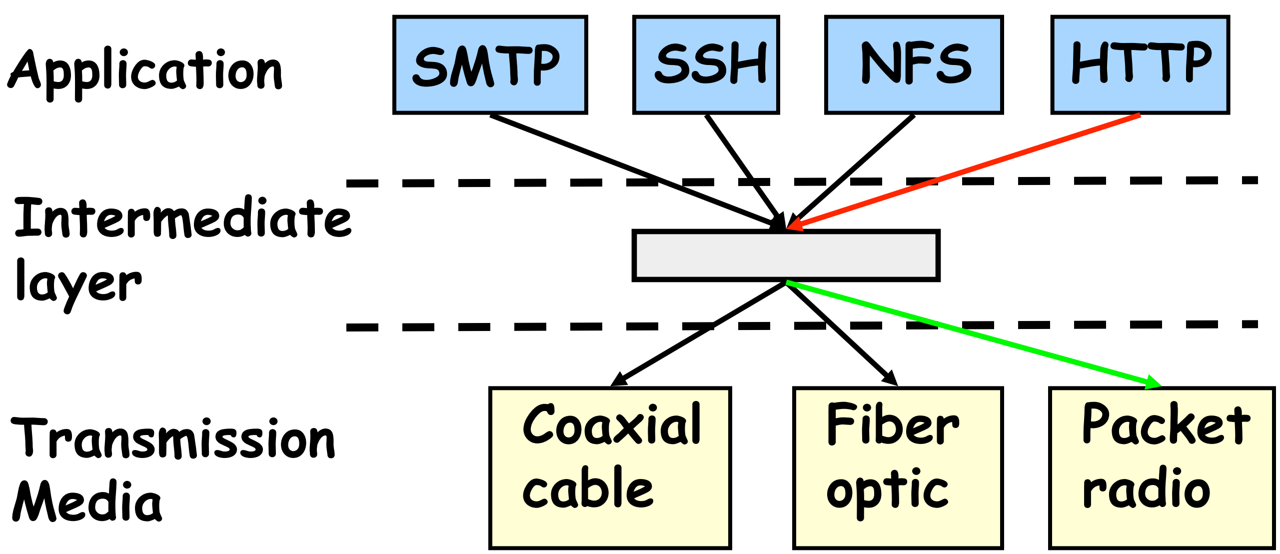
Why Layering? The Problem



- Re-implement every application for every technology?
- No! But how does the Internet architecture avoid this?

Network Layering: Solution

- Introduce an intermediate layer that provides a single abstraction for various network technologies
  - New application just need to be written for intermediate layer
  - New transmission media just need to provide abstraction of intermediate layer



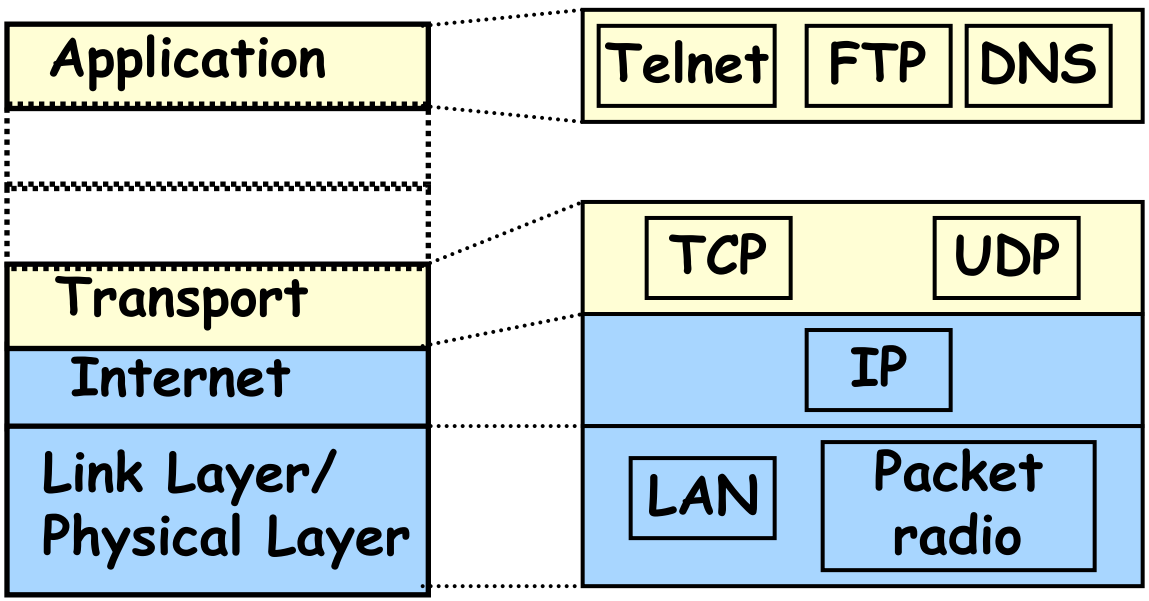
Layering

- Layering is a particular form of modularization
- System is broken into a vertical hierarchy of logically distinct entities (layers)
- Service provided by one layer is based solely on the service provided by layer below
- Rigid structure: easy reuse, performance suffers

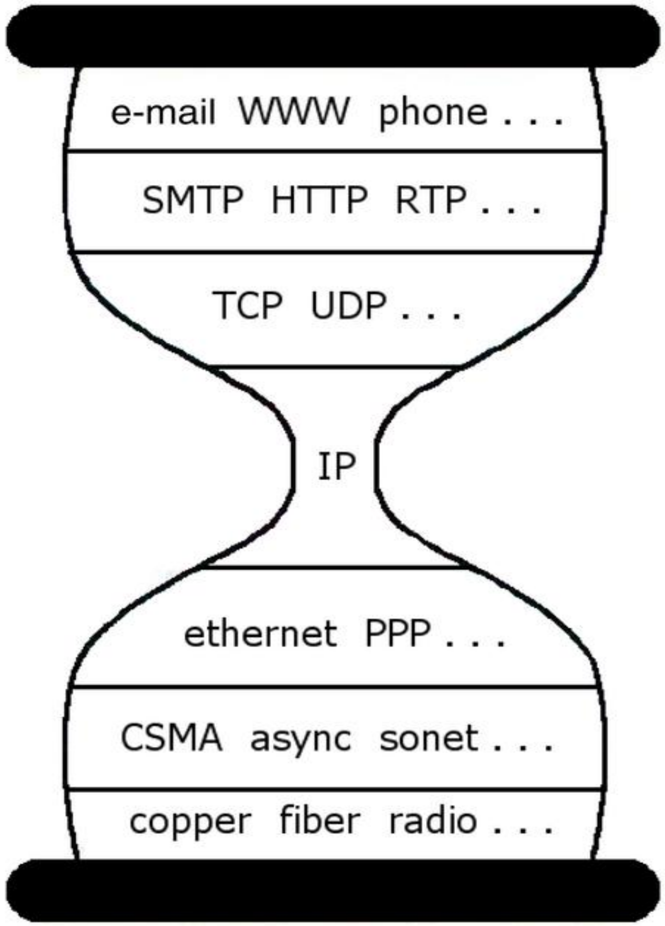
Layering: Internet

Universal Internet layer:

- Internet has only Internet Protocol (IP) at the Internet layer
- Many options for modules above IP
- Many options for modules below IP



Hourglass


e-mail WWW phone . . .
SMTP HTTP RTP . . .
TCP UDP . . .
IP
ethernet PPP . . .
CSMA async sonet . . .
copper fiber radio . . .

Implications of Hourglass

Single Internet layer module:

- Allows networks to interoperate
  – Any network technology that supports IP can exchange packets
- Allows applications to function on all networks
  – Applications that can run on IP can use any network

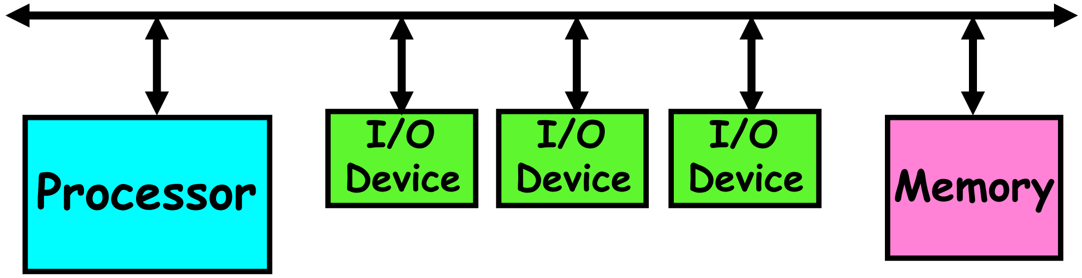
- Simultaneous developments above and below IP

Link Layer

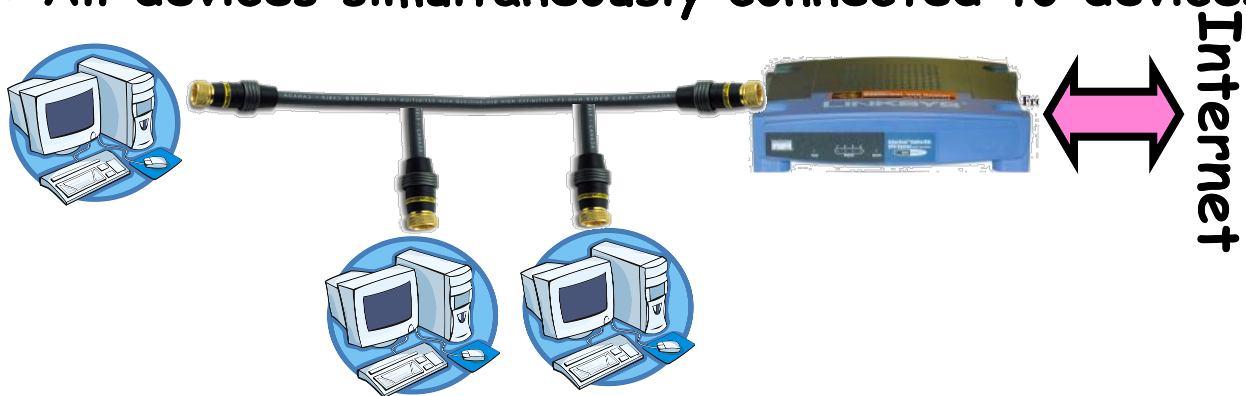
- Shared broadcast network: a packet reaches everyone in same network
- Frames: units of data exchanged at the link layer
- Main Functions
  - Create frames, adding header, trailer
  - Error correction
  - Send data between peers
  - Arbitrate access to physical media (Multiple Access)

Broadcast Networks

- **Broadcast Network:** Shared Communication Medium

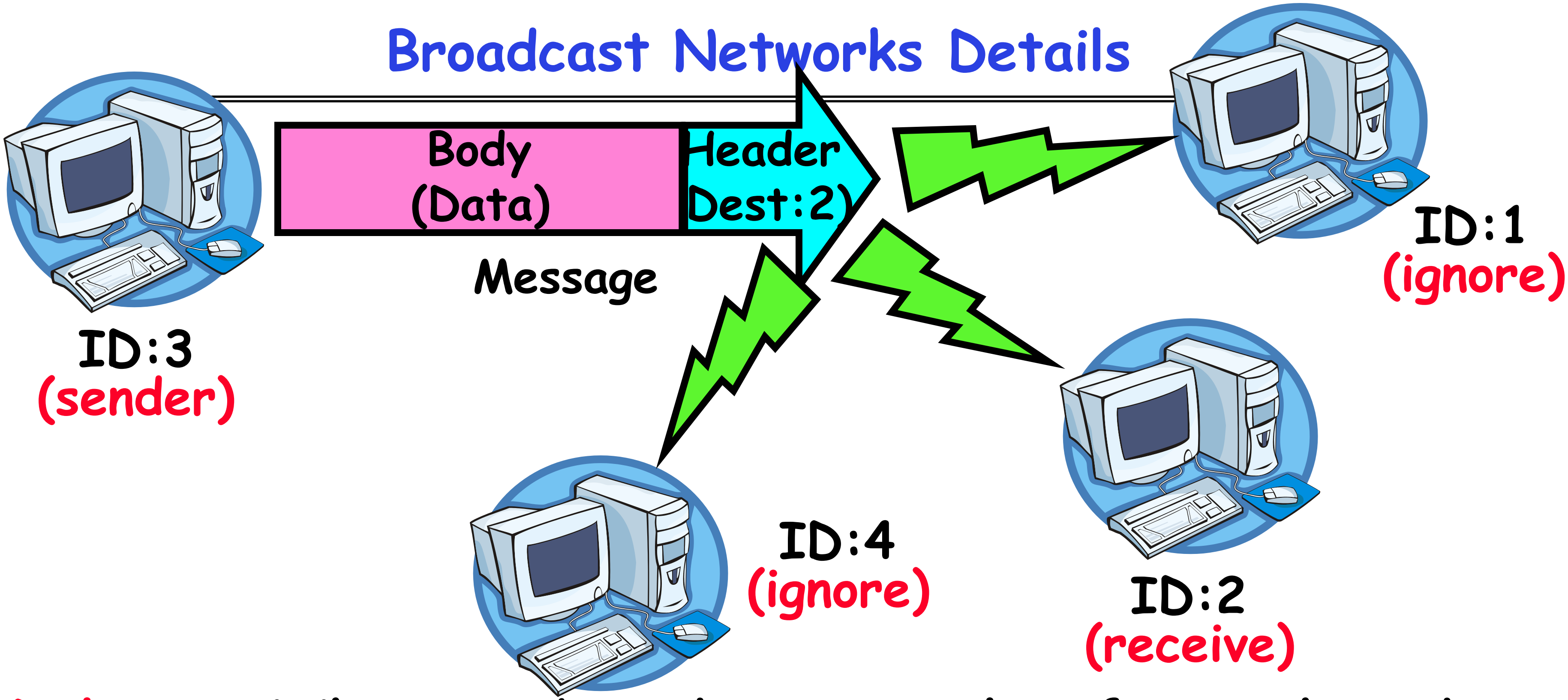


  - Shared Medium can be a set of wires
    - » Inside a computer, this is called a bus
    - » All devices simultaneously connected to devices



  - Originally, Ethernet was a broadcast network
    - » All computers on local subnet connected to one another
  - More examples (wireless: medium is air): cellular phones, GSM GPRS, EDGE, CDMA 1xRTT, and 1EvDO

Broadcast Networks Details



- **Delivery:** When you broadcast a packet farme, how does a receiver know who it is for? (frame goes to everyone!)
  - Put header on front of frame: [Destination | Packet]
  - Everyone gets frame, discards if not the target
  - In Ethernet, this check is done in hardware
    - » No OS interrupt if not for particular destination
  - This is layering: we're going to build complex network protocols by layering on top of the packet

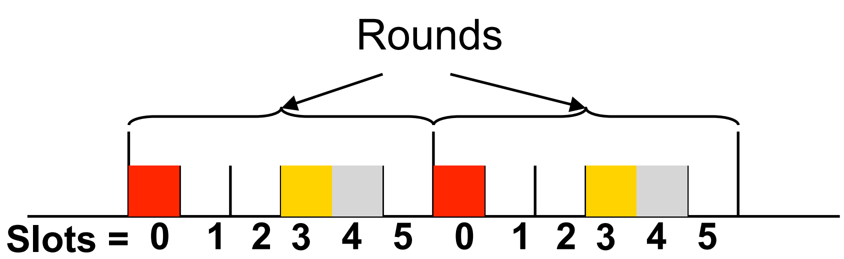
Multiple Access Algorithm

- Single shared broadcast channel
  - Avoid having multiple nodes speaking at once
  - Otherwise, collisions lead to garbled data
- Multiple access mechanism
  - Distributed algorithm for sharing the channel
  - Algorithm determines which node can transmit
- Classes of techniques
  - Channel partitioning: divide channel into pieces
  - Taking turns: scheme for trading off who gets to transmit
  - Random access: allow collisions, and then recover
    - » *Optimizes for the common case of only one sender*

Channel Partitioning: TDMA

TDMA: Time Division Multiple Access

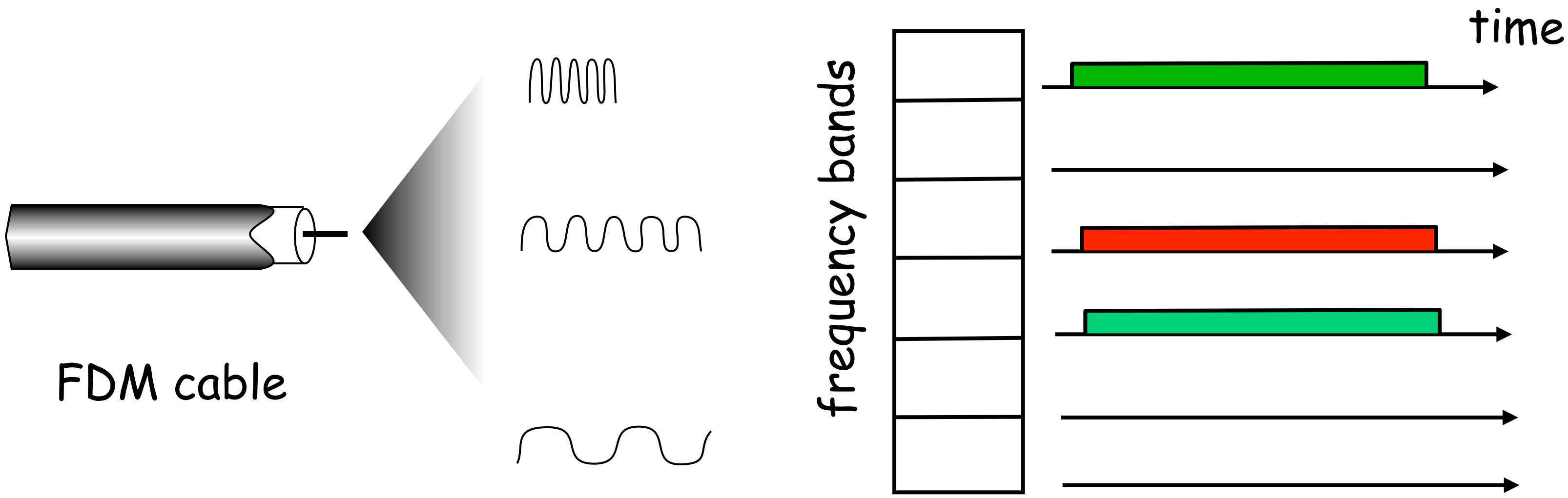
- Access to channel in "rounds"
  - Each station gets fixed length slot in each round
- Time-slot length is packet transmission time
  - *Unused slots go idle*
- Example: 6-station LAN with slots 0, 3, and 4



Channel Partitioning: FDMA

FDMA: Frequency Division Multiple Access

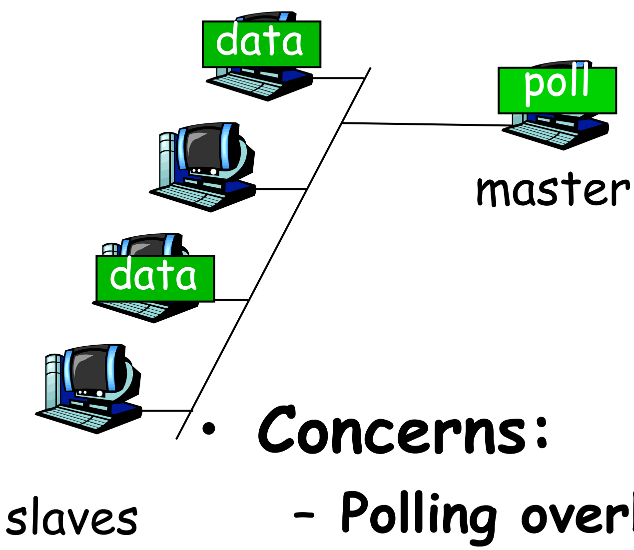
- Channel spectrum divided into frequency bands
- Each station assigned fixed frequency band
- Unused transmission time in frequency bands go idle
- Example: 6-station LAN, 1,3,4 have pkt, frequency bands 2,5,6 idle



"Taking Turns" MAC protocols

Polling

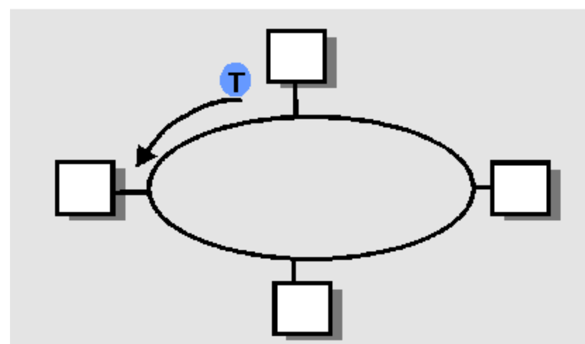
- Master node "invites" slave nodes to transmit in turn



- Concerns:
  - Polling overhead
  - Latency
  - Single point of failure (master)

Token passing

- Control token passed from one node to next sequentially
- Node must have token to send
- Concerns:
  - Token overhead
  - Latency
  - Single point of failure (token)



Random Access Protocol: AlohaNet

- Norm Abramson left Stanford in search of surfing
- Set up first radio-based data communication system connecting the Hawaiian islands
  - Hub at Alohanet HQ (Univ. Hawaii, Oahu)
  - Other sites spread among the islands
- Had two radio channels:
  - Random access: sites sent data on this channel
  - Broadcast: only used by hub to rebroadcast incoming data

Aloha Transmission Strategy

- When new data arrived at site, send to hub for transmission
- Site listened to broadcast channel
  - If it heard data repeated, knew transmission was rec'd
  - If it didn't hear data correctly, it assumed a collision
- If collision, site waited *random* delay before retransmitting
- Problem: Stability: what if load increases?
  - More collisions ⇒ less gets through ⇒more resent ⇒ more load… ⇒ More collisions…
  - Unfortunately: some sender may have started in clear, get scrambled without finishing

Ethernet

- Bob Metcalfe, Xerox PARC, visits Hawaii and gets an idea!
- Shared medium (coax cable)
- Can "sense" carrier to see if other nodes are broadcasting at the same time
  - Sensing is subject to time-lag
  - Only detect those sending a short while before
- Monitor channel to detect collisions
  - Once sending, can tell if anyone else is sending too

Ethernet's CSMA/CD

- CSMA: Carrier Sense Multiple Access
- CD: Collision detection
- Sense channel, if idle
  - If detect another transmission
    - » Abort, send jam signal
    - » Delay, and try again
  - Else
    - » Send frame
- Receiver accepts:
  - Frames addressed to its own address
  - Frames addressed to the broadcast address (broadcast)
  - Frames addressed to a multicast address, if it was instructed to listen to that address
  - All frames (promiscuous mode)

Ethernet's CSMA/CD (more)

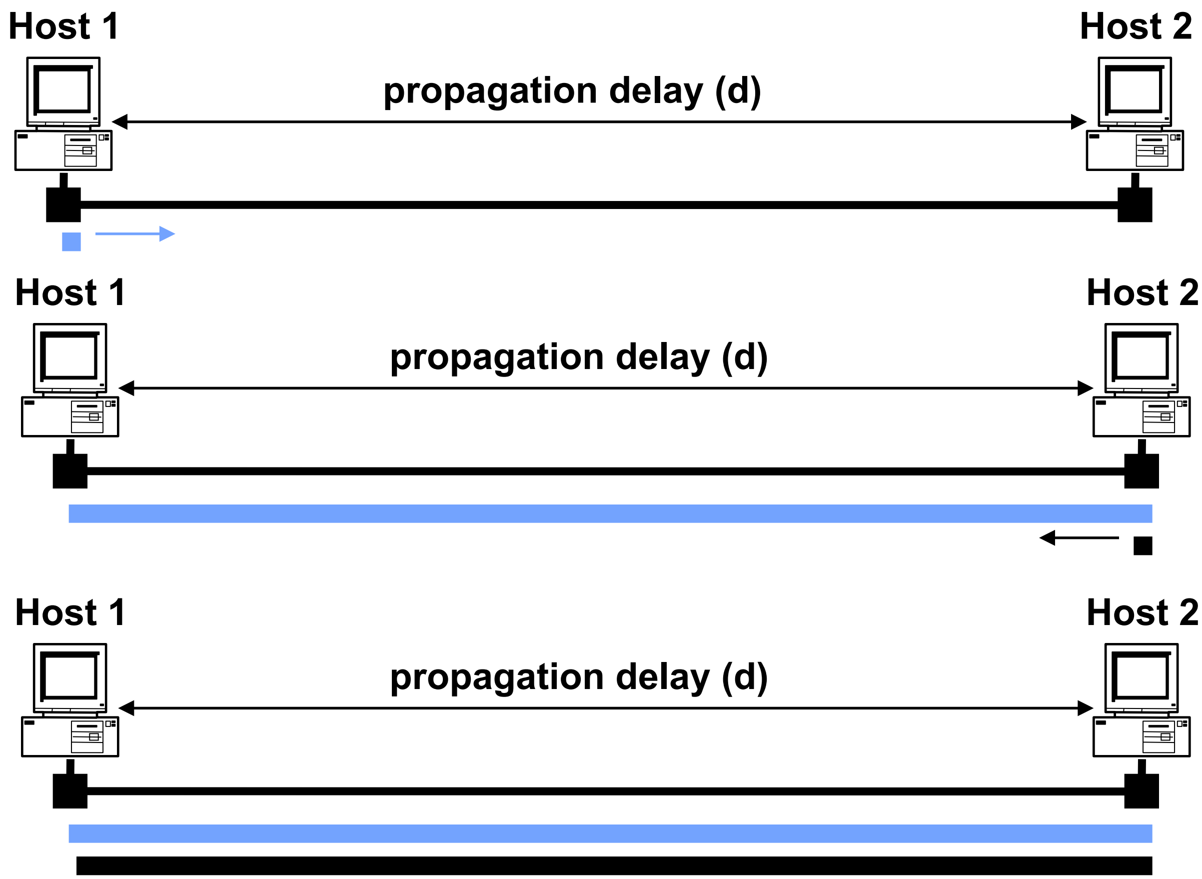
- Exponential back-off
  - Goal: adapt retransmission attempts to estimated current load
  - Heavy load: random wait will be longer
  - First collision: choose K from {0,1}; delay is K x 512 bit transmission times
  - After second collision: choose K from {0,1,2,3}…
  - After ten or more collisions, choose K from {0,1,2,3,4,…,1023}
- Minimum packet size
  - Give a host enough time to detect collisions
  - In Ethernet, minimum packet size = 64 bytes
  - What is the relationship between minimum packet size and the length of the LAN?

Minimum Packet Size (more)

a) Time = t; Host 1 starts to send frame

b) Time = t + d; Host 2 starts to send a frame, just before it hears from host 1's frame

c) Time = t + 2*d; Host 1 hears Host 2's frame → detects collision



d = LAN_length/ligh_speed = min_frame_size/(2*bandwidth) →

LAN_length = (min_frame_size)*(light_speed)/(2*bandwidth) =

= $(8*64b)*(2.5*10^8 mps)/(2*10^7 bps)$ = 6400m approx

What about 100 mbps? 1 gbps? 10 gbps?

Conclusion

- Authorization
  - Controlling access to resources using
    - » Access Control Lists
    - » Capabilities
- Network: physical connection that allows two computers to communicate
  - Packet: unit of transfer, sequence of bits carried over the network
- **Broadcast Network:** Shared Communication Medium
  - Transmitted packets sent to all receivers
  - Arbitration: act of negotiating use of shared medium
    - » Ethernet: Carrier Sense, Multiple Access, Collision Detect
- **Protocol:** Agreement between two parties as to how information is to be transmitted