Prove sufficient ("if"), and necessary ("only if") to prove "if and only if"

People

Baran (USAF, survivable netwoks) and Kleinrock advocate packet switching; Licklicker vision of Galactic Network; Roberts vision of ARPANET, Kahn advocates Open Architecture networking (networks should be independent and not required to change, best-effort connection, routers between networks, no global control), David Clark was Chief Architect (authored end-to-end principle, adhered to the basic design principles)

Circuit Switching

Pros: predictable performance (known delays, no drops); simple abstraction (reliable communication, no worries about lost/out-of-order packets), simple forwarding (basedon time or frequency)
Cons: not resilient (any failure prevents transmission, and entire transmission has to be restarted), wastes bandwidth, inefficient for bursty apps, blocked connections (conn. rfused when insufficient resources)

Packet Switching

"Statistical multiplexing" gambles that packets don't all arrive at the same time, so we don't need capacity for all of them at peak transmission rates
Pros: easy recovery from failures (routers don't have individual flows), efficiency (from multiplexing), deployability (easy to link different networks together)
Cons: worse service for flows (packet delays, drops, out-of-order), must deal with congestion, complicated forwarding

Datagram Network Properties

Latency (delay): propogation time for data sent along link; **Bandwidth (capacity)**: amount of data sent/received per unit time; **Bandwidth-delay product (BDP)**: amount of data "in flight" at any time; **Utilization**: arrival/bandwidth;

Networking Delays

Transmission delay: time spent transmitting data, from first bit pushed to wire until last bit pushed on; **Propogation delay**: time a bit spends traversing the link(limited by light); **Queueing delay**: time spent waiting in a queue, caused by burstiness of arrivals and variations in packet lengths; **End to end delay** between when sender starts sending and when receiver finishes receiving; **Roundtrip delay (RTT)**: total time for packet to reach destination and response to return to sender; **Jitter**: difference between min and max delay
Little's Law: avg. # of packets waiting = average arrival rate $\times$ avg. time packets wait in queue

Clark's Internet Design Goals

Connect existing networks (using IP), *robust* (as long as network not partitioned, 2 hosts should be able to communicate, and failures should not interfere with endpoint semantics), *support many types of delivery services* (build lowest common demoninator service, application-neutral network), *accomodate variety of networks* (successful because of minimal requirements on networks), *allow distributed maangement* (easy deployment, but makes management hard), *host attachment* (hosts expensiver because need to be smart, but administrative cost of adding hosts is low) *cost effective* (low end cheaper than circuit switching, expensiver than circuit switching at high end), *resource accountability* (failed)

3 Design Principles

Layering

Application: HTTP; **Transport (L4)** communication between processes, end-to-end delivery (possibly reliable): TCP, UDP; **Network (L3)** global best-effort delivery: IP; **Datalink (L2)** local best-effort delivery: ethernet; **Physical** bits on wire
All layers must exist at host, routers only implements everything Network and below

End-to-End Principle

Only-if-sufficient: implement function in lower levels only if it can be completely implemented at this level;
Only-if-necessary: make network layer absolutely minimal (increases flexibility), don't implement *anything* in the network that can be done correctly by the hosts;
Only-if-useful: if hosts can implement correctly, implement it in a lower layer only as a performance enhancement, but only if it does not impose burden on apps that don't require the functionality
Ignores: stakeholders besides users (ISP, commercial, money-chain), the need for middlebox functionality (some functions are more easily done in the network)

Fate-Sharing

When storying state in a distributed system, co-locate it with entities that rely on that state. Keep network state at end hosts instead of inside routers

Reliable Transport Goals

Goals: correctness, timeliness, efficiency, "fairness"
Correctness: always resend packet if previous transmission lost or corrupted, maybe resent at other times;
Window algorithms: takes advantage of bandwidth, limits bandwidth used, limits buffering needed at receiver; separates concerns: size of W, nature of feedback ACKs (full information, individual packets, cumulative), response to loss (resend on timeout, duplicate ACKs, or NACKs)
Ratelsss Codes: receipt of M packets allows recovery of file (where M is close to size of file), receiver only sends ACK when M are receive, sender keeps sending until receives ACK, is timely and correct
Paradox: majority of flows are short, but majority of bytes are in long flows
Resending packet until you receive ACK wastes $B \times RTT$ bandwidth

Routing

Forarding decision are deterministic and based on routing state (table) in switch, mapping packetState (destination, source, incoming switch port, other info) + rouyingState to outgoingPort
Destination-Based Routing: paths to same destination never cross, paths never split once they meet, creates a spanning tree rooted at destination covering every node once
Local routing state: state in a single router
Global routing state: collection of routing state in each router, determines which paths packets take
Global routing state is *valid* iff there are no dead ends (ex: there is no outgoing port on non-destination node) and no loops (hard).
Forwarding (*data plane*) is directing a packet to an outgoing link in individual router using routing state. Routing (*control plane*) is computing paths packets will follow using routing state jointed created with other routers.

Routing on Spanning Tree

Spanning tree is selection of edges that form a tree spanning every vertex without forming loops. Only one path from source to destination. Send packets along all paths. They won't loop, and some will hit deadends, but one will reach destination.

Self-Learning Switch

Store mapping from source ID of packet with incoming port to switch table. Use time-to-live field to eventually forget mapping. **Weaknesses**: requires loop-free topology, very little control over paths, tree must be recomputed after failure, entries must time out when hosts move

```
index the switch table using destination ID
if entry found for destination{
    if dest on port from which packet arrived { drop packet }
    else { forward packet on port indicated }
} else { flood }
```

Link-State

Each router tracks incident links, and floods it, so each node has same global view. Each router computes path using same algo. Global state, local computation. Challenges: scaling, transient disruptions.
"Least Cost" routes are destination-based and easy way to avoid loops.
Reliability: ensure all nodes receive link-state info, and all use latest version; **Challenges**: packet loss and out-of-order arrival; **Solutions**: ACKs, retransmissions, and seqno;
Initiate on: topology change (failure, recovery), config change, periodically **Convergence**: forwarding is consistent after convergence, but before, there is risk of lost, looping, or out-of-order packets

Distance Vector

Each router knows links to its neighbors and has provisional "shortest path". Routes exchange distance-vector info with neighbors. Routers update their idea of best path using info from neighbors.

Bellman-Ford Algorithm

Router's Table: entry in row Y and column Z of node X means "best known distance from X to Y, via Z as next hop = $D_z(X, Y)$
After X rounds of exchange, we get the best (X+1)-hop paths. If all nodes minimize same metric, and that metric increases around loops, convergence is guaranteed. If router lies, all traffic from nearby routers get sent there.

IP

Steps: accesing network from laptop (wireles/ethernet, NAT/firewall network management), mapping real world name to network name (getting host), mapping network name to location (IP addr), downloading content from location (TCP)
Network Management: most undeveloped part of Internet architecture; **Security concerns**: privacy, integrity, provenance (not imposter)

IP Packet Structure (Bits)

Read packet correctly: version number (4), header length (4), total length (16)
Get packet to destination and back: source and destination IP addresses (32 x 2)
Carry data Tell host what to do when packet arrives: protocol for demuxing at receiver (8) (eg: 6 for TCP, 17 for UDP)
Specify special network handling: type-of-service (8) and options *Deal with path problems*: check for header corruption with checksum (16), loop with TTL (8) (decremented each hop, discarded when 0), packet too large with fragmentation (32 bit info for packet identifier, flags(**R**eserved: ignore, **D**F: don't fragment, **M**F: more fragments coming), and fragment offset (in 8-byte units, to allow further fragmentation))

IPv6

Ends deal with problems: removed fragmentation and checksum, kept TTL; **Simplify handling**: new options mechanism (next header), and eliminated header length; **Provide general flow label for packet**: not tied to semantics, gives flexibility

Sender Attacks

Use fake source addr for DOS attack, evading detection, or framing spoofed host. IP options often processed in router's slow path (allowing DOS). If attacker sets TOS, and regular traffic doesn't, then network prefers attack traffic. Evade network management by splitting attack across fragments. Send fragments exceeding IP datagram limit. State-holding attack is when attacker doesn't send all fragments, and receiver waits.

IP Addressing

Layer 2 addressing uses MAC addresses. Use dotted-quad notation. IPv4 addresses 32 bits. Add layer of indirection for flexibility, hierarchical structure for scalability. **Prefix** is network address, **suffix** is hostaddress. "Slash X" means a X-bit prefix with 2^{32-X} addresses. **Subnet** is region without routers,containing addresses within the "subnet mask".
Original addrs used 8 bits for network addresses (assumed 256 networks were enough). **Classful** addrs used opening bits to determine prefix length (0for /8, 10 for /16, 110 for /24, 1110 for multicast), routers ended up knowing many class C's (/24), wasted addr space. **CIDR**: Classless Interdomain Routing. Must specify both addr and mask.
Aggregation not possible when multi-homed customer has 2 providers.

DHCP

uses UDP. Only uses local broadcast. Allocation of addr is "soft state" (forgotten if no refresh received when timer expires), so if request isn't refreshed, server takes addr back (in case host doesn't release).
DHCP discover (broadcast): client sends on layer 2.
DHCP offer (broadcast): multiple servers send configuration parameters with lease time
DHCP request (broadcast): client sends request echoing params
DHCP ACK (boardcast): server confirms, and so other servers see they weren't chosen

Network Address Translation

Many hosts can share single address. Uses port numbers (fields in transport layer) to multiplex single address. Early exmaple of middlebox injecting functionality into network.

Forwarding

If no match in forwarding table, take default route (ie: if not on subnet, send to ISP). Because can't tell where network addr ends in CIDR, we must use LPM. **Longest Prefix Match**: record port associatd with first math, and only over-ride when it matches another prefix during walk down tree. Decreases size of routing table.

Transport

Communication between application processes on different hosts. Sender breaks app messages into segments, passes to network layer. UDP uses destination port (and addr). TCP uses source/destination ports & addrs.

UDP

IP plus port numbers to support de/muxing. Optional error checking on packet contents. Finer control over when data is sent. No delay for connection establishment. No connection state. Small packet header overhead (8 bytes). Used by DNS.

TCP

Checksum: detect corrupted data at receiver, and drop.
Sequence numbers: detect missing data, and putting back in order. Receiver sends ACK with the # of the next expected seqno.
Retransmission: sender retransmits with timout based on estimate of rount-trip time. Rapid retransmission with fast retransmit algo.
Sliding Window flow control: advertised window W (can send W bytes beyond next expected byte), receiver uses W to limit number of bytes sender can have in flight. If $\frac{W}{RTT} < B$, transfer has speed $\frac{W}{RTT}$. Else, the transfer has speed B. Left edge of window for sender is beginning of unACK'd data, for receiver is beginning of undelivered data. Window advances for sende when new data ack'd, for receiver when receiving process consumes data. Receiver advertises to sender where the receiver window ends (righthand edge), and sender does not exceed.
Segment: IP packet smaller than Maximum Transmission Unit (1500 bytes). TCP packet has TCP header $\geq$ 20 bytes. TCP segment smaller than Maximum Segment Size bytes. $MSS = MTU - (IPheader) - (TCPheader)$ **Initial Sequence Number**: since port #s might get used again, TCP requires changing ISN (set from 32 bit clock tick every 4 ms, wraps around once per 4.55 hours). Hosts exchange ISNs.
Connection Establishment: 3-way handshake. A's SYN (seqnum x) to B. A sets timer (default 3 sec, sometimes 6) and waits for SYNACK. B's SYNACK (seqno y, ack x+1) to A. A's ACK (ack y+1) to B, and then A's data to B. Each host tells its ISN to the other.
Connection Teardown: A sends FIN to B. B sends ACK. Connection now half-closed. A sends ACK to B. A sets timeout to avoid reincarnation, and then closes connection. Or, it can go FIN (from A), FIN+ACK (from B), ACK (from A). *Abrupt termination*: A sends RESET to B. B does not ACK RST, so RST is not delivered reliably. If B sends anything else, A sends another RST.
Retransmission: set retransmission timeout (RTO) based on RTT. Karn/Partridge algo measures SampleRTT for original transmissions, and uses exponential backoff (when RTO timer expires, double it up to max $\geq$ 60 sec. After successful original trans, collapse RTO back to computed value). Exponential averaging:
$estimate\,(n) = \alpha estimate\,(n-1) + (1-\alpha)\,value(n)$.
$estimate\,(n) = (1-\alpha)\,Sum\left(\alpha^k value\,(n-k)\right)$. RTT estimation with $\alpha = \frac{7}{8}$. **Jacobson/Karels algo**: Deviation=abs of sampleRTT - estimatedRTT.
EstimatedDeviation is exp average of Debiation.
RTO=EstimatedRTT + 4xEstimatedDeviation.