Lecture 11: Routing

11.0 Main Points

Distance vector
Link state
Hierarchical routing

11.1 Motivation

How do we set up routing tables?

Use graph theory to compute "shortest path" between all pairs

net	graph
switches	nodes
links	edges
hops, bandwidth, delay, congestion, ...	cost

Why not: compute all to all shortest path once, program all switches?

Need dynamic route computation that adapts to changes in topology

Two choices:
1. distance vector (RIP, BGP) -- exchange routing tables with neighbors; no one knows complete topology!
    original Internet approach (RIP), but doesn't work!
    enhanced version (BGP), now used between admin domains

2. link state (OSPF) -- send everyone your routing table, everyone computes shortest path; not as scalable
    OSPF now used within admin domains

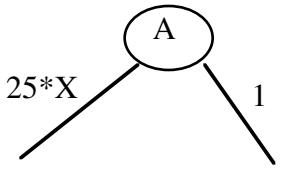
11.2 Distance Vector

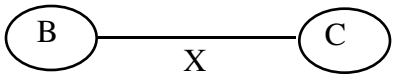
At each router, table of host, direction, distance (cost)

Initially, can only get to self, with 0 cost

Iterate
    exchange tables with neighbors
    if neighbor has lower cost, update table

What can go wrong?





what if A->C fails?

Lots of proposed solutions -- for example, never send table entry back to node that gave it to you. Unfortunately, this doesn't work in all cases (e.g., consider what happens when there is a loop -- can get the incorrect info from multiple directions).

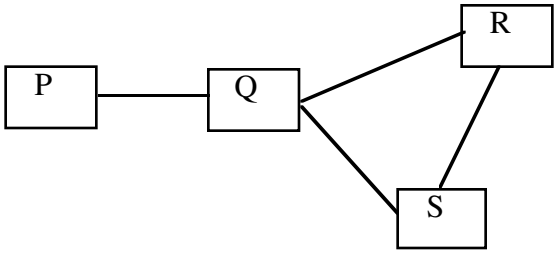
BGP (Border Gateway Protocol) -- at each router, store entire shortest path to each destination, and forward entire path in exchanges between neighbors.

By keeping the entire path (not just next hop), when a link is no longer available, will be able to tell which *paths* are invalidated.

11.3 Link State

1. each node gets complete topology, via reliable flooding

```
Each node determines who it can directly communicate with
Puts list of direct connections into numbered link state packet.
Forwards link state packet  to all neighbors
On receipt of a link state packet from neighbor Q
        if seen before, drop (throw out all but newest)
        else {
                process
                forward to all neighbors but Q
        }
```



If link state changes, generate a new link state message with a higher sequence #, and invoke reliable flooding.

Note that link state messages must be reliable -- if one gets lost, routing tables could be incorrect indefinitely. One solution: each node can periodically rebroadcast latest link state. Message will eventually get through.

2. given complete topology, each node computes shortest path

Idea is to build up list of nodes (M) where you know the shortest path; all other nodes are in candidate set where we don't know shortest path yet (N - M). Then iteratively add the node in N-M that has the shortest remaining path.

Current node, s, is shortest path to itself, so initially M = s
Iterate
    Find node x in M, node y in N-M, that minimizes the sum of the path from s to x plus the arc from x to y. This is the shortest path from s to any node in N-M, and therefore must be the shortest path to y.
    Add y to M
3. set table at each node to route in direction of shortest path

To sum:

Distance vector: node talks only to neighbors, but tells them everything it knows

Link state: node talks to everyone, but tells it only about its neighbors (what it knows for sure)

11.4 Hierarchical Routing

Does the entire Internet have to agree on a single routing algorithm? No!

Internet is segragated into cooperating autonomous systems (AS's).

Each AS can choose its own routing policy internally, usually link state.

BGP used to route between AS's -- path vector, with policy control.

By default, BGP optimizes for shortest number of AS's traversed. Doesn't correlate with hop count, latency, bandwidth, etc. For example, all of MCI's international network is a single AS.

Also, policy routing -- AS can choose to have its border routers follow any policy, for example, to favor cheaper, faster, or more reliable carrier, etc.

Why not base routing decisions on the performance of links, e.g., to avoid queuing delays?

Problem is stability of control loop; if route to avoid congestion, will cause congestion on new route! Adaptive routing abandoned in early Internet because couldn't make it stable.

11.5 Address Resolution Protocol

Internet is network of networks. Suppose I have a sender on one Ethernet, sending to a receiver on another, separated by some number of Internet routers.

Distance vector and link state protocols can get from one router to another; how do I get from the sender to the first router, and from the last router to the destination?

Sender broadcasts message on its LAN, what is the (Ethernet) address of my router? Router replies with its Ethernet address. Sender can then send packet with Ethernet destination = router, IP destination = receiver IP. Router gets packet, strips off Ethernet header, then routes IP packet.

At far end, router gets packet with IP address of destination. How does router know Ethernet address of destination?

Router broadcasts on LAN, what is Ethernet address corresponding to this IP address? Receiving host replies with its information, so router can build Ethernet frame to send to receiver.

ARPs are cached , so don't have to broadcast "where is" message for every Internet message arrival.

ARP caches are "soft state" -- information is timed-out periodically, to make sure the data stays up to date.

11.6 Router Table Size

How big do router tables get? Could be enormous.

Try #1: all nodes on the same LAN share the same prefix in their IP address. Route to LAN, then use ARP to get from last router to destination.

Thus, need only one entry for every LAN. Still too big!

Try #2: all LANs in same organization share the same prefix.

External routers only need table entries for each organization (roughly 50K separate autonomous systems); routers inside the organization need table entries for each of their local LANs, use ARP to get from last router to destination machine.

Routing table lookup = match longest prefix

11.7 IP Address Allocation

Originally, IP addresses split into 3 classes, with option for future expansion (e.g., multicast):

A: 0 | network (7 bits) | host (24 bits)

B: 10 | network (14 bits) | host (16 bits)

C: 110 | network (21 bits) | host (8 bits)

D: 1110 | multicast group # (28 bits)

Idea:

big organizations get class A addresses (1M hosts each)
medium organizations get class B (64K hosts each)
small organizations get class C addresses (255 hosts each)

Once asigned a network #, organization can do its own internal address assignment, without central control

IP address allocation issues:

1. we can run out of IP addresses

2. we'll run out faster if IP addresses are sparsely allocated
3. Need IP addresses to be aggregated to keep routing tables small
Four solutions:

1. Subnets -- split network addresses between multiple sites
2. Supernets -- assign adjacent network addresses to the same organization, or to physically close organizations

CIDR -- classless interdomain routing. Combine routing table entry whenever all nodes with the same prefix share next hop.

3. IP v6 -- increase IP address space to 128 bits.

Different packet format, selected by first few bits of IP header

Problem -- how do IPv6 systems interoperate with legacy IPv4 systems?

At first, few routers will understand IPv6, so two IPv6 nodes can talk via tunneling.

Also, map every IPv4 address to an IPv6 address.

Works for sending from a v6 machine to a v4 machine; but what about the reverse?

4. Network address translation (NAT)

Machines within organization are assigned unique addresses (relative to other machines in the same organization); can communicate normally with each other.

All communication with outside world via NAT; dynamically allocates an externally visible IP address.

NAT translates all packets; uses outside address on outgoing packets, uses internal address on incoming packets.

11.8 IP Fragmentation

IP packets traverse heterogeneous networks; each network can have a different maximum packet size (MTU).

Ethernet MTU ~ 1400 bytes
FDDI MTU ~ 4500 bytes
P2P ~ 532 bytes
ATM ~ 53 bytes (5 bytes header + 48 bytes data)

Aloha net ~ 80 bytes

Path is chosen dynamically -- source doesn't know path, and path can change from packet to packet, even to the same destination.

Try #1: always send packets no bigger than minimum size across all networks?

Try #2: Use MTU of local network; IP fragments if packet traverses a network with a smaller MTU.

IP fragments reassembled at destination, to keep routers simple.

Inefficient if routing algorithm chooses to route packets over network with smaller than expected MTU.

Try #3: MTU discovery

IP packet has "don't fragment bit". If need to fragment, will instead send back a message to the host. Source can use binary search to identify max packet size to a destination, cache result!

11.9 Wormhole routing and ATM cells

Store and forward: latency = RTT/2 + (packet size / link BW) * # of hops

Wormhole routing: latency = RTT/2 + packet size/link BW + epsilon

Reduces latency, but
- requires complex hardware support in the router
- need to restrict routes, add buffering to avoid deadlock
- limits scheduling flexibility (e.g., real time)

Fragment into smaller chunks (e.g., ATM uses 53 byte cells)
- each cell is store and forward, but can forward first cell before last arrives

ATM supports variable sized packets by assembling sequence of fixed size cells; fixed size cells simplify switch design (every link scheduled synchronously)

What should fragment size be?
- too small => high overhead
- too large => store and forward delay