COMP 3331/9331: Computer Networks and Applications

Week 9

Network Layer: Control Plane (Routing)

Chapter 5: Section 5.1 – 5.2, 5.6

Network layer control plane

Goals: understand principles behind network control plane

- traditional routing algorithms

and their instantiation, implementation in the Internet:

- RIP, OSPF, BGP (NOT COVERED)

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

Network-layer functions

Recall: two 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

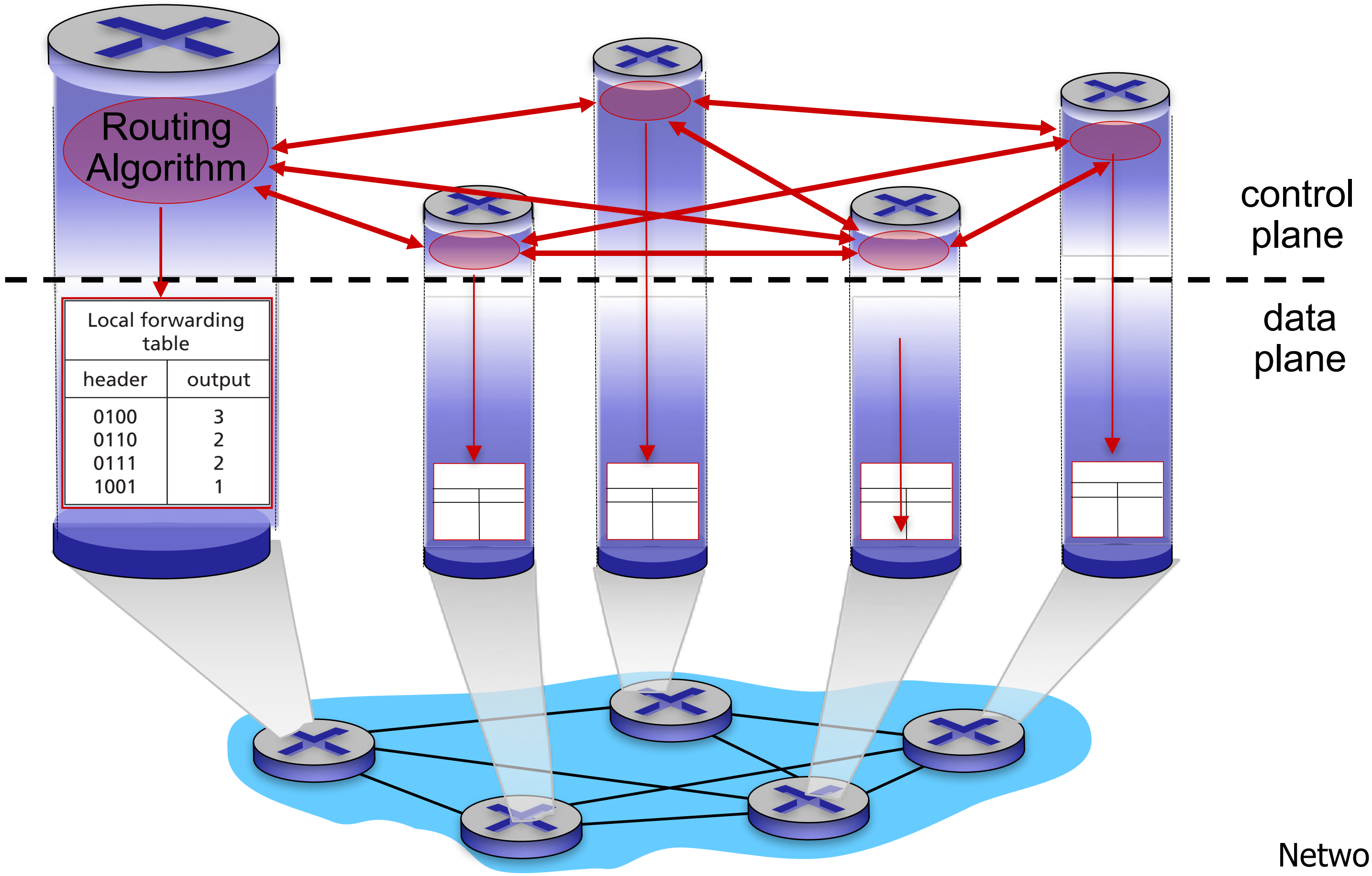
control plane

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 with each other in control plane to compute forwarding tables



Routing Algorithm

control plane

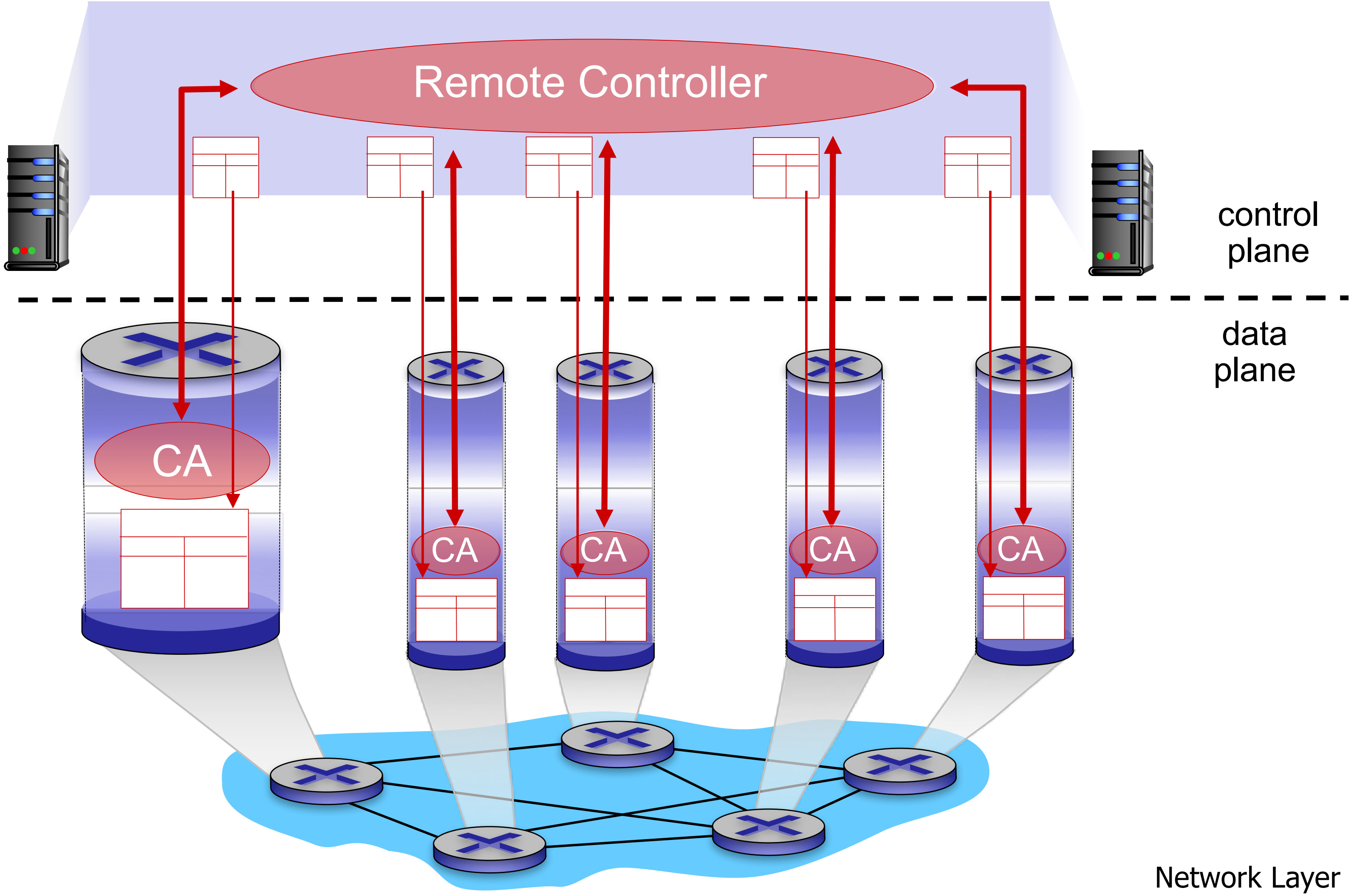
data plane

Local forwarding table

header	output
0100	3
0110	2
0111	2
1001	1

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

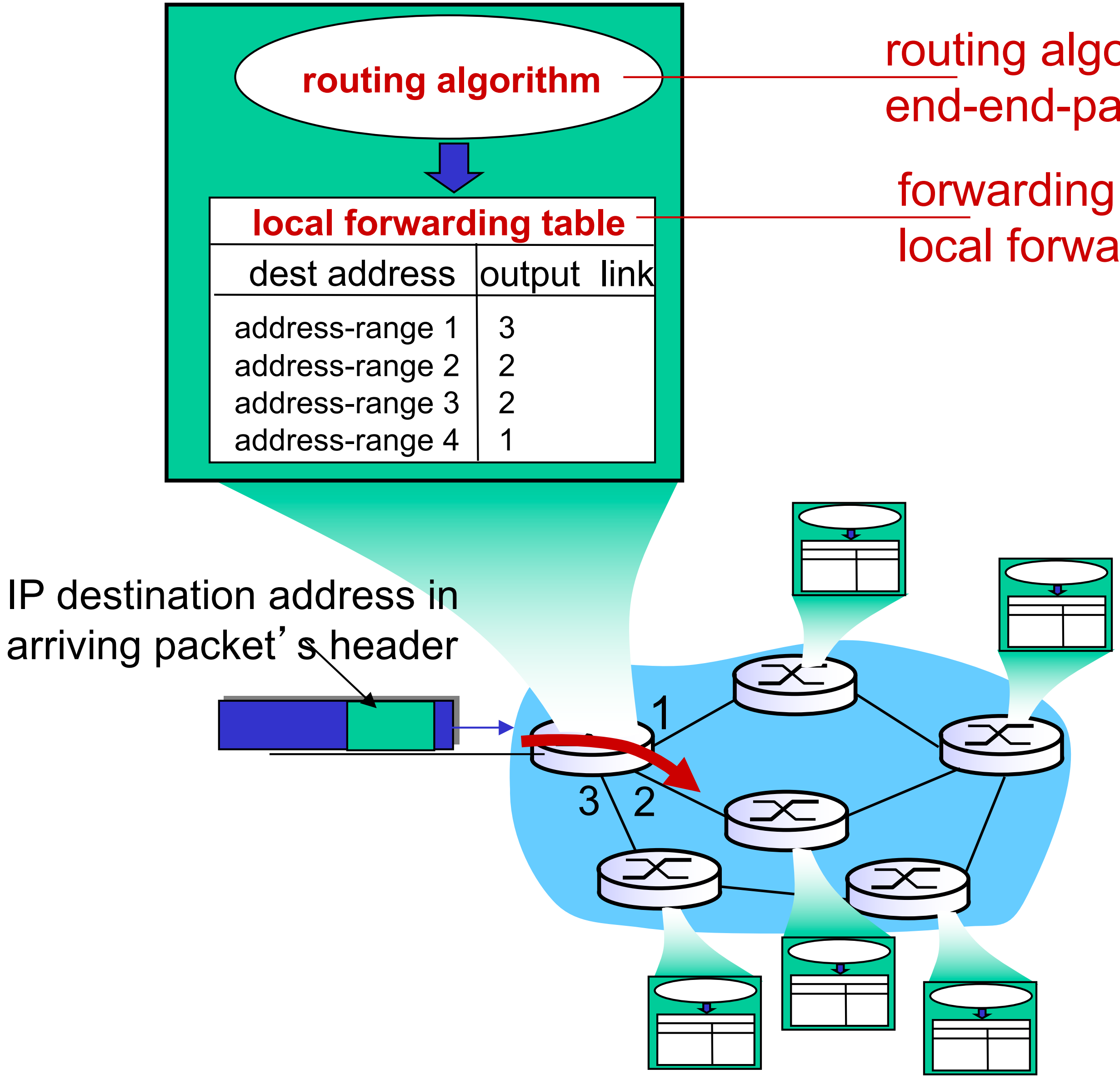
5.1 introduction

5.2 routing protocols

- link state
- distance vector
- Hierarchical routing

5.6 ICMP: The Internet Control Message Protocol

Interplay between routing, forwarding



routing algorithm

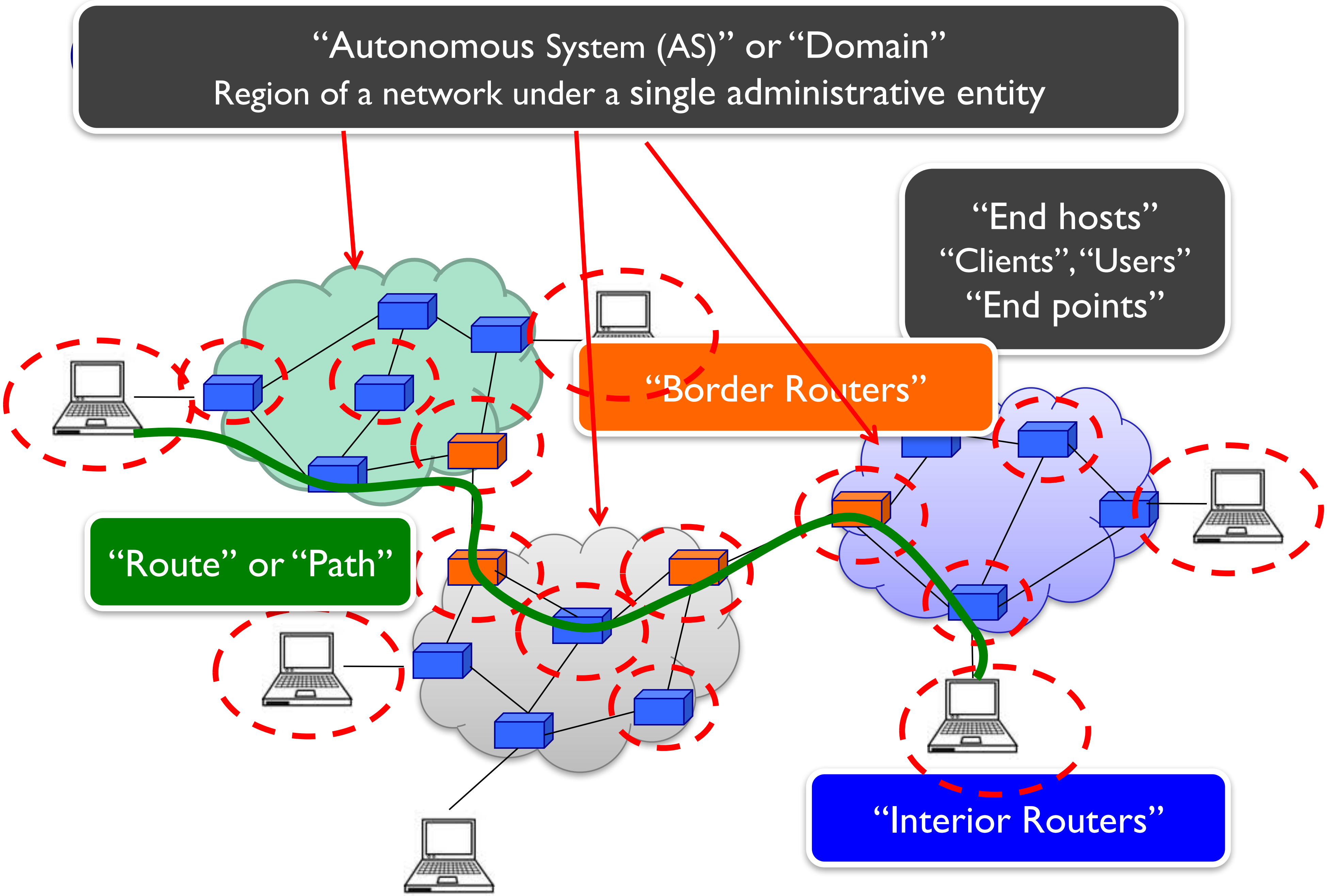
local forwarding table

dest address	output link
address-range 1	3
address-range 2	2
address-range 3	2
address-range 4	1

IP destination address in arriving packet's header

routing algorithm determines end-end-path through network

forwarding table determines local forwarding at this router



“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”

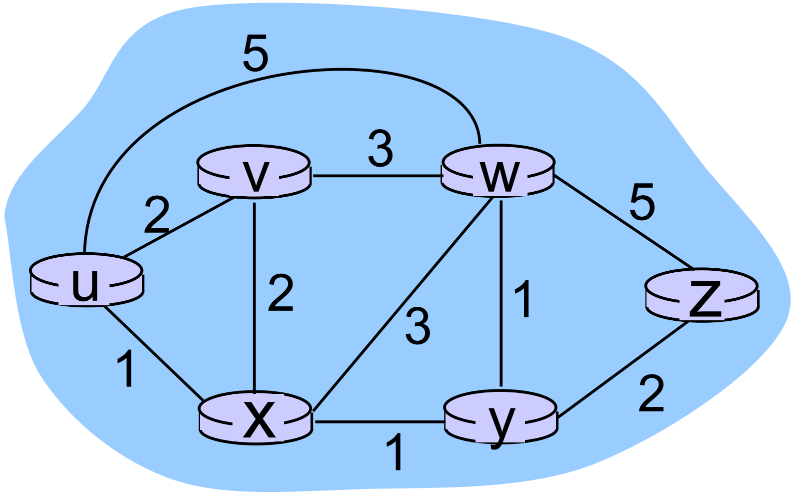
Internet Routing

- 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

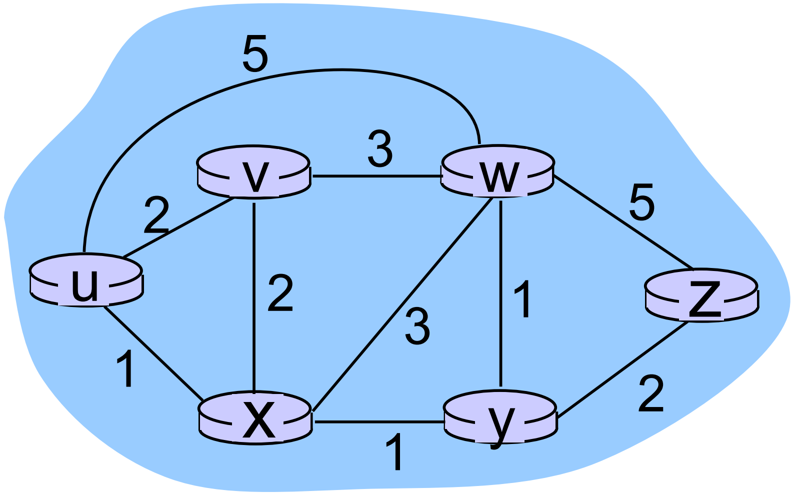


graph: G = (N,E)

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

Quiz: How should link costs be determined?

A: They should all be equal.

B: They should be a function of link capacity.

C: They should take current traffic characteristics into account (congestion, delay, etc.).

D: They should be manually determined by network administrators.

E: They should be determined in some other way.

Link Cost

- Typically simple: all links are equal
- Least-cost paths => shortest paths (hop count)
- Network operators add policy exceptions
  - Lower operational costs
  - Peering agreements
  - Security concerns

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

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

Network layer, control plane: outline

5.1 introduction

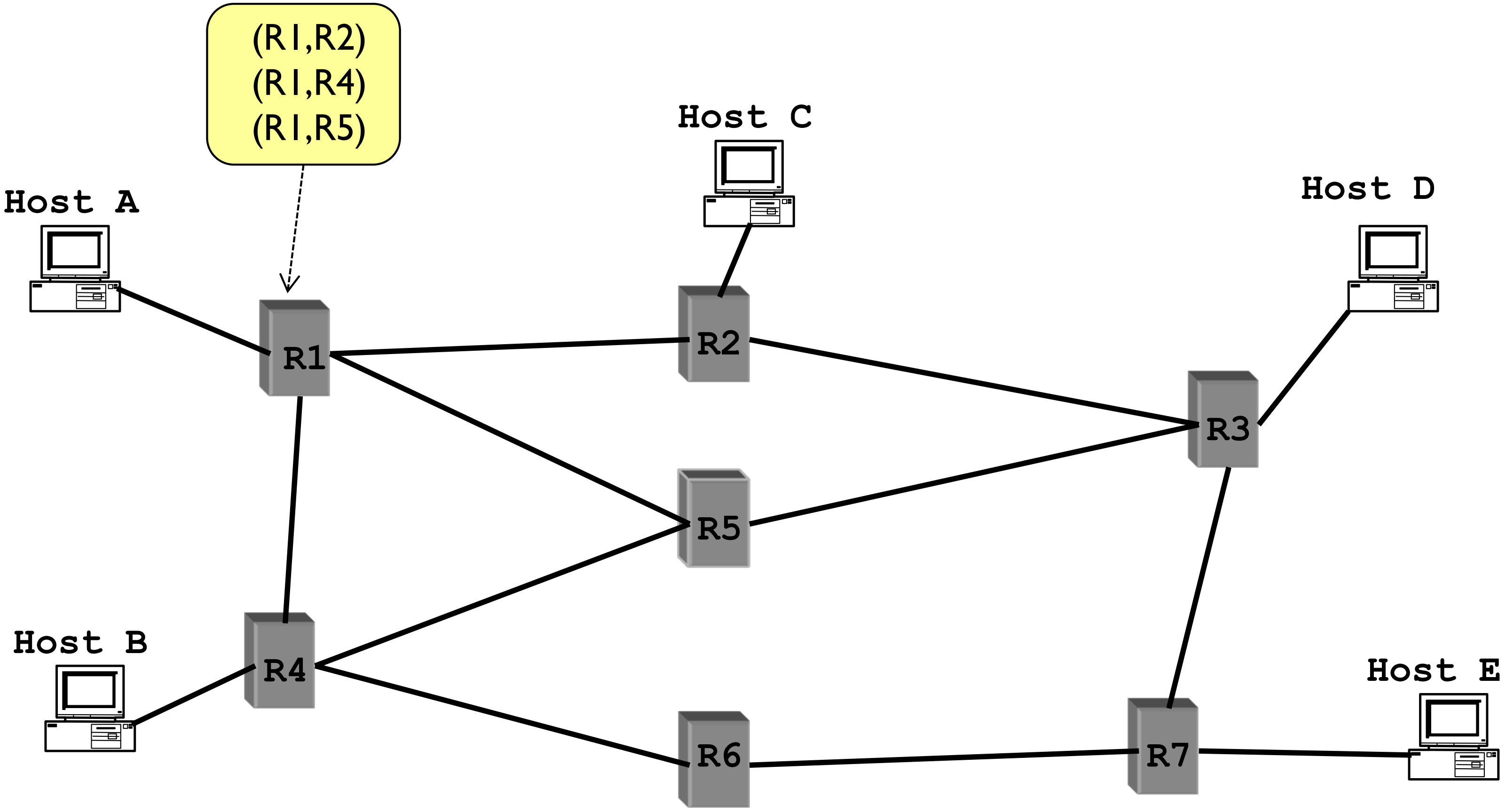
5.2 routing protocols

- link state
- distance vector
- hierarchical routing

5.6 ICMP: The Internet Control Message Protocol

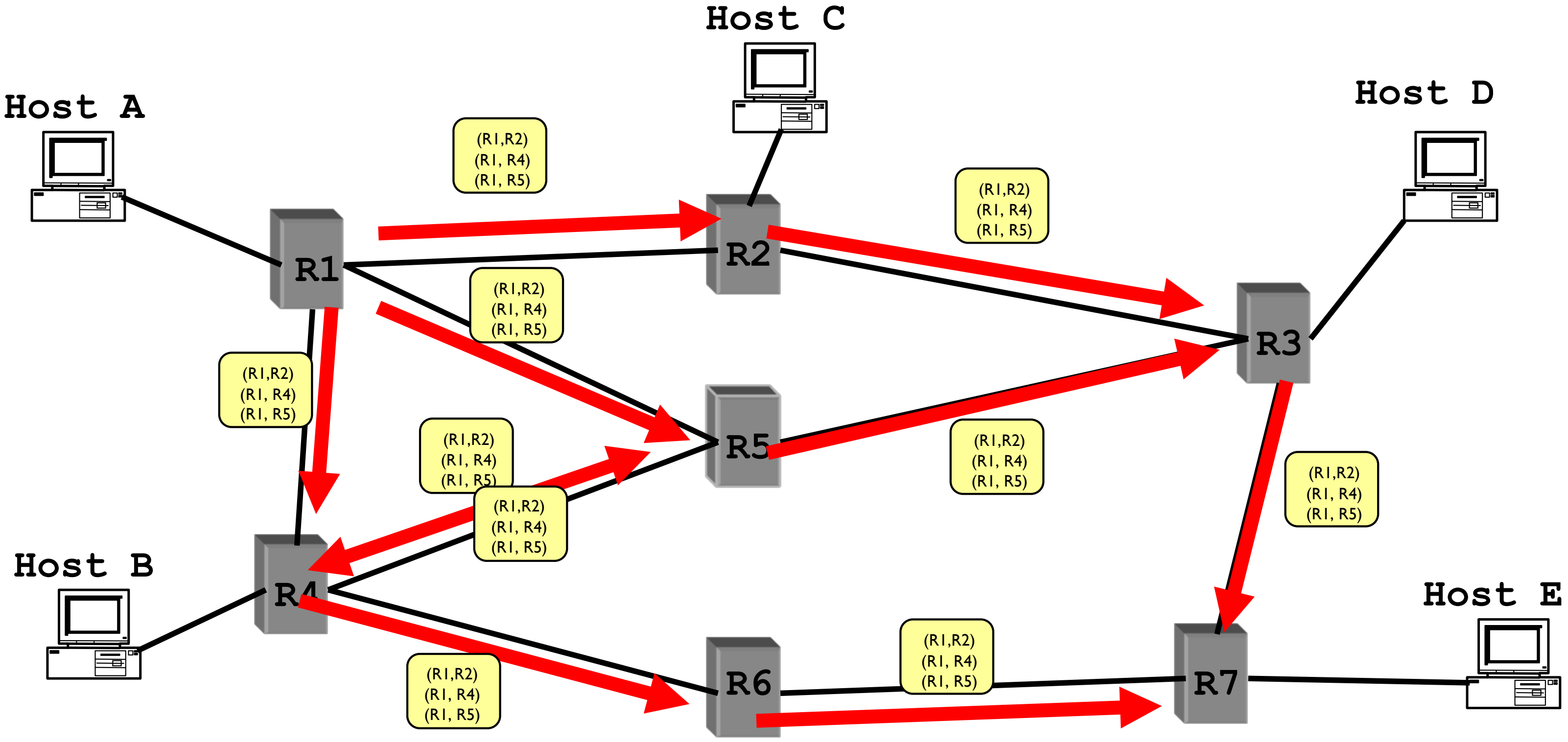
Link State Routing

- 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

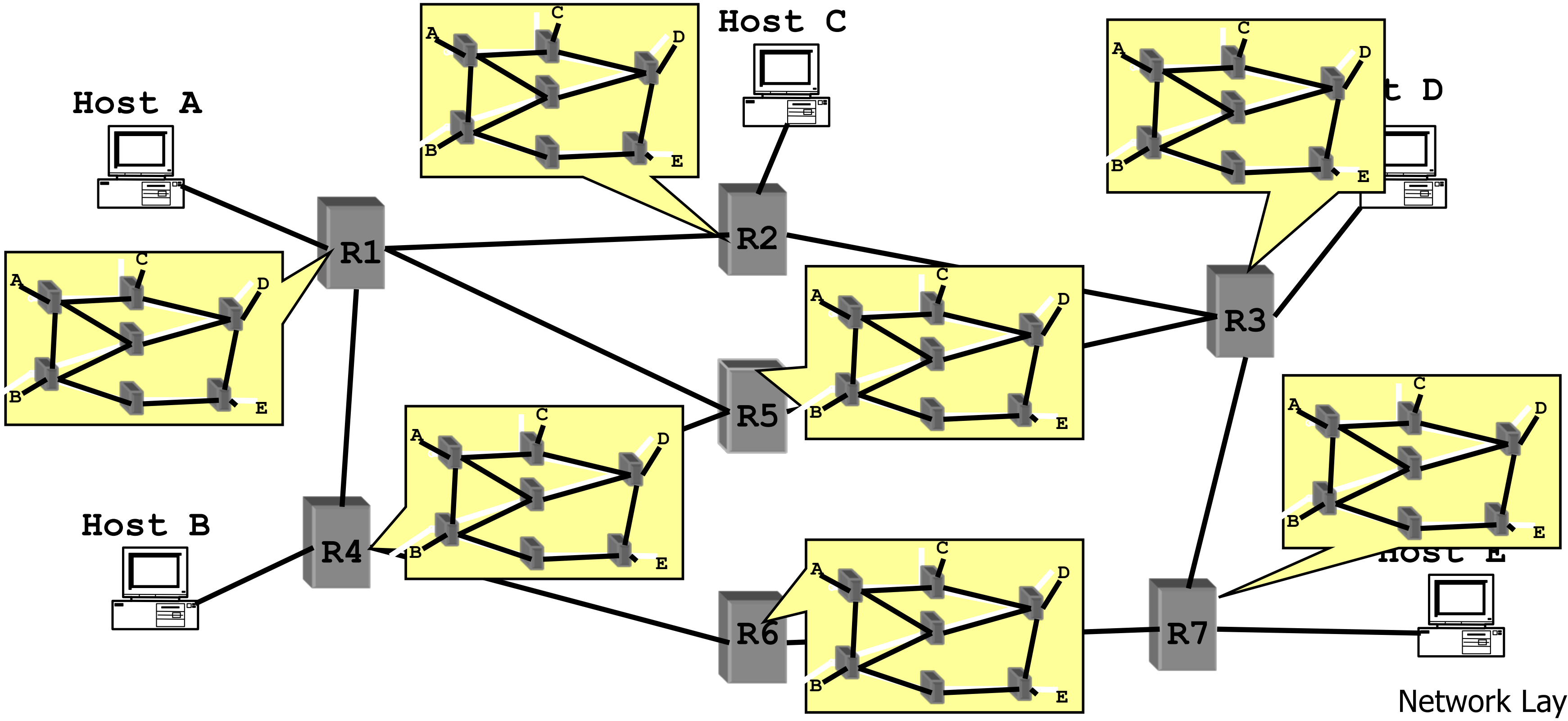


Flooding LSAs

- 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



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

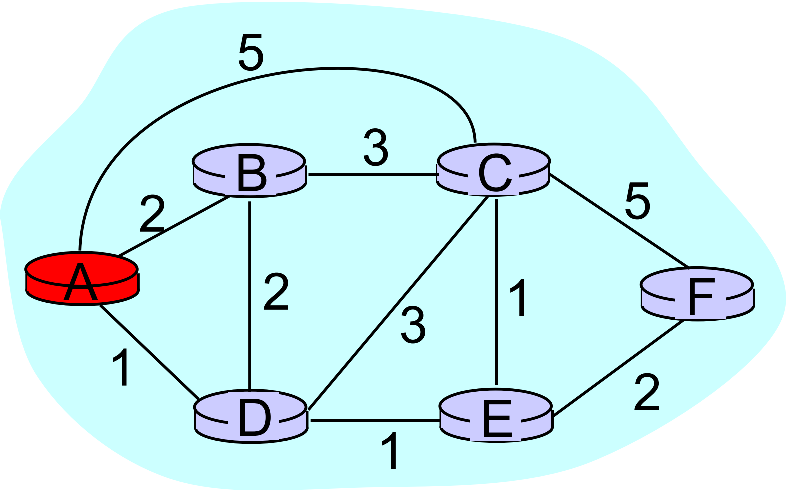
Dijsktra’ s Algorithm

```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    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 cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N'
```

Example: Dijkstra’s Algorithm

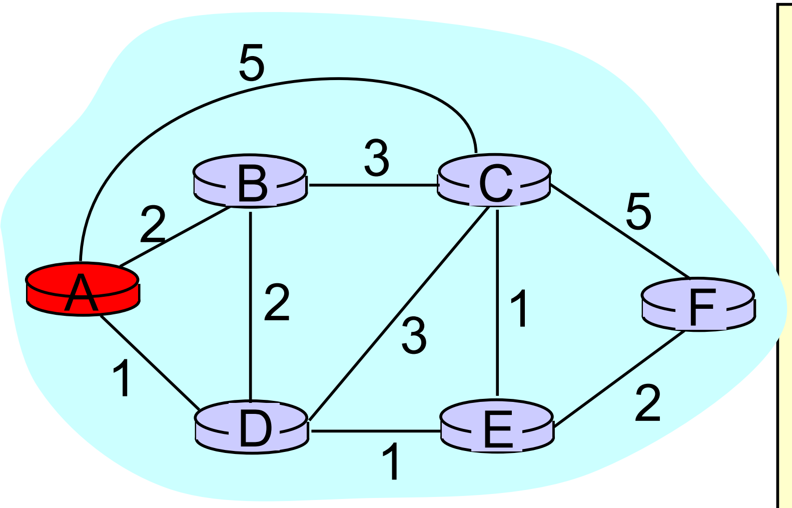
Step	Set N’	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						



```
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

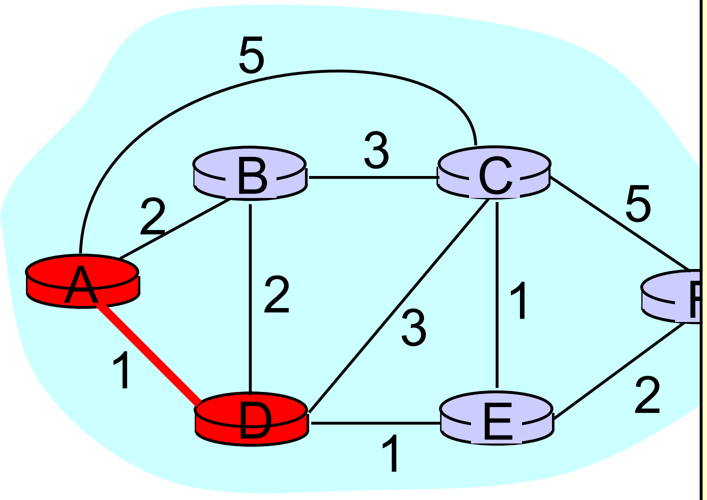
Step	Set N’	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						



```
…
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

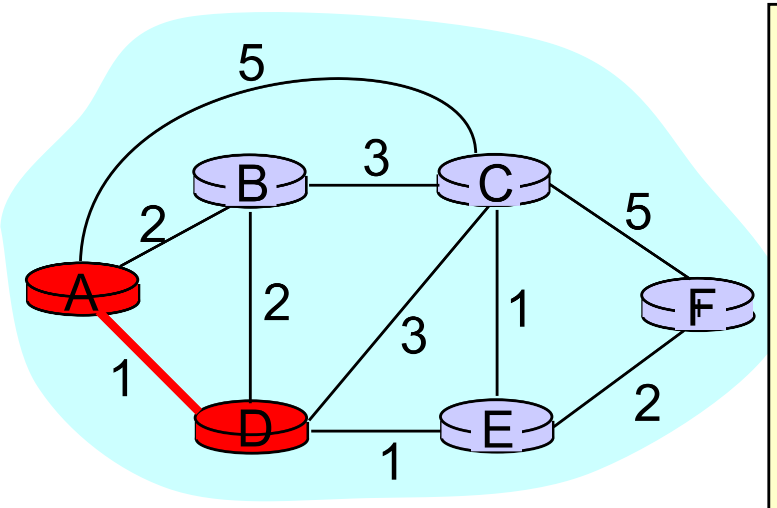
Step	Set N’	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD					
2						
3						
4						
5						



```
…
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

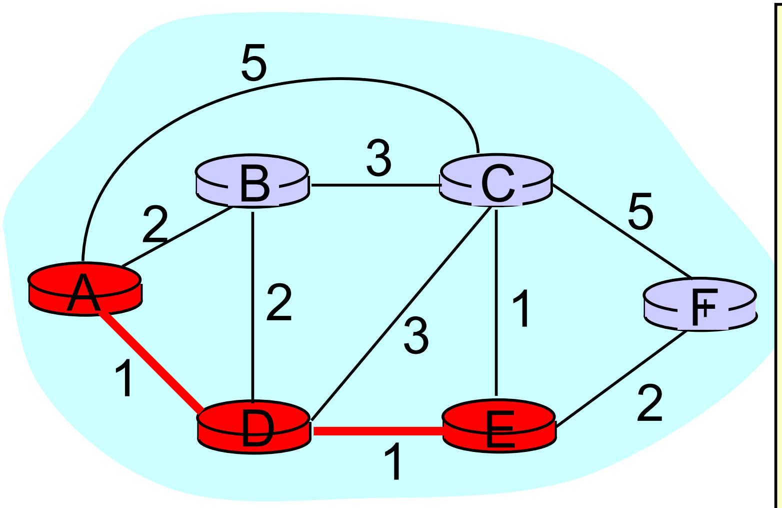
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2, A	4,D		2,D	
2						
3						
4						
5						



```
…
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

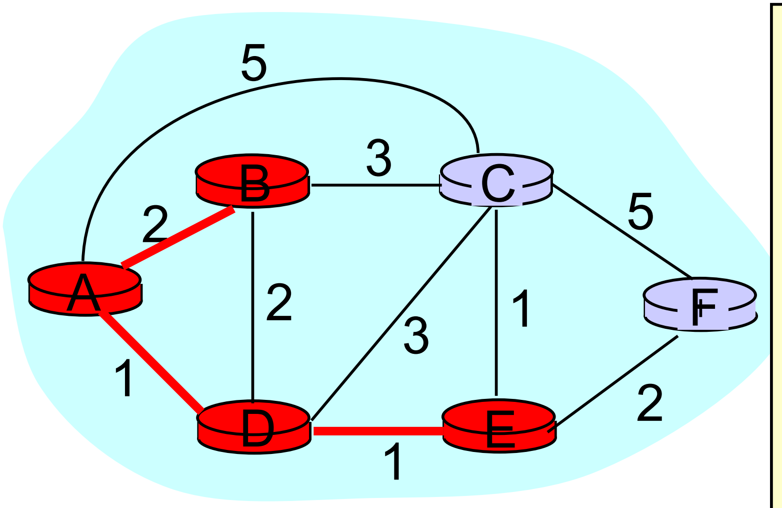
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2, A	4,D		2,D	
2	ADE	2, A	3,E			4,E
3						
4						
5						



```
…
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

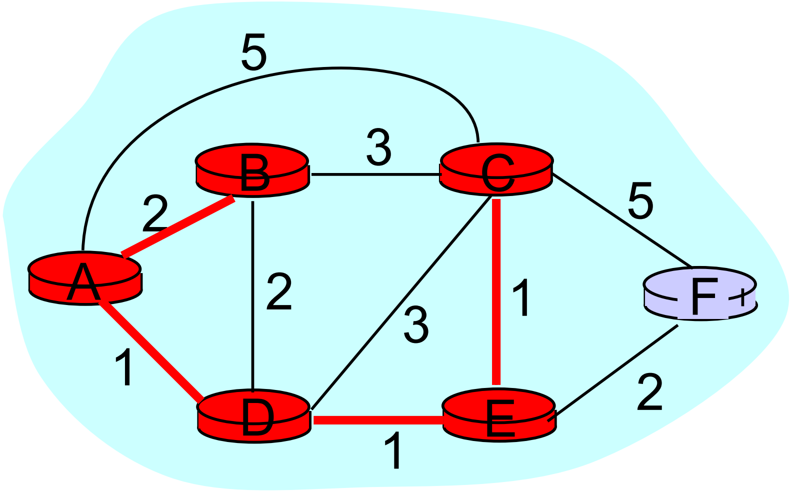
Step	Set N’	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4						
5						



```
…
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

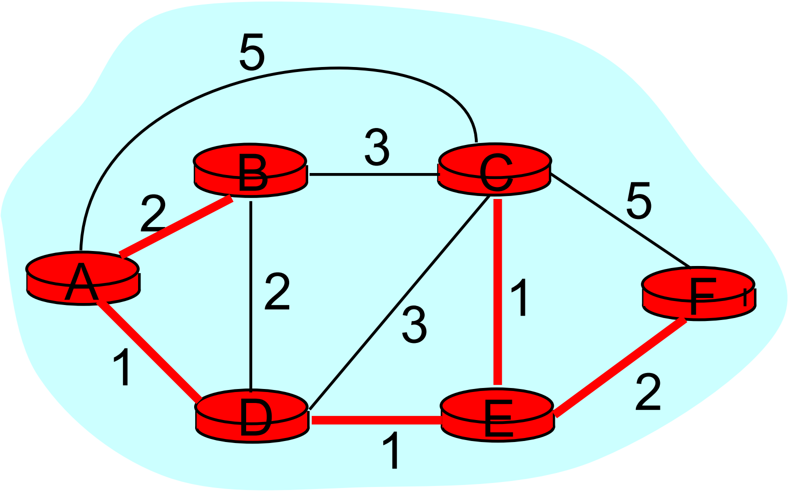
Step	Set N’	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5						



```
…
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

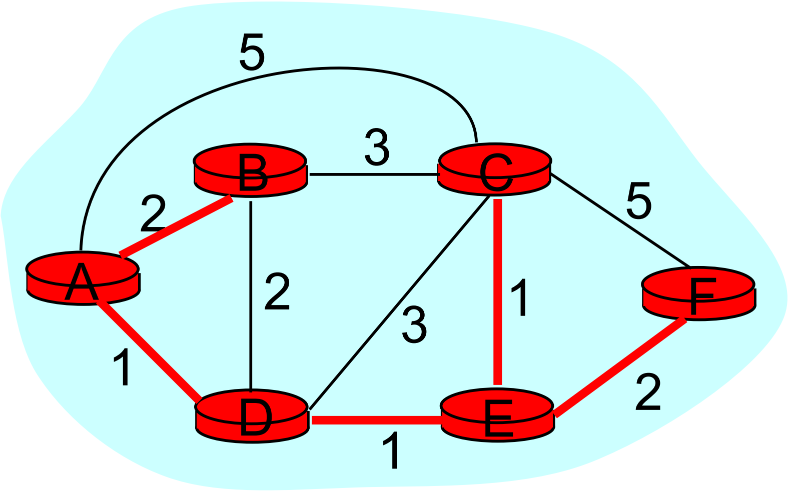
Step	Set N’	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					



```
…
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					

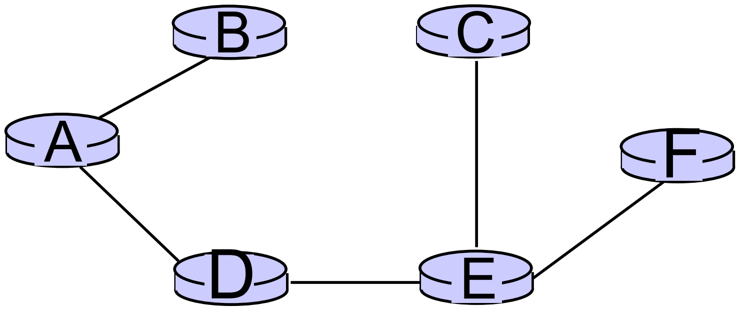


To determine path A → C (say), work backward from C via p(v)

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:



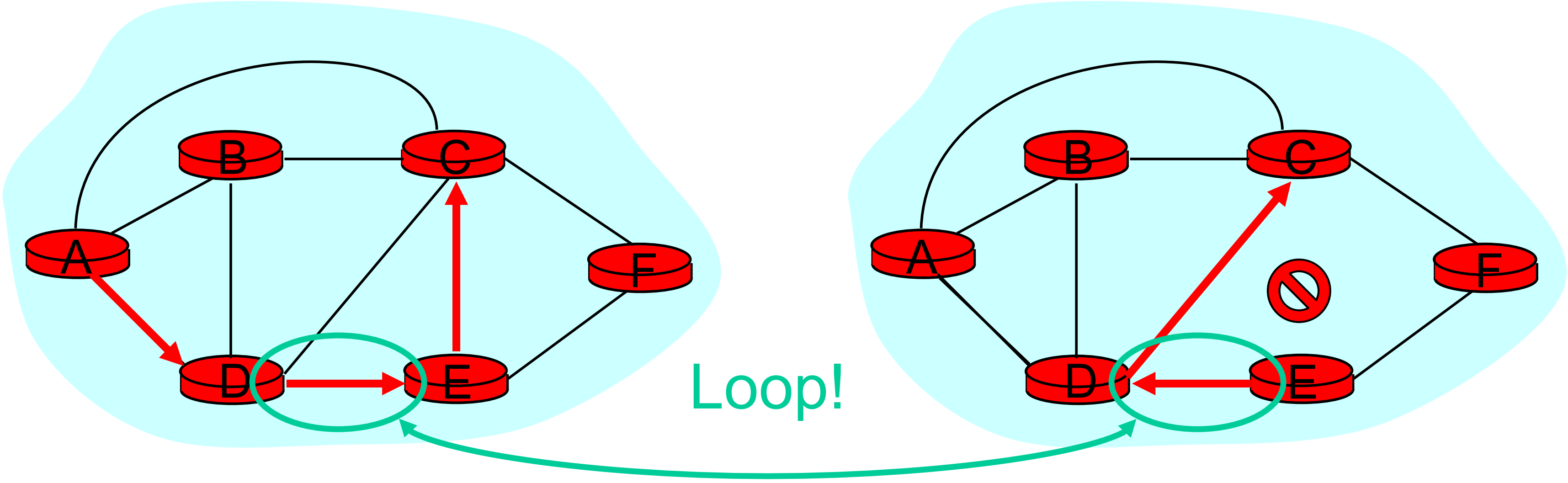
Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

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



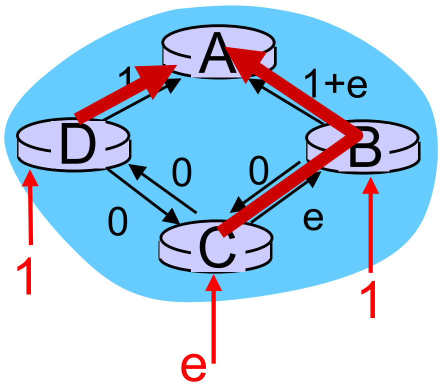
A and D think that this is the path to C

E thinks that this is the path to C

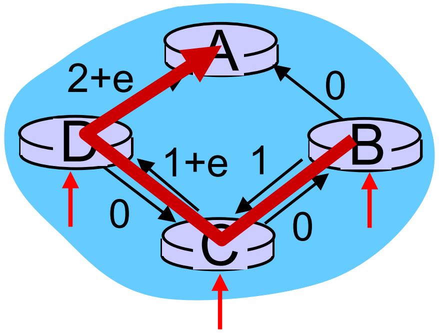
Oscillations

oscillations possible:

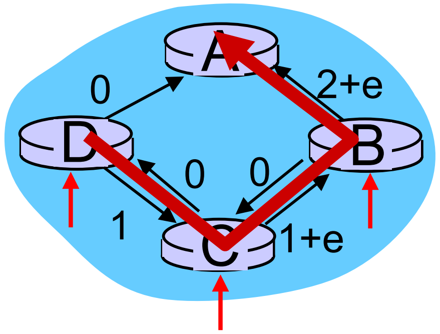
- e.g., suppose link cost equals amount of carried traffic:



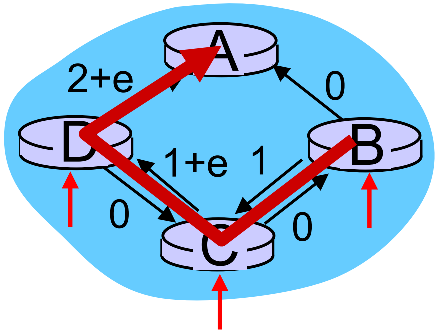
initially



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

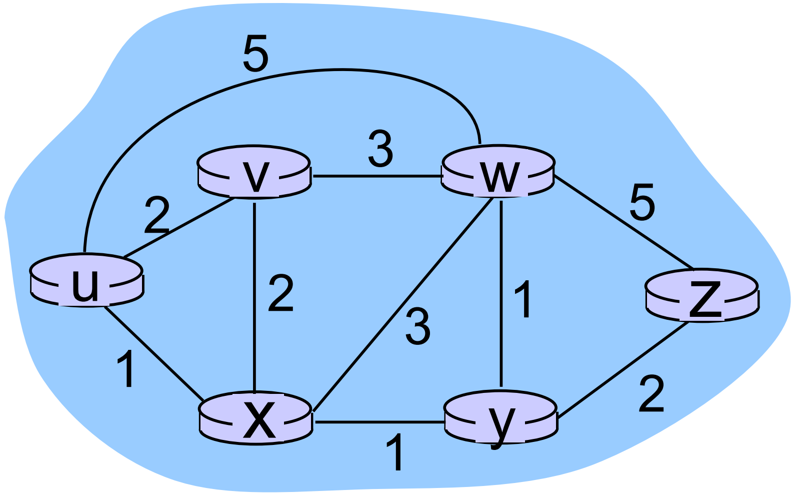
$$d_x(y) = \underset{v}{\mathit{min}} \{c(x,v) + d_v(y)\}$$

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

Bellman-Ford example



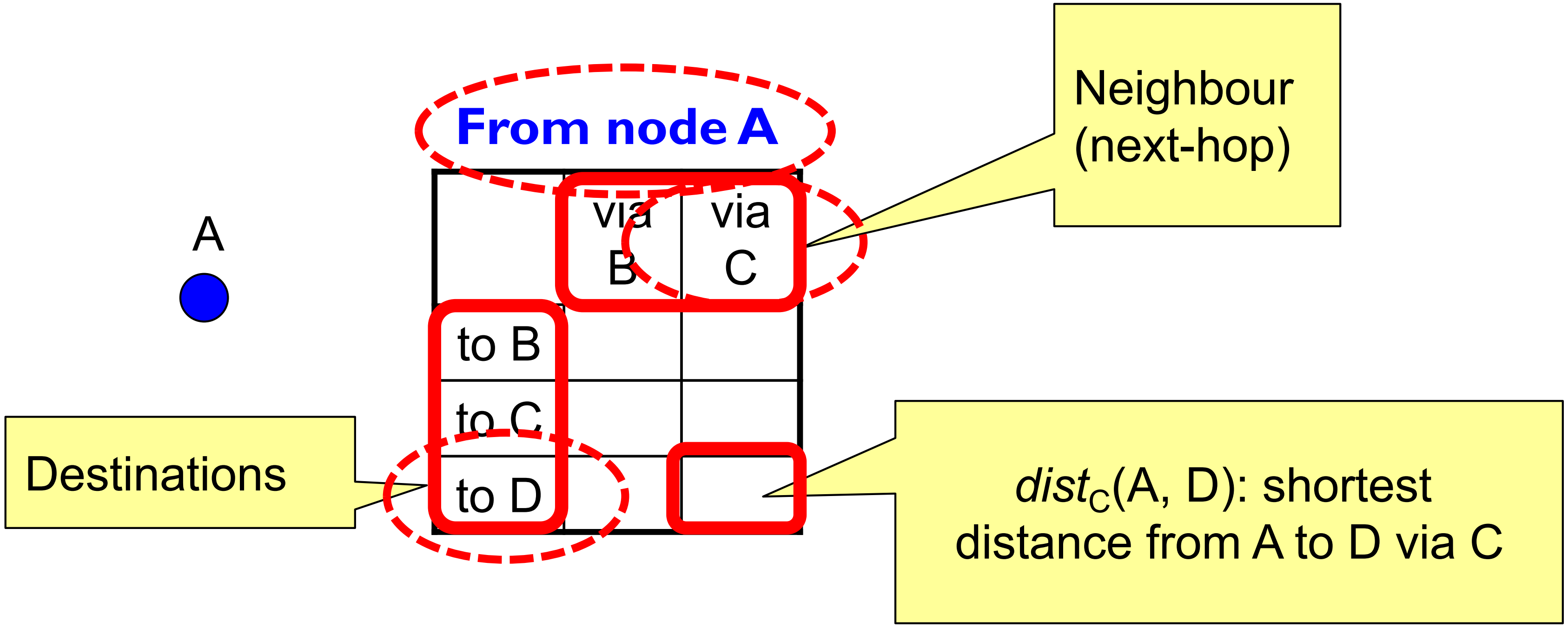
clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

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

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

How Distance-Vector (DV) works



From node A

	via B	via C
to B		
to C		
to D		

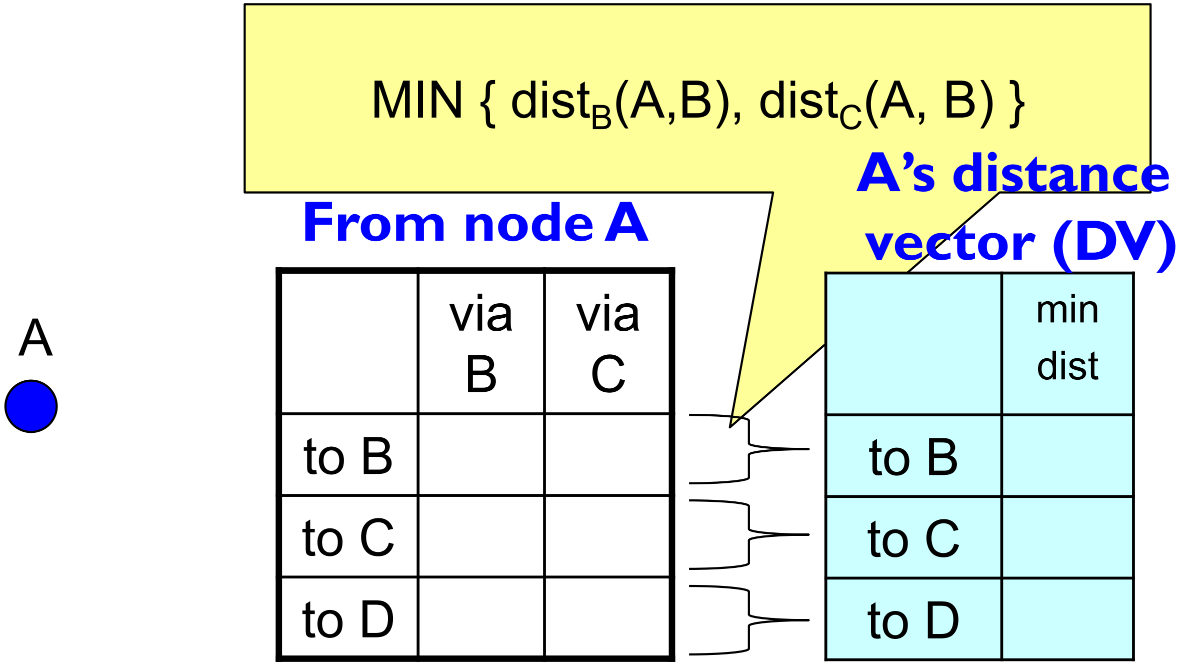
Neighbour (next-hop)

Destinations

dist$_C$(A, D): shortest distance from A to D via C

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

How Distance-Vector (DV) works



MIN { $\text{dist}_B(A,B)$, $\text{dist}_C(A, B)$ }

From node A

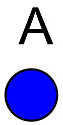
	via B	via C
to B		
to C		
to D		

A's distance vector (DV)

	min dist
to B	
to C	
to D	

Each router computes its shortest distance to every destination via <u>any</u> of its neighbors

How Distance-Vector (DV) works



From node A

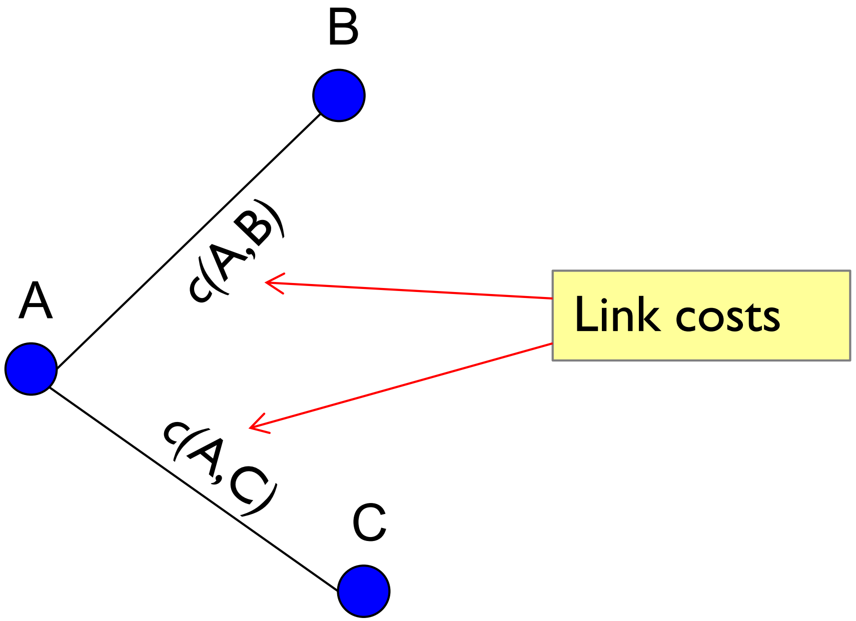
	via B	via C
to B	?	?
to C	?	?
to D	?	?

A's DV

	min dist
to B	?
to C	?
to D	?

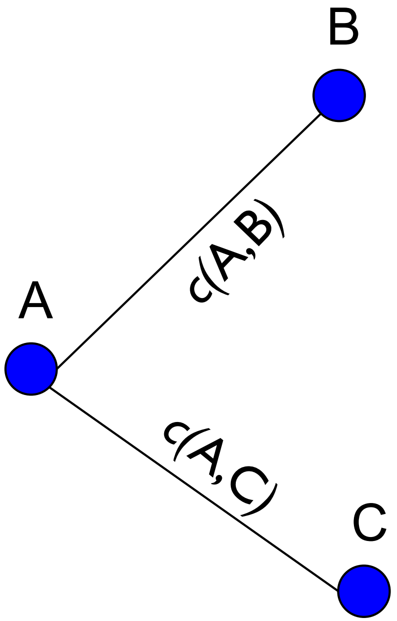
How does A initialize its dist() table and DV?

How Distance-Vector (DV) works



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

How Distance-Vector (DV) works



From node A

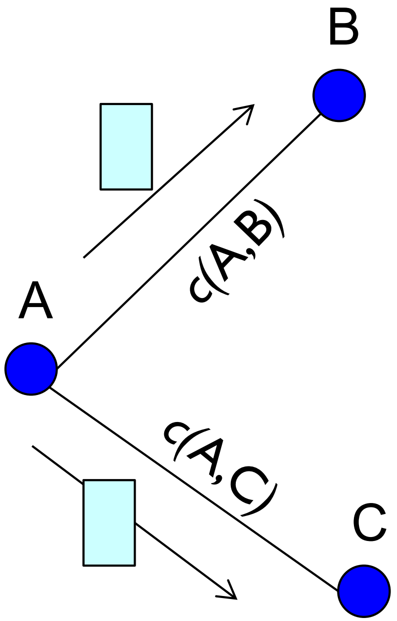
	via B	via C
to B	*c*(A,B)	∞
to C	∞	*c*(A,C)
to D	∞	∞

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

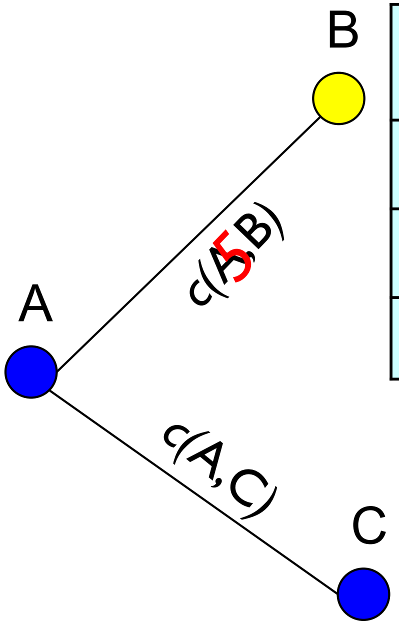
	via B	via C
to B	*c*(A,B)	∞
to C	∞	*c*(A,C)
to D	∞	∞

A's DV

	mindist
to B	5
to C	6
to D	2

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

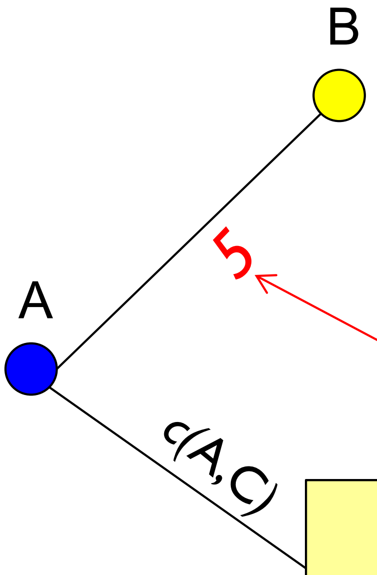
	via A	via C
to A	5	∞
to C	15	1
to D	∞	∞

B's DV

	mindist
to A	5
to C	1
to D	∞

Routers process received DVs

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	11	1
to D	7	∞

B's DV

	mindist
to A	5
to C	1
to D	7

new = c(B,A) + *mindist*(A, D)

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

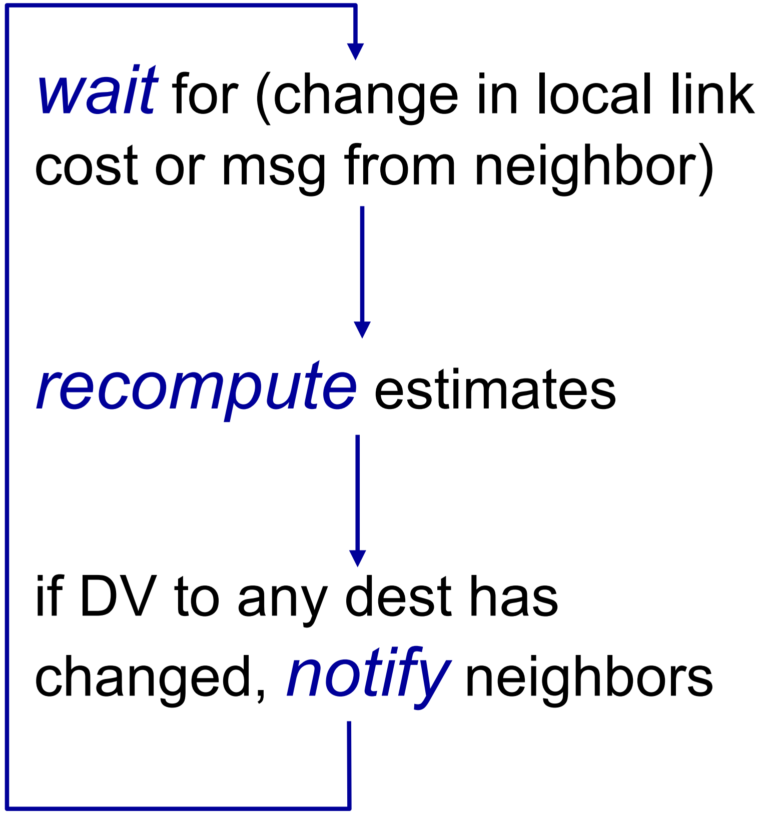
iterative, asynchronous:
each local iteration caused by:

- 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:



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

recompute estimates

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

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

```
At node A
Initialization:
  for all destinations V do
    if V is neighbor of A
      $dist_V(A, V)$ = mindist(A,V) = c(A,V);
    else
      $dist_V(A, V)$ = mindist(A,V) = $\infty$;
  send mindist(A, *) to all neighbors
loop:
  wait (until A sees a link cost change to neighbor V  /* case 1 */
        or until A receives mindist(V,*) from neighbor V)    /* case 2 */
  if (c(A,V) changes by $\pm d$)  /* $\Leftarrow$ case 1 */
    for all destinations Y do
      $dist_V(A,Y) = dist_V(A,Y) \pm d$
  else /* $\Leftarrow$ case 2: */
    for all destinations Y do
      $dist_V(A,Y)$ = c(A,V) + mindist(V, Y);
  update mindist(A,*)
  if (there is a change in mindist(A, *))
    send mindist(A, *) to all neighbors
forever
```

Distance Vector

- c(i,j): link cost from node *i* to *j*
- $\text{dist}_Z(A,V)$: shortest dist. from A to V via Z
- mindist(A,V): shortest dist. from A to V

```
At node A
Initialization:
  for all destinations V do
      if V is neighbor of A
          $\text{dist}_V(A, V) = \text{mindist}(A,V) = c(A,V)$;
      else
          $\text{dist}_V(A, V) = \text{mindist}(A,V) = \infty$;
  send mindist(A, *) to all neighbors
loop:
  wait (until A sees a link cost change to neighbor V  /* case 1 */
        or until A receives mindist(V,*) from neighbor V)   /* case 2 */
  if (c(A,V) changes by $\pm d$)  /* $\Leftarrow$ case 1 */
      for all destinations Y do
          $\text{dist}_V(A,Y) = \text{dist}_V(A,Y) \pm d$
  else /* $\Leftarrow$ case 2: */
      for all destinations Y do
          $\text{dist}_V(A,Y) = c(A,V) + \text{mindist}(V, Y)$;
  update mindist(A,*)
  if (there is a change in mindist(A, *))
      send mindist(A, *) to all neighbors
forever
```

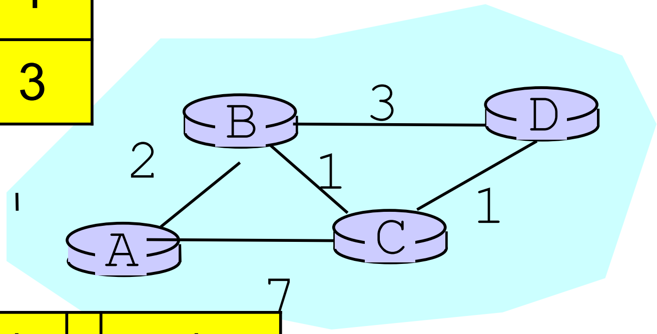
Example: Initialization

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

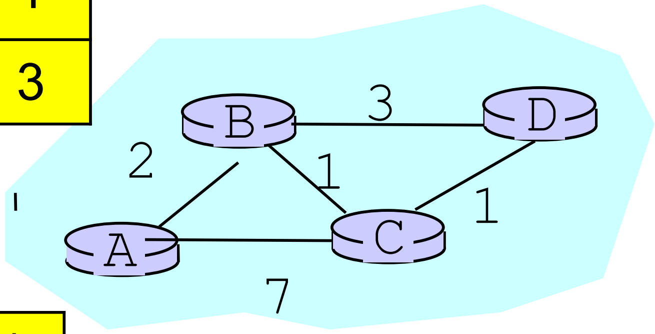
Example: C sends update to A

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



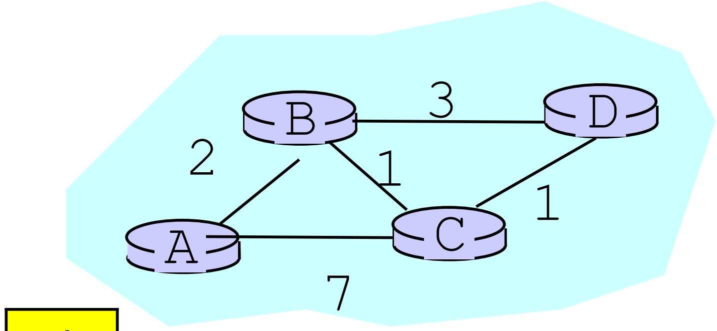
from Node A

	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



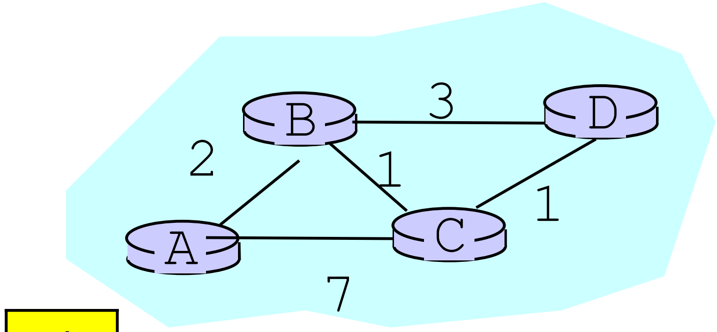
from Node A

	via B	via C
to A	-	-
to B	2	∞
to C	∞	7
to D	∞	∞

min dist
0
2
7
∞

min dist
7
1
0
1

Example: C sends update to A



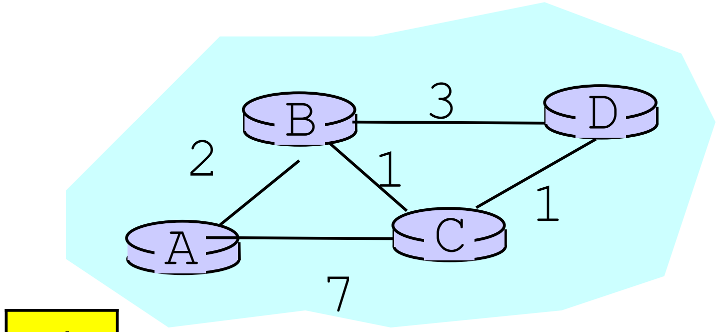
from Node A

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

min dist
0
2
7
∞

min dist
7
1
0
1

Example: C sends update to A



from Node A

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

min dist
0
2
7
8

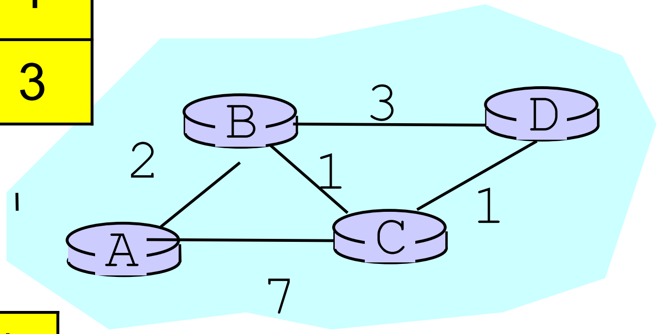
Example: now B sends update to A

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

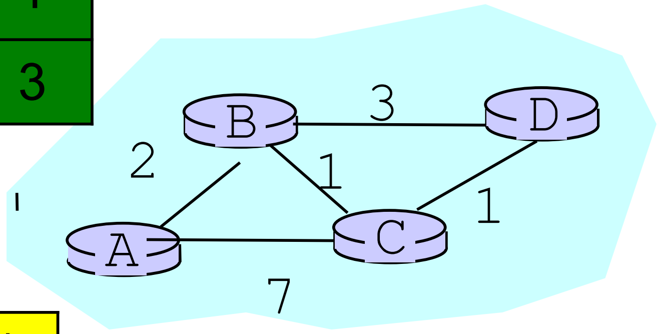
Example: now B sends update to A

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

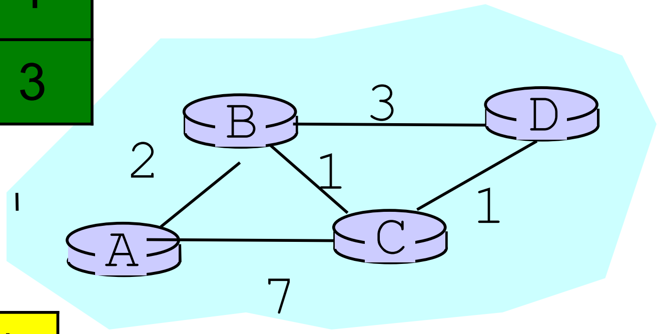
Example: now B sends update to A

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
				7
				1
to C	-	-	-	0
to D	∞	∞	1	1

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

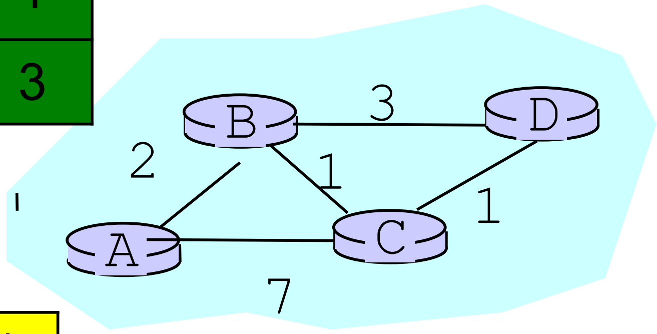
Example: now B sends update to A

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.

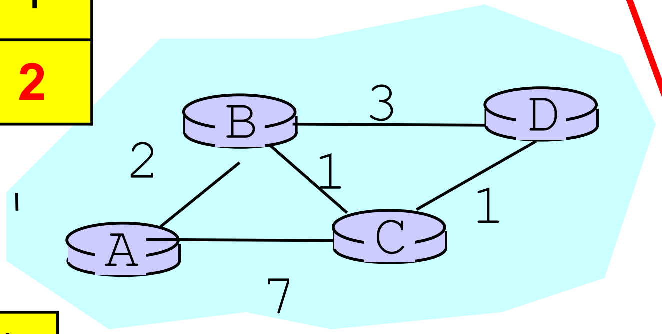
Make sure you believe this

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

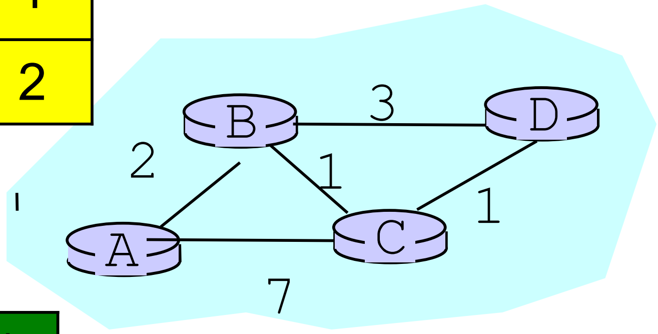
Example: Now A sends update to B

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

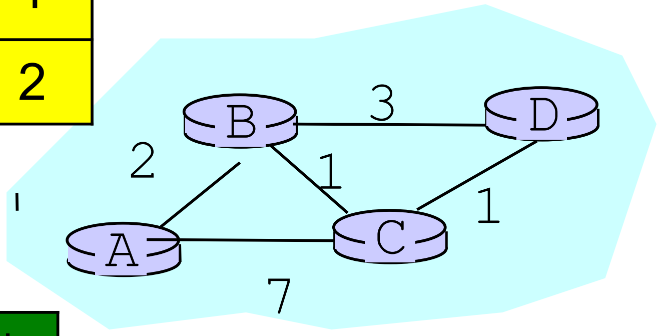
Updated

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

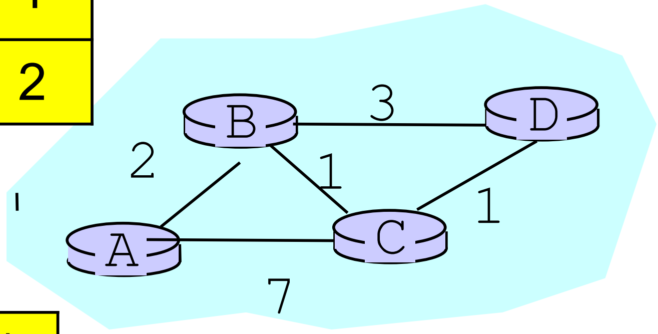
Check: *All nodes know the best three-hop paths.*

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 3rd Full Exchange

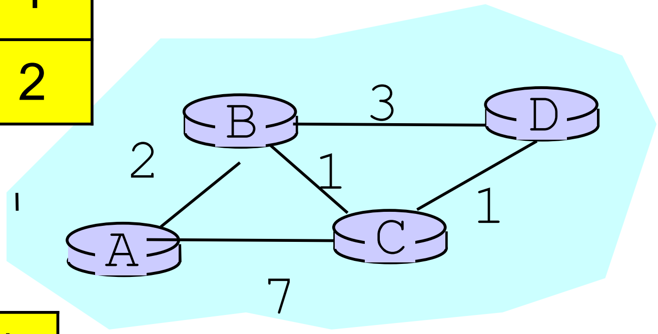
No further change in DVs → Convergence!

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

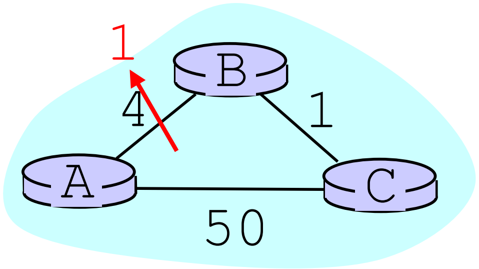
Intuition

- 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?

The key here is that the starting point is not the initialization, but some other set of entries. Convergence could be different!

DV: Link Cost Changes



Stable state

via

to

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

	B	C
B	1	51
C	2	5

	A	C
A	1	6
C	6	1

	A	B
A	50	5
B	54	1

A sends its DV to B, C

	B	C
B	1	51

C	3	1

	A	B
A	50	5
B	51	1

B sends its DV to A, C

	B	C
B	1	51

C	3	1

	A	B
A	50	2
B	51	1

C sends its DV to A, B

	B	C
B	1	51
		50

		C
		3
C	3	1

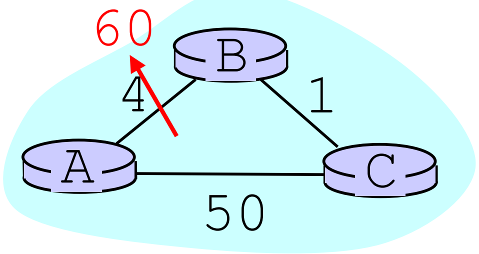
	A	B
A	50	2
B	51	1

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

via

to

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

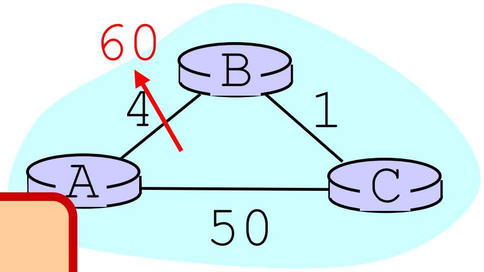
Node C

	A	B
A	50	5
B	54	1

add 56 to distances $\text{dist}_B(A,*)$ and $\text{dist}_A(B,*)$

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

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

C sends its DV to A, B

	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

A sends its DV to B, C

	A	C
A	60	6
C	110	1

B sends its DV to A, C

	A	C
A	60	6
C	110	1

C sends its DV to A, B

	A	C
A	60	8
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	101	1

B sends its DV to A, C

	A	B
A	50	7
B	101	1

C sends its DV to A, B

	A	B
A	50	7
B	101	1

Link cost changes here

"bad news travels slowly"
(not yet converged)

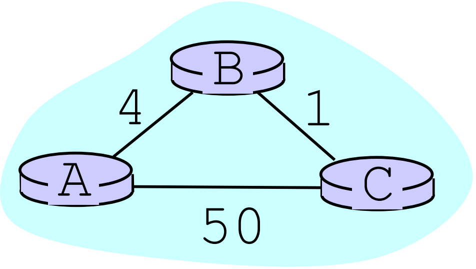
The “Poisoned Reverse” Rule

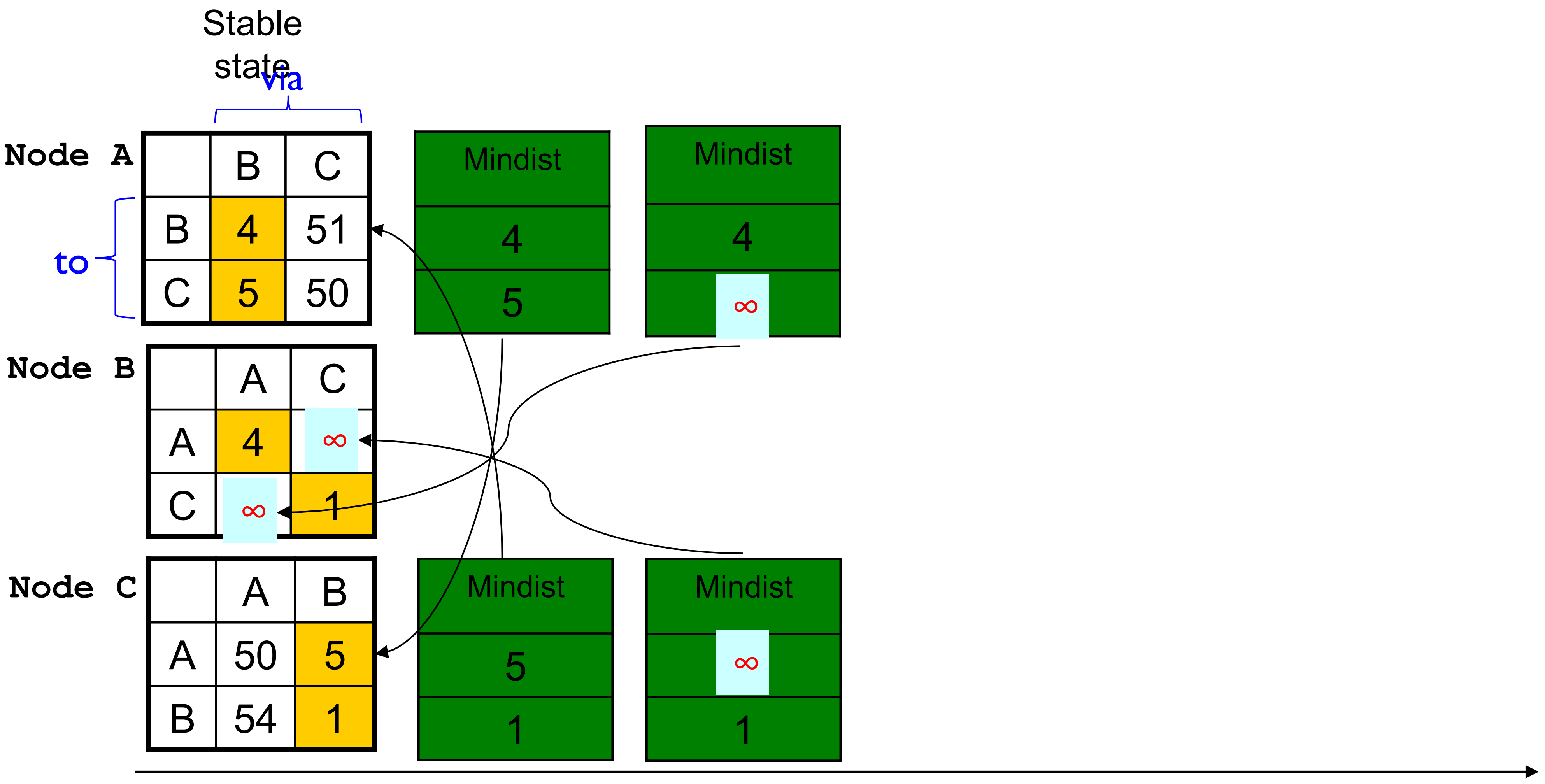
- Heuristic to avoid count-to-infinity

- 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





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
4
5

Mindist
4
∞

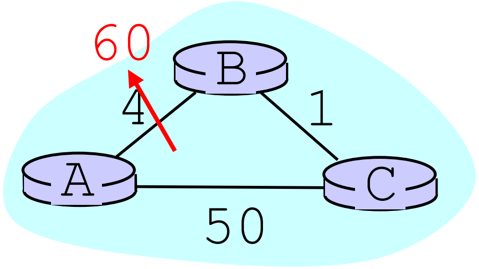
Mindist
5
1

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

to

Node A

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	∞
C	∞	1

Node C

Stable state:

	A	B
A	50	5
B	54	1

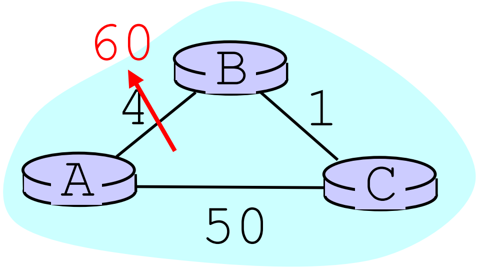
A-B changed:

	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



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	∞
C	61	50

	B	C
B	60	∞
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

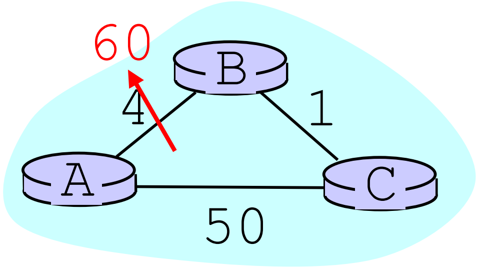
	A	B
A	50	∞
B	54	1

	A	B
A	50	5
B	∞ l	1

	A	B
A	50	5
B	∞ l	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	∞
C	61	50

	B	C
B	60	∞
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	∞
B	54	1

	A	B
A	50	5
B	∞ I	1

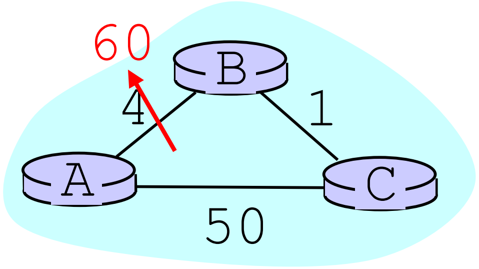
	A	B
A	50	61
B	∞ I	1

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

via

to

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	∞
B	54	1

A sends its DV to B, C

Node A

	B	C
B	60	∞
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	∞
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

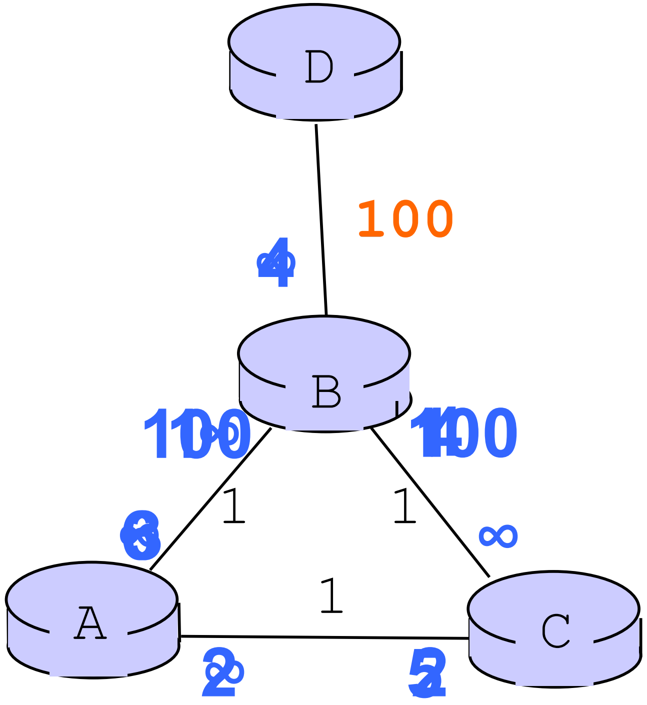
Node C

	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:*** 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)

Quiz: Impact of link cost

A problem that can arise when a link state algorithm is used in a network where link costs equal to load is:

a) Count to infinity

b) Poisoned reverse

c) The infinite reverse

d) oscillation

Quiz: Impact of link cost

When compensating for link cost changes in the distance vector algorithm, it can be generally said that:

a) Increased costs are propagate quickly, i.e., “bad news” travels fast

b) Decreased costs are propagated rapidly, i.e. “good news” travels fast

c) Increased costs do not converge

d) Decreased costs propagate slowly, i.e., “good news” travels slowly.

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

Hierarchical routing

our routing study thus far - idealization

- all routers identical
- network "flat"

… *not* true in practice

scale: with billions destinations:

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

administrative autonomy

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

Internet approach to scalable routing

aggregate routers into regions known as "autonomous systems" (AS) (a.k.a. "domains")

intra-AS routing

- routing among hosts, routers in same AS ("network")
- all routers in AS must run same intra-domain protocol
- routers in different AS can run different intra-domain routing protocol
- gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

inter-AS routing

- routing among AS'es
- gateways perform inter-domain routing (as well as intra-domain routing)

Autonomous Systems (AS)

- AS is a network under a single administrative control
  - currently over 30,000 ASes
  - Think AT&T, France Telecom, UNSW, IBM, *etc.*

- ASes are sometimes called “domains”.
  - Hence, “interdomain routing”

- Each AS is assigned a unique identifier
  - 16 bit AS Number (ASN)

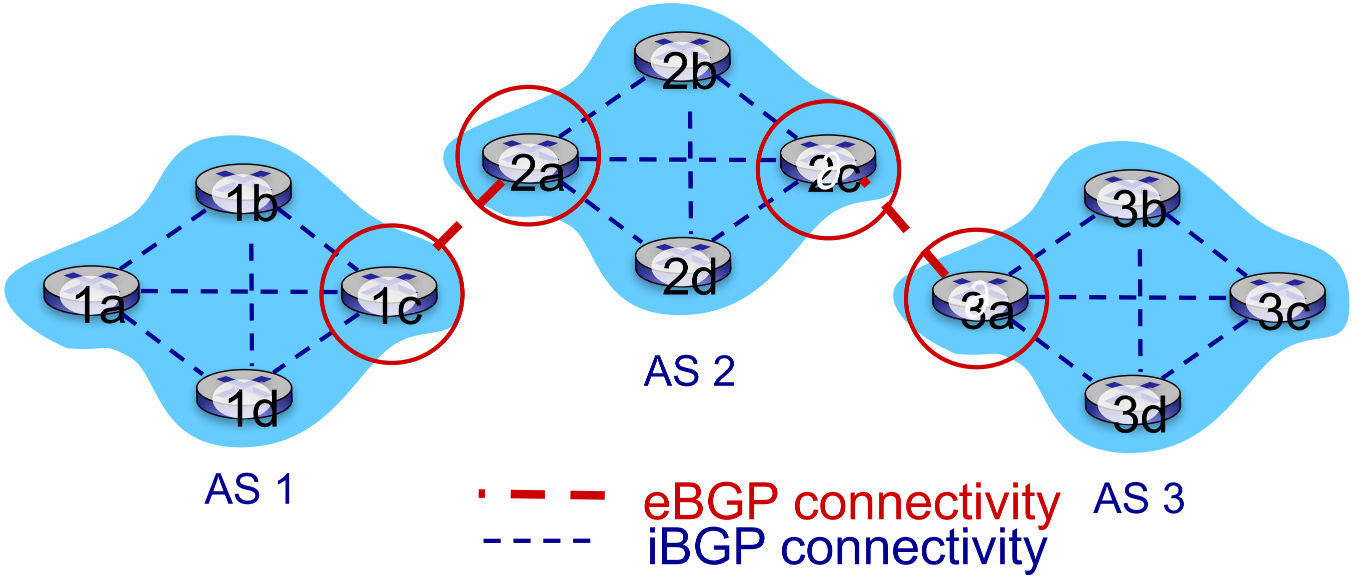
Internet inter-AS routing: BGP

- BGP (Border Gateway Protocol): *the* de facto inter-domain routing protocol
  - "glue that holds the Internet together"
- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from neighboring ASes
  - iBGP: propagate reachability information to all AS-internal routers.
  - determine "good" routes to other networks based on reachability information and *policy*
- allows subnet to advertise its existence to rest of Internet: *"I am here"*

Border Gateway Protocol(BGP)

- BGP-- standard exterior routing protocol in the Internet
- Neither a pure distance vector protocol nor a pure link state protocol
- When a pair of AS's agree to exchange routing information, each must designate a router that will speak BGP on its behalf. These two routers are said to become BGP peers of one another
- They are normally near the edge of the AS(hence called Border Router)
- Each AS can have more than one BGP router

eBGP, iBGP connections

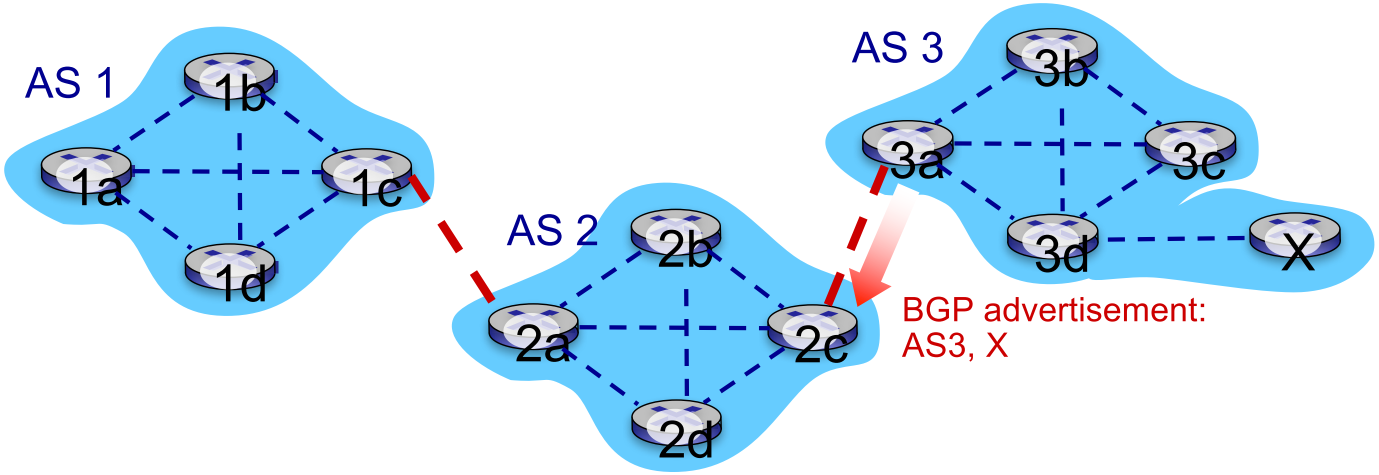


gateway routers run both eBGP and iBGP protools

BGP basics

- BGP session: two BGP routers ("peers") exchange BGP messages over TCP connection:
  - advertising paths to different destination network prefixes (BGP is a "path vector" protocol)

- when AS3 gateway router 3a advertises path AS3,X to AS2 gateway router 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X

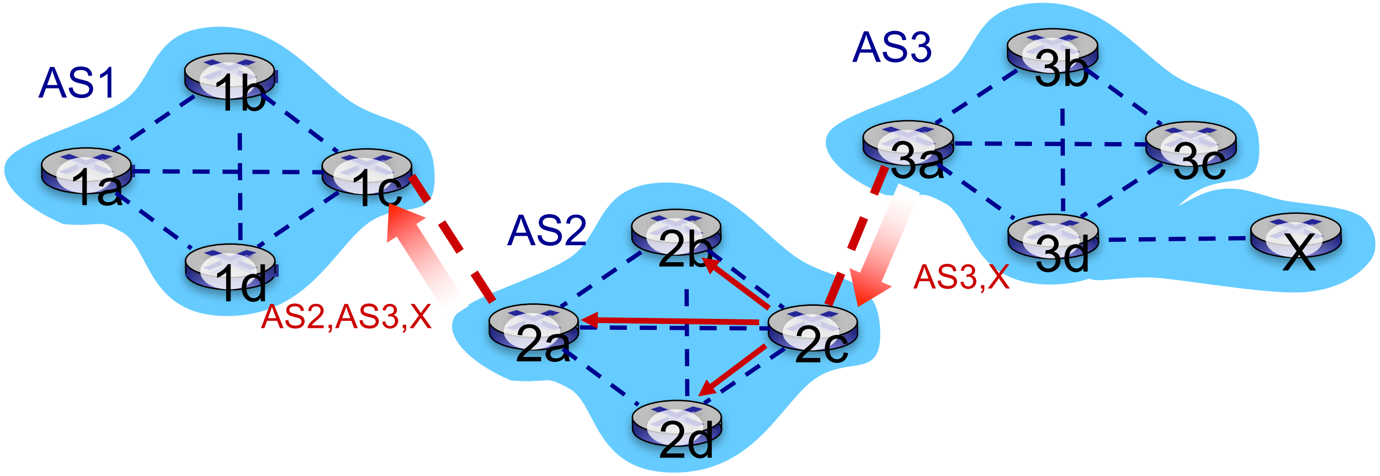


BGP advertisement:
AS3, X

Path attributes and BGP routes

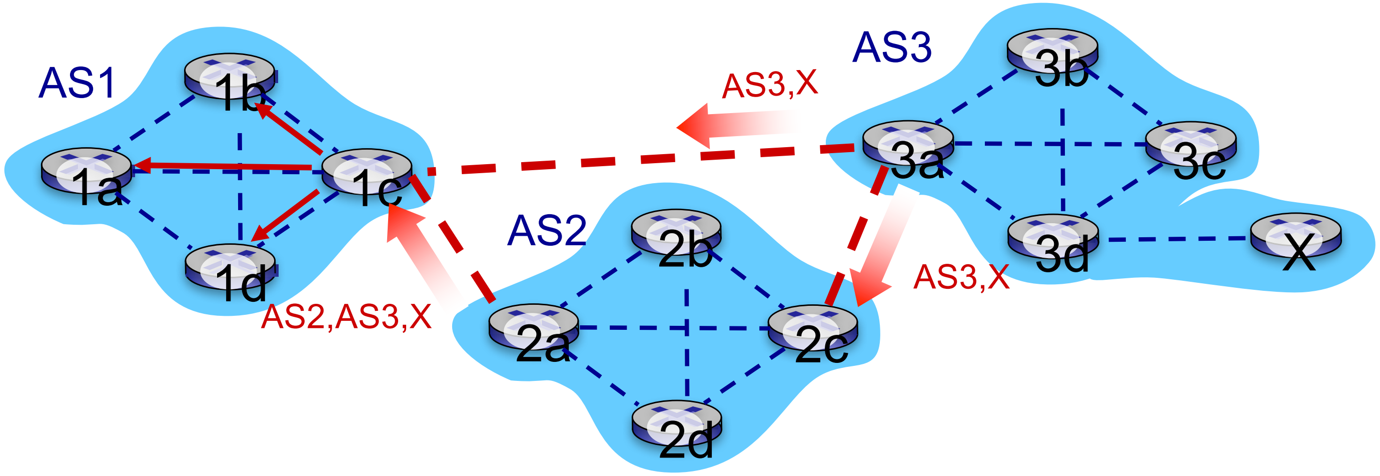
- advertised prefix includes BGP attributes
  - prefix + attributes = "route"
- two important attributes:
  - AS-PATH: list of ASes through which prefix advertisement has passed
  - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- *Policy-based routing:*
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes

BGP path advertisement



- AS2 router 2c receives path advertisement AS3,X (via eBGP) from AS3 router 3a
- Based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- Based on AS2 policy, AS2 router 2a advertises (via eBGP) path AS2, AS3, X to AS1 router 1c

BGP path advertisement



gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path AS3,X from 3a
- Based on policy, AS1 gateway router 1c chooses path AS3,X, and advertises path within AS1 via iBGP

BGP : Dissection

- BGP does not communicate or interpret distance metrics, even if metrics are available
  - BGP speaker can declare that a destination has become unreachable or give a list of AS's on the path to the destination
  - It cannot transmit or compare the cost of two routes unless routes come from the same AS
- If a router learns about two paths to the same network, it cannot know which path is shorter because it cannot know the cost of routes across intermediate AS
- BGP is thus a reachability protocol rather than routing protocol

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