9:
Intro to Routing Algorithms

Last Modified:
3/24/2003 2:08:40 PM

Routing

- IP Routing – each router is supposed to send each IP datagram one step closer to its destination
- How do they do that?
  - Static Routing
    - Hierarchical Routing – in ideal world would that be enough? Well its not an ideal world
  - Dynamic Routing
    - Routers communicate amongst themselves to determine good routes (ICMP redirect is a simple example of this)
    - Before we cover specific routing protocols we will cover principles of dynamic routing protocols

Routing Algorithm classification: Static or Dynamic?

Choice 1: Static or dynamic?

Static:
- routes change slowly over time
- Configured by system administrator
- Appropriate in some circumstances, but obvious drawbacks (routes added/removed? sharing load?)
- Not much more to say?

Dynamic:
- routes change more quickly
  - periodic update
  - in response to link cost changes

Routing Algorithm classification: Global or decentralized?

Choice 2, if dynamic: global or decentralized information?

Global:
- all routers have complete topology, link cost info
- "link state" algorithms

Decentralized:
- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors (gossip)
- "distance vector" algorithms

Roadmap

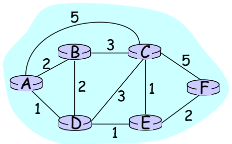
- Details of Link State
- Details of Distance Vector
- Comparison

Routing

Routing protocol

Goal: determine "good" path (sequence of routers) thru network from source to dest.

Graph abstraction for routing algorithms:
- graph nodes are routers
- graph edges are physical links
  - link cost: delay, $ cost, or congestion level

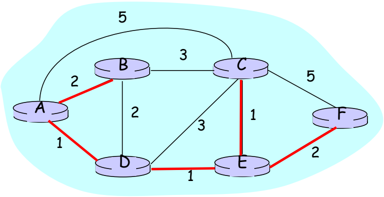


- "good" path:
  - typically means minimum cost path
  - other definitions possible

Global Dynamic Routing

See the big picture; Find the best Route

What algorithm do you use?



A Link-State Routing Algorithm

Dijkstra's algorithm

- Know complete network topology with link costs for each link is known to all nodes
  - accomplished via "link state broadcast"
  - In theory, all nodes have same info
- Based on info from all other nodes, each node individually computes least cost paths from one node ('source") to all other nodes
  - gives routing table for that node
- iterative: after k iterations, know least cost path to k dest.'s

Link State Algorithm: Some Notation

Notation:

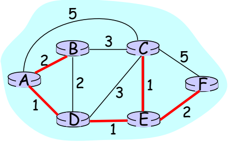
- c(i,j): link cost from node i to j. cost infinite if not direct neighbors
- D(v): current value of cost of path from source to dest. V
- n(v): next hop from this source to v along the least cost path
- N: set of nodes whose least cost path definitively known

Dijsktra's Algorithm

```
Initialization – know c(I,j) to start:
  N = {A}
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v)
      else D(v) = infty

Loop
   find w not in N such that D(w) is a minimum (optional?)
   add w to N
   update D(v) for all v adjacent to w and not in N:
      D(v) = min( D(v), D(w) + c(w,v) )
   /* new cost to v is either old cost to v or known
    shortest path cost to w plus cost from w to v */
until all nodes in N
```

Dijkstra's algorithm: example

Step	start N	D(B),n(B)	D(C),n(C)	D(D),n(D)	D(E),n(E)	D(F),n(F)
0	A	2,B	5,C	1,A	infinity	infinity
1	AD	2,B	4,D		2,D	infinity
2	ADE	2,B	3,D			4,D
3	ADEB		3,D			4,D
4	ADEBC					4,D
5	ADEBCF					



Dijkstra's Algorithm gives routing table

A	Outgoing Link
A	n(A) = A
B	n(B) = B
C	n (C)= D
D	n(D) = D
E	n(E) = D
F	n(F) = D

destination

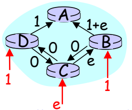
Complexity of Link State

Algorithm complexity: n nodes

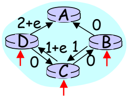
- each iteration
  - Find next w not in N such that D(w) is a minimum
  - Then for that w, check its best path to other destinations
  - => n*(n+1)/2 comparisons: O(n^2)
- more efficient implementations possible using a heap: O(nlogn)

Oscillations

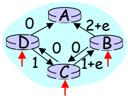
- Assume:
  - Link cost = amount of carried traffic
  - Link cost is not symmetric
  - B and D sending 1 unit of traffic; C send e units of traffic
- Initially start with slightly unbalanced routes
- Everyone goes with least loaded, making them most loaded for next time, so everyone switches
- Herding effect!



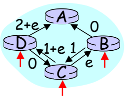
Initially start with almost equal routes



… B and C go clockwise to A



… B, C and D go counterclockwise



… B,C,D go clockwise

Preventing Oscillations

- Avoid link costs based on experienced load
  - But want to be able to route around heavily loaded links…
- Avoid "herding" effect
  - Avoid all routers recomputing at the same time
  - Not enough to start them computing at a different time because will synchronize over time as send updates
  - Deliberately introduce randomization into time between when receive an update and when compute a new route

---

Distance Vector Routing Algorithm

distributed:

- each node communicates *only* with directly-attached neighbors

iterative:

- continues until no nodes exchange info.
- *self-terminating*: no "signal" to stop

asynchronous:

- nodes need *not* exchange info/iterate in lock step!

Distance Vector Routing Algorithm

Distance Table data structure

- each node has its own row for each possible destination
- column for each directly-attached neighbor to node
- example: in node X, for dest. Y via neighbor Z:

$D^X(Y,Z)$ = distance *from* X *to* Y, *via* Z as next hop

$= c(X,Z) + \min_w \{D^Z(Y,w)\}$

Column only for each neighbor

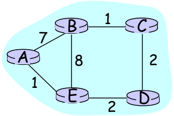
cost to destination via

$D^X()$	Z
Y	$D^x(Y,Z)$

destination

Rows for each possible dest !

Example: Distance Table for E



D^E(row, col)

$D^E(C,D) = c(E,D) + \min_w \{D^D(C,w)\}$
$= 2+2 = 4$

$D^E(A,D) = c(E,D) + \min_w \{D^D(A,w)\}$
$= 2+3 = 5$ Loop back through E!

$D^E(A,B) = c(E,B) + \min_w \{D^B(A,w)\}$
$= 8+6 = 14$ Loop back through E!

Column only for each neighbor

cost to destination via

$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

destination

Rows for each possible dest !

Distance table gives routing table

◯ = least cost

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

		Outgoing link to use, cost
destination	A	A,1
	B	D,5
	C	D,4
	D	D,4

Distance table → Routing table

Distance Vector Routing: overview

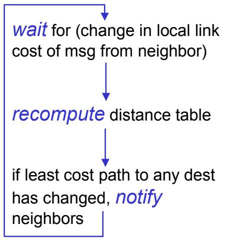
Iterative, asynchronous:
each local iteration caused by:

- local link cost change
- message from neighbor: its least cost path change from neighbor

Distributed:

- each node notifies neighbors *only* when its least cost path to any destination changes
  - neighbors then notify their neighbors if necessary

Each node:



wait for (change in local link cost of msg from neighbor)

recompute distance table

if least cost path to any dest has changed, *notify* neighbors

---

Distance Vector Algorithm:

At all nodes, X:

```
Initialization (don't start knowing link costs for all links in graph):
 for all adjacent nodes v:
    $D^X(*,v)$ = infty        /* the * operator means "for all rows" */
    $D^X(v,v) = c(X,v)$
 for all destinations, y
    send $\min_w D^X(y,w)$ to each neighbor  /* w over all X's neighbors */
```

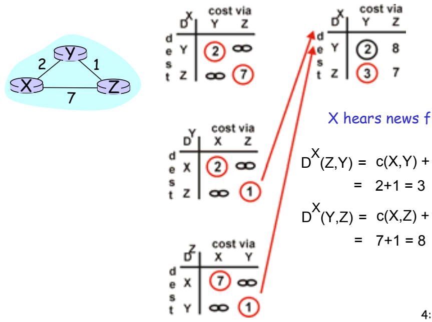
Then in steady state…

Distance Vector Algorithm (cont.):

```
loop
  wait (until I see a link cost change to neighbor V
       or until I receive update from neighbor V)

  if (c(X,V) changes by d)
    /* change cost to all dest's via neighbor v by d */
    /* note: d could be positive or negative */
    for all destinations y:  $D^X(y,V) = D^X(y,V) + d$

  else if (update received from V wrt destination Y)
    /* shortest path from V to some Y has changed  */
    /* V has sent a new value for its  $\min_w DV(Y,w)$ */
    /* call this received new value is "newval"     */
    for the single destination y: $D^X(Y,V) = c(X,V) +$ newval

  if we have a new $\min_w D^X(Y,w)$for any destination Y
     send new value of $\min_w D^X(Y,w)$ to all neighbors

forever
```

Distance Vector Algorithm: example

To start just know directly connected links…tell neighbors



	D^X	cost via	
		Y	Z
dest	Y	②	∞
	Z	∞	⑦

	D^Y	cost via	
		X	Z
dest	X	②	∞
	Z	∞	①

	D^Z	cost via	
		X	Y
dest	X	⑦	∞
	Y	∞	①

	D^X	cost via	
		Y	Z
dest	Y	②	8
	Z	③	7

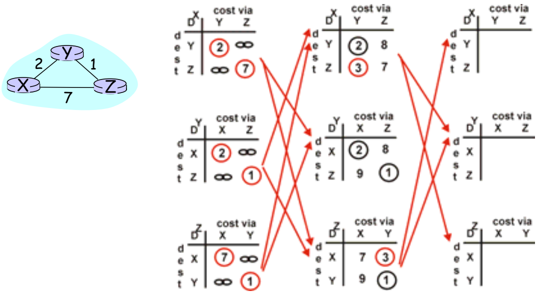
X hears news from Y and Z

$$D^X(Z,Y) = c(X,Y) + \min_w\{D^Y(Z,w)\}$$
$$= 2+1 = 3$$

$$D^X(Y,Z) = c(X,Z) + \min_w\{D^Z(Y,w)\}$$
$$= 7+1 = 8$$

Distance Vector Algorithm: example

In steady state, when have good news tell neighbor



	D^X	cost via	
		Y	Z
dest	Y	②	∞
	Z	∞	⑦

	D^Y	cost via	
		X	Z
dest	X	②	∞
	Z	∞	①

	D^Z	cost via	
		X	Y
dest	X	⑦	∞
	Y	∞	①

	D^X	cost via	
		Y	Z
dest	Y	②	8
	Z	③	7

	D^Y	cost via	
		X	Z
dest	X	②	8
	Z	9	①

	D^Z	cost via	
		X	Y
dest	X	7	③
	Y	9	①

	D^X	cost via	
		Y	Z
dest	Y		
	Z		

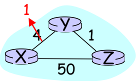
	D^Y	cost via	
		X	Z
dest	X		
	Z		

	D^Z	cost via	
		X	Y
dest	X		
	Y		

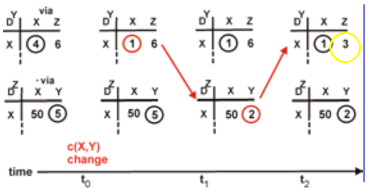
Distance Vector: link cost changes

Link cost changes:

- node detects local link cost change
- updates distance table (line 15)
- if cost change in least cost path, notify neighbors (lines 23,24)



"good news travels fast"



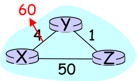
algorithm terminates

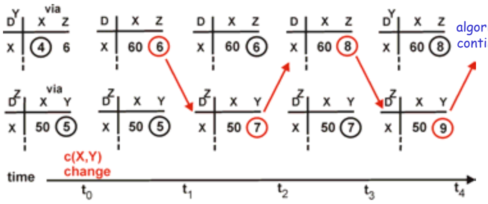
Anyone see a problem?

Distance Vector: link cost changes

Link cost changes:

- good news travels fast
- bad news travels slow - "count to infinity" problem!



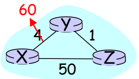


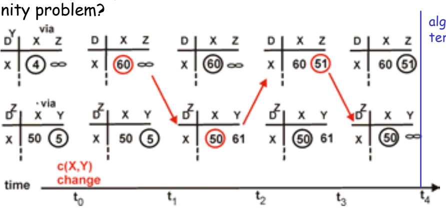
algorithm continues on!

Distance Vector: poisoned reverse

If Z routes through Y to get to X :

- Originally, Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- In end, Y tells Z infinity
- will this completely solve count to infinity problem?



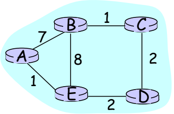


algorithm terminates

Bigger Loops and Poison Reverse

$D^E(A,D) = c(E,D) + \min_w \{D^D(A,w)\}$

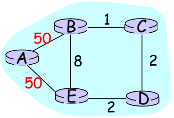
$= 2+3 = 5$



Loop back through E! Poison reverse will fix this
D tells E infinity because D's route to A through E

$D^E(A,B) = c(E,B) + \min_w \{D^B(A,w)\}$

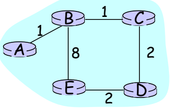
$= 8+6 = 14$

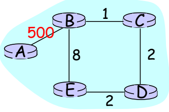


Loop back through E! Poison reverse will not fix this
B's route to A is through E but B doesn't know that so does not tell E infinity
B's route is through C so no poison reverse
E will try to send through B

Count to Infinity Example with Bigger Loop

B will learn bad news
C will have told B infinity because its route to A is through B, so B won't reroute through C
E however will have told B about a good route to A through D (cost 6)
B will choose that route instead and advertise it as the new best to C (cost 6+8 = 14); it will be worse than the old one it advertised to C (old cost = 1)
C will propagate this updated "best" route to D (cost 15)
D will propagate this new "best" route to E (cost 17)
E will update the "best" route to B (cost 19)
Last time it advertised cost 6 to B
It will loop around adding 13 each time (cost of loop)
Will continue until cost E advertises to B is bigger than 500





Comparison of LS and DV algorithms

Message complexity

- LS: nodes send info on directly connections to all other nodes
  - More, smaller messages
- DV: nodes send info on best paths to all destinations to neighbors
  - Fewer, larger messages

Speed of Convergence

- LS: O(n^2) algorithm
  - may have oscillations
- DV: convergence time varies
  - may be routing loops
  - count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagate thru network