COMP 3331/9331: Computer Networks and Applications

Week 8

Network Layer: 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

these two are not examinable!

Self study

5.6 ICMP: The Internet Control Message Protocol

Network-layer functions

Recall: two network-layer functions:

- *forwarding:* move packets from router's input to appropriate router output

decrement TTL, recompute checksum, find appropriate outgoing link, fragment if it wont fit that link's MTU, send out

data plane

- *routing:* determine route taken by packets from source to destination

control plane

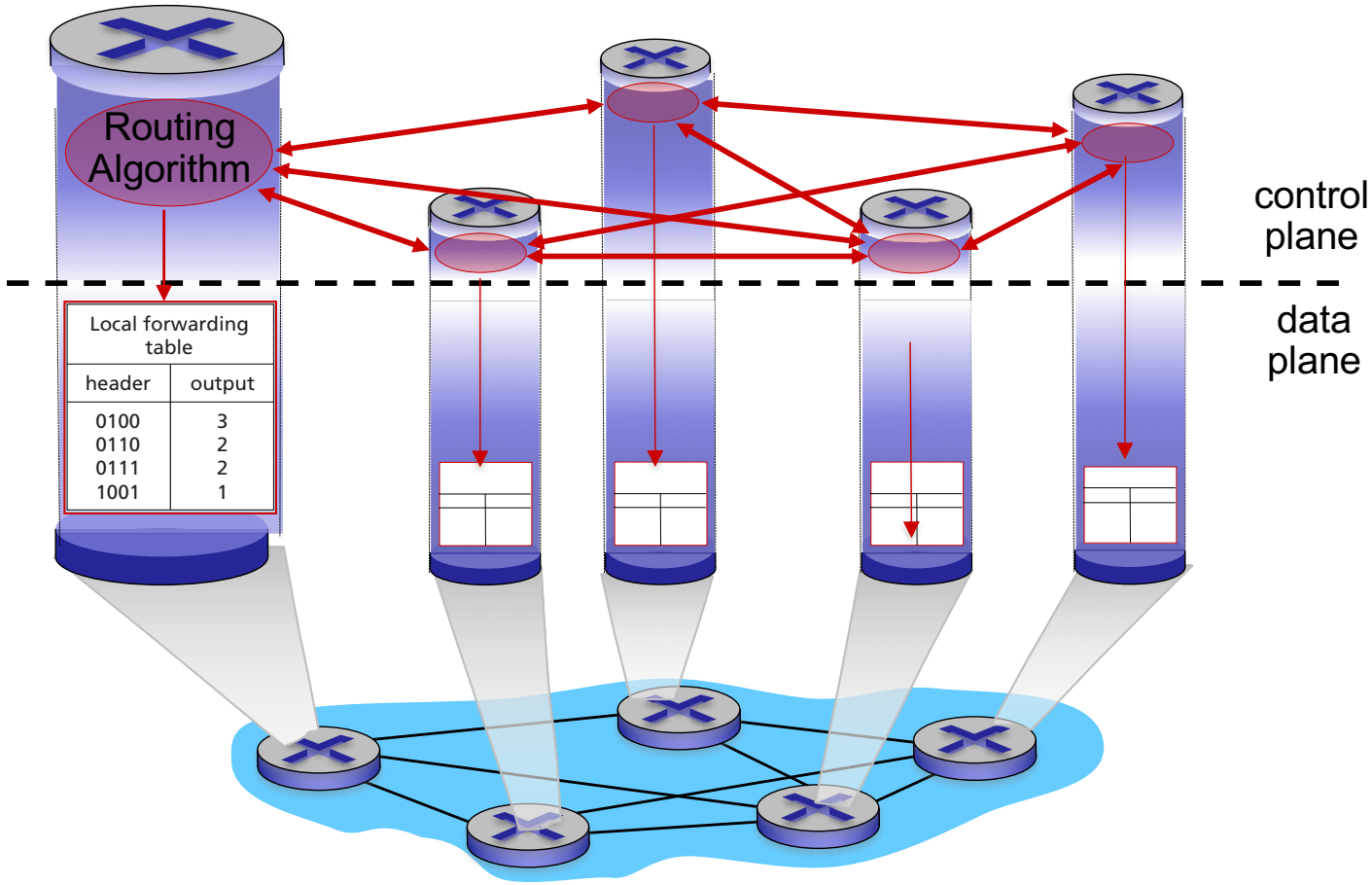
how the forwarding table is built

Two approaches to structuring network control plane:

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

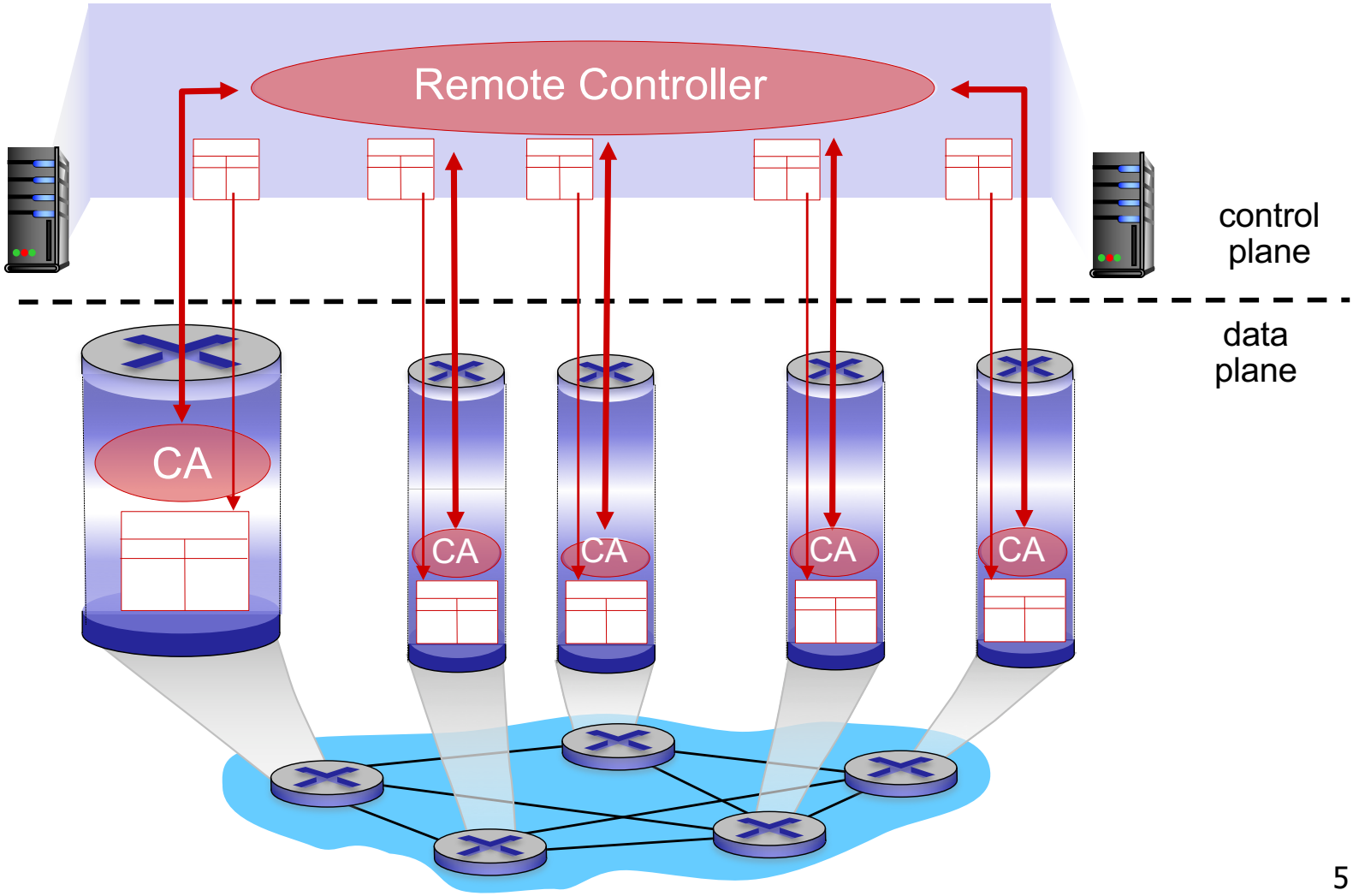
Per-router control plane

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



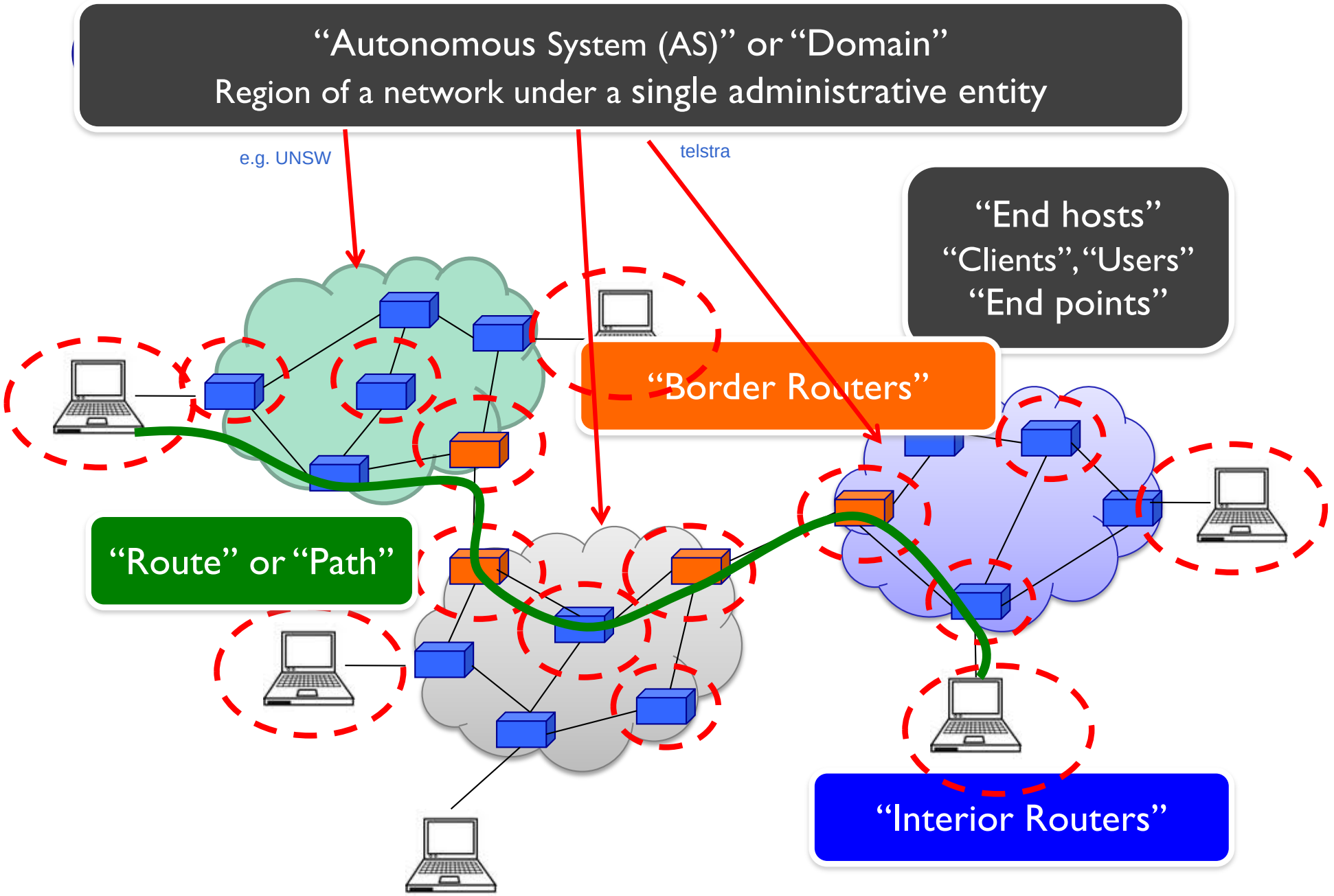
Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector
- Hierarchical routing

5.6 ICMP: The Internet Control Message Protocol


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

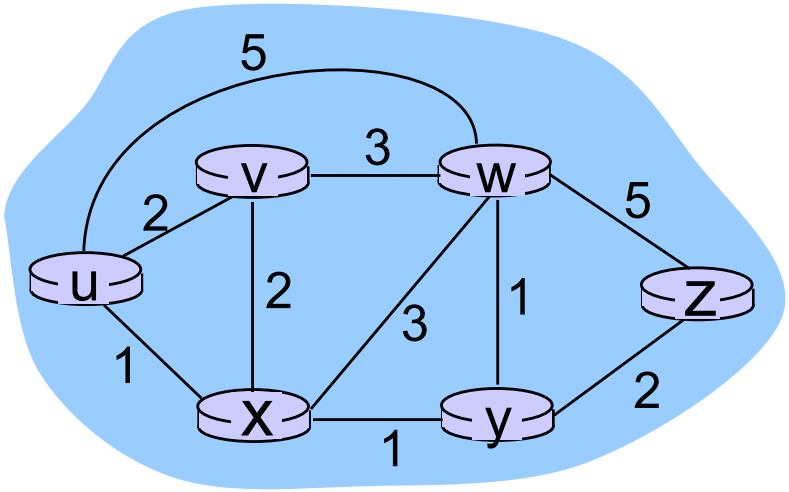
Internet Routing

- Internet Routing works at two levels
- Each AS runs an intra-domain routing protocol that establishes routes within its domain

  i.e. the network admin or administrative entity for that AS chooses which routing protocol they want to use within that AS
  - 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

  between the different AS's
  - Path Vector, e.g., Border Gateway Protocol (BGP)

Graph abstraction



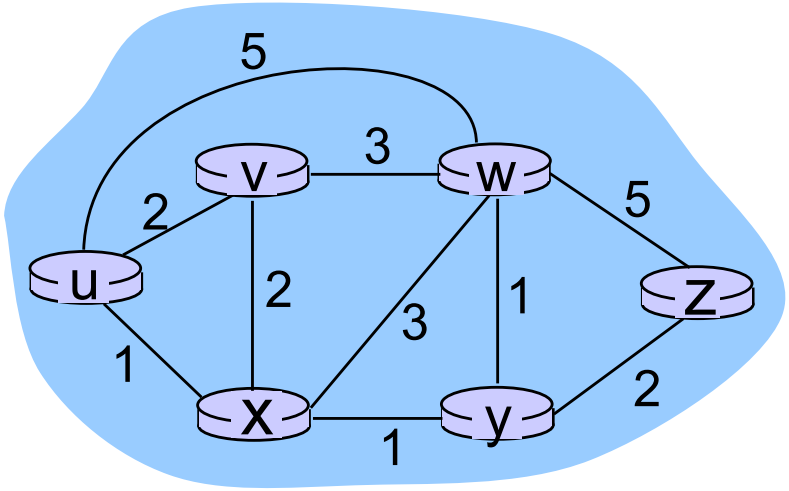
using letter on routers here for human readability and cleanliness. realistically, they should each have a network address with /x notation which advertises which block of IP addresses they are handling

graph: G = (N,E)

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

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

Graph abstraction: costs



c(x,x') = cost of link (x,x')
e.g., c(w,z) = 5

cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

key question: what is the least-cost path between u and z ?
routing algorithm: algorithm that finds that least cost path

Link Cost

- Typically simple: all links are equal

  link bandwidth generally not taken into account for these cost algorithms, KISS

- Least-cost paths => shortest paths (hop count)

  Dijkstra

- Network operators add policy exceptions
  - Lower operational costs
  - Peering agreements
  - Security concerns

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

Routing algorithm classes

Link State (Global)

- Routers maintain cost of each link in the network
- Connectivity/cost changes flooded to all routers
- Converges quickly (less inconsistency, looping, etc.)
- Limited network sizes

per router, each router has a "map" of the entire network (this requires routers to send each other who their immediately connected routers are, which is why this is only good for small networks)

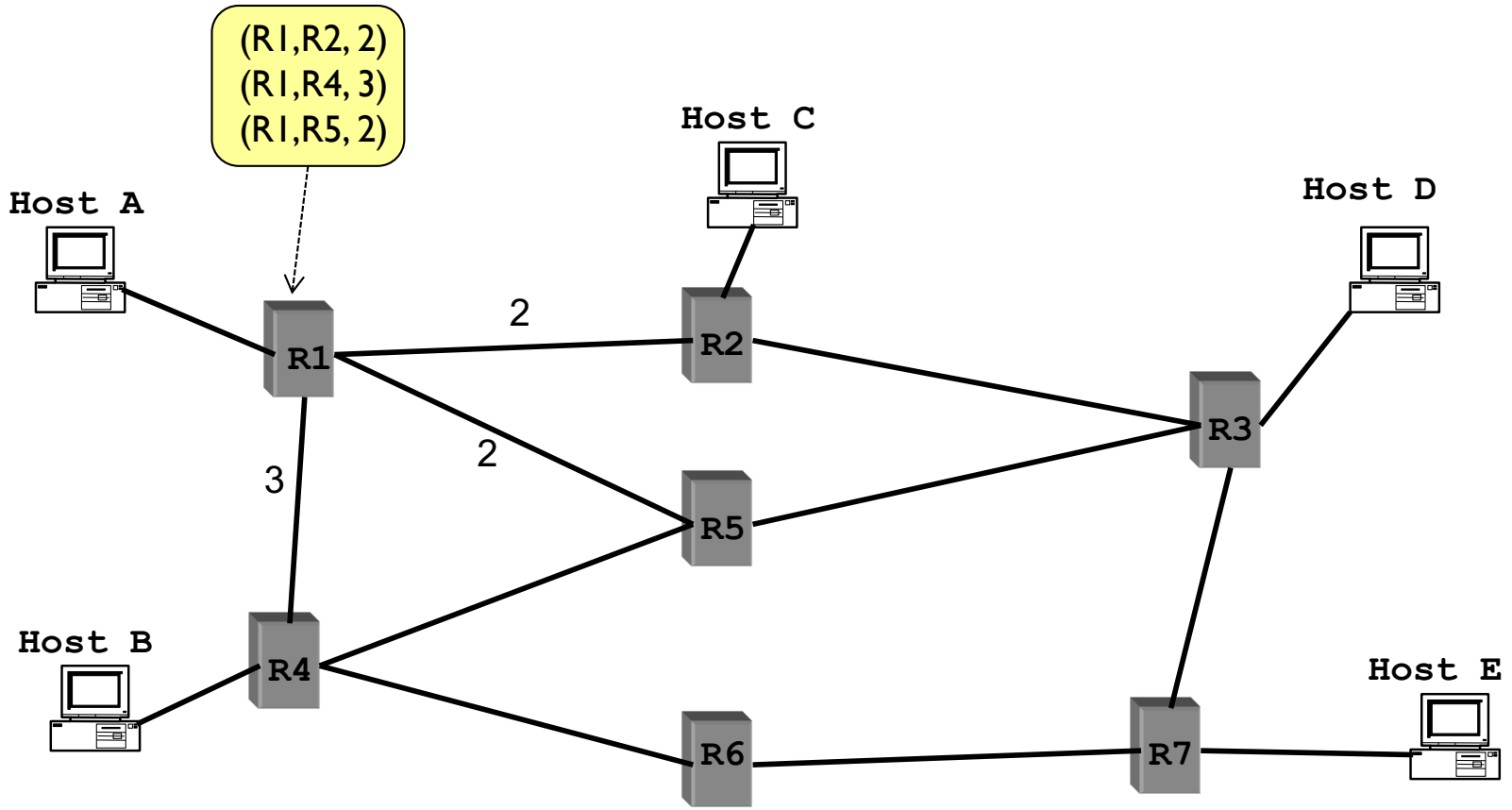
Distance Vector (Decentralised)

- Routers maintain next hop & cost of each destination.
- Connectivity/cost changes iteratively propagate form neighbour to neighbour
- Requires multiple rounds to converge
- Scales to large networks

only information about direct neighbours is stored (cost to that neighbour)

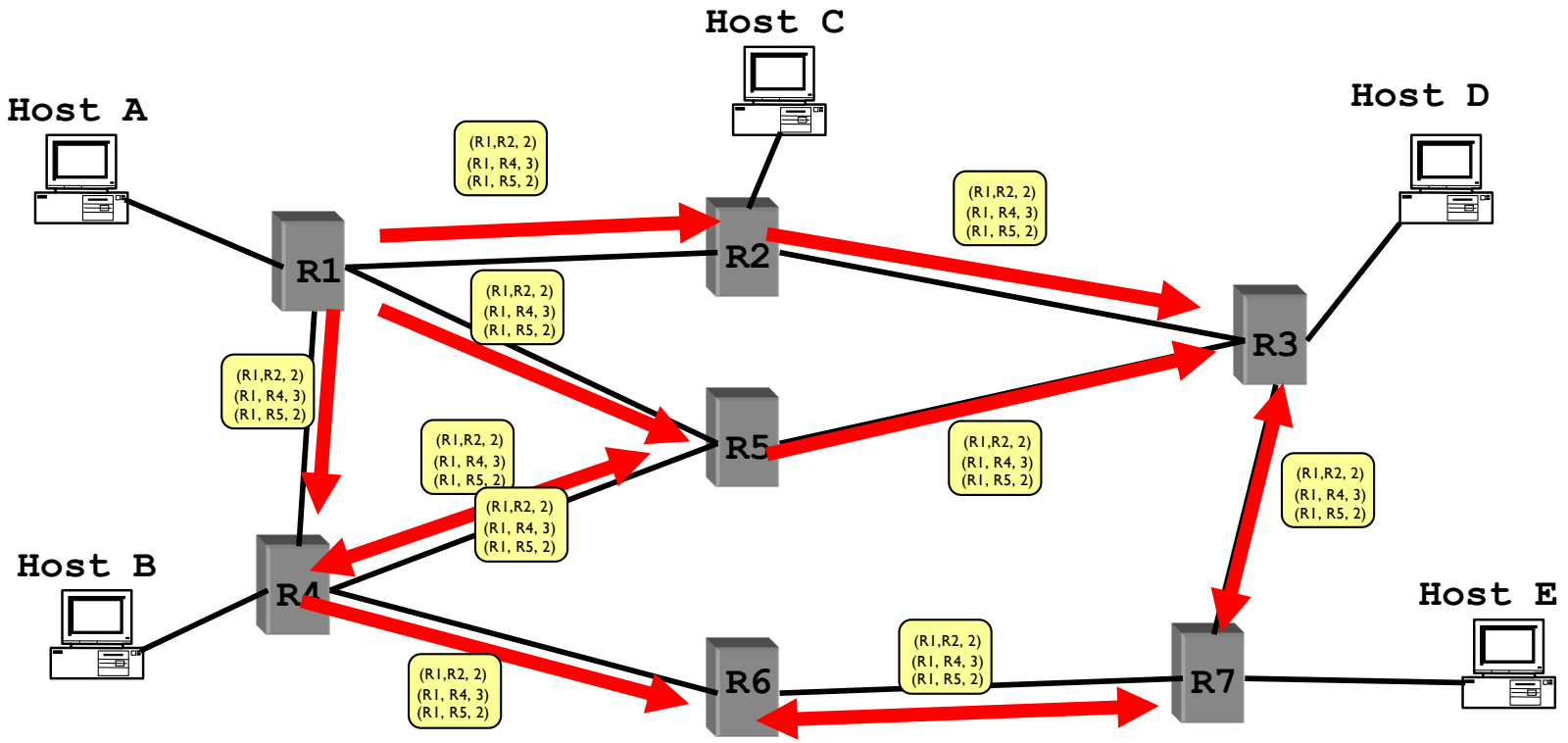
Link State Routing

- 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

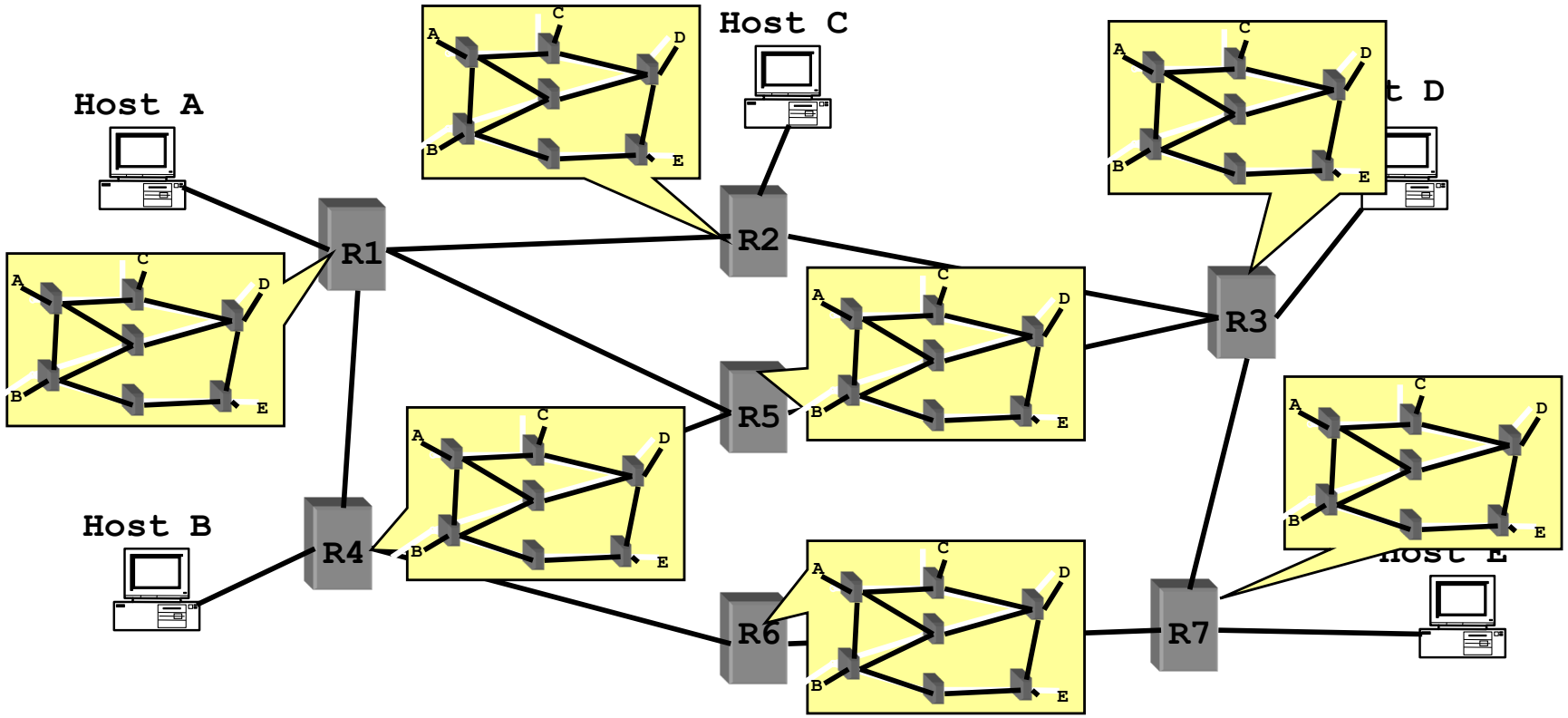


Flooding LSAs

- 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



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 *forwarding table* for that node
- iterative: after k iterations, know least cost path to k dest.'s

notation:

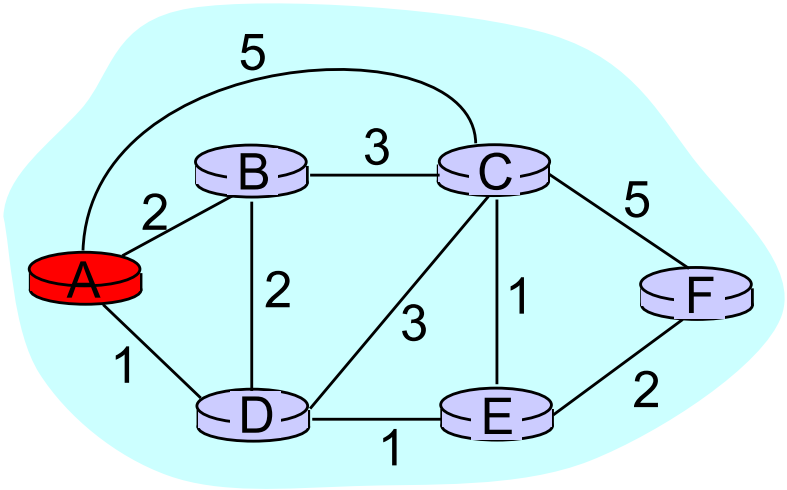
- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

Dijsktra's Algorithm

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v adjacent to u
5          then D(v) = c(u,v)
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 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'
```

Example: Dijkstra's Algorithm

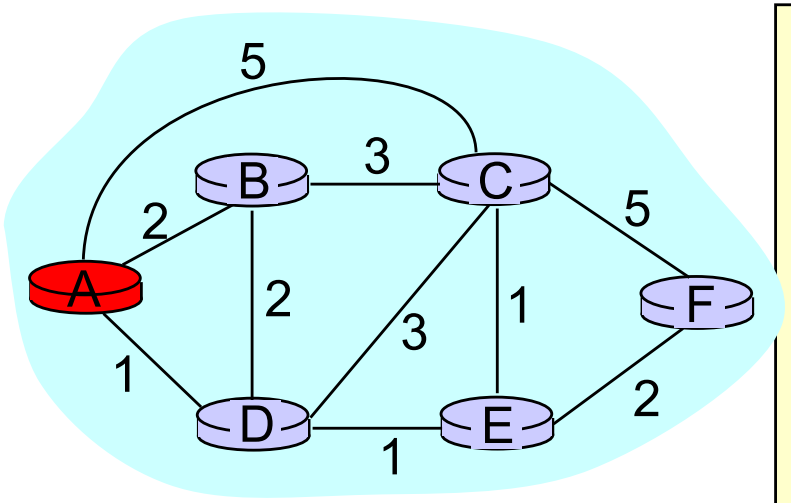
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
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) = ∞;
…
```

Example: Dijkstra's Algorithm

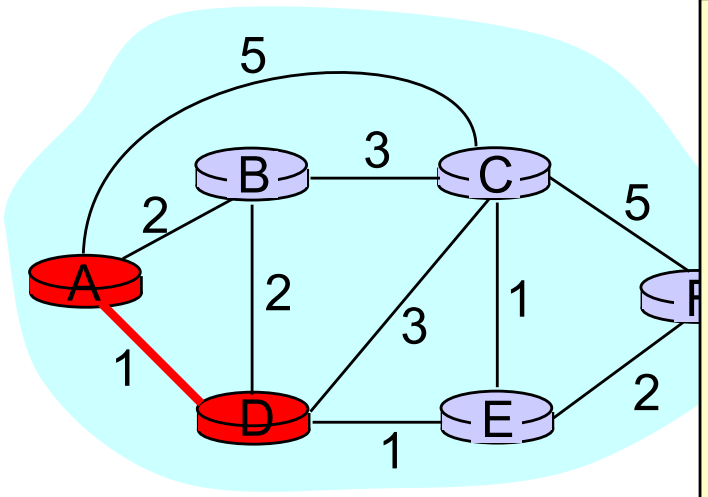
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
...
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

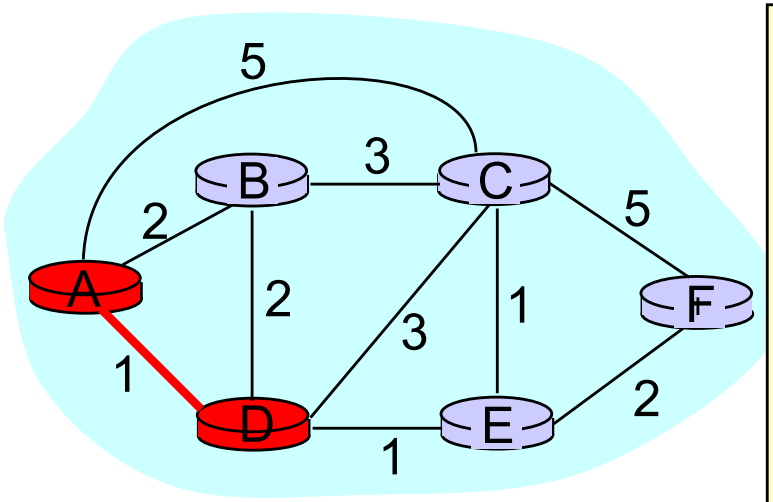
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD					
2						
3						
4						
5						




```
…
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

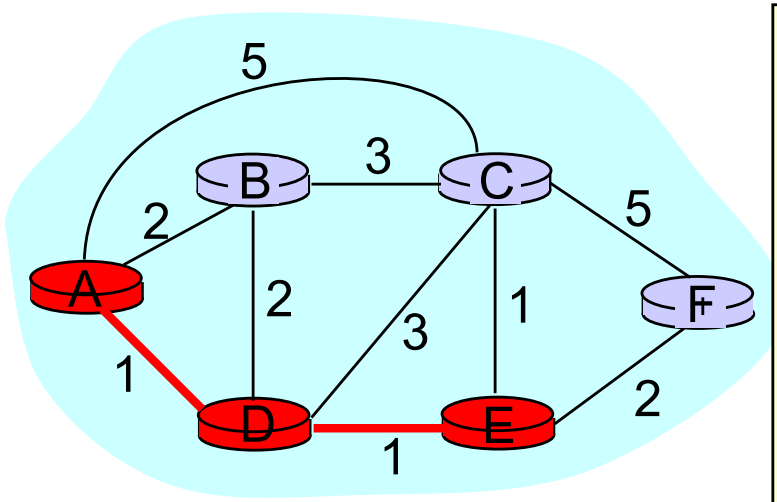
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2, A	4,D		2,D	
2						
3						
4						
5						




```
...
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

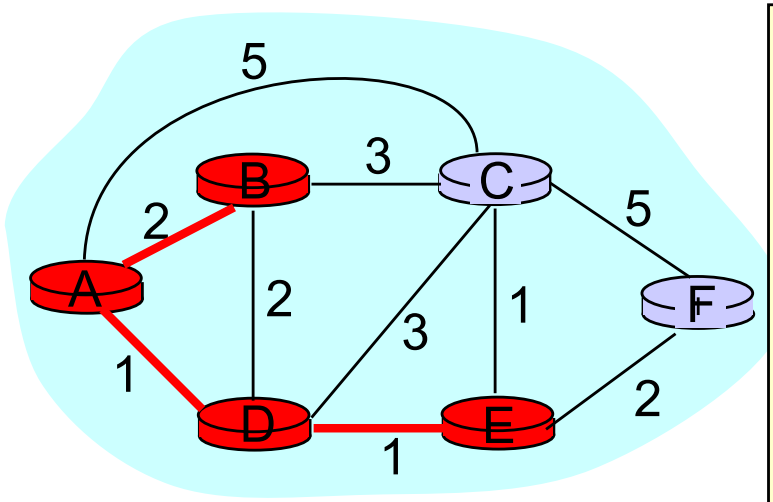
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2, A	4,D		2,D	
2	ADE	2, A	3,E			4,E
3						
4						
5						




```
...
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

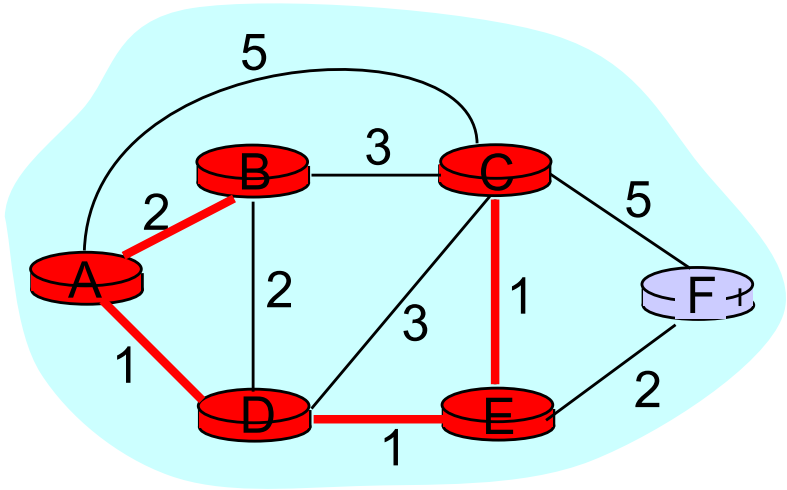
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4						
5						




```
...
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

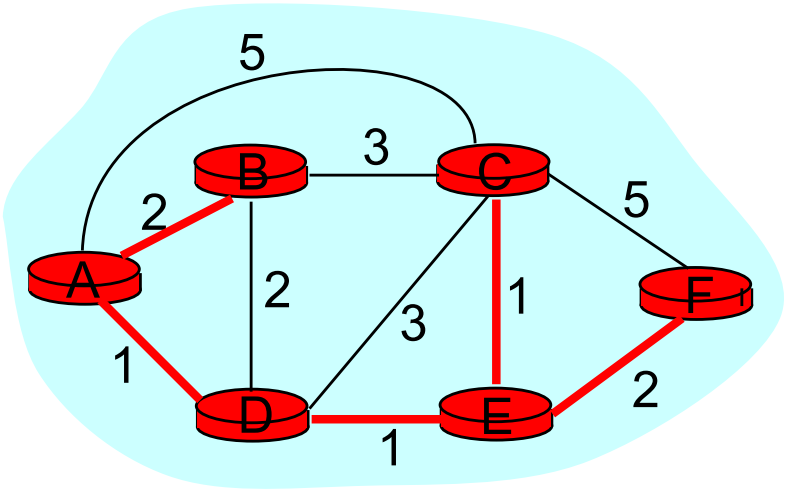
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5						




```
...
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

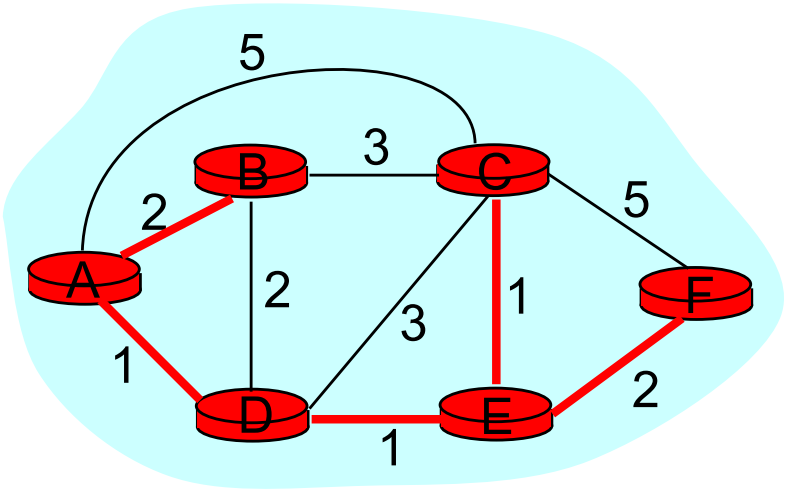
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
→ 5	ADEBCF					




```
...
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		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					

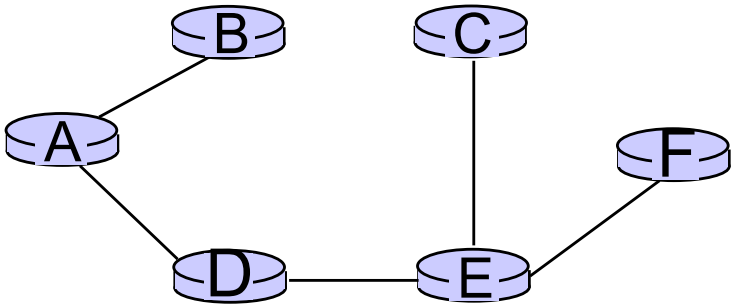


To determine path A → C (say), work backward from C via p(v)

The Forwarding Table

- Running Dijkstra at node A gives the shortest path from A to all destinations
- We then construct the *forwarding table*

resulting shortest-path tree from A:



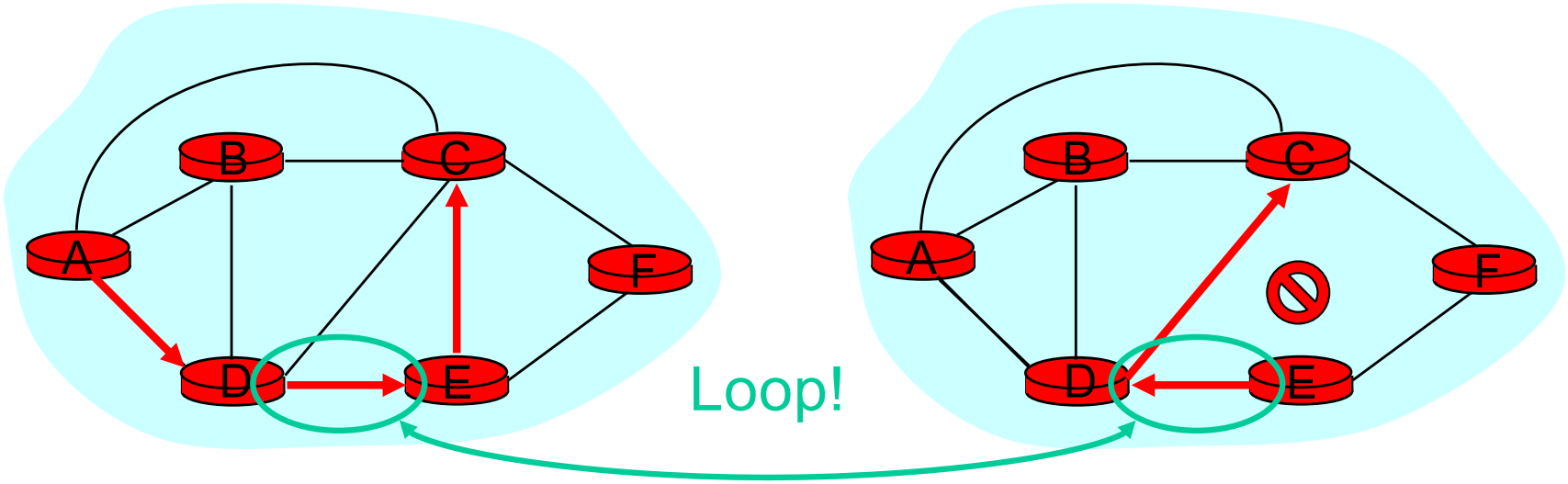
Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

Issue #1: Scalability

- How many messages needed to flood link state messages?
  - O(N x E), where N is #nodes; E is #edges in graph
- Processing complexity for Dijkstra's algorithm?
  - $O(N^2)$, because we check all nodes w not in N' at each iteration and we have O(N) iterations
- How many entries in the LS topology database? O(E)
- How many entries in the forwarding table? O(N)

Issue#2: Transient Disruptions

- Inconsistent link-state database
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - Can cause **transient** forwarding loops



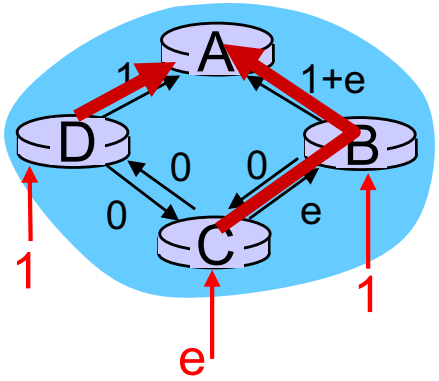
A and D think that this is the path to C

E thinks that this is the path to C

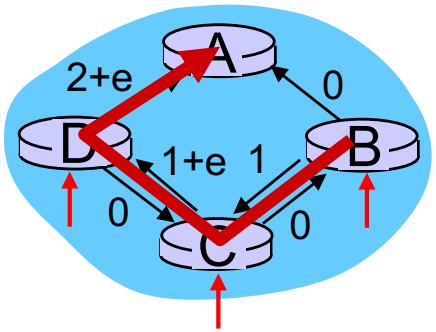
Oscillations

oscillations possible:

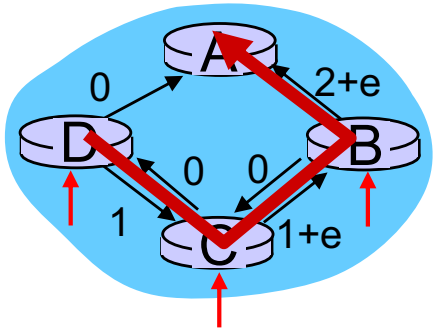
- e.g., suppose link cost equals amount of carried traffic:



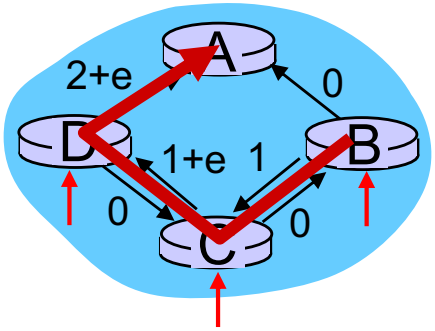
initially



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

Bellman-Ford equation

let

$d_x(y)$:= cost of least-cost path from x to y

then

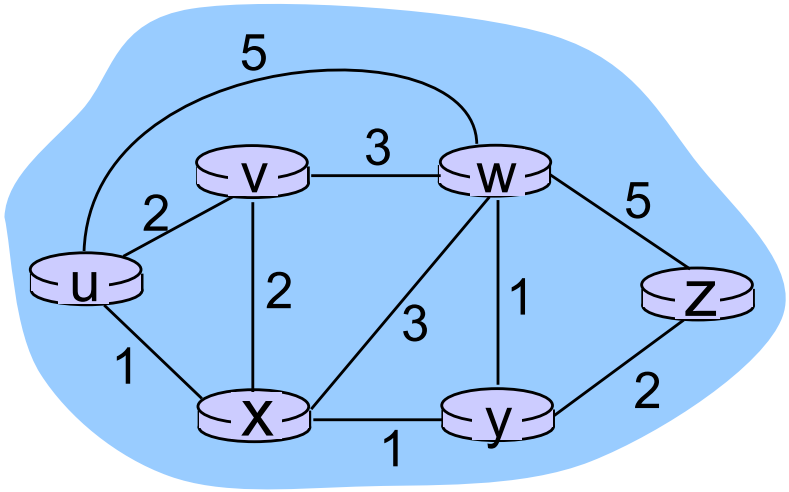
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

$\min_v$: *min* taken over all neighbors v of x

$c(x,v)$: cost to neighbor v

$d_v(y)$: cost from neighbor v to destination y

Bellman-Ford example



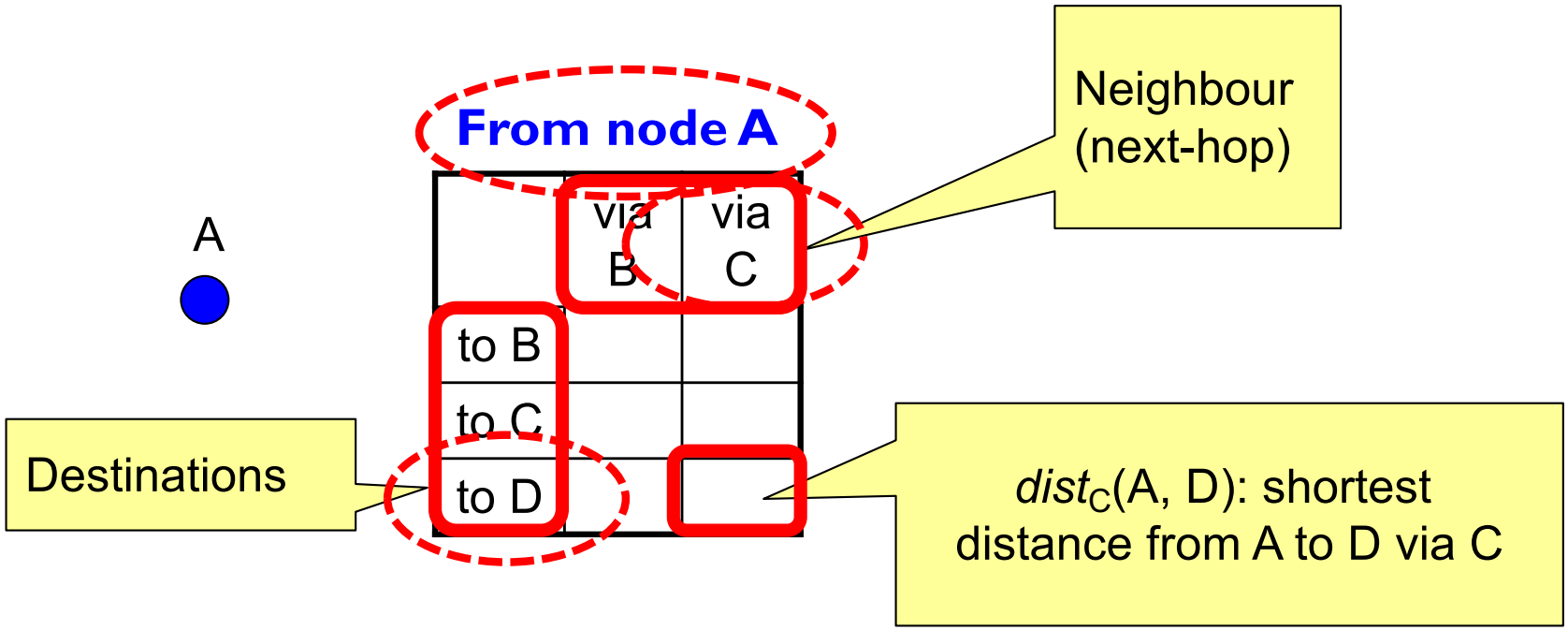
clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), \; c(u,x) + d_x(z), \; c(u,w) + d_w(z) \}$$
$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

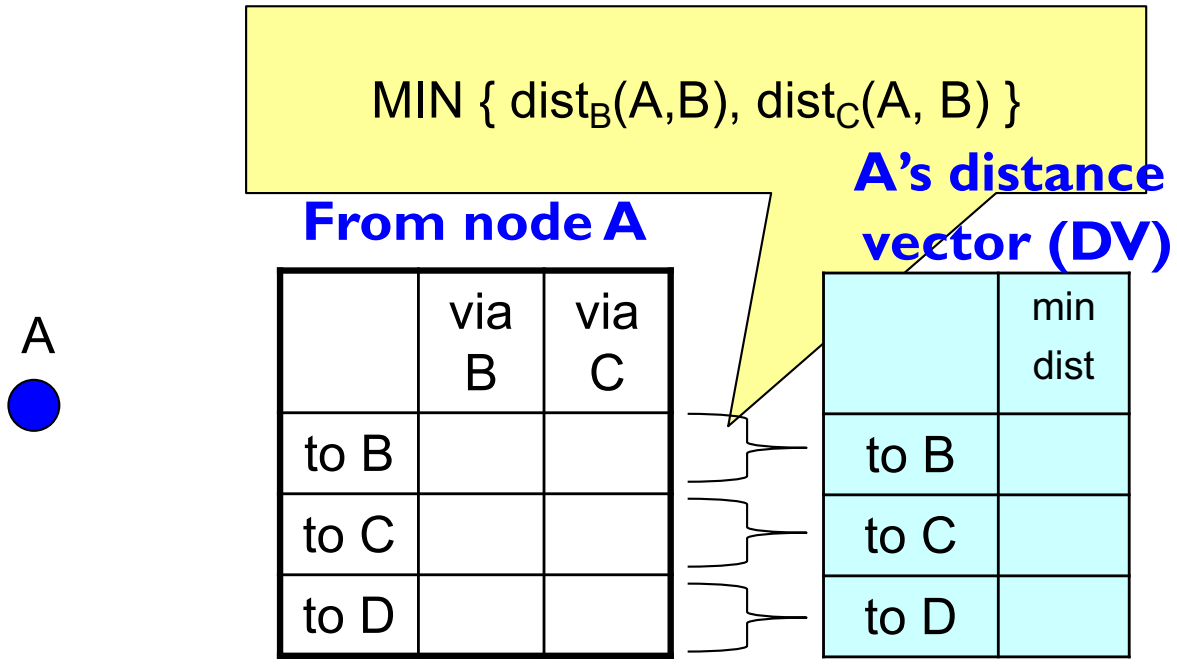
node achieving minimum is next
hop in shortest path, used in forwarding table

How Distance-Vector (DV) works



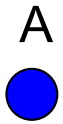
Each router maintains its shortest distance to every destination via each of its neighbours

How Distance-Vector (DV) works



Each router computes its shortest distance to every destination via any of its neighbors

How Distance-Vector (DV) works



From node A

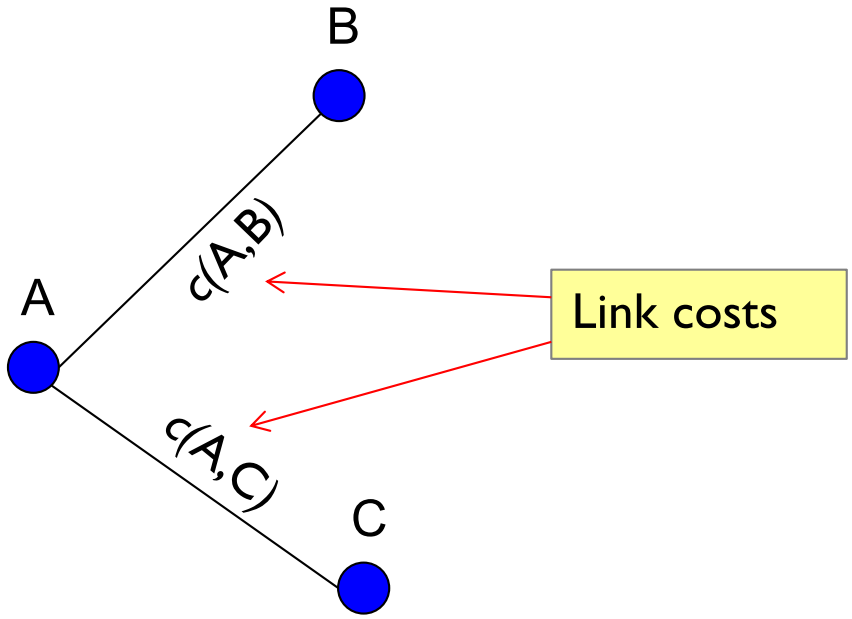
	via B	via C
to B	?	?
to C	?	?
to D	?	?

A's DV

	min dist
to B	?
to C	?
to D	?

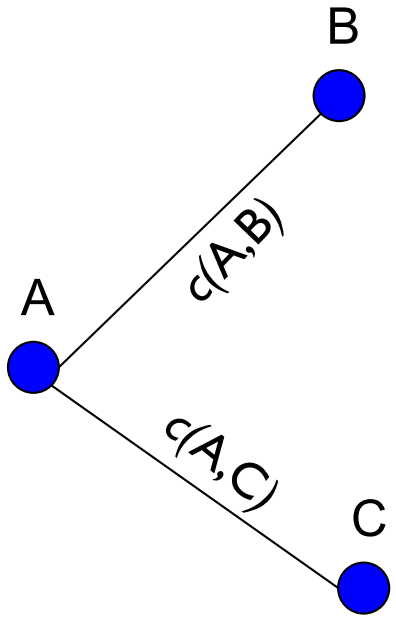
How does A initialize its dist() table and DV?

How Distance-Vector (DV) works



How does A initialize its dist() table and DV?

How Distance-Vector (DV) works



From node A

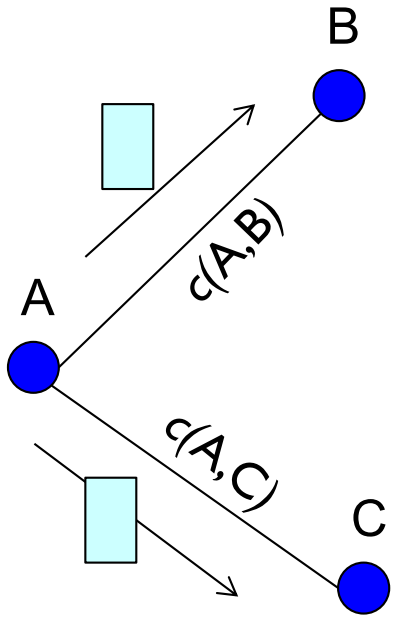
	via B	via C
to B	*c*(A,B)	∞
to C	∞	*c*(A,C)
to D	∞	∞

A's DV

	mindist
to B	c(A,B)
to C	c(A,C)
to D	∞

Each router initializes its *dist*() table based on its immediate neighbors and link costs

How Distance-Vector (DV) works



Assume that A's DV is as follows at some later time

From node A

	via B	via C
to B	*c*(A,B)	∞
to C	∞	*c*(A,C)
to D	∞	∞

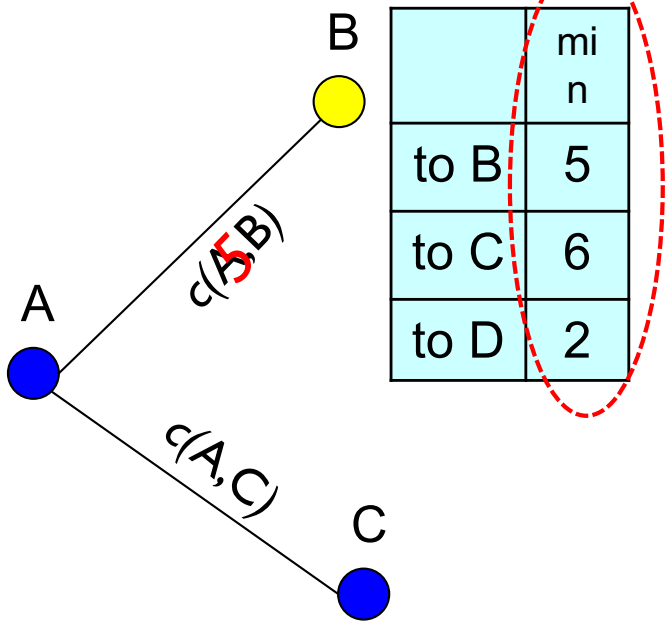
A's DV

	mindist
to B	5
to C	6
to D	2

note: these numbers are just made up example ones

Each router sends its DV to its immediate neighbors

How Distance-Vector (DV) works



A's DV

	min
to B	5
to C	6
to D	2

From node B

	via A	via C
to A	5	∞
to C	15	1
to D	∞	∞

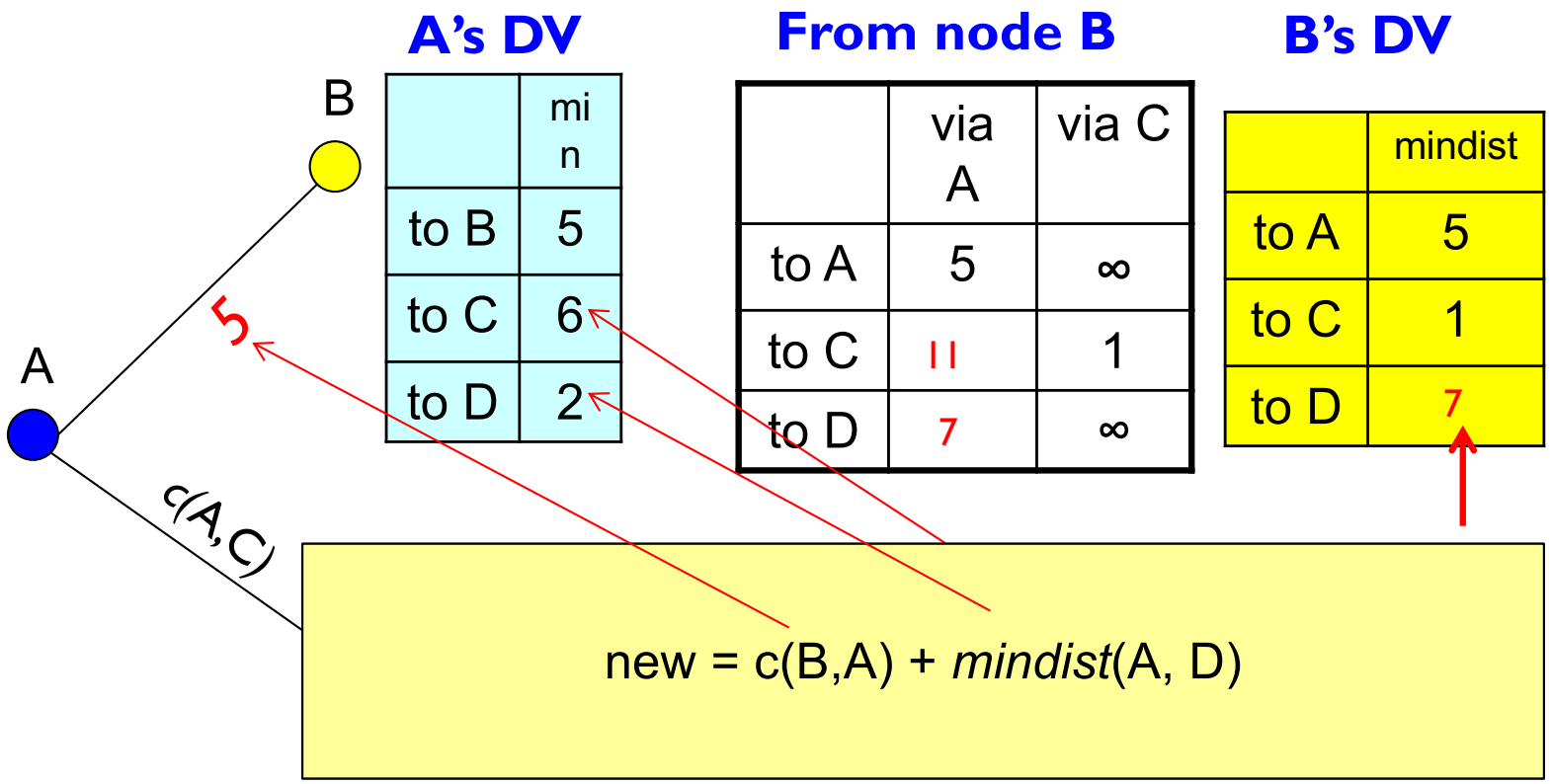
only have to look at this column because B has received DV from A, meaning that only paths via A have a chance to update

B's DV

	mindist
to A	5
to C	1
to D	∞

Routers process received DVs

How Distance-Vector (DV) works



Routers process received DVs

And repeat…

Distance Vector Routing

- Each router knows the links to its neighbors
- Each router has provisional “shortest path” to every other router -- its distance vector (DV)
- Routers exchange this DV with their neighbors
- Routers look over the set of options offered by their neighbors and select the best one
- Iterative process converges to set of shortest paths

Distance vector routing

iterative, asynchronous:

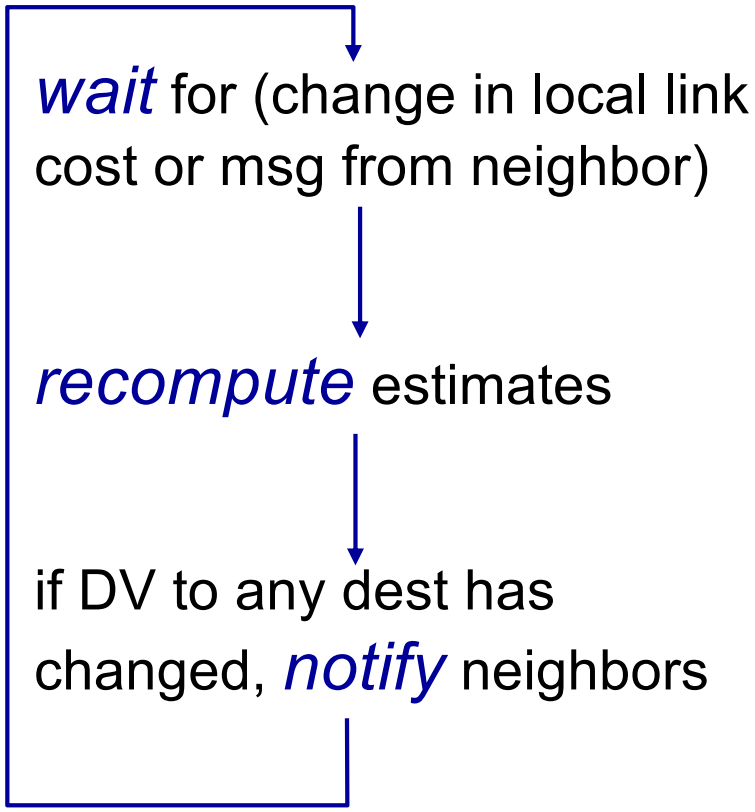
each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed:

- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

each node:



Distance Vector

- c(i,j): link cost from node *i* to *j*
- $dist_Z(A,V)$: shortest dist. from A to V via Z
- mindist(A,V): shortest dist. from A to V

```
0 At node A
1 Initialization:
2    for all destinations V  do
3         if V is neighbor of A
4              distV(A, V) = mindist(A,V) = c(A,V);
5         else
6              distV(A, V) = mindist(A,V) = ∞;
7      send mindist(A, *) to all neighbors
  loop:
8    wait (until A sees a link cost change to neighbor V  /* case 1 */
9           or until A receives mindist(V,*) from neighbor V)    /* case 2 */
10    if (c(A,V) changes by ±d)  /* ⇐ case 1 */
11       for all destinations Y do
12                distV(A,Y) =  distV(A,Y) ± d
13    else /* ⇐ case 2: */
14       for all destinations Y do
15                distV(A,Y) = c(A,V) + mindist(V, Y);
16    update mindist(A,*)
15   if (there is a change in mindist(A, *))
16         send mindist(A, *) to all neighbors
17  forever
```

Distance Vector

- c(i,j): link cost from node *i* to *j*
- $dist_Z(A,V)$: shortest dist. from A to V via Z
- mindist(A,V): shortest dist. from A to V

```
0 At node A
1 Initialization:
2    for all destinations V  do
3         if V is neighbor of A
4              distV(A, V) = mindist(A,V) = c(A,V);
5         else
6              distV(A, V) = mindist(A,V) = ∞;
7      send mindist(A, *) to all neighbors
   loop:
8    wait (until A sees a link cost change to neighbor V  /* case 1 */
9           or until A receives mindist(V,*) from neighbor V)    /* case 2 */
10    if (c(A,V) changes by ±d)  /* ⇐ case 1 */
11       for all destinations Y do
12                distV(A,Y) =  distV(A,Y) ± d
13    else /* ⇐ case 2: */
14       for all destinations Y do
15                distV(A,Y) = c(A,V) + mindist(V, Y);
16    update mindist(A,*)
15   if (there is a change in mindist(A, *))
16         send mindist(A, *) to all neighbors
17  forever
```

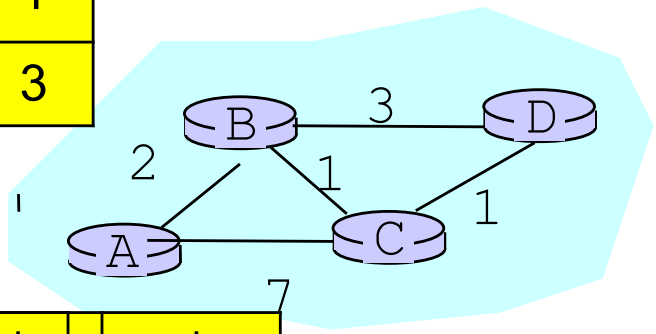
Example: Initialization

from Node B

	via A	via C	via D	min dist
to A	2	∞	∞	2
to B	-	-	-	0
to C	∞	1	∞	1
to D	∞	∞	3	3

from Node D

	via B	via C	min dist
to A	∞	∞	∞
to B	3	∞	3
to C	∞	1	1
to D	-	-	0



from Node A

	via B	via C	min dist	min dist
to A	-	-	0	0
to B	2	∞	2	2
to C	∞	7	7	7
to D	∞	∞	∞	∞

from Node C

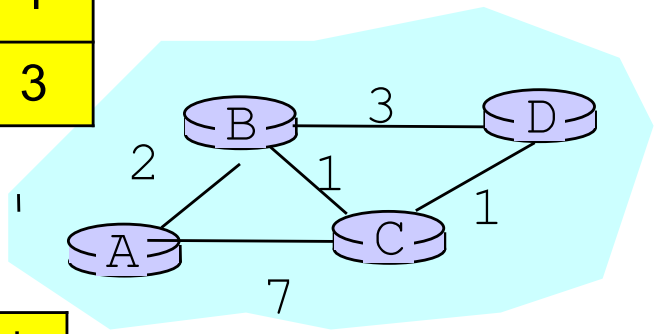
	via A	via B	via D	min dist
to A	7	∞	∞	7
to B	∞	1	∞	1
to C	-	-	-	0
to D	∞	∞	1	1

A likely doesn even have a D row entry to begin with because it only knows about its immediate neighbours, put it in here at the start for simplicity though

Example: C sends update to A

from Node B

	via A	via C	via D	min dist
to A	2	∞	∞	2
to B	-	-	-	0
to C	∞	1	∞	1
to D	∞	∞	3	3



from Node D

	via B	via C	min dist
to A	∞	∞	∞
to B	3	∞	3
to C	∞	1	1
to D	-	-	0

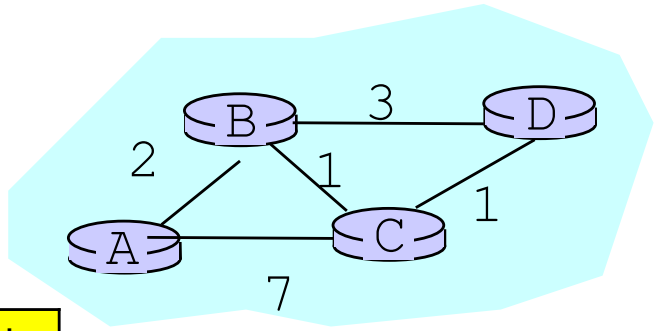
from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	∞	2
to C	∞	7	7
to D	∞	∞	∞

from Node C

	via A	via B	via D	min dist
to A	7	∞	∞	7
to B	∞	1	∞	1
to C	-	-	-	0
to D	∞	∞	1	1

Example: C sends update to A

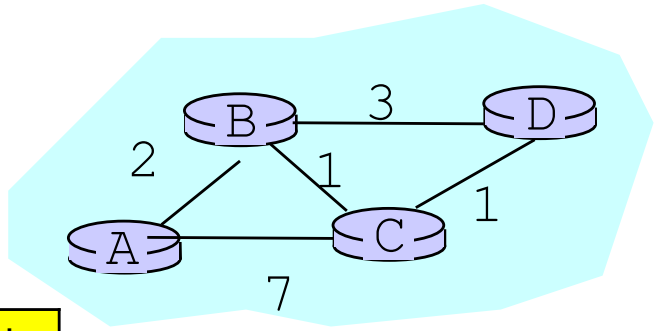


from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	∞	2
to C	∞	7	7
to D	∞	∞	∞

min dist
7
1
0
1

Example: C sends update to A

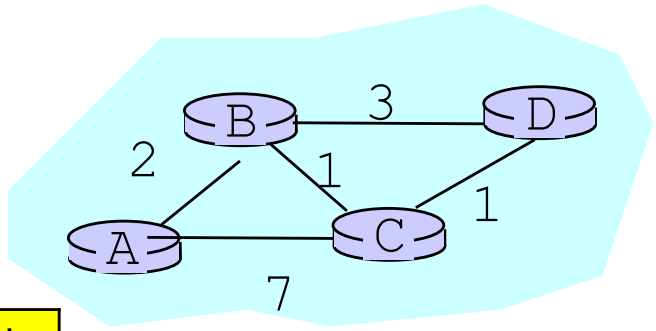


from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	∞	7	7
to D	∞	8	∞

min dist
7
1
0
1

Example: C sends update to A



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	**8**	2
to C	∞	7	7
to D	∞	**8**	**8**

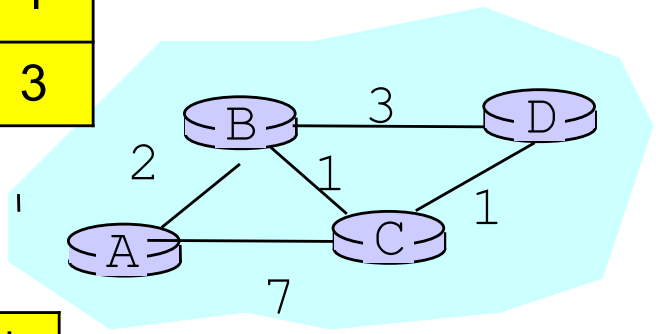
Example: now B sends update to A

from Node B

	via A	via C	via D	min dist
to A	2	∞	∞	2
to B	-	-	-	0
to C	∞	1	∞	1
to D	∞	∞	3	3

from Node D

	via B	via C	min dist
to A	∞	∞	∞
to B	3	∞	3
to C	∞	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	∞	7	7
to D	∞	8	8

from Node C

	via A	via B	via D	min dist
to A	7	∞	∞	7
to B	∞	1	∞	1
to C	-	-	-	0
to D	∞	∞	1	1

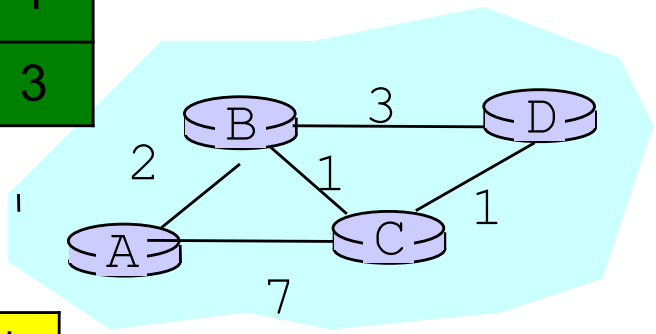
Example: now B sends update to A

from Node B

	via A	via C	via D	min dist
to A	2	∞	∞	2
to B	-	-	-	0
to C	∞	1	∞	1
to D	∞	∞	3	3

from Node D

	via B	via C	min dist
to A	∞	∞	∞
to B	3	∞	3
to C	∞	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	∞	7	7
to D	∞	8	8

from Node C

	via A	via B	via D	min dist
to A	7	∞	∞	7
to B	∞	1	∞	1
to C	-	-	-	0
to D	∞	∞	1	1

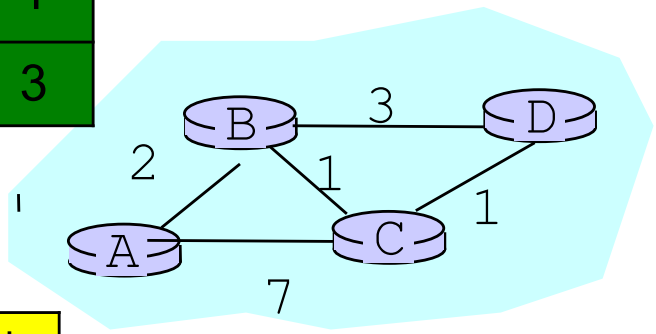
Example: now B sends update to A

from Node B

	via A	via C	via D	min dist
to A	2	∞	∞	2
to B	-	-	-	0
to C	∞	1	∞	1
to D	∞	∞	3	3

from Node D

	via B	via C	min dist
to A	∞	∞	∞
to B	3	∞	3
to C	∞	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	7
to D	5	8	8

from Node C

	via	via	via	min dist
to A	∞	∞	∞	7
to B	∞	∞	∞	1
to C	-	-	-	0
to D	∞	∞	1	1

Make sure you know why this is 5, not 4!

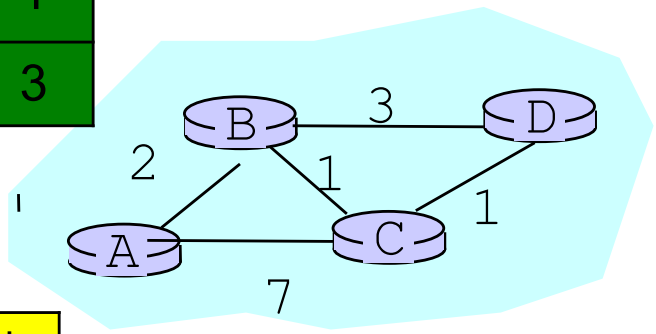
Example: now B sends update to A

from Node B

	via A	via C	via D	min dist
to A	2	∞	∞	2
to B	-	-	-	0
to C	∞	1	∞	1
to D	∞	∞	3	3

from Node D

	via B	via C	min dist
to A	∞	∞	∞
to B	3	∞	3
to C	∞	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	**3**	7	**3**
to D	**5**	8	**5**

from Node C

	via A	via B	via D	min dist
to A	7	∞	∞	7
to B	∞	1	∞	1
to C	-	-	-	0
to D	∞	∞	1	1

All nodes know the best two-hop paths.

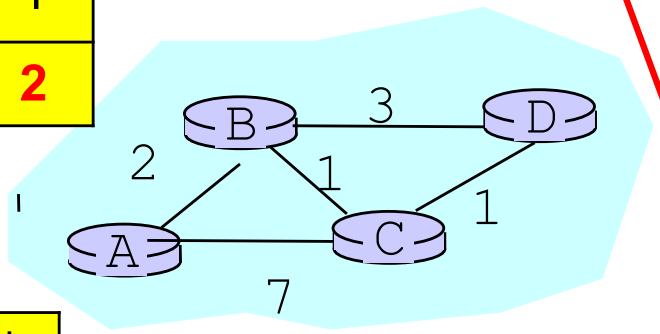
Make sure you believe this

from Node B

	via A	via C	via D	min dist
to A	2	8	∞	2
to B	-	-	-	0
to C	9	1	4	1
to D	∞	2	3	**2**

from Node D

	via B	via C	min dist
to A	5	8	**5**
to B	3	2	**2**
to C	4	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	**3**
to D	5	8	**5**

from Node C

	via A	via B	via D	min dist
to A	7	3	∞	**3**
to B	9	1	4	1
to C	-	-	-	0
to D	∞	4	1	1

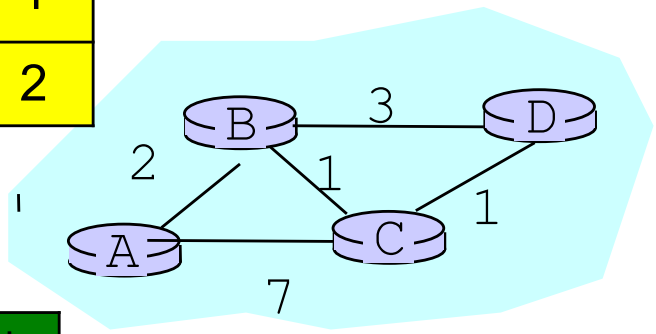
Example: Now A sends update to B

from Node B

	via A	via C	via D	min dist
to A	2	8	∞	2
to B	-	-	-	0
to C	9	1	4	1
to D	∞	2	3	2

from Node D

	via B	via C	min dist
to A	5	8	5
to B	3	2	2
to C	4	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	3
to D	5	8	5

from Node C

	via A	via B	via D	min dist
to A	7	3	∞	3
to B	9	1	4	1
to C	-	-	-	0
to D	∞	4	1	1

Example: Now

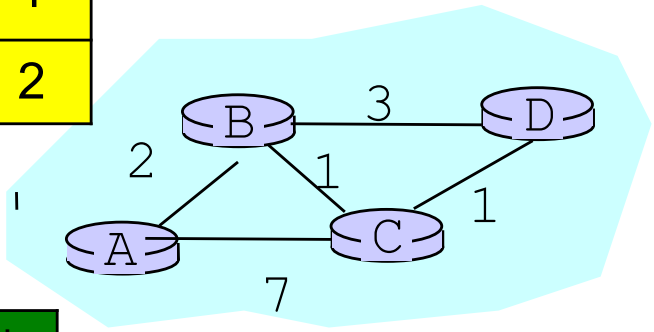
Updated

from Node B

	via A	via C	via D	min dist
to A	2	8	∞	
to B	-	-		0
to C	5	1	4	1
to D	7	2	3	2

from Node D

	via B	via C	min dist
to A	5	8	5
to B	3	2	2
to C	4	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	3
to D	5	8	5

from Node C

	via A	via B	via D	min dist
to A	7	3	∞	3
to B	9	1	4	1
to C	-	-	-	0
to D	∞	4	1	1

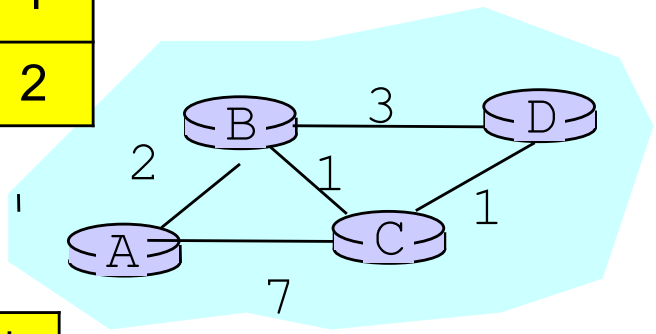
Check: *All nodes know the best three-hop paths.*

from Node B

	via A	via C	via D	min dist
to A	2	4	8	2
to B	-	-	-	0
to C	5	1	4	1
to D	7	2	3	2

from Node D

	via B	via C	min dist
to A	5	4	4
to B	3	2	2
to C	4	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	3
to D	4	8	4

Check

from Node C

	via A	via B	via D	min dist
to A	7	3	6	3
to B	9	1	3	1
to C	-	-	-	0
to D	12	3	1	1

Example: End of 3nd Full Exchange

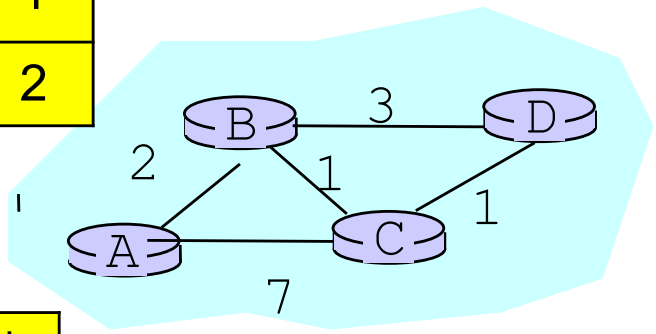
No further change in DVs → Convergence!

from Node B

	via A	via C	via D	min dist
to A	2	4	7	2
to B	-	-	-	0
to C	5	1	4	1
to D	6	2	3	2

from Node D

	via B	via C	min dist
to A	5	4	4
to B	3	2	2
to C	4	1	1
to D	-	-	0



from Node A

	via B	via C	min dist
to A	-	-	0
to B	2	8	2
to C	3	7	3
to D	4	8	4

from Node C

	via A	via B	via D	min dist
to A	7	3	5	3
to B	9	1	3	1
to C	-	-	-	0
to D	11	3	1	1

Intuition

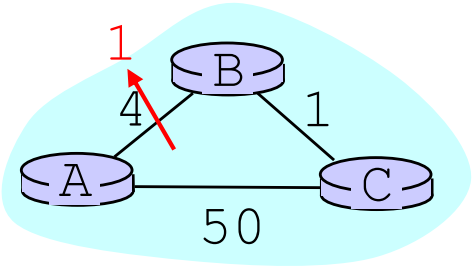
- Initial state: best one-hop paths
- One simultaneous round: best two-hop paths
- Two simultaneous rounds: best three-hop paths
- …
- Kth simultaneous round: best (k+1) hop paths

- Must eventually converge
  - as soon as it reaches longest best path
- …..but how does it respond to changes in cost?

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



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

via

Node A

to	B	C
B	4	51
C	5	50

to	B	C
B	1	51
C	2	50

to	B	C
B	1	51
C	3	1

to	B	C
B	1	51
C	3	1

to	B	C
B	1	51
		50

Node B

to	A	C
A	4	6
C	9	1

to	A	C
A	1	6
C	6	1

to	A	C
		3
C	3	1

Node C

to	A	B
A	50	5
B	54	1

to	A	B
A	50	5
B	54	1

to	A	B
A	50	5
B	51	1

to	A	B
A	50	2
B	51	1

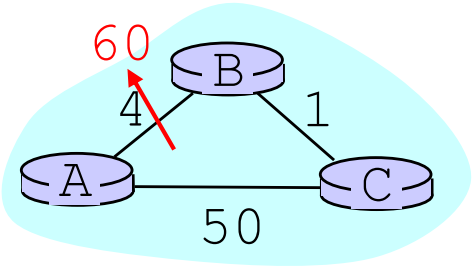
to	A	B
A	50	2
B	51	1

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	B	C
B	4	51
C	5	50

	B	C
B	60	51
C	61	50

Node B

	A	C
A	4	6
C	9	1

	A	C
A	60	6
C	65	1

Node C

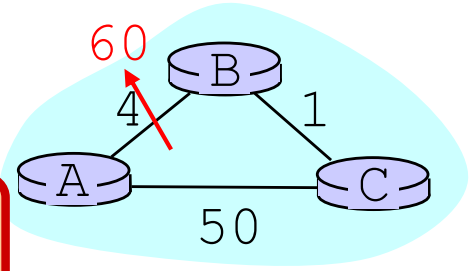
	A	B
A	50	5
B	54	1

	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

Stable state

A-B changed

A sends its DV to B, C

B sends its DV to A, C

C sends its DV to A, B

via

to

Node A

	B	C
B	4	51
C	5	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

Node B

	A	C
A	4	6
C	9	1

	A	C
A	60	6
C	65	1

	A	C
A	60	6
C	110	1

	A	C
A	60	6
C	110	1

	A	C
A	60	8
C	110	1

Node C

	A	B
A	50	5
B	54	1

	A	B
A	50	5
B	54	1

	A	B
A	50	5
B	101	1

	A	B
A	50	7
B	101	1

	A	B
A	50	7
B	101	1

Link cost changes here

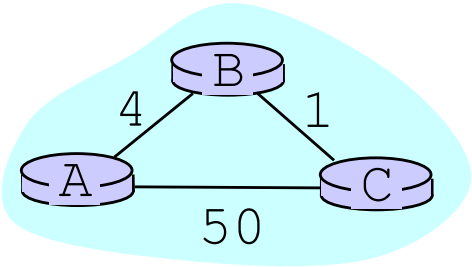
"bad news travels slowly"
(not yet converged)

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

Node A

to	B	C
B	4	51
C	5	50

Mindist: 4, 5

Mindist: 4, ∞

Node B

	A	C
A	4	∞
C	∞	1

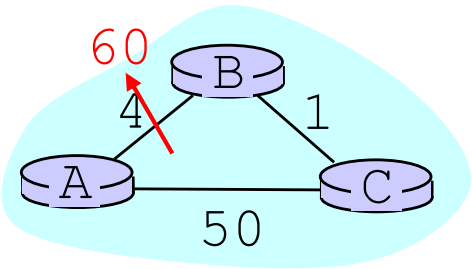
Node C

	A	B
A	50	5
B	54	1

Mindist: 5, 1

Mindist: ∞, 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

via

Node A

to	via B	via C
B	4	51
C	5	50

to	via B	via C
B	60	51
C	61	50

Node B

to	via A	via C
A	4	∞
C	∞	1

to	via A	via C
A	60	6
C	65	1

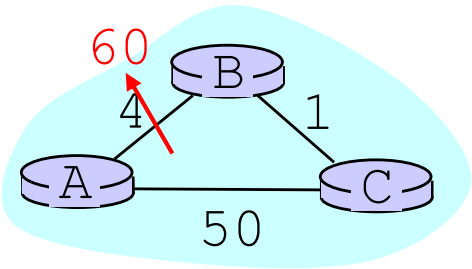
Node C

to	via A	via B
A	50	5
B	54	1

to	via A	via 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

Stable state

A-B changed

A sends its DV to B, C

B sends its DV to A, C

via

to

Node A

	B	C
B	4	51
C	5	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

Node B

	A	C
A	4	∞
C	∞	1

	A	C
A	60	∞
C	∞	1

	A	C
A	60	∞
C	110	1

	A	C
A	60	∞
C	110	1

Node C

	A	B
A	50	5
B	54	1

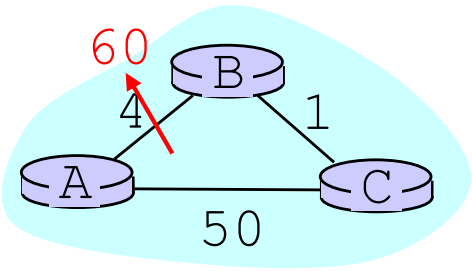
	A	B
A	50	5
B	54	1

	A	B
A	50	5
B	∞	1

	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

via

to

Node A

	B	C
B	4	51
C	5	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

Node B

	A	C
A	4	∞
C	∞	1

	A	C
A	60	∞
C	∞	1

	A	C
A	60	∞
C	110	1

	A	C
A	60	∞
C	110	1

Node C

	A	B
A	50	5
B	54	1

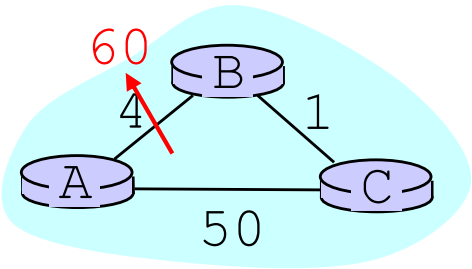
	A	B
A	50	5
B	54	1

	A	B
A	50	5
B	∞	1

	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

Stable state

A-B changed

A sends its DV to B, C

B sends its DV to A, C

C sends its DV to A, B

Node A

via B C

to	B	C
B	4	51
C	5	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

	B	C
B	60	51
C	61	50

Node B

	A	C
A	4	∞
C	∞	1

	A	C
A	60	∞
C	∞	1

	A	C
A	60	∞
C	110	1

	A	C
A	60	∞
C	110	1

	A	C
A	60	51
C	110	1

Node C

	A	B
A	50	5
B	54	1

	A	B
A	50	5
B	54	1

	A	B
A	50	5
B	∞	1

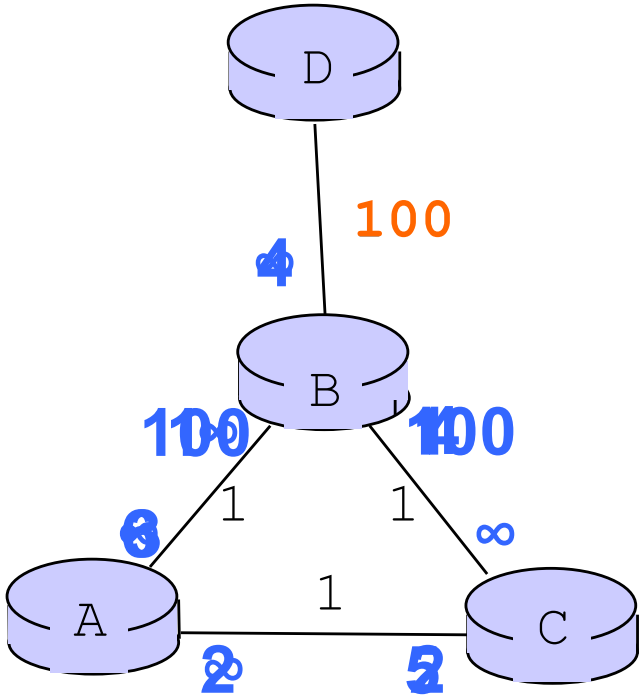
	A	B
A	50	61
B	∞	1

	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

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

www.zeetings.com/salil

Quiz: Distance-vector routing

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

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

www.zeetings.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

Comparison of LS and DV algorithms

message complexity

- ***LS:*** with n nodes, E links, O(nE) msgs sent
- ***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

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)

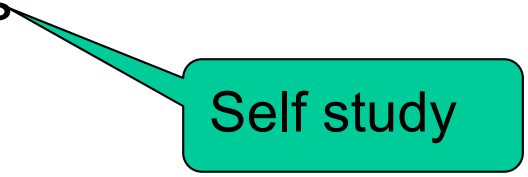
Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

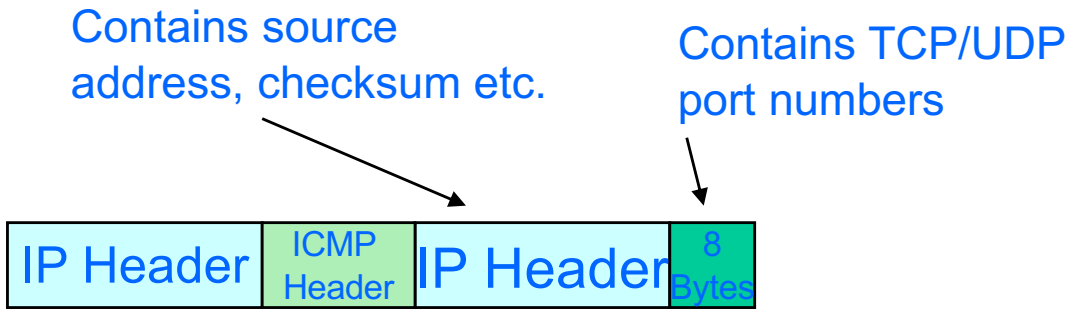
- link state
- distance vector
- hierarchical routing

5.6 ICMP: The Internet Control Message Protocol



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

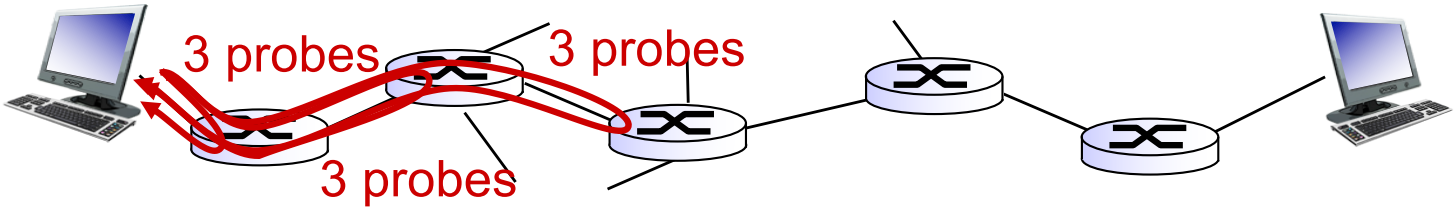


ICMP: Internet Control Message Protocol

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