

CS 140: Operating Systems Lecture 25: Network Layer

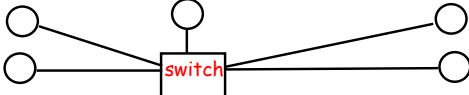
Mendel Rosenblum

Past & Present

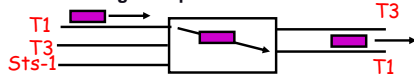
- ◆ Last time: pushing bits from with hardware
Link layer. How to: encode bits on wire, parse bits into packets, arbitrate between senders, name receivers
- ◆ Today: the network layer (portable bit pushing)
Network layer: given a packet, get it to the other side of a large (or HUGE) collection of networks.
Issue 1: portability. provides 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



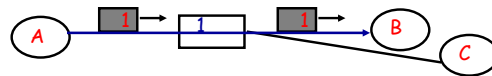
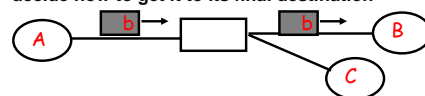
- ◆ Two parts:
Forwarding: take packets arriving on an input and forward them to the right output.



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

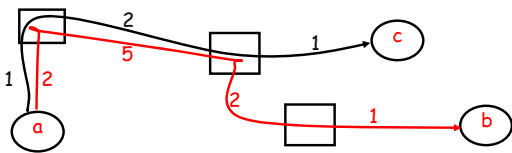
Two connection models

- ◆ 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)

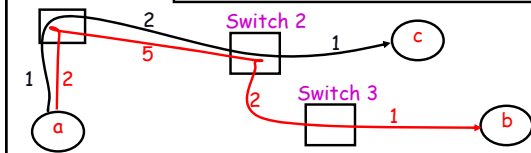
- ◆ Forming a circuit:
Send a connection request from A to B. Contains VCI + address of B.
Rule: VCI must be unique on the link its used on.
Switch creates an entry mapping input messages with VCI to output port.
Switch picks a new VCI unique between it and next switch.



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)



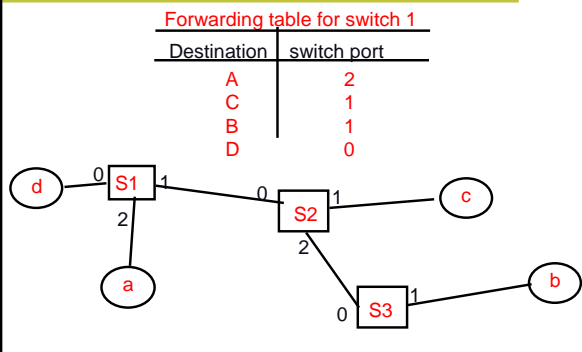
Virtual circuit issues

- ◆ 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

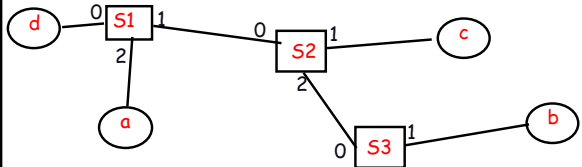


Datagram Tradeoffs

- ◆ Good:
 - No round-trip delay to setup connection.
 - Each packet forwarded independently of last: if switch or link fails, will 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 supports datagram (IP protocol)

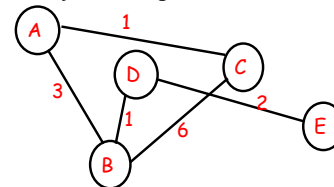
Some problems

- ◆ 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 $O(\text{hosts})$
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
- ◆ But doesn't work.
 - Real networks are just too big. K gets crushed.
 - Centralization is the enemy of scalability, so good routing protocols are distributed.

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

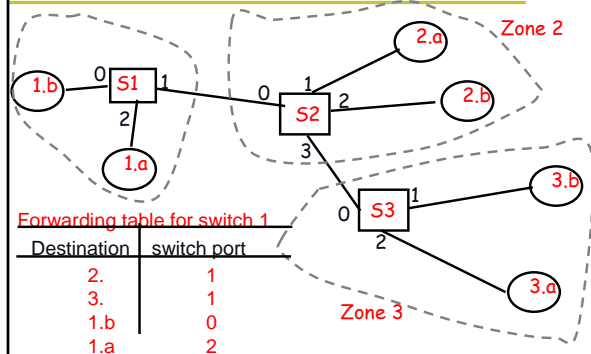
```

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



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 addresses are 32 bits. Are included in every IP pkt.
- ◆ Three classes: A, B, C
- | | | | | | | | | |
|---|-----|---------|-----|-----|---------|-------|-----|--------|
| 0 | net | host | 1 0 | net | host | 1 1 0 | net | host |
| 1 | 7 | 24 bits | 2 | 14 | 16 bits | 3 | 21 | 8 bits |

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 (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 reassembly packet as it traverses over different physical networks.
A global, network-wide address space.

Portable datagrams

- ◆ Every datagram carries enough information to forward packet
- | | | | |
|------|----------|----------|------|
| info | Src addr | dst addr | data |
|------|----------|----------|------|
- ◆ 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.

network id	host id
------------	---------

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.