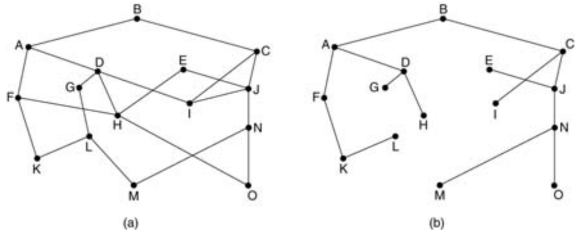
Digital Communication in the Modern World

Network Layer: Routing Classifications; Shortest Path Routing

http://www.cs.huji.ac.il/~com1
com1@cs.huji.ac.il

Some of the slides have been borrowed from:
Computer Networking: A Top Down Approach Featuring the Internet,
2nd edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2002.

Network Layer's main problem: To get efficiently from one point to the other in a dynamic environment



(a)

(b)

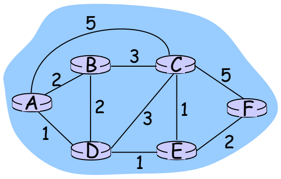
Routing

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 def's possible (min. num of links)

Datagram Routing Algorithm Classification

- Global (Link State) Routing
  - Shortest Path routing
    - Dijkstra routing
- Decentralized
  - Distance Vector routing
- Hierarchical Routing
  - Broadcast routing
  - Multicast routing

A Link-State Routing Algorithm

Dijkstra's algorithm

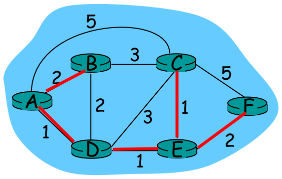
- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ('source") to all other nodes
  - gives routing table for that node
- iterative: after k iterations, know least cost path to k dest.'s

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
- p(v): predecessor node along path from source to V, that is next v
- N: set of nodes whose least cost path definitively known

Dijkstra's Algorithm

```
Initialization:
 N = {A}
 for all nodes v
   if v adjacent to A
     then D(v) = c(A,v)
     else D(v) = infinity

Loop
  find w not in N such that D(w) is a minimum
   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 in C

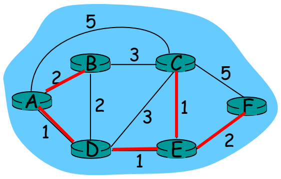
```
#define MAX  NODES 1024              /* maximum number of nodes */
#define INFINITY 1000000000          /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];/* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                     /* the path being worked on */
      int predecessor;               /* previous node */
      int length;                    /* length from source to this node */
      enum {permanent, tentative} label; /* label state */
  } state[MAX_NODES];

  int i, k, min;
  struct state *p;

  for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
      p->predecessor = –1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0;  state[t].label = permanent;
  k = t;                             /* k is the initial working node */
```

Dijkstra's Algorithm in C

```
  do {                                     /* Is there a better path from k? */
     for (i = 0; i < n; i++)               /* this graph has n nodes */
         if (dist[k][i] != 0 && state[i].label == tentative) {
             if (state[k].length + dist[k][i] < state[i].length) {
                 state[i].predecessor = k;
                 state[i].length = state[k].length + dist[k][i];
             }
         }

     /* Find the tentatively labeled node with the smallest label. */
     k = 0; min = INFINITY;
     for (i = 0; i < n; i++)
         if (state[i].label == tentative && state[i].length < min) {
             min = state[i].length;
             k = i;
         }
     state[k].label = permanent;
  } while (k != s);

  /* Copy the path into the output array. */
  i = 0;  k = s;
  do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

Dijkstra's algorithm: example

Step	start 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	infinity	infinity
1	AD	2,A	4,D		2,D	infinity
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					



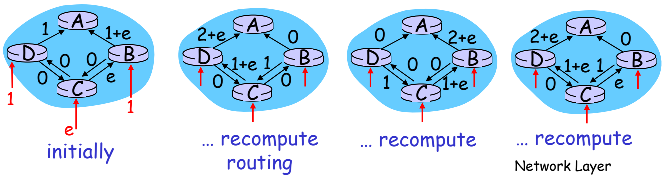
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

- ❐ each iteration: need to check all nodes, w, not in N
- ❐ $\sum_{i=1}^{n-1} n-i = \frac{n(n+1)}{2} = O(n^2)$
- ❐ more efficient implementations possible: O(nlogn)

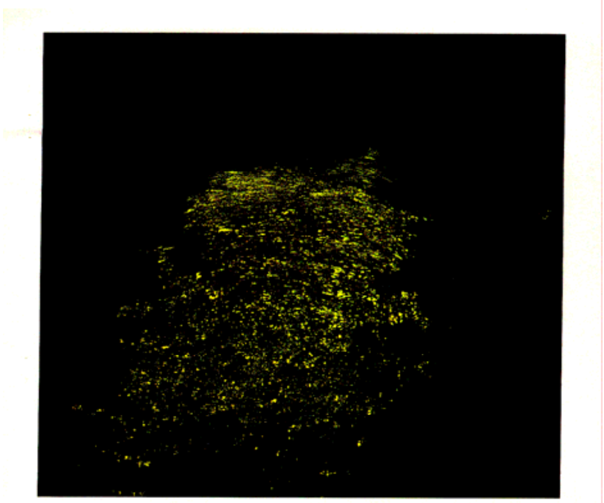
Oscillations possible:

- ❐ e.g., if link cost = amount of carried traffic



Spontaneous synchronization

- ❐ To avoid oscillations make the routers recompute&send the link costs at different times?
- ❐ Turns out that if the recomputation periodicity is more or less the same on all routers then they eventually synchronize their execution times!
- ❐ The phenomenon of spontaneous synchronization occurs in physics, biology, chemistry, sociology, medicine, etc.



SCIENTIFIC AMERICAN December 1993 69

Shortest Path Routing Summary

Each router does the following:

- Discover its neighbors, learn their network address and UP state (HELLO message)
- Measure the delay or cost to each of its neighbors (ECHO message or cost function)
- Construct a packet telling what it knows (LS message)
- Send this packet to all other routers (every ROUTE REFRESH INTERVAL)
- Compute the shortest path to every other router (Dijkstra)

Shortest Path Routing Summary

Moreover:

- On every Link State change flood LS to all other routers
- Avoid oscillations through different periods
- Keep LS message counter to keep flooding in check
- Keep LS message age to keep counter in check
- Counter and age also used for fault tolerance

Distance Vector Routing Algorithm

- ❒ Each node communicates only with directly-attached neighbors
- ❒ Measures the cost to the directly-attached neighbors only
- ❒ Estimates the cost to the rest of the nodes
- ❒ Each node has a vector with the estimated cost to every other node
- ❒ Info change done with the neighbors every time period
- ❒ Vector updated according to neighbors routing table

Distance Vector Routing Algorithm

iterative:

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

asynchronous:

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

distributed:

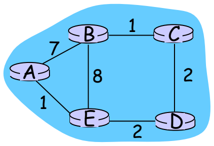
- ❒ each node communicates *only* with directly-attached neighbors

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:

$$D^X(Y,Z) = \text{distance \textit{from} X \textit{to} Y, \textit{via} Z as next hop} = c(X,Z) + \min_w\{D^Z(Y,w)\}$$

Distance Table: example



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

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

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

cost to destination via

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

Distance table gives routing table

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

destination

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

destination

Distance table → Routing table

Distance Vector Routing: overview

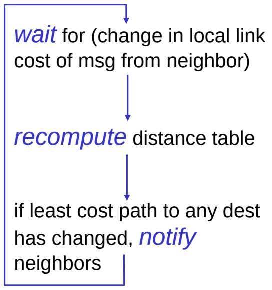
Iterative, asynchronous: each local iteration caused by:

- 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:

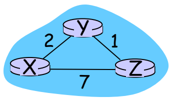
At all nodes, X:

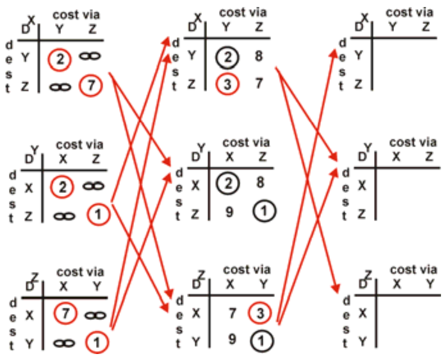
```
Initialization:
 for all adjacent nodes v:
    $D^X(*,v)$ = infinity        /* the * operator means "for all rows" */
    $D^X(v,v)$ = c(X,v)
 for all destinations, y
    send $\min_w D^X(y,w)$ to each neighbor  /* w over all X's neighbors */
```

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





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

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

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

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

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

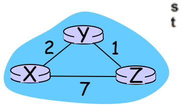
D^Z	cost via X	Y
dest X	7	3
dest Y	9	1

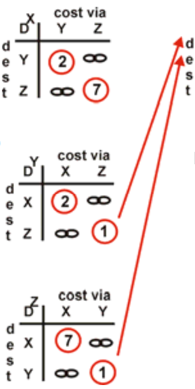
D^X	cost via Y	Z
dest Y		
dest Z		

D^Y	cost via X	Z
dest X		
dest Z		

D^Z	cost via X	Y
dest X		
dest Y		

Distance Vector Algorithm: example





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

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

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

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

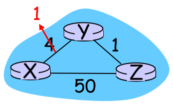
$$D^X(Y,Z) = c(X,Z) + \min_w\{D^Z(Y,w)\}$$
$$= 7+1 = 8$$

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

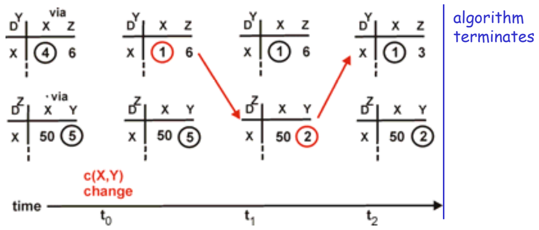
Distance Vector: link cost changes

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)



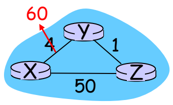
"good news travels fast"

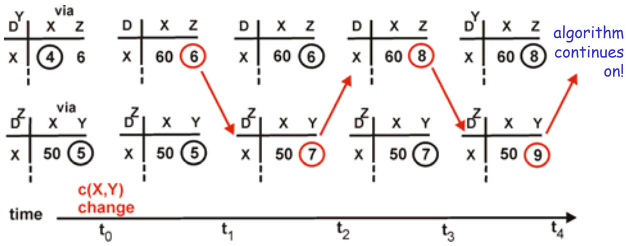


Distance Vector: link cost changes

Link cost changes:

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

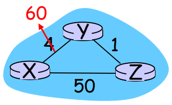


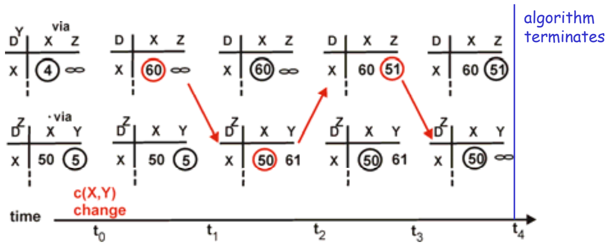


Distance Vector: poisoned reverse

If Z routes through Y to get to X :

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





Comparison of LS and DV algorithms

Message complexity

- LS: with n nodes, E links, O(nE) msgs sent each
- DV: exchange between neighbors only
  - convergence time varies

Speed of Convergence

- LS: O(n^2) algorithm requires O(nE) msgs
  - 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

Hierarchical Routing

Our routing study thus far - idealization

- all routers identical
- network "flat"

… *not* true in practice

scale: with 200 million destinations:

- can't store all dest's in routing tables!
- routing table exchange would swamp links!

administrative autonomy

- internet = network of networks
- each network admin may want to control routing in its own network

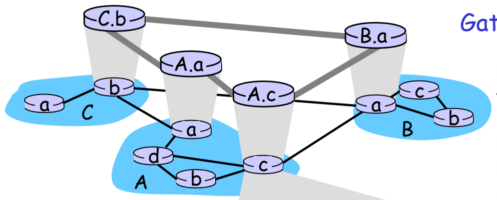
Hierarchical Routing

- aggregate routers into regions, "autonomous systems" (AS)
- routers in same AS run same routing protocol
  - "intra-AS" routing protocol
  - routers in different AS can run different intra-AS routing protocol

gateway routers

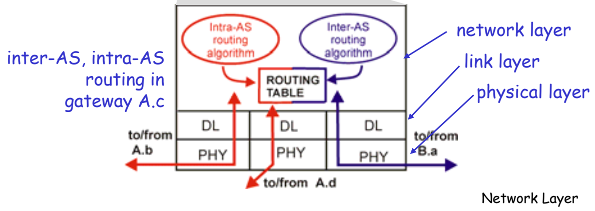
- special routers in AS
- run intra-AS routing protocol with all other routers in AS
- *also* responsible for routing to destinations outside AS
  - run *inter-AS routing* protocol with other gateway routers

Intra-AS and Inter-AS routing

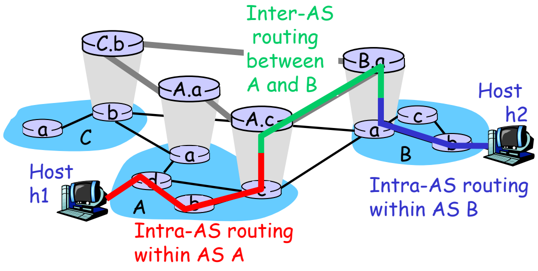


Gateways:

- •perform inter-AS routing amongst themselves
- •perform intra-AS routers with other routers in their AS



Intra-AS and Inter-AS routing


Inter-AS routing between A and B
C.b
B.a
A.a
A.c
a
b
C
a
c
b
B
Host h2
Host h1
a
d
b
c
A
Intra-AS routing within AS B
Intra-AS routing within AS A

- ❒ We'll examine specific inter-AS and intra-AS Internet routing protocols (RIP, BGP, OSPF)