Networked Systems

Distributed system

A collection of physically separate computers working together

- cheaper to build
- easier to add capabilities incrementally

Decade	Technology	$/machine	Sales volume	Users/machine
50's	custom	$10M	100	1000's
60's	mainframe	$1M	10K	100's
70's	minicomputers	$100K	1M	10's
80's	PCs	$10K	100M	1
90's	PCs, portables, PDAs	$1K	1B	1/10
00's	appliances	$0.1K	10B	1/100
	cloud	$1K	???	1/1K - 1/10K

Diamonds and Rust

- Higher availability
  - if one machine fails, another can step in
- Better reliability
  - replication
- More security
  - easier to make each smaller piece secure
  - in cloud, professional system management

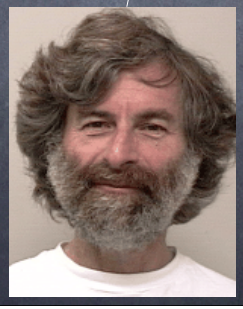
Diamonds and Rust

- Higher availability
  - if one machine fails, another can step in
- Better reliability
  - replication
- More security
  - easier to make each smaller piece secure
  - in cloud, professional system management
- Lower availability
  - may stop if any machine fails

What is a distributed system?

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

Leslie Lamport

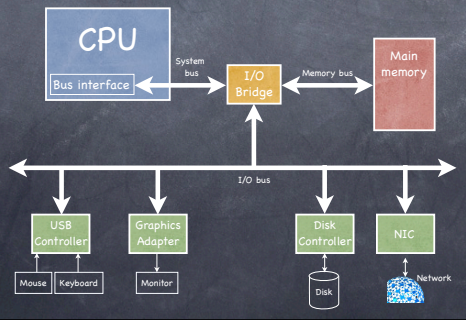


Diamonds and Rust

- Higher availability
  - if one machine fails, another can step in
- Better reliability
  - redundancy through replication
- More security
  - easier to make each smaller piece secure
  - in cloud, professional system management
- Lower availabity
  - may stop if any machine fails
- Worse reliability
  - hard to coordinate replicas
- Less security
  - anyone in the world can break into the system

Message transmission

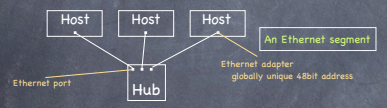
- OS sees network as just another device
  - Network Interface Controller (NIC) added to bus
  - transfer data to/from memory to NIC through DMA or memory mapped I/O



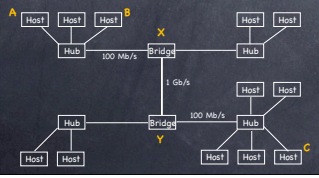
Networks

Ethernet

- Xerox PARC, 1973
  - co-invented by Bob Metcalfe, now at UT!
  - bandwidth: from 3Mb/s to 1 Gb/s

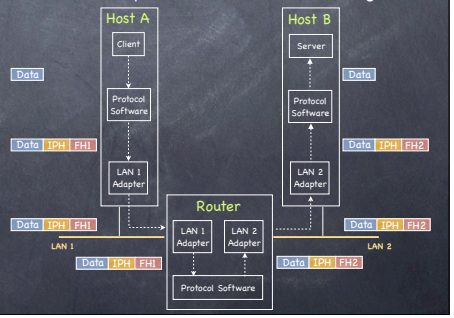


  - hub copies every bit it receives on a port to every other port
  - multiple segments can be connected into a larger LAN (Local Area Network)



Multiple LAN can be connected in a WAN (Wide Area Network)

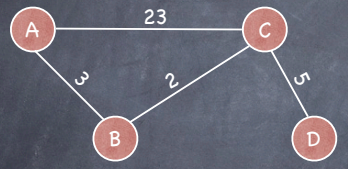
- Encapsulation allows internetworking

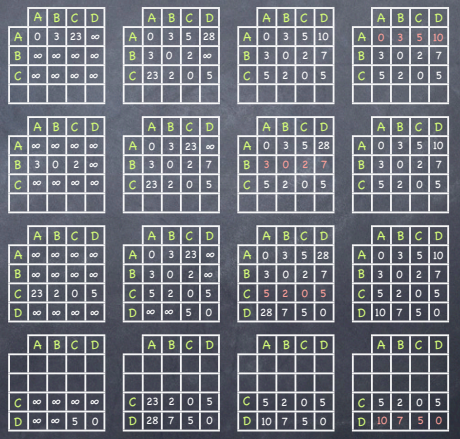


Routing

- Need to send a message to the right process on the right host
- Each host has a unique network ID (e.g. IP address 128.83.120.156)
  - a process can create a port on the host
    - UTCS web server: 128.83.120.39:80
- Sender finds IP address of intended receiver through Domain Name Service (DNS)
  - distributed hierarchical database
  - maps names such as nikon.cs.utexas.edu to 128.83.120.156

Distance Vector Routing

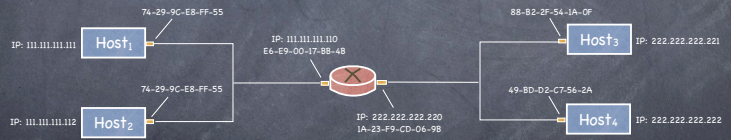




```
1   Initialization
2     for all destinations y in N:
3       Dx(y) = c(x,y) /* ∞ if not neighbor */
4     for all neighbor w
5       Dw(y) = ∞ for all destinations y in N
6     for all neighbor w
7       send distance vector Dx = [Dx(y) ; y in N] to w
8
9   loop
10    wait (until link cost changes for some w or
11          received a distance vector from some w)
12
13    for all y in N:
14      Dx(y) = minv{c(x,v) + Dv(y)}
15
16    if Dx(y) changed for any destination y
17      for all neighbor w
18        send distance vector Dx = [Dx(y) ; y in N] to w
19
20  forever
```

Address Resolution Protocol

- Each adapter has a unique link layer address (MAC)
  - 6 bytes, burned in ROM
  - ARP (Address Resolution Protocol) maps between link and network address (MAC and IP)



What if Host 1 want to send a message to Host 4?

1) Host1 contacts DNS to get IP address of Host4
2) adds TCP header
   - specifies port
3) adds IP header
4) contacts DNS to learn router's IP address
5) uses ARP to get router's MAC address and adds header

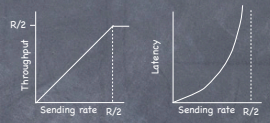
1) Router strips MAC header
2) checks IP header to determine interface on which to forward
   - 222.222.222.220
3) uses ARP to find MAC of Host4
4) adds header and forwards

Message Loss

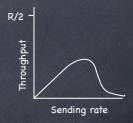
- Buffers overflow, interference, etc
- Solution 1: use acks
  a) sender sends (msg, msgId) and sets timer
  b) receiver receives message and sends (ack, msgId)
  c) sender receives (ack, msgId) and clears timer
     - if timer goes off, go to a)
  - "at least once" semantics
- Low throughput
  - 1 packet/roundtrip latency
    - 1KB per 10ms = 100 KB/s
- Solution2: pipeline Solution 1
  - multiple packets in flight
  - resend unacked packets after timeout
- Optimizations
  - cumulative acks
    - ack i acks packets 1 to i
    - immediate resend on nack (or repeated acks to previously acked message)
  - delayed acks
    - for bidirectional communication, use application response as implicit ack
  - 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
    - good for telnet, bad for real-time applications

Congestion: causes and costs

- Two senders, a router with infinite buffer
  - senders send at same rate
  - linK capacity R

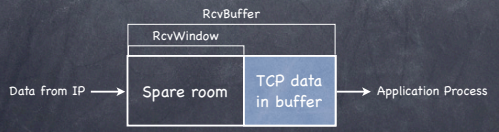


- Two senders and a router, with finite buffer
  - needed retransmissions due to overflow reduce effective throughput
  - unneeded retransmissions caused by large delays reduce effective throughput
- Multiple senders, multihop paths
  - dropped packets waist resources used to forward them to the place they are dropped



TCP Flow Control

- Prevents sender from overflowing receiver's buffer
- Sender maintains
  - RcvWindow - estimate of buffer space at receiver
  - LastByteSent, LastByteAcked
- Receiver maintains
  - RcvBuffer
  - LastByteRead, LastByteRcvd
- To avoid overflowing
  - LastByteRcvd-LastByteRead ≤ RcvBuffer
- Hence, sender ensures
  - LastByteSent - LastByteAcked ≤ RcvWindow = RcvBuffer - (LastByteRcvd - LastByteRead)



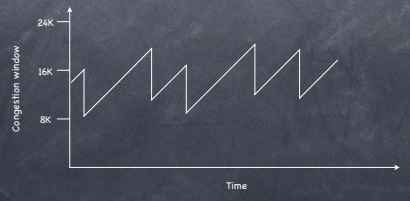
TCP Congestion Control

- Both sides of a connection keep track of CongWin
  - LastByteRcvd-LastByteRead ≤ min{ CongWin, RcvBuffer}
    - send rate: CongWin/RTT (assuming negligible retransmission and loss)
- If acks are received regularly, CongWin grows
- If ack loss is detected, CongWin shrinks
  - ack loss detected as
    - timeout
    - receipt of 3 duplicate acks
- Congestion Control algorithm has five major components
  - additive increase, multiplicative decrease
  - slow start
  - reaction to timeout events
  - round trip variance estimation
  - exponential retransmit timer backoff

Additive increase, multiplicative decrease

- Multiplicative decrease
  - half CongWin after loss event* (until one reaches 1 MSS)
- Additive increase
  - increase CongWin by 1 MSS every roundtrip
    - on each received ack, increase by MSS x (MSS/CongWin)

$$\frac{1}{\text{segments in flight}}$$



Slow Start

- At start, CongWin set to 1 MSS
- Too slow to grow linearly
  - during slow start phase, exponential growth...
    - CongWin grows by 1 MSS for each received ack
  - until first loss event occurs

Reaction to time out events

- TCP actually treats loss events differently
  - on receipt of three duplicate acks
    - multiplicative decrease, additive increase
  - on a timeout
    - drop to 1 MSS
    - slow start mode until CongWin reaches a threshold (typically half of CongWin before loss)
    - additive increase after threshold reached
- Why the difference?
  - old versions of TCP (TCP Tahoe) resorted to slow start also on receipt of duplicate acks, but...
  - ...receipt of acks, even if duplicate, shows that some packets get to destination
    - fast recovery eliminates slow start phase (TCP Reno)
- Recent proposals (TCP Vegas)
  - detect congestion in the router before packet loss occurs
  - lower rate linearly when imminent packet loss is detected
    - longer RTT indicates greater congestion

RTT variance and retransmit backoff

- Original TCP set timeout to twice the estimated RTT
  - but with high load (above 75%) RTT can vary by 16X
    - lots of unnecessary sends under load (great...)
- Current versions set timeout to
  - estimated RTT + 4 x MeanDev(RTT)
- Retransmit backoff is exponential
  - provably needed for stability
  - this is why web browser stalls for 5 sec, then for 10 then...
    - hint: hit reload if page no there after 5 sec

TCP friendly rate control

- Measure loss probability L and RTT
- Use them as parameters to model TCP throughput

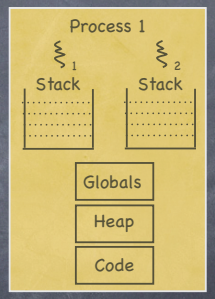
$$\text{Average throughput of a connection} = \frac{1.22 \times \text{MSS}}{\text{RTT} \times \sqrt{L}}$$

- TCP friendly protocols use a congestion control mechanism that consumes no more bandwidth than

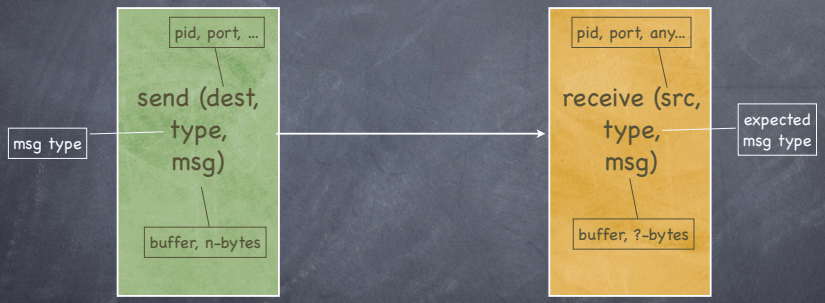
$$\frac{1.22 \times \text{MSS}}{\text{RTT} \times \sqrt{L}}$$

Process coordination: two fundamental approaches

- Communication and synchronization based on...
  - shared memory
    - assume processes/threads can read & write a set of shared memory locations
    - implicit inter-process communication
    - explicit synchronization
    - difficult to provide across machine boundaries
  - message passing
    - explicit inter process communication
    - implicit synchronization



Send and Receive Primitives



- Many ways to design the message passing API

Semantics of message passing

send(receiver,message)

Synchronization

Naming	Blocking	Non-blocking
Explicit (single)	Send message to receiver Wait until message is accepted	Send message to receiver
Implicit (group)	Broadcast message to all receivers. Wait until message is accepted by all	Broadcast message to all receivers

Semantics of message passing

receive(sender,message)

Synchronization

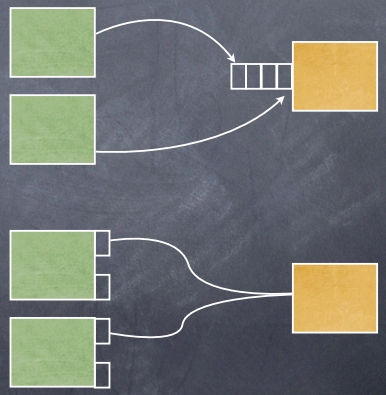
Naming	Blocking	Non-blocking
Explicit (single)	Wait for a message from sender	If there is a message from sender then receive it, else continue
Implicit (group)	Wait for a message from any sender	If there is a message from any sender then receive it, else continue

Buffering messages

- No buffering
  - sender must wait until receiver receives message
  - rendezvous on each message
- Bounded buffer
  - finite size
  - sender blocks on buffer full
- Unbounded buffer
  - "infinite" size
  - sender never blocks

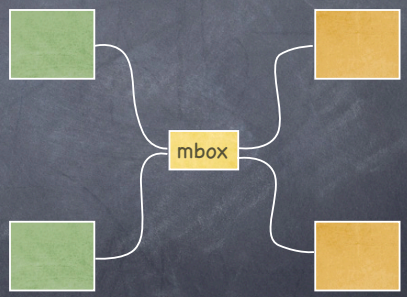
Direct Communication

- A single buffer at the receiver
  - multiple processes may send messages to the receiver
  - receiving from a specific sender requires searching entire buffer
- A buffer at each sender
  - sender may send messages to multiple receivers
  - receiving a message requires searching through whole buffer



Indirect Communication

- Mailbox abstraction
  - many-to-many communication
  - requires open/close of mailbox
- Buffering
  - buffer, mutex and condition variables at the mailbox



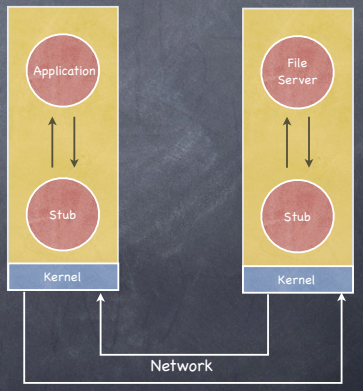
Limitations of message passing

- Easy for OS, hard for programmer
  - programmer must code synchronization
  - programmer may have to code format conversions, flow control, error control
  - no dynamic resource discovery

Remote Procedure Call (RPC)

Birrell & Nelson, 1984

- RPC mechanism
  - hides message passing I/O from programmer
  - (almost) a procedure call–but on the server
- RPC invocation
  - calling process (client) is suspended
  - procedure parameters passed across network to called process (server)
  - return parameters send back across network
  - calling process resumes



More about RPC

- Similarities between procedure call and RPC
  - paramenters ⟷ request message
  - result ⟷ reply message
  - name of procedure ⟷ passed in request message
  - return address ⟷ mailbox of client
- Implementation issues
  - stub generation
    - can be automated
    - requires signature of the procedure

      types of arguments and return value
  - binding (how client locates a server)
    - static (fixed at compile time)
    - dynamic (at run time, with the help of a name service)
      - if server fails, automatic fail over

Problems with RPC

- Achieves location transparency, except for
  - performance
    - cost of procedure call << same machine RPC << network RPC
  - failures (message loss, machine crash)
    - three execution semantics
      - "at least once" (SUN RPC) — implemented by retransmission on timeout; works only if operation is idempotent
      - "at most once" (Java RMI) — implemented by server filtering duplicate requests
      - "exactly once" — impossible

      if succeeds: at least one execution
      if fails: none, partial, multiple

      if succeeds: exactly once
      if fails: none, partial, one

      if succeeds: exactly once
      if fails: none,
  - consistency/replication
    - can't automatically update an object replicated at multiple machines
      - what would I need to do so?
  - security