Computer Networks

Wenzhong Li, Chen Tian

Nanjing University

Material with thanks to James F. Kurose, Mosharaf Chowdhury, and other colleagues.

Chapter 3. Network Layer

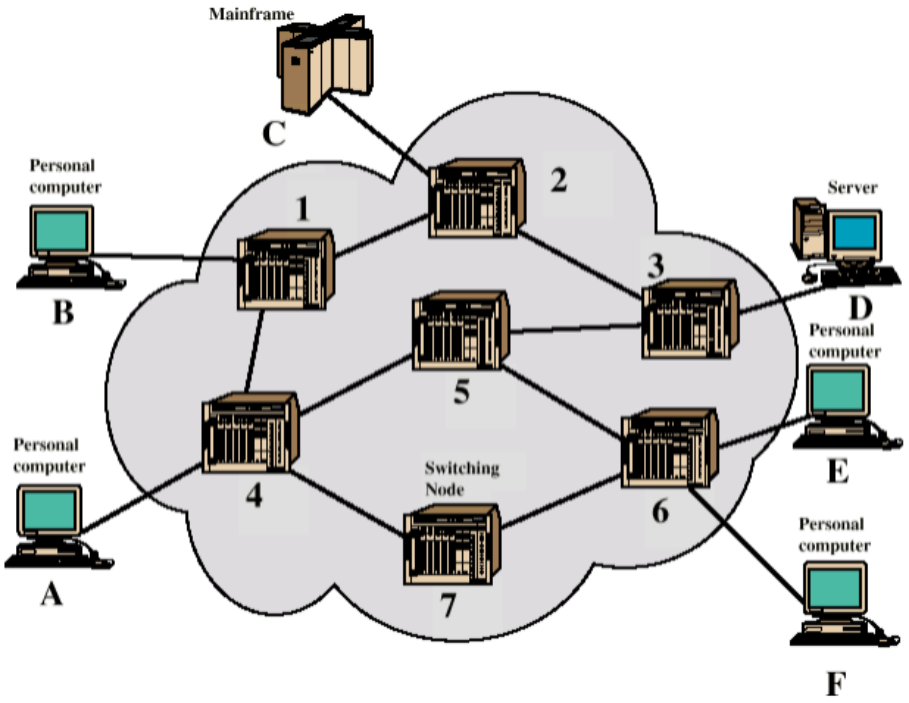
- Network Layer Functions
- IP Protocol Basic
- IP Protocol Suit
- Routing Fundamentals
- Internet Routing Protocols
- IP Multicasting

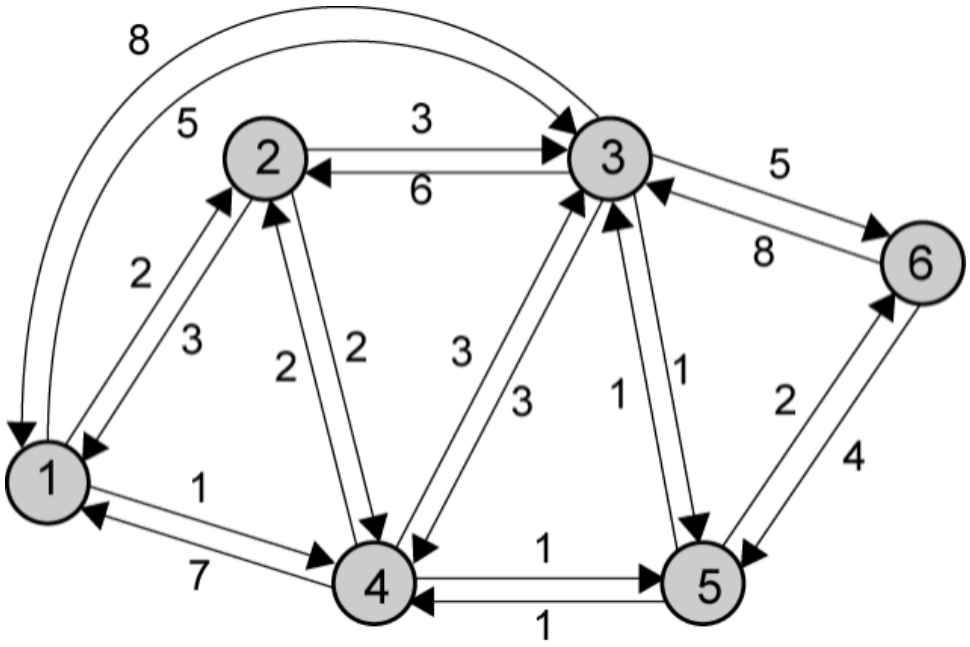
Routing Fundamentals

Routing

- Objective
  - Build routing tables on switches for datagram networks
  - Choose paths and build forwarding tables when setting up connections for VC networks
- Characteristics required
  - Efficiency: e.g. smallest possible line or switch
  - Resilience: peak load, switch or line failure
  - Stability: avoid oscillation

Network Abstraction



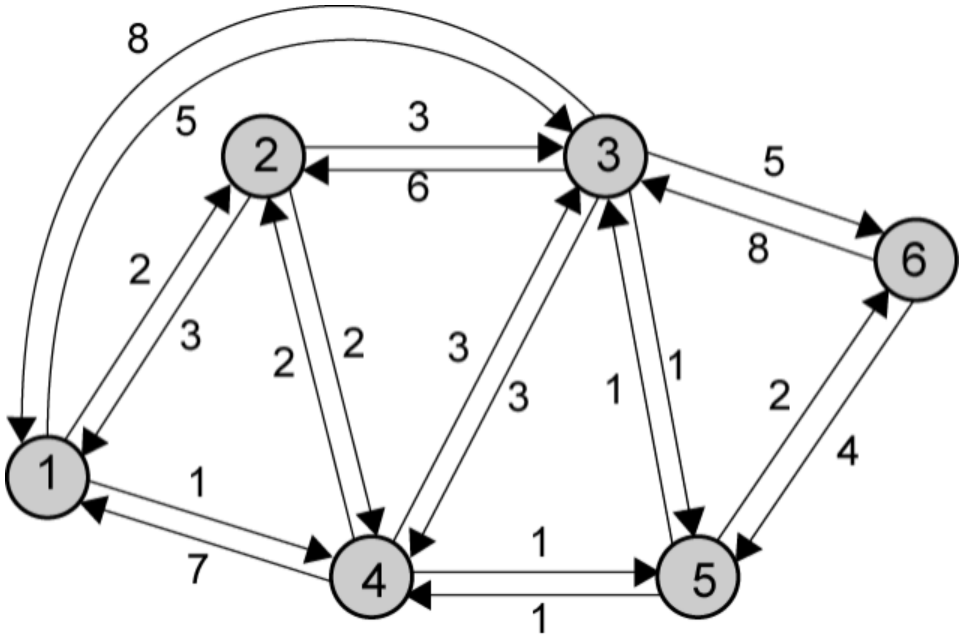


Routing Elements

- Performance criteria
- Decision time
- Decision place
- Network info source
- Network info update timing

Performance Criteria

- Minimum hop
  - e.g. 1–3–6
- Least cost
  - e.g. 1–4–5–6
- Determine cost
  - Minimum delay: queue length
  - Largest throughput: reverse of transmission rate



Decision Time and Place

- Time
  - For each packet --- datagram networks
  - At the start of each virtual circuit --- VC networks
- Place
  - Centralized
  - Source --- source routing
  - Distributed --- by each switch node

Network Info Source and Update Timing

- Info source
  - Local information
  - Adjacent switches
  - All switches in the network
- Update timing
  - Update periodically
  - Upon major changes in switches or links
  - Fixed (manual configuration)

Different Routing Strategies

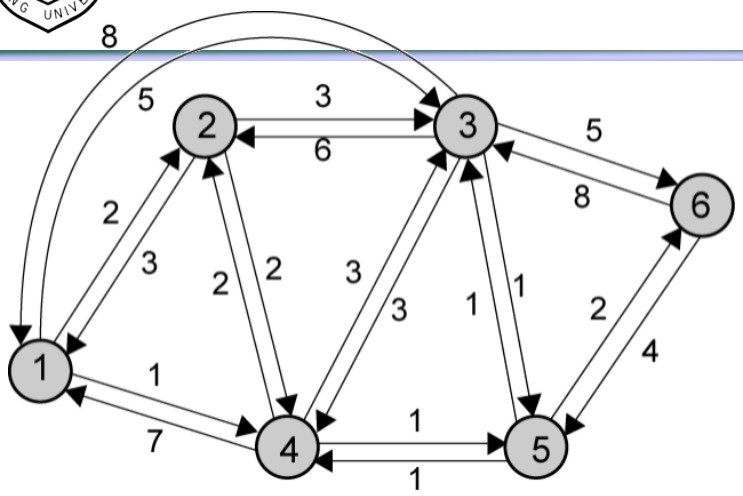
- Central (static)
  - Fixed and configured
- Distributed
  - Flooding
  - Random
  - Adaptive

Central Routing

- Single fixed route for each source to destination pair
- Determine routes using a least cost algorithm
- Routes re-config upon major changes in network topology

Fixed Routing Tables

1 → 6



CENTRAL ROUTING DIRECTORY

To Node \ From Node	1	2	3	4	5	6
1	—	1	5	2	4	5
2	2	—	5	2	4	5
3	4	3	—	5	3	5
4	4	4	5	—	4	5
5	4	4	5	5	—	5
6	4	4	5	5	6	—

Centralized

Distributed

Node 1 Directory

Destination	Next Node
2	2
3	4
4	4
5	4
6	4

Node 2 Directory

Destination	Next Node
1	1
3	3
4	4
5	4
6	4

Node 3 Directory

Destination	Next Node
1	5
2	5
4	5
5	5
6	5

Node 4 Directory

Destination	Next Node
1	2
2	2
3	5
5	5
6	5

Node 5 Directory

Destination	Next Node
1	4
2	4
3	3
4	4
6	6

Node 6 Directory

Destination	Next Node
1	5
2	5
3	5
4	5
5	5

Flooding

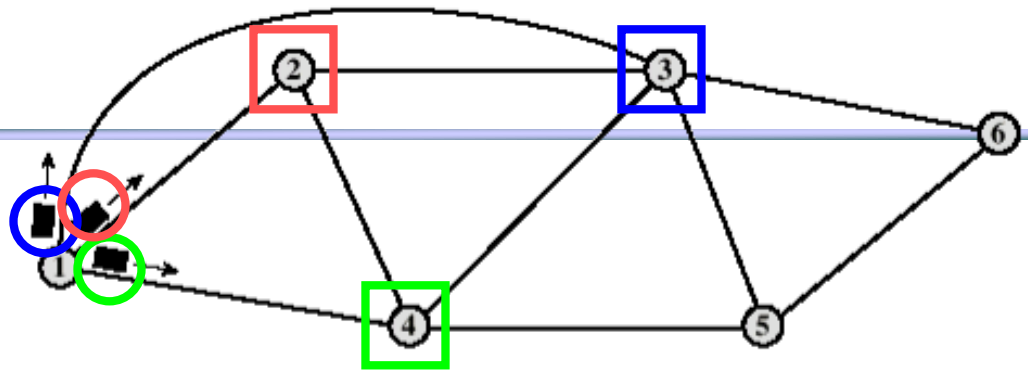
- No network info required
- Packet sent by switch to every neighbor
  - Packets retransmitted on every link except incoming link
- Eventually a number of copies will arrive at destination

- Duplicates
  - Many copies of the same packet is created
- Cycle problem
  - These copies may circling around the network forever
  - A hop count in packets can handle the problem

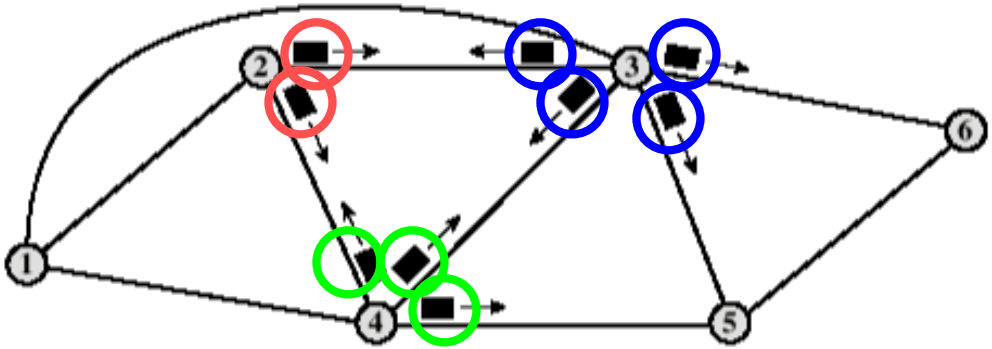
Flooding Example

Hop count = 3

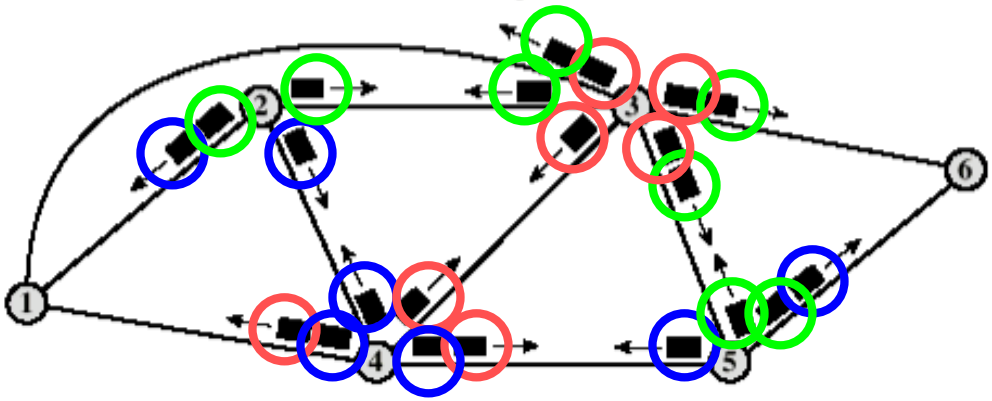
- Initial
  - 3 packets
- 1st hop
  - 9 packets
- 2nd hop
  - 23 packets



(a) First hop



(b) Second hop



(c) Third hop

Properties of Flooding

- All possible routes are tried
  - Very robust
- At least one packet will take minimum cost route
  - Can be used to set up virtual circuit
- All switches are visited
  - Useful to distribute information (e.g. routing)

Random Routing

- Node selects one outgoing path for retransmission of incoming packet
  - Selection can be random or round robin
  - Or based on probability calculation
- No network info needed
- Suitable for strongly-connected network
- Route is typically not optimal

Assign Probabilities

- $P_i = R_i / \Sigma_j R_j$
  - P_i – Probability of selecting out-link i
  - R_i – Cost factor of link i

- Possible cost factor
  - Transmission rate – for throughput
  - Reverse of queue size – for delay

Adaptive Routing

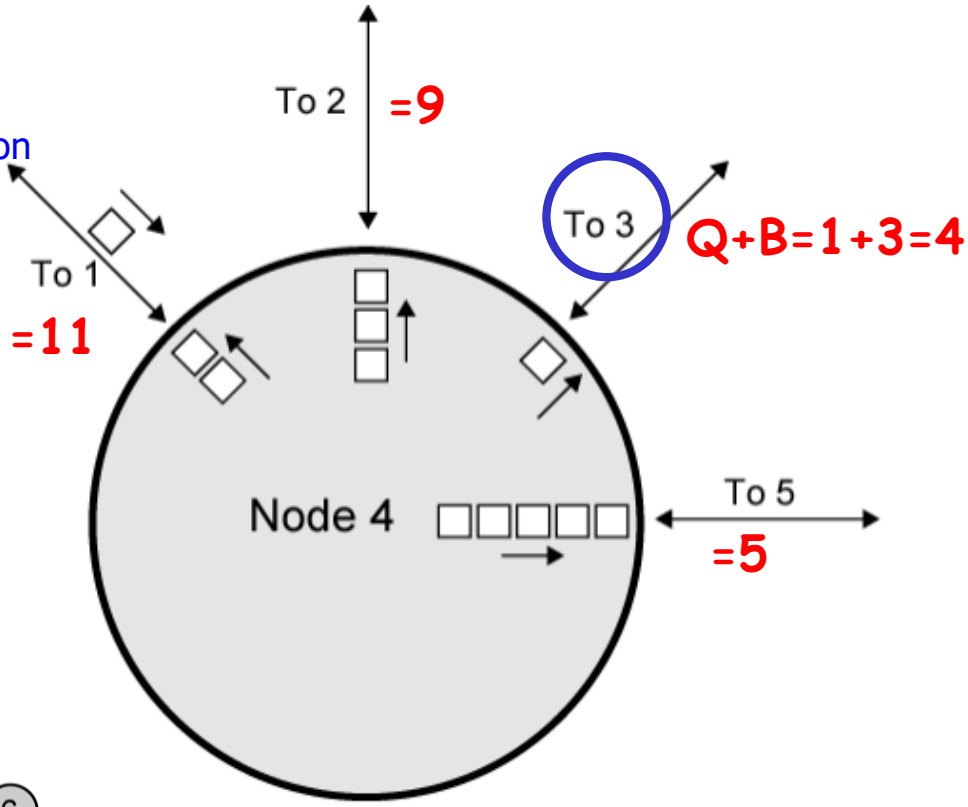
- Used by almost all packet switching networks
- Routing decisions change as conditions on the network change
- Requires info about network
  - Tradeoff between quality of network info and overhead
- Aid in congestion control

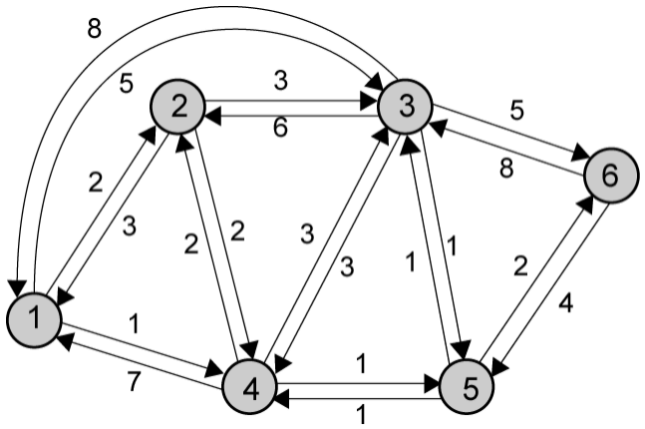
An Isolated Adaptive Routing

- Only local info used
- Strategy 1: route to the outgoing link with shortest queue length Q
  - Pros. Load balancing
  - Cons. May away from the destination
- Strategy 2: take direction into account
  - Each link has a bias B for the destination
  - Route to minimize Q+B

Node 4's Bias Table for Destination 6

Next Node	Bias
1	9
2	6
3	3
5	0





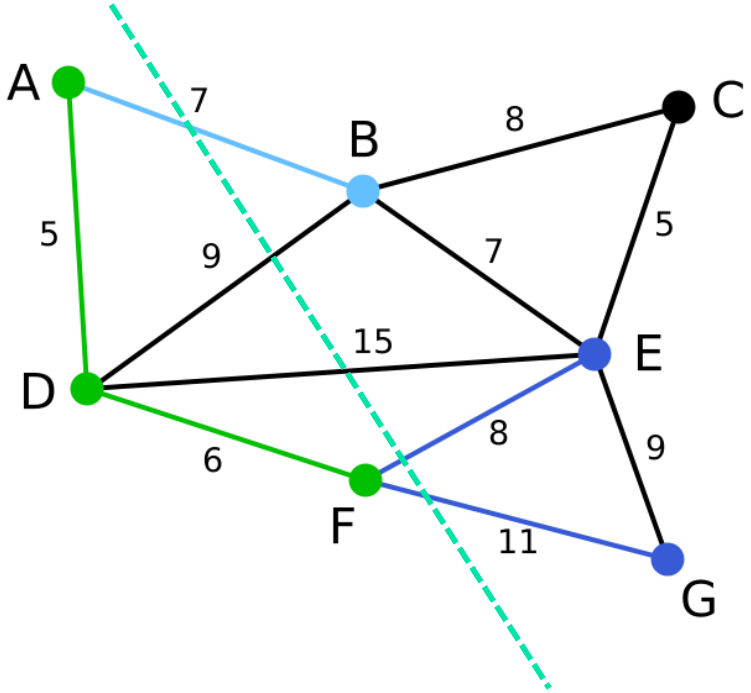
2 Least Cost Algorithms

- For each pair of nodes, find a path with the least cost

- Dijkstra's Algorithm
- Bellman-Ford Algorithm

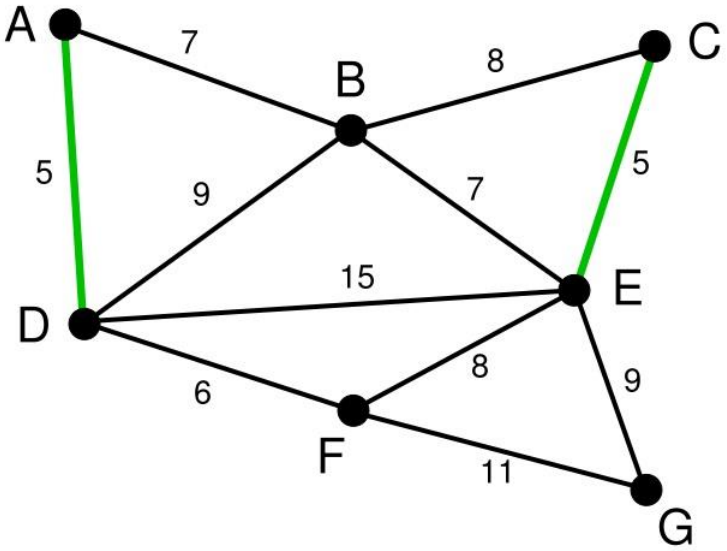
Recap: Prim's and Kruskal's algorithm

- Minimum Spanning Tree Algorithm



Prim's algorithm

Explored node set {A,D,F}
To be explored set {B,E,G}
Next node chosen: B



Kruskal's algorithm

Sort according to edge weight (ascending)
Add edges one-by-one (without loop)

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

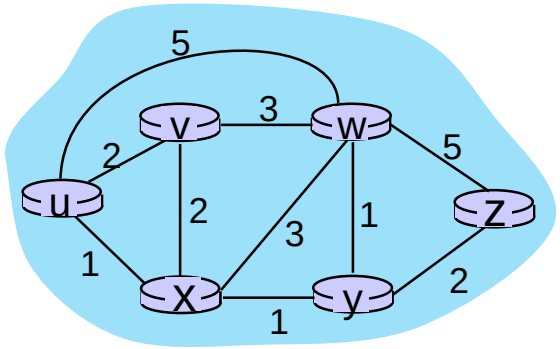
- $c_{x,y}$: direct link cost from node *x* to *y*; = ∞ if not direct neighbors
- *D(v)*: *current* estimate of cost of least-cost-path from source to destination *v*
- *p(v)*: predecessor node along path from source to *v*
- *N'*: set of nodes whose least-cost-path *definitively* known

Dijkstra's link-state routing algorithm

```
1  Initialization:
2    N' = {u}                    /* compute least cost path from u to all other nodes */
3    for all nodes v
4      if v adjacent to u        /* u initially knows direct-path-cost only to  direct neighbors  */
5        then D(v) = c_{u,v}     /* but may not be minimum cost!                                   */
6      else D(v) = ∞
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 least-path-cost to v is either old least-cost-path to v or known
14     least-cost-path to w plus direct-cost from w to v */
15   until all nodes in N'
```

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1						
2						
3						
4						
5						

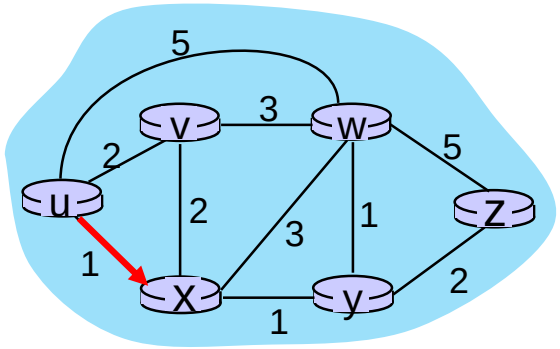


Initialization (step 0):

For all *a*: if *a* adjacent to *u* then $D(a) = c_{u,a}$

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux					
2						
3						
4						
5						



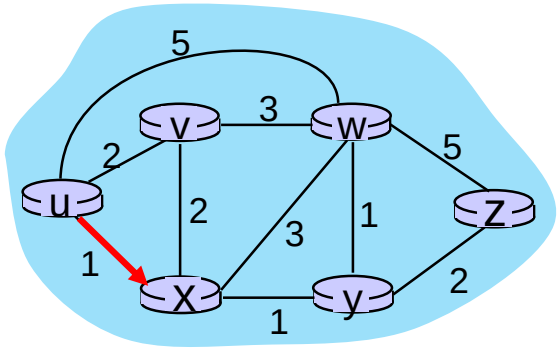
8 Loop

9 find *a* not in *N'* such that *D(a)* is a minimum

10 add *a* to *N'*

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2						
3						
4						
5						



8 *Loop*

9 find *a* not in *N'* such that *D(a)* is a minimum

10 add *a* to *N'*

11 update *D(b)* for all *b* adjacent to *a* and not in *N'* :

$D(b) = \min (D(b), D(a) + c_{a,b})$

$D(v) = \min (D(v), D(x) + c_{x,v}) = \min(2, 1+2) = 2$

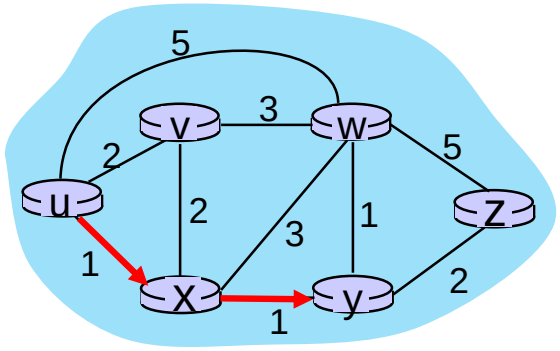
$D(w) = \min (D(w), D(x) + c_{x,w}) = \min (5, 1+3) = 4$

$D(y) = \min (D(y), D(x) + c_{x,y}) = \min(\inf, 1+1) = 2$

NEW! NEW!

Dijkstra's algorithm: an example

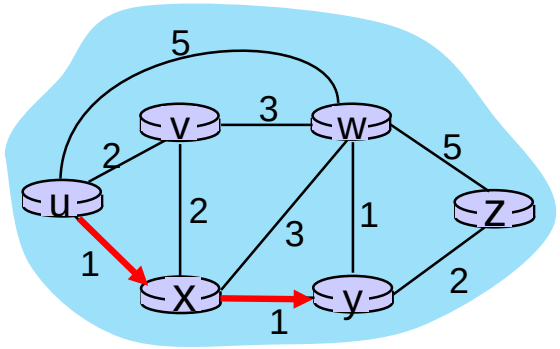
Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy					
3						
4						
5						



8 *Loop*
9 find *a* not in *N'* such that *D(a)* is a minimum
10 add *a* to *N'*

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3						
4						
5						



8 *Loop*

9 find *a* not in *N'* such that *D(a)* is a minimum

10 add *a* to *N'*

11 update *D(b)* for all *b* adjacent to *a* and not in *N'* :

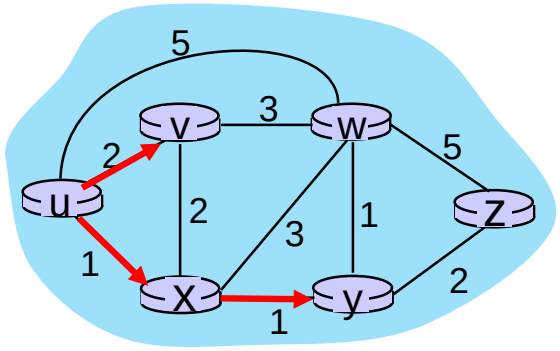
$D(b) = \min (D(b), D(a) + c_{a,b})$

$D(w) = \min (D(w), D(y) + c_{y,w}) = \min (4, 2+1) = 3$ NEW!

$D(z) = \min (D(z), D(y) + c_{y,z}) = \min(\inf, 2+2) = 4$ NEW!

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv					
4						
5						



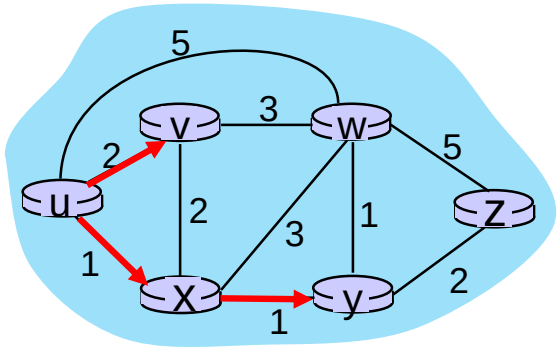
8 Loop

9 find *a* not in *N'* such that *D(a)* is a minimum

10 add *a* to *N'*

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4						
5						



8 *Loop*

9 find *a* not in *N'* such that *D(a)* is a minimum

10 add *a* to *N'*

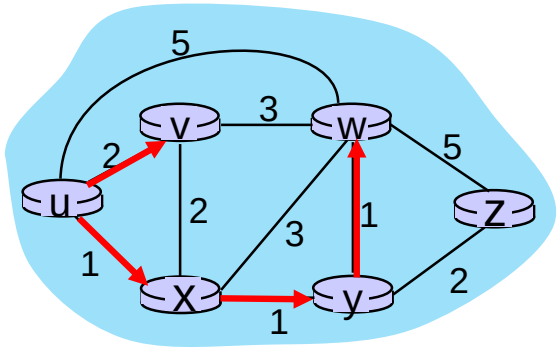
11 update *D(b)* for all *b* adjacent to *a* and not in *N'* :

$D(b) = \min (D(b), D(a) + c_{a,b})$

$D(w) = \min (D(w), D(v) + c_{v,w}) = \min (3, 2+3) = 3$

Dijkstra's algorithm: an example

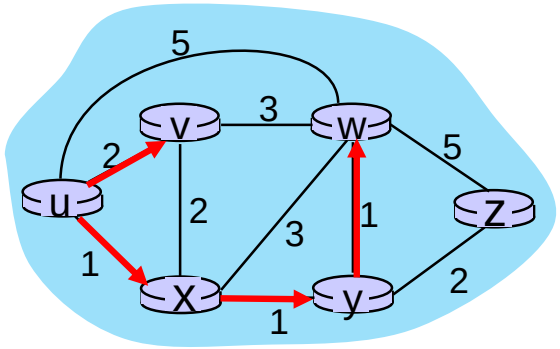
Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					
5						




```
8  Loop
9     find a not in N' such that D(a) is a minimum
10    add a to N'
```

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5						



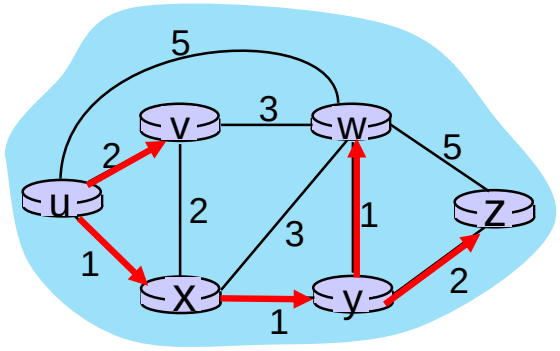
8 *Loop*
9 find *a* not in *N'* such that *D(a)* is a minimum
10 add *a* to *N'*
11 update *D(b)* for all *b* adjacent to *a* and not in *N'* :

$$D(b) = \min (D(b), D(a) + c_{a,b})$$

$D(z) = \min (D(z), D(w) + c_{w,z}) = \min (4, 3+5) = 4$

Dijkstra's algorithm: an example

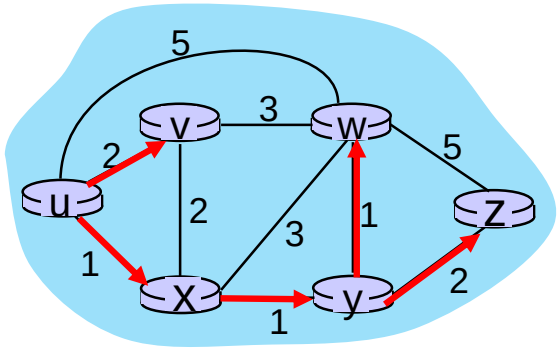
Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



8 *Loop*
9 find *a* not in *N'* such that *D(a)* is a minimum
10 add *a* to *N'*

Dijkstra's algorithm: an example

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



8 Loop

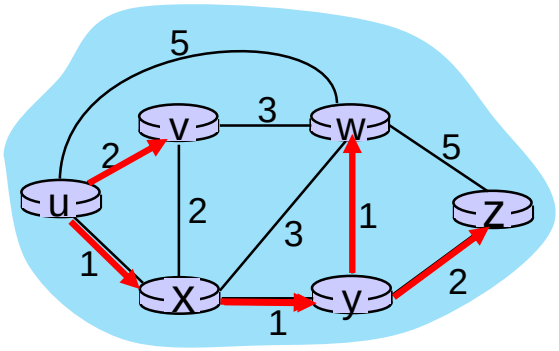
9 find *a* not in *N'* such that *D(a)* is a minimum

10 add *a* to *N'*

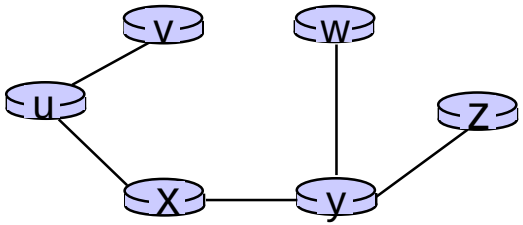
11 update *D(b)* for all *b* adjacent to *a* and not in *N'* :

$D(b) = \min (D(b), D(a) + c_{a,b})$

Dijkstra's algorithm: an example



resulting least-cost-path tree from u:



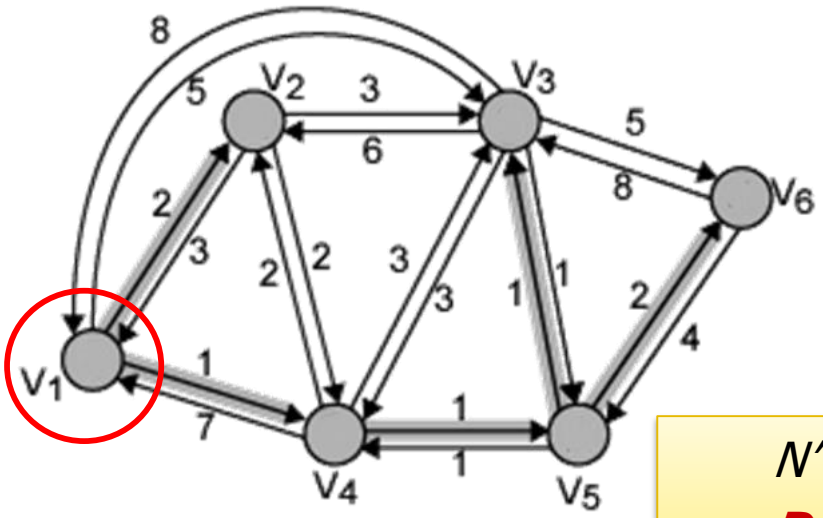
resulting forwarding table in u:

destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

route from *u* to *v* directly

route from u to all other destinations via *x*

Exercise



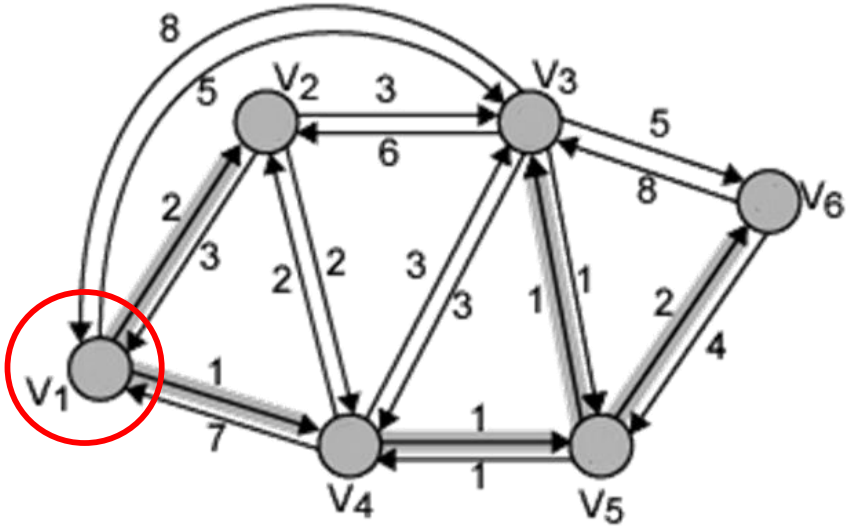
$N' = N' + \{x\}$ //x is the min cost in $N - N'$

$D(b) = \min (D(b), D(a) + c_{a,b})$ for all n $\notin N'$

No	N'	D(2)	Path	D(3)	Path	D(4)	Path	D(5)	Path	D(6)	Path
1	$\{V_1\}$	2	V_1-V_2	5	V_1-V_3	1	V_1-V_4	∞	–	∞	–
2	$\{V_1,V_4\}$	2	V_1-V_2	4	V_1-V_4-V_3			2	V_1-V_4-V_5	∞	–
3											
4											
5	{										
6	{										

Results of Dijkstra's Algorithm

No	N'	D(2)	Path	D(3)	Path	D(4)	Path	D(5)	Path	D(6)	Path
1	$\{V_1\}$	2	V_1-V_2	5	V_1-V_3	1	V_1-V_4	∞	–	∞	–
2	$\{V_1,V_4\}$	2	V_1-V_2	4	V_1-V_4-V_3			2	V_1-V_4-V_5	∞	–
3	$\{V_1, V_2, V_4\}$			4	V_1-V_4-V_3			2	V_1-V_4-V_5	∞	–
4	$\{V_1, V_2, V_4, V_5\}$			3	V_1-V_4-V_5-V_3					4	V_1-V_4-V_5-V_6
5	$\{V_1, V_2, V_3, V_4, V_5\}$									4	V_1-V_4-V_5-V_6
6	$\{V_1 V_2,V_3,V_4,V_5,V_6\}$	2	V_1-V_2	3	V_1-V_4-V_5-V_3	1	V_1-V_4	2	V_1-V_4-V_5	4	V_1-V_4-V_5-V_6



Destination	Next-Hop	Distance
V_2	V_2	2
V_3	V_4	3
V_4	V_4	1
V_5	V_4	2
V_6	V_4	4

Dijkstra's algorithm: discussion

algorithm complexity: n nodes

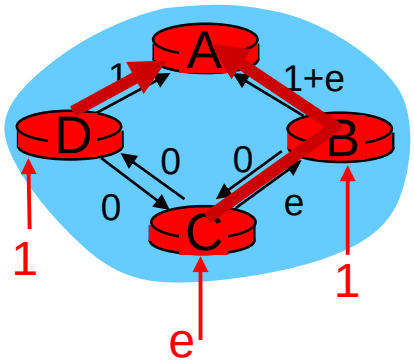
- each of n iteration: need to check all nodes, w, not in N
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n\log n)$

message complexity:

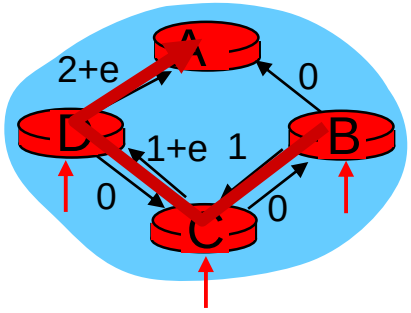
- each router must *broadcast* its link state information to other n routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Dijkstra's algorithm: oscillations possible

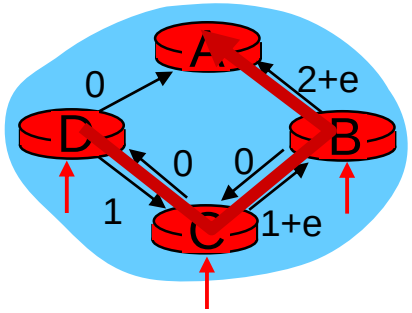
- when link costs depend on traffic volume, route oscillations possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent



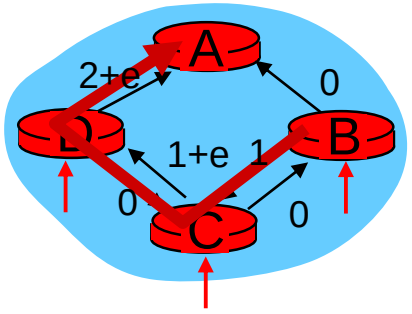
initially



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

震荡问题：解决办法-非同步运行路由算法，链路代价更新随机化

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:

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

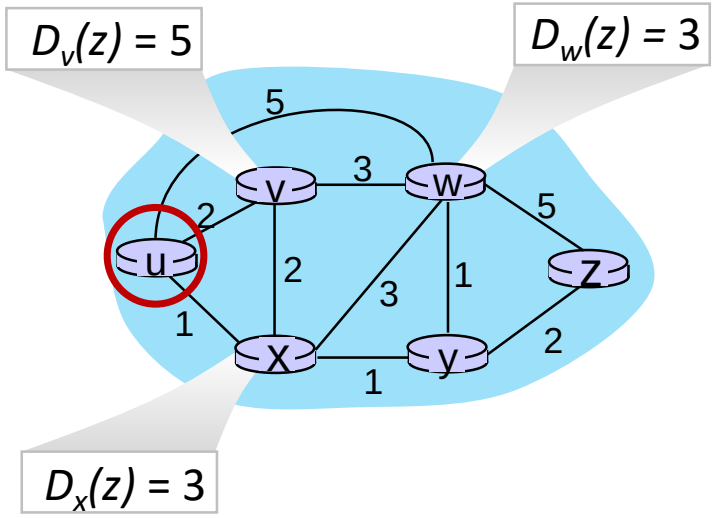
v's estimated least-cost-path cost to y

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:

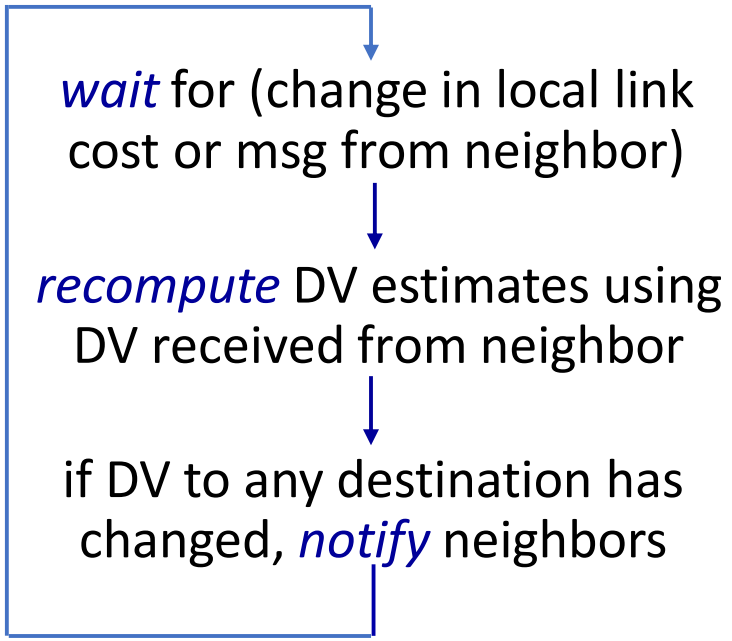
- from time-to-time, each node sends its own distance vector estimate to neighbors
- when *x* receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

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

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

Distance vector algorithm:

each node:



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

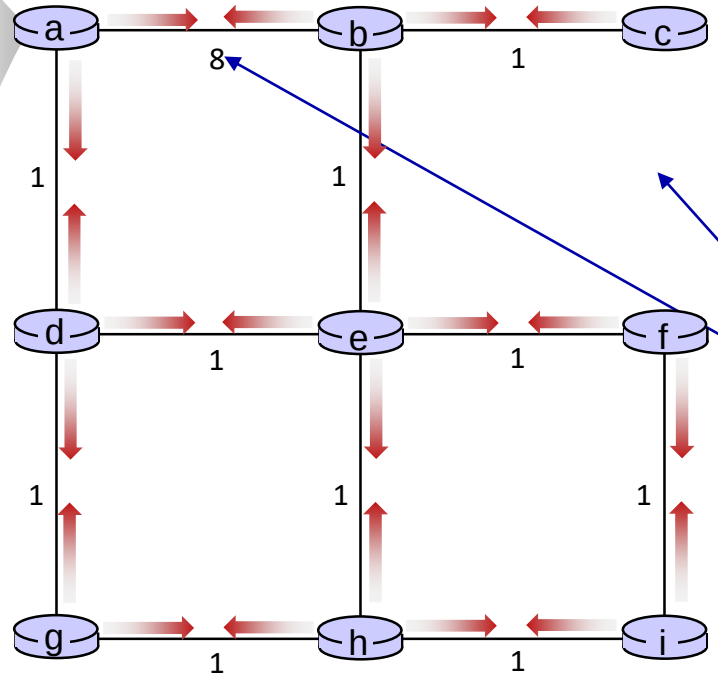
- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

Distance vector: example

t=0

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

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



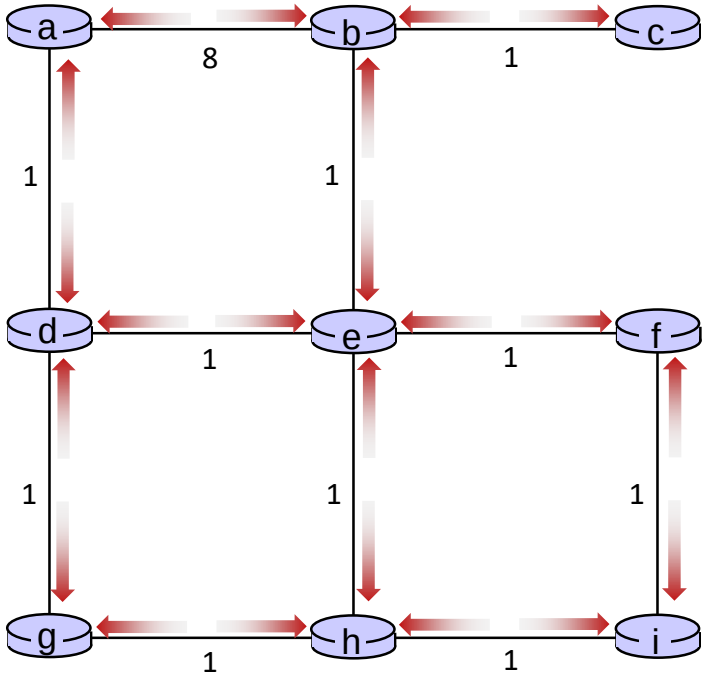
A few asymmetries:
- missing link
- larger cost

Distance vector: example

t=1

All nodes:

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

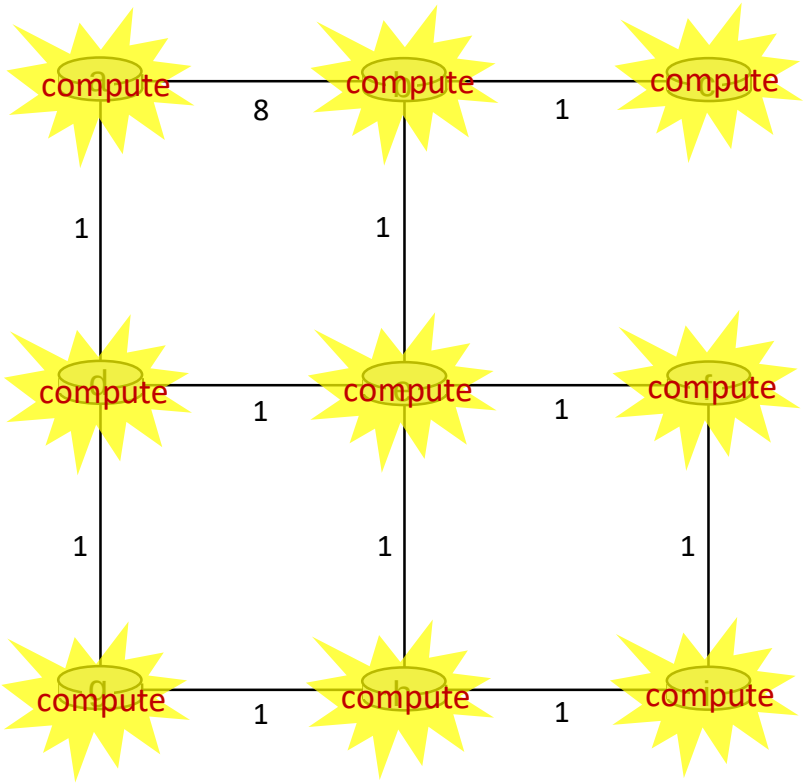


Distance vector: example

t=1

All nodes:

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

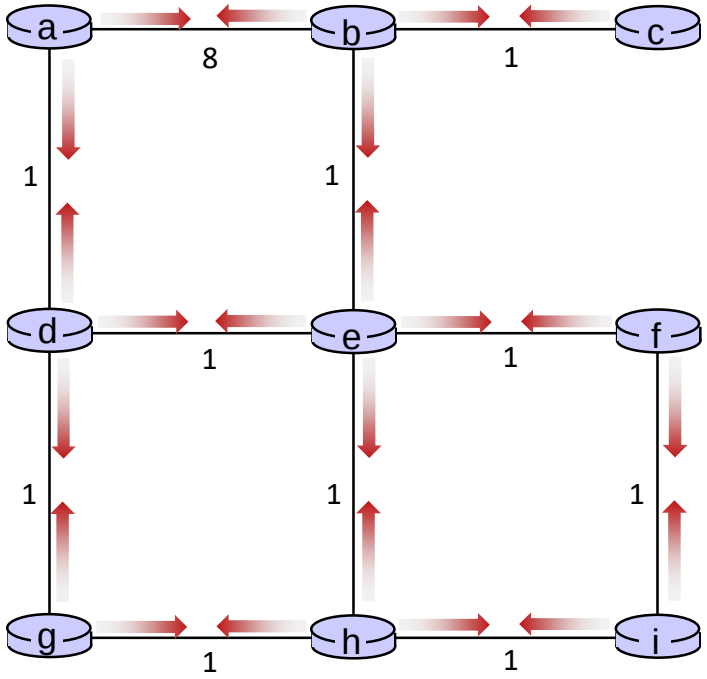


Distance vector: example

t=1

All nodes:

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

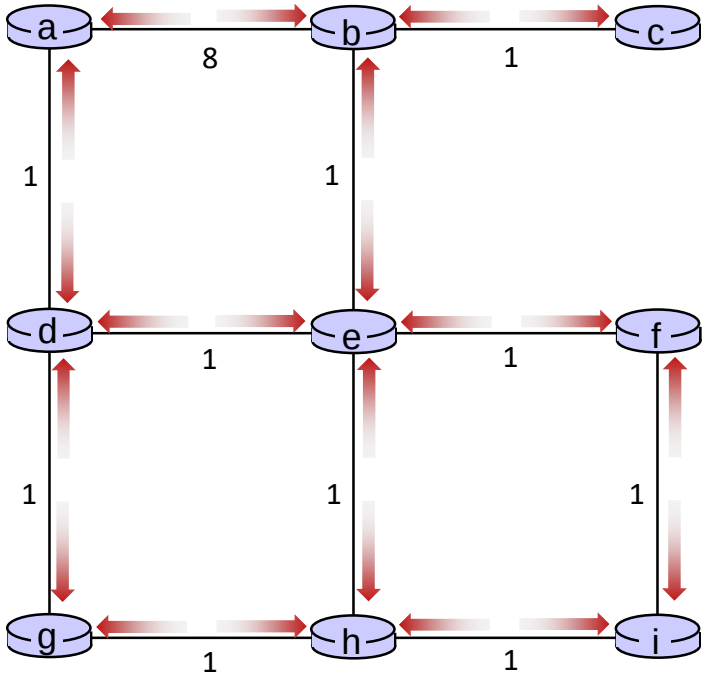


Distance vector: example

t=2

All nodes:

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

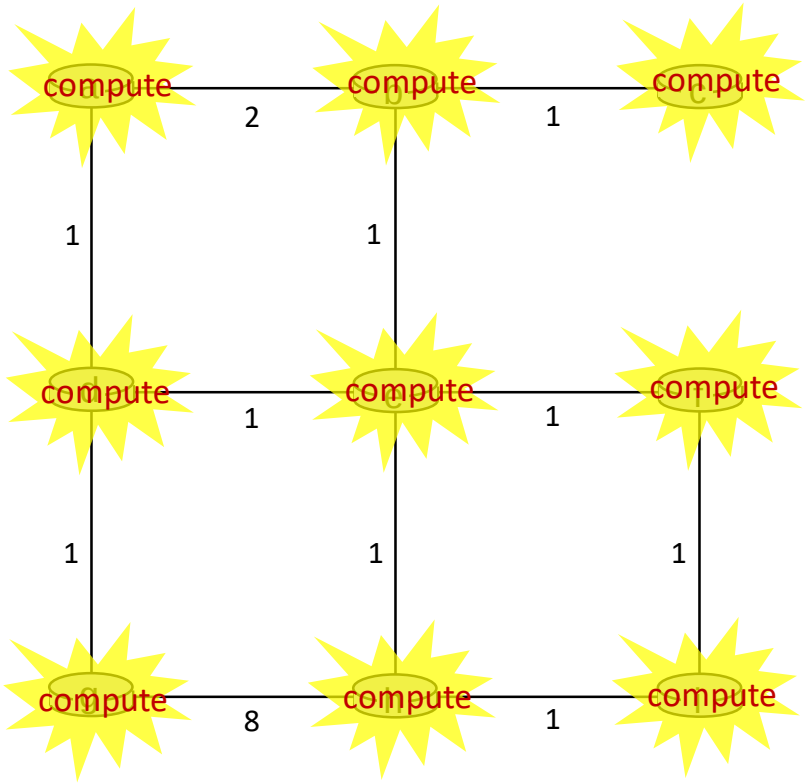


Distance vector: example

t=2

All nodes:

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

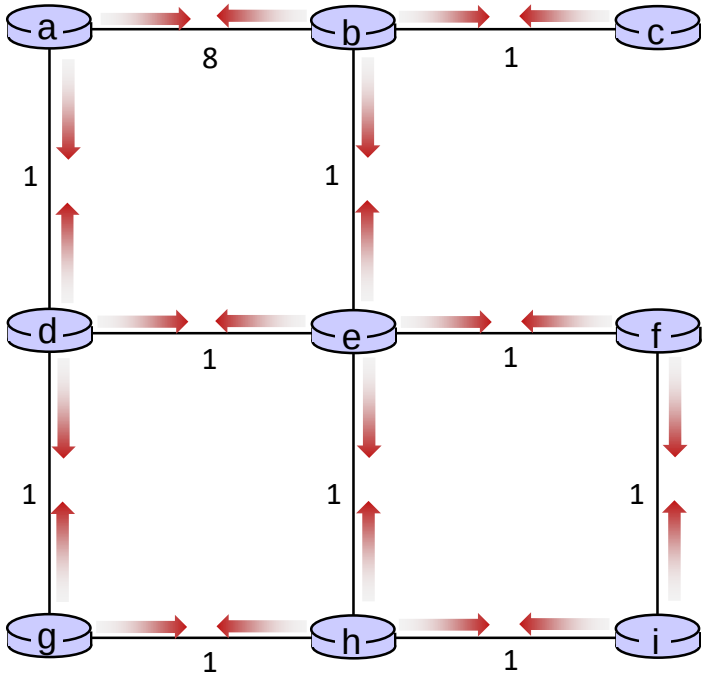


Distance vector: example

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

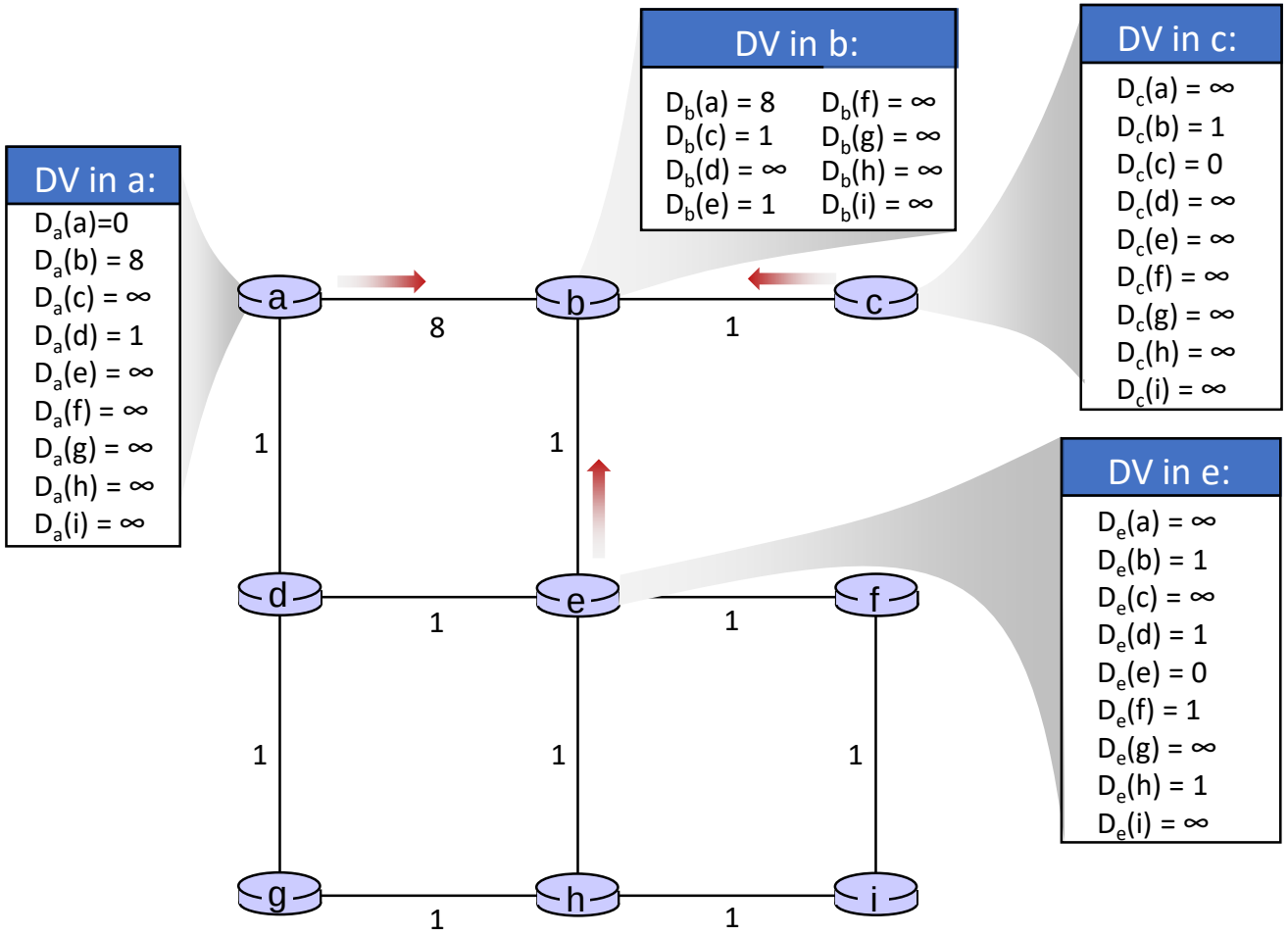
…. and so on

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

Distance vector example: computation

t=1

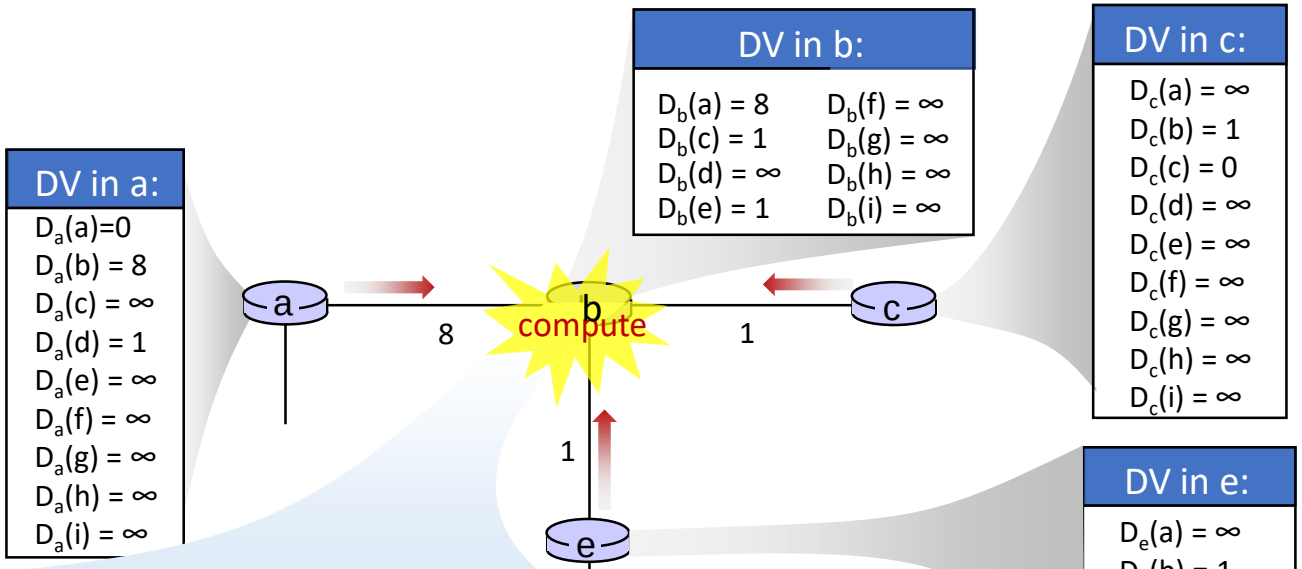
- b receives DVs from a, c, e



Distance vector example: computation

t=1

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



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

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

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

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

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

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

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

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

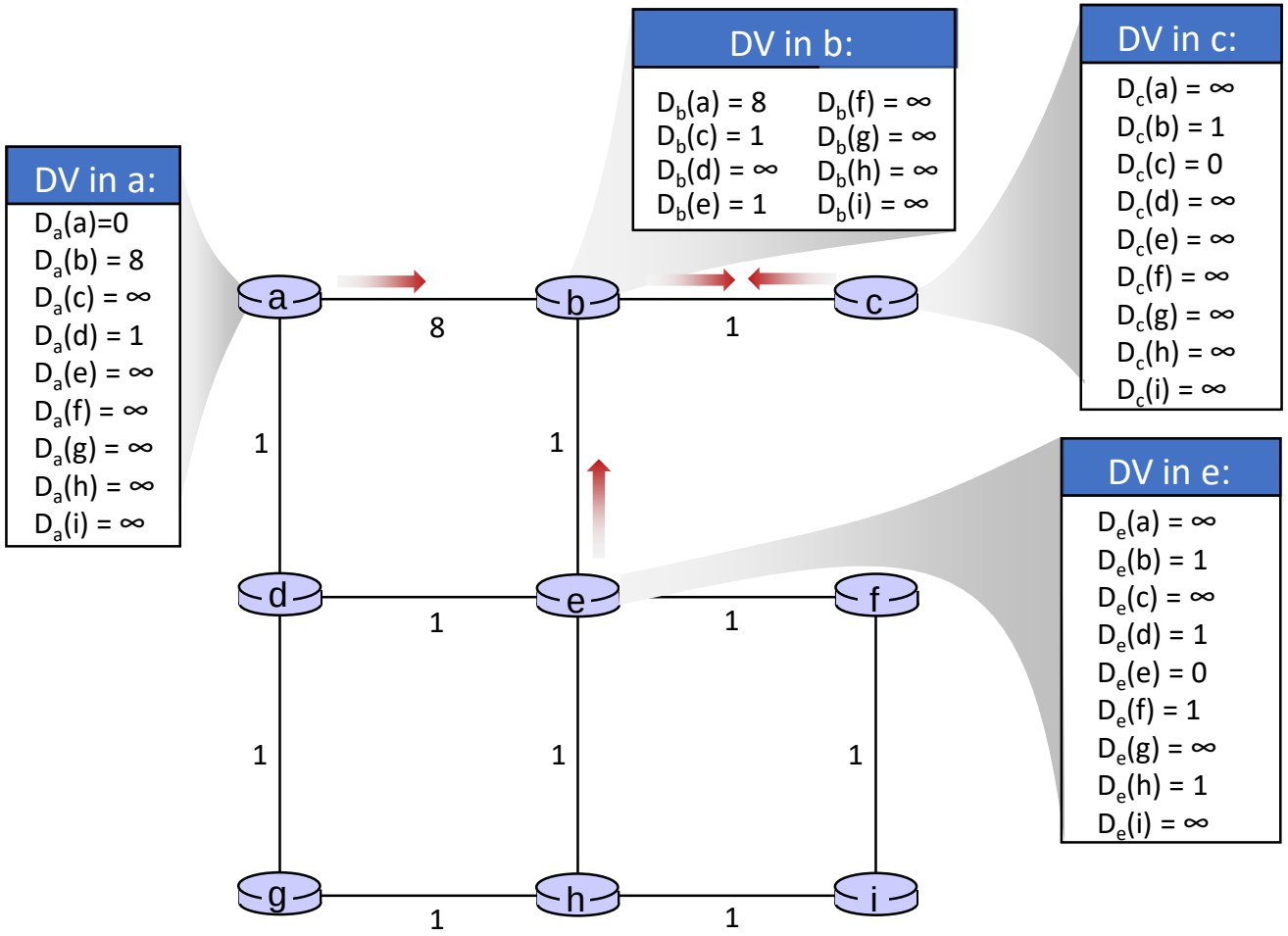
DV in b:

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

Distance vector example: computation

t=1

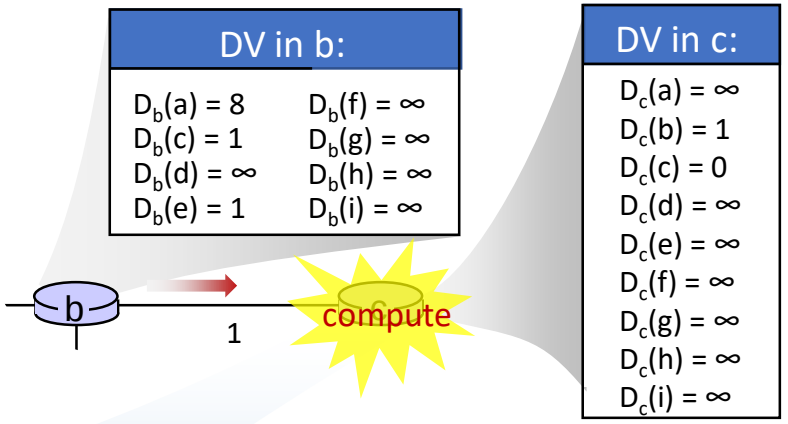
- c receives DVs from b



Distance vector example: computation

t=1

- c receives DVs from b computes:



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

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

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

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

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

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

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

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

DV in c:

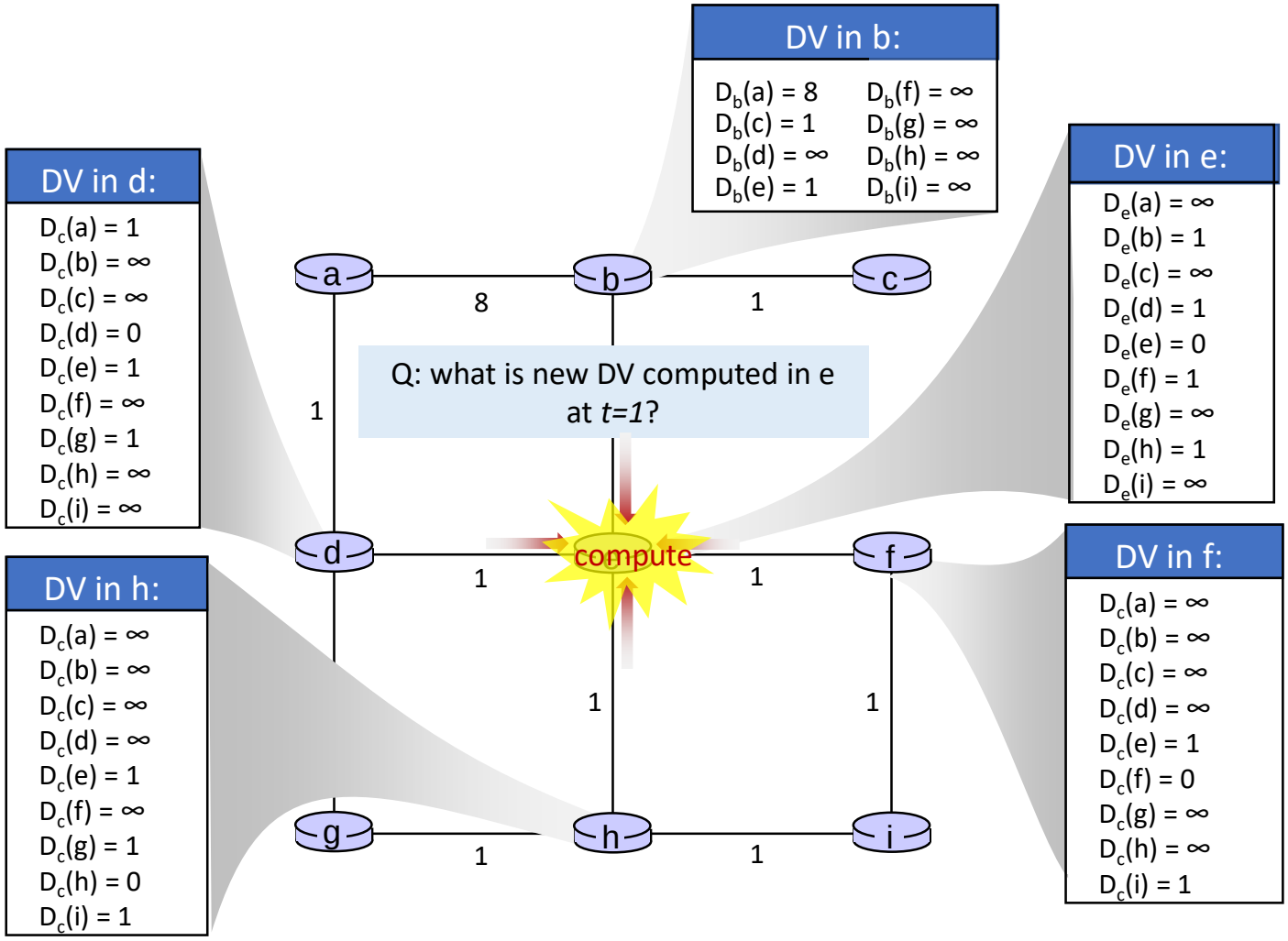
$D_c(a) = 9$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = 2$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Distance vector example: computation

t=1

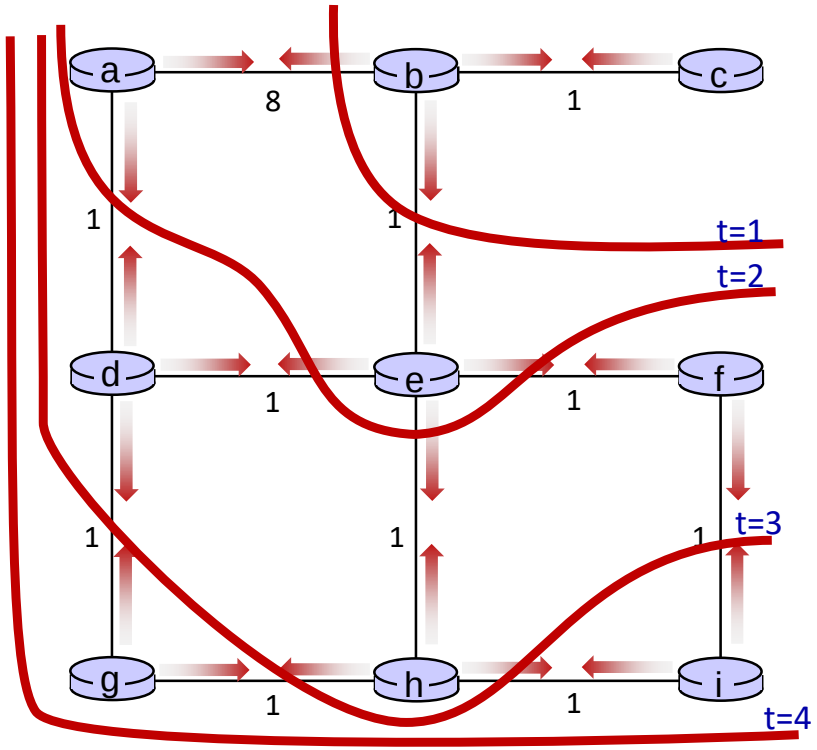
- e receives DVs from b, d, f, h



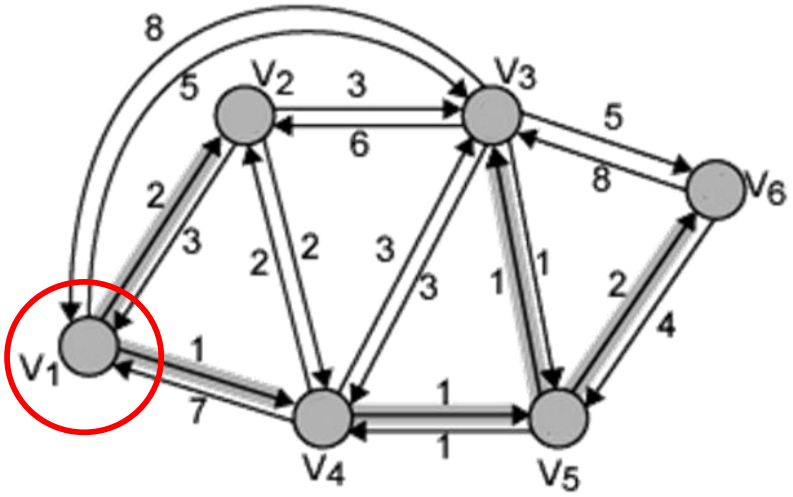
Distance vector: state information diffusion

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 d, f, h
- t=4 c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at g, i



Exercise

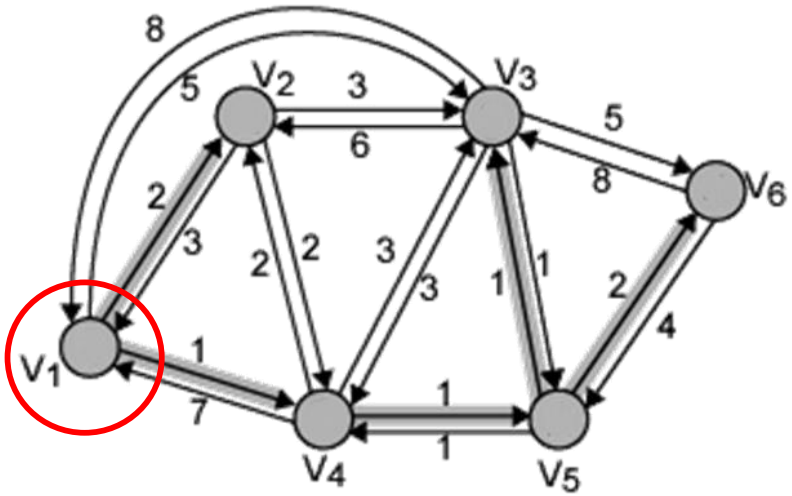


$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

h	$D_h(2)$	Path	$D_h(3)$	Path	$D_h(4)$	Path	$D_h(5)$	Path	$D_h(6)$	Path
0	∞	–	∞	–	∞	–	∞	–	∞	–
1	2	V_1-V_2	5	V_1-V_3	1	V_1-V_4	∞	–	∞	–
2										
3										
4										

Results of Bellman-Ford Algorithm

h	$D_h(2)$	Path	$D_h(3)$	Path	$D_h(4)$	Path	$D_h(5)$	Path	$D_h(6)$	Path
0	∞	–	∞	–	∞	–	∞	–	∞	–
1	2	V_1-V_2	5	V_1-V_3	1	V_1-V_4	∞	–	∞	–
2			4	V_1-V_4-V_3			2	V_1-V_4-V_5	10	V_1-V_3-V_6
3			3	V_1-V_4-V_5-V_3					4	V_1-V_4-V_5-V_6
4	2	V_1-V_2	3	V_1-V_4-V_5-V_3	1	V_1-V_4	2	V_1-V_4-V_5	4	V_1-V_4-V_5-V_6

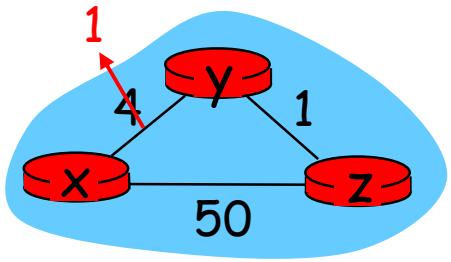


Destination	Next-Hop	Distance
V_2	V_2	2
V_3	V_4	3
V_4	V_4	1
V_5	V_4	2
V_6	V_4	4

Link cost changes

link cost changes:

- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



"good news travels fast"

t_0 : *y* detects link-cost change, updates its DV, informs its neighbors.

t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

Link cost changes

link cost changes:

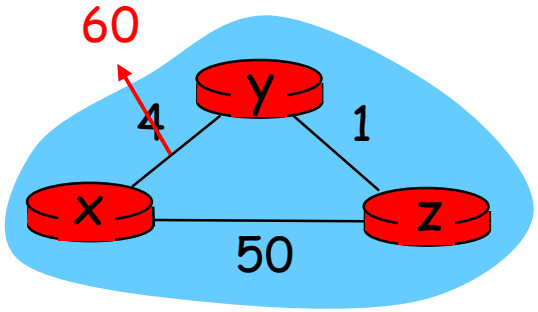
- node detects local link cost change
- *bad news travels slow* –

"count to infinity" problem!

- 44 iterations before algorithm stabilizes: see text

poisoned reverse: (毒性逆转)

- 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?
  - No,for 3 or more nodes circle it still exists.



Before:
Dy(x)=4,
Dy(z)=1, Dz(y)=1, Dz(x)=5

After *:
Dy(x)=min{60, w(y,z)+Dz(x)}=6
Dy(z)=1, Dz(y)=1, Dz(x)=5

After **:
Dy(x)=6;
Dy(z)=1
Dz(y)=1,
Dz(x)=min{50, w(z,y)+Dy(x)}=7

After ***:

Dijkstra vs. Bellman-Ford

- Routing based on Dijkstra
  - Link states flood to all other nodes
  - Each node will have complete topology and build its own routing table
  - Cannot deal with negative weight

- Routing based on Bellman-Ford
  - Each node maintain distance vectors to other known nodes
  - Vectors exchanged with direct neighbours to update the paths and costs
  - Routing tables built in a distributed way

Dijkstra vs. Bellman-Ford

Message complexity

- DK: **n** nodes, **e** links, **O(ne)** messages
- BF: Depends on convergence time

Speed of convergence

- DK: $\mathbf{O(n^2)}$ and quick; May have oscillations
- BF: Slow and depends on changes; May contain routing loops

Robustness: what happens if node malfunctions

- DK: Advertise incorrect direct links cost; Error range constrained
- BF: Error node can exchange incorrect paths cost; Error may propagate through the network

Routing algorithm classification

Q: global or local information?

centralized:

- 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
- "distance vector" algorithms

Q: static or dynamic?

static:

- routes change slowly over time

dynamic:

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

Determine Link Cost

- 3 stages in ARPANET
- First stage in 1969
  - Output queue length is used to define a link cost
  - Bellman-Ford algorithm is used for routing
- Second stage in 1979
  - Measured delay is used to define a link cost
  - Mix queuing, transmission, and propagation
  - **Time of retransmit – Time of arrive + Transmission time + Propagation time**
  - Dijkstra's algorithm is used for routing

Determine Link Cost

- To handle the oscillation problem of Dijkstra
- Let some stay on loaded links to balance the traffic

- Apply Link utilization to represent a link's state
- Leveling based on previous value and new utilization
- Use hop normalized metric to calculate link cost

Calculate Link Cost

- Uses the single-server queuing model
- Link utilization
  - $\rho = 2(Ts - T)/(Ts - 2T)$
  - T – current measured delay
  - Ts – mean packet length (600 bit) / transmission rate of the link
- Leveling
  - $U_n = \alpha \times \rho_n + (1-\alpha) \times U_{n-1}$
  - U_n – leveled link utilization at time n
  - α – constant, now set 0.5

Summary

- 集中式路由
- 分布式路由：洪泛，随机行走，自适应路由
- 最小代价路由算法及其性能分析
  - Dijkstra Algorithm（集中式、全局信息）
  - Bellman-Ford（分布式、局部信息）
- 链路代价的计算