COMP 3331/9331: Computer Networks and Applications

Week 9

Network Layer: Routing

Reading Guide: Chapter 4: Sections 4.5

Announcements

- Labs
  - Lab 4 – Congestion Control
  - Lab 5 – Simple Router (start up for Assignment 2, not marked)
  - Lab 6 – Security
- Mininet Help Session
  - Monday and Tuesday, 10:30 – 11:30, Level 5 Consultation Room, CSE Building (K17)

Network Layer: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

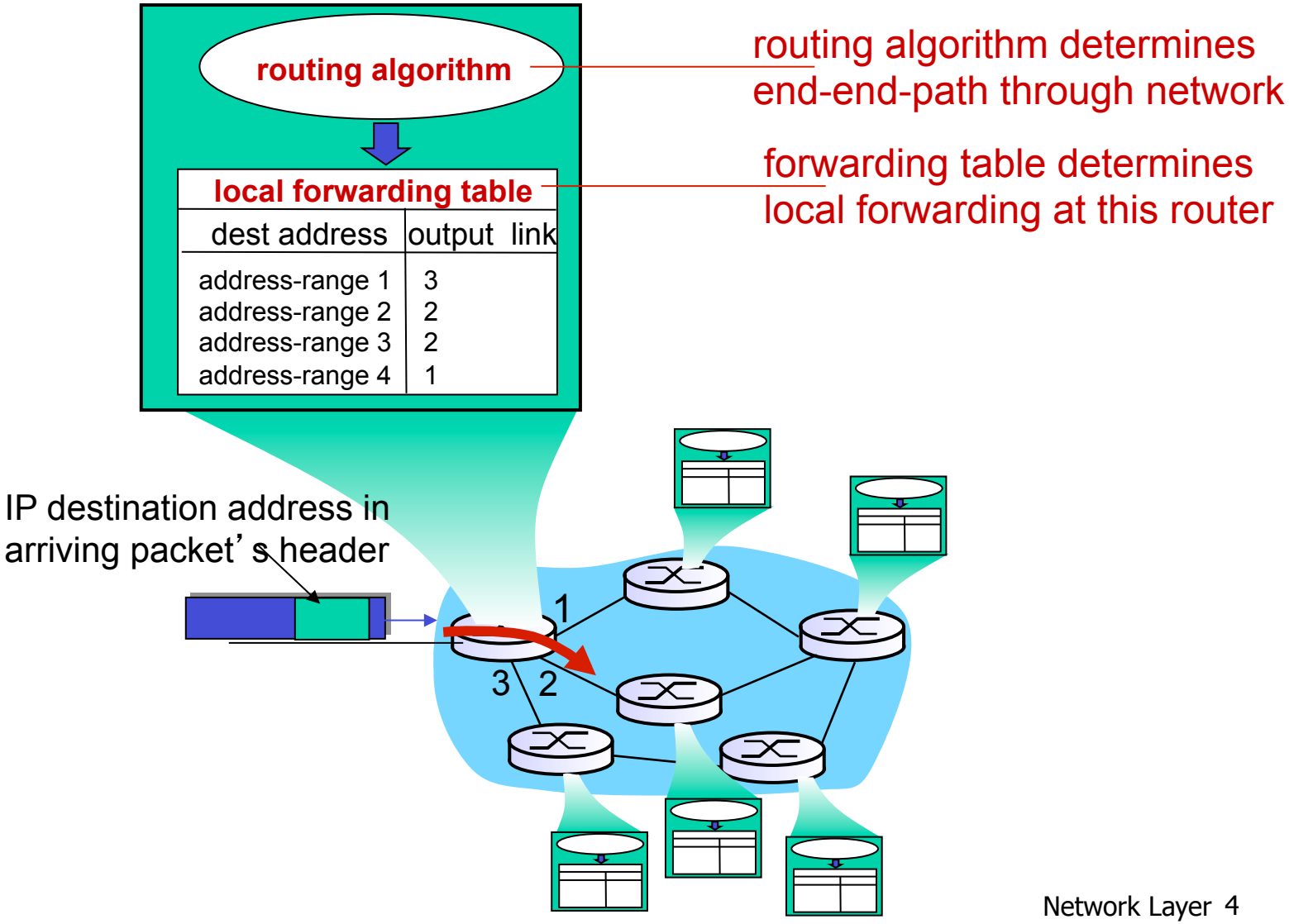
- link state
- distance vector
- hierarchical routing

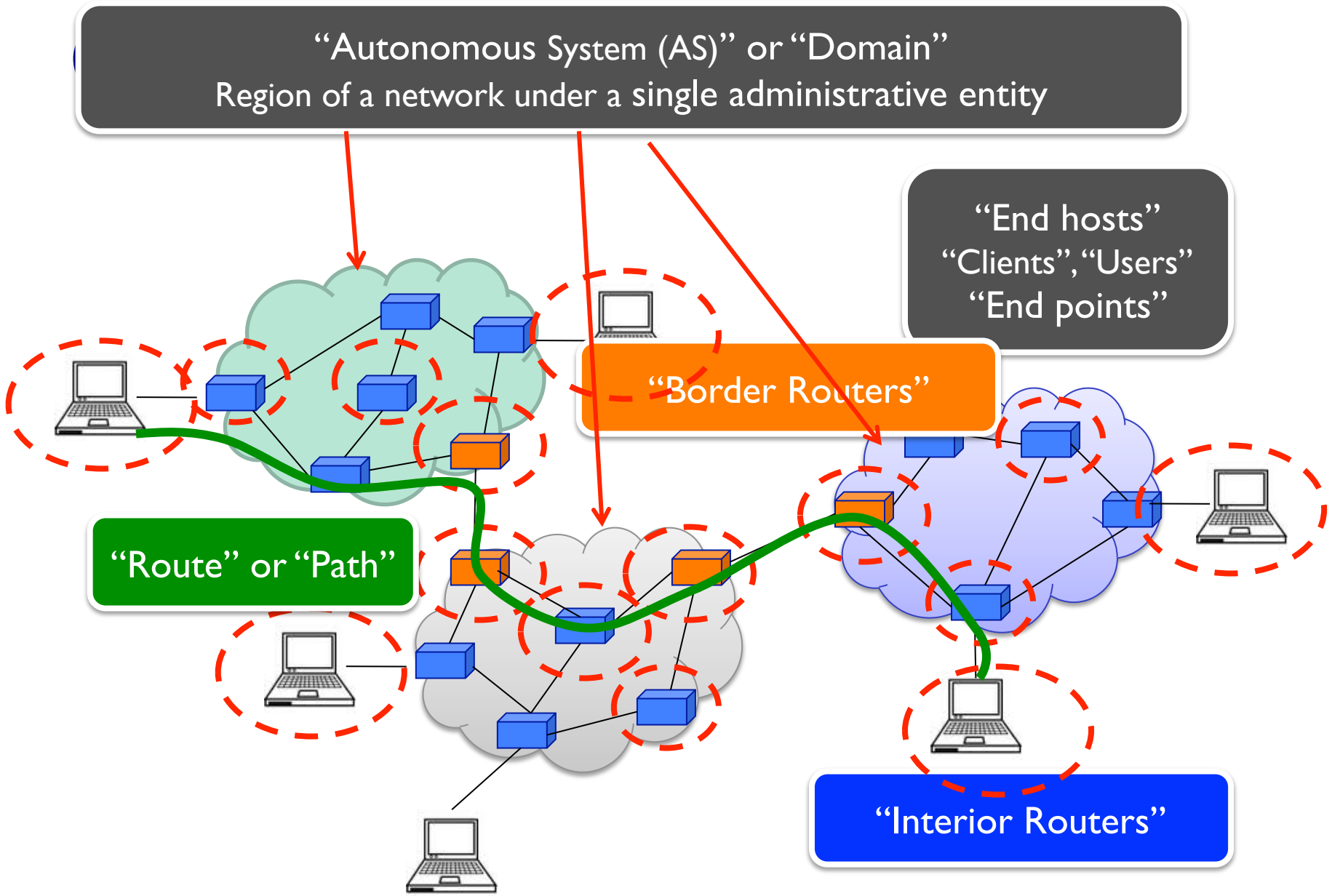
4.6 et

4.7 ast

r

Interplay between routing, forwarding

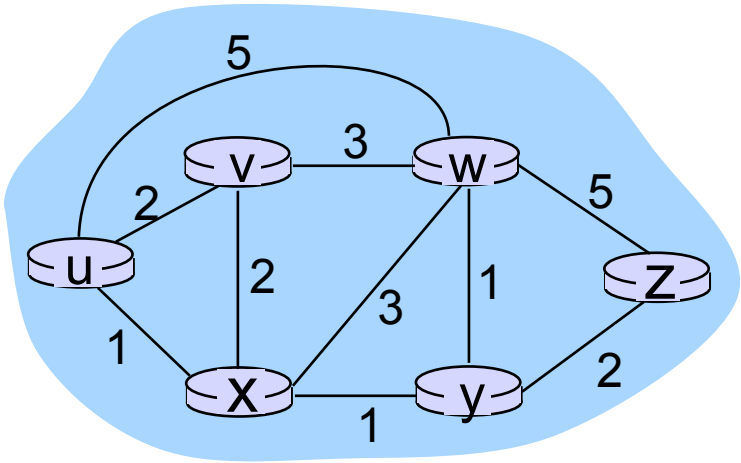



"Autonomous System (AS)" or "Domain"
Region of a network under a single administrative entity
"End hosts"
"Clients", "Users"
"End points"
"Border Routers"
"Route" or "Path"
"Interior Routers"

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 }

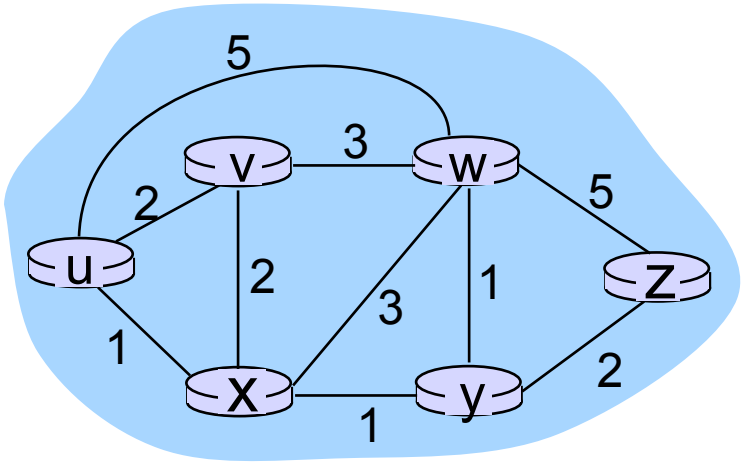
E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

aside: graph abstraction is useful in other network contexts, e.g., P2P, where *N* is set of peers and *E* is set of TCP connections

Addressing (abstraction)

- Assume each host has a unique ID (address)
- No particular structure to those IDs
- Earlier in course we talked about real IP addressing

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

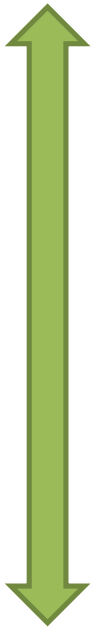
Quiz: How much information should a router know about the network?

A: the next hop and cost of forwarding to its neighbour(s)

B: the next hop and cost of forwarding to any destination

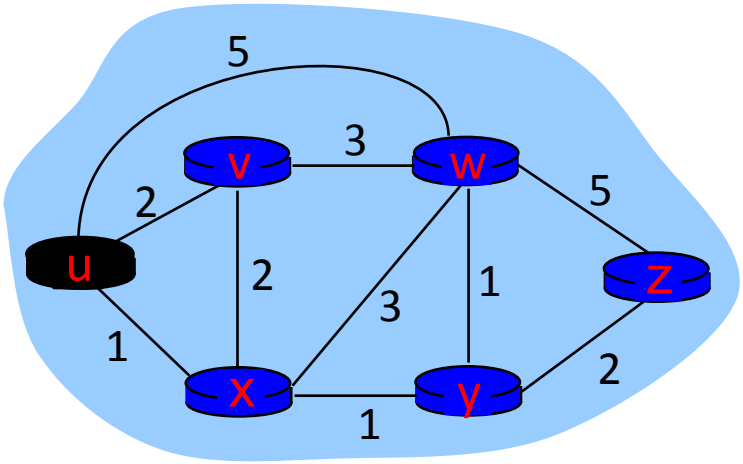
C: the status and cost of every link in the network.

D: some other amount of information



Routing Table

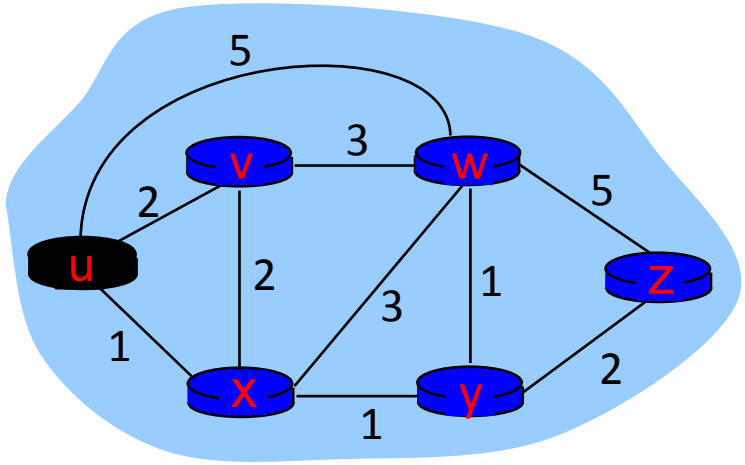
Dest	Next Hop
V	V
X	X
W	X
Y	X
Z	X



- At a **minimum**, the routing table at U needs to know the next hop for each possible destination

Routing Table

Dest	Next Hop	Cost (Path)
V	V	2
X	X	1
W	X	4
Y	X	2
Z	X	4



- At a **minimum**, the routing table at U needs to know the next hop for each possible destination
- Probably want more info (e.g. path cost, may be path itself)
- This is a key difference between routing & forwarding

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: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

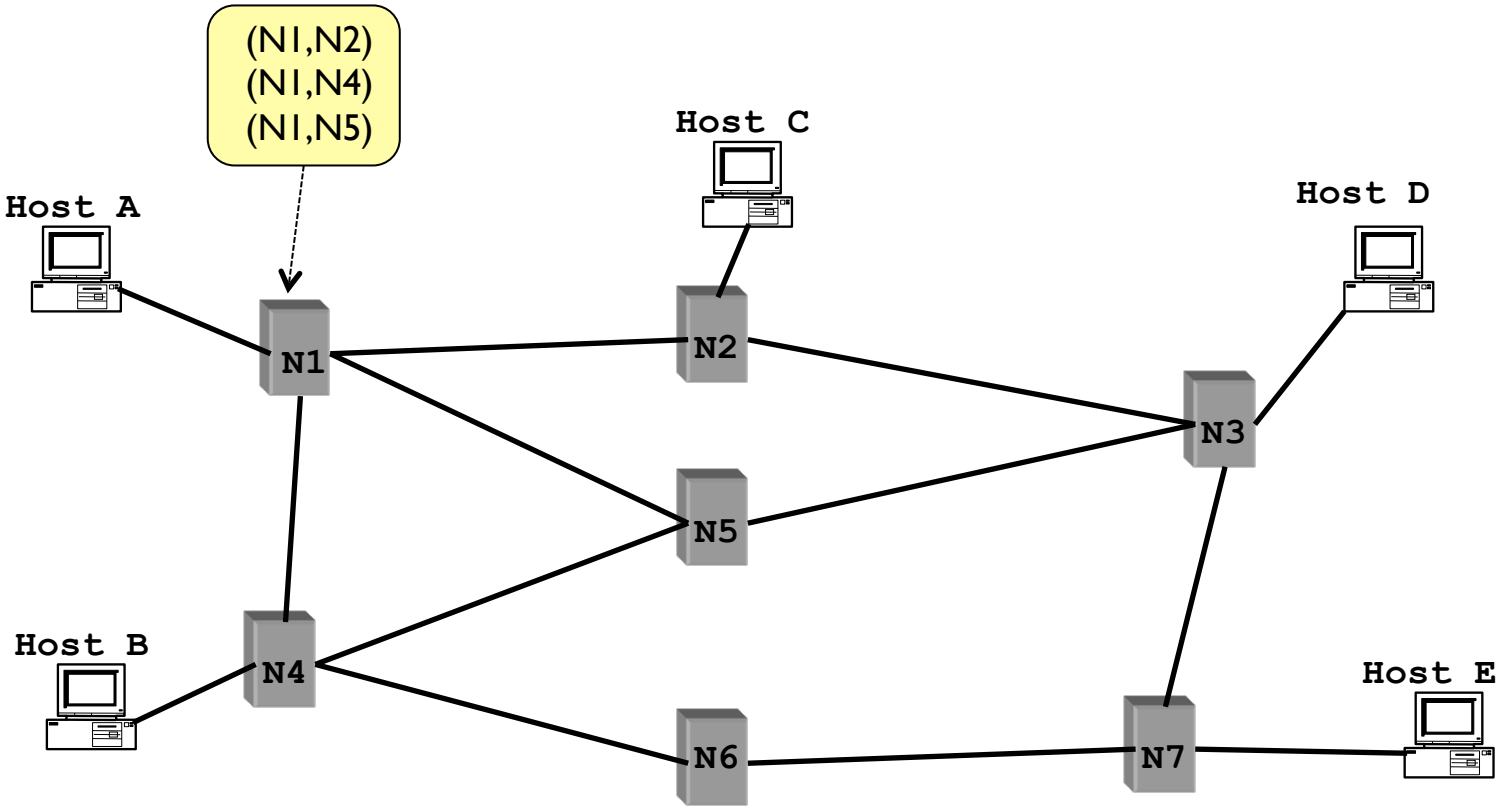
4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

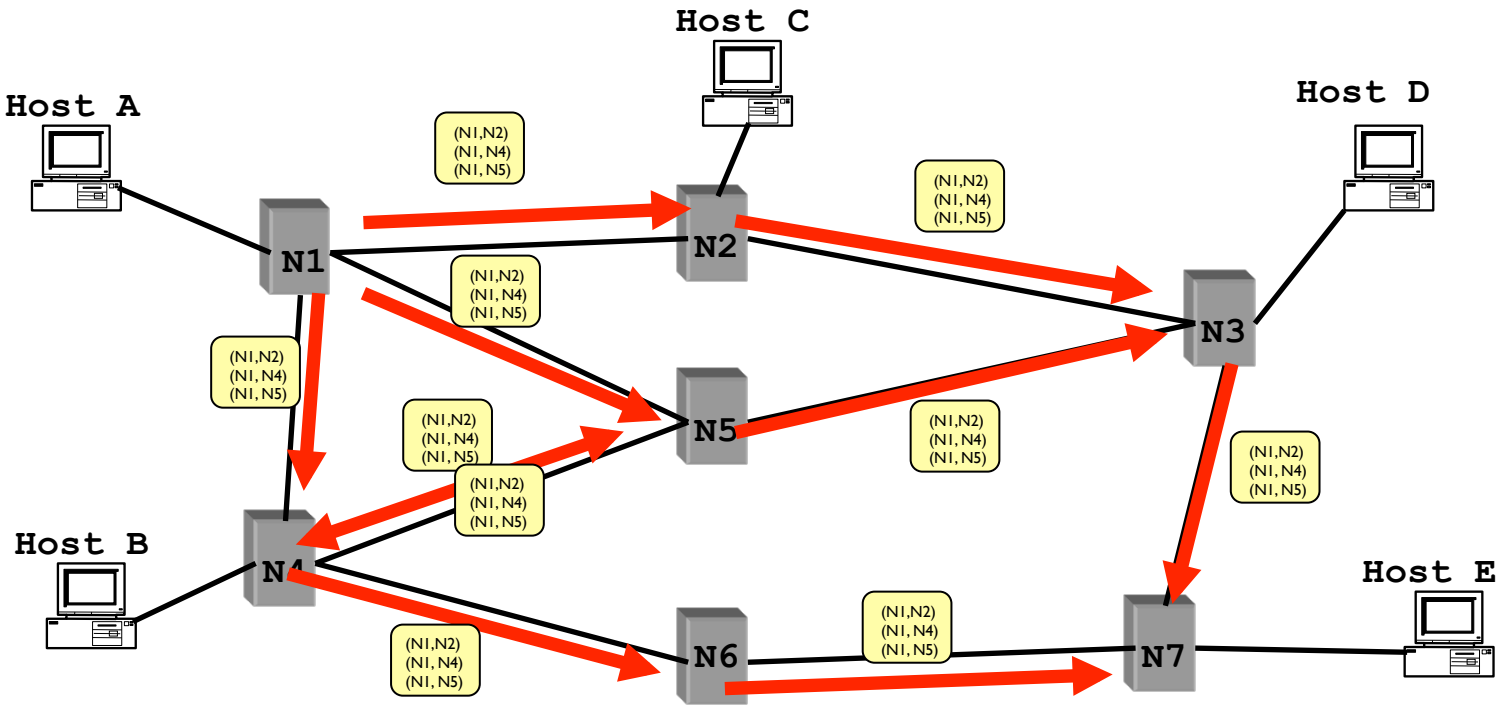
Link State Routing

- 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

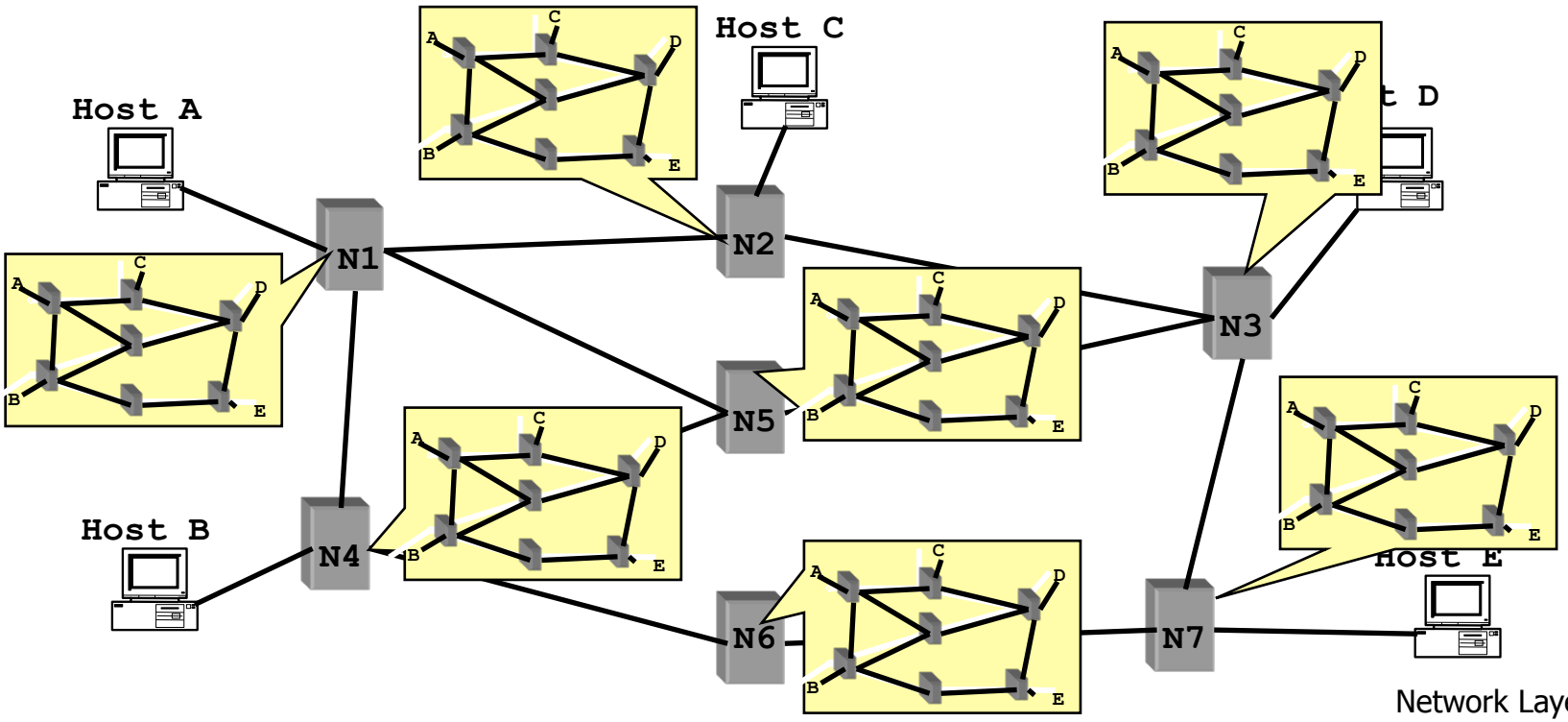


Flooding LSAs

- 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
- Hence, 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:

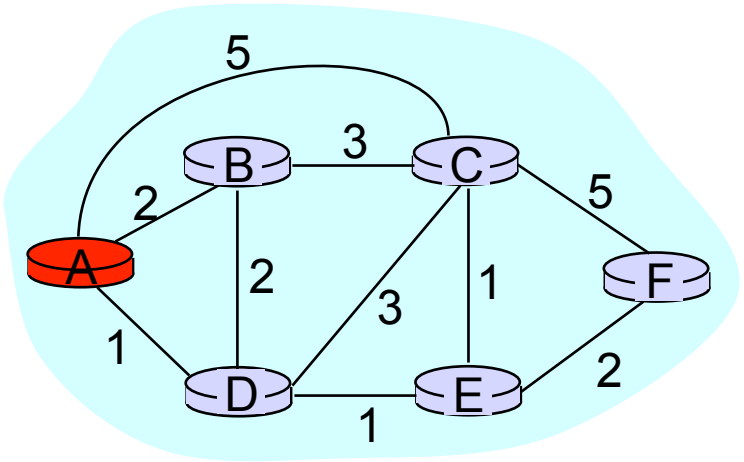
- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

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

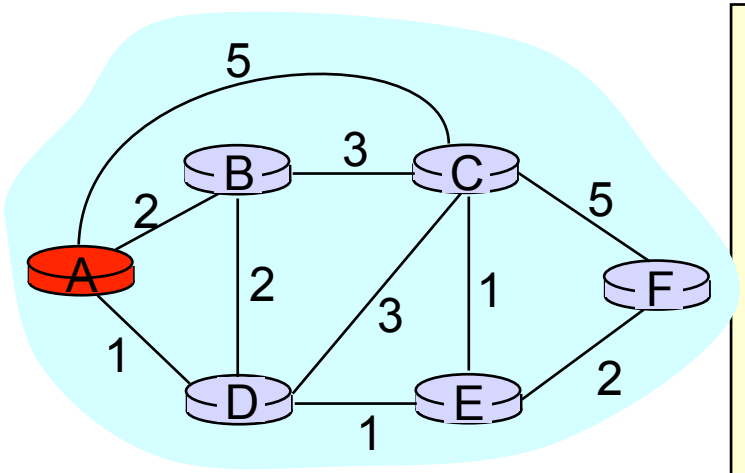
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
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

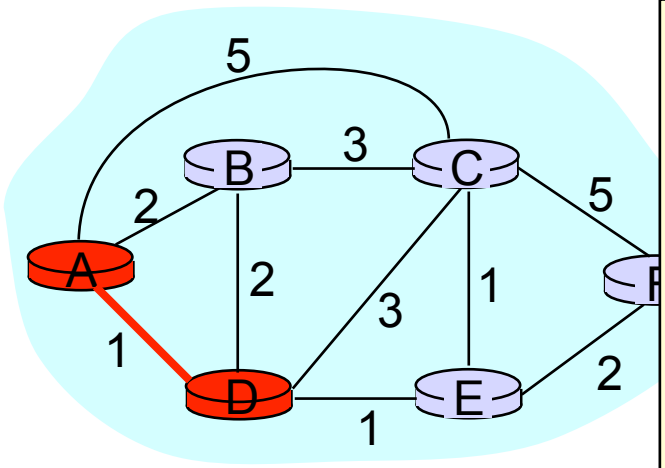
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
…
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

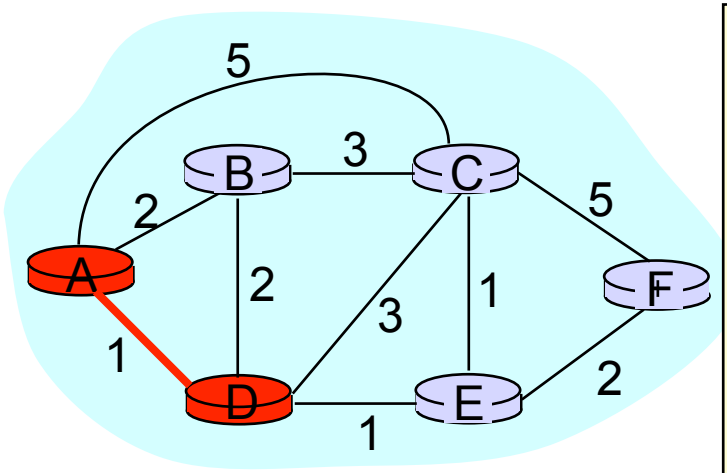
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD					
2						
3						
4						
5						




```
...
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

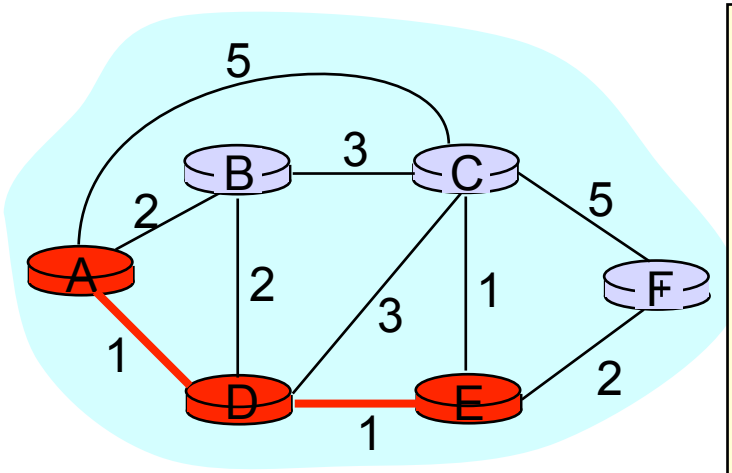
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2						
3						
4						
5						




```
…
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

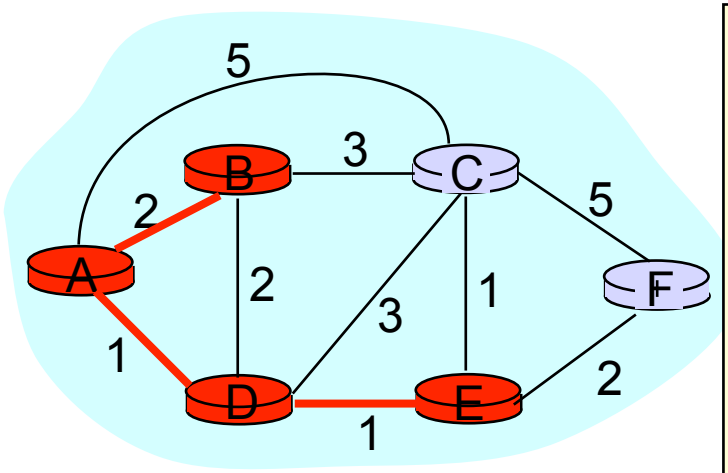
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3						
4						
5						




```
…
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

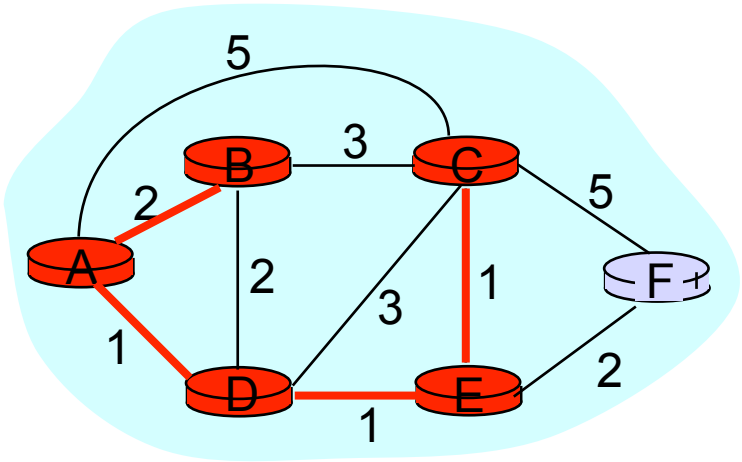
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4						
5						




```
...
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

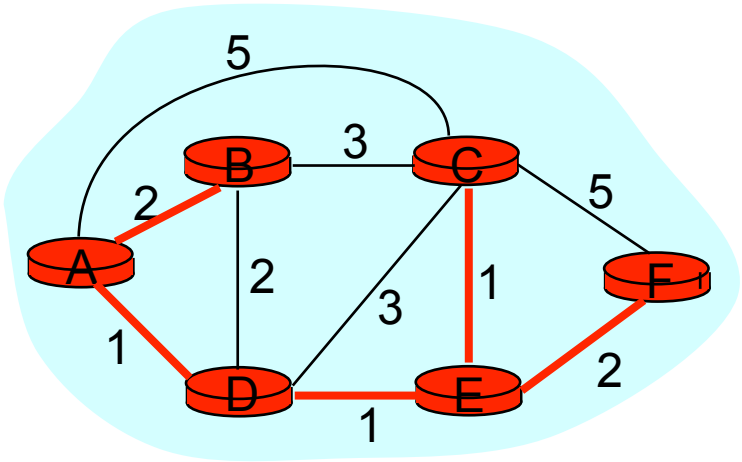
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5						




```
…
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

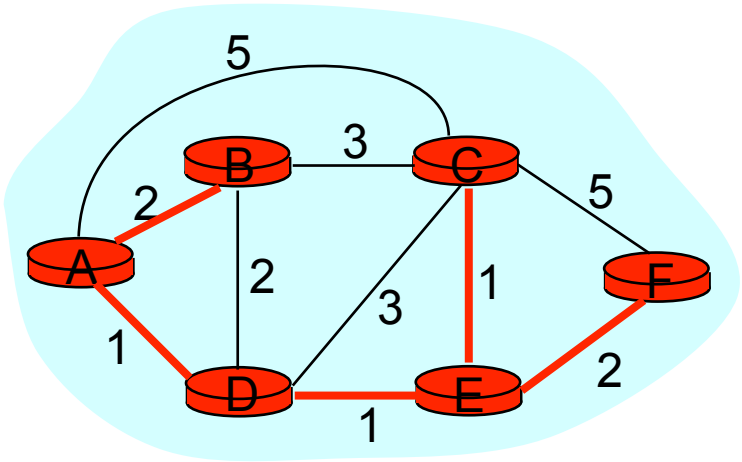
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					




```
...
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					

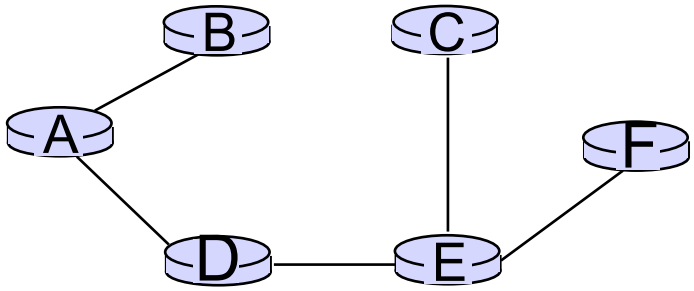


To determine path A → C (say), work backward from C via p(v)

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:



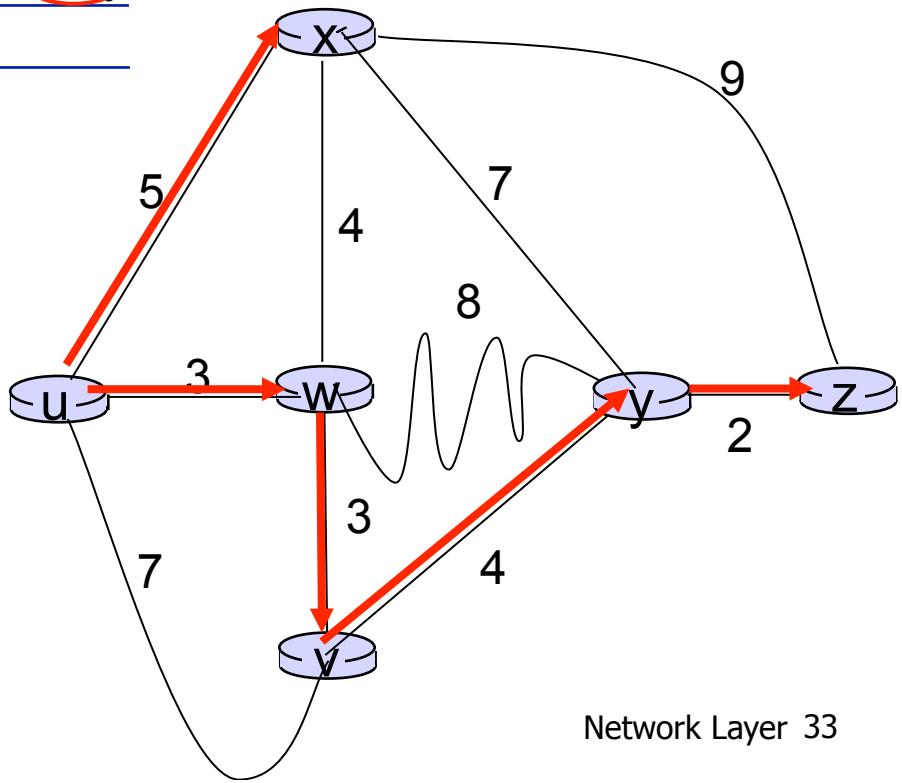
Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

Dijkstra's algorithm: example

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

notes:

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



Practice Problem

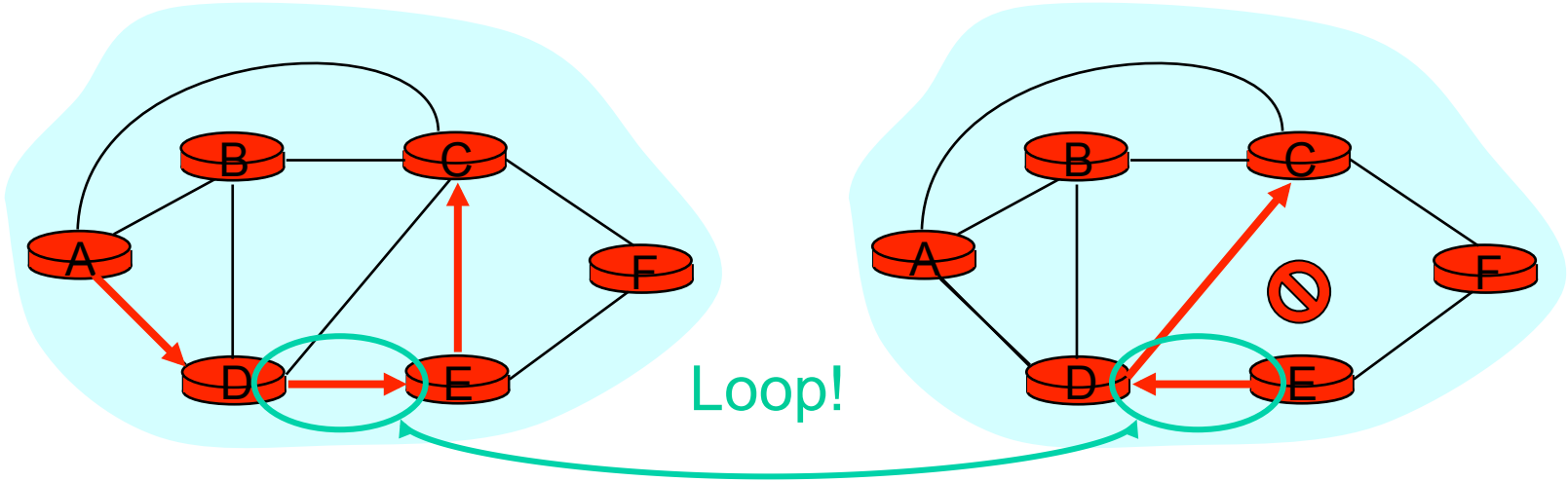
http://gaia.cs.umass.edu/kurose_ross/interactive/dij.php

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 S at each iteration and we have O(N) iterations
  - more efficient implementations: O(N log(N) +E) using min-heap
- 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



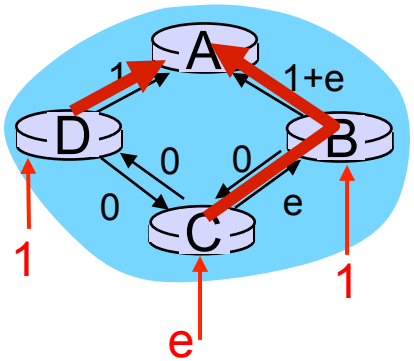
A and D think that this is the path to C

E thinks that this is the path to C

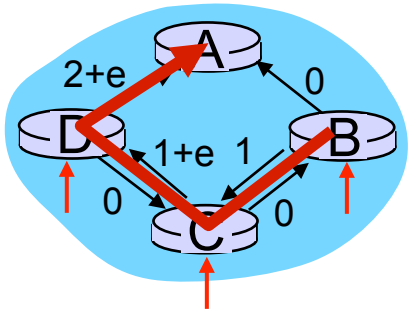
Oscillations

oscillations possible:

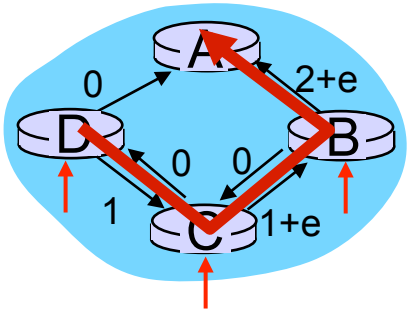
- e.g., support link cost equals amount of carried traffic:



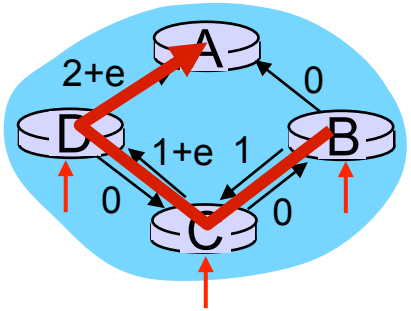
initially



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: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

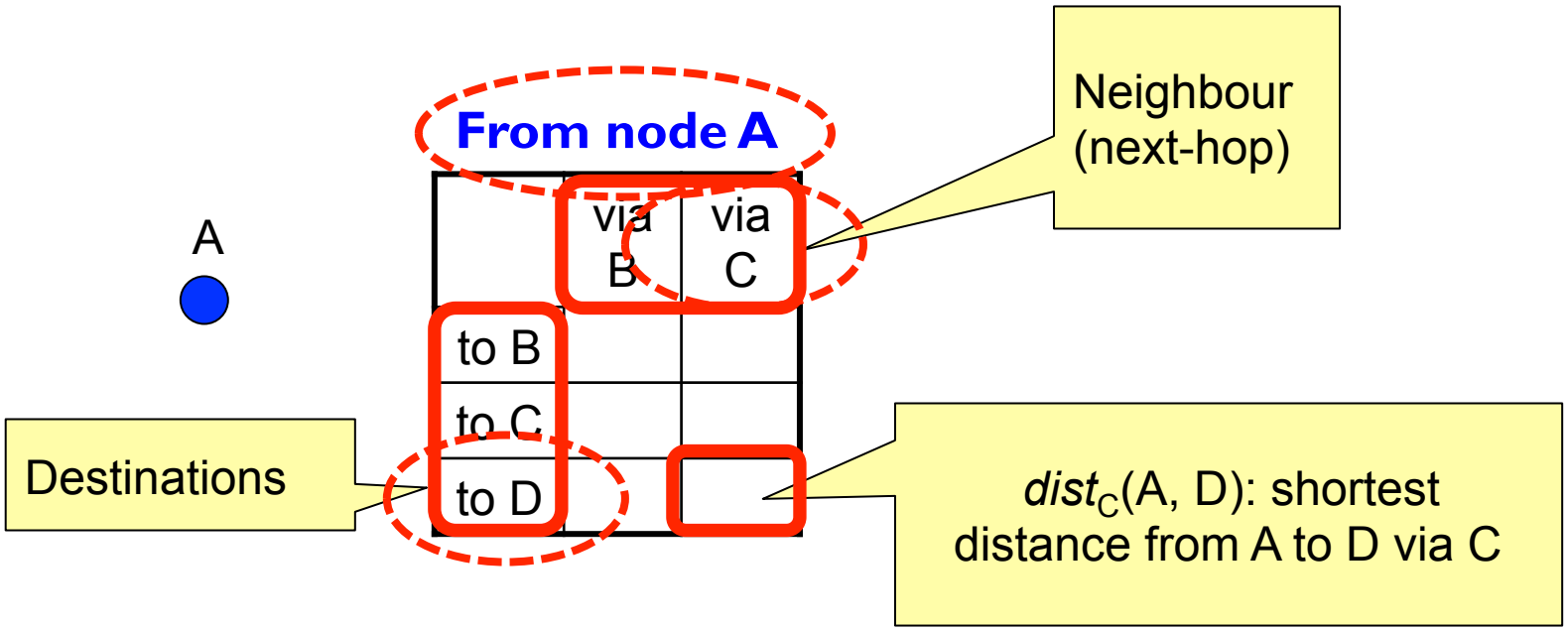
- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

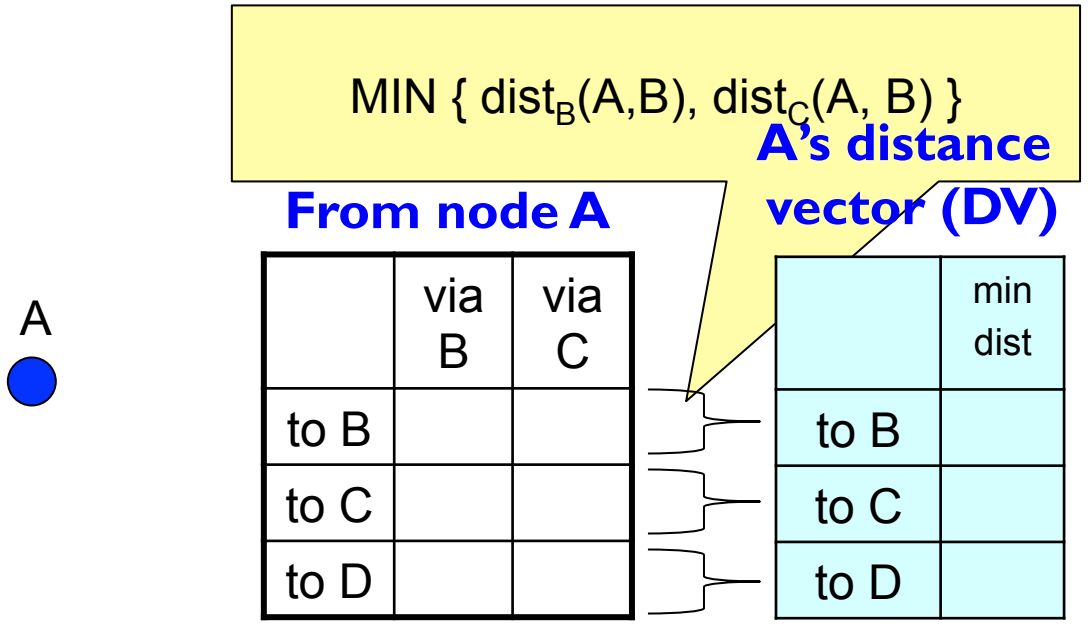
4.7 broadcast and multicast routing

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

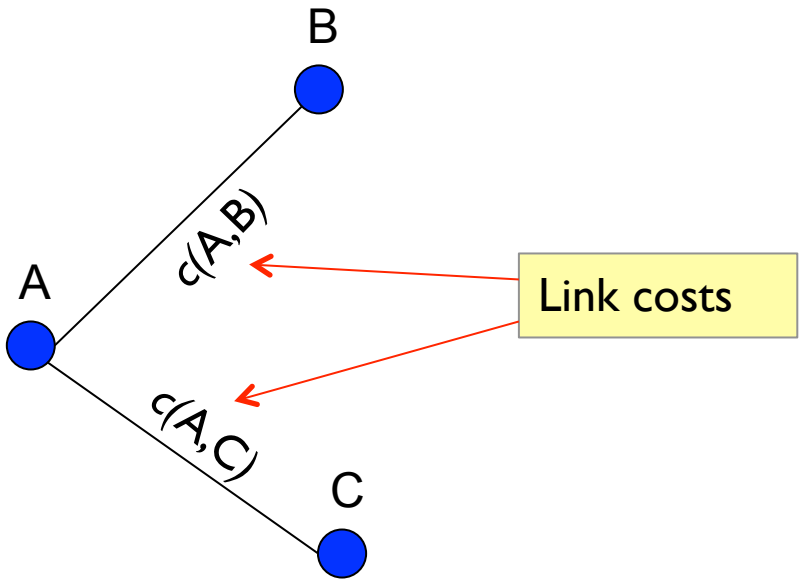
	via B	via C
to B	?	?
to C	?	?
to D	?	?

A's DV

	min dist
to B	?
to C	?
to D	?

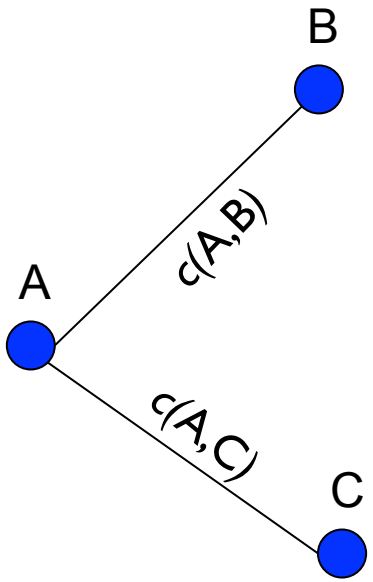
How does A initialize its dist() table and DV?

How Distance-Vector (DV) works



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

How Distance-Vector (DV) works



From node A

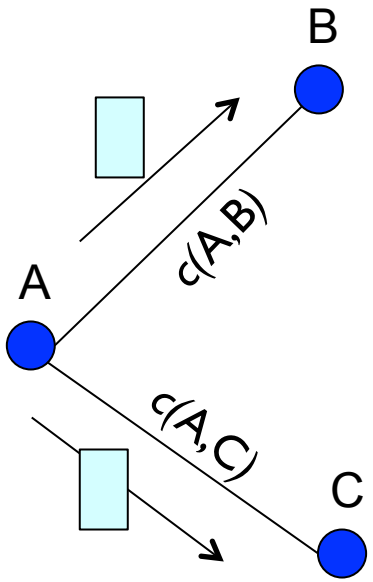
	via B	via C
to B	*c*(A,B)	∞
to C	∞	*c*(A,C)
to D	∞	∞

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



From node A

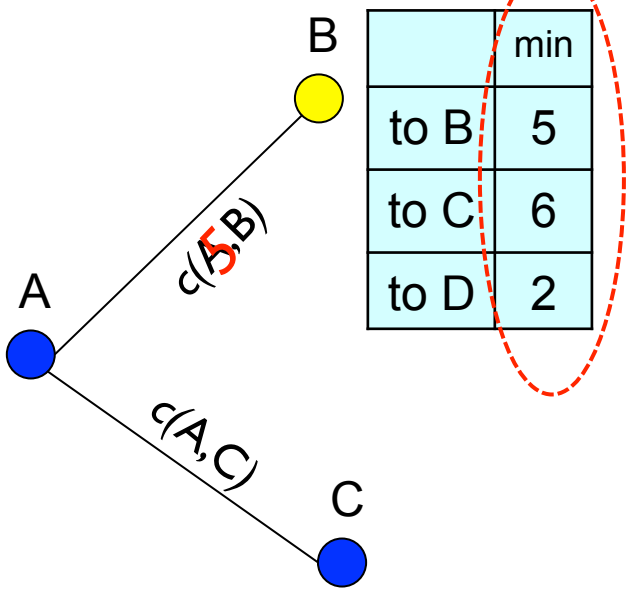
	via B	via C
to B	*c*(A,B)	∞
to C	∞	*c*(A,C)
to D	∞	∞

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

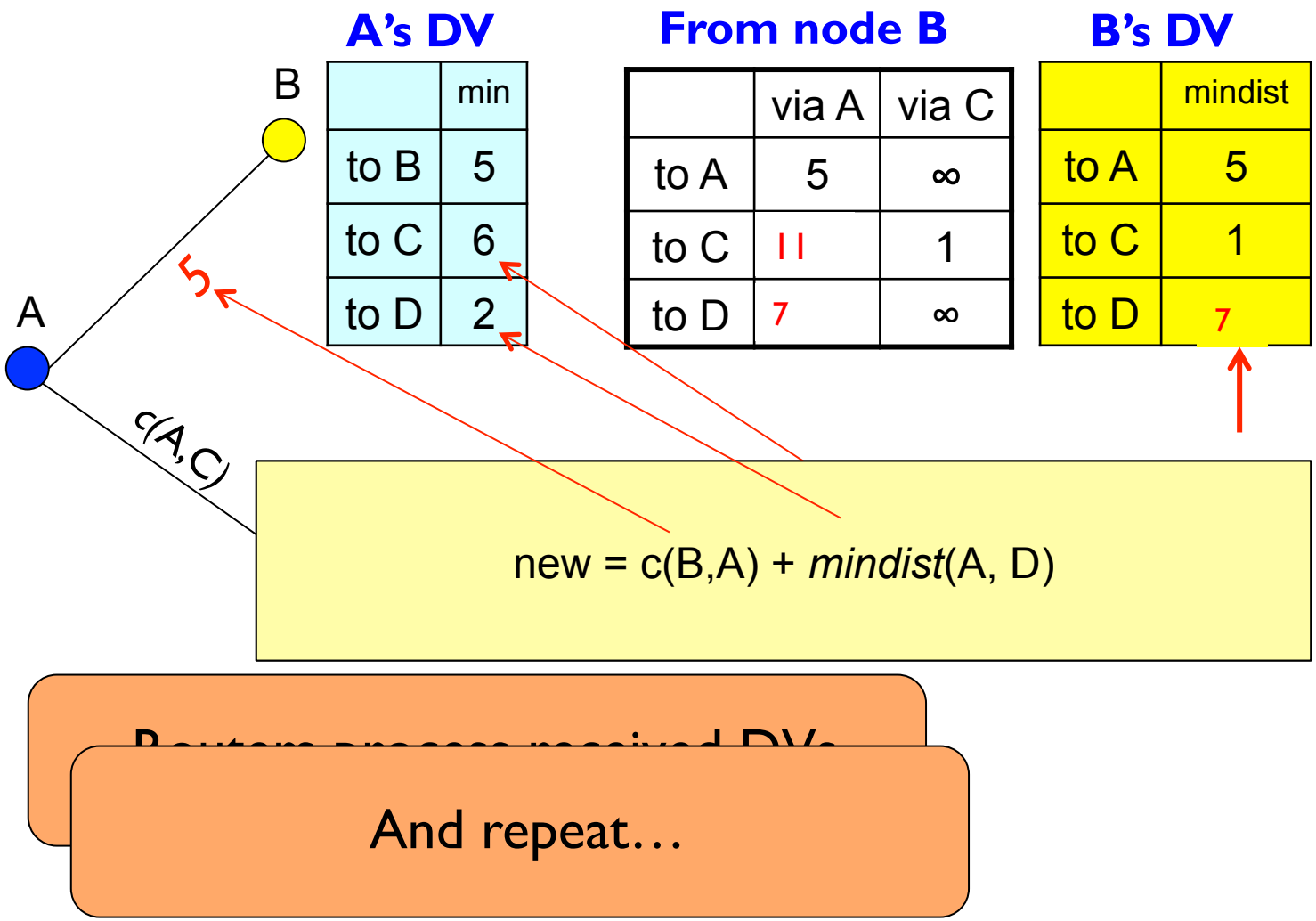
	via A	via C
to A	5	∞
to C	15	1
to D	∞	∞

B's DV

	mindist
to A	5
to C	1
to D	∞

Routers process received DVs

How Distance-Vector (DV) works



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

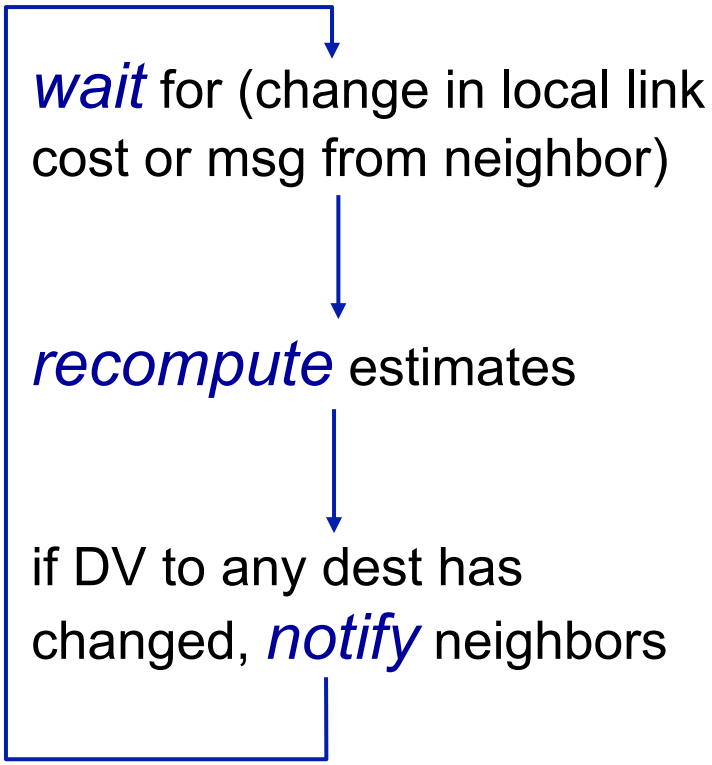
iterative, asynchronous:
each local iteration caused by:

- 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
```

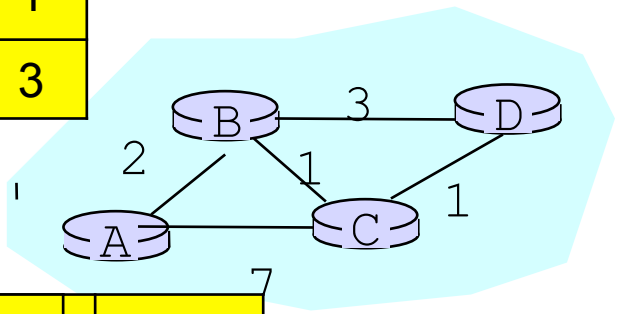
Example: Initialization

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

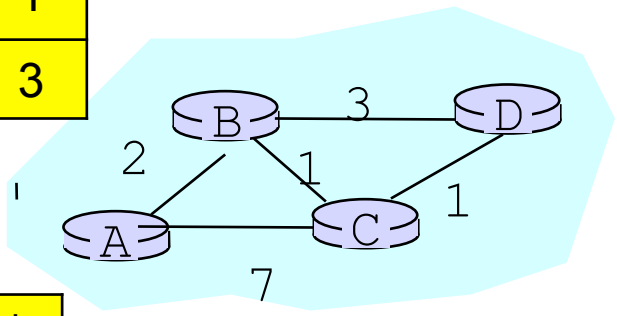
Example: C sends update to A

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



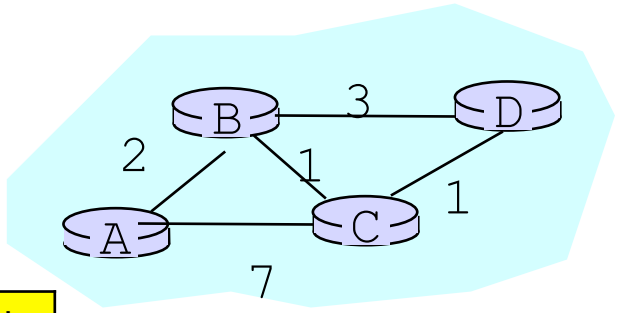
from Node A

	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

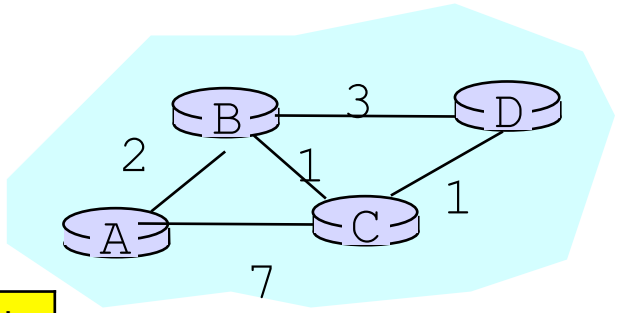


from Node A

	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

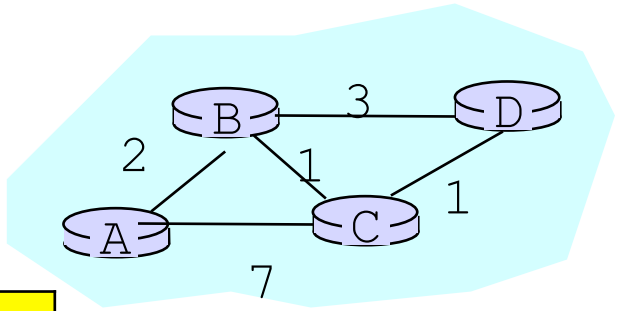


from Node A

	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



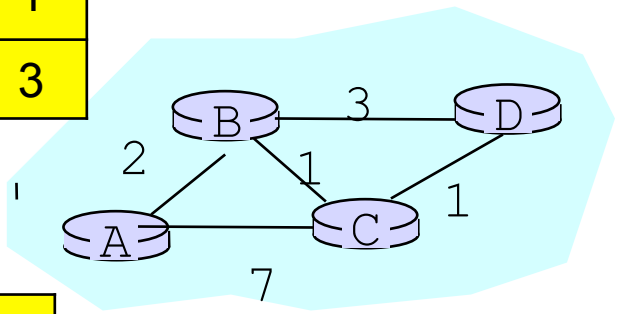
from Node A

	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

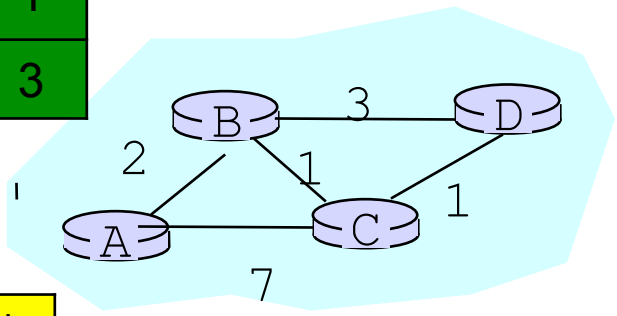
Example: now B sends update to A

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

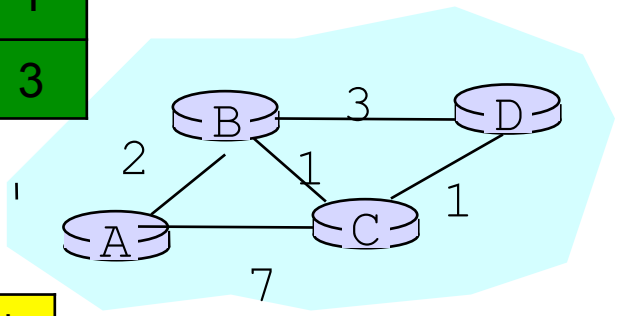
Example: now B sends update to A

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

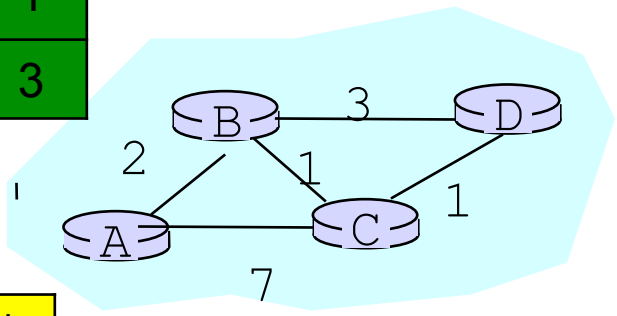
	via	via	via	min dist
				7
				1
to C	-	-	-	0
to D	∞	∞	1	1

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 knows the best two-hop paths.

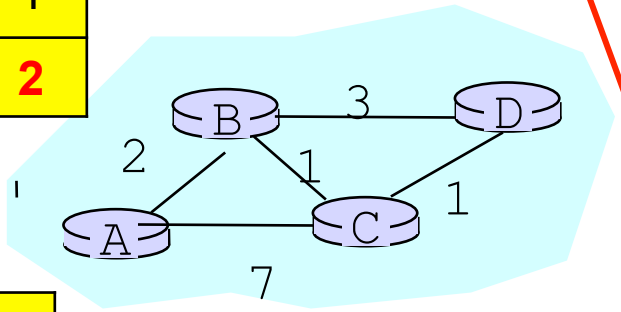
Make sure you believe this

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

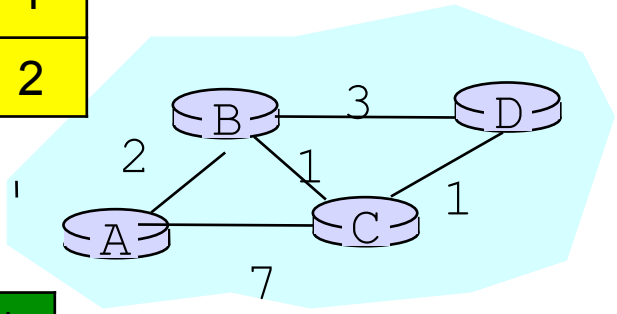
Example: Now A sends update to B

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: Nov

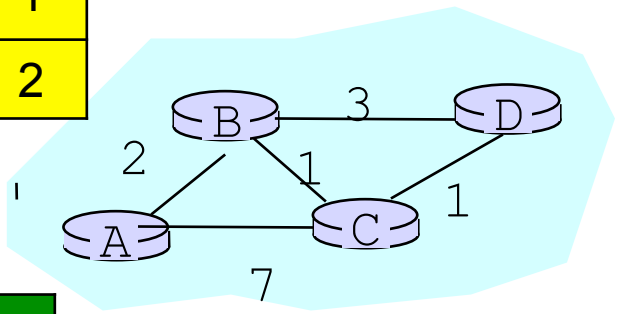
This will come back to bite us

from Node B

	via A	via C	via D	min
to A	2	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	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

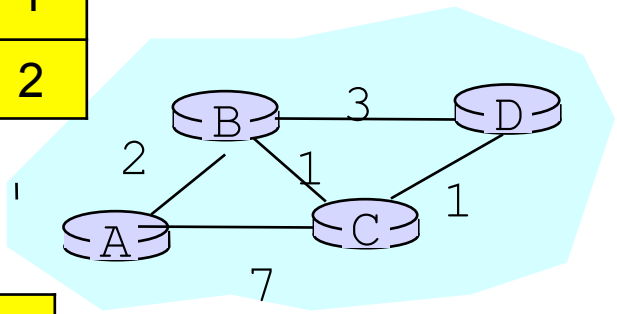
Check: *All nodes knows the best three-hop paths.*

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

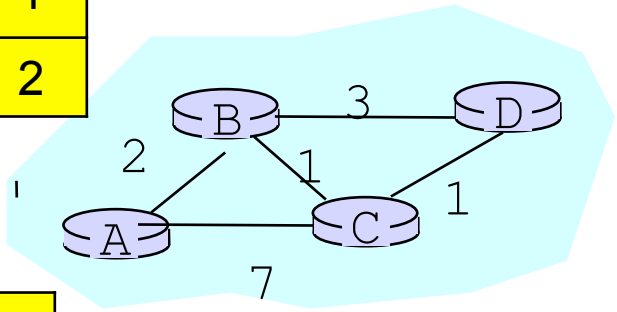
No change in DVs → Convergence!

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

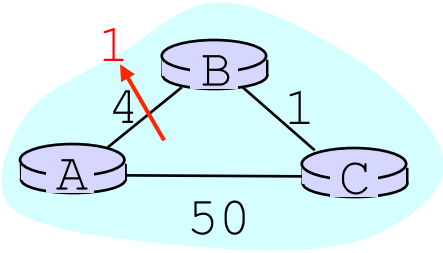
Intuition

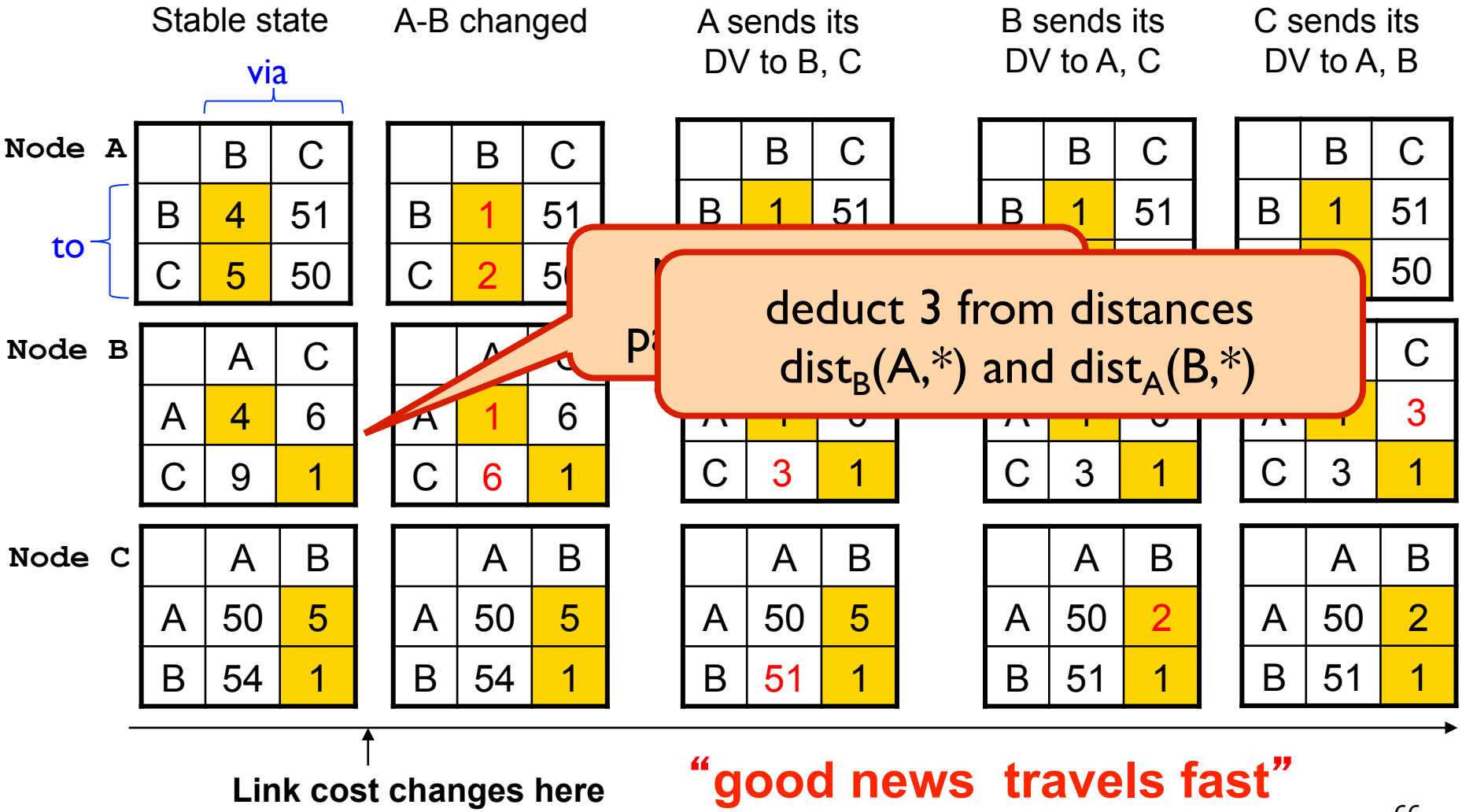
- Initial state: best one-hop paths
- One simultaneous round: best two-hop paths
- Two simultaneous rounds: best three-hop paths
- …
- Kth simultaneous round: best (k+1) hop paths

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

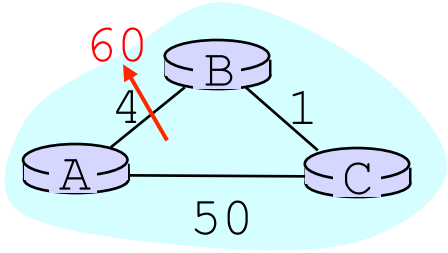
The key here is that the starting point is not the initialization, but some other set of entries. Convergence could be different!

DV: Link Cost Changes





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

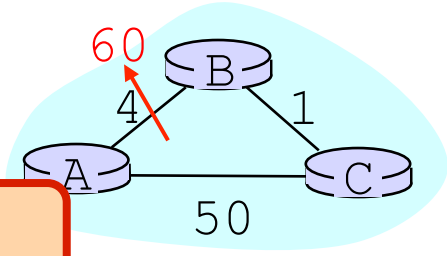
	A	B
A	50	5
B	54	1

	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
Node A (via B, C; to B)	B: 4, 51 / C: 5, 50	B: 60, 51 / C: 61, 50	B: 60, 51 / C: 61, 50	B: 60, 51 / C: 61, 50	B: 60, 51 / C: 61, 50
Node B (via A, C; to A)	A: 4, 6 / C: 9, 1	A: 60, 6 / C: 65, 1	A: 60, 6 / C: 110, 1	A: 60, 6 / C: 110, 1	A: 60, 8 / C: 110, 1
Node C (via A, B; to A)	A: 50, 5 / B: 54, 1	A: 50, 5 / B: 54, 1	A: 50, 5 / B: 101, 1	A: 50, 7 / B: 101, 1	A: 50, 7 / B: 101, 1

Link cost changes here

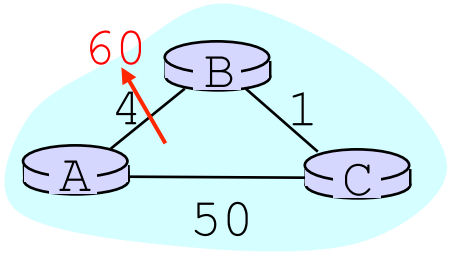
"bad news travels slowly"
(not yet converged)

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 A-B changed

Node A

via

to	B	C
B	4	51
C	5	50

	B	C
B	60	51
C	61	50

Node B

	A	C
A	4	∞
C	∞	1

	A	C
A	60	6
C	65	1

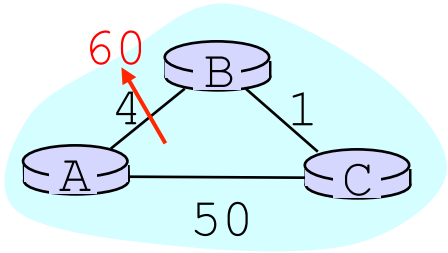
Node C

	A	B
A	50	5
B	54	1

	A	B
A	50	5
B	54	1

Link cost changes here

DV: Poisoned Reverse

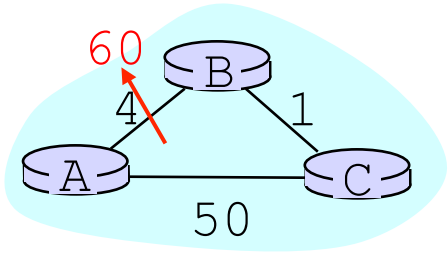


If B routes through C to get to A:
B tells C its (B's) distance to A is infinite

	Stable state	A-B changed	A sends its DV to B, C	B sends its DV to A, C
Node A (via B, C; to B, C)	B: 4, 51 C: 5, 50	B: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50
Node B (via A, C; to A, C)	A: 4, ∞ C: ∞, 1	A: 60, ∞ C: ∞, 1	A: 60, ∞ C: 110, 1	A: 60, ∞ C: 110, 1
Node C (via A, B; to A, B)	A: 50, 5 B: 54, 1	A: 50, 5 B: 54, 1	A: 50, 5 B: ∞, 1	A: 50, 7 B: ∞, 1

Link cost changes here

DV: Poisoned Reverse

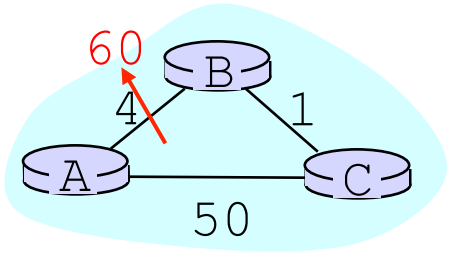


If B routes through C to get to A:
B tells C its (B's) distance to A is infinite

	Stable state	A-B changed	A sends its DV to B, C	B sends its DV to A, C
Node A (via B, C; to B, C)	B: 4, 51 C: 5, 50	B: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50
Node B (via A, C; to A, C)	A: 4, ∞ C: ∞, 1	A: 60, ∞ C: ∞, 1	A: 60, ∞ C: 110, 1	A: 60, ∞ C: 110, 1
Node C (via A, B; to A, B)	A: 50, 5 B: 54, 1	A: 50, 5 B: 54, 1	A: 50, 5 B: ∞, 1	A: 50, 61 B: ∞, 1

Link cost changes here

DV: Poisoned Reverse



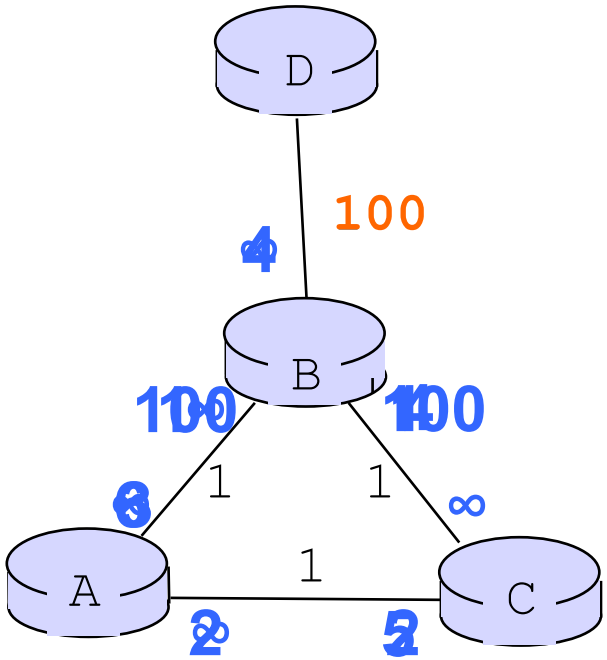
If B routes through C to get to A:
B tells C its (B's) distance to A is infinite

	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: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50	B: 60, 51 C: 61, 50
Node B (via A, C; to A, C)	A: 4, ∞ C: ∞, 1	A: 60, ∞ C: ∞, 1	A: 60, ∞ C: 110, 1	A: 60, ∞ C: 110, 1	A: 60, 51 C: 110, 1
Node C (via A, B; to A, B)	A: 50, 5 B: 54, 1	A: 50, 5 B: 54, 1	A: 50, 5 B: ∞, 1	A: 50, 61 B: ∞, 1	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: Routing

- In this link-state routing network running Dijkstra's algorithm, the set N' (the set of nodes to which the least cost is definitively known) is initially {u}. After two iterations, which nodes belong to N'?

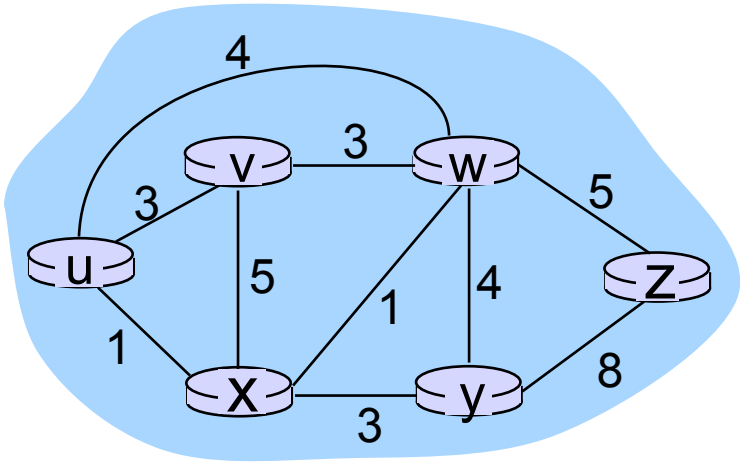
A. u

B. ux

C. uw

D. uvx

E. uwx



Quiz: Routing

- In link state routing, the time for routing to re-converge after a link-cost change does NOT significantly depend on which one of the following?

A. Number of nodes
B. Diameter of the network
C. Whether the link cost increased or decreased
D. Whether routing is load-dependent or not

Network Layer: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

4.6 routing in the Internet

- RIP
- OSPF
- BGP

4.7 broadcast and multicast routing

Hierarchical routing

our routing study thus far - idealization

- all routers identical
- network "flat"

… *not* true in practice

scale: with 600 million destinations:

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

administrative autonomy

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

Hierarchical routing

- aggregate routers into regions, "autonomous systems" (AS)
- routers in same AS run same routing protocol
  - "intra-AS" routing protocol
  - routers in different AS can run different intra-AS routing protocol

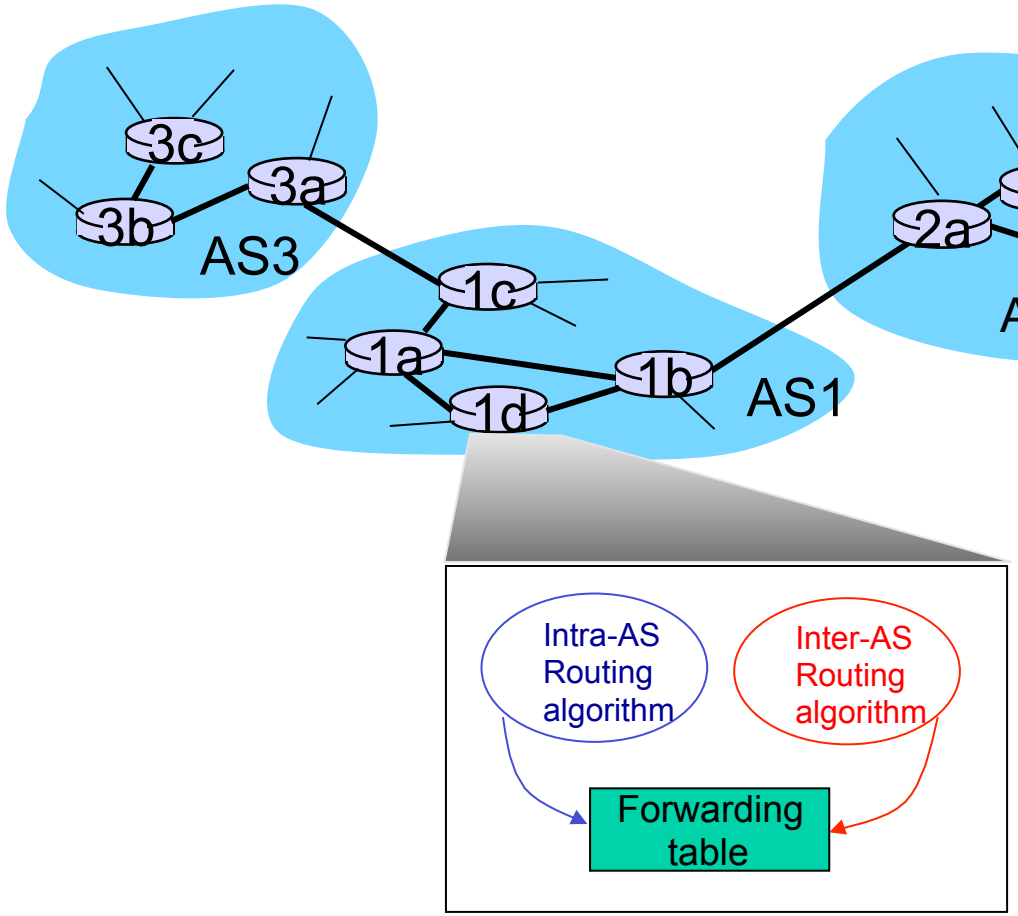
gateway router:

- at "edge" of its own AS
- has link to router in another AS

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)

Interconnected ASes



- forwarding table configured by both intra- and inter-AS routing algorithm
  - intra-AS sets entries for internal dests
  - inter-AS & intra-AS sets entries for external dests

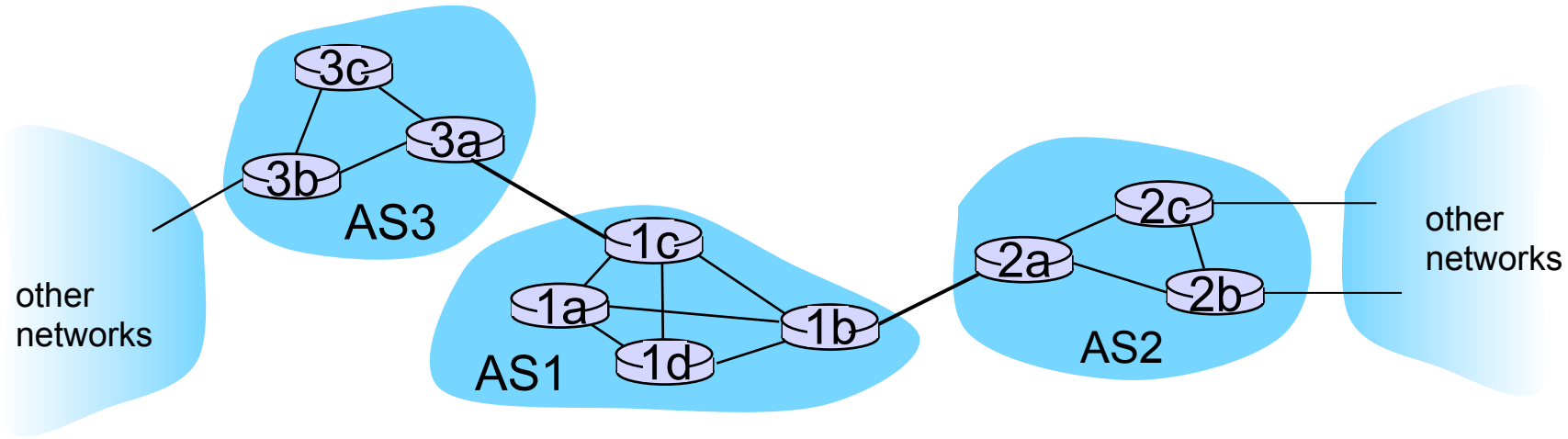
Inter-AS tasks

- suppose router in AS1 receives datagram destined outside of AS1:
  - router should forward packet to gateway router, but which one?

AS1 must:

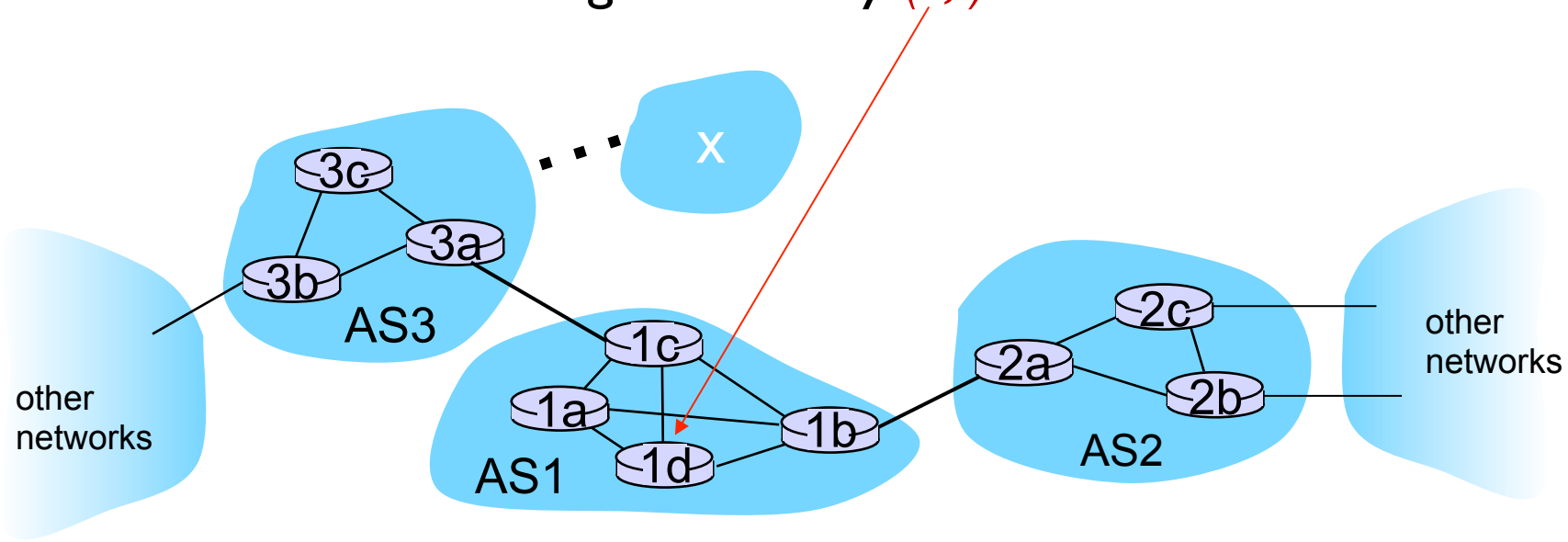
1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

job of inter-AS routing!



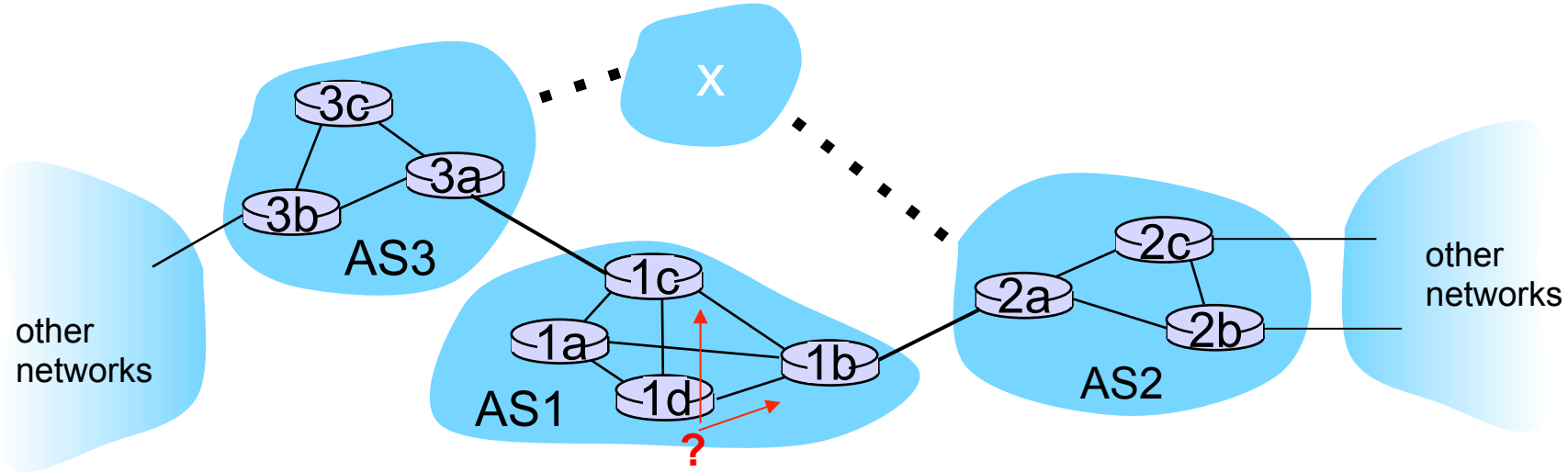
Example: setting forwarding table in router 1d

- suppose AS1 learns (via inter-AS protocol) that subnet *x* reachable via AS3 (gateway 1c), but not via AS2
  - inter-AS protocol propagates reachability info to all internal routers
- router 1d determines from intra-AS routing info that its interface *I* is on the least cost path to 1c
  - installs forwarding table entry *(x,I)*



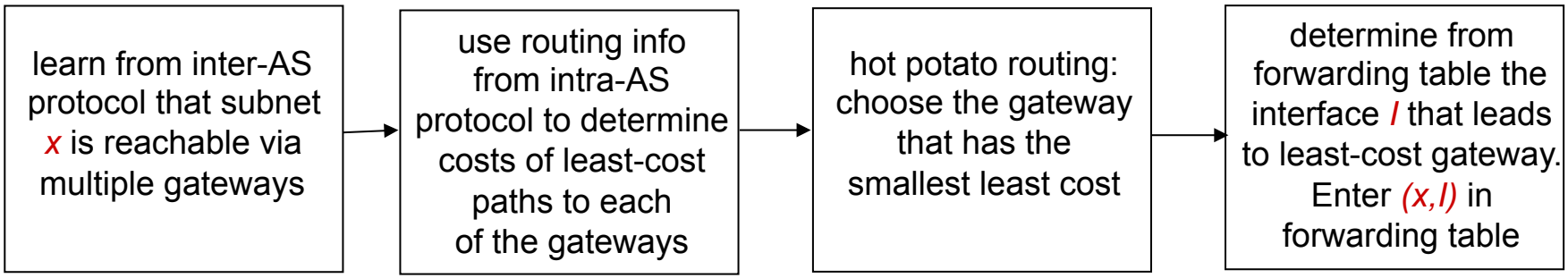
Example: choosing among multiple ASes

- now suppose AS1 learns from inter-AS protocol that subnet *x* is reachable from AS3 *and* from AS2.
- to configure forwarding table, router 1d must determine which gateway it should forward packets towards for dest x
  - this is also job of inter-AS routing protocol!



Example: choosing among multiple ASes

- now suppose AS1 learns from inter-AS protocol that subnet *x* is reachable from AS3 *and* from AS2.
- to configure forwarding table, router 1d must determine towards which gateway it should forward packets for dest x
  - this is also job of inter-AS routing protocol!
- *hot potato routing: send* packet towards closest of two routers.



Network Layer: *done!*

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

- datagram format, IPv4 addressing, ICMP, IPv6

4.5 routing algorithms

- link state, distance vector, hierarchical routing

4.6 routing in the Internet (NOT COVERED)

- RIP, OSPF, BGP

4.7 broadcast and multicast routing (NOT COVERED)

- understand principles behind network layer services:
  - network layer service models, forwarding versus routing how a router works, routing (path selection), broadcast, multicast
- instantiation, implementation in the Internet