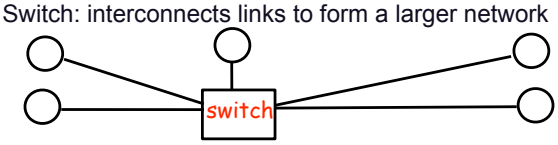
CS 140 – Summer 2008 – Lecture #23: Network and End-to-End layers

Past & Present

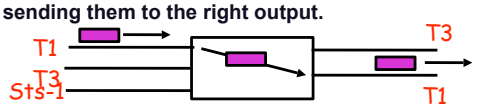
- Last time: pushing bits out of/into hardware
  - **Link layer. How to: encode bits on wire (inc. antenna), parse bits into packets, arbitrate between senders, name receivers**
- Today: network layer (portable bit pushing) and end-to-end (useful bit pushing) layers
- Network layer: given a packet, get it to the other side of a large (or HUGE) collection of networks.
  - **Issue 1: Portability. Provide an interface that works across heterogeneous networks.**
  - **Issue 2: Scalability. Provide names and routing that works with billions of end hosts.**

Moving packet from a to b

- Switch: interconnects links to form a larger network



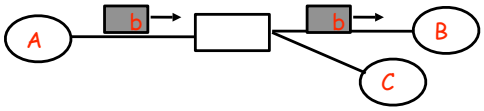
- Two parts:
  - **Forwarding: Taking packets arriving on an input and sending them to the right output.**



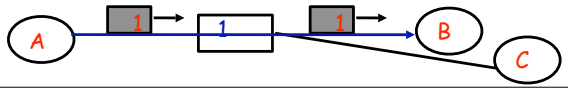
  - **Routing: Collecting the information that tells you possible routes to destination (and thus which output link to send the packet on).**

Two connection models

- Connectionless (or "datagram"):
  - **Each packet contains enough information that routers can decide how to get it to its final destination**

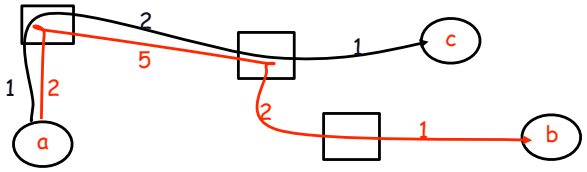


- Connection-oriented (or "virtual circuit")
  - **First set up a connection between two nodes.**
  - **Label it (called a virtual circuit identifier (VCI)).**
  - **All packets carry label.**



Virtual circuit switching (what ATM does)

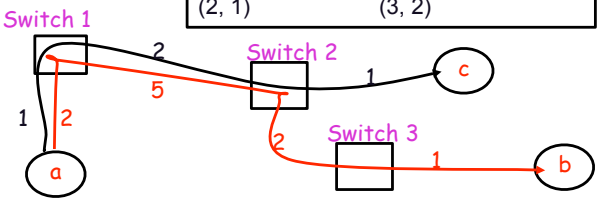
- Forming a circuit:
  - **Send a connection request from A to B. Contains VCI + address of B.**
  - **Rule: VCI must be unique on the link its used on.**
  - **Switch creates an entry mapping input messages with VCI to output port.**
  - **Switch picks a new VCI unique between it and next switch.**



Virtual circuit forwarding

- For each VCI switch has a table which maps input link to output link and gives the new VCI to use
  - **If a's messages come into switch 1 on link 2 and go out on link 3 then the table will be:**

(Input link,VCI)	(output link, new VCI)
(2, 2)	(3, 5)
(2, 1)	(3, 2)



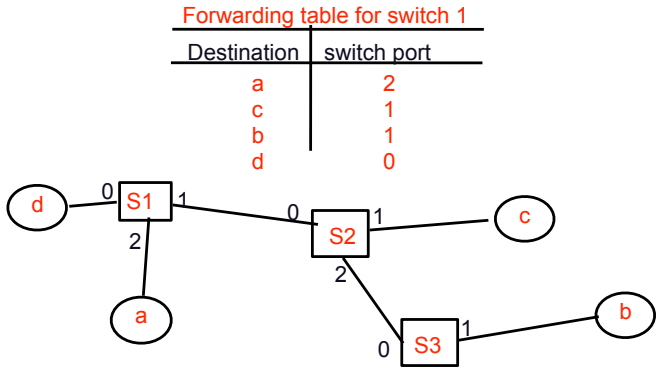
Virtual circuit issues

- Good: easy to associate resources with flows
  - **Can guarantee buffering and delay, which makes "quality of service" guarantees (QoS) easy to provide.**
  - **Also good: VCI small, making per-packet overhead small.**
- Bad: not good in the face of crashes
  - **Doesn't handle host crashes well: each connection has state strewn throughout network. to close connection, host must explicitly issue a "tear down."**
  - **In general, to survive failure, want to make stuff as "stateless" as possible, trivially eliminating any storage management problems.**
  - **Doesn't handle switch crashes well: have to teardown and reinitiate a new circuit.**
- Telephone network is connection-based

Datagrams

- Simple idea:
  - **Don't set up a connection, just make sure each packet contains enough information to get it to destination.**
  - **What is this? Complete destination address.**
  - ***"In a connectionless network, you are always connected." D. Cheriton***
- Forwarding:
  - **Switch creates a "forwarding table", mapping destinations to output port (ignores input ports).**
  - **When a packet with a destination address in the table arrives, it pushes it out on the appropriate output port.**
  - **When a packet with a destination address not in the table arrives, it must find out more routing information (next problem).**

Datagram example

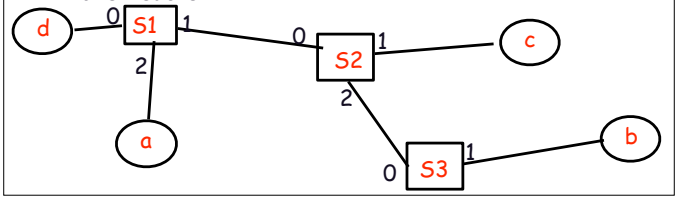


Datagram Tradeoffs

- Good:
  - **No round-trip delay to setup connection.**
  - **Each packet forwarded independently of last: if switch or link fails, can be routed around it.**
  - **Resources allocated dynamically (adaptively) rather than statically bound at connection time.**
  - **Lets each "flow" achieve peak bandwidth of idle link.**
- Bad:
  - **Busy link = unpredictable, wild service fluctuations.**
  - **Each packet carries full destination address, which makes per packet overhead higher.**
- Internet uses datagrams ("Internet Protocol" or **IP**)

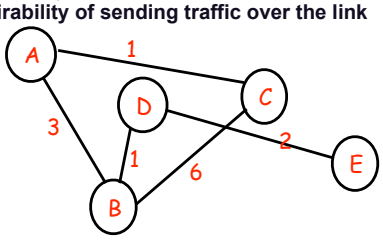
Some problems

- Where do the forwarding tables come from?
  - **Could hand-enter into a central table.**
  - **But this doesn't work well if nodes crash, and as the number of nodes goes to infinity (internet).**
- And what about scale????
  - **Recall: size of forwarding table grew O(hosts)**
  - **this... sucks.**



Building routing tables

- Routing = graph theory problem. The graph:
  - **Nodes = switches or hosts**
  - **Edges = links, have an associated cost which approximates the desirability of sending traffic over the link**



The routing problem: find the lowest-cost path between any two nodes where the cost of path = sum of all edges that make up the path.

A simple centralized routing scheme

- At creation time:
  - **Have one central node K.**
  - **Have every switch send a vector containing (neighbor, cost) for each of its outgoing links to K.**
  - **From this information, K can compute a graph that gives the topology of the network and then whip out a graph theory algorithm to find shortest path.**
  - **K then sends this matrix to all switches.**
- Nice and simple
- Problem: didn't scale to large networks
  - **Real networks were just too big. K got crushed.**
  - **CW: Centralization is the enemy of scalability, so good routing protocols are distributed.**
- (Now: faster hardware? Centralized OK for Stanford?)

Link state routing (sort of used in Internet)

- Basic idea:
  - **Every node knows how to reach its direct neighbors.**
  - **If this information can be disseminated to every node, then we will have enough information to good routes.**
- Relies on two mechanisms:
  - ***Reliable flooding* of link-state information.**
  - **Calculation of routes from sum of all accumulated knowledge (uses a modified form of Dijkstra's algorithm).**
- A link state packet:
  - **[ID of creating node, list of (neighbor, cost), sequence number, time to live].**
  - **Sequence number: monotonically increasing integer used to order link state packets.**
  - **Time-to-live: make sure packet doesn't circulate forever.**

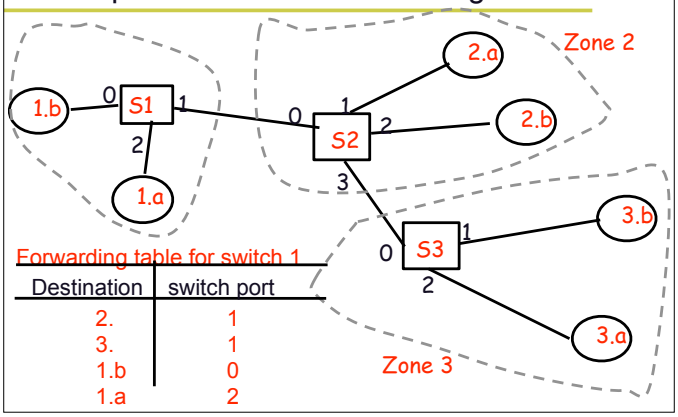
A node-level view of reliable flooding

```
receive(pkt)
    If already have a copy of LSP from pkt.ID
        if pkt's sequence number <= copy's
                discard pkt
        else
                decrement pkt.TTL
                replace copy with pkt
                forward pkt to all links besides the
                     one that we received it on
# done every 10 minutes or so
gen_LSP()
        increment node's sequence # by one
        recompute cost vector
        send created LSP to all neighbors
```

Scalable routing

- Problem: our routing tables grow with the number of nodes. This is a real problem.
  - **What was the cause? Our addresses are flat. Every router needs an entry for each.**
  - **Solution: hierarchy! (or, structured grouping)**
- Hierarchical addressing:
  - **Divide network into zones. Label these uniquely. (1,2,…)**
  - **Have node addresses include the zone that the node is in. (make sub-zones and sub-sub-zones as needed).**
  - **Top level routers know how to forward packets to the router in charge of zone.**
  - **Zone routers know how to forward to every node in their domain (or to the next level down).**

Example: hierarchical addressing



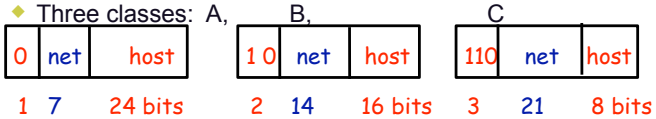
Forwarding table for switch 1

Destination	switch port
2.	1
3.	1
1.b	0
1.a	2

Example: the internet protocol (IP)

- IP addresses form a 2-level hierarchy
  - **Two parts: network and host. network tells which network host is on. Hosts on same network have same prefix.**

  - **Maps well to internetwork (network of multiple networks)**
  - **IP(V4) addresses are 32 bits. Are included in every IP pkt.**
- Three classes: A, B, C



ARP: Mapping IP addresses to link-level (LL)

- We can forward IP packet to a physical network, but how to get it to a host on that network?
  - **e.g., need a translation between IP address of host and its Ethernet address so that the router can encapsulate the packet in an Ethernet packet and send it to host.**
  - **How to get these mappings? "address resolution protocol"**
- Router (or switch) keeps a table of (IP->LL) mappings.
  - **If it gets a packet for an IP address not in this "ARP cache" it broadcasts a query containing the IP address.**
  - **Every host checks if its IP address matches and, if so, sends a response with its link-level address back to originator.**
  - **This can work in the reverse: RARP/BOOTP/DHCP**
  - **(ARP cache entries are aged. Why?)**

IP: best-effort, host-to-host protocol

- IP = portable, connectionless (datagram) protocol
- Host-to-host:
  - **IP gives each host a globally unique IP address**
- Best effort "service model"
  - **Host gives datagram to IP; IP does its best to deliver it.**
  - **No attempt is made to recover from lost, reordered, duplicated, or corrupted packets.**
  - **Synthesize reliability at higher levels (what about delay?)**
- IP provides portability by:
  - **A common packet format that gets used on all networks.**
  - **Invisibly translating, splitting and reassembling packet as it traverses over different physical networks.**
  - **A global, network-wide address space.**

Portable datagrams

- Every datagram carries enough information to forward packet

- IP goal: combine many physically distinct networks into one logical network. How?
  - **Every host and router in logical network must understand IP packets; every router be able to forward them.**
  - **Key: "best effort" service model. About the simplest service you can ask for from the underlying network**
  - **(IP goal: to "run over anything")**
- Network independence? fragmentation and reassembly

Fragmentation and reassembly

- Problem: physical networks have different MTUs
  - **"maximum transmission unit": Ethernet: 1500B, FDDI 4500B, ATM 48B(!)**
  - **Choice 1: packet small enough to fit in anything?**
- Choice 2: fragmentation and reassembly
  - **If packet > MTU of network, split (fragment) into pieces.**

  - **Put address into each piece, along with id + byte offset so it can be put back together (reassembled) by host.**
  - **How to pick initial packet size? (Hint: "usually" packets intended for machines on same network).**

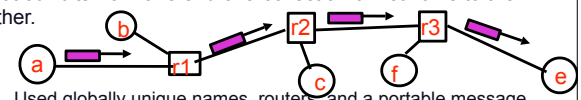
Summary: IP's mechanisms for scalability

- Hook many networks together?
  - **Billions of hosts + lots of weird constraints.**
- How to handle billions of hosts?
  - **Hierarchical addresses.**

  - **Routers only need to know how to forward packet to other networks, rather than to all hosts.**
  - **Called "hierarchical aggregation:" condenses all hosts on entire network into a single integer (the network #).**
- How to accommodate weird physical networks?
  - **Its connectionless, best-effort service model: a too stringent service model won't work in real world.**
  - **IP philosophy: make undemanding enough that just about any network can provide the necessary service.**

End-to-end layer - Making the network useful/usable

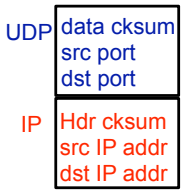
- Network Layer
  - Got our bits from one end of a collection of networks to the other.

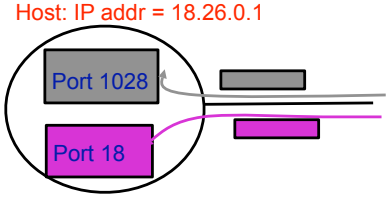


  - Used globally unique names, routers, and a portable message abstraction. Example: the internet protocol (IP)
  - Great, except... how can a process use it?
- Making it useful/usable: end-to-end layer
  - **Interface between applications and network**
  - **Some successful end-to-end protocols**
  - **Turning "best effort" into a usable interface (or, deriving TCP from first principles)**

Success story #1: UDP

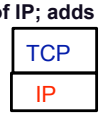
- Unreliable/User Datagram Protocol: best-effort, process-to-process
  - **Lives on top of IP: adds corruption detection and "ports" so packets can be addressed to a process on host.**
  - **(note there are many other fields in header)**





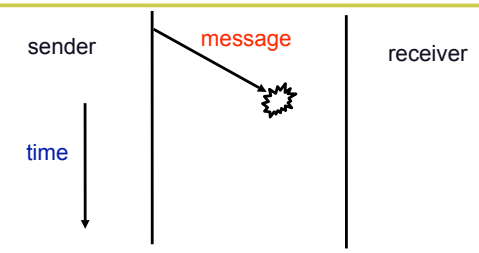
Success story #2: TCP

- "Transmission control protocol"
  - **Reliable, in-order, process-to-process, two-way byte stream.**
- Like UDP:
  - **Layered on top of IP; adds ports and data checksums.**



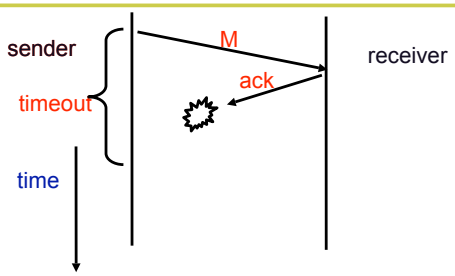
- Unlike UDP in two ways
  - **Connection based.**
  - **The important difference: recovers from packet loss, duplication, corruption, reordering.**
  - **Next: answering "how????" from first principles.**

Lost packets



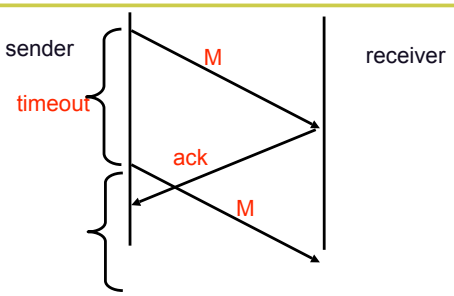
- Some causes:
  - **Traffic burst causes router buffers to overflow.**
  - **Link corrupts packet (e.g. wireless: up to 40% packet loss).**
  - **How to fix?**

Lost acks



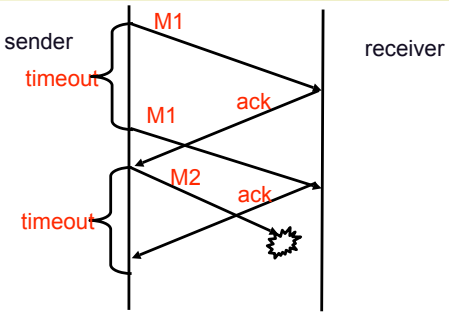
- What will the result be?

Delayed acks = packet duplication



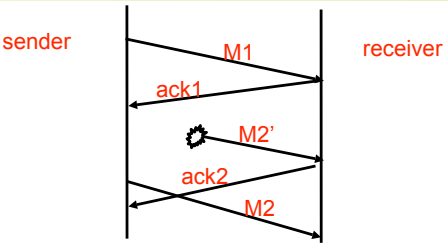
- Possible causes:
  - **"Congestion" in the network that delays packet.**
  - **A too-short timeout (how big should it be anyway?)**

Delayed acks take 2



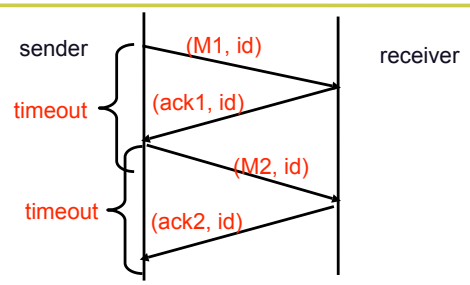
- What will the symptom be?

Spontaneous generation ("insertion")



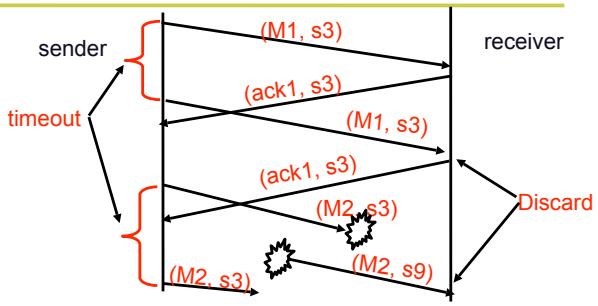
- Possible causes?
  - **Messages and ACKs are numbered 1, 2, 3, … A message M2' from a previous (closed) connection wanders around network and then shows up at end host.**

Summary of mechanisms



- Each message *logically* has two things:
  - **A message #, ordering msg with respect to all messages.**
  - **A "unique for all time" session id.**

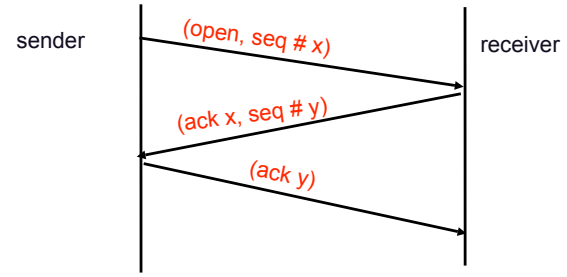
Summary: synthesized reliability



Core: timeout+retransmission to handle lost packets.

Use msg # to detect duplicate & reordered packets; associates ACK's with packets;

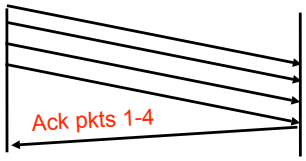
Connection setup: three way handshake



- TCP is a two-way (duplex) pipe
  - **There is a different sequence number for each direction**
  - **Asymmetric open: server does a passive "listen", client does an active "open" (sends first message).**

Sliding window algorithm

- We've been sending data: send, ack, send, ack, ...
  - **This is called a "stop and go protocol".**
  - **It's simple, but delivers really low bandwidth (each packet has a latency of least 1 round-trip time).**
  - **Sliding window allows multiple packets to be in flight.**

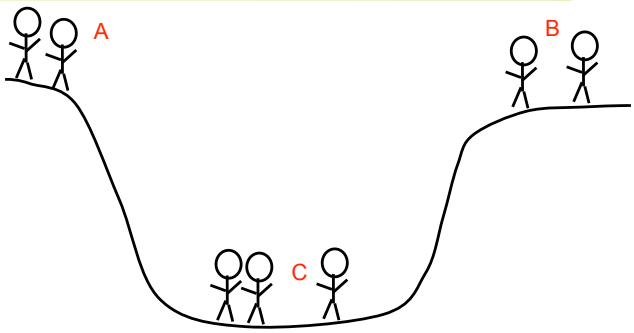


- Problem: have to make sure to not overwhelm receiver
  - **Have a "window" = # of bytes receiver is willing to buffer.**

What we can and cannot do

- We can
  - **Synthesize reliability on an unreliable network.**
  - **Eliminate jitter on a bursty network (given enough buffering at the receiver)**
- We cannot
  - **Recreate low-latency responses on a network that can induce arbitrary latencies.**
  - **Is there anything else?**
- An impossibility: consensus on an unreliable network
  - **Consensus: you and I agree on 1 bit of information.**
  - **A famous result: The two army problem.**

The two (stick figure) army problem

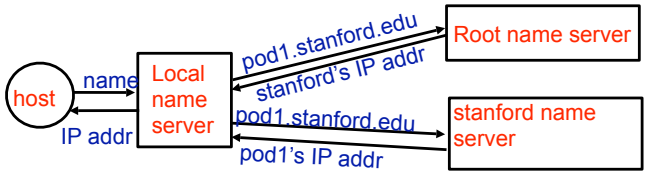


C can beat either A or B, but not both.

A and B want to agree on a time to simultaneously attack

Making names pretty

- IP addresses are not very memorable
  - **Internet uses the "domain name system" (DNS) to layer (sort of) human consumable names on top of them.**
  - **DNS type names: lcs.mit.edu, pod1.stanford.edu, etc.**
  - **hierarchical. Gives scalability and distributed naming.**
- Simplified resolving pod1.stanford.edu to IP address:



Protocol: naming + setup + guarantees

- Addressing: identifying receivers
  - **IP = host-to-host: every host has a unique IP address.**
  - **UDP & TCP = process-to-process: extend IP with a port that is coupled to a process within a host.**
  - **Telephone: "host-to-host": every connection point associated with a unique phone number.**
- Connection model:
  - **IP, UDP, Ethernet: connectionless. Send packet.**
  - **Telephone, ATM, TCP: connection-based. Explicitly set up connection before sending data. Tear down connection when done. (Makes richer service models easier.)**
- Data delivery model:
  - once data entrusted to protocol, what guarantees are made?

Some successful data delivery models

- Best-effort:
  - **IP: packets can be lost, delayed & packet data corrupted.**
  - **UDP: layers corruption detection on top of IP.**
  - **Ethernet: may lose packets, detects corruption.**
  - **ATM: can lose packets, detects corruption & makes reordering less likely.**
- Reliable, in-order byte stream: TCP
  - **Two-way byte stream. Prevents: loss, duplication, corruption, reorder.**
- Mostly reliable, quality-of-service: telephone
  - **Two-way voice service with small end-to-end delays.**
  - **Guarantees accepted calls will run to completion.**
  - **(easier to build on ATM than non-connection based)**

Future: challenging the assumptions of IP

- Globally unique addresses? Geographical routing?
- IPV4 address space "running out" (circa 1993)
  - IPV6 – bigger (128-bit) addresses. Huge! Nobody uses it.
  - Instead: NAT. Everyone on 10.x.x.x or 192.168.x.x
- Mobility
  - Want to move around and not lose connections.
  - Want access to (secure?) work network from home.
  - Want to keep same cell phone # in different city.
- Possible solutions? VLANs, VPNs, roaming...
  - **now subnet/area code isn't geographical**.
- Future: complex routing/switching for a complex world
  - IP address -> connection/virtual circuit identifier. Metro Enet.