# COMP 3331/9331: Computer Networks and Applications

Week 8

Control Plane (Routing)

**Chapter 5: Section 5.1 – 5.2, 5.6**

# Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector
- Hierarchical routing (NOT ON EXAM)

5.6 ICMP: The Internet Control Message Protocol

# Network-layer functions

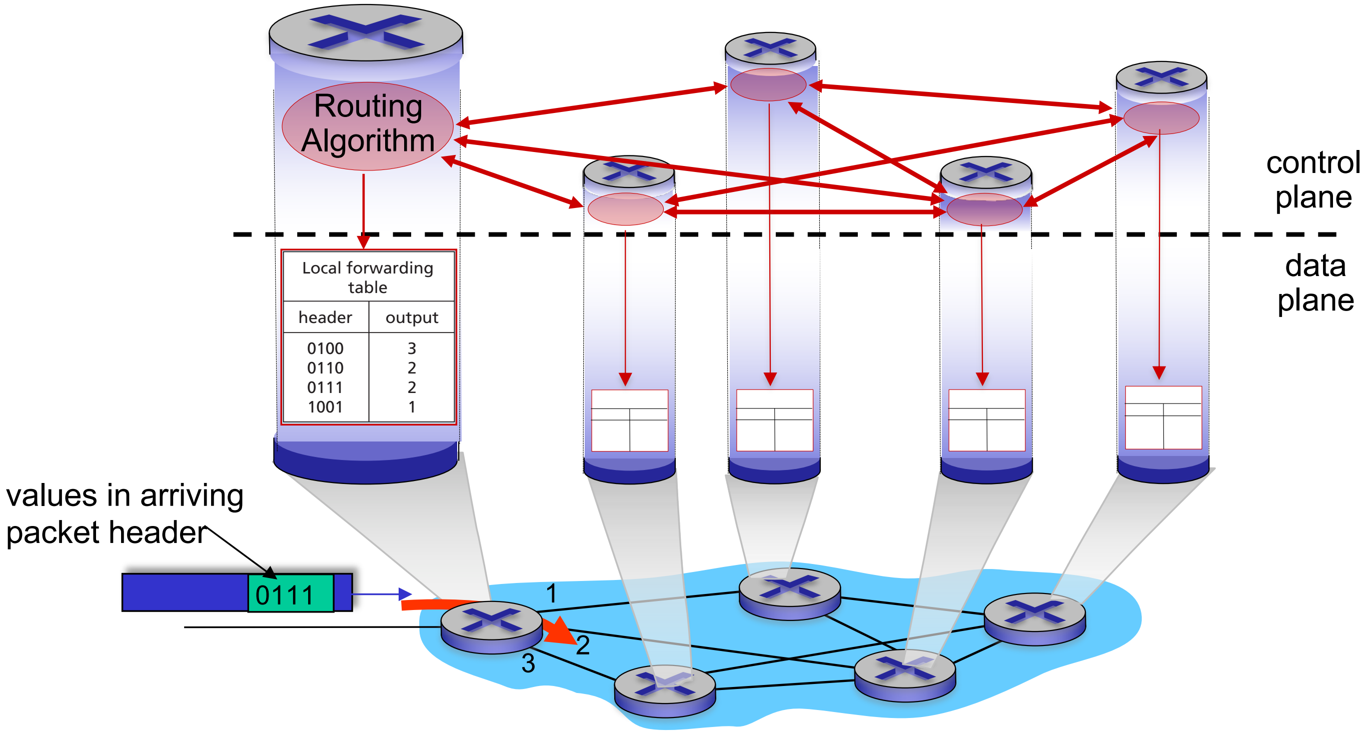
- forwarding: move packets from router's input to appropriate router output — *data plane*
- routing: determine route taken by packets from source to destination — *control plane*

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

# Per-router control plane

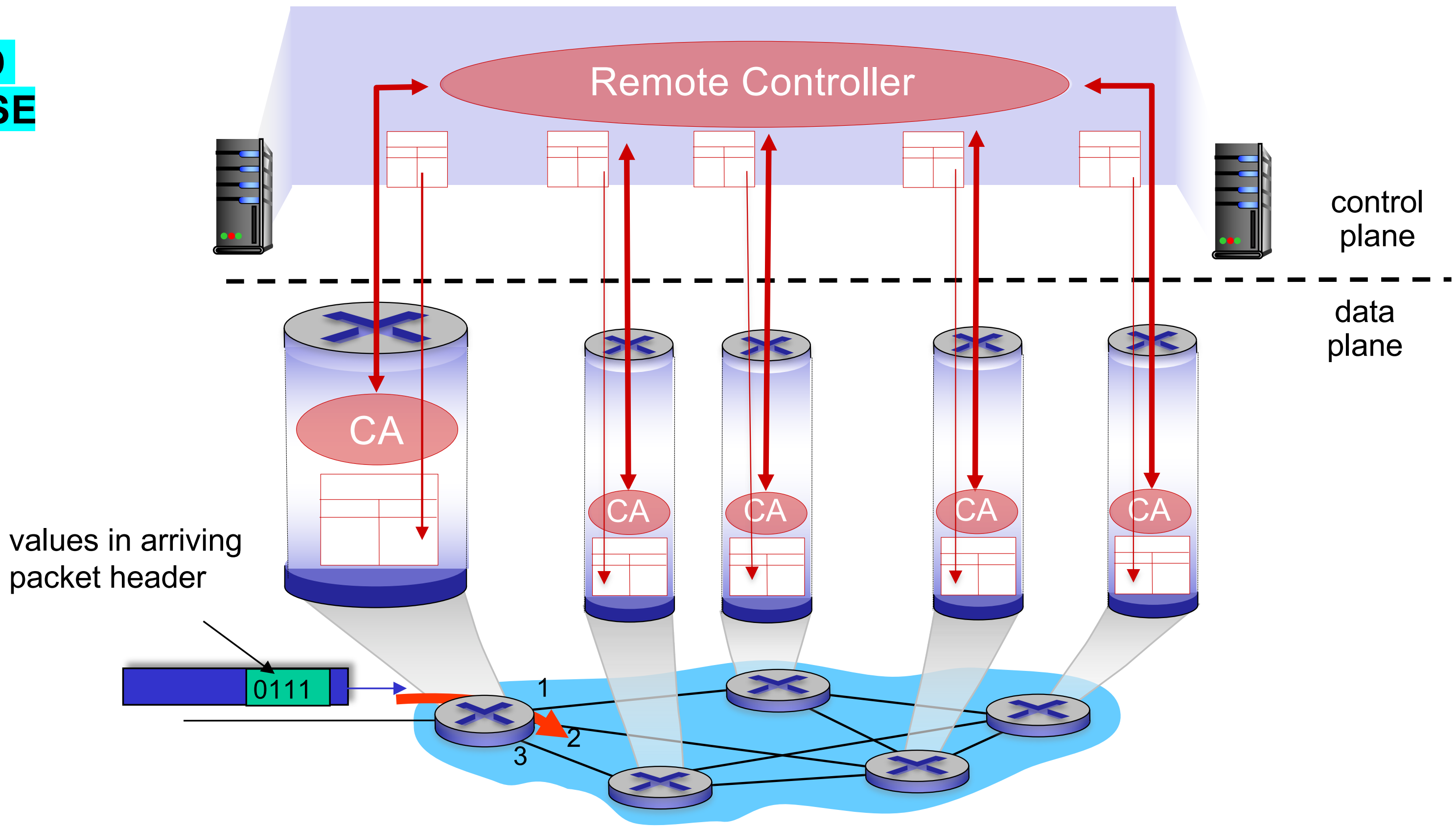
Individual routing algorithm components *in each and every router* interact in the control plane



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers

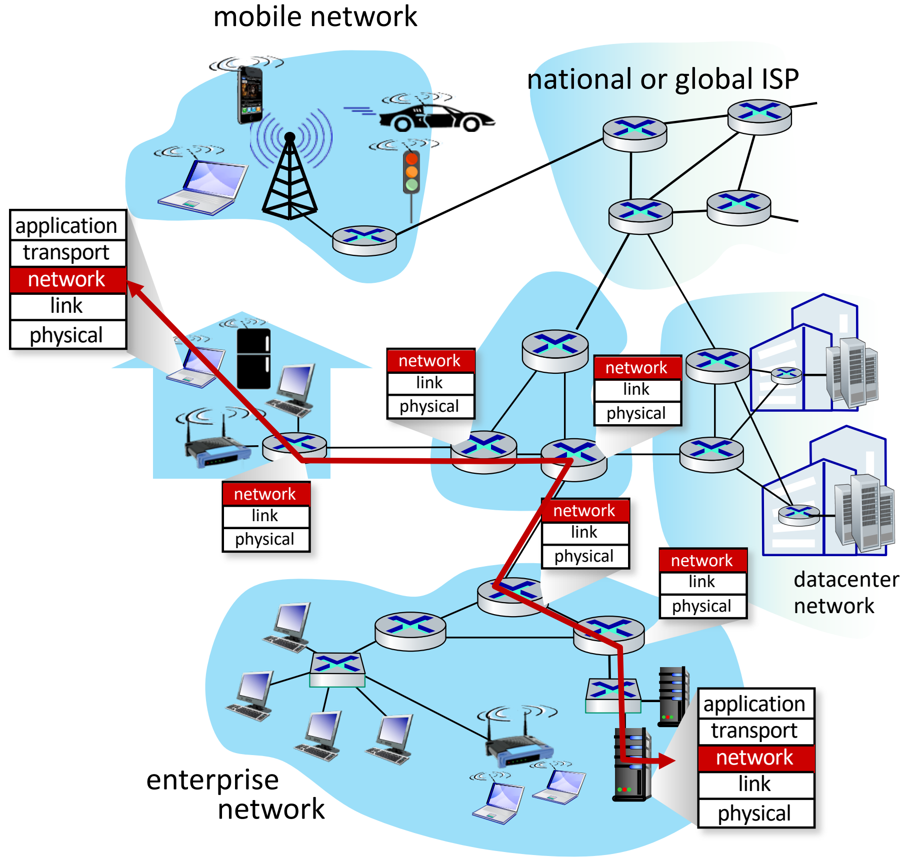
**NOT COVERED IN THIS COURSE**



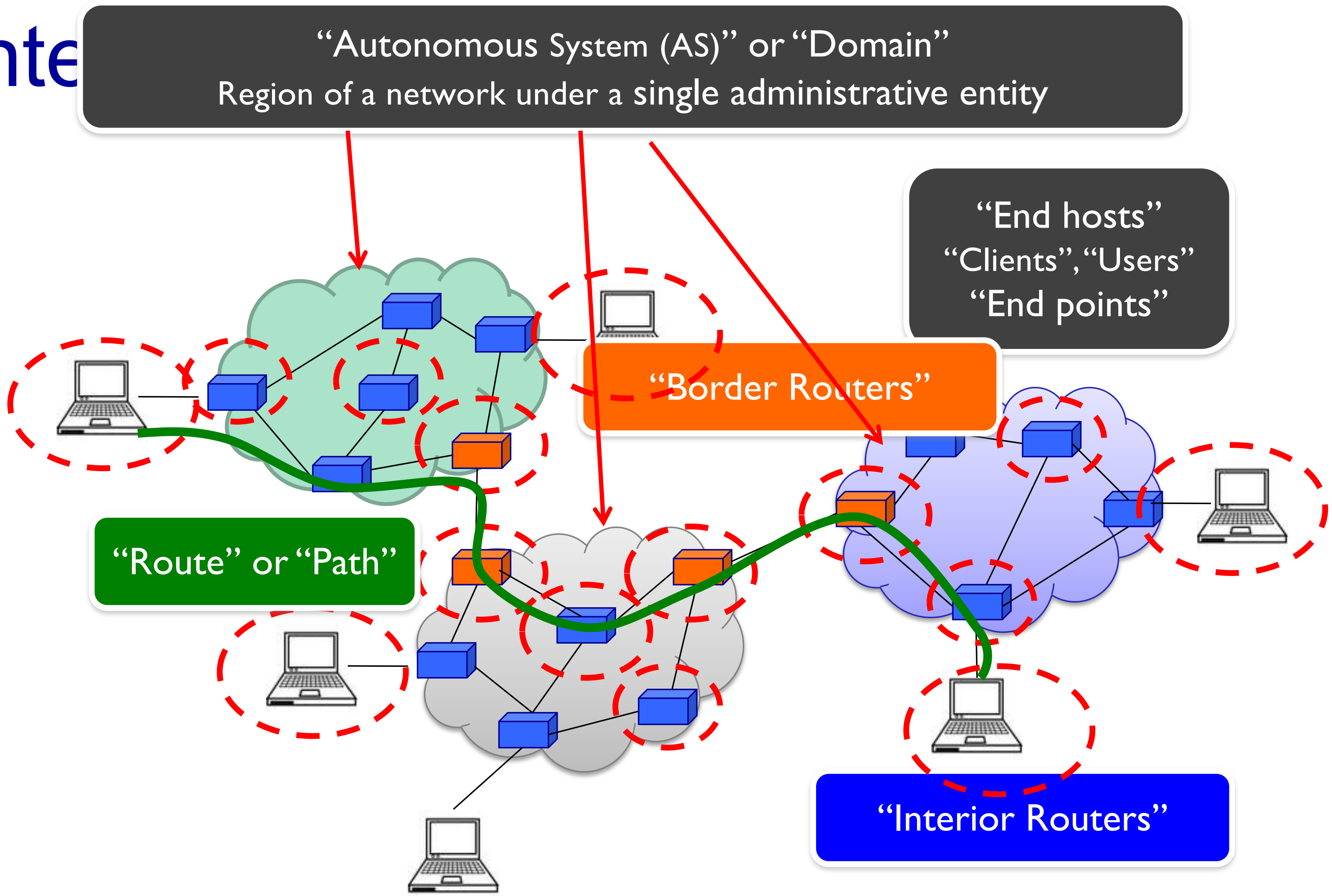
# Routing protocols

Routing protocol goal: determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- path: sequence of routers packets traverse from given initial source host to final destination host
- "good": least "cost", "fastest", "least congested"
- routing: a "top-10" networking challenge!



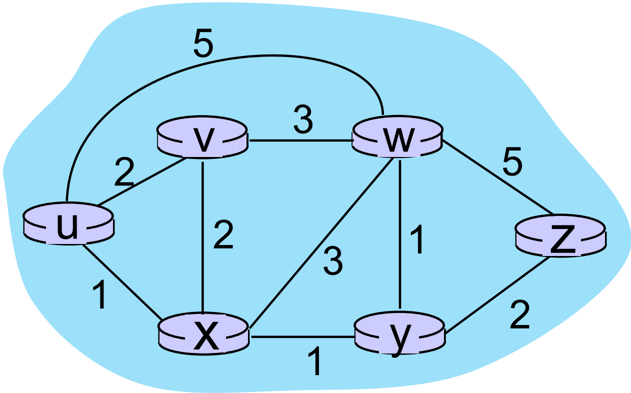
# Conte


"Autonomous System (AS)" or "Domain"
Region of a network under a single administrative entity
"End hosts"
"Clients", "Users"
"End points"
"Border Routers"
"Route" or "Path"
"Interior Routers"

# Internet Routing

- Internet Routing works at two levels
- Each AS runs an intra-domain routing protocol that establishes routes within its domain
  - AS -- region of network under a single administrative entity
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Distance Vector, e.g., Routing Information Protocol (RIP)
- ASes participate in an inter-domain routing protocol that establishes routes between domains
  - Path Vector, e.g., Border Gateway Protocol (BGP)

# Graph abstraction: link costs




$c_{a,b}$: cost of *direct* link connecting *a* and *b*

*e.g.*, $c_{w,z} = 5$, $c_{u,z} = \infty$

cost defined by network operator: could always be 1, or inversely related to bandwidth, or inversely related to congestion

graph: $G = (N,E)$

$N$: set of routers = { *u, v, w, x, y, z* }

$E$: set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

# Routing algorithm classification

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

*How fast do routes change?*

**static:** routes change slowly over time

**dynamic:** routes change more quickly
- periodic updates or in response to link cost changes

**decentralized:** iterative process of computation, exchange of info with neighbors
- routers initially only know link costs to attached neighbors
- "distance vector" algorithms

*global or decentralized information?*

# Network layer, control plane: outline

5.1 introduction

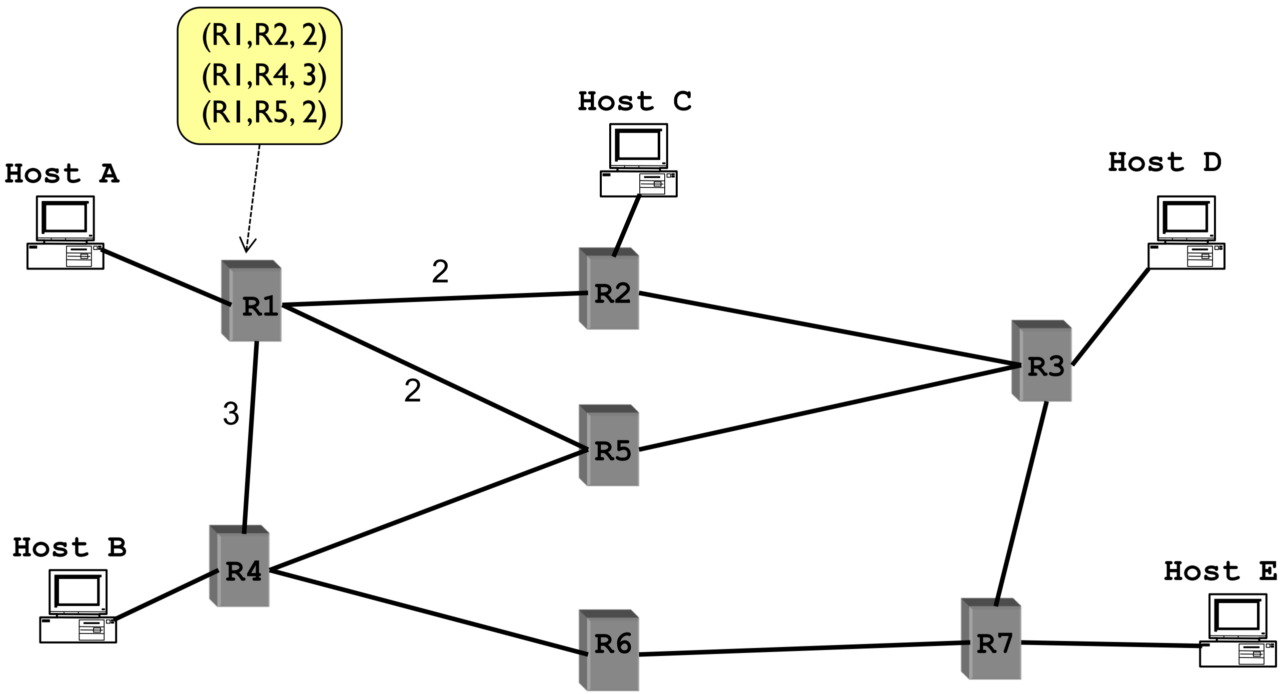
5.2 routing protocols

- link state
- distance vector
- hierarchical routing

5.6 ICMP: The Internet Control Message Protocol

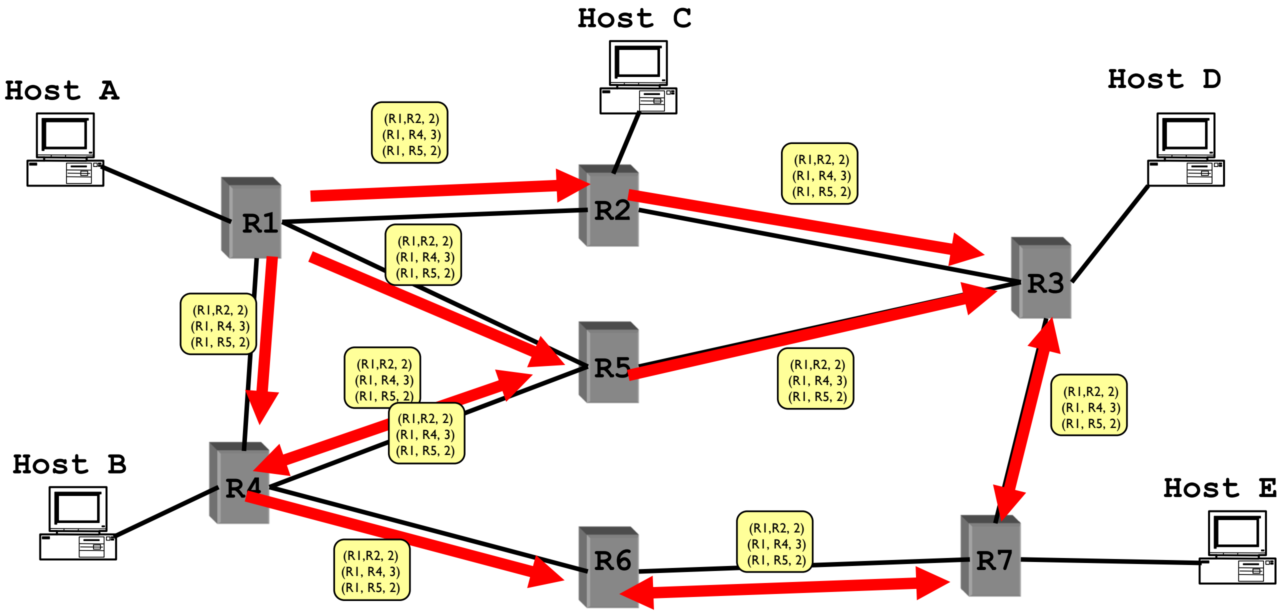
# Link State Routing

- Each node maintains its local "link state" (LS)
  - i.e., a list of its directly attached links and their costs



# Link State Routing

- Each node maintains its local "link state" (LS)
- Each node floods its local link state
  - on receiving a new LS message, a router forwards the message to all its neighbors other than the one it received the message from

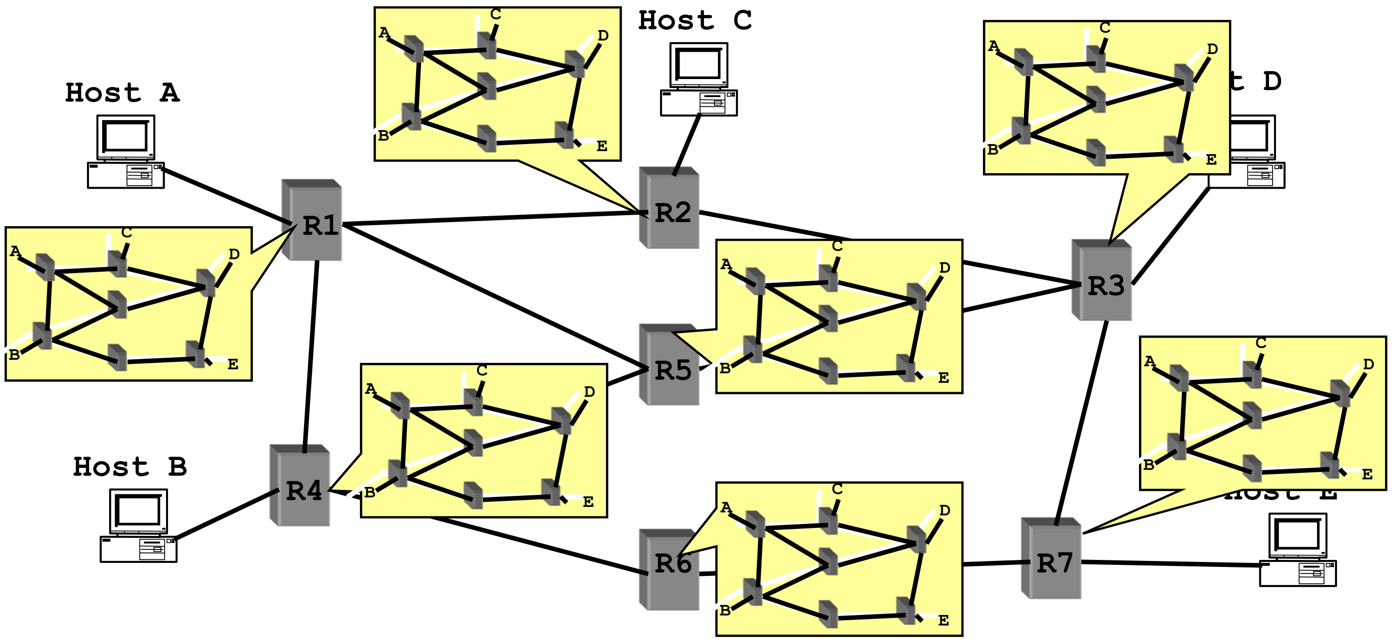


# Flooding LSAs

- Routers transmit Link State Advertisement (LSA) on links
  - A neighbouring router forwards out on all links except incoming
  - Keep a copy locally; don’t forward previously-seen LSAs
- Challenges
  - Packet loss
  - Out of order arrival
- Solutions
  - Acknowledgements and retransmissions
  - Sequence numbers
  - Time-to-live for each packet

# Link State Routing

- Each node maintains its local "link state" (LS)
- Each node floods its local link state
- Eventually, each node learns the entire network topology
  - Can use Dijkstra's to compute the shortest paths between nodes



# Dijkstra's link-state routing algorithm

- centralized: network 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 *forwarding table* for that node
- iterative: after *k* iterations, know least cost path to *k* destinations

**notation**

- $c_{x,y}$: direct link cost from node *x* to *y*; = ∞ if not direct neighbors
- *D(v)*: *current* estimate of cost of least-cost-path from source to destination *v*
- *p(v)*: predecessor node along path from source to *v*
- *N'*: set of nodes whose least-cost-path *definitively* known

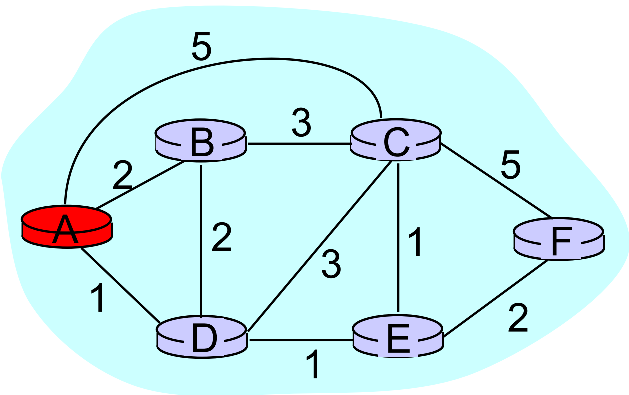
# Dijkstra's link-state routing algorithm

```
Initialization:
  N' = {u}                          /* compute least cost path from u to all other nodes */
  for all nodes v
    if v adjacent to u              /* u initially knows direct-path-cost only to  direct neighbors */
      then D(v) = c_{u,v}           /* but may not be minimum cost! */
    else D(v) = ∞

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 least-path-cost to v is either old least-cost-path to v or known
  least-cost-path to w plus direct-cost from w to v */
until all nodes in N'
```

# Example: Dijkstra's Algorithm

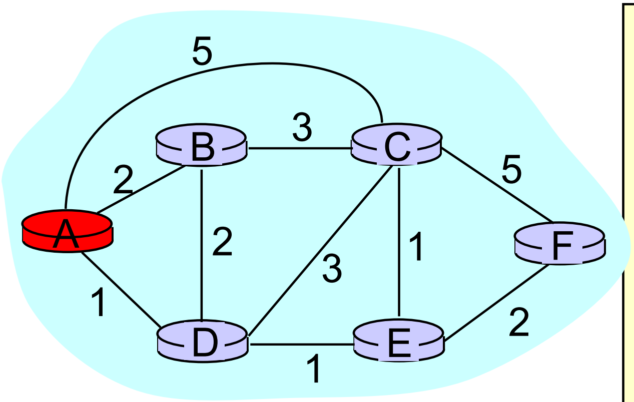
| Step | Set N' | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




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

# Example: Dijkstra's Algorithm

| Step | Set 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 | ∞ | ∞ |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




…

***Loop***

find **w** not in **N'** s.t. D(w) is a minimum;

add **w** to **N'**;

update D(v) for all **v** adjacent to **w** and not in **N'**:

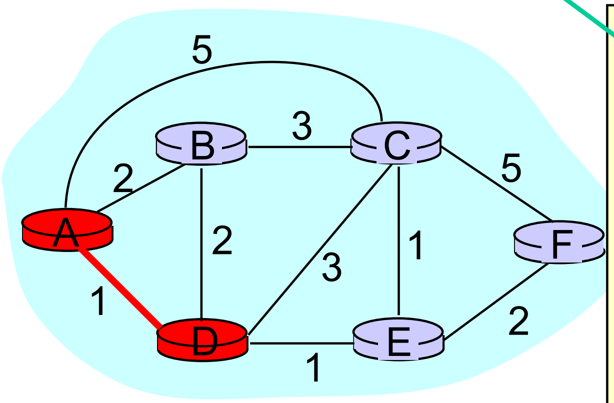
If D(w) + c(w,v) < D(v) then

D(v) = D(w) + c(w,v); p(v) = w;

***until all nodes in N';***

# Example: Dijkstra’s Algorithm

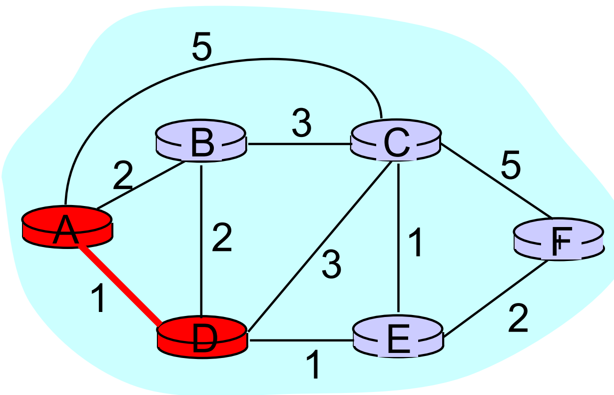
| Step | Set N’ | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
…
Loop
  find w not in N’ s.t. D(w) is a minimum;
  add w to N’;
  update D(v) for all v adjacent
    to w and not in N’:
  If D(w) + c(w,v) < D(v) then
     D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in N’;
```

# Example: Dijkstra's Algorithm

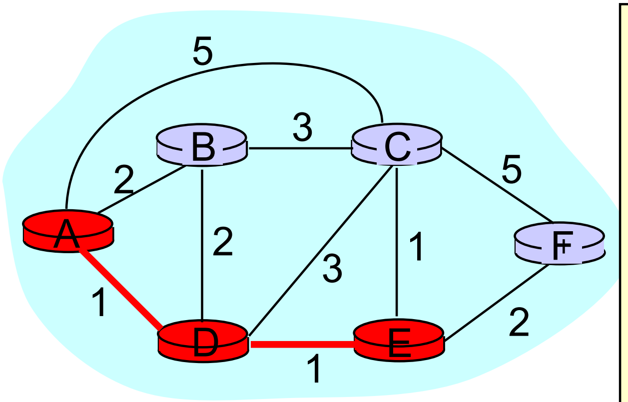
| Step | Set N' | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | 2, A | 4,D | | 2,D | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
…
8   Loop
9     find w not in N' s.t. 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    If D(w) + c(w,v) < D(v) then
13       D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in N';
```

# Example: Dijkstra's Algorithm

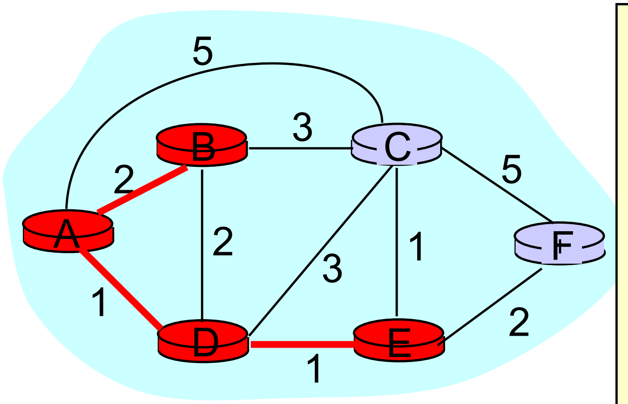
| Step | Set N' | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | 2, A | 4,D | | 2,D | |
| 2 | ADE | 2, A | 3,E | | | 4,E |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
...
Loop
  find w not in N' s.t. D(w) is a minimum;
  add w to N';
  update D(v) for all v adjacent
     to w and not in N':
  If D(w) + c(w,v) < D(v) then
     D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in N';
```

# Example: Dijkstra's Algorithm

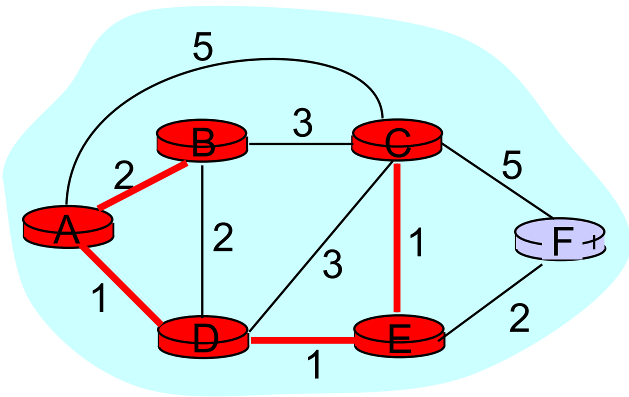
| Step | Set N' | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | 2,A | 4,D | | 2,D | |
| 2 | ADE | 2,A | 3,E | | | 4,E |
| 3 | ADEB | | 3,E | | | 4,E |
| 4 | | | | | | |
| 5 | | | | | | |




```
…
Loop
  find w not in N' s.t. D(w) is a minimum;
  add w to N';
  update D(v) for all v adjacent
    to w and not in N':
  If D(w) + c(w,v) < D(v) then
    D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in N';
```

# Example: Dijkstra’s Algorithm

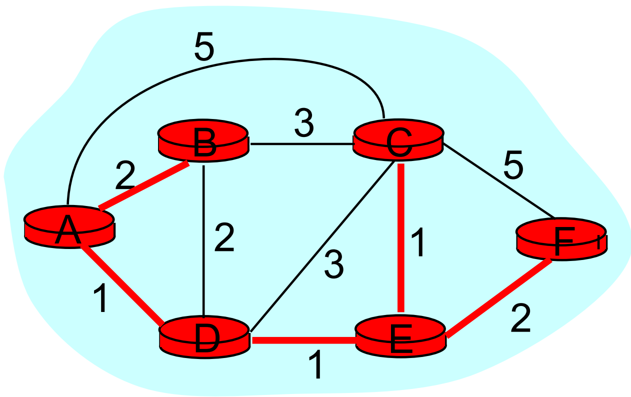
| Step | Set N’ | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | 2,A | 4,D | | 2,D | |
| 2 | ADE | 2,A | 3,E | | | 4,E |
| 3 | ADEB | | 3,E | | | 4,E |
| 4 | ADEBC | | | | | 4,E |
| 5 | | | | | | |




...
**_Loop_**
find **w** not in **N’** s.t. D(w) is a minimum;
add **w** to **N’**;
update D(v) for all **v** adjacent
to **w** and not in **N’**:
If D(w) + c(w,v) < D(v) then
D(v) = D(w) + c(w,v); p(v) = w;
**_until all nodes in N’;_**

# Example: Dijkstra’s Algorithm

| Step | Set 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 | ∞ | ∞ |
| 1 | AD | 2,A | 4,D | | 2,D | |
| 2 | ADE | 2,A | 3,E | | | 4,E |
| 3 | ADEB | | 3,E | | | 4,E |
| 4 | ADEBC | | | | | 4,E |
| 5 | ADEBCF | | | | | |




…

***Loop***

find **w** not in **N’** s.t. D(w) is a minimum;

add **w** to **N’**;

update D(v) for all **v** adjacent to **w** and not in **N’**:

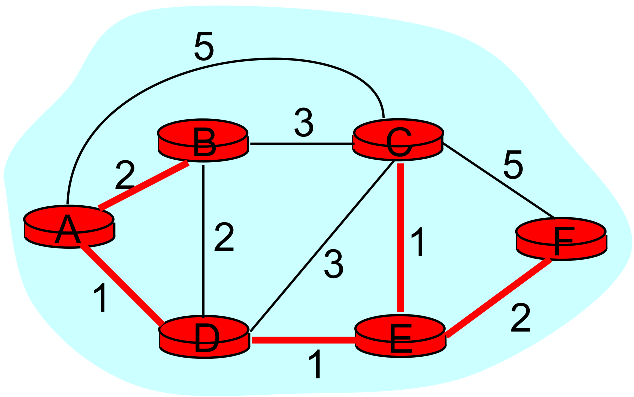
If D(w) + c(w,v) < D(v) then

D(v) = D(w) + c(w,v); p(v) = w;

***until all nodes in N’;***

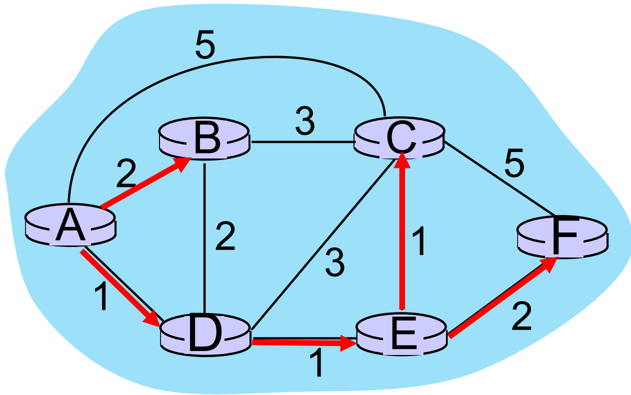
# Example: Dijkstra's Algorithm

| Step | Set 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 | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | ADEB | | | | | |
| 4 | ADEBC | | | | | |
| 5 | ADEBCF | | | | | |

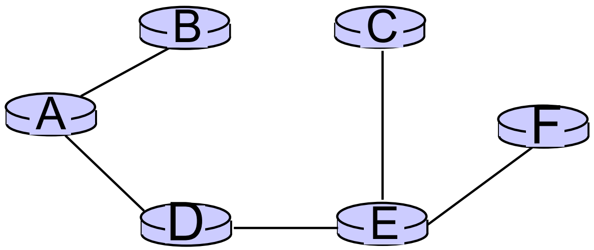



To determine path A → C (say), work backward from C via p(v)

# Example: Dijkstra's Algorithm




resulting least-cost-path tree from A:




resulting forwarding table in A:

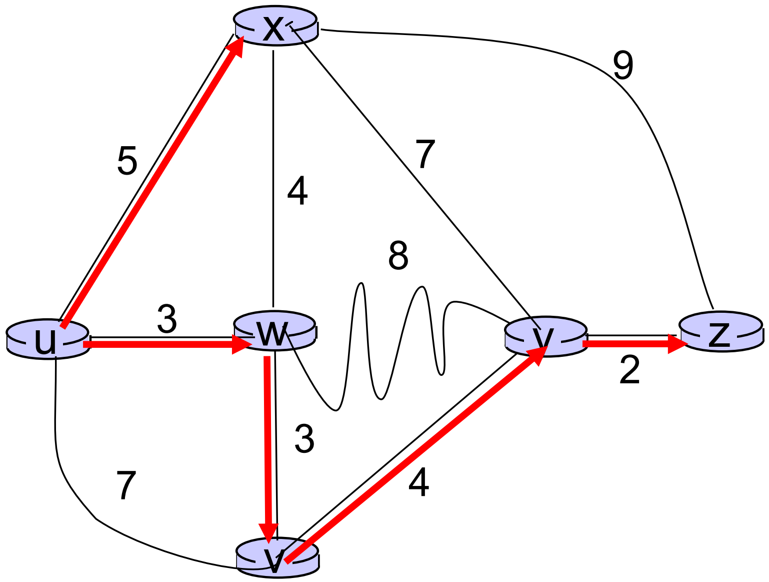
| destination | outgoing link |
|---|---|
| B | (A,B) |
| C | (A,D) |
| D | (A,D) |
| E | (A,D) |
| F | (A,D) |

route from *A* to *B* directly

route from A to all other destinations via *D*

# Dijkstra's algorithm: another example

| Step | N' | *v*<br>*D(v), p(v)* | *w*<br>*D(w), p(w)* | *x*<br>*D(x), p(x)* | *y*<br>*D(y), p(y)* | *z*<br>*D(z), p(z)* |
|---|---|---|---|---|---|---|
| 0 | u | 7,u | 3,u | 5,u | ∞ | ∞ |
| 1 | uw | 6,w | | 5,u | 11,w | ∞ |
| 2 | uwx | 6,w | | | 11,w | 14,x |
| 3 | uwxv | | | | 10,v | 14,x |
| 4 | uwxvy | | | | | 12,y |
| 5 | uwxvyz | | | | | |




notes:

- construct least-cost-path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

# Dijkstra's algorithm: discussion

algorithm complexity: *n* nodes

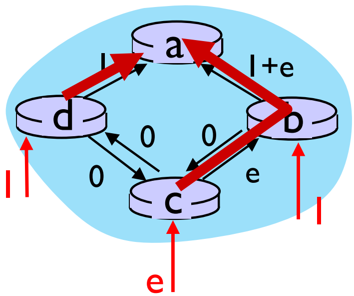
- each of *n* iteration: need to check all nodes, *w*, not in *N*
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n \log n)$

message complexity:

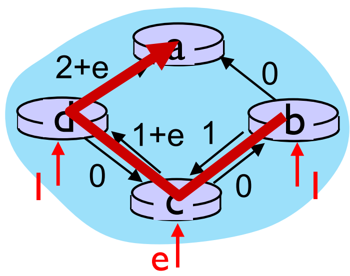
- each router must *broadcast* its link state information to other *n* routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

# Dijkstra's algorithm: oscillations possible

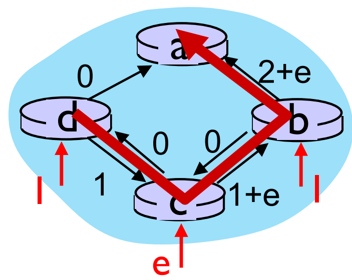
- when link costs depend on traffic volume, route oscillations possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent



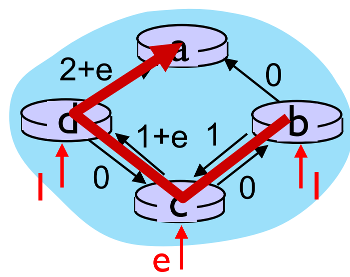

initially




given these costs,
find new routing….
resulting in new costs




given these costs,
find new routing….
resulting in new costs




given these costs,
find new routing….
resulting in new costs

# Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector
- hierarchical routing

5.6 ICMP: The Internet Control Message Protocol

# Distance vector algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

**Bellman-Ford equation**

Let $D_x(y)$: cost of least-cost path from $x$ to $y$.

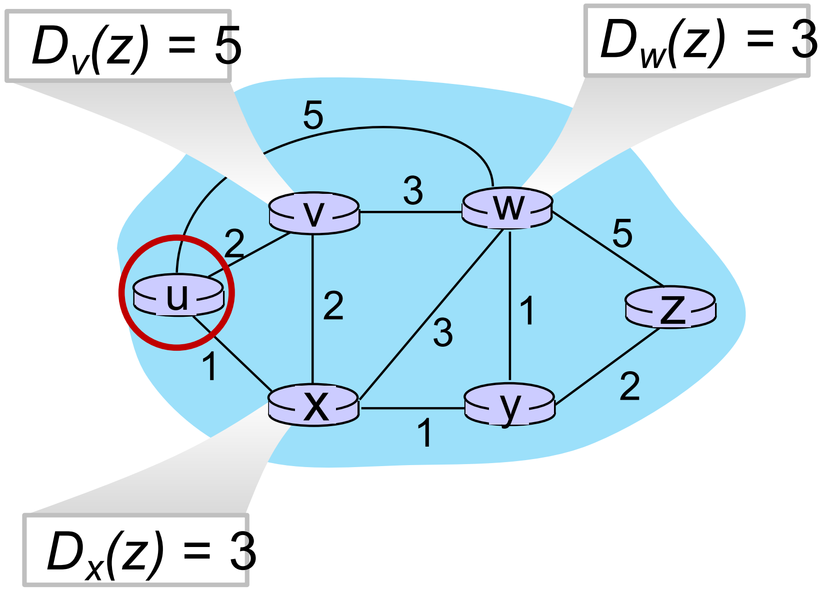
Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

- $v$'s estimated least-cost-path cost to $y$
- *min* taken over all neighbors $v$ of $x$
- direct cost of link from $x$ to $v$

# Bellman-Ford Example

Suppose that *u*'s neighboring nodes, *x,v,w*, know that for destination *z*:




Bellman-Ford equation says:

$$D_u(z) = \min \{ c_{u,v} + D_v(z), \; c_{u,x} + D_x(z), \; c_{u,w} + D_w(z) \}$$

$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

*node achieving minimum (x) is next hop on estimated least-cost path to destination (z)*

# Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when *x* receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow min_v\{c_{x,v} + D_v(y)\} \quad \text{for each node } y \in N$$

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

# Distance vector algorithm:

each node:

*wait* for (change in local link cost or DV from neighbor)

↓

*recompute* DV estimates using DV received from neighbor

↓

if DV to any destination has changed, *notify* neighbors

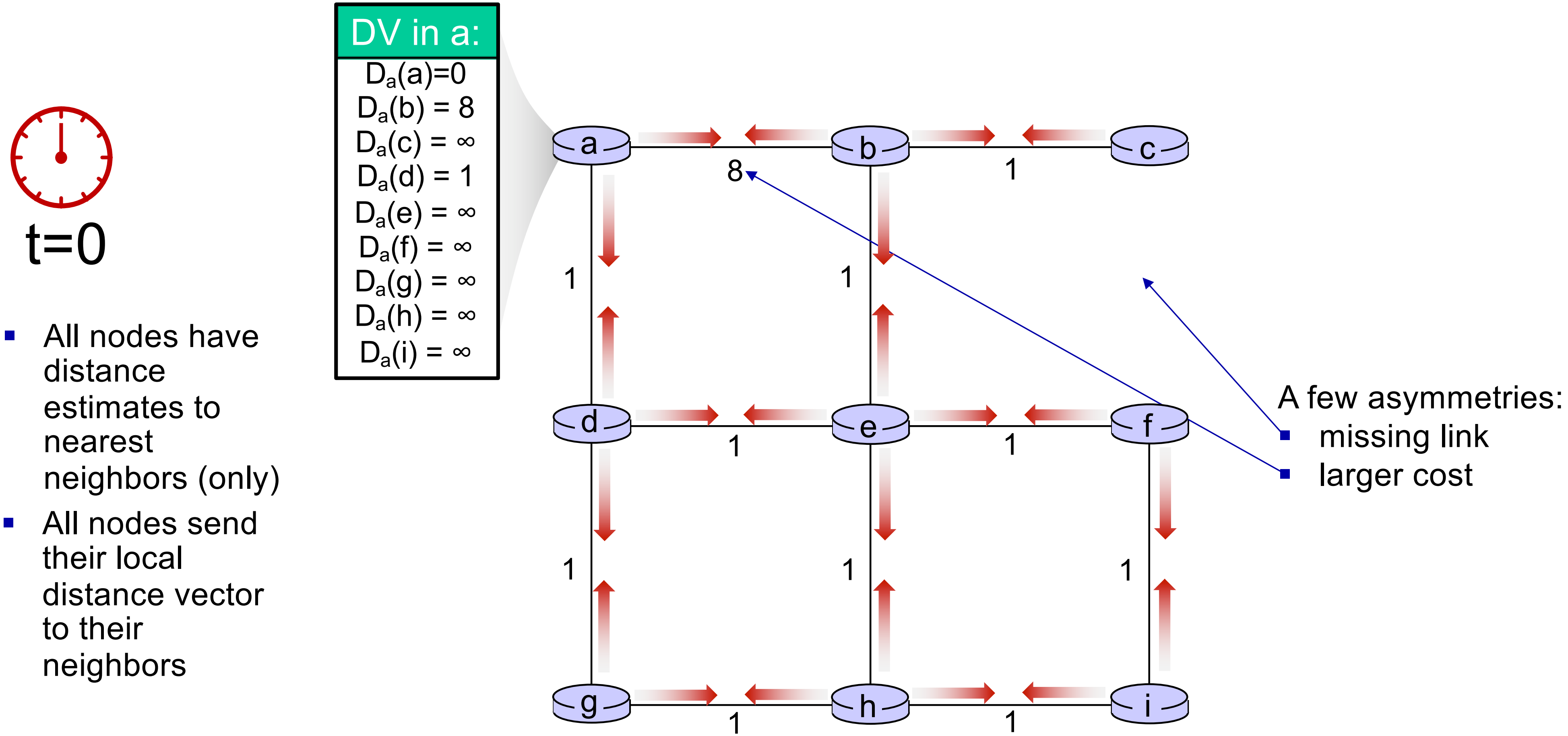
iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed, self-stopping: each node notifies neighbors *only* when its DV changes

- neighbors then notify their neighbors – *only if necessary*
- no notification received; no actions taken!

# Distance vector: example



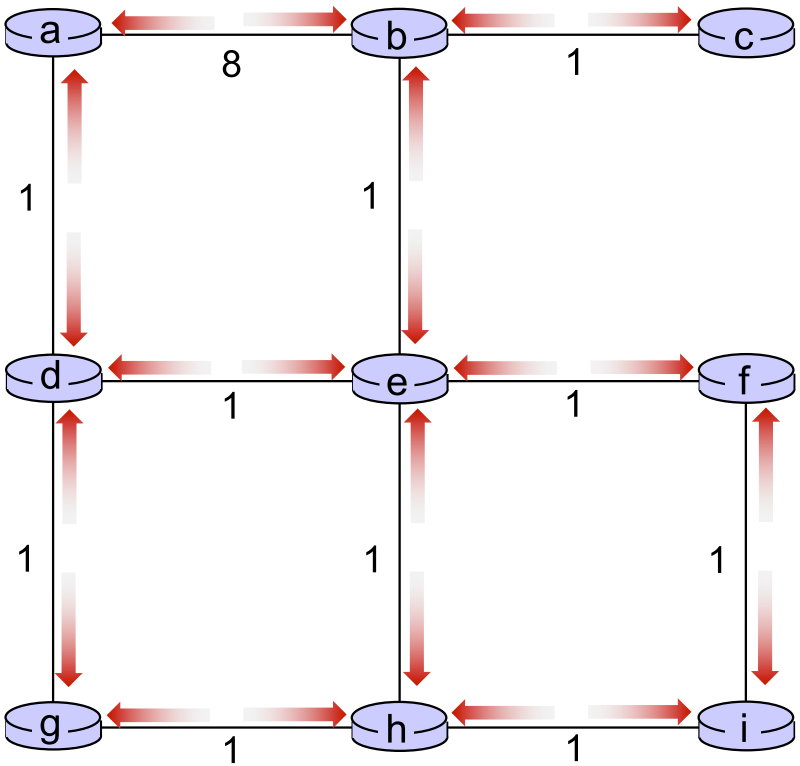

- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

# Distance vector example: iteration

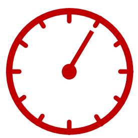
t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



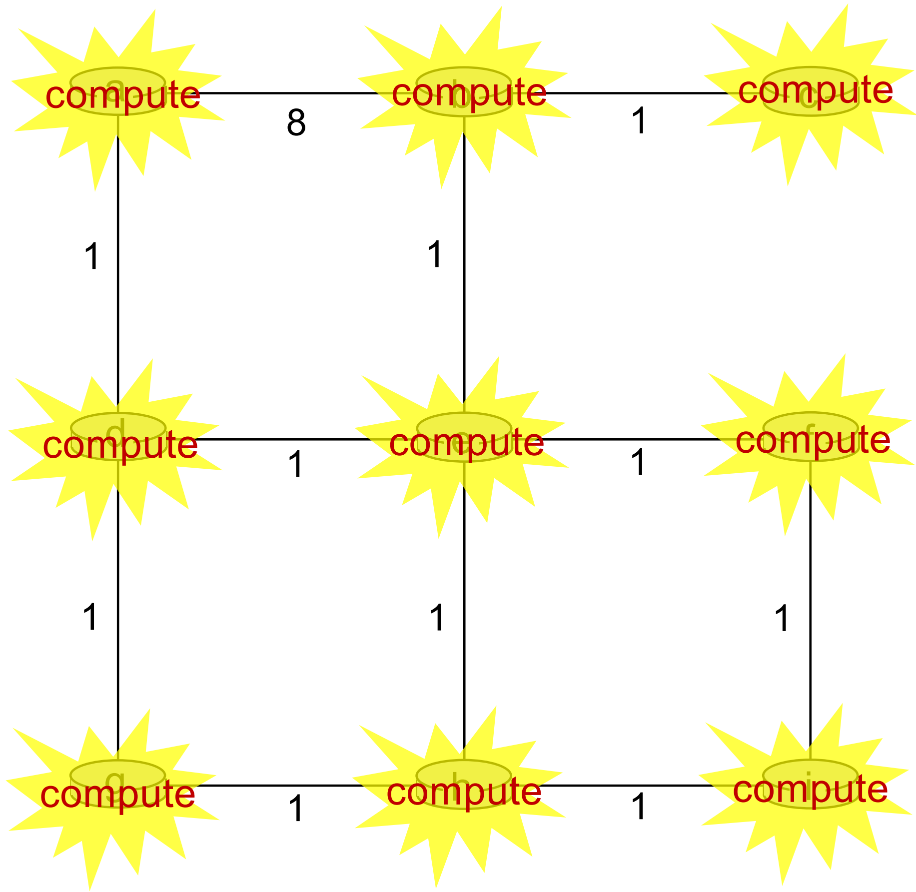
# Distance vector example: iteration



t=1

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

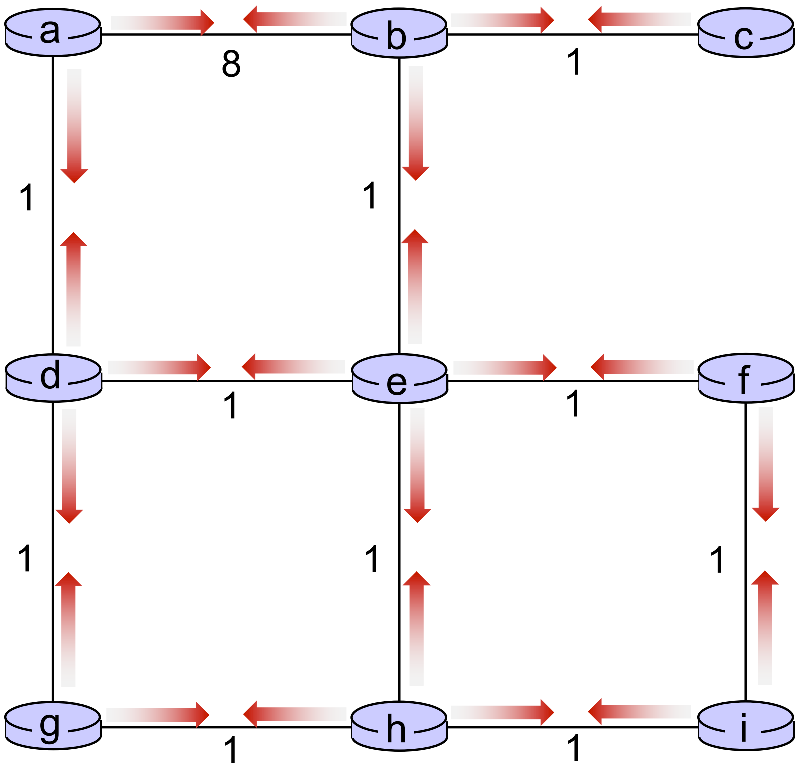


# Distance vector example: iteration

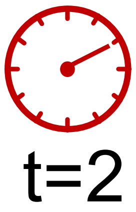
t=1

All nodes:

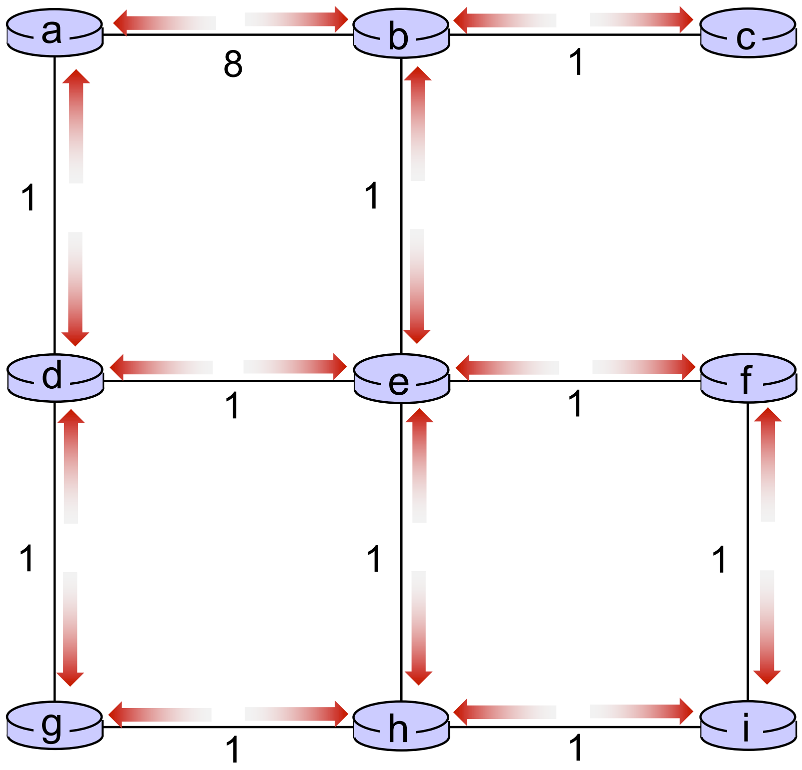
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



# Distance vector example: iteration




All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors

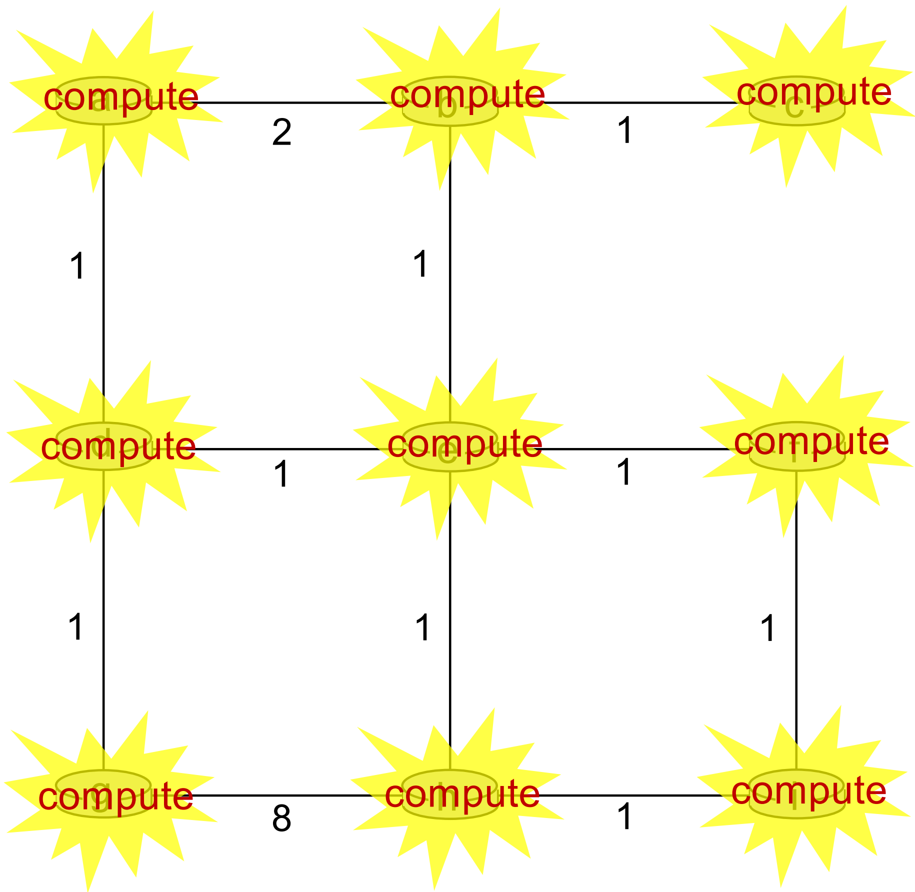


# Distance vector example: iteration

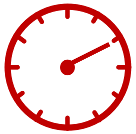
t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



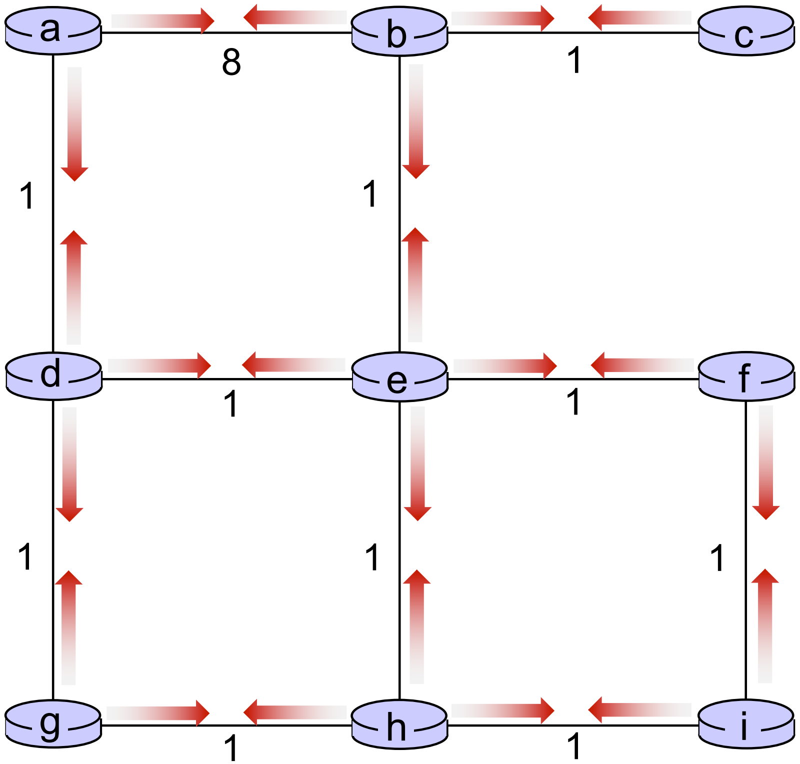
# Distance vector example: iteration



t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



# Distance vector example: iteration

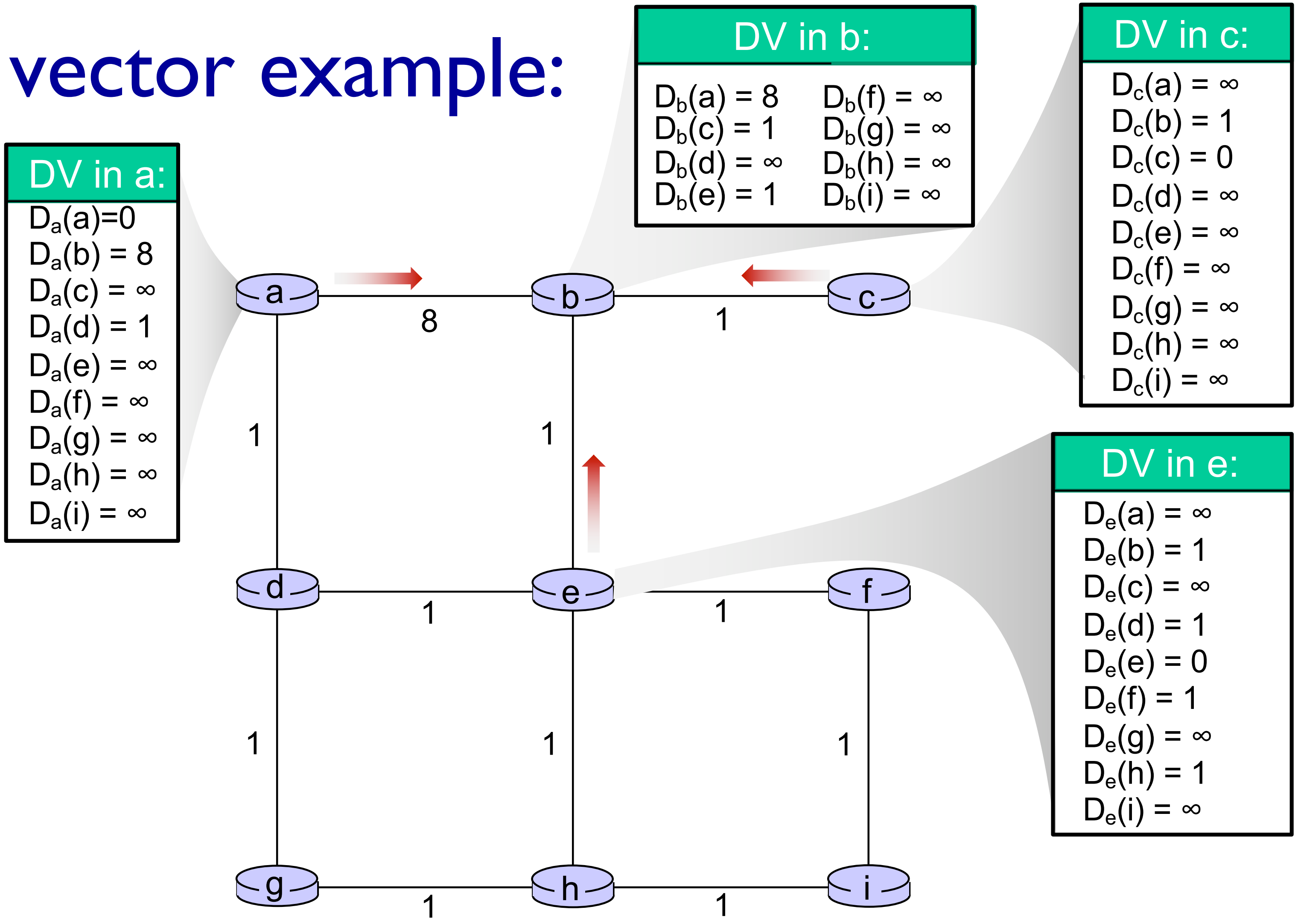
…. and so on

Let's next take a look at the iterative *computations* at nodes

# Distance vector example:

t=1

- b receives DVs from a, c, e



# Distance vector example: c

**DV in b:**

$D_b(a) = 8$ $D_b(f) = \infty$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = \infty$ $D_b(h) = \infty$
$D_b(e) = 1$ $D_b(i) = \infty$

**DV in c:**

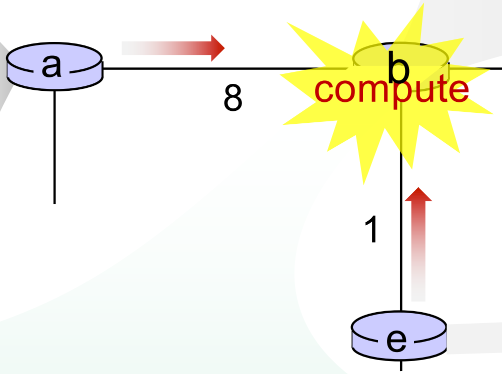
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

**DV in a:**

$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

t=1

- b receives DVs from a, c, e, computes:




**DV in e:**

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

$D_b(a) = \min\{c_{b,a}+D_a(a),\ c_{b,c}+D_c(a),\ c_{b,e}+D_e(a)\} = \min\{8,\infty,\infty\} = 8$

$D_b(c) = \min\{c_{b,a}+D_a(c),\ c_{b,c}+D_c(c),\ c_{b,e}+D_e(c)\} = \min\{\infty,1,\infty\} = 1$

$D_b(d) = \min\{c_{b,a}+D_a(d),\ c_{b,c}+D_c(d),\ c_{b,e}+D_e(d)\} = \min\{9,2,\infty\} = 2$

$D_b(e) = \min\{c_{b,a}+D_a(e),\ c_{b,c}+D_c(e),\ c_{b,e}+D_e(e)\} = \min\{\infty,\infty,1\} = 1$

$D_b(f) = \min\{c_{b,a}+D_a(f),\ c_{b,c}+D_c(f),\ c_{b,e}+D_e(f)\} = \min\{\infty,\infty,2\} = 2$

$D_b(g) = \min\{c_{b,a}+D_a(g),\ c_{b,c}+D_c(g),\ c_{b,e}+D_e(g)\} = \min\{\infty,\ \infty,\ \infty\} = \infty$

$D_b(h) = \min\{c_{b,a}+D_a(h),\ c_{b,c}+D_c(h),\ c_{b,e}+D_e(h)\} = \min\{\infty,\ \infty,\ 2\} = 2$

$D_b(i) = \min\{c_{b,a}+D_a(i),\ c_{b,c}+D_c(i),\ c_{b,e}+D_e(i)\} = \min\{\infty,\ \infty,\ \infty\} = \infty$

**DV in b:**

$D_b(a) = 8$ $D_b(f) =2$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = 2$ $D_b(h) = 2$
$D_b(e) = 1$ $D_b(i) = \infty$

# Distance vector example:

## t=1

- c receives DVs from b

**DV in a:**

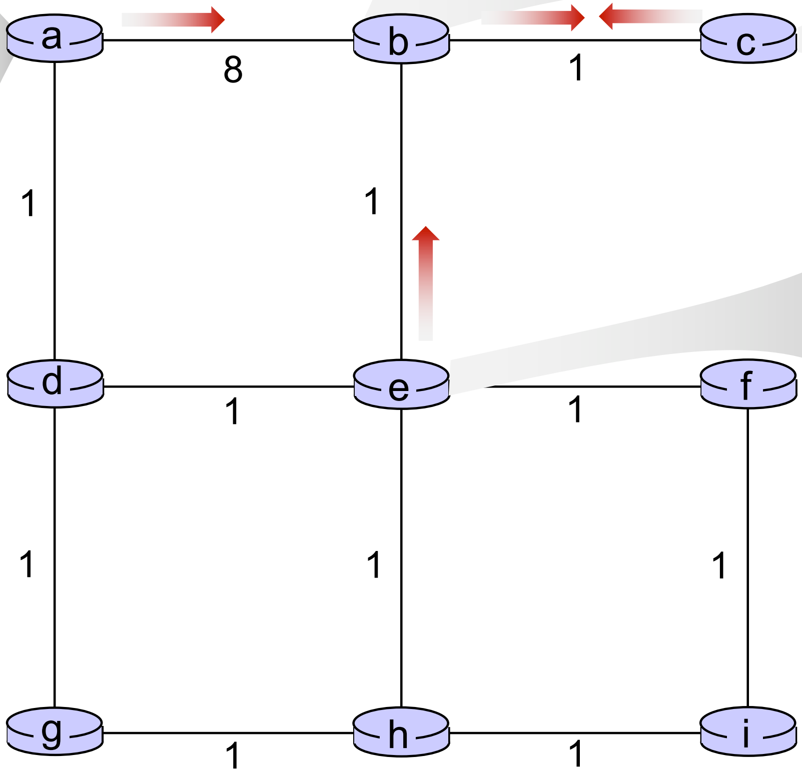
$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

**DV in b:**

$D_b(a) = 8$  $D_b(f) = \infty$
$D_b(c) = 1$  $D_b(g) = \infty$
$D_b(d) = \infty$  $D_b(h) = \infty$
$D_b(e) = 1$  $D_b(i) = \infty$

**DV in c:**

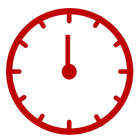
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$




**DV in e:**

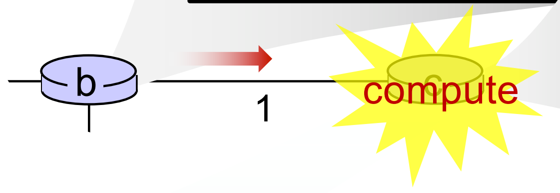
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

# Distance vector example:



t=1

- c receives DVs from b computes:




**DV in b:**

| | |
|---|---|
| $D_b(a) = 8$ | $D_b(f) = \infty$ |
| $D_b(c) = 1$ | $D_b(g) = \infty$ |
| $D_b(d) = \infty$ | $D_b(h) = \infty$ |
| $D_b(e) = 1$ | $D_b(i) = \infty$ |

**DV in c:**

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

$D_c(a) = \min\{c_{c,b}+D_b(a)\} = 1 + 8 = 9$

$D_c(b) = \min\{c_{c,b}+D_b(b)\} = 1 + 0 = 1$

$D_c(d) = \min\{c_{c,b}+D_b(d)\} = 1+ \infty = \infty$

$D_c(e) = \min\{c_{c,b}+D_b(e)\} = 1 + 1 = 2$

$D_c(f) = \min\{c_{c,b}+D_b(f)\} = 1+ \infty = \infty$

$D_c(g) = \min\{c_{c,b}+D_b(g)\} = 1+ \infty = \infty$

$D_c(h) = \min\{c_{bc,b}+D_b(h)\} = 1+ \infty = \infty$

$D_c(i) = \min\{c_{c,b}+D_b(i)\} = 1+ \infty = \infty$

**DV in c:**

$D_c(a) = 9$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = 2$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

# Distance vector example:

t=1

- e receives DVs from b, d, f, h

**DV in d:**

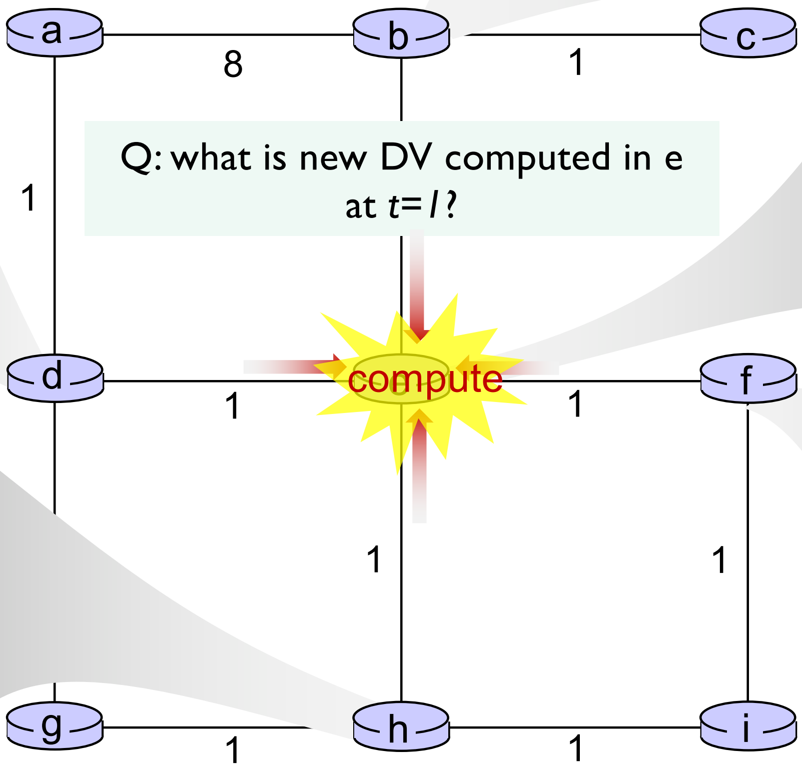
$D_c(a) = 1$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = 0$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = \infty$
$D_c(i) = \infty$

**DV in h:**

$D_c(a) = \infty$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = \infty$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = 0$
$D_c(i) = 1$

**DV in b:**

$D_b(a) = 8$　$D_b(f) = \infty$
$D_b(c) = 1$　$D_b(g) = \infty$
$D_b(d) = \infty$　$D_b(h) = \infty$
$D_b(e) = 1$　$D_b(i) = \infty$




**DV in e:**

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

**DV in f:**

$D_c(a) = \infty$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = \infty$
$D_c(e) = 1$
$D_c(f) = 0$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = 1$

# Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

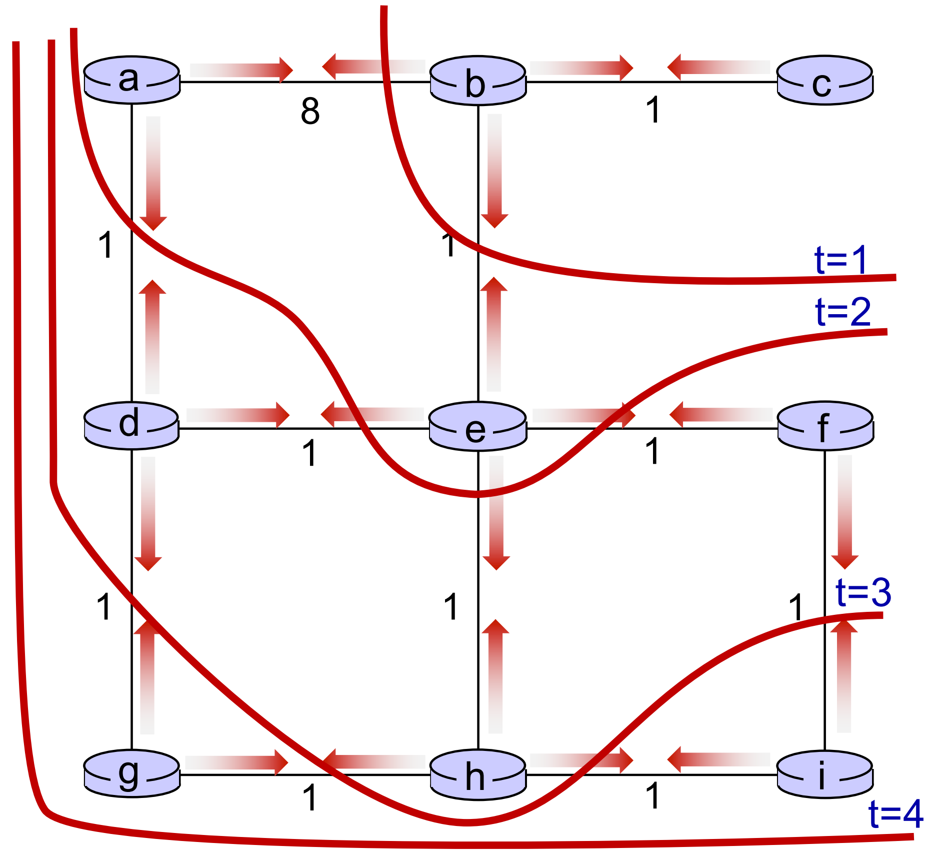
t=0 c's state at t=0 is at c only

t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b

t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well

t=3 c's state at t=0 may influence distance vector computations up to **3** hops away, i.e., at b,a,e and now at c,f,h as well

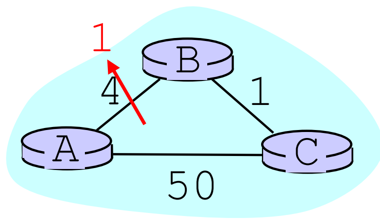
t=4 c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at b,a,e, c, f, h and now at g,i as well



# Problems with Distance Vector

- A number of problems can occur in a network using distance vector algorithm
- Most of these problems are caused by slow convergence or routers converging on incorrect information
- *Convergence* is the time during which all routers come to an agreement about the best paths through the internetwork
  - whenever topology changes there is a period of instability in the network as the routers converge
- Reacts rapidly to good news, but leisurely to bad news

# DV: Link Cost Changes




NOTE: DIFFERENT REPRESENTATION FROM BEFORE. YELLOW ENTRIES ARE THE DV

(rows = to, columns = via)

**Stable state**

Node A

| | B | C |
|---|---|---|
| B | 4 | 51 |
| C | 5 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 4 | 6 |
| C | 9 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

**A-B changed**

Node A

| | B | C |
|---|---|---|
| B | 1 | 51 |
| C | 2 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 1 | 6 |
| C | 6 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

**A sends its DV to B, C**

Node A

| | B | C |
|---|---|---|
| B | 1 | 51 |
| C | [illegible] | [illegible] |

Node B

| | [illegible] | [illegible] |
|---|---|---|
| [illegible] | [illegible] | [illegible] |
| C | 3 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 51 | 1 |

**B sends its DV to A, C**

Node A

| | B | C |
|---|---|---|
| B | 1 | 51 |
| C | [illegible] | [illegible] |

Node B

| | [illegible] | [illegible] |
|---|---|---|
| [illegible] | [illegible] | [illegible] |
| C | 3 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 2 |
| B | 51 | 1 |

**C sends its DV to A, B**

Node A

| | B | C |
|---|---|---|
| B | 1 | 51 |
| C | [illegible] | 50 |

Node B

| | [illegible] | C |
|---|---|---|
| [illegible] | [illegible] | 3 |
| C | 3 | 1 |

Node C

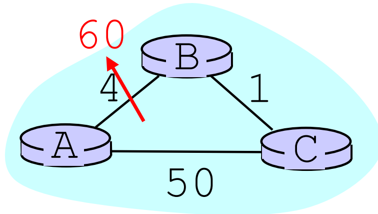
| | A | B |
|---|---|---|
| A | 50 | 2 |
| B | 51 | 1 |

deduct 3 from distances $dist_B(A,*)$ and $dist_A(B,*)$

Link cost changes here

**"good news travels fast"**

# DV: Link Cost Changes




Stable state | A-B changed

via

**Node A** (to)

Stable state:

| | B | C |
|---|---|---|
| B | 4 | 51 |
| C | 5 | 50 |

A-B changed:

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

**Node B**

Stable state:

| | A | C |
|---|---|---|
| A | 4 | 6 |
| C | 9 | 1 |

A-B changed:

| | A | C |
|---|---|---|
| A | 60 | 6 |
| C | 65 | 1 |

**Node C**

Stable state:

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

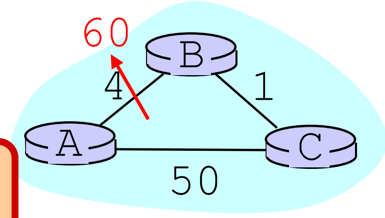
A-B changed:

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

add 56 to distances
$dist_B(A,*)$ and $dist_A(B,*)$

**Link cost changes here**

# DV: Link Cost Changes




**This is the "Counting to Infinity" Problem**

(Rows: "to"; columns: "via")

**Stable state**

Node A

| | B | C |
|---|---|---|
| B | 4 | 51 |
| C | 5 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 4 | 6 |
| C | 9 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

**A-B changed**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | 6 |
| C | 65 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

**A sends its DV to B, C**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | 6 |
| C | 110 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 101 | 1 |

**B sends its DV to A, C**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | 6 |
| C | 110 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 7 |
| B | 101 | 1 |

**C sends its DV to A, B**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | 8 |
| C | 110 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 7 |
| B | 101 | 1 |

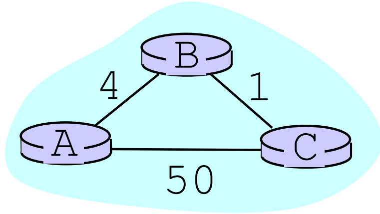
**Link cost changes here**

**"bad news travels slowly"**
**(not yet converged)**

# The “Poisoned Reverse” Rule

- Heuristic to avoid count-to-infinity
- If B routes via C to get to A:
  - B tells C its (B’s) distance to A is infinite
    (so C won’t route to A via B)

# DV: Poisoned Reverse




*If B routes through C to get to A:*

*B tells C its (B's) distance to A is infinite*

Stable state

via (columns) / to (rows)

**Node A**

| | B | C |
|---|---|---|
| B | 4 | 51 |
| C | 5 | 50 |

**Node B**

| | A | C |
|---|---|---|
| A | 4 | ∞ |
| C | ∞ | 1 |

**Node C**

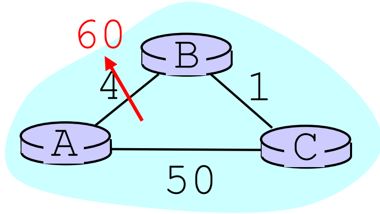
| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

| Mindist | Mindist |
|---|---|
| 4 | ∞ |
| 5 | ∞ |

| Mindist | Mindist |
|---|---|
| 5 | ∞ |
| 1 | ∞ |

# DV: Poisoned Reverse

*If B routes through C to get to A:*
*B tells C its (B's) distance to A is infinite*




Stable state | A-B changed

**Node A** (rows: to; columns: via)

Stable state:

| | B | C |
|---|---|---|
| B | 4 | 51 |
| C | 5 | 50 |

A-B changed:

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

**Node B**

Stable state:

| | A | C |
|---|---|---|
| A | 4 | ∞ |
| C | ∞ | 1 |

A-B changed:

| | A | C |
|---|---|---|
| A | 60 | 6 |
| C | 65 | 1 |

**Node C**

Stable state:

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

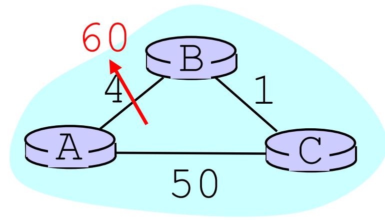
A-B changed:

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

**Link cost changes here**

# DV: Poisoned Reverse

*If B routes through C to get to A:*
*B tells C its (B's) distance to A is infinite*




(Tables: rows = "to", columns = "via")

**Stable state**

Node A

| | B | C |
|---|---|---|
| B | 4 | 51 |
| C | 5 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 4 | ∞ |
| C | ∞ | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

**A-B changed**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | ∞ |
| C | ∞ | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

**A sends its DV to B, C**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | ∞ |
| C | 110 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | ∞ | 1 |

**B sends its DV to A, C**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

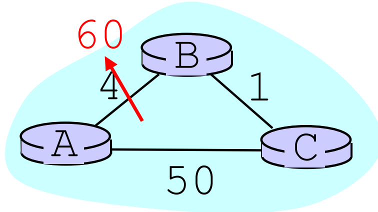
| | A | C |
|---|---|---|
| A | 60 | ∞ |
| C | 110 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 7 |
| B | ∞ | 1 |

**Link cost changes here**

# DV: Poisoned Reverse




*If B routes through C to get to A:*

*B tells C its (B's) distance to A is infinite*

Stable state | A-B changed | A sends its DV to B, C | B sends its DV to A, C

(rows: to; columns: via)

**Node A**

Stable state:

| | B | C |
|---|---|---|
| B | 4 | 51 |
| C | 5 | 50 |

A-B changed:

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

A sends its DV to B, C:

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

B sends its DV to A, C:

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

**Node B**

Stable state:

| | A | C |
|---|---|---|
| A | 4 | ∞ |
| C | ∞ | 1 |

A-B changed:

| | A | C |
|---|---|---|
| A | 60 | ∞ |
| C | ∞ | 1 |

A sends its DV to B, C:

| | A | C |
|---|---|---|
| A | 60 | ∞ |
| C | 110 | 1 |

B sends its DV to A, C:

| | A | C |
|---|---|---|
| A | 60 | ∞ |
| C | 110 | 1 |

**Node C**

Stable state:

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

A-B changed:

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

A sends its DV to B, C:

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | ∞ | 1 |

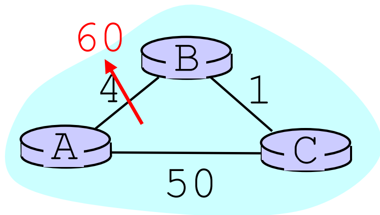
B sends its DV to A, C:

| | A | B |
|---|---|---|
| A | 50 | 61 |
| B | ∞ | 1 |

**Link cost changes here**

# DV: Poisoned Reverse

*If B routes through C to get to A:*
*B tells C its (B's) distance to A is infinite*




(Rows = to, columns = via)

**Stable state**

Node A

| | B | C |
|---|---|---|
| B | 4 | 51 |
| C | 5 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 4 | ∞ |
| C | ∞ | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

**A-B changed**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | ∞ |
| C | ∞ | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | 54 | 1 |

**A sends its DV to B, C**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | ∞ |
| C | 110 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 5 |
| B | ∞ | 1 |

**B sends its DV to A, C**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | ∞ |
| C | 110 | 1 |

Node C

| | A | B |
|---|---|---|
| A | 50 | 61 |
| B | ∞ | 1 |

**C sends its DV to A, B**

Node A

| | B | C |
|---|---|---|
| B | 60 | 51 |
| C | 61 | 50 |

Node B

| | A | C |
|---|---|---|
| A | 60 | 51 |
| C | 110 | 1 |

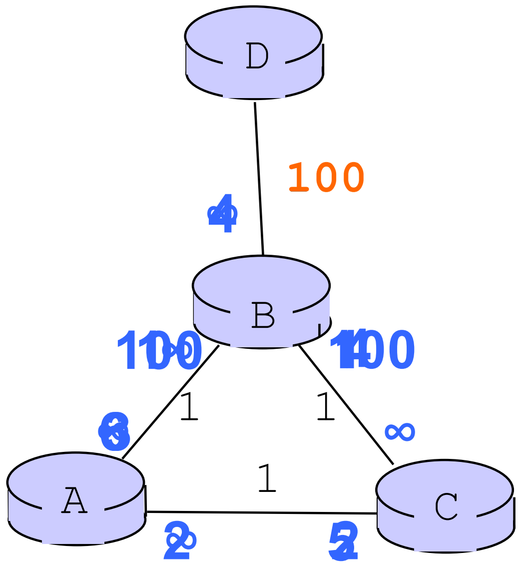
Node C

| | A | B |
|---|---|---|
| A | 50 | ∞ |
| B | ∞ | 1 |

**Link cost changes here**

Converges after C receives another update from B

# Will Poison-Reverse Completely Solve the Count-to-Infinity Problem?




Numbers in blue denote the best cost
to destination D advertised along the link

# Comparison of LS and DV algorithms

**message complexity**

LS: $n$ routers, $O(n^2)$ messages sent

DV: exchange between neighbors; convergence time varies

**speed of convergence**

LS: $O(n^2)$ algorithm, $O(n^2)$ messages

- may have oscillations

DV: convergence time varies

- may have routing loops
- count-to-infinity problem

**robustness:** what happens if router malfunctions, or is compromised?

LS:

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

DV:

- DV router can advertise incorrect *path* cost ("I have a *really* low cost path to everywhere"): black-holing
- each router's table used by others: error propagate thru network

# Real Protocols

*Link State*

Open Shortest Path First (OSPF)

Intermediate system to intermediate system (IS-IS)

*Distance Vector*

Routing Information Protocol (RIP)

Interior Gateway Routing Protocol (IGRP-Cisco)

Border Gateway Protocol (BGP) - variant

# Quiz: Link-state routing

- In link state routing, each node sends information of its direct links (i.e., link state) to _________?

A. Immediate neighbours
B. All nodes in the network
C. Any one neighbor
D. No one

**Answer: B**

**www.pollev.com/salil**

# Quiz: Distance-vector routing

- In distance vector routing, each node shares its distance table with __________?

A. All Immediate neighbours
B. All nodes in the network
C. Any one neighbor
D. No one

**Answer: A**

**www.pollev.com/salil**

# Quiz: Distance-vector routing

- Which of the following is true of distance vector routing?

A. Convergence delay depends on the topology (nodes and links) and link weights
B. Convergence delay depends on the number of nodes and links
C. Each node knows the entire topology
D. A and C
E. B and C

**Answer: A**

**www.pollev.com/salil**

# Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

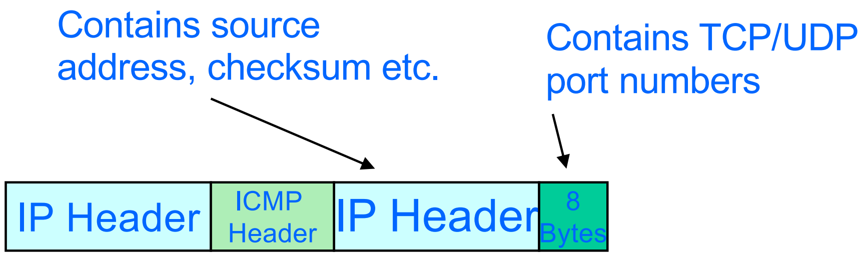
- link state
- distance vector
- hierarchical routing

5.6 ICMP: The Internet Control Message Protocol

Self study (not on exam)

# ICMP: Internet Control Message Protocol

- Used by hosts & routers to communicate network level infromation
  - Error reporting: unreachable host, network, port
  - Echo request/reply (used by ping)
- Works above IP layer
  - ICMP messages carried in IP datagrams
- ICMP message: type, code plus IP header and first 8 bytes of IP datagram payload causing error



# ICMP: Internet Control Message Protocol

| Type | Code | Description |
|---|---|---|
| 0 | 0 | echo reply(ping) |
| 3 | 0 | dest. network unreachable |
| 3 | 1 | dest host unreachable |
| 3 | 3 | dest port unreachable |
| 3 | 4 | frag needed; DF set |
| 8 | 0 | echo request(ping) |
| 11 | 0 | TTL expired |
| 11 | 1 | frag reassembly time exceeded |
| 12 | 0 | bad IP header |

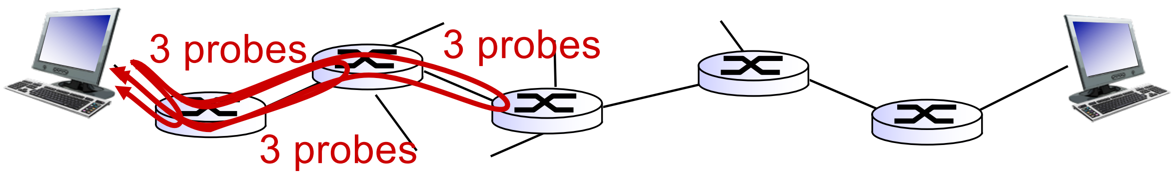
# Traceroute and ICMP

- Source sends series of UDP segments to dest
  - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number
- When *n*th set of datagrams arrives to nth router:
  - router discards datagrams
  - and sends source ICMP messages (type 11, code 0)
  - ICMP messages includes IP address of router

- when ICMP messages arrives, source records RTTs

*stopping criteria:*

- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops



# Summary

- Network Layer: Data Plane
  - Overview
  - IP
- Network Layer: Control Plane
  - Routing Protocols
    - Link—state
    - Distance Vector
  - ICMP