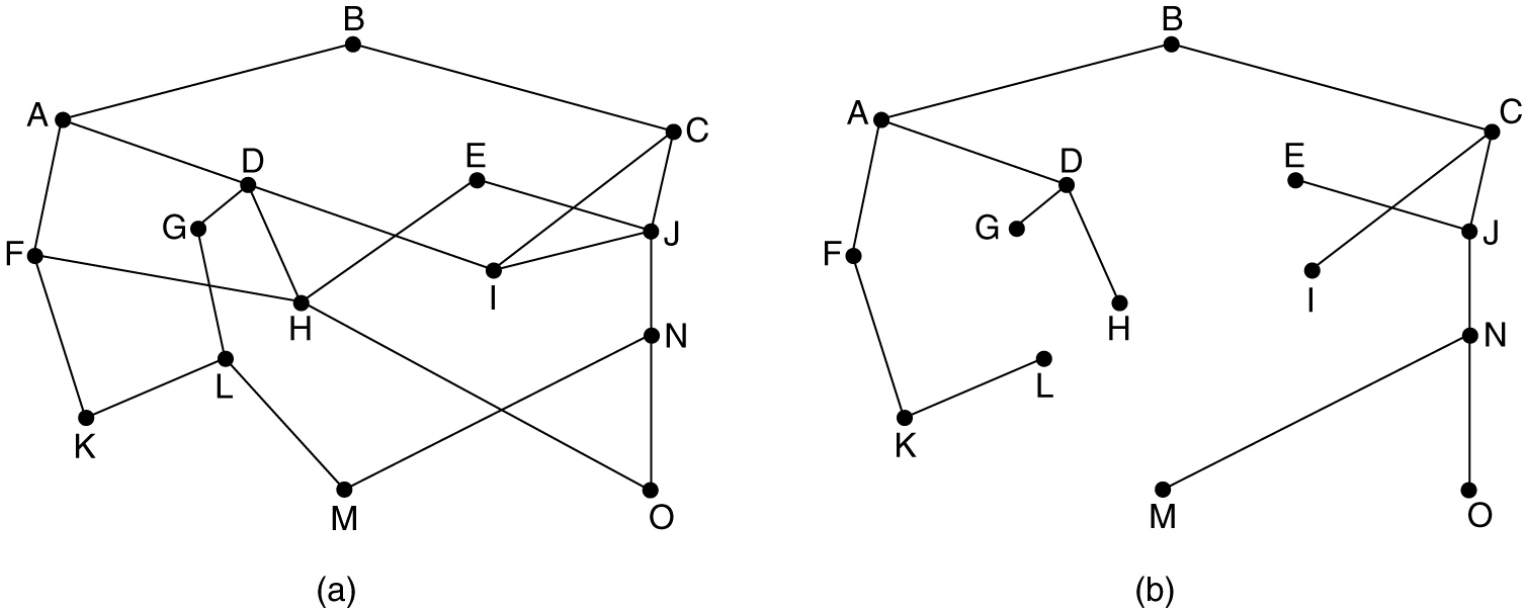
Digital Communication in the Modern World

# 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.

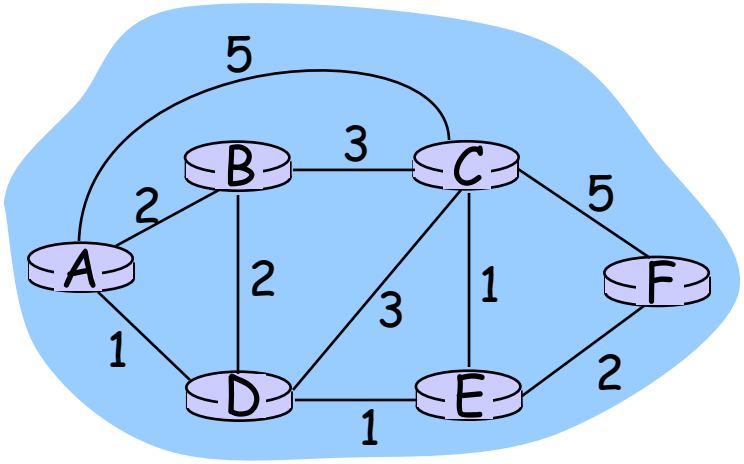
# Network Layer's main problem: To get efficiently from one point to the other in a dynamic environment



# 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

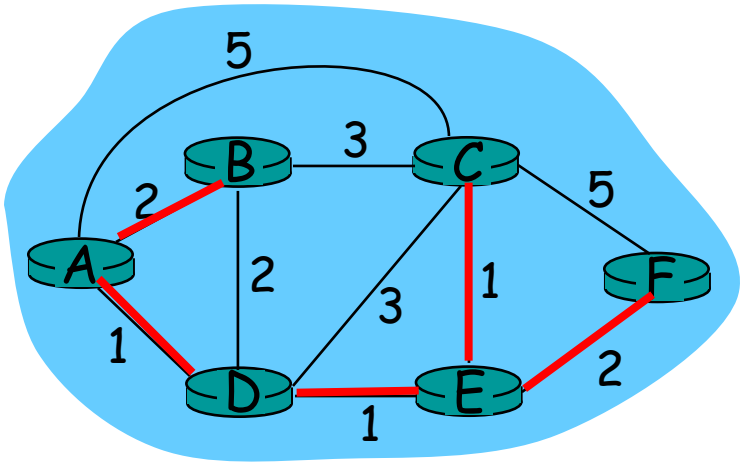
- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ('source") to all other nodes
  - gives routing table for that node
- iterative: after k iterations, know least cost path to k dest.'s

## 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

1 ***Initialization:***
2  N = {A}
3  for all nodes v
4  if v adjacent to A
5  then D(v) = c(A,v)
6  else D(v) = infinity
7
8 ***Loop***
9  find w not in N such that D(w) is a minimum
10  add w to N
11  update D(v) for all v adjacent to w and not in N:
12  D(v) = min( D(v), D(w) + c(w,v) )
13  /* new cost to v is either old cost to v or known
14  shortest path cost to w plus cost from w to v */
15 ***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

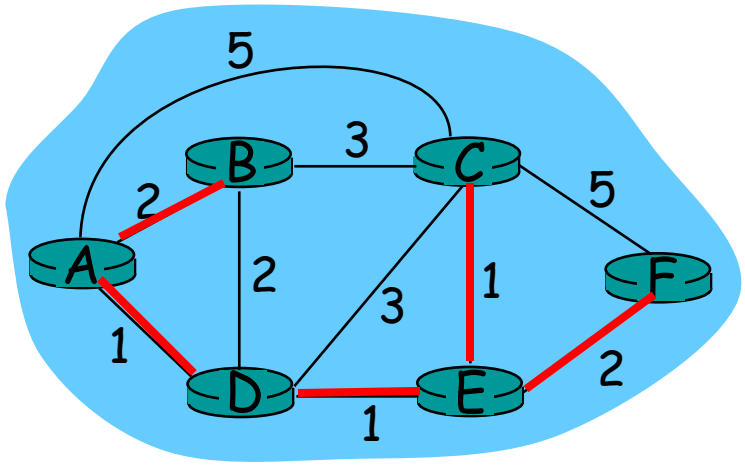
```
  do {                                          /* Is there a better path from k? */
       for (i = 0; i < n; i++)                  /* this graph has n nodes */
            if (dist[k][i] != 0 && state[i].label == tentative) {
                 if (state[k].length + dist[k][i] < state[i].length)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



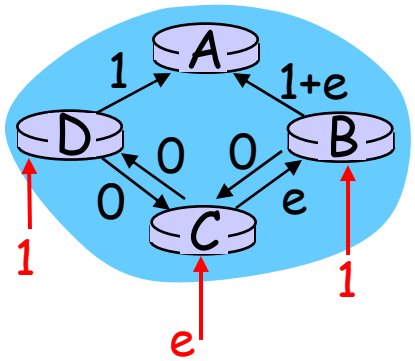
# Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

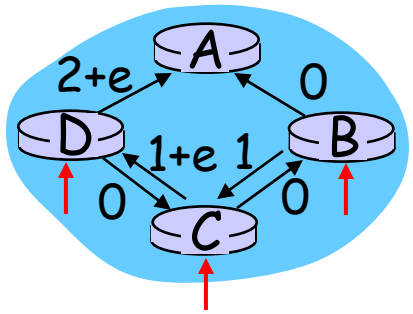
- each iteration: need to check all nodes, w, not in N
- $\sum_{i=1}^{n-1} n-i = \frac{n(n+1)}{2} = O(n^2)$
- more efficient implementations possible: O(nlogn)

Oscillations possible:

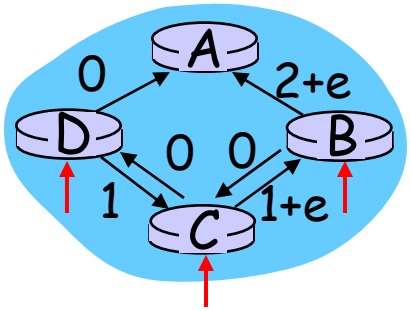
- e.g., if link cost = amount of carried traffic



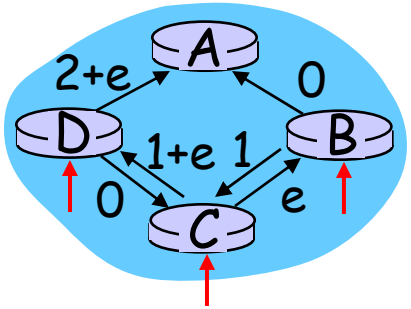

initially




… recompute routing



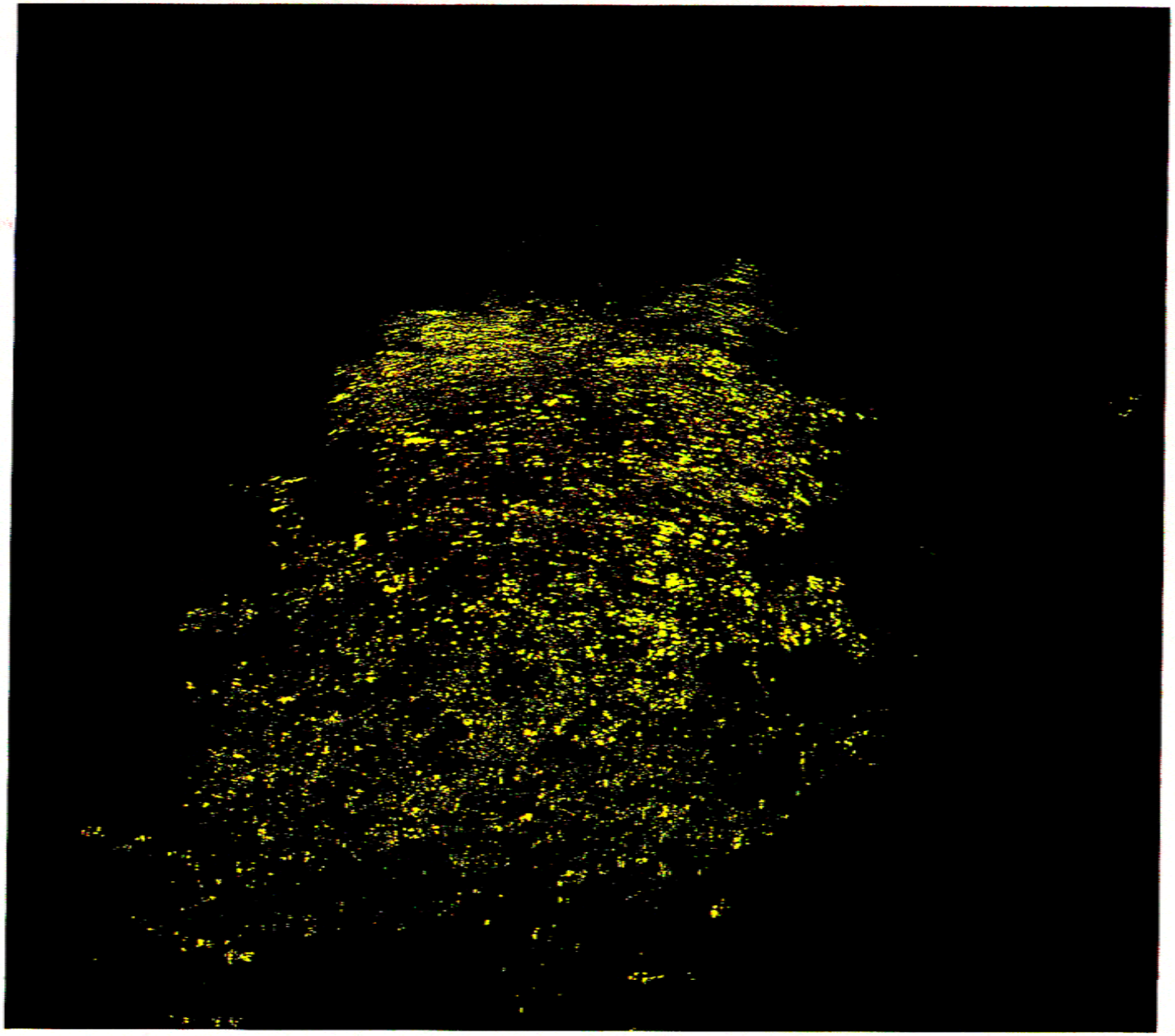

… recompute




… recompute

# 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.



# 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

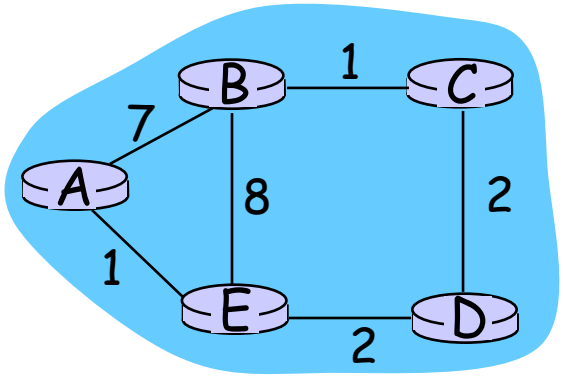
Distance Table data structure

- 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:

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

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

# Distance Table: example




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

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

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

$D^E()$ | cost to destination via

| destination | A | B | D |
|---|---|---|---|
| A | 1 | 14 | 5 |
| B | 7 | 8 | 5 |
| C | 6 | 9 | 4 |
| D | 4 | 11 | 2 |

# Distance table gives routing table

cost to destination via

$D^E()$

| destination | A | B | D |
|---|---|---|---|
| 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

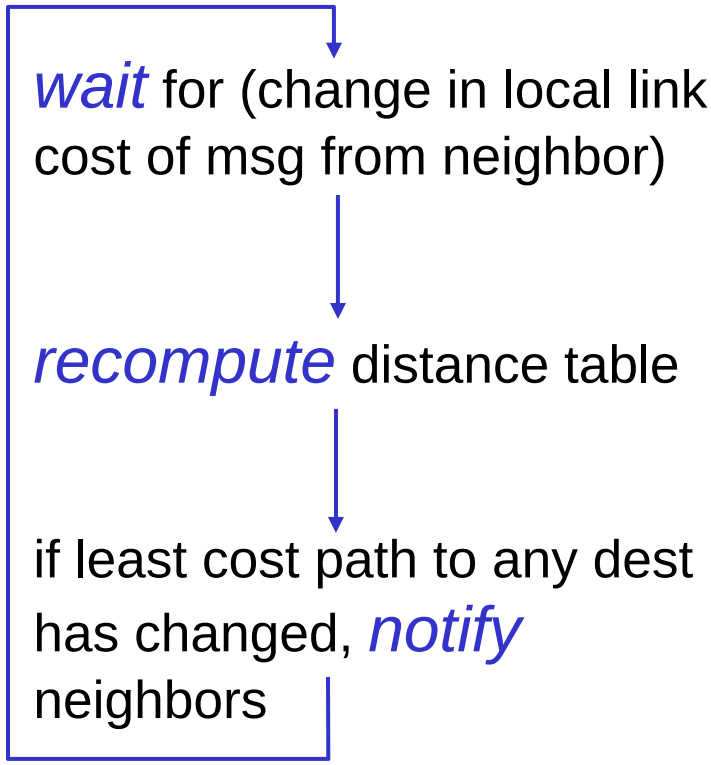
Iterative, asynchronous:
each local iteration caused by:

- 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:



# Distance Vector Algorithm:

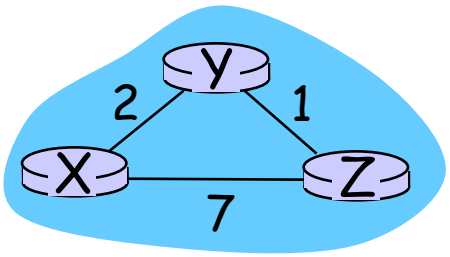
## At all nodes, X:

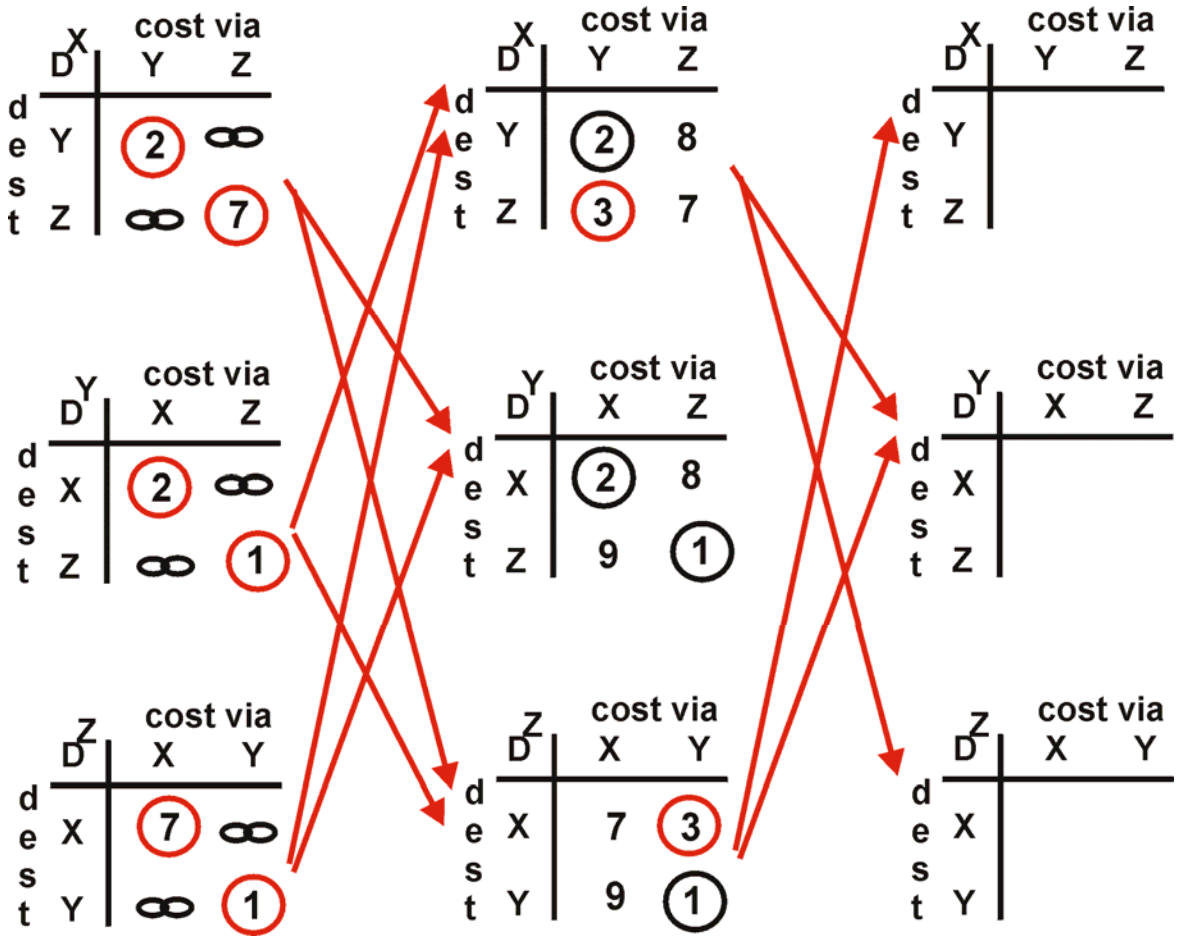
1 Initialization:
2 for all adjacent nodes v:
3 $D^X(*,v) = \text{infinity}$ /* the * operator means "for all rows" */
4 $D^X(v,v) = c(X,v)$
5 for all destinations, y
6 send $\min_w D^X(y,w)$ to each neighbor /* w over all X's neighbors */

# Distance Vector Algorithm (cont.):

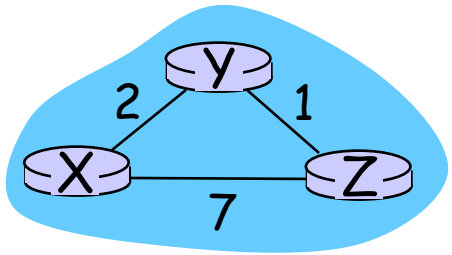
```
8  loop
9    wait (until I see a link cost change to neighbor V
10         or until I receive update from neighbor V)
11
12   if (c(X,V) changes by d)
13     /* change cost to all dest's via neighbor v by d */
14     /* note: d could be positive or negative */
15     for all destinations y:  D^X(y,V) =  D^X(y,V) + d
16
17   else if (update received from V wrt destination Y)
18     /* shortest path from V to some Y has changed  */
19     /* V has sent a new value for its  min_w DV(Y,w) */
20     /* call this received new value is "newval"     */
21     for the single destination y: D^X(Y,V) = c(X,V) + newval
22
23   if we have a new min_w D^X(Y,w)for any destination Y
24      send new value of min_w D^X(Y,w) to all neighbors
25
26 forever
```

# Distance Vector Algorithm: example



# Distance Vector Algorithm: example




$D^X$

| dest | via Y | via Z |
|---|---|---|
| Y | 2 | ∞ |
| Z | ∞ | 7 |

$D^Y$

| dest | via X | via Z |
|---|---|---|
| X | 2 | ∞ |
| Z | ∞ | 1 |

$D^Z$

| dest | via X | via Y |
|---|---|---|
| X | 7 | ∞ |
| Y | ∞ | 1 |

$D^X$

| dest | via Y | via Z |
|---|---|---|
| Y | 2 | 8 |
| Z | 3 | 7 |

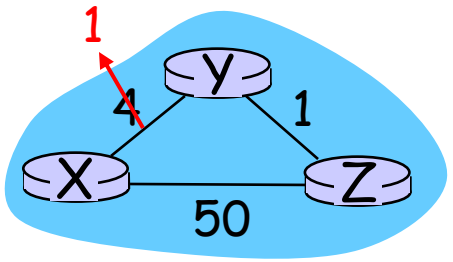
$$D^X(Y,Z) = c(X,Z) + \min_w\{D^Z(Y,w)\}$$
$$= 7+1 = 8$$

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

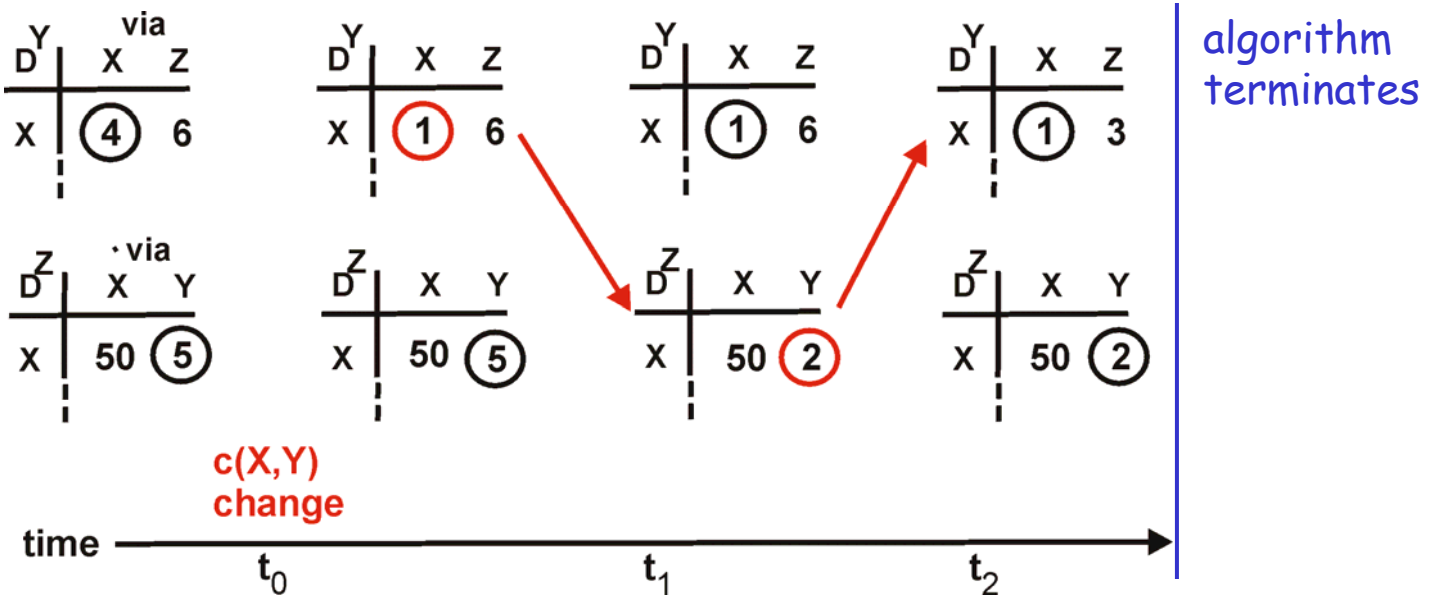
# Distance Vector: link cost changes

## 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)



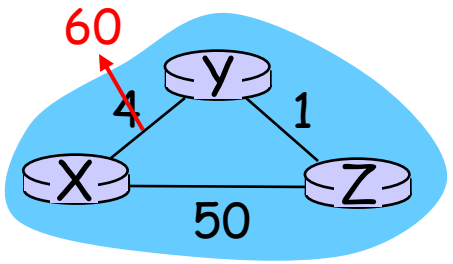

"good news travels fast"

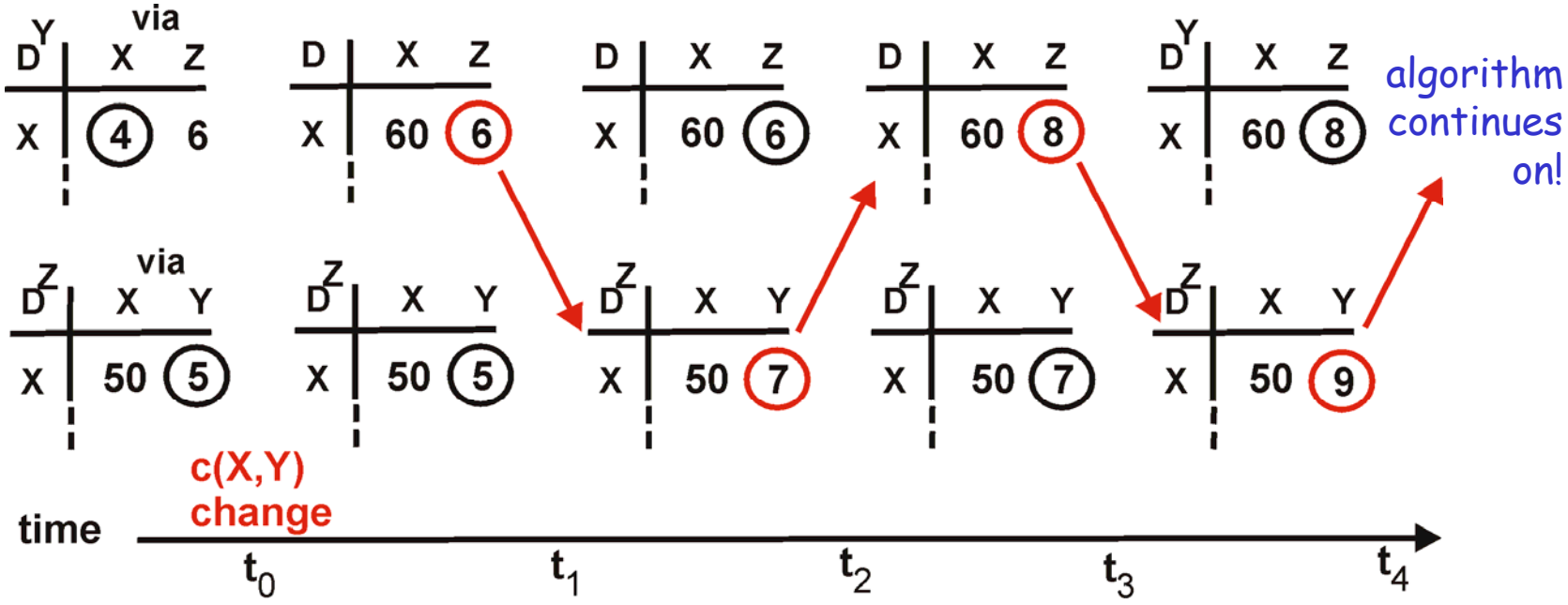


# Distance Vector: link cost changes

## Link cost changes:

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

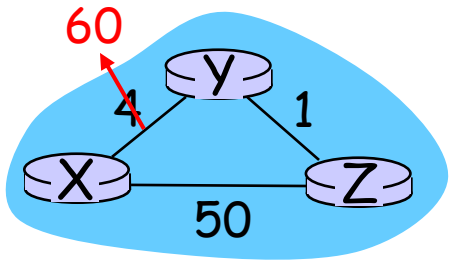


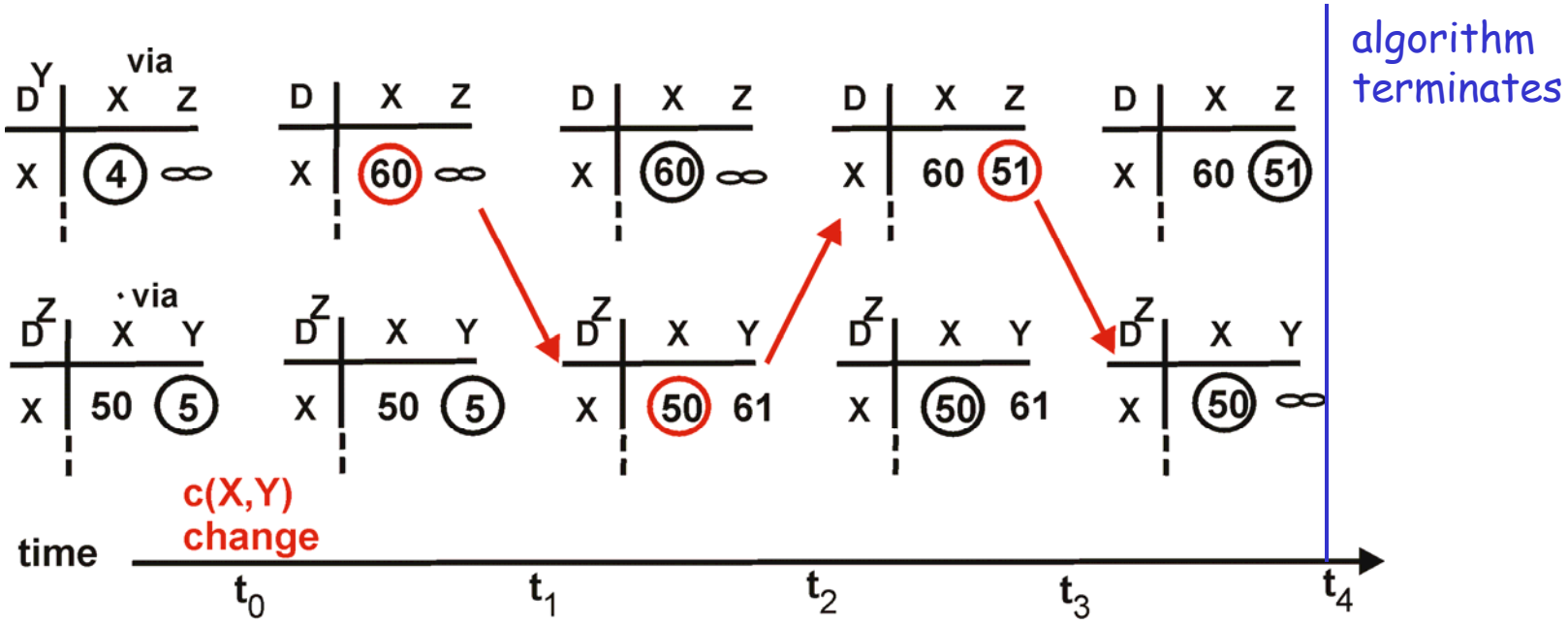



# Distance Vector: poisoned reverse

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

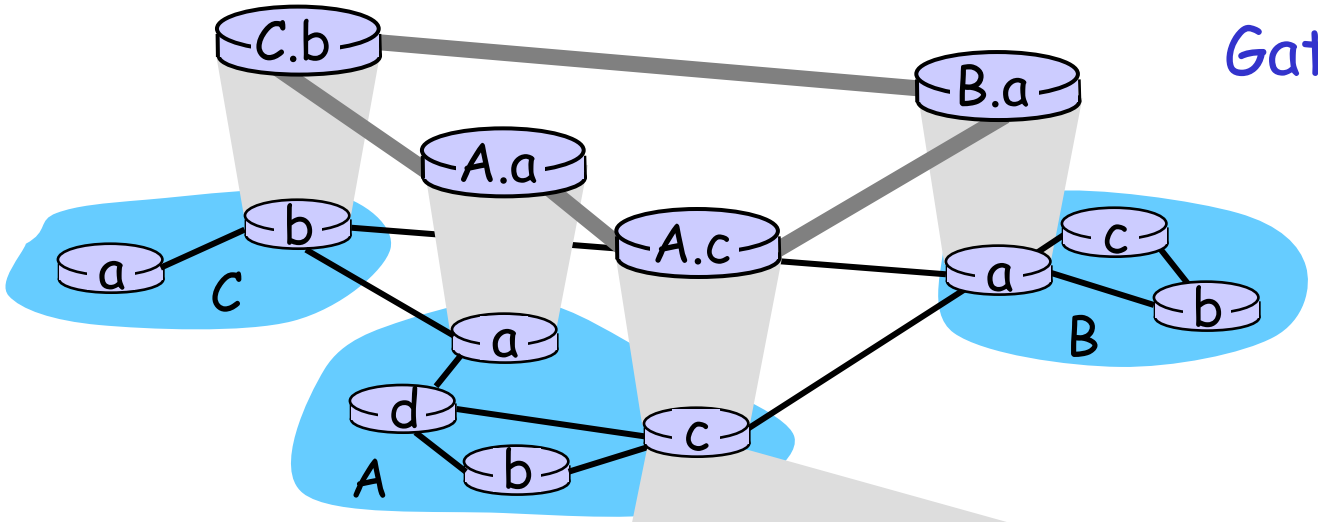
# Hierarchical Routing

- 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

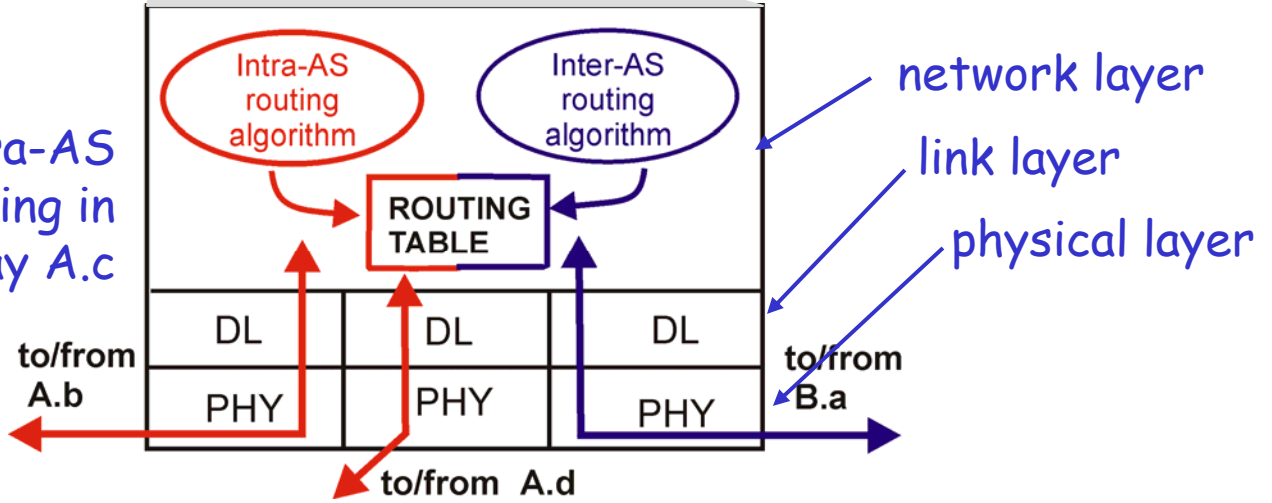
# Intra-AS and Inter-AS routing



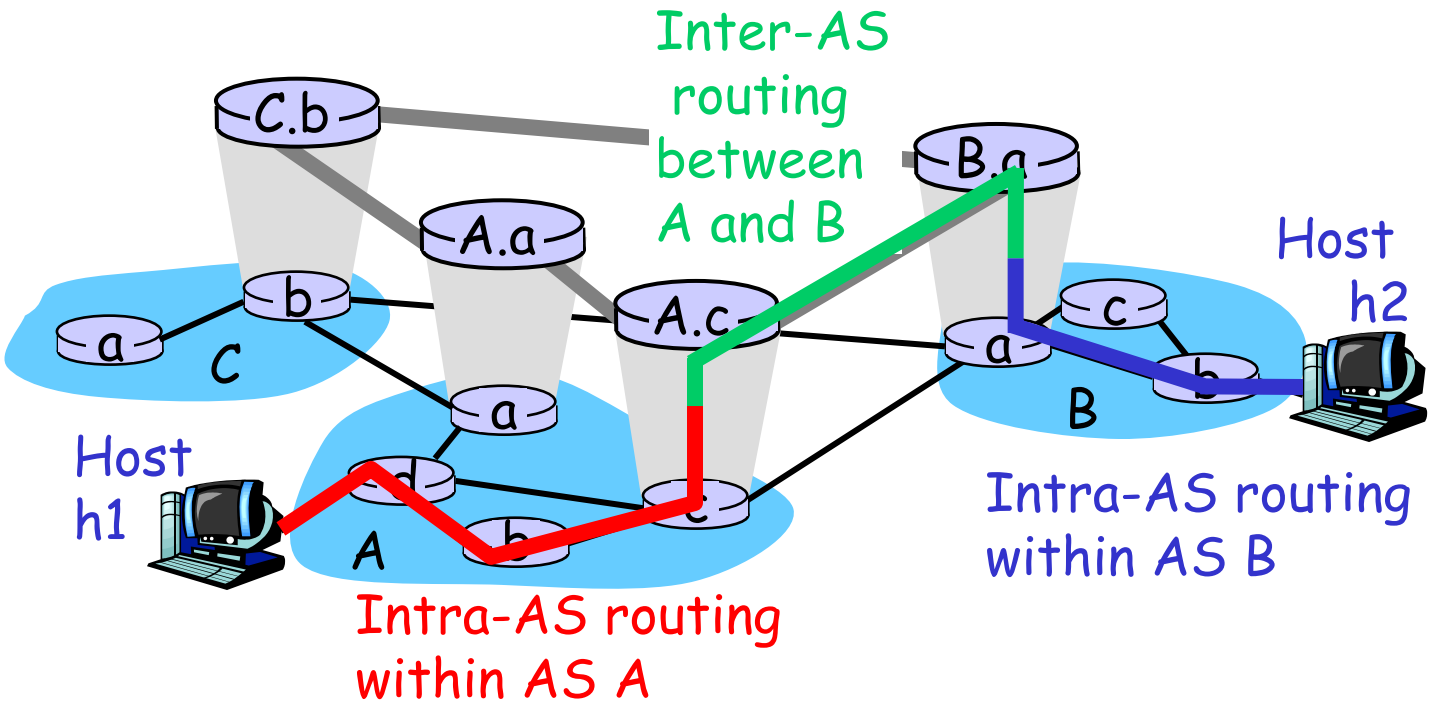

Gateways:

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

inter-AS, intra-AS routing in gateway A.c



# Intra-AS and Inter-AS routing




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