Lecture N1: Networks and Distributed systems

****

Review -- 1 min

****

RAID

Rethink the sync (guest lecture)

- Performance v. durability
- Example
  - T1 begin
  - W1
  - W2 — barrier — barrier
  - T1 end — Barrier + block
  - T2 begin
  - W3
  - W4 — barrier — barrier
  - T2 end — Barrier + block
  - T3 begin
  - W5
  - W6
  - T3 end — barrier — barrier — Barrier + block
  - Print/send message "done" — Block

Barriers (write scheduler), block (sync)
→ Better performance
→ Better reliability (current disks "cheat" because otherwise performance is too horrible)

****

Outline - 1 min

****

Distributed systems intro
Basic NW communication
-- send/recv packet

-- routing
-- DNS
-- reliability
-- sharing
-- performance
-- RPC

II Distributed systems
3 problems

- performance
- consistency
- reliability
- security

Case study: Distributed file systems

****

Preview - 1 min

****

Today: motivation, basics, file system example, performance
Monday/Wednesday: Reliability:

Network failures:

- Retransmission, idempotent requests

Machine failures

- Careful protocol construction (e.g., ad hoc solutions)
- 2 phase commit
- Reliable asynchronous messaging

if time: security

****

Lecture - 20 min

****

Motivation

Technology trends:

Decade	Tech	$/ machine	Sales Volume	Users/ machine
50's	custom	$10M	100	1000's
60's	mainframe	$1M	10K	100's
70's	mini-computers	$100K	1M	10's
80's	PCs	$10K	100M	1
90's	PCs, portables, PDAs	$1K	1B	1/10
00's	appliances cloud	$0.1K $1K	10B ???	1/100 1/1K-1/10K (bursty)

Centralized v. Distributed systems

Distributed system: physically separate computers working together

Why do we need distributed systems?

- Cheaper to build lots of simple computers
  - Mfg rule of thumb: 2x increase in quantity → 10% reduction in cost per unit
- Easier to add power incrementally (v. design whole new machine)

Promise of distributed systems

- Higher availability – one machine goes down, use another
- Better reliability – store data in multiple locations
- More security – easier to make each (small) piece secure; professional management of system

If we're not careful, reality will be disappointing

- Worse availability – depend on every machine being up

Lamport: "A distributed system is a system where I can't get any work done if a machine I've never heard of crashes."

- Worse reliability – can lose data if any machine crashes
- Worse security – anyone in the world can break into my systems

Key idea: coordination is more difficult b/c can only use network for coordination and because of *partial failures* – part of the system (a connection, a machine) fails while the rest keeps running

Physical reality v. desired abstractions

Desired abstraction: Programming/using distributed system looks like programming/using centralized system

- Location independence
- Performance
- consistency
- Failures, reliability
- security

Location independence – step 1 – how to assemble distributed system…

Message transmission/delivery

From the point of view of operating system, network is just another I/O device

In particular, NIC -- network interface controller on bus

Send/receive messages by DMA or PIO/Memory mapped I/O -- transfer message from memory to NIC or vice versa

[[picture]]

Routing

Routing -- need to get message to right process on right machine

Each machine has an ID (e.g., IP address 128.83.141.37)
A process on a machine can create a port
--> e.g., utcs web server is 128.83.120.139:80

So, task is to get packet from a port on one machine to a port on another machine

Example: RIP routing
(old/simplified version of Internet routing; can be used within an organization; not sufficient across organizations -- security, policy issues; BGP there...)

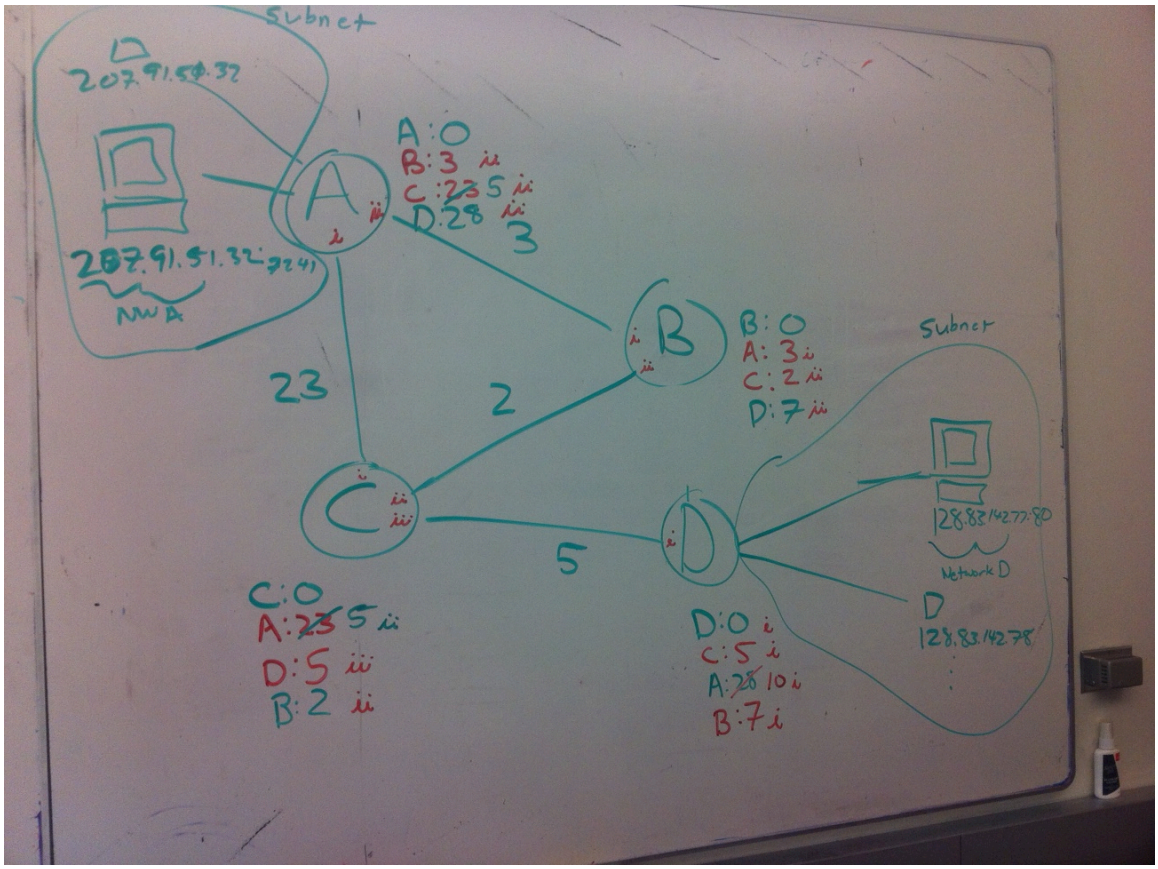
For Internet IP routing, machine IP address is <network><host> --> route to right network, then switch(es) send packet to right hpst on network

(1) Learning routes -- RIP

RIP protocol builds shortest path tables in router e.g., (simplified -- just to get intuition that this all plausibly can be done...)

Distance vector protocol

- each node has a vector – foreach entry: shortest known distance to that destination and corresponding outgoing route
- at each step, send vector to neighbors, receive vectors from neighbors; increment each entry in received vectors by 1; then for each entry, take min (current, neighbor1+1, neighbor2+1, …)
-



(1a) Learning routes – hierarchical

In above – everyone learns about everyone

- Works on a given network
- Won't scale to Internet
  - memory, update bandwidth, …
- Hierarchical version
  - Need way to summarize what's on network
    - HW addresses "random"
    - → add a layer of structured addresses
    - MAC v. IP
  - Do something like above within a network (everyone can learn about everyone's MACs)
  - Router to other networks summarizes IP range 128.83.141.*
  - Use similar principles (but more secure protocol BGP) for distributing/learning IP routes (only routers need to participate)

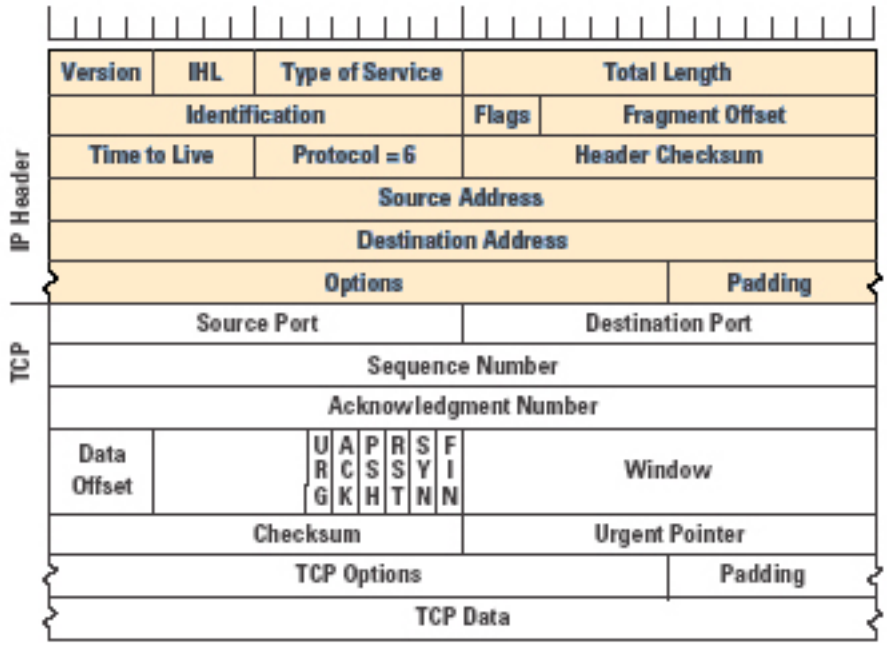
Binding between link layer (MAC) and IP address – ARP

- ARP
  - basic idea broadcast "[MAC] is using [IP addr]" or send "[MAC] wants to know who is using [IP addr]" and receive reply
  - ARP cache

  Notice – IP<-> ARP translation may happen at every hop…

(2) sending packet
-- device driver puts packet in NIC. Needs to specify destination

layered address:
Link header (e.g., Ethernet MAC): source/dest MAC addr [local addr]
IP header: source/destination IP addresses [global addr]
TCP header: source/destintation port numbers



--> OK. So now we can get packets from here to there (and back)

so outbound data path is [cover of Comer's book]

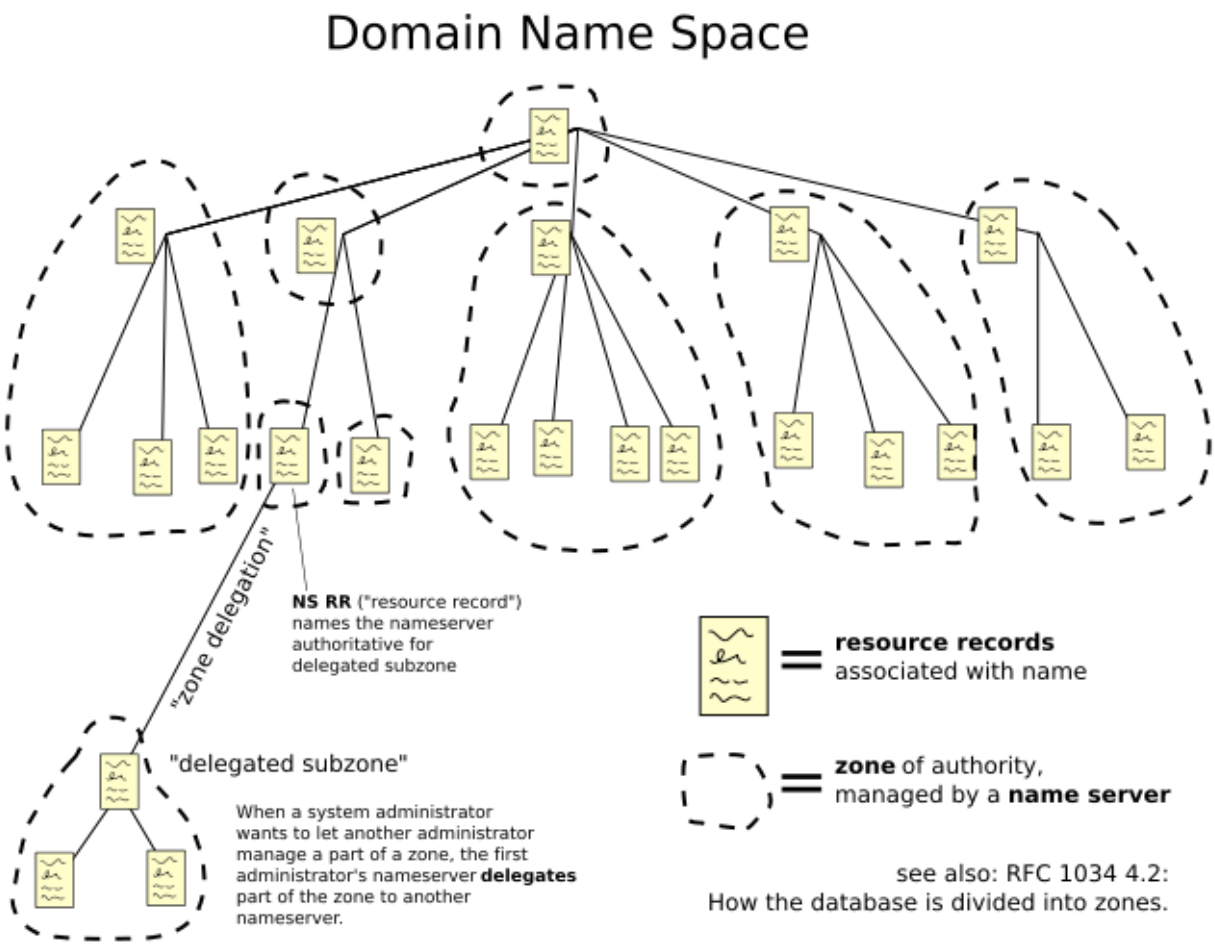
application
TCP queue of sent packets
TCP output <--- TCP timer OR UDP
IP
ARP
DRIVER

DNS

How know IP address of www.cs.utexas.edu

[source: http://en.wikipedia.org/wiki/File:Domain_name_space.svg]

Domain name system (DNS)
-- client knows IP address of DNS server
-- client can ask "give me IP address for <name>"
-- DNS, itself, is a distributed protocol (different servers cooperate to provide service) – Logically, DNS could be a big database at a huge server (once it was centralized!)



Hierarchical:

(1) Divide database into zones
(2) Lookup table for any zone can be at its own name server
(3) Start with root zone, whicih knows name servers for top level domains (.com, .edu, .gov, .fr, …), which know name servers for subdomains (google.com, utexas.edu, …)
    a. Nameserver for each zone generally replicated for reliability, load balancing
    b. Caching and recursive lookup for scalability

Notice –all of this can be done with IP addresses only (so given ability to route anywhere and given ID of my parent name server, I can route packets to anyone…); when I connect my machine to a network, it gets

- an IP address it should use

- an IP address of the gateway router
- (often) an IP address of a name server
- → off to the races

OK. Now I can send to anyone I want…

Message loss

Problem: packets can be lost
-- interference (especially wireless network), overflowing buffers at routers or receivers

example: 2 nodes sending at full speed to 1 node [picture]

Solution 1: Request/reply or receive/acknowledge

Simple solution:
Request/acknowledge protocol
Common case:
1) Sender sends message (msg, msgId) and sets timer
2) Receiver receives message and sends (ack, msgId)
3) Sender receives (ack, msgId) and clears timer

If timer goes off, goto (1)

--> "At least once" semantics -- receiver receives each packet at least once (but maybe multiple times) (assuming neither sender nor receiver crashes or gives up)

+ Simple, good match with request/reply communications patterns
- Low throughput for large requests (1 packet per round trip latency. e.g., 1KB per 10ms --> 100 KB/s)

Solution 2: Pipeline solution 1 -- multiple packets in flight; resend unacked packets after timeout

Optimizations:

(1) cumulative acks -- ack of packet i means that all packets up to i have been received

(1a) Combine acks -- don't send ack for each packet; send for every other packet, etc.

(2) immediate resend on nack -- when receiver recieves packet i, ack i; then receives packet i+2 (missing i+1). Can't ack i+2 (b/c cumulative ack); instead resend ack i; sender receives "ack i; ack i" and knows that i+1 was not received --> resend it immediately

(3) [often bad] Delayed acks -- for bidirectional communication, application layer at receiver will likely send data back to sender; so, don't ack the packet at network level; instead, count the reply as the ack. (In TCP, each data packet I send also carries acks for all that I've received on stream)

(4) [often bad] Nagle's algorithm -- combine small packets to reduce overheads ("as long as there is a sent packet for which sender has not received ack, buffer output until packet is full")
--> Made sense for telnet on modem; probably not useful for real time video game on LAN...
--> HORRIBLE interaction with delayed acks (optimizations were introduced by different groups at about the same time -- early 1980s...)

Sharing the network

Network is shared resource with no global "root/administrator"
How do we keep a malicious user or faulty program from hogging the network

ANSWER: We can't (DDOS attacks)

OK. How do we get normal users and programs from hogging network/how do we divide network resources fairly?

IP level: Overloaded switches drop packets

PICTURE

TCP level: Adaptive send rate
-- start slow
-- if no losses, increase rate
-- if loss, reduce rate
--> Overloaded router causes TCP to slow down

Details
Van Jacobson and Michael Karels "Congestion avoidance and control"
(Classic paper)

" In October of '86, the Internet had the first of what became a series of 'congestion collapses'. During this period, the data throughput from LBL to UC Berkeley (sites separated by 400 yards and two IMP hops) dropped from 32 Kbps to 40 bps."

Problem: congestion -> loss -> timeout -> resend -> more packets -> more congestion

Key idea (solution): Conservation of packets -- when running near capacity, don't put a new packet in until an old packet leaves network

5 fixes to previous TCP to get conservation of packets:

(i) slow-start
-- congestion window -- cnwd -- max # of packets in flight
-- on start or cnwd = 1
-- on ack, increase cwnd by 1
(not so slow -- doubles cnwd on each round trip)

[[aside -- sender or receiver may have max cwnd. This may limit bandwidth for long paths -- if RTT is high, need deep pipeline to fill it.]]

(ii) round-trip-time variance estimation
-- TCP uses resend on timeout
-- Problem: variance rises rapidly with load

-- e.g., at 75% load, round trip times can vary by a factor of 16
-- old timer caused many unnecessary retransmissions under load
--> throwing gasoline on a fire
-- new timer much better

(ii) exponential retransmit timer backoff
-- you provably need this for stability
-- this is why your web browser stalls for 5 seconds then 30 then...
(hint: hit "reload" if page not there in 5 seconds...)

(iv) more aggressive receiver ack policy

(v) dynamic window sizing on congestion
-- additive increase, multiplicative decrease
-- halve cwnd on loss
-- increment by 1/cwnd packets on each successful ack (1 packet per round trip; much slower than "slow start")
-- Turns out you need to back off **really** aggressively to guarantee "don't put more in than you take out"
-- "traffic jam effect" -- easier to get into congestion than to get out of it...

--> reasonably fair sharing

bandwidth = k(B/R sqrt(p))
-- B packet size
-- R round trip time
-- p packet drop probability

--> flows at congested link with same packet size and same round trip time will get same fraction of bandwidth (since they have same drop probability)
--> if different round trip times, then "closer" one can do much better

"TCP Friendly" -- protocols expected to be TCP friendly -- whatever congestion avoidance algorithm they use, it should not send more than k(B/R sqrt(p))

-- easy to write code that is not tcp friendly; don't do this.

(k = 1.2247)

Bonus observation -- small loss rates kill you for long-haul links

Be afraid…

- Internet stability relies on "everyone backs off"
- In fact, Thou Shalt Be TCP-Friendly is a "law" of the IETF
- Bad guy: not back off
  - TCP backs off more
  - → Incentive to be bad; if everyone is bad, we're dead?