CS162
Operating Systems and Systems Programming
Lecture 22

Distributed Decision Making (Finished), TCP/IP Networking, RPC

April 21st, 2020

Prof. John Kubiatowicz

http://cs162.eecs.Berkeley.edu

Recall: Distributed Consensus Making

- Consensus problem
  - All nodes propose a value
  - Some nodes might crash and stop responding
  - Eventually, all remaining nodes decide on the same value from set of proposed values
- Distributed Decision Making
  - Choose between "true" and "false"
  - Or Choose between "commit" and "abort"
- Equally important (but often forgotten!): make it durable!
  - How do we make sure that decisions cannot be forgotten?
    - » This is the "D" of "ACID" in a regular database
  - In a global-scale system?
    - » What about erasure coding or massive replication?
    - » Like BlockChain applications!

Recall: Two-Phase Commit

- Since we can't solve the General's Paradox (i.e. simultaneous action), let's solve a related problem
  - Distributed transaction: Two machines agree to do something, or not do it, atomically
- Two-Phase Commit protocol:
  - Prepare Phase:
    - » The global coordinator requests that all participants will promise to commit or rollback the transaction
    - » Participants record promise in log, then acknowledge
    - » If anyone votes to abort, coordinator writes `"Abort"` in its log and tells everyone to abort; each records `"Abort"` in log
  - Commit Phase:
    - » After all participants respond that they are prepared, then the coordinator writes `"Commit"` to its log
    - » Then asks all nodes to commit; they respond with ack
    - » After receive acks, coordinator writes `"Got Commit"` to log
- Persistent stable log on each machine:
  - Help nodes remember what they have said that they would do
    - » If a machine crashes, when it wakes up it first checks its log to recover state of world at time of crash
    - » Log can be used to complete this process such that all machines either commit or don't commit

Two-Phase Commit: Setup

- One machine *(coordinator)* initiates the protocol
- It asks *every* machine to **vote** on transaction
- Two possible votes:
  - **Commit**
  - **Abort**
- Commit transaction only if unanimous approval

Two-Phase Commit: Preparing

Agree to Commit
- Machine has **guaranteed** that it will accept transaction
- Must be **recorded in log** so machine will remember this decision if it fails and restarts

Agree to Abort
- Machine has **guaranteed** that it will **never accept** this transaction
- Must be **recorded in log** so machine will remember this decision if it fails and restarts

Two-Phase Commit: Finishing

Commit Transaction
- Coordinator learns *all machines have agreed to commit*
- Record decision to commit in local log
- Apply transaction, inform voters

Abort Transaction
- Coordinator learns *at least on machine has voted to abort*
- Record decision to abort in local log
- Do not apply transaction, inform voters

Two-Phase Commit: Finishing

Commit Transaction
- Coordinator learns *all machines have agreed to commit*
- Record decision to commit in local log
- Apply transaction, inform voters

Abort Transaction
- Coordinator learns *at least on machine has voted to abort*
- Record decision to abort in local log
- Do not apply transaction, inform voters

Because no machine can take back its decision, exactly one of these will happen

Detailed Algorithm

Coordinator Algorithm | **Worker Algorithm**

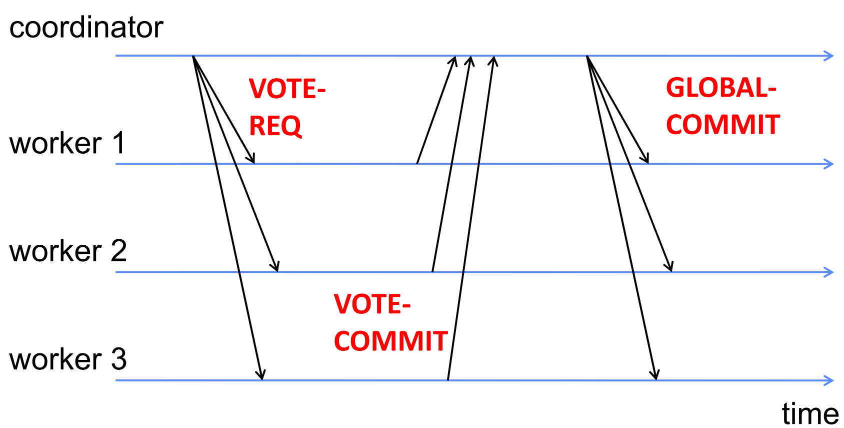
Coordinator sends **VOTE-REQ** to all workers

- Wait for **VOTE-REQ** from coordinator
- If ready, send **VOTE-COMMIT** to coordinator
- If not ready, send **VOTE-ABORT** to coordinator
  - And immediately abort

- If receive **VOTE-COMMIT** from all N workers, send **GLOBAL-COMMIT** to all workers
- If don't receive **VOTE-COMMIT** from all N workers, send **GLOBAL-ABORT** to all workers

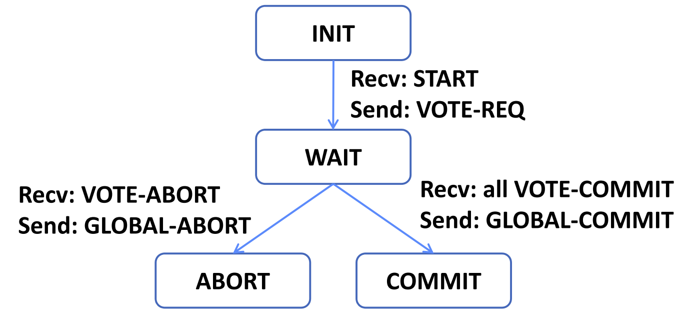
- If receive **GLOBAL-COMMIT** then commit
- If receive **GLOBAL-ABORT** then abort

Failure Free Example Execution

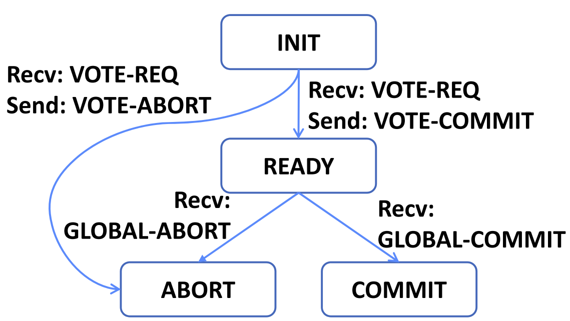


State Machine of Coordinator

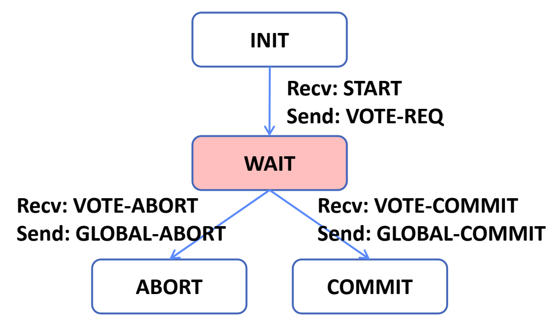
- Coordinator implements simple state machine:



State Machine of Workers

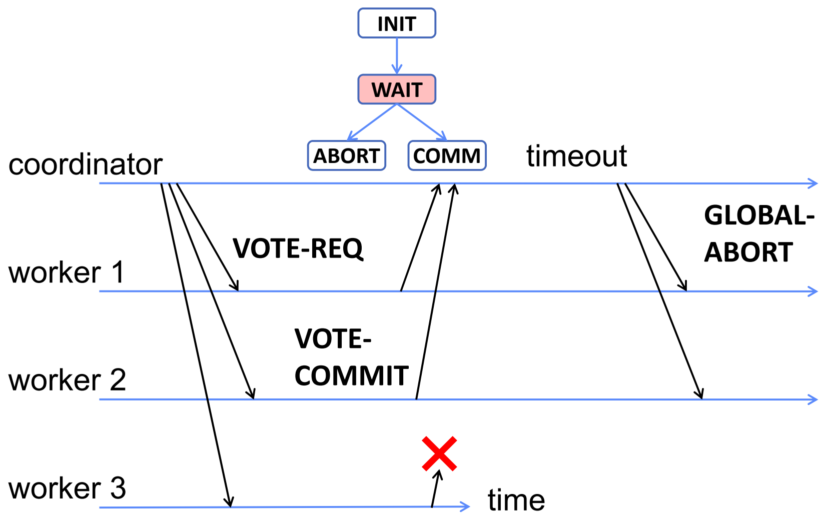


Dealing with Worker Failures

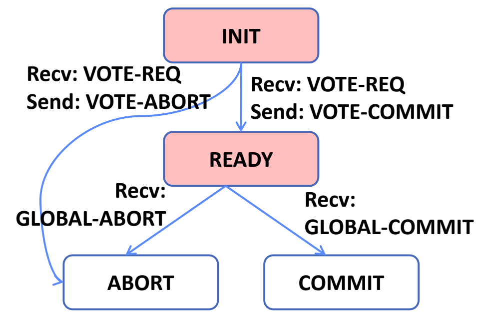


- Failure only affects states in which the coordinator is waiting for messages
- Coordinator only waits for votes in "WAIT" state
- In WAIT, if doesn't receive N votes, it times out and sends GLOBAL-ABORT

Example of Worker Failure

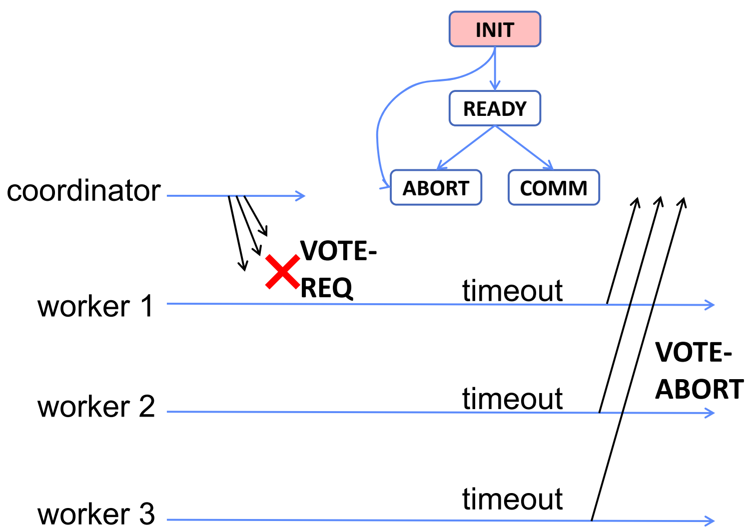


Dealing with Coordinator Failure

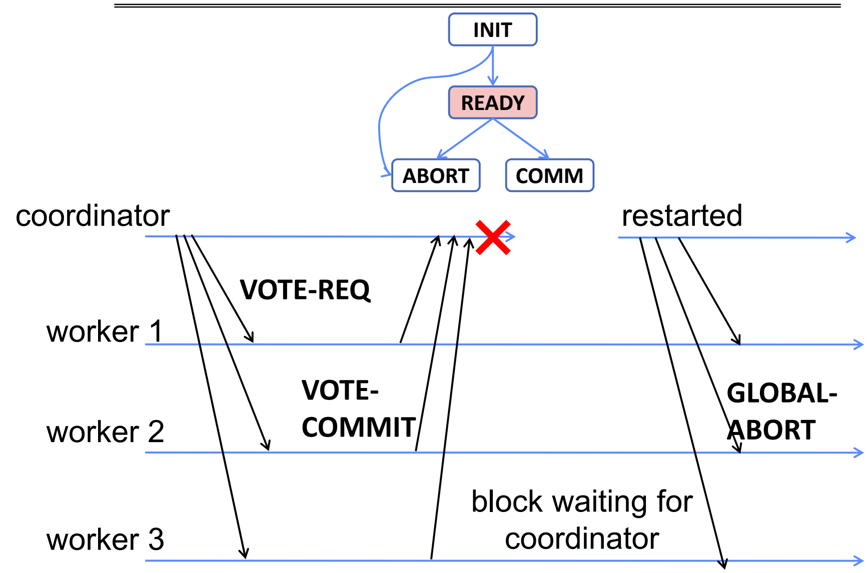


- Worker waits for VOTE-REQ in INIT
  - Worker can time out and abort (coordinator handles it)
- Worker waits for GLOBAL-* message in READY
  - If coordinator fails, workers must **BLOCK** waiting for coordinator to recover and send GLOBAL_* message

Example of Coordinator Failure #1



Example of Coordinator Failure #2

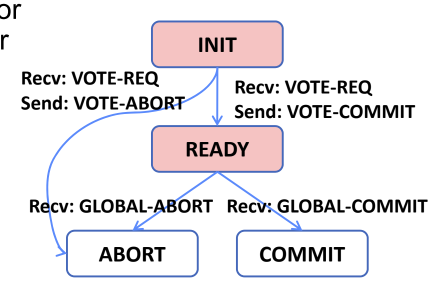


Durability

- All nodes use stable storage to store current state
  - stable storage is non-volatile storage (e.g. backed by disk) that guarantees atomic writes.
  - E.g.: SSD, NVRAM
- Upon recovery, nodes can restore state and resume:
  - Coordinator aborts in INIT, WAIT, or ABORT
  - Coordinator commits in COMMIT
  - Worker aborts in INIT, ABORT
  - Worker commits in COMMIT
  - Worker "asks" Coordinator in READY

Blocking for Coordinator to Recover

- A worker waiting for global decision can ask fellow workers about their state
  - If another worker is in ABORT or COMMIT state then coordinator must have sent GLOBAL-*
    - » Thus, worker can safely abort or commit, respectively
  - If another worker is still in INIT state then both workers can decide to abort



  - If all workers are in ready, need to BLOCK (don't know if coordinator wanted to abort or commit)

Distributed Decision Making Discussion (1/2)

- Why is distributed decision making desirable?
  - Fault Tolerance!
  - A group of machines can come to a decision even if one or more of them fail during the process
    - » Simple failure mode called "failstop" (different modes later)
  - After decision made, result recorded in multiple places
- Why is 2PC not subject to the General's paradox?
  - Because 2PC is about *all nodes eventually coming to the same decision – not necessarily at the same time!*
  - Allowing us to reboot and continue allows time for collecting and collating decisions

Distributed Decision Making Discussion (2/2)

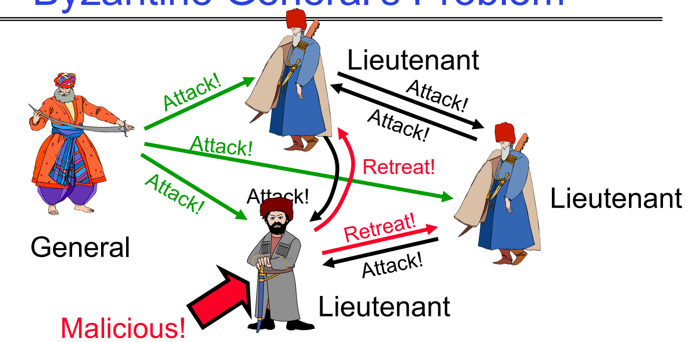
- Undesirable feature of Two-Phase Commit: Blocking
  - One machine can be stalled until another site recovers:
    - » Site B writes "prepared to commit" record to its log, sends a "yes" vote to the coordinator (site A) and crashes
    - » Site A crashes
    - » Site B wakes up, check its log, and realizes that it has voted "yes" on the update. It sends a message to site A asking what happened. At this point, B cannot decide to abort, because update may have committed
    - » B is blocked until A comes back
  - A blocked site holds resources (locks on updated items, pages pinned in memory, etc) until learns fate of update

Alternatives to 2PC

- Three-Phase Commit: One more phase, allows nodes to fail or block and still make progress.
- PAXOS: An alternative used by Google and others that does not have 2PC blocking problem
  - Develop by Leslie Lamport (Turing Award Winner)
  - No fixed leader, can choose new leader on fly, deal with failure
  - Some think this is extremely complex!
- RAFT: PAXOS alternative from John Osterhout (Stanford)
  - Simpler to describe complete protocol

- What happens if one or more of the nodes is malicious?
  - Malicious: attempting to compromise the decision making

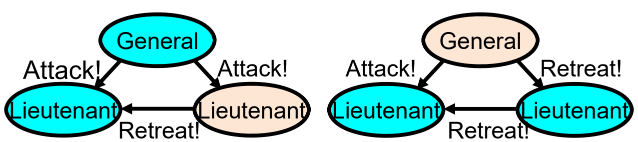
Byzantine General's Problem



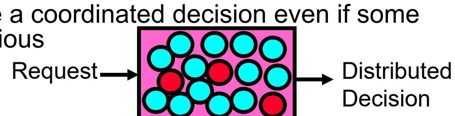
- Byazantine General's Problem (n players):
  - One General and n-1 Lieutenants
  - Some number of these (f) can be insane or malicious
- The commanding general must send an order to his n-1 lieutenants such that the following Integrity Constraints apply:
  - IC1: All loyal lieutenants obey the same order
  - IC2: If the commanding general is loyal, then all loyal lieutenants obey the order he sends

Byzantine General's Problem (con't)

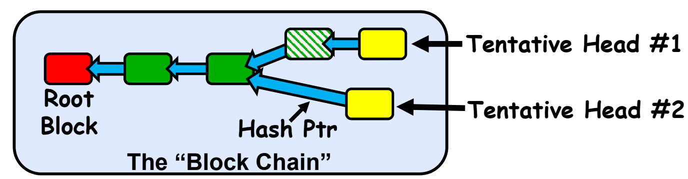
- Impossibility Results:
  - Cannot solve Byzantine General's Problem with n=3 because one malicious player can mess up things



  - With f faults, need n > 3f to solve problem
- Various algorithms exist to solve problem
  - Original algorithm has #messages exponential in n
  - Newer algorithms have message complexity O(n2)
    - » One from MIT, for instance (Castro and Liskov, 1999)
- Use of BFT (Byzantine Fault Tolerance) algorithm
  - Allow multiple machines to make a coordinated decision even if some subset of them (< n/3) are malicious

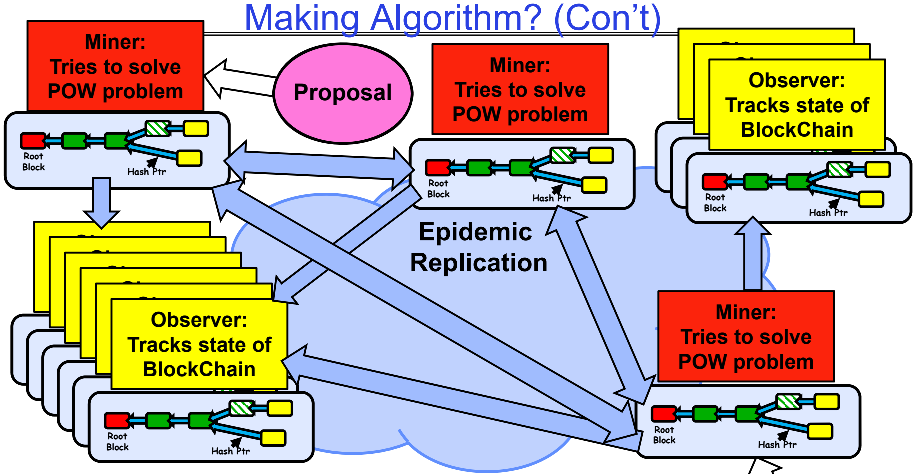


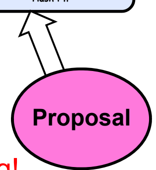
Is a BlockChain a Distributed Decision Making Algorithm?



- BlockChain: a chain of blocks connected by hashes to root block
  - The Hash Pointers are unforgeable (assumption)
  - The Chain has no branches except perhaps for heads
  - Blocks are considered "authentic" part of chain when they have authenticity info in them
- How is the head chosen?
  - Some consensus algorithm
  - In many BlockChain algorithms (e.g. BitCoin, Ethereum), the head is chosen by solving hard problem
    - » This is the job of "miners" who try to find "nonce" info that makes hash over block have specified number of zero bits in it
    - » The result is a "Proof of Work" (POW)
    - » Selected blocks above (green) have POW in them and can be included in chains
  - Longest chain wins

Is a Blockchain a Distributed Decision Making Algorithm? (Con't)

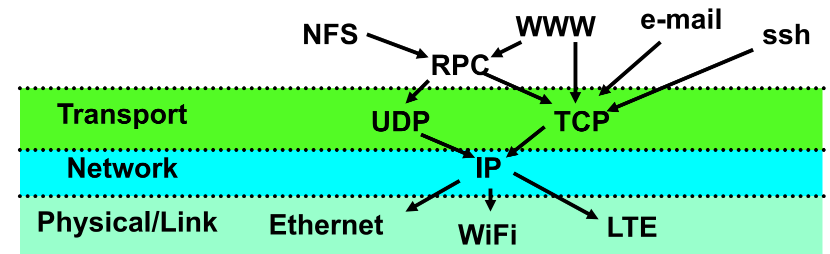




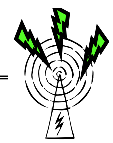
- Decision means: Proposal is locked into BlockChain
  - Could be Commit/Abort decision
  - Could be Choice of Value, State Transition, ….
- NAK: Didn't make it into the block chain (must retry!)
- Anyone in world can verify the result of decision making!

Network Protocols

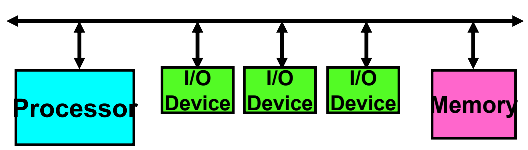
- Networking protocols: many levels
  - Physical level: mechanical and electrical network (e.g., how are 0 and 1 represented)
  - Link level: packet formats/error control (for instance, the CSMA/CD protocol)
  - Network level: network routing, addressing
  - Transport Level: reliable message delivery
- Protocols on today's Internet:



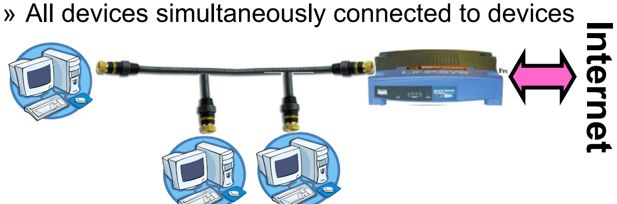
Broadcast Networks



- Broadcast Network: Shared Communication Medium

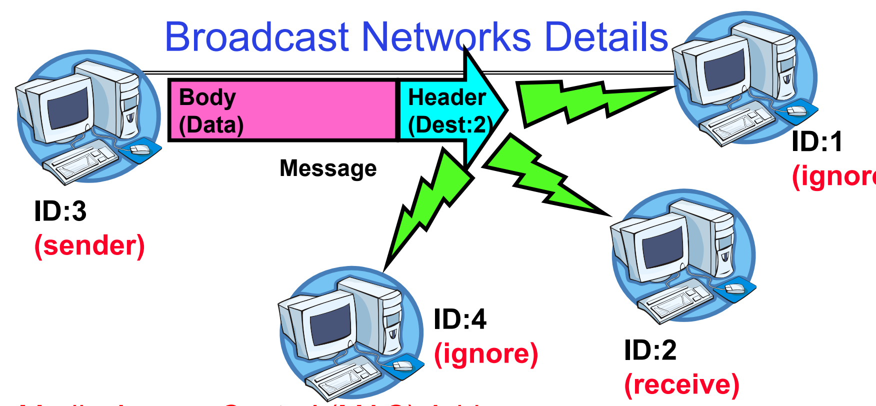


  - Shared Medium can be a set of wires
    - » Inside a computer, this is called a bus
    - » All devices simultaneously connected to devices



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

Broadcast Networks Details



- Media Access Control (MAC) Address:
  - 48-bit physical address for hardware interface
  - Every device (in the world!?) has a unique address
- Delivery: When you broadcast a packet, how does a receiver know who it is for? (packet goes to everyone!)
  - Put header on front of packet: [Destination MAC Addr | Packet]
  - Everyone gets packet, discards if not the target
  - In Ethernet, this check is done in hardware
    - » No OS interrupt if not for particular destination

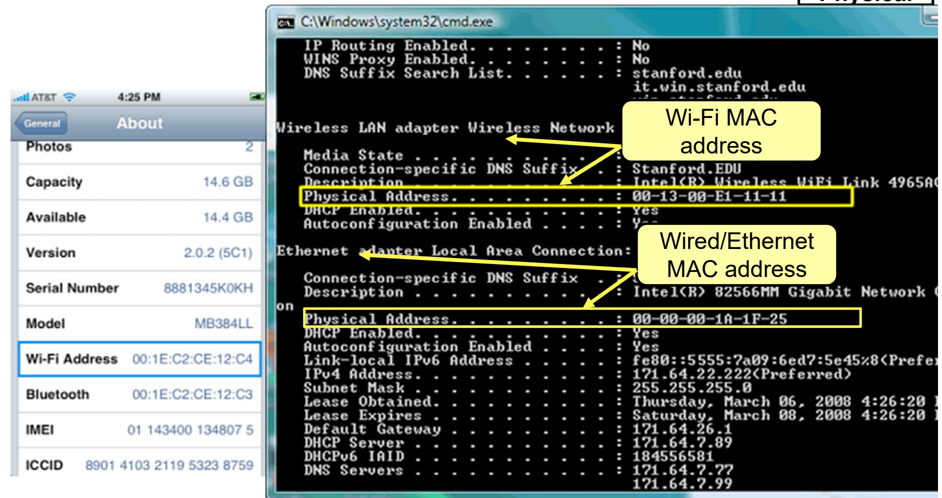
Carrier Sense, Multiple Access/Collision Detection

- Ethernet (early 80's): first practical local area network
  - It is the most common LAN for UNIX, PC, and Mac
  - Use wire instead of radio, but still broadcast medium
- Key advance was in arbitration called CSMA/CD: Carrier sense, multiple access/collision detection
  - Carrier Sense: don't send unless idle
    - » Don't mess up communications already in process
  - Collision Detect: sender checks if packet trampled.
    - » If so, abort, wait, and retry.
  - Backoff Scheme: Choose wait time before trying again
- How long to wait after trying to send and failing?
  - What if everyone waits the same length of time? Then, they all collide again at some time!
  - Must find way to break up shared behavior with nothing more than shared communication channel
- Adaptive randomized waiting strategy:
  - Adaptive and Random: First time, pick random wait time with some initial mean. If collide again, pick random value from bigger mean wait time. Etc.
  - Randomness is important to decouple colliding senders
  - Scheme figures out how many people are trying to send!

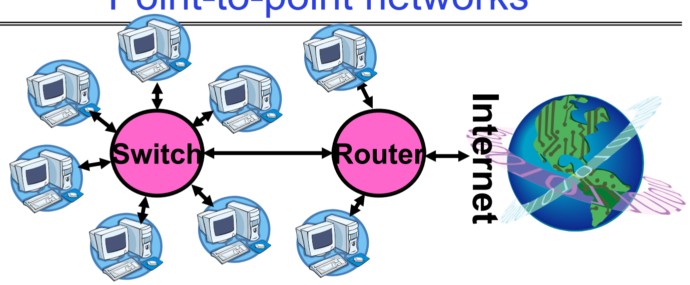
MAC Address: Unique Physical Address of Interface

Application
Present.
Session
Transport
Network
Datalink
Physical

- Can easily find MAC addr. on your machine/device:
  - E.g., ifconfig (Linux, Mac OS X), ipconfig (Windows)



Point-to-point networks

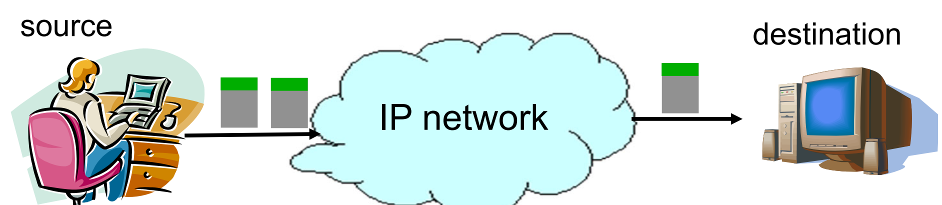


- Why have a shared bus at all? Why not simplify and only have point-to-point links + routers/switches?
  - Originally wasn't cost-effective, now hardware is cheap!
- Point-to-point network: a network in which every physical wire is connected to only two computers
- Switch: a bridge that transforms a shared-bus (broadcast) configuration into a point-to-point network
  - Adaptively figures out which ports have which MAC addresses
- Router: a device that acts as a junction between physical networks to transfer data packets among them
  - Routes between switching domains using (for instance) IP addresses

The Internet Protocol (IP)

Application
Present.
Session
Transport
Network
Datalink
Physical

- Internet Protocol: Internet's network layer
- Service it provides: "Best-Effort" Packet Delivery
  - Tries it's "best" to deliver packet to its destination
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order
- IP Is a Datagram service!
  - Routes across many physical switching domains (subnets)



IPv4 Address Space

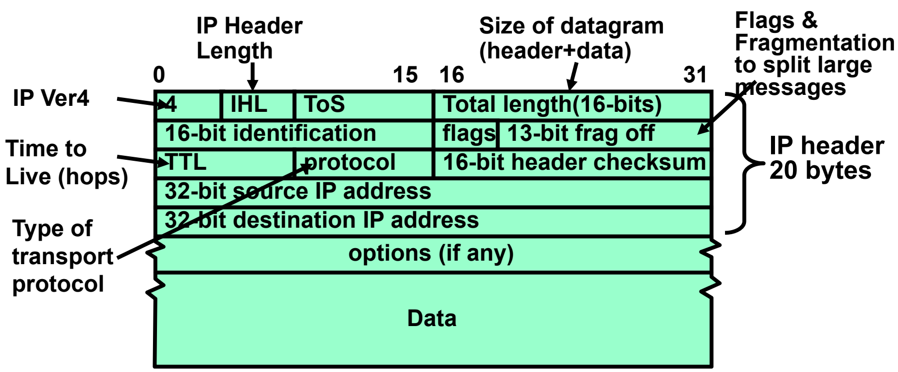
- IP Address: a 32-bit integer used as destination of IP packet
  - Often written as four dot-separated integers, with each integer from 0—255 (thus representing 8x4=32 bits)
  - Example CS file server is: 169.229.60.83 ≡ 0xA9E53C53
- Internet Host: a computer connected to the Internet
  - Host has one or more IP addresses used for routing
    - » Some of these may be private and unavailable for routing
  - Not every computer has a unique IP address
    - » Groups of machines may share a single IP address
    - » In this case, machines have private addresses behind a "Network Address Translation" (NAT) gateway
- Subnet: network connecting hosts with related IP addresses
  - A subnet is identified by 32-bit value, with the bits which differ set to zero, followed by a slash and a mask
    - » Example: 128.32.131.0/24 designates a subnet in which all the addresses look like 128.32.131.XX
    - » Same subnet: 128.32.131.0/255.255.255.0
  - Mask: The number of matching prefix bits
    - » Expressed as a single value (e.g., 24) or a set of ones in a 32-bit value (e.g., 255.255.255.0)
  - Often routing *within* subnet is by MAC address (smart switches)

Address Ranges in IPv4

- IP address space divided into prefix-delimited ranges:
  - Class A: NN.0.0.0/8
    - » NN is 1–126 (126 of these networks)
    - » 16,777,214 IP addresses per network
    - » 10.xx.yy.zz is private
    - » 127.xx.yy.zz is loopback
  - Class B: NN.MM.0.0/16
    - » NN is 128–191, MM is 0-255 (16,384 of these networks)
    - » 65,534 IP addresses per network
    - » 172.[16-31].xx.yy are private
  - Class C: NN.MM.LL.0/24
    - » NN is 192–223, MM and LL 0-255 (2,097,151 of these networks)
    - » 254 IP addresses per networks
    - » 192.168.xx.yy are private
- Address ranges are often owned by organizations
  - Can be further divided into subnets

IPv4 Packet Format

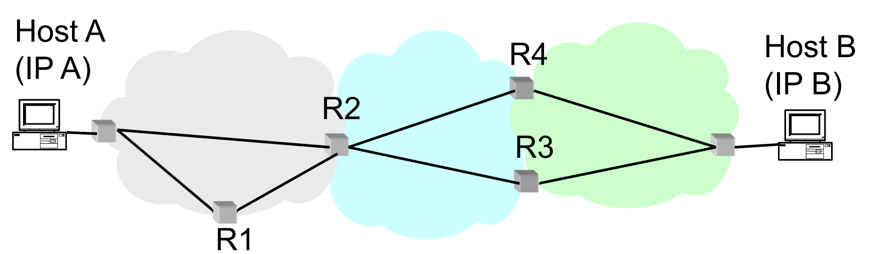
- IP Packet Format:



- IP Datagram: an unreliable, unordered, packet sent from source to destination
  - Function of network – deliver datagrams!

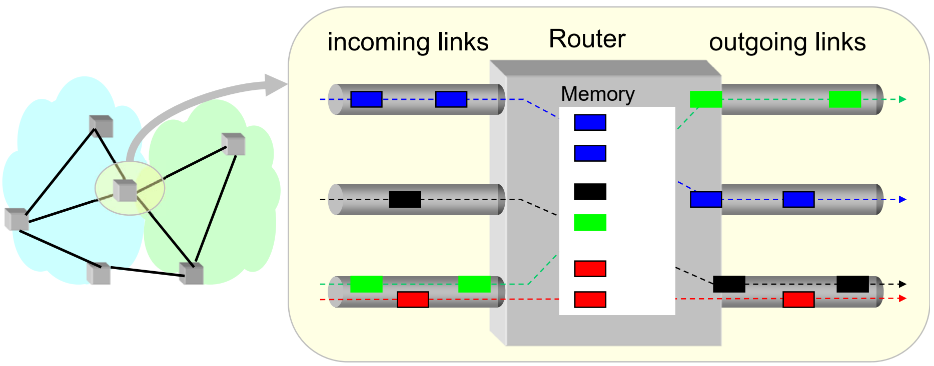
Wide Area Network

- **Wide Area Network** (WAN): network that covers a broad area (e.g., city, state, country, entire world)
  - E.g., Internet is a WAN
- WAN connects multiple physical (datalink) layer networks (LANs)
- Datalink layer networks are connected by **routers**
  - Different LANs can use different communication technology (e.g., wireless, cellular, optics, wired)



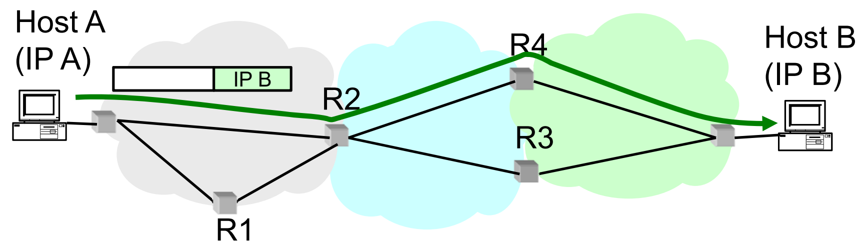
Routers

- **Forward** each packet received on an **incoming link** to an **outgoing link** based on packet's destination IP address (towards its destination)
- **Store & forward**: packets are buffered before being forwarded
- **Forwarding table**: mapping between IP address and the output link



Packet Forwarding

- Upon receiving a packet, a router
  - read the IP destination address of the packet
  - consults its forwarding table → output port
  - forwards packet to corresponding output port
- Default route (for subnets without explicit entries)
  - Forward to more authoritative router



IP Addresses vs. MAC Addresses

- Why not use MAC addresses for routing?
  - Doesn't scale
- Analogy
  - MAC address → SSN
  - IP address → (unreadable) home address
- MAC address: uniquely associated to the device for the entire lifetime of the device
- IP address: changes as the device location changes
  - Your notebook IP address at school is different from home

IP Addresses vs. MAC Addresses

- Why does packet forwarding using IP addr. scale?
- Because IP addresses can be aggregated
  - E.g., all IP addresses at UC Berkeley start with **0xA9E5**, i.e., any address of form 0xA9E5**** belongs to Berkeley
  - Thus, a router in NY needs to keep a **single** entry for **all** hosts at Berkeley
  - If we were using MAC addresses the NY router would need to maintain **an entry for every** Berkeley host!!

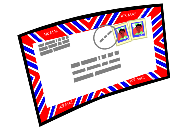
- Analogy:
  - Give this letter to person with SSN: 123-45-6789 vs.
  - Give this letter to "John Smith, 123 First Street, LA, US"

Setting up Routing Tables

- How do you set up routing tables?
  - Internet has no centralized state!
    - » No single machine knows entire topology
    - » Topology constantly changing (faults, reconfiguration, etc.)
  - Need dynamic algorithm that acquires routing tables
    - » Ideally, have one entry per subnet or portion of address
    - » Could have "default" routes that send packets for unknown subnets to a different router that has more information
- Possible algorithm for acquiring routing table
  - Routing table has "cost" for each entry
    - » Includes number of hops to destination, congestion, etc.
    - » Entries for unknown subnets have infinite cost
  - Neighbors periodically exchange routing tables
    - » If neighbor knows cheaper route to a subnet, replace your entry with neighbors entry (+1 for hop to neighbor)
- In reality:
  - Internet has networks of many different scales
  - Different algorithms run at different scales

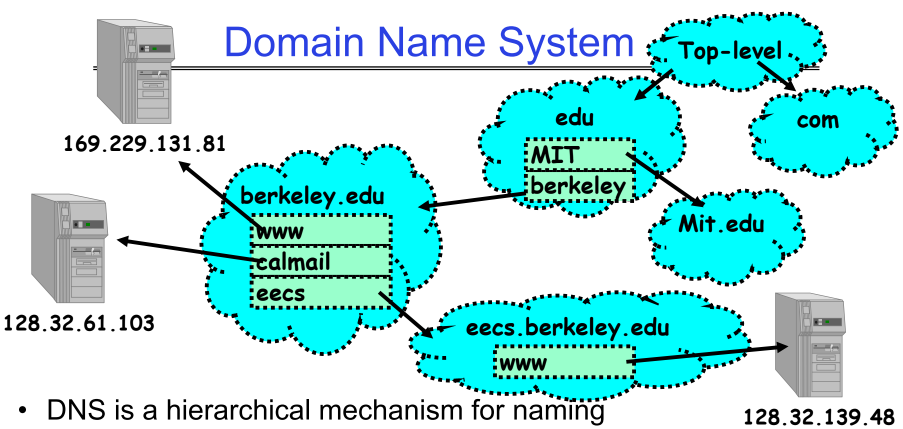
Naming in the Internet

Name ⇒ Address



- How to map human-readable names to IP addresses?
  - E.g. www.berkeley.edu ⇒ 128.32.139.48
  - E.g. www.google.com ⇒ different addresses depending on location, and load
- Why is this necessary?
  - IP addresses are hard to remember
  - IP addresses change:
    - » Say, Server 1 crashes gets replaced by Server 2
    - » Or – google.com handled by different servers
- Mechanism: Domain Naming System (DNS)

Domain Name System



- DNS is a hierarchical mechanism for naming
  - Name divided in domains, right to left: www.eecs.berkeley.edu
- Each domain owned by a particular organization
  - Top level handled by ICANN (Internet Corporation for Assigned Numbers and Names)
  - Subsequent levels owned by organizations
- Resolution: series of queries to successive servers
- Caching: queries take time, so results cached for period of time

How Important is Correct Resolution?

- If attacker manages to give incorrect mapping:
  - Can get someone to route to server, thinking that they are routing to a different server
    - » Get them to log into "bank" – give up username and password
- Is DNS Secure?
  - Definitely a weak link
    - » What if "response" returned from different server than original query?
    - » Get person to use incorrect IP address!
  - Attempt to avoid substitution attacks:
    - » Query includes random number which must be returned
- In July 2008, hole in DNS security located!
  - Dan Kaminsky (security researcher) discovered an attack that broke DNS globally
    - » One person in an ISP convinced to load particular web page, then all users of that ISP end up pointing at wrong address
  - High profile, highly advertised need for patching DNS
    - » Big press release, lots of mystery
    - » Security researchers told no speculation until patches applied

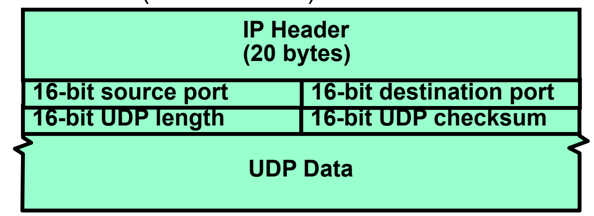
Network Layering

- Layering: building complex services from simpler ones
  - Each layer provides services needed by higher layers by utilizing services provided by lower layers
- The physical/link layer is pretty limited
  - Packets are of limited size (called the "Maximum Transfer Unit or MTU: often 200-1500 bytes in size)
  - Routing is limited to within a physical link (wire) or perhaps through a switch
- Our goal in the following is to show how to construct a secure, ordered, message service routed to anywhere:

Physical Reality: Packets	Abstraction: Messages
Limited Size	Arbitrary Size
Unordered (sometimes)	Ordered
Unreliable	Reliable
Machine-to-machine	Process-to-process
Only on local area net	Routed anywhere
Asynchronous	Synchronous
Insecure	Secure

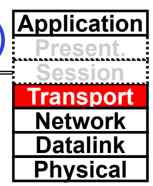
Building a messaging service on IP

- Process to process communication
  - Basic routing gets packets from machine→machine
  - What we really want is routing from process→process
    - » Add "ports", which are 16-bit identifiers
    - » A communication channel (connection) defined by 5 items: [source addr, source port, dest addr, dest port, protocol]
- For example: The Unreliable Datagram Protocol (UDP)
  - Layered on top of basic IP (IP Protocol 17)
    - » Datagram: an unreliable, unordered, packet sent from source user → dest user (Call it UDP/IP)

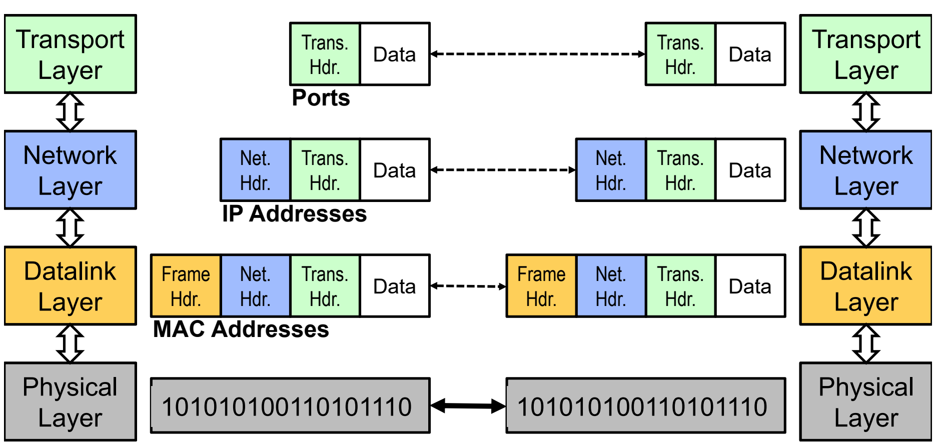


  - Important aspect: low overhead!
    - » Often used for high-bandwidth video streams
    - » Many uses of UDP considered "anti-social" – none of the "well-behaved" aspects of (say) TCP/IP

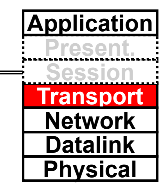
Process-to-Process Delivery (layer 4)



- Five-tuple unique connection from process at one side to process at other side:
  - [Src Addr, Src Port, Dest Addr, Dest Port, Protocol]



Internet Transport Protocols

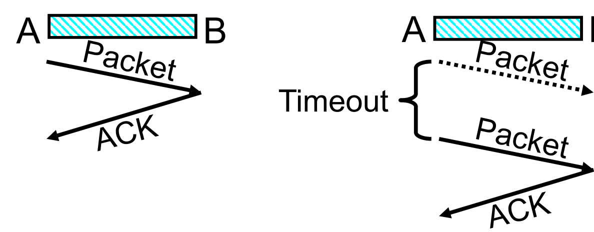


- Datagram service (**UDP**): IP Protocol 17
  - No-frills extension of "best-effort" IP
  - Multiplexing/Demultiplexing among processes
- Reliable, in-order delivery (**TCP**): IP Protocol6
  - Connection set-up & tear-down
  - Discarding corrupted packets (segments)
  - Retransmission of lost packets (segments)
  - Flow control
  - Congestion control
- Other examples:
  - DCCP (33), Datagram Congestion Control Protocol
  - RDP (26), Reliable Data Protocol
  - SCTP (132), Stream Control Transmission Protocol
- Services not available
  - Delay and/or bandwidth guarantees
  - Sessions that survive change-of-IP-address
  - Security/denial of service resilience/…

Reliable Message Delivery: the Problem

- All physical networks can garble and/or drop packets
  – Physical media: packet not transmitted/received
    » If transmit close to maximum rate, get more throughput – even if some packets get lost
    » If transmit at lowest voltage such that error correction just starts correcting errors, get best power/bit
  – Congestion: no place to put incoming packet
    » Point-to-point network: insufficient queue at switch/router
    » Broadcast link: two host try to use same link
    » In any network: insufficient buffer space at destination
    » Rate mismatch: what if sender send faster than receiver can process?
- Reliable Message Delivery on top of Unreliable Packets
  – Need some way to make sure that packets actually make it to receiver
    » Every packet received at least once
    » Every packet received at most once
  – Can combine with ordering: every packet received by process at destination exactly once and in order

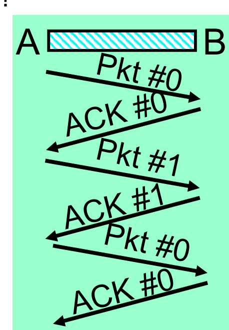
Using Acknowledgements



- How to ensure transmission of packets?
  – Detect garbling at receiver via checksum, discard if bad
  – Receiver acknowledges (by sending "ACK") when packet received properly at destination
  – Timeout at sender: if no ACK, retransmit
- Some questions:
  – If the sender doesn't get an ACK, does that mean the receiver didn't get the original message?
    » No
  – What if ACK gets dropped? Or if message gets delayed?
    » Sender doesn't get ACK, retransmits, Receiver gets message twice, ACK each

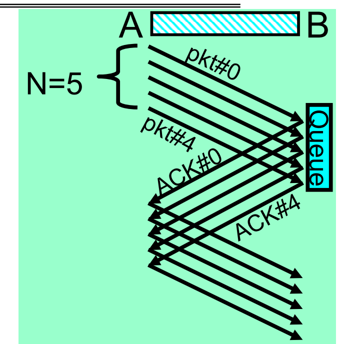
How to Deal with Message Duplication?

- Solution: put sequence number in message to identify re-transmitted packets
  – Receiver checks for duplicate number's; Discard if detected
- Requirements:
  – Sender keeps copy of unACK'd messages
    » Easy: only need to buffer messages
  – Receiver tracks possible duplicate messages
    » Hard: when ok to forget about received message?
- Alternating-bit protocol:
  – Send one message at a time; don't send next message until ACK received
  – Sender keeps last message; receiver tracks sequence number of last message received
- Pros: simple, small overhead
- Con: Poor performance
  – Wire can hold multiple messages; want to fill up at (wire latency × throughput)
- Con: doesn't work if network can delay or duplicate messages arbitrarily



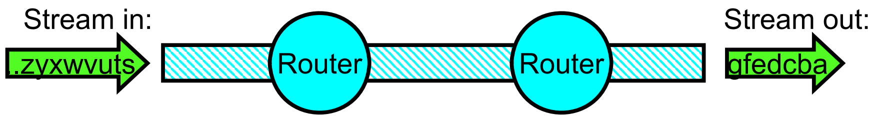
Better Messaging: Window-based Acknowledgements

- Windowing protocol (not quite TCP):
  – Send up to N packets without ack
    » Allows pipelining of packets
    » Window size (N) < queue at destination
  – Each packet has sequence number
    » Receiver acknowledges each packet
    » ACK says "received all packets up to sequence number X"/send more



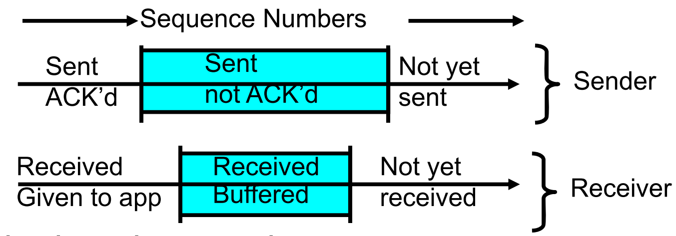
- ACKs serve dual purpose:
  – Reliability: Confirming packet received
  – Ordering: Packets can be reordered at destination
- What if packet gets garbled/dropped?
  – Sender will timeout waiting for ACK packet
    » Resend missing packets ⇒ Receiver gets packets out of order!
  – Should receiver discard packets that arrive out of order?
    » Simple, but poor performance
  – Alternative: Keep copy until sender fills in missing pieces?
    » Reduces # of retransmits, but more complex
- What if ACK gets garbled/dropped?
  – Timeout and resend just the un-acknowledged packets

Transmission Control Protocol (TCP)



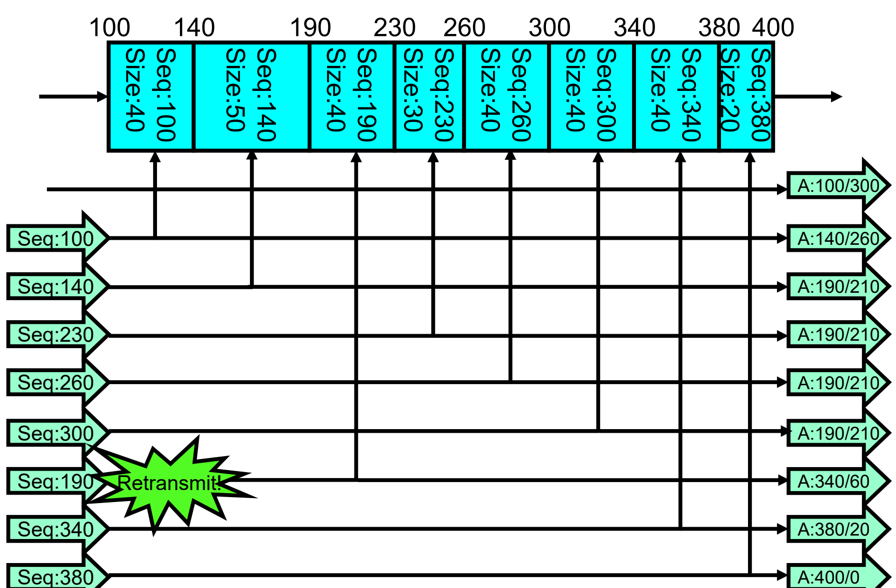
- Transmission Control Protocol (TCP)
  - TCP (IP Protocol 6) layered on top of IP
  - Reliable byte stream between two processes on different machines over Internet (read, write, flush)
- TCP Details
  - Fragments byte stream into packets, hands packets to IP
    - » IP may also fragment by itself
  - Uses window-based acknowledgement protocol (to minimize state at sender and receiver)
    - » "Window" reflects storage at receiver – sender shouldn't overrun receiver's buffer space
    - » Also, window should reflect speed/capacity of network – sender shouldn't overload network
  - Automatically retransmits lost packets
  - Adjusts rate of transmission to avoid congestion
    - » A "good citizen"

TCP Windows and Sequence Numbers



- Sender has three regions:
  - Sequence regions
    - » sent and ACK'd
    - » sent and not ACK'd
    - » not yet sent
  - Window (colored region) adjusted by sender
- Receiver has three regions:
  - Sequence regions
    - » received and ACK'd (given to application)
    - » received and buffered
    - » not yet received (or discarded because out of order)

Window-Based Acknowledgements (TCP)



Congestion Avoidance

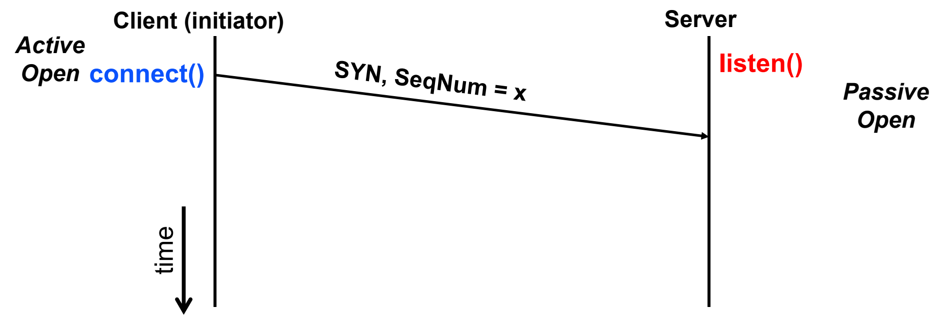
- Congestion
  - How long should timeout be for re-sending messages?
    - » Too long → wastes time if message lost
    - » Too short → retransmit even though ACK will arrive shortly
  - Stability problem: more congestion ⇒ ACK is delayed ⇒ unnecessary timeout ⇒ more traffic ⇒ more congestion
    - » Closely related to window size at sender: too big means putting too much data into network
- How does the sender's window size get chosen?
  - Must be less than receiver's advertised buffer size
  - Try to match the rate of sending packets with the rate that the slowest link can accommodate
  - Sender uses an adaptive algorithm to decide size of N
    - » Goal: fill network between sender and receiver
    - » Basic technique: slowly increase size of window until acknowledgements start being delayed/lost
- TCP solution: "slow start" (start sending slowly)
  - If no timeout, slowly increase window size (throughput) by 1 for each ACK received
  - Timeout ⇒ congestion, so cut window size in half
  - "*Additive Increase, Multiplicative Decrease*"

Open Connection: 3-Way Handshaking

- Goal: agree on a set of parameters, i.e., the start sequence number for each side
  - Starting sequence number (first byte in stream)
  - Must be unique!
    - » If it is possible to predict sequence numbers, might be possible for attacker to hijack TCP connection
- Some ways of choosing an initial sequence number:
  - Time to live: each packet has a deadline.
    - » If not delivered in X seconds, then is dropped
    - » Thus, can re-use sequence numbers if wait for all packets in flight to be delivered or to expire
  - Epoch #: uniquely identifies *which* set of sequence numbers are currently being used
    - » Epoch # stored on disk, Put in every message
    - » Epoch # incremented on crash and/or when run out of sequence #
  - Pseudo-random increment to previous sequence number
    - » Used by several protocol implementations

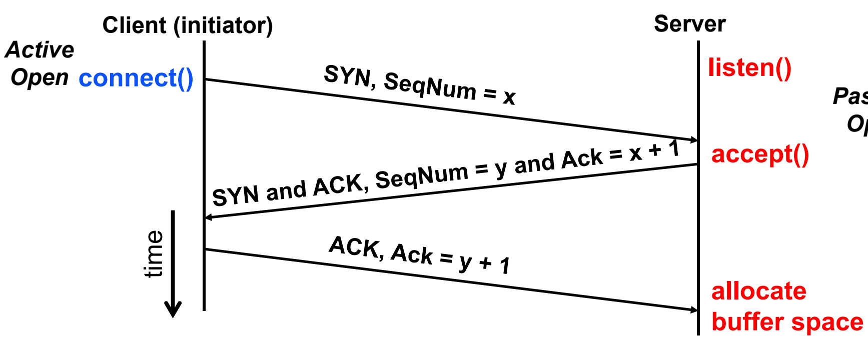
Open Connection: 3-Way Handshaking

- Server waits for new connection calling listen()
- Sender call connect() passing socket which contains server's IP address and port number
  - OS sends a special packet (SYN) containing a proposal for first sequence number, x

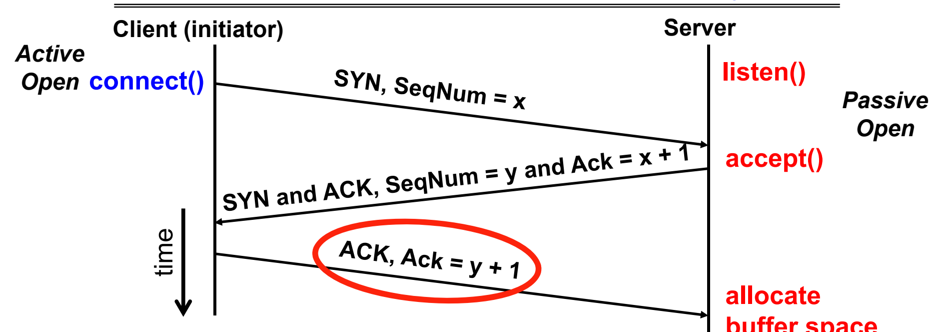


Open Connection: 3-Way Handshaking

- If it has enough resources, server calls accept() to accept connection, and sends back a SYN ACK packet containing
  - Client's sequence number incremented by one, (x + 1)
    - » Why is this needed?
  - A sequence number proposal, y, for first byte server will send



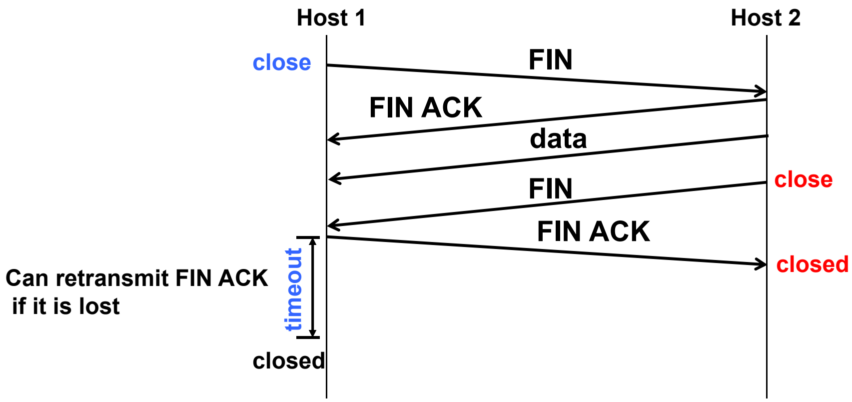
Denial of Service Vulnerability



- SYN attack: send a huge number of SYN messages
  - Causes victim to commit resources (768 byte TCP/IP data structure)
- Alternatives: Do not commit resources until receive final ACK
  - *SYN Cache*: when SYN received, put small entry into cache (using hash) and send SYN/ACK, If receive ACK, then put into listening socket
  - *SYN Cookie*: when SYN received, encode connection info into sequence number/other TCP header blocks, decode on ACK

Close Connection

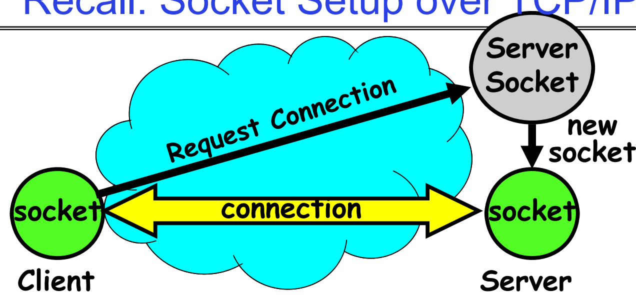
- Goal: both sides agree to close the connection
- 4-way connection tear down



Use of TCP: Sockets

- Socket: an abstraction of a network I/O queue
  - Embodies one side of a communication channel
    - » Same interface regardless of location of other end
    - » Could be local machine (called "UNIX socket") or remote machine (called "network socket")
  - First introduced in 4.2 BSD UNIX: big innovation at time
- Using Sockets for Client-Server (C/C++ interface):
  - On server: set up "server-socket"
    - » Create socket, Bind to protocol (TCP), local address, port
    - » Call listen(): tells server socket to accept incoming requests
    - » Multiple accept() calls on socket to accept incoming connection requests
    - » Each successful accept() returns a new socket for a new connection
  - On client:
    - » Create socket, Bind to protocol (TCP), remote address, port
    - » Perform connect() on socket to make connection
    - » If connect() successful, have socket connected to server
- Network Address Translation (NAT):
  - Local subnet (non-routable IP addresses) ⇒external IP
  - Client-side firewall replaces local IP address/port combination with external IP address/new port
  - Firewall handles translation between different address domains using table of current connections

Recall: Socket Setup over TCP/IP



- Things to remember:
  - Connection involves 5 values:
    [Client Addr, Client Port, Server Addr, Server Port, Protocol]
  - Often, Client Port "randomly" assigned
  - Server Port often "well known"
    - » 80 (web), 443 (secure web), 25 (sendmail), etc
    - » Well-known ports from 0—1023
- Network Address Translation (NAT) allows many internal connections (and/or hosts) with a single external IP address

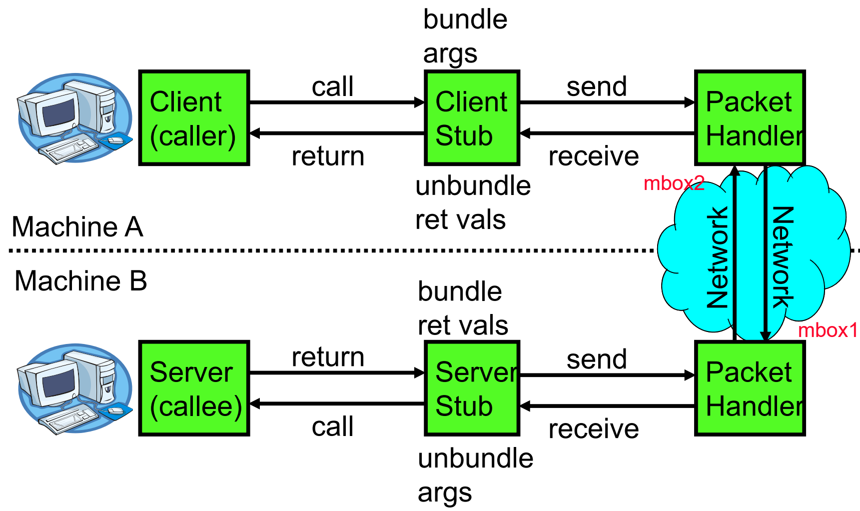
Remote Procedure Call (RPC)

- Raw messaging is a bit too low-level for programming
  - Must wrap up information into message at source
  - Must decide what to do with message at destination
  - May need to sit and wait for multiple messages to arrive
- Another option: Remote Procedure Call (RPC)
  - Calls a procedure on a remote machine
  - Client calls:
    remoteFileSystem→Read("rutabaga");
  - Translated automatically into call on server:
    fileSys→Read("rutabaga");

RPC Implementation

- Request-response message passing (under covers!)
- "Stub" provides glue on client/server
  – Client stub is responsible for "marshalling" arguments and "unmarshalling" the return values
  – Server-side stub is responsible for "unmarshalling" arguments and "marshalling" the return values.
- Marshalling involves (depending on system)
  – Converting values to a canonical form, serializing objects, copying arguments passed by reference, etc.

RPC Information Flow



RPC Details (1/3)

- Equivalence with regular procedure call
  – Parameters ⇔ Request Message
  – Result ⇔ Reply message
  – Name of Procedure: Passed in request message
  – Return Address: mbox2 (client return mail box)
- Stub generator: Compiler that generates stubs
  – Input: interface definitions in an "interface definition language (IDL)"
    » Contains, among other things, types of arguments/return
  – Output: stub code in the appropriate source language
    » Code for client to pack message, send it off, wait for result, unpack result and return to caller
    » Code for server to unpack message, call procedure, pack results, send them off

RPC Details (2/3)

- Cross-platform issues:
  – What if client/server machines are different architectures/ languages?
    » Convert everything to/from some canonical form
    » Tag every item with an indication of how it is encoded (avoids unnecessary conversions)
- How does client know which mbox to send to?
  – Need to translate name of remote service into network endpoint (Remote machine, port, possibly other info)
  – Binding: the process of converting a user-visible name into a network endpoint
    » This is another word for "naming" at network level
    » Static: fixed at compile time
    » Dynamic: performed at runtime

RPC Details (3/3)

- Dynamic Binding
  - Most RPC systems use dynamic binding via name service
    - » Name service provides dynamic translation of service → mbox
  - Why dynamic binding?
    - » Access control: check who is permitted to access service
    - » Fail-over: If server fails, use a different one
- What if there are multiple servers?
  - Could give flexibility at binding time
    - » Choose unloaded server for each new client
  - Could provide same mbox (router level redirect)
    - » Choose unloaded server for each new request
    - » Only works if no state carried from one call to next
- What if multiple clients?
  - Pass pointer to client-specific return mbox in request

Problems with RPC: Non-Atomic Failures

- Different failure modes in dist. system than on a single machine
- Consider many different types of failures
  - User-level bug causes address space to crash
  - Machine failure, kernel bug causes all processes on same machine to fail
  - Some machine is compromised by malicious party
- Before RPC: whole system would crash/die
- After RPC: One machine crashes/compromised while others keep working
- Can easily result in inconsistent view of the world
  - Did my cached data get written back or not?
  - Did server do what I requested or not?
- Answer? Distributed transactions/Byzantine Commit

Problems with RPC: Performance

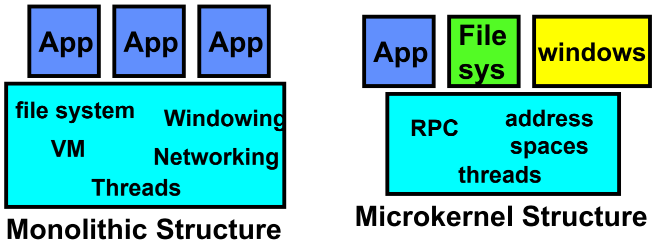
- Cost of Procedure call « same-machine RPC « network RPC
- Means programmers must be aware that RPC is not free
  - Caching can help, but may make failure handling complex

Cross-Domain Communication/ Location Transparency

- How do address spaces communicate with one another?
  - Shared Memory with Semaphores, monitors, etc…
  - File System
  - Pipes (1-way communication)
  - "Remote" procedure call (2-way communication)
- RPC's can be used to communicate between address spaces on different machines or the same machine
  - Services can be run wherever it's most appropriate
  - Access to local and remote services looks the same
- Examples of RPC systems:
  - CORBA (Common Object Request Broker Architecture)
  - DCOM (Distributed COM)
  - RMI (Java Remote Method Invocation)

Microkernel operating systems

- Example: split kernel into application-level servers.
  - File system looks remote, even though on same machine



- Why split the OS into separate domains?
  - Fault isolation: bugs are more isolated (build a firewall)
  - Enforces modularity: allows incremental upgrades of pieces of software (client or server)
  - Location transparent: service can be local or remote
    - » For example in the X windowing system: Each X client can be on a separate machine from X server; Neither has to run on the machine with the frame buffer.

Summary (1/2)

- Two-phase commit: distributed decision making
  - First, make sure everyone guarantees they will commit if asked (prepare)
  - Next, ask everyone to commit
- Byzantine General's Problem: distributed decision making with malicious failures
  - One general, n-1 lieutenants: some number of them may be malicious (often "f" of them)
  - All non-malicious lieutenants must come to same decision
  - If general not malicious, lieutenants must follow general
  - Only solvable if $n \geq 3f+1$
- BlockChain protocols
  - Cryptographically-driven ordering protocol
  - Could be used for distributed decision making

Summary (2/2)

- Internet Protocol (IP): Datagram packet delivery
  - Used to route messages through routes across globe
  - 32-bit addresses, 16-bit ports
- DNS: System for mapping from names⇒IP addresses
  - Hierarchical mapping from authoritative domains
  - Recent flaws discovered
- Ordered messages:
  - Use sequence numbers and reorder at destination
- Reliable messages:
  - Use Acknowledgements
- TCP: Reliable byte stream between two processes on different machines over Internet (read, write, flush)
  - Uses window-based acknowledgement protocol
  - Congestion-avoidance dynamically adapts sender window to account for congestion in network
- Remote Procedure Call (RPC): Call procedure on remote machine
  - Provides same interface as procedure
  - Automatic packing and unpacking of arguments without user programming (in stub)