Routing

from Kurose's slides

Routing protocols

Goal: set routing tables for packet forwarding in hosts and routers, typically based on some optimality criterion.

Questions:

- who determines entries?
- based on what information (hops, delay, cost, …) ?
- how often does it change (hop vs. delay)?
- where is routing information stored?
- algorithm used to compute routes?

Goals for routing algorithms

- scalability
- "safe" interconnection of different organizations
- adopt quickly to changes in topology
- avoid routing loops or at least terminate them quickly
- self-healing, robust
- efficient: can't use 90% of bandwidth for routing info
- multiple metrics (QOS, price, politics, …) ➠ not yet
- routes should be (near) "optimal"
- can't have all hosts/networks in single table ➠ hierarchical

Routing algorithms

- centralized vs. decentralized
  - *centralized*: a central site computes and distributes routes or information to compute routes
  - *decentralized*: each router sees only local information
- static vs. adaptive
  - *static*: routing tables change very slowly, often in response to human intervention (German X.25)
  - *adaptive*: routing tables change with traffic or topology
- intra-domain vs. inter-domain
  - *intra-domain*: one administration ➠ fewer rules, changes?, **not** smaller
  - *inter-domain*: between administrations (*autonomous systems*) ➠ security, larger geographic reach

Internet Routing

- inter-domain: BGP, about 3,000 AS, 97,000 networks,
- about 32,000 active routes in Merit routing arbiter ($\subset$ Internet Routing Registry)

fokus

Link state (LS) routing

- each node knows cost associated with each of its outgoing links:
  - queueing delay on link, instantaneous or time-averaged
  - bandwidth of link
  - cost ($): leased line vs. dial-up
  - notion of desirability
  - simply one “hop” per link
- quasi-centralized: link costs periodically broadcast to all routers
- least-cost path from source to all other nodes ➠ Dijkstra’s shortest-path algorithm
- used in OSPF (+ ISO IS-IS)

fokus

Dijkstra's algorithm

N : set of all nodes to which we know shortest path; initially empty.

$d(v)$: distance (cost) of known least cost path from source to v

$c(i, j)$: cost of link from node i to j; $c(i, j) = \infty$ if not directly connected

$p(v)$: predecessor node (closest neighbor of v) along shortest path from source to v

After k steps, we know shortest path to nearest k neighbors from source.

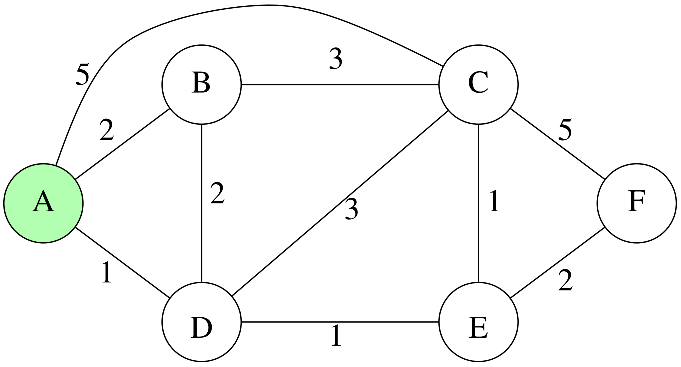
Find known nearest neighbor and see if we can reach others from that neighbor by a shorter route than previously. Using nearest ensures that there can be no shorter path.

Dijkstra's algorithm

1. **Initialization**
   $N = \{A\}$
   for all nodes v:
      if v adjacent to A
         then $d(v) = c(A, v)$
         else $d(v) = \infty$

2. **loop**
      find node w not in set N such that $d(w)$ is smallest
      add w into N
         update $d(v)$ for all v not in N:
         $d(v) = \min\{d(v), d(w) + c(w, v)\}$
   until all nodes are in N

Example of Dijkstra's algorithm



Example of Dijkstra's algorithm

		distance from A to …				
step	N	$d(B), p(B)$	$d(C), p(C)$	$d(D), p(D)$	$d(E), p(E)$	$d(F), p(F)$
0	A	2,A	5,A	1,A	$\infty, -$	$\infty, -$
1	AD	2,A	4,D		2,D	$\infty, -$
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					

- example (step 1): $d(\text{C}) \rightarrow d(\text{D}) + c(\text{D}, \text{C}) = 1 + 3 < 5$
- for each column, last entry gives immediate neighbor along least-cost path to/from A, and cost to that destination
- worst case running time: $O(n^2)$ per source node: n steps, $n - 1$ comparisons

Example of Dijkstra's algorithm (reordered)

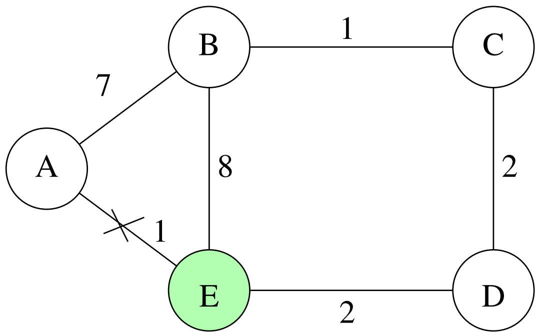
		distance from A to ...				
step	N	$d(B), p(B)$	$d(C), p(C)$	$d(D), p(D)$	$d(E), p(E)$	$d(F), p(F)$
0	A	2,A	5,A	1,A	$\infty, -$	$\infty, -$
1	AD	2,A	4,D		2,D	$\infty, -$
2	ADB		3,E		2,D	4,E
3	ADBE		3,E			4,E
4	ADBEC					4,E
5	ADBECF					

➠ no change

➠ exercise: use asymmetric weights

Distance vector (DV) routing

- asynchronous, iterative distributed computation
  - computation step
  - exchange of routing information step



E's view:

destination $D^E()$	cost from E to destination via A	B	D
A	$\boxed{1}$	14	5
B	7	8	$\boxed{5}$
C	6	9	$\boxed{4}$
D	4	11	$\boxed{2}$

Distance table:

- per-node table with cost to all other nodes via each neighbor
- $D^E(\mathrm{A}, \mathrm{B})$ gives cost from E to A, via link to B
- here, $D^E(\mathrm{A}, \mathrm{B}) = 14$
- distance table immediately gives routing table:
  - minimum cost to each destination (row) is smallest value in row
  - column containing minimum value gives outgoing link for routing to that destination

Distributed, asynchronous shortest path algorithm

- based on Bellman-Ford algoritm
- used in many routing protocols: BGP, ISO IDRP, Novell IPX, RIP
- run by each node
- exchange (node,distance) information with network neighbor only
- find
  - least cost path to every other node
  - next (neighboring) node on least cost path to destination ➠ not complete path

DV algorithm (at node X):

1. initialization:
   for all adjacent nodes (column) v:
   $D(*, v) = \infty$
   $D(v, v) = c(X, v)$

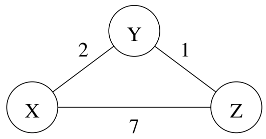
2. **loop**
   execute distributed topology update procedure
   until hell freezes over

Topology update algorithm

At node X:

1. wait until X sees link cost change to Y, or receives message from neighbor
2. if $c(X, Y)$ changes by δ (cost to neighbor has changed):
   change all column-Y entries in distance table by δ
   if this changes cost of best path to Z, inform neighbors
3. if control message from W ➠ shortest path via W to some Z has changed
   $D^X(Z, W) = c(X, W) +$ new distance W to Z
   if cost of best path to Z has changed, inform neighbors

Distance vector routing: example



	via	
D^X	Y	Z
Y	∞	∞
Z	∞	∞

	via	
D^X	Y	Z
Y		∞
Z	∞	

	via	
D^X	Y	Z
Y		
Z		

D^Y	X	Z
X	∞	∞
Z	∞	∞

D^Y	X	Z
X		∞
Z	∞	

D^Y	X	Z
X		
Z		

D^Z	X	Y
X	∞	∞
Y	∞	∞

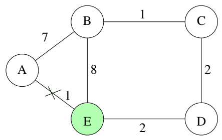
D^Y	X	Y
X		∞
Y	∞	

D^Z	X	Y
X		
Y		

Distance vector routing: recovery from link failure

- if link XY fails, X and Y set $c(X, Y)$ to ∞ and run topology update algorithm
- "good news travels fast, bad news travels slowly"
- looping:
  - inconsistent routing tables: to A, D $\rightarrow$ E, E $\rightarrow$ D.
  - loops disappear eventually
  - performance degradation during looping
  - out-of-order end-end delivery possible

Distance vector routing with link failures



shortest path to A and next node; $\boxed{\cdot}$: new shortest path

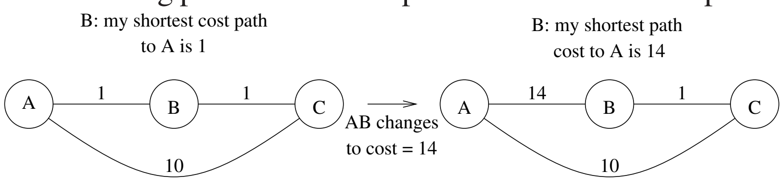
	as seen from			
	B	C	D	E
initially	6,C	5,D	3,E	1,A
AE ↓	6,C	5,D	3,E ⟷	$\boxed{5,D}$
step 1	6,C	5,D	$\boxed{7,E}$ ⟷	5,D
step 2	6,C ⟷	$\boxed{7,B}$	7,E ⟷	$\boxed{9,D}$
step 3	$\boxed{7,A}$	7,B	$\boxed{9,C}$	9,D
step 4	7,A	$\boxed{8,B}$	9,C	$\boxed{11,D}$
step 5	7,A	8,B	$\boxed{10,C}$	11,D
step 6	7,A	8,B	10,C	$\boxed{12,D}$

Distance vector routing: split horizon algorithm

- change topology update algorithm to avoid count-to-infinity problem
- if A routes to Z via B, then A tells B that its distance to Z is ∞
- example A — B — Z: B will not route to Z via A if link BZ fails
- will not always avoid count-to-infinity problem

Distance vector routing: hold down algorithm

- when shortest path cost change:
  - start "hold down" timer
  - advertise new shortest path cost
    as the new cost along previous shortest path route until timer expires:



  - will force large costs to propagate quickly
  - will not always avoid count-to-infinity problem

Comparison of LS and DV algorithms

- "LS is better": DV requires iteration with messages being exchanged at each iteration
- "DV is better": if link costs changes do not affect shortest paths, no messages exchanged

Robustness of LS and DV algorithms

- what happens if router fails, misbehaves, or is sabotaged?
- link state could:
  - report incorrect distance to all neighbors
  - corrupt or lose any LS broadcast messages passing through it
  - report incorrect neighbors
- distance vector could:
  - advertise incorrect shortest distance to any/all destinations ("from me, zero hops to everywhere")
  - report incorrect neighbors

Convergence of LS and DV algorithms

- want to keep network routes stable as often as possible
- distance vector:
  - may iterate many times while converging
  - can suffer from loops and oscillations
  - cannot propagate new information from other routers until it recomputes new routes
- link state
  - requires one broadcast pro node
  - can suffer from oscillations

GMD fokus

RIP (Routing Information Protocol)

- RFC 1058 (1988)
- intra-domain only
- distance vector algorithm with split horizon
- metric: hop count (maximum 15 ➠ limited network size)
- distance vectors exchanged via UDP port 520 every 30 seconds
- `routed` daemon
- RIP-2 (RFC 1388, RFC 1387, 1993): subnet masks, route tag to identify external routes, authentication

RIP-2 packet header

0	8	16	24	32
Command (1 or 2)	Version (2)	routing domain (AS)		
address family identifier (AFI)		route tag		
IP address				one per destination
subnet mask				
next hop IP address				
metric (0..15)				

OSPF

- open (= non-proprietary) shortest path first (RFC 1247, 1991)
- link state routing using Dijkstra's algorithm
- reliable flooding with sequence numbers, aging
- two-level hierarchy: backbone and attached areas
- allows level-2 routers to send path cost to level-1 routers
- handles network partitioning (somehow…)
- uses IP packets to communicate

BGP

- inter-domain routing protocol
- uses TCP
- exchanges paths: list of transit AS, networks, properties

netstat: inspect routing table

```
netstat -r
routing tables
Destination     Gateway         Flags  Refcnt Use     Interface
localhost       localhost       UH     3      7013    lo0
default         gmdbgate        UG     0      107416  le0
gmd             129.26.216.231  U      0      19      qaa1
gmd-fokus       atmos           U      33     211181  le0
fokus-atm       atmos           U      1      561634  qaa0
bali.de         atmos.bali.de   U      0      1487638 fa0

netstat -rn
Routing tables
Destination      Gateway          Flags Refcnt Use       Interface
127.0.0.1        127.0.0.1        UH    3      7521      lo0
default          192.35.149.248   UG    0      107452    le0
129.26.0.0       129.26.216.231   U     0      19        qaa1
192.35.149.0     192.35.149.117   U     35     215346    le0
193.175.134.0    193.175.134.117  U     1      561641    qaa0
194.94.246.0     194.94.246.65    U     0      1487639   fa0
```

Flags: U = up, G = gateway, D = redirect