CS162
Operating Systems and Systems Programming
Lecture 22

Networking III

April 22, 2010
Ion Stoica
http://inst.eecs.berkeley.edu/~cs162

Review

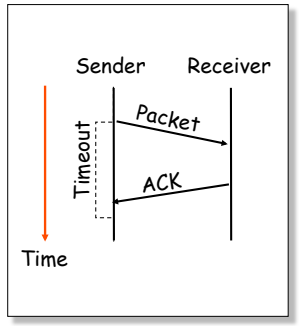
- Link (datalink) layer: Broadcast network; frames sent by one host reaches every other host in same network
  - Multi-access protocol
  - (didn't go over) construct frames, error detection and correction, flow control, …
- Network layer: stitch together multiple link layer networks
  - Deliver a packet to specified network destination
  - (didn't go over) segmentation/reassemble, packet scheduling, buffer management
- Transport layer
  - Multiplexing/demultiplexing (two lectures ago)
  - Flow & congestion control, in-order delivery, reliability (today)

Transport Protocol

- Flow control keeps *one fast sender* from overwhelming *a slow receiver*
- Congestion control keeps a *set of senders* from overloading the *network*
- Reliability makes sure the *receiver got all packets* sent by *sender*
- In-order delivery makes sure the *receiver delivers the packet to application in same order sender* sent them
- Two protocols:
  - Stop-and-Wait
  - Window based

Automatic Repeat reQuest (ARQ)

- Automatic Repeat Request
  - Receiver sends acknowledgment (ACK) when it receives packet
  - Sender waits for ACK and *times out* if does not arrive within some time period
- Simplest ARQ protocol
  - Stop and Wait
  - Send a packet, stop and wait until ACK arrives

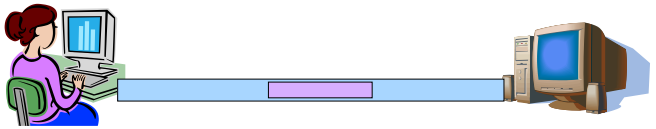


Stop-and-Wait Properties

- Flow control: yes
  - Receiver can implicitly slow down sender by acking a packet only if it has room for at lest another packet
  - Assumption: timeout doesn't trigger before receiving ack
- Congestion control: yes
  - Sender sends a new packet only after previous one made it
  - If network is congested packet or ack is lost → sender doesn't send new data
- Reliability: yes
  - If a packet is lost, sender timeouts and resends the packet
- In-order delivery: yes
  - Receiver doesn't get next packet before receiving (and acking) previous one
- So what's the problem with Stop-and-Wait? Efficiency!

How Fast Can Stop-and-Wait Go?

- Suppose we're sending from UCB to New York:
  - Bandwidth = 1 Mbps (megabits/sec)
  - RTT = 100 msec
  - Maximum packet size a.k.a. Maximum Transmission Unit (MTU) = 1500 B = 12,000 b
  - No other load on the path and no packet loss
- What (approximately) is the fastest we can transmit using Stop-and-Wait?
  - Answer: 12,000b/0.1s = 120 kbps
- How about if Bandwidth = 1 Gbps?

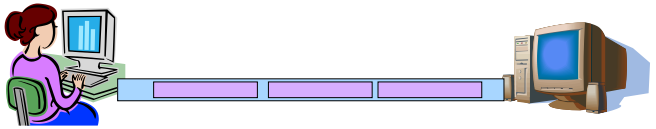


Administrivia

- Keys to access AWS will be sent today
- Last two lectures on security
- Final Exam
  - Friday, May 14, 7:00PM-10:00PM
  - All material from the course
    - » With slightly more focus on second half, but you are still responsible for all the material
  - Two sheets of notes, both sides

Sliding Window

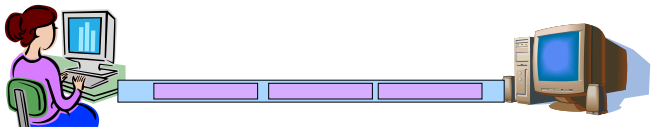
- Idea: allow multiple packets in-flight
  - "In-flight" = un-acked packets
- Window size (W): number of packets the sender can send without receiving an ack
  - E.g., after receiving ack for all packet before and including K, send packets K+1, K+2, …, K+W+1
  - Stop-and-wait: particular case of sliding window, W=1
- Receiver tells sender W
  - W cannot be larger than receiver's buffer!



Throughput

- Up to W packets (or bytes) per RTT
- Throughput = W/RTT

- How large should be the window to fully utilize a link with bandwidth B?
  - W = Bandwidth x RTT (i.e., "Bandwidth-Delay" or "Delay-Bandwidth" product)

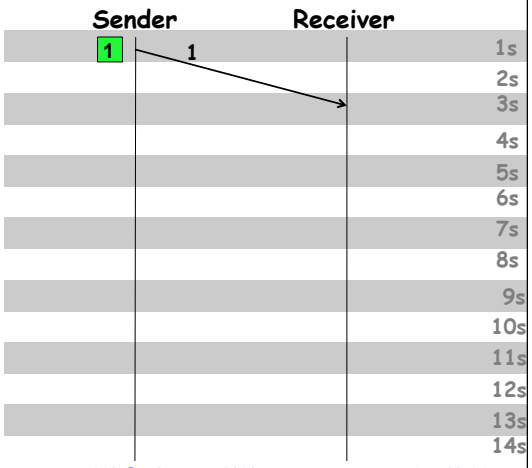


Sliding Window Example (This is NOT TCP !)

- Sender
  - Sending rate = 1 pkt/s
  - Packet size = 1000b
- Receiver:
  - Delivering rate = 0.5 pkt/s
  - Delivers packets in sequence to application
  - Acknowledges (acks) each delivered pkt
  - Send negative ack. (nack) if packet lost
- Round-trip time = 4 sec, 2sec each way
- Receiver Window = 4 packets
- Note: max. achievable throughput = 0.5pkt/s = 500b/s

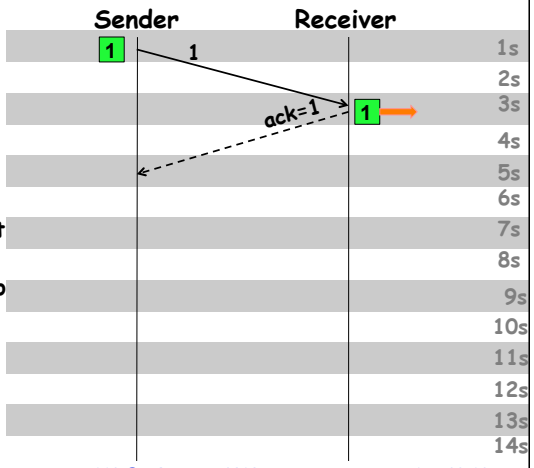
Sliding Window Example

- Sender, at 1s
  - Send 1st pkt



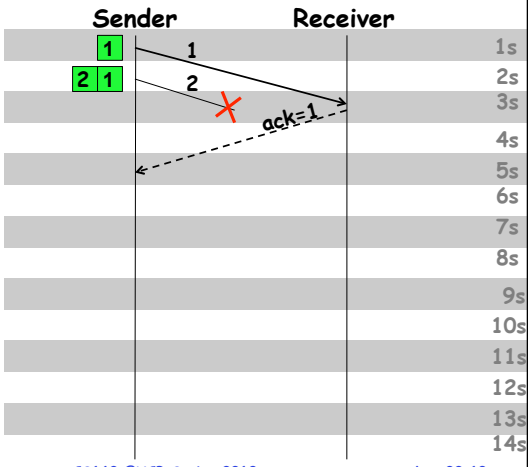
Sliding Window Example

- Sender, at 1s
  - Send 1st pkt
- Receiver, at 3s
  - Get 1st pkt
  - Deliver 1st pkt to appl.
  - Send ack=1 to sender



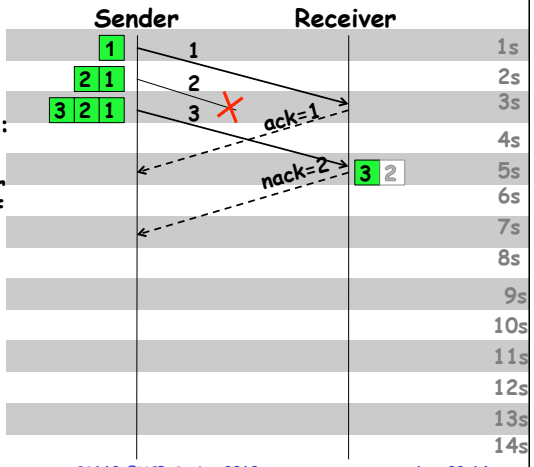
Sliding Window Example

- Sender, at 2s
  - Send 2nd pkt, which is lost



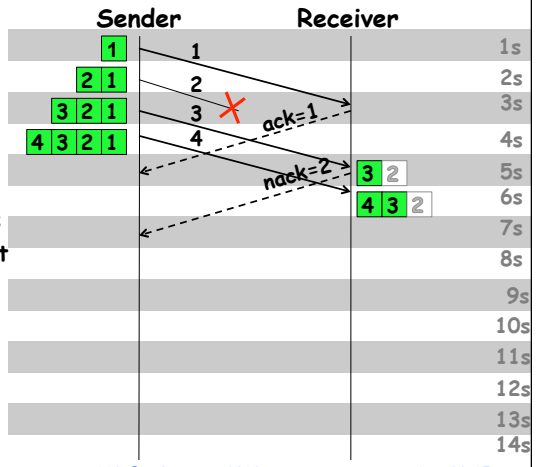
Sliding Window Example

- Sender, at 3s
  - Send 3nd pkt
- Receiver, at 5s:
  - Get 3rd pkt; doesn't deliver it since out of seq.
  - Send nack=2 (request 2nd pkt)



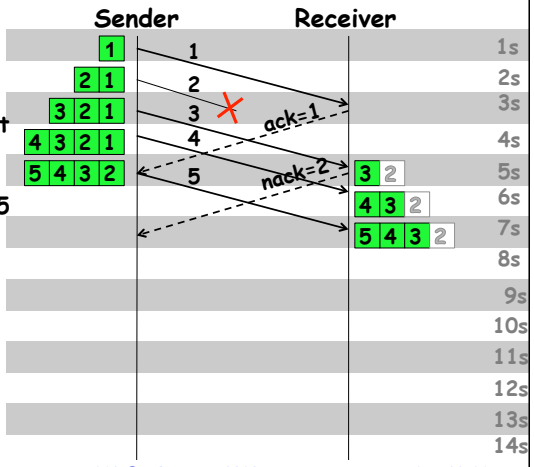
Sliding Window Example

- Sender, at 4s
  - Send 4th pkt
  - Receiver window full!
- Receiver, at 6s
  - Get 4th packet



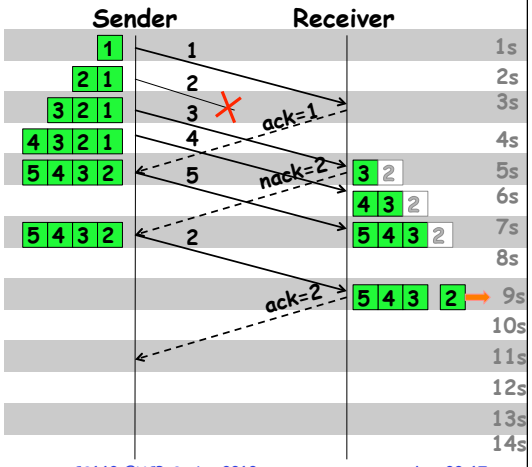
Sliding Window Example

- Sender, at 5s
  - Get ack=1
  - Remove 1st pkt from buffer
  - Send 5th pkt; now 2, 3, 4, 5 are in flight (window full!)
- Receiver, at 7s
  - Get 5th pkt



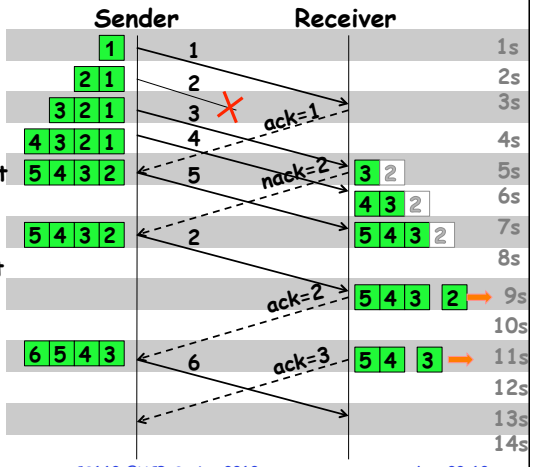
Sliding Window Example

- Sender, at 7s
  - Get nack=2
  - Resend pkt 2
- Receiver, at 9s
  - Get 2nd pkt
  - Deliver it to appl.
  - Send ack=2



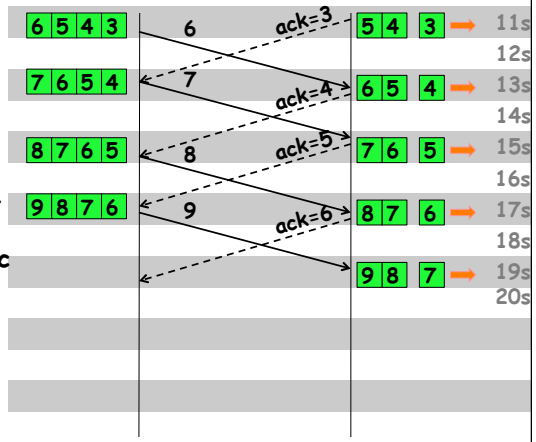
Sliding Window Example

- Sender, at 11s
  - Get ack=2
  - Send pkt 6; pkts 3, 4, 5, 6 are in-flight
- Receiver, at 11s
  - Deliver 3d pkt to appl. (recall, delivery rate is 1pkt every 2s)
  - Send ack=3



Sliding Window Example

- If no more losses, throughput = 0.5pkt/sec
- This is max throughput as receiver cannot deliver more than 0.5pkt/sec



Performance with Sliding Window

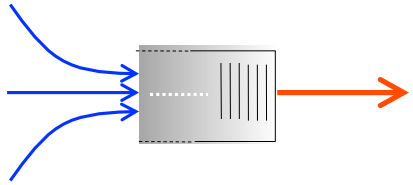
- Given previous
  - UCB ↔ New York 1 Mbps path with 100 msec RTT, and
  - Sender (and Receiver) window = 100 Kb = 12.5 KB
- How fast can we transmit?
  - Answer: min(100Kb/0.1s, 1Mbps) = 1 Mbps
- What about with 12.5 KB window & 1 Gbps path?
- Window required to fully utilize path:
  - W = Bandwidth x RTT = 1 Gbps * 100 msec = 100 Mb = 12.5 MB
  - Note: large window = many packets in flight

Sliding Window Properties

- Flow control: yes
  - Receiver tells the sender how many packets it can send without hearing an ack (windaw size)
- Congestion control: not really. Why?
- Reliability: yes
  - Sender resends lost packet on receiving "nack" or on timeout
- In-order delivery: yes
  - Use sequence numbers for packets;
  - Receiver delivers in-sequence packets to app; if a packet is missing, stop and wait for the packet to be retransmitted;

Congestion

- Two packets arrive at the same time
  - The node can only transmit one
  - … and either buffers or drops the other
- If many packets arrive in a short period of time
  - The node cannot keep up with the arriving traffic
  - … and the buffer may eventually overflow



Congestion Collapse

- Definition: Increase in network load results in a decrease of useful work done
- Due to:
  - Undelivered packets
    - » Packets consume resources and are dropped later in network
  - Spurious retransmissions of packets still in flight
    - » Unnecessary retransmissions lead to more load!
    - » *Pouring gasoline on a fire*
- Mid-1980s: Internet grinds to a halt
  - Until Jacobson/Karels (Berkeley!) devise TCP congestion control

Two Basic Components (TCP)

- Detect congestion = detect packet loss
  - ACK denotes next byte (*n*) expected to be received
    - » Receiver acks it has received all bytes up to *n-1*
  - Two signs of packet loss
    - » No ACK after certain time interval: time-out
    - » Several duplicate ACKs (receiver misses packet starting with byte *n+1*, and has received several packets after that)
- Dealing with congestion:
  - Probe network to test level of congestion
  - Speed up when no congestion
  - Slow down when congestion
  - Suboptimal, messy dynamics, simple to implement

TCP Congestion Control

- TCP connection has window
  - Controls number of unacknowledged packets
- Sending rate: ~Window/RTT
- Vary window size to control sending rate

Sizing the Windows

- cwnd (Congestion Windows)
  - How many bytes can be sent without overflowing routers
  - Computed by congestion control algorithm
- AdvertisedWindow
  - How many bytes can be sent without overflowing the sender (flow control)
  - Determined by the receiver
- Sender uses min between the two
  - MaxWindow = min(cwnd, AdvertisedWindow)

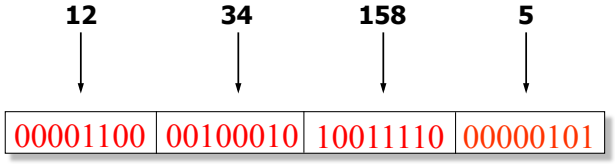
Rate Adjustment

- Basic structure:
  - Upon receipt of ACK (of new data): increase rate
  - Upon detection of loss: decrease rate
- But what increase/decrease functions should we use?
  - Increase window by 1 packet every RTT
  - Decrease window by half if packet loss
  - [Far more in the networking class]

Addresses and Names

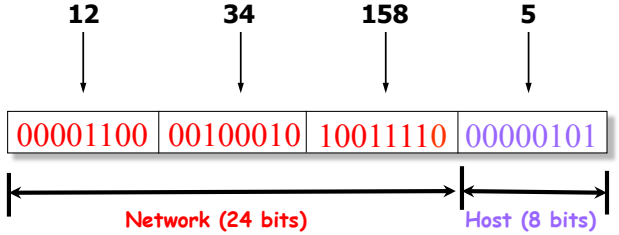
IP Addresses (IPv4)

- A unique 32-bit number
- Identifies an interface (on a host, on a router, …)
- Represented in dotted-quad notation. E.g, 12.34.158.5:



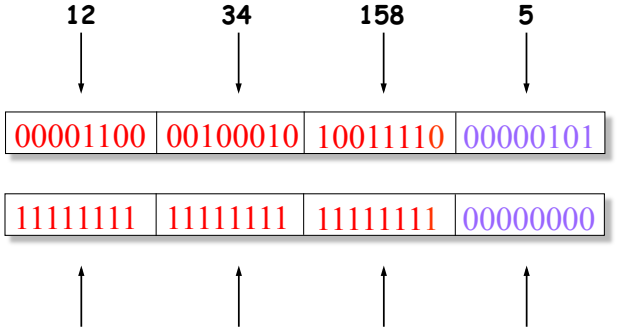
Hierarchical Addressing: IP Prefixes

- Divided into network (left) & host portions (right)
- 12.34.158.0/24 is a 24-bit prefix with 2^9 addresses
  - Terminology: "Slash 24"



IP Address and a 24-bit Subnet Mask

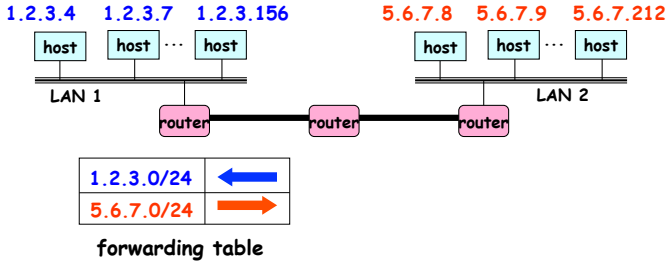
Address



Mask

Hierarchical Addressing Example

- Number related hosts from a common subnet
  - 1.2.3.0/24 on the left LAN (Local Area Network)
  - 5.6.7.0/24 on the right LAN



IP addresses vs. Host Name

- IP addresses
  - Numerical address appreciated by routers
  - Fixed length, binary number
  - Hierarchical, related to host location
  - Examples: 64.236.16.20 and 212.58.224.131
- Host names
  - Mnemonic name appreciated by humans
  - Variable length, full alphabet of characters
  - Provide little (if any) information about location
  - Examples: www.cnn.com and bbc.co.uk

Separating Naming and Addressing

- Names are easier to remember
  - www.cnn.com vs. 64.236.16.20
- Addresses can change underneath
  - Move www.cnn.com to 64.125.91.21
  - E.g., renumbering when changing providers
- Name could map to multiple IP addresses
  - www.cnn.com to multiple (8) replicas of the Web site
  - Enables
    - » Load-balancing
    - » Reducing latency by picking nearby servers
    - » Tailoring content based on requester's location/identity
- Multiple names for the same address
  - E.g., aliases like www.cnn.com and cnn.com

Scalable (Name ↔ Address) Mappings

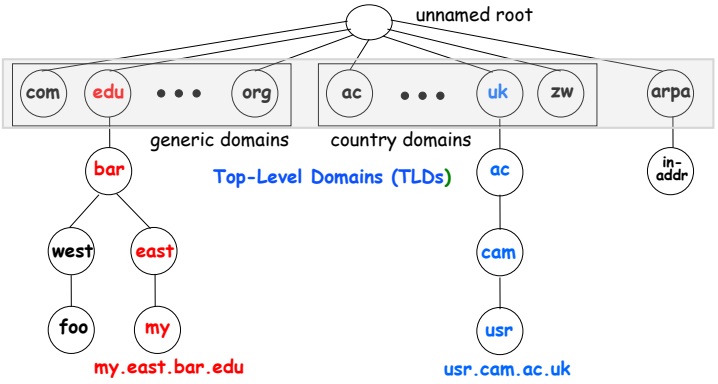
- Originally: per-host file
  - Flat namespace
  - `/etc/hosts` (what is this on your computer today?)
  - SRI (Menlo Park) kept master copy
  - Downloaded regularly
- Single server doesn't scale
  - Traffic implosion (lookups & updates)
  - Single point of failure

Need a distributed, hierarchical collection of servers

Domain Name System (DNS)

- Properties of DNS
  - Hierarchical name space divided into zones
  - Zones distributed over collection of DNS servers
- Hierarchy of DNS servers
  - Root (hardwired into other servers)
  - Top-level domain (TLD) servers
  - Authoritative DNS servers
- Performing the translations
  - Local DNS servers
  - Resolver software

Distributed Hierarchical Database

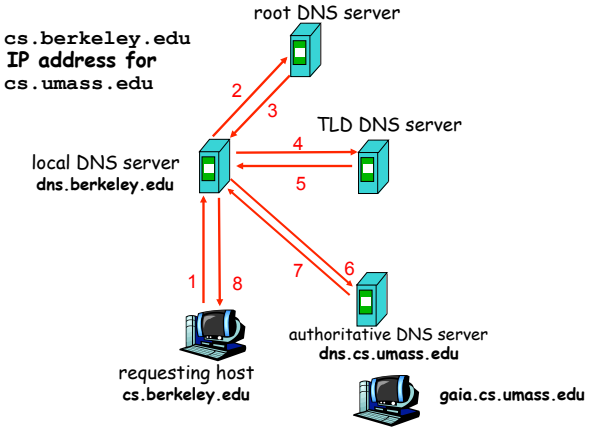


Using DNS

- Local DNS server ("default name server")
  - Usually near the endhosts that use it
  - Local hosts configured with local server (e.g., `/etc/resolv.conf`) or learn server automatically (via DHCP)
- Client application
  - Extract server name (e.g., from the URL)
  - Do *gethostbyname()* to trigger resolver code
- Server application
  - Extract client IP address from connection
  - Optional *gethostbyaddr()* to translate into name

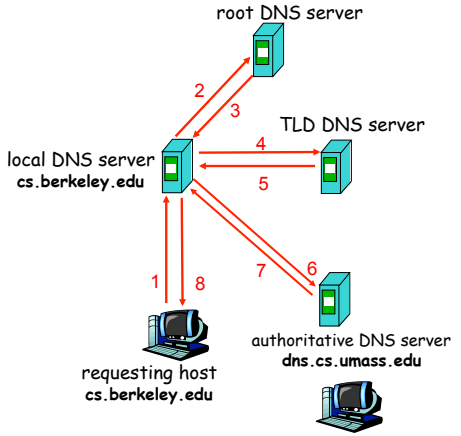
Example

Host at `cs.berkeley.edu` wants IP address for `gaia.cs.umass.edu`



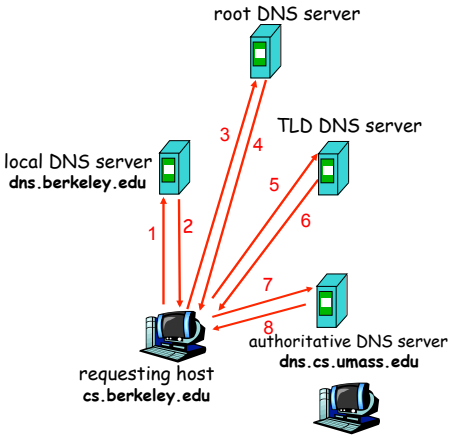
Recursive vs. Iterative Queries

- **Recursive** query
  - Ask server to get answer for you
  - E.g., request 1 and response 8



Recursive vs. Iterative Queries

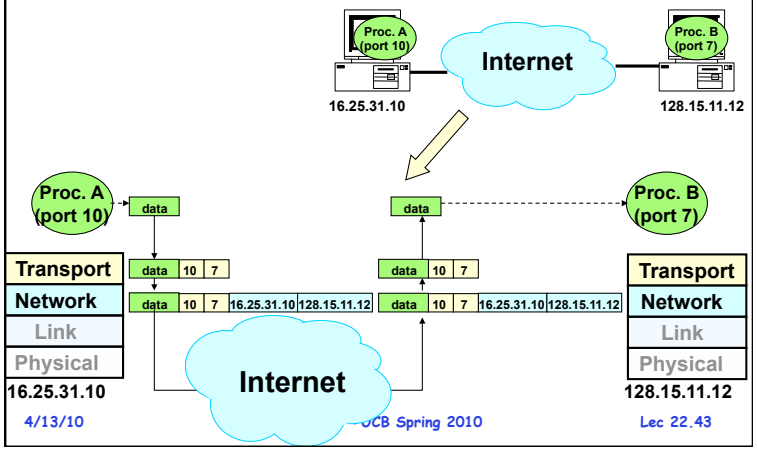
- **Iterative** query
  - Ask server who to ask next
  - E.g., all other request-response pairs



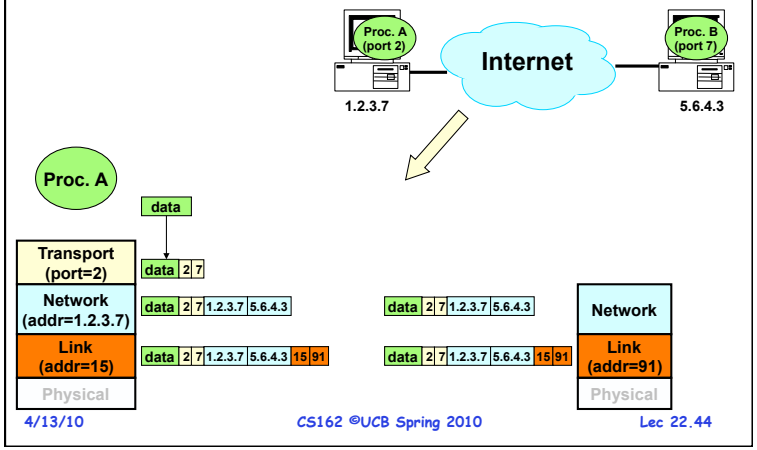
Conclusion

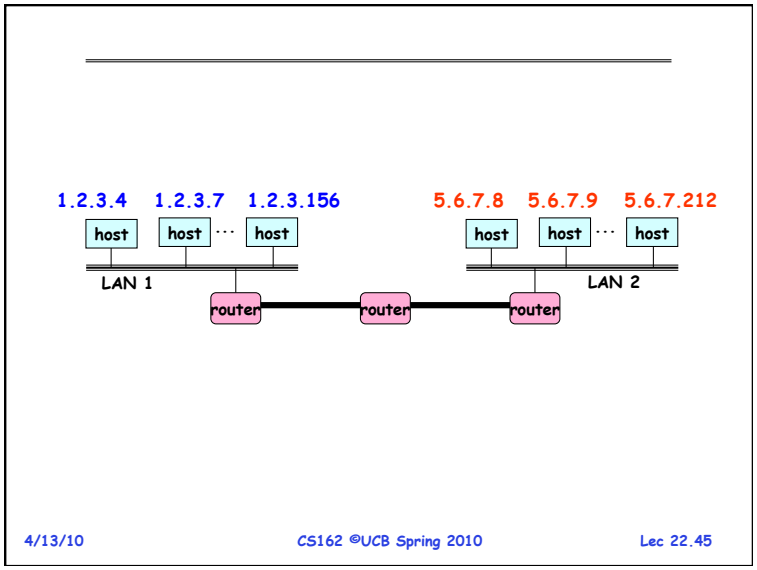
- **Transport layer:**
  - Main service (TCP & UDP): port multiplexing/ demultiplexing
  - Other services (TCP):
    - » reliability
    - » congestion control: avoid overloading the network
    - » Flow control: allow overflowing the receiver
    - » in-order delivery
- IP Addressing
  - 32b (IP v4), quad notation
  - Capture host location
  - Network and host portions
- **DNS:** System for mapping from names⇒IP addresses
  - Hierarchical mapping from authoritative domains
  - Recursive vs. iterative lookup

Putting Everything Together



Putting Everything Together




1.2.3.4
1.2.3.7
1.2.3.156
5.6.7.8
5.6.7.9
5.6.7.212
host
host
host
host
host
host
LAN 1
LAN 2
router
router
router