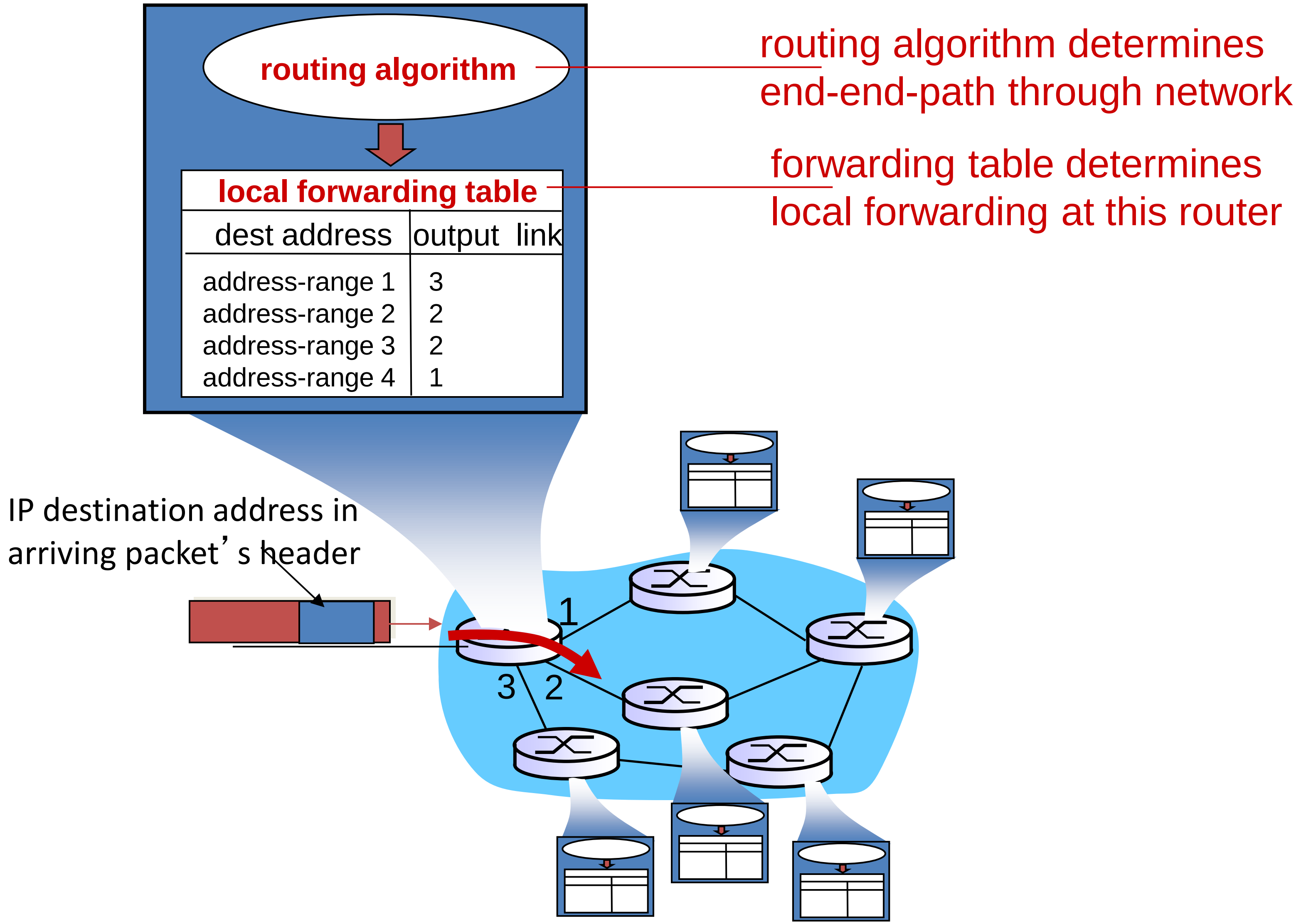
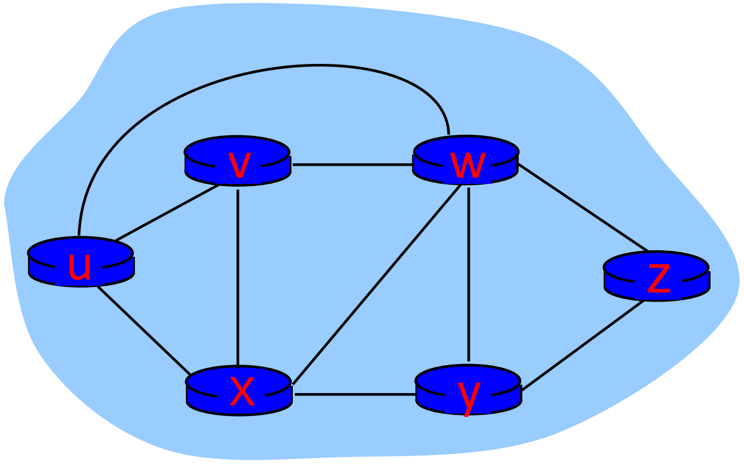
CS 43: Computer Networks Routing

Kevin Webb
Swarthmore College
November 14, 2017

Interplay between routing, forwarding



Graph Abstraction

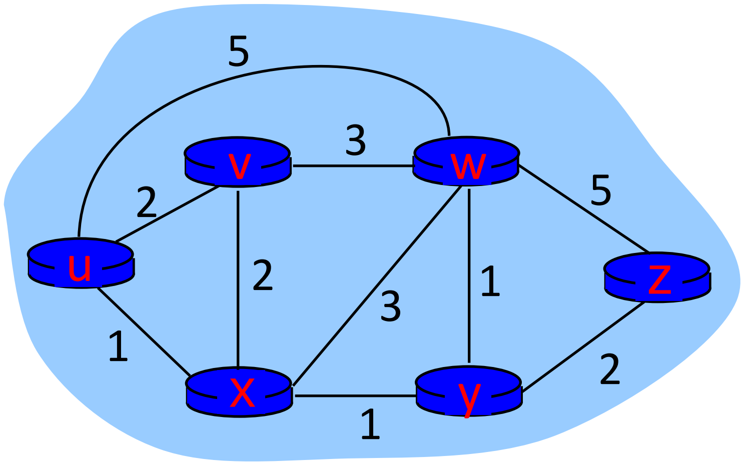


graph: G = (N,E)

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) }

Link Cost



$c(x,x')$ = 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

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 Challenges

- How to choose best path?
  - Defining “best” can be slippery
- How to scale to millions of users?
  - Minimize control messages and routing table size
- How to adapt quickly to failures or changes?
  - Node and link failures, plus message loss

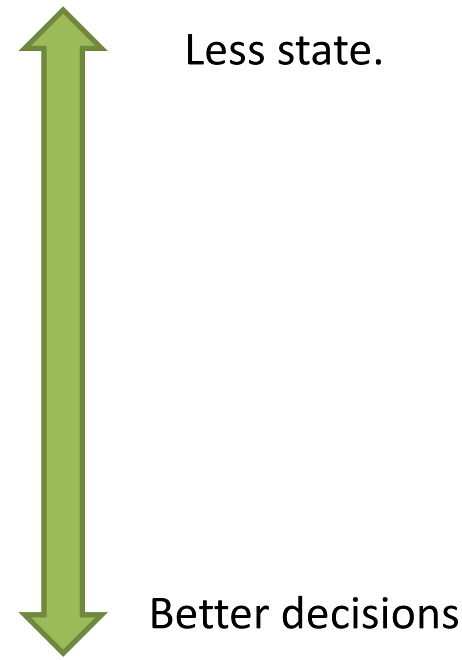
How much information should a router know about the network?

A. The next hop and cost of forwarding to its neighbor(s).

B. The next hop and cost of forwarding to any destination.

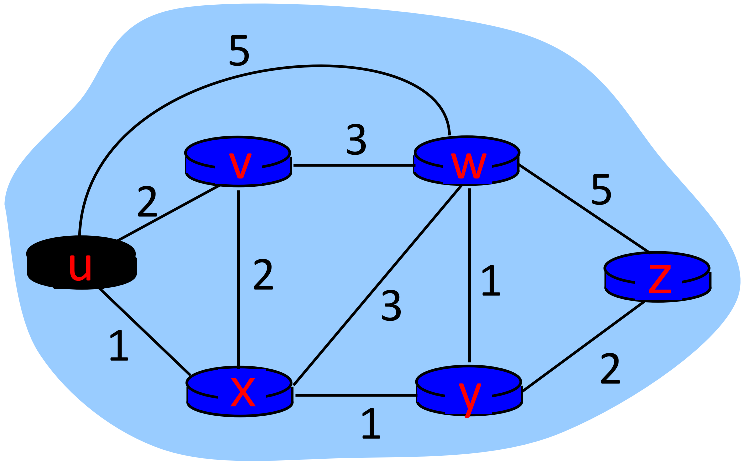
C. The status and cost of every link in the network.

D. Some other amount of information.



Routing Table?

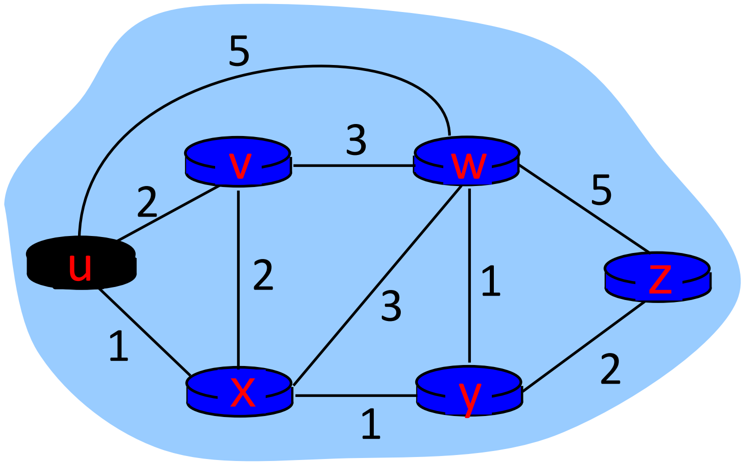
Dest	Next Hop
V	V
X	X
W	X
Y	X
Z	X



- *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, maybe path itself)
- <u>This is a key difference between routing & forwarding!</u>

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 (Decentralized)

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

Link-state Routing

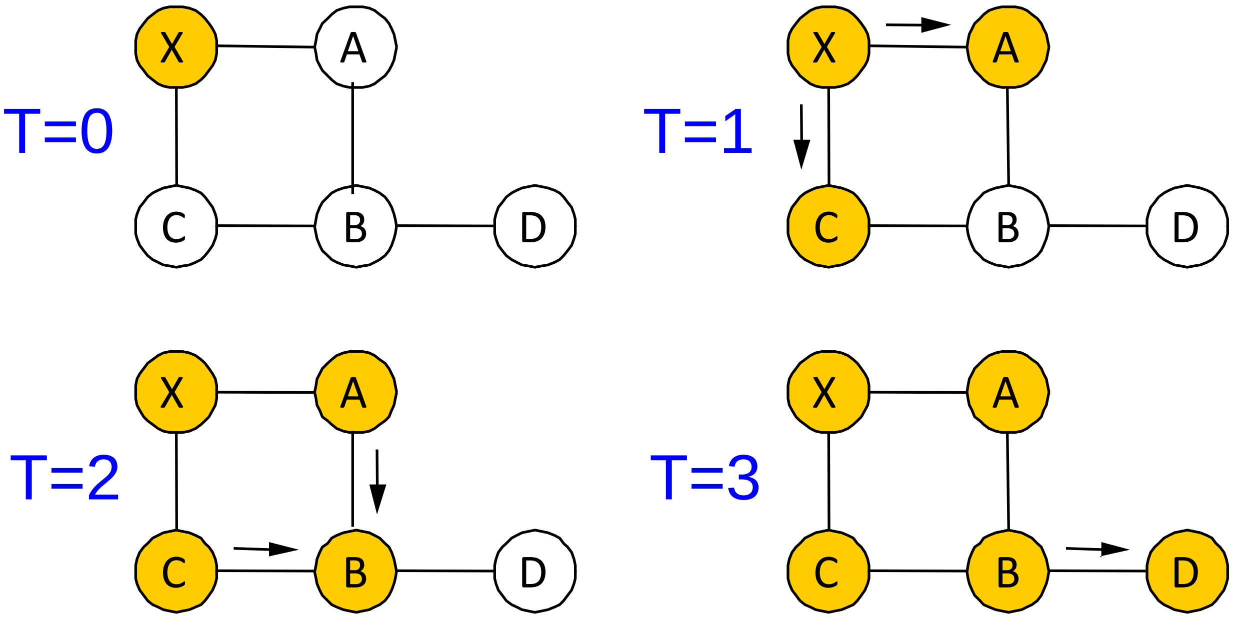
- Two phases
  - Reliable flooding
    - Tell all routers what you know about your links
    - Typically in response to event: link failure/recovery/cost
  - Path calculation (Dijkstra's algorithm)
    - Each router computes best path over complete network
- Motivation
  - Global information allows optimal routing
  - Straightforward to implement and verify

Flooding LSAs

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

Flooding Example

- LSA generated by X at T=0



Dijkstra’s Algorithm

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

Nodes we’ve determined lowest-cost path for already.

Best known cost for reaching node v.

Dijkstra's Algorithm

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

Only know best route to self so far.

For every other router, set it's known distance to link cost if it's a neighbor. Otherwise, set it to infinity.

Dijkstra's Algorithm

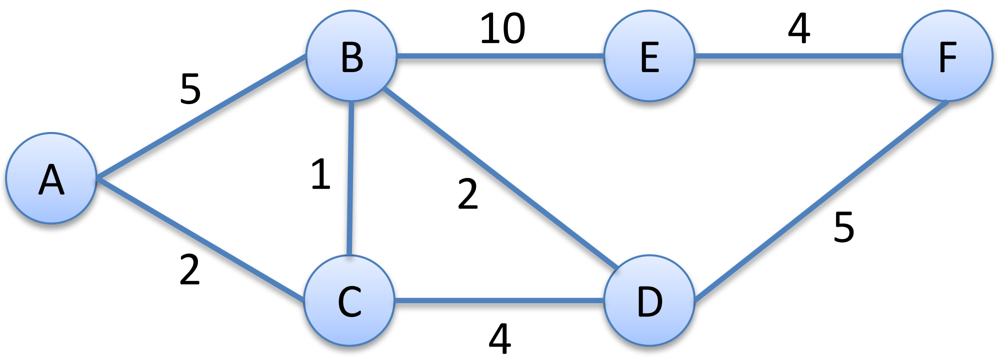
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

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'
```

Pick the node (w) that isn't already in N' with the shortest distance (least cost path) and add it to N'.

Check all possible destinations from w. If going through w gives a lower cost to destination v, update D(v).

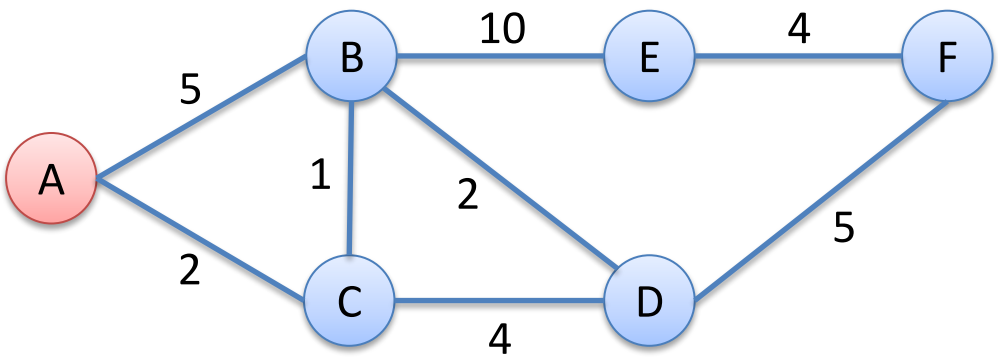
Dijkstra's Algorithm Example



- Goal: From the perspective of node A:
  - Determine shortest path to every destination

- Other perspectives:
  - Look up "Dijkstra's Algorithm" on YouTube

Dijkstra’s Algorithm – Step 0



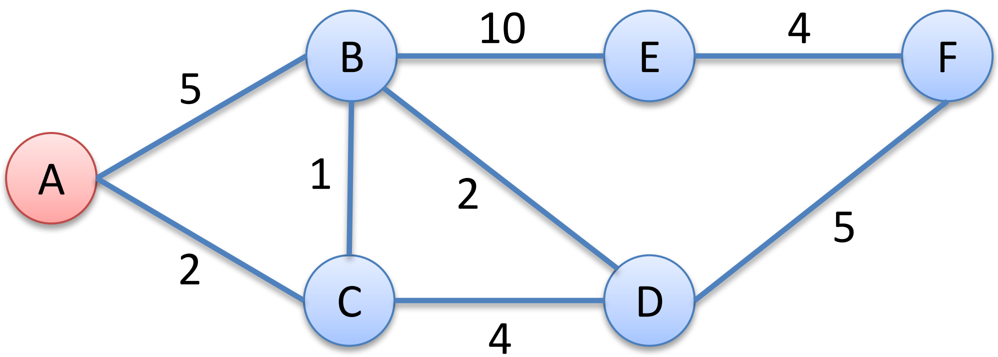
Previous Step

Dest	Path	Cost D(v)
A		
B		
C		
D		
E		
F		

This Step

Dest	Path	Cost D(v)
✓ A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

Dijkstra’s Algorithm – Step 1



Previous Step

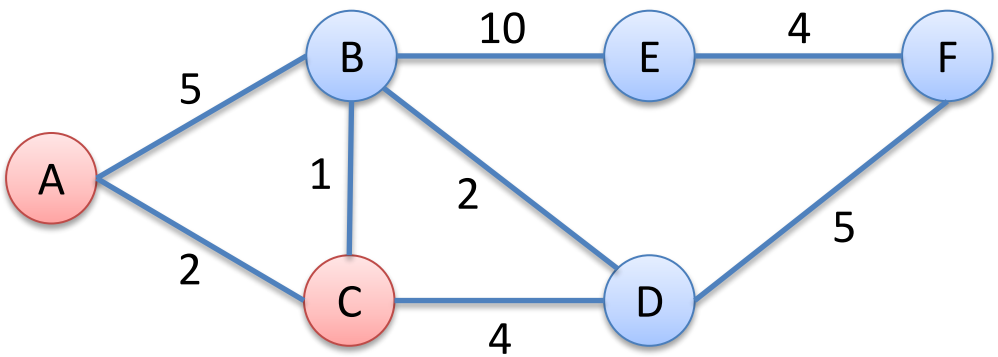
Dest	Path	Cost D(v)
A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

Pick
Min

This Step

Dest	Path	Cost D(v)
A	A	0
B		
C		
D		
E		
F		

Dijkstra's Algorithm – Step 1



Can we find lower cost to any other node by going through C?

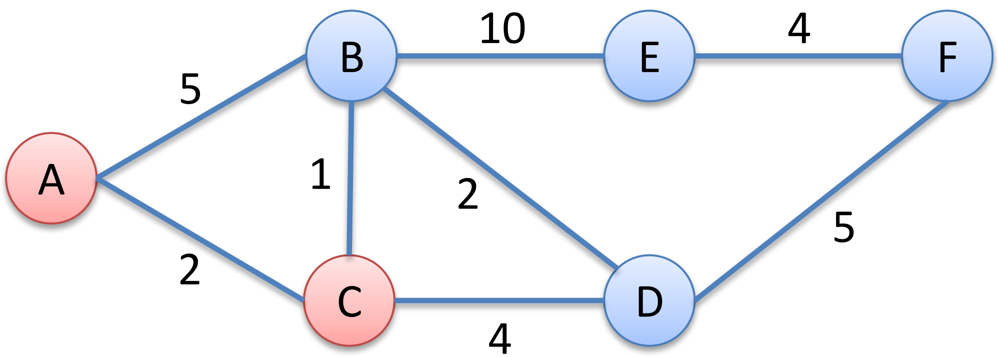
Previous Step

	Dest	Path	Cost D(v)
✓	A	A	0
	B	B	5
	C	C	2
	D	?	∞
	E	?	∞
	F	?	∞

This Step

	Dest	Path	Cost D(v)
✓	A	A	0
	B		
✓	C	C	2
	D		
	E		
	F		

Dijkstra’s Algorithm – Step 1



Consider path to B:

D(B)
or
D(C) + cost(C, B)

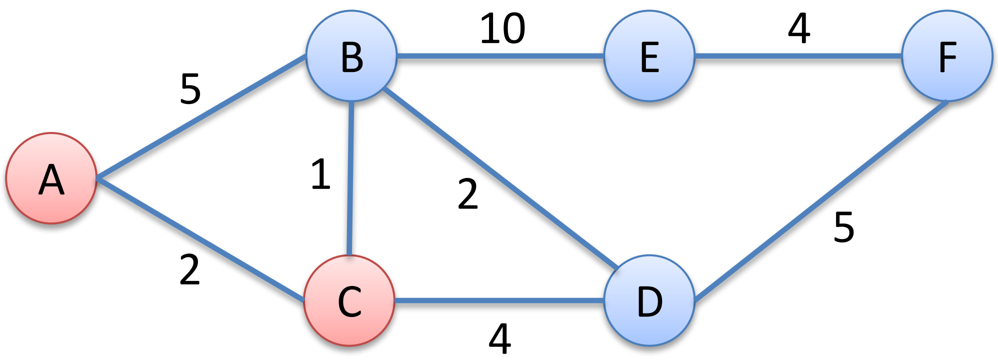
Previous Step

Dest	Path	Cost D(v)
A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
A	A	0
B		
C	C	2
D		
E		
F		

Dijkstra’s Algorithm – Step 1



Consider path to B:

D(B) = 5
or
D(C) + cost(C, B)
2 + 1 = 3

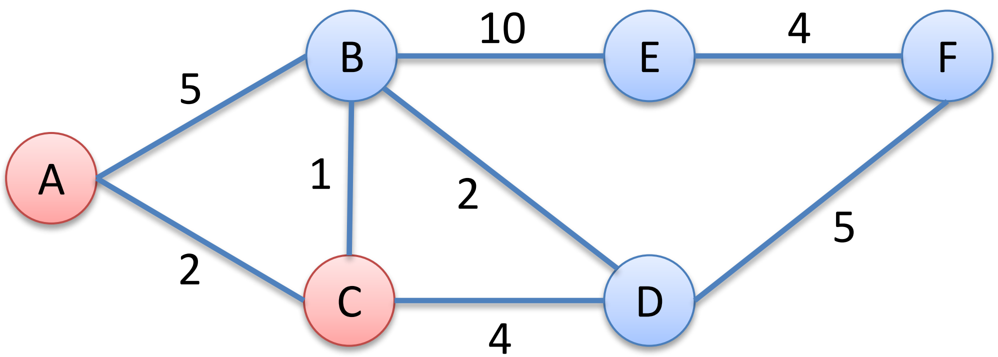
Previous Step

Dest	Path	Cost D(v)
A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D		
E		
F		

Dijkstra's Algorithm – Step 1



Consider path to D:

D(D) = ∞
or
D(C) + cost(C, D)
2 + 4 = 6

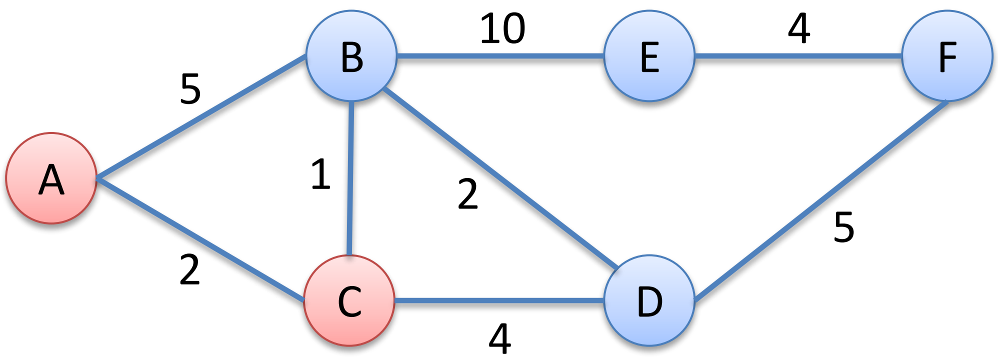
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
B	C, B	3
✓ C	C	2
D	C, D	6
E		
F		

Dijkstra's Algorithm – Step 1



Still no information about E or F.

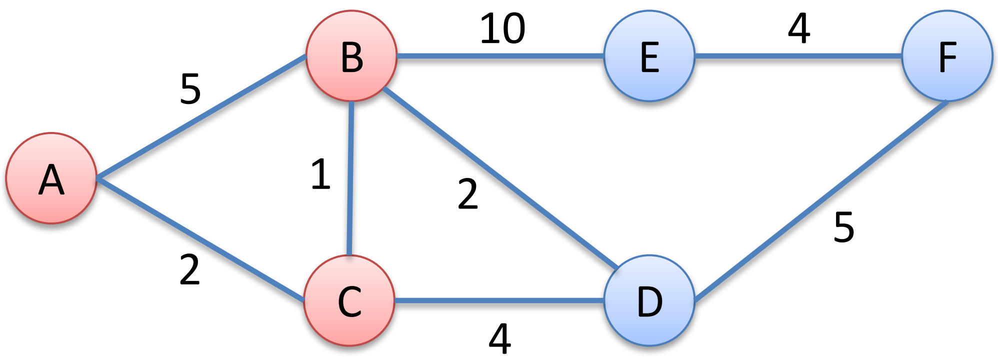
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
B	C, B	3
✓ C	C	2
D	C, D	6
E	?	∞
F	?	∞

Dijkstra's Algorithm – Step 2



Choose B.

Previous Step

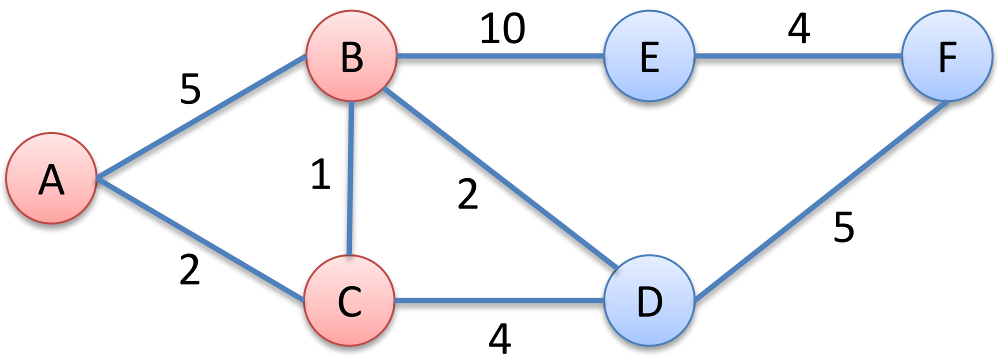
Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D	C, D	6
E	?	∞
F	?	∞

Pick Min

This Step

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D		
E		
F		

Dijkstra's Algorithm – Step 2



Consider path to D:

D(D) = 6
or
D(B) + cost(B, D)
3 + 2 = 5

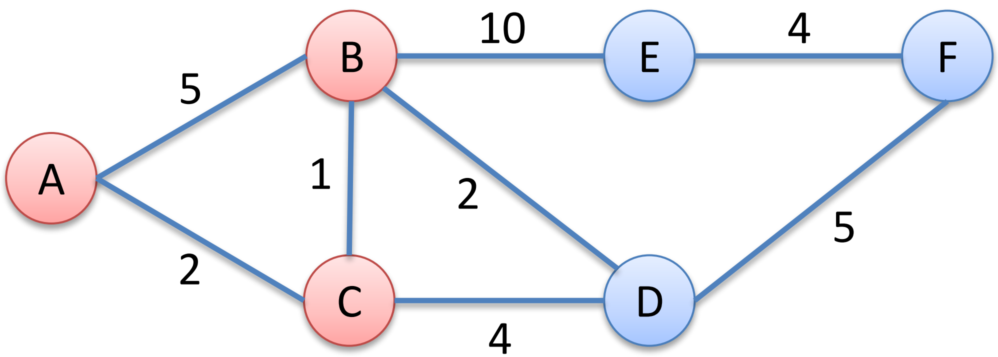
Previous Step

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D	C, D	6
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D	C, B, D	5
E		
F		

Dijkstra’s Algorithm – Step 2



Consider path to E:

D(E) = ∞
or
D(B) + cost(B, E)
3 + 10 = 13

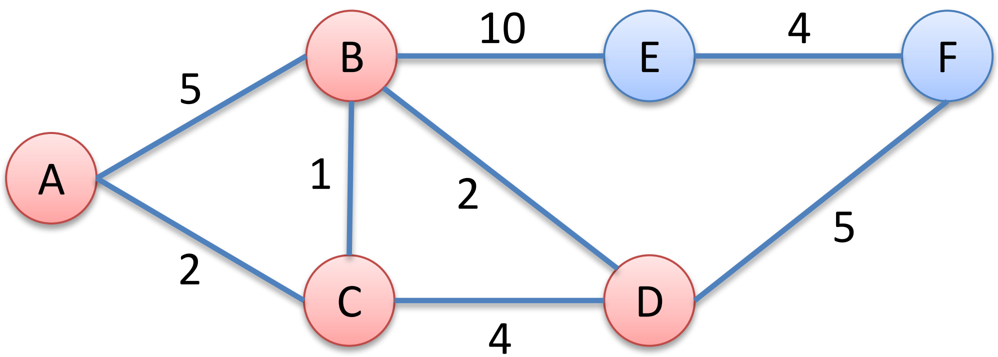
Previous Step

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D	C, D	6
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D	C, B, D	5
E	C, B, E	13
F	?	∞

Dijkstra’s Algorithm – Step 3



Choose D.

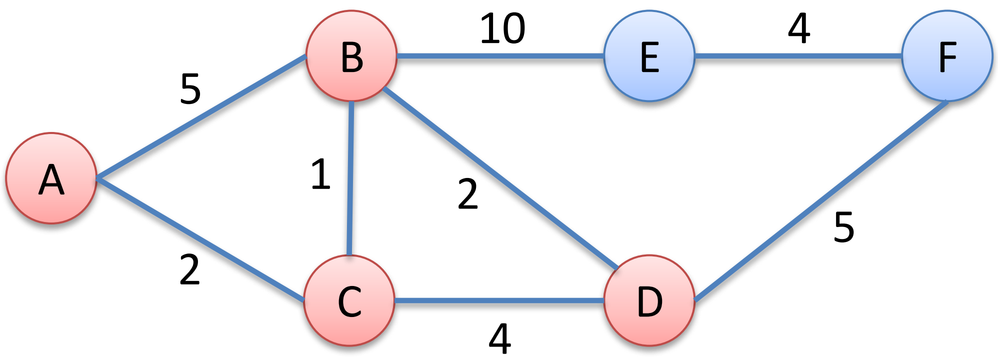
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
D	C, B, D	5
E	C, B, E	13
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
✓ D	C, B, D	5
E		
F		

Dijkstra's Algorithm – Step 3



No change for E.

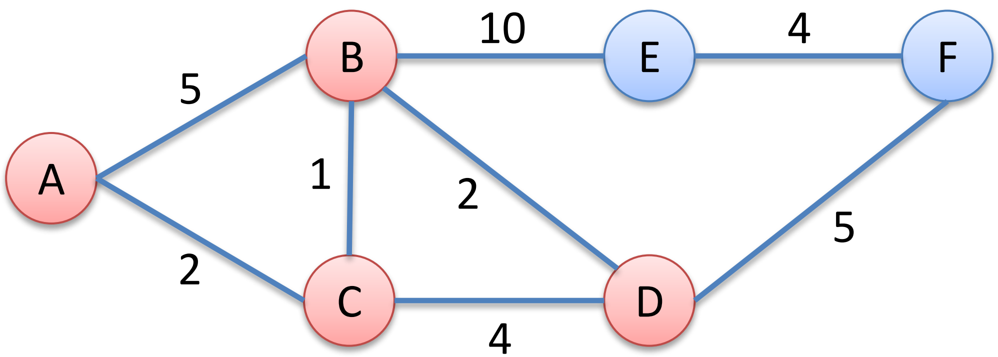
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
D	C, B, D	5
E	C, B, E	13
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
✓ D	C, B, D	5
E	C, B, E	13
F		

Dijkstra’s Algorithm – Step 3



Consider path to F:

D(F) = ∞
or
D(D) + cost(D, F)
5 + 5 = 10

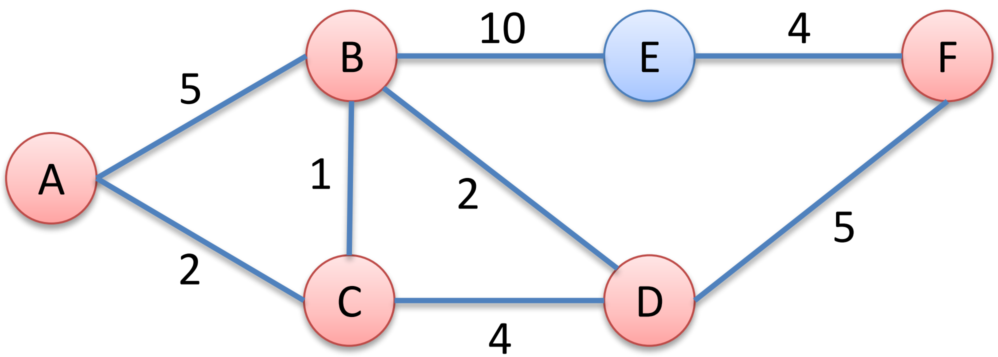
Previous Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
	D	C, B, D	5
	E	C, B, E	13
	F	?	∞

This Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
✓	D	C, B, D	5
	E	C, B, E	13
	F	C, B, D, F	10

Dijkstra's Algorithm – Step 4



Choose F.

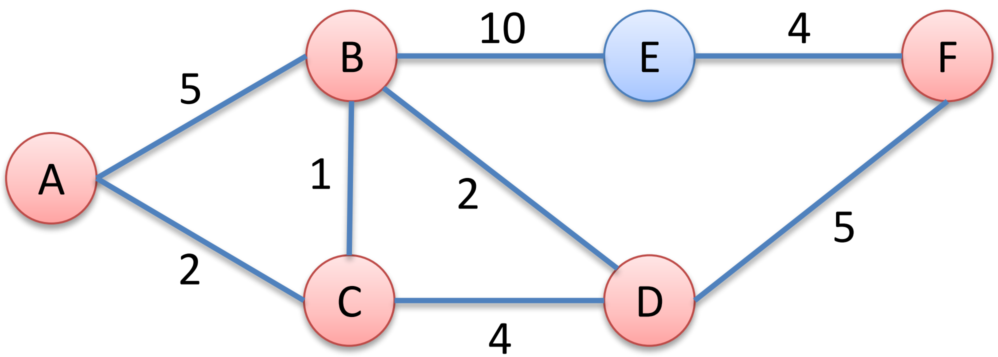
Previous Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
✓	D	C, B, D	5
	E	C, B, E	13
	F	C, B, D, F	10

This Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
✓	D	C, B, D	5
✓	F	C, B, D, F	10

Dijkstra's Algorithm – Step 4



Consider path to E:

D(E) = 13
or
D(F) + cost(F, E)
10 + 4 = 14

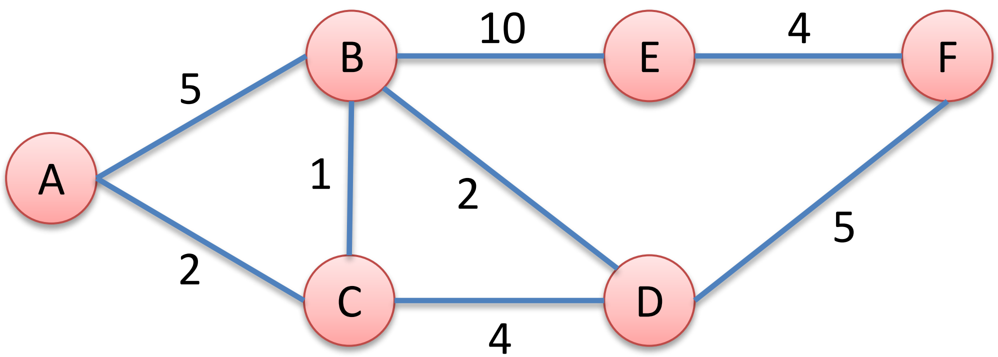
Previous Step

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D	C, B, D	5
E	C, B, E	13
F	C, B, D, F	10

This Step

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D	C, B, D	5
E	C, B, E	13
F	C, B, D, F	10

Dijkstra’s Algorithm – Step 5



Choose E.

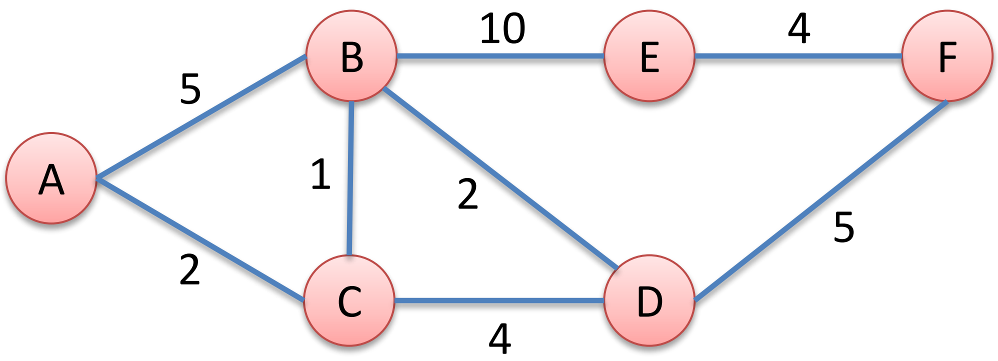
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
✓ D	C, B, D	5
E	C, B, E	13
✓ F	C, B, D, F	10

This Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
✓ D	C, B, D	5
✓ E	C, B, E	13
✓ F	C, B, D, F	10

Dijkstra’s Algorithm – Done!



Final Answer

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D	C, B, D	5
E	C, B, E	13
F	C, B, D, F	10

Populate Forwarding Table

Forwarding Table

Dest	Forward To
B	C
C	C
D	C
E	C
F	C

Dijkstra’s Algorithm – Complexity

- With N nodes and E edges…
- As previously described it’s $O(N^2)$
  - At each step, there are N nodes to choose next
  - Total of N steps (each node must be chosen)
- Fastest known is $O(N \log N + E)$
  - Uses a min-heap

Link State - Summary

- + Fast convergence (reacts to events quickly)
- + Small window of inconsistency

- − Large number of messages sent on events
- − Large routing tables as network size grows

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 (Decentralized)

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

Bellman-Ford Equation

let

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

then

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

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

Distance Vectors

- Let $D_x(y)$ = vector of least cost from x to y
- Node x:
  - Knows cost to each neighbor v: $c(x,v)$
  - Maintains its neighbors' distance vectors. For each neighbor v, x maintains: $\mathbf{D}_v = [D_v(y): y \in N]$
- **As opposed to link state:**
  - **Only keeps state for yourself and direct neighbors**

Distance Vector Algorithm

- Periodically, each node sends its own distance vector to neighbors
- Upon receiving new DV from neighbor, update its local DV using B-F equation:

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

- Under typical conditions, $D_x(y)$ will converge to the least cost $d_x(y)$

Distance Vector Algorithm

Iterative, asynchronous:

Iteration when:

- Local link cost change
- DV update from neighbor
- Periodic timer

Distributed:

- Each node knows only a portion of global link info

each node:

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

recompute estimates

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

$$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\} = \min\{2+0\ ,\ 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\} = \min\{2+1\ ,\ 7+0\} = 3$$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

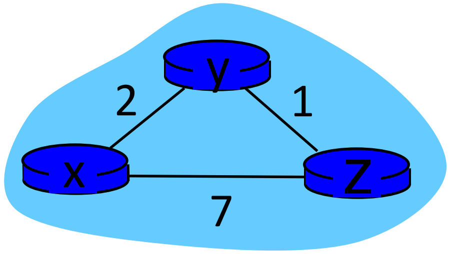
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



time

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y) +$
$D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

node x table

	cost to		
from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

	cost to		
from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

	cost to		
from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

	cost to		
from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

	cost to		
from	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	cost to		
from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

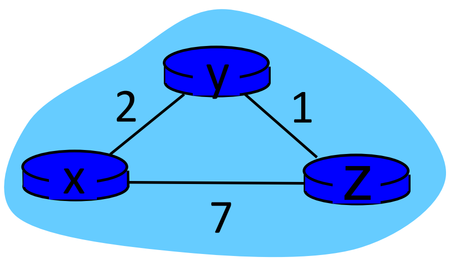
node z table

	cost to		
from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

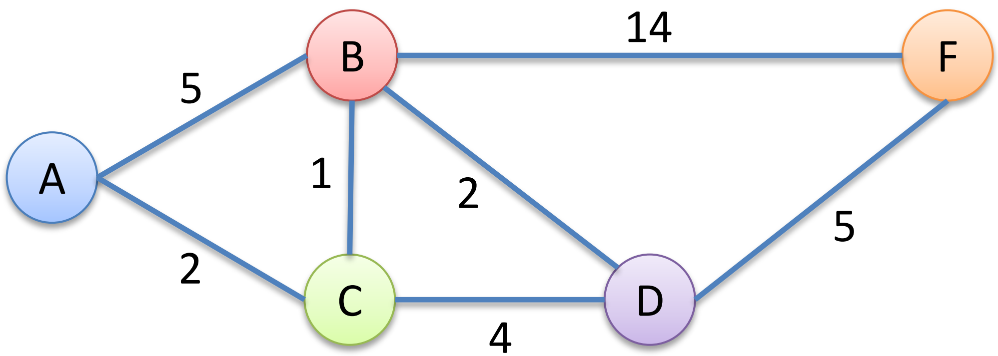
	cost to		
from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

	cost to		
from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

time

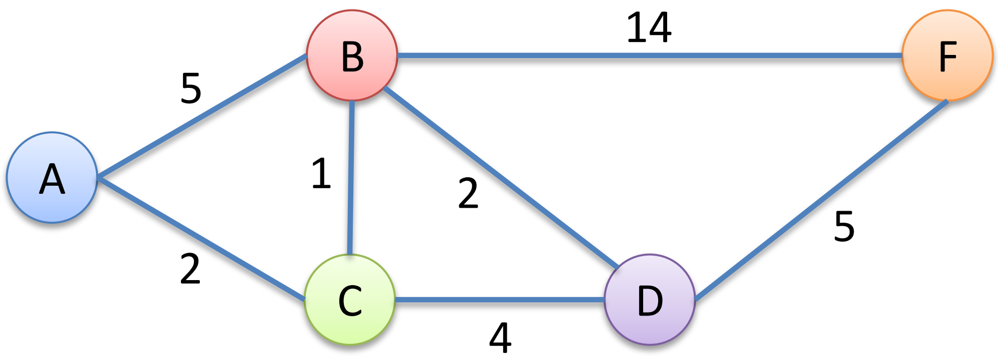


Distance Vector Example



- Same network as Dijkstra's example, without node E.
- What I'll show you next is routing table (of distance vectors) at each router.

Distance Vector – Round 0



Routers populate their forwarding table by taking the row minimum.

Router F

Via→ ↓ To	B	D
A		
B	14	
C		
D		5

Router A

Via→ ↓ To	B	C
B	5	
C		2
D		
F		

Router B

Via→ ↓ To	A	C	D	F
A	5			
C		1		
D			2	
F				14

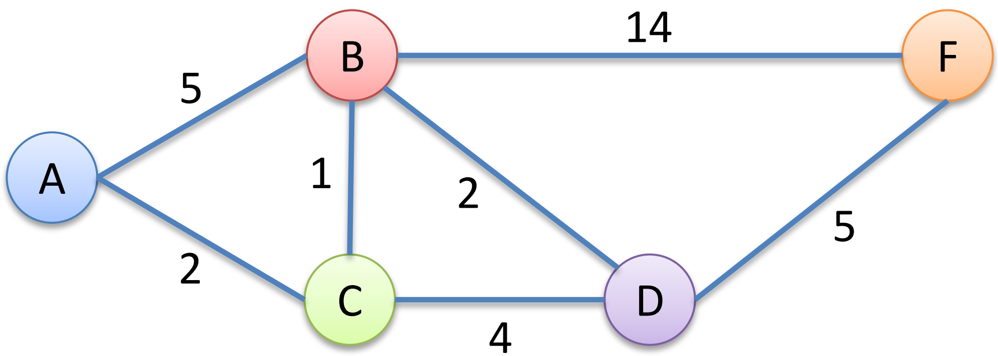
Router C

Via→ ↓ To	A	B	D
A	2		
B		1	
D			4
F			

Router D

Via→ ↓ To	B	C	F
A			
B	2		
C		4	
F			5

Distance Vector – Round 0



Router exchange their local vectors with direct neighbors.
We'll assume they all exchange at once (synchronous). (Not realistic)

Router F

Via→ ↓ To	B	D
A		
B	14	
C		
D		5

Router A

Via→ ↓ To	B	C
B	5	
C		2
D		
F		

Router B

Via→ ↓ To	A	C	D	F
A	5			
C		1		
D			2	
F				14

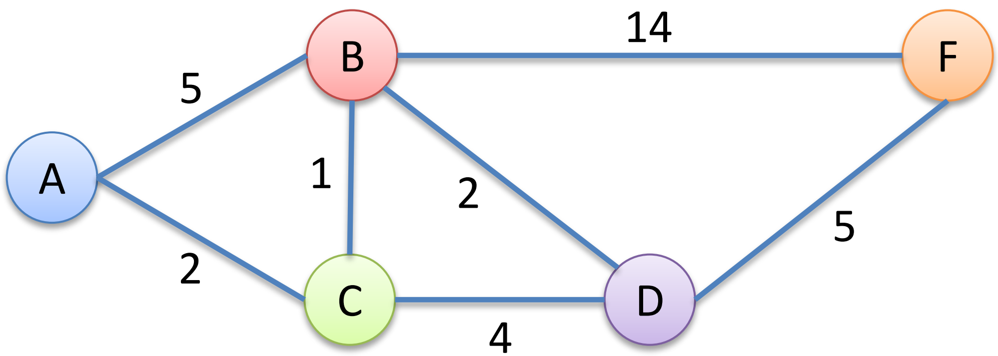
Router C

Via→ ↓ To	A	B	D
A	2		
B		1	
D			4
F			

Router D

Via→ ↓ To	B	C	F
A			
B	2		
C		4	
F			5

Distance Vector – Round 1



A will send to neighbors (B & C):
I can get to B in 5 and C in 2.

Router F

Via→ ↓ To	B	D
A		
B	14	
C		
D		5

Router A

Via→ ↓ To	B	C
B	5	
C		2
D		
F		

Router B

Via→ ↓ To	A	C	D	F
A	5			
C	7	1		
D			2	
F				14

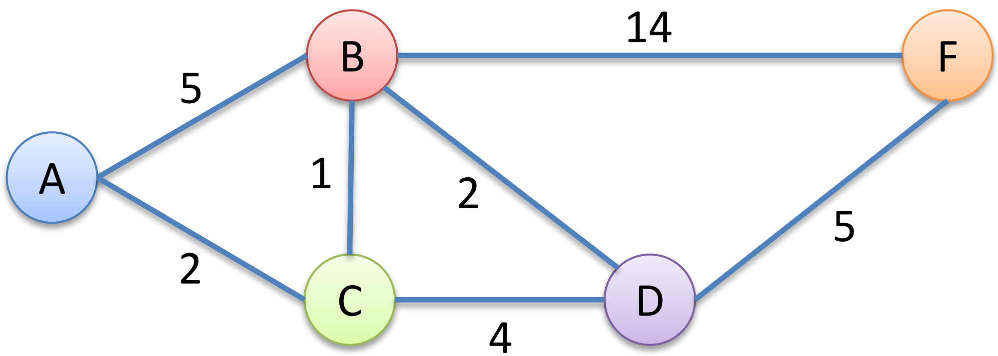
Router C

Via→ ↓ To	A	B	D
A	2		
B	7	1	
D			4
F			

Router D

Via→ ↓ To	B	C	F
A			
B	2		
C		4	
F			5

Distance Vector – Round 1



Router F

Via→ ↓ To	B	D
A	19	
B	14	
C	15	
D	16	5

B will send to neighbors (A, C, D, F):
I can get to A in 5, C in 1, D in 2, and F in 14.

Router A

Via→ ↓ To	B	C
B	5	
C	6	2
D	7	
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5			
C	7	1		
D			2	
F				14

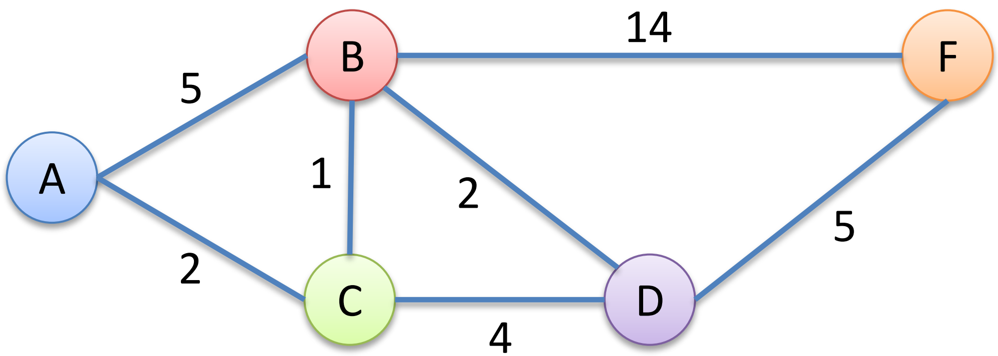
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	
D		3	4
F		15	

Router D

Via→ ↓ To	B	C	F
A	7		
B	2		
C	3	4	
F	16		5

Distance Vector – Round 1



C will send to neighbors (A, B, D):
I can get to A in 2, B in 1, and D in 4.

Router F

Via→ ↓ To	B	D
A	19	
B	14	
C	15	
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1		
D		5	2	
F				14

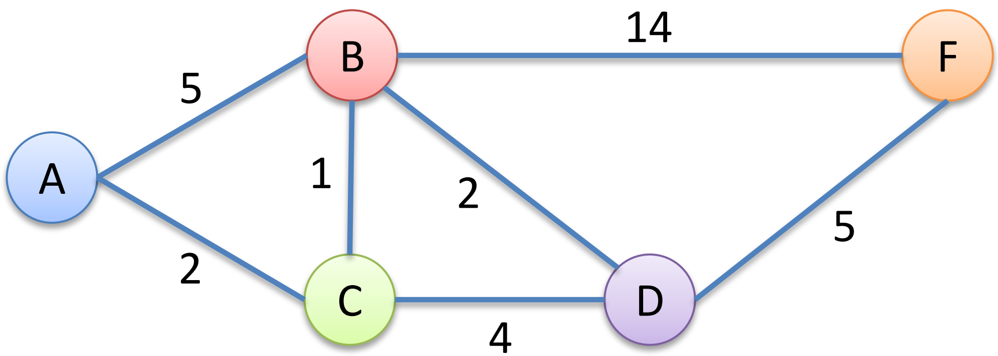
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	
D		3	4
F		15	

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	
C	3	4	
F	16		5

Distance Vector – Round 1



D will send to neighbors (B, C, F):
I can get to B in 2, C in 4, and F in 5.

Router F

Via→ ↓ To	B	D
A	19	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	
F			7	14

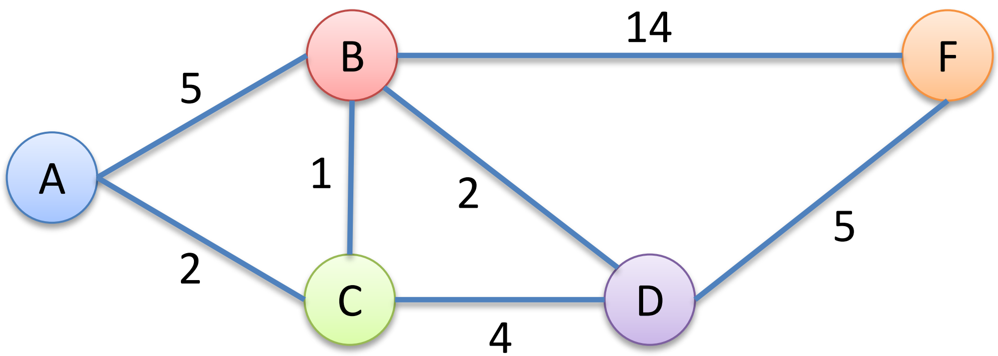
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	6
D		3	4
F		15	9

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	
C	3	4	
F	16		5

Distance Vector – Round 1



F will send to neighbors (B, D):
I can get to B in 14, D in 5.

Router F

Via→ ↓ To	B	D
A	19	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

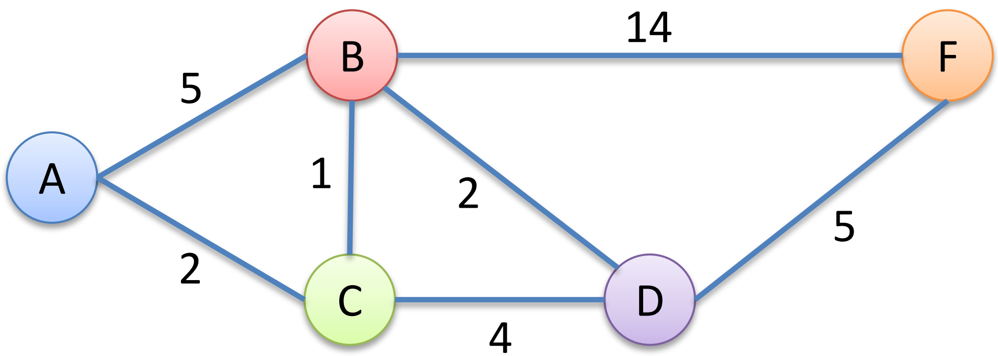
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	6
D		3	4
F		15	9

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	19
C	3	4	
F	16		5

At the end of round 1, how many routers need to update their forwarding tables?



A – 1, B – 2, C – 3, D – 4, E – 5

Router F

Via→ ↓ To	B	D
A	19	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

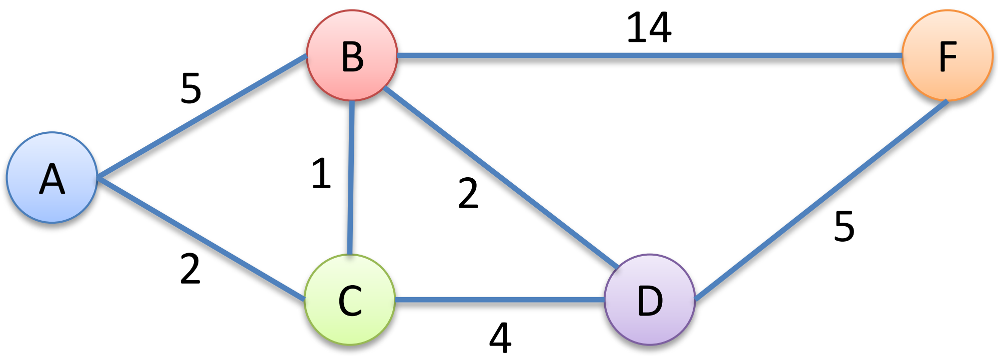
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	6
D		3	4
F		15	9

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	19
C	3	4	
F	16		5

Distance Vector – Round 2



Router F

Via→ ↓ To	B	D
A	19	
B	14	7
C	15	9
D	16	5

Each router advertises the best cost it has to each destination.
Nothing new to learn from A or F, so we'll skip their announcements.

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

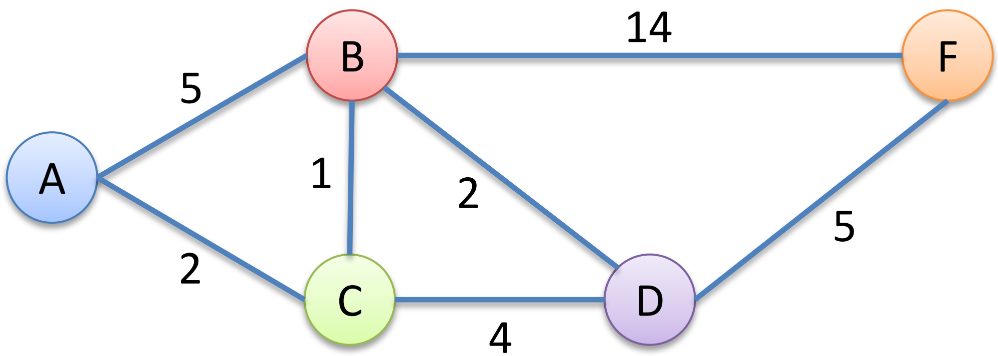
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	6
D		3	4
F		15	9

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	19
C	3	4	
F	16		5

Distance Vector – Round 2



B will send to neighbors (A, C, D, F):
I can get to A in 3, C in 1, D in 2, and F in 7.

Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	12	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

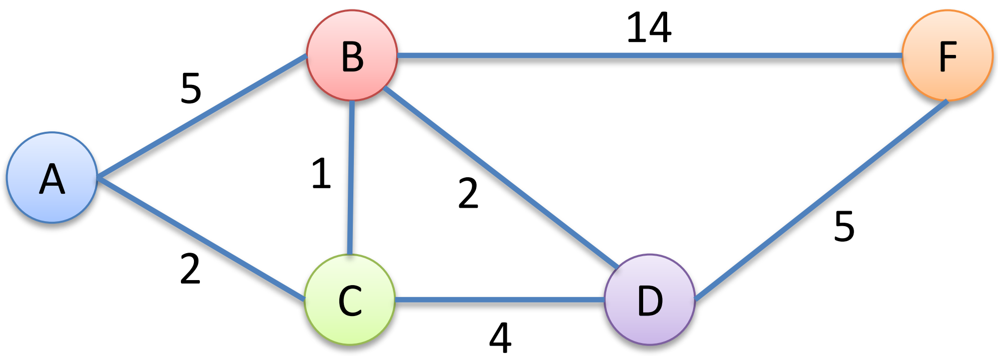
Router C

Via→ ↓ To	A	B	D
A	2	4?	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?		5

Distance Vector – Round 2



C will send to neighbors (A, B, D):
I can get to A in 2, B in 1, D in 3, and F in 9.

Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	11

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		4?	2	19
F		10	7	14

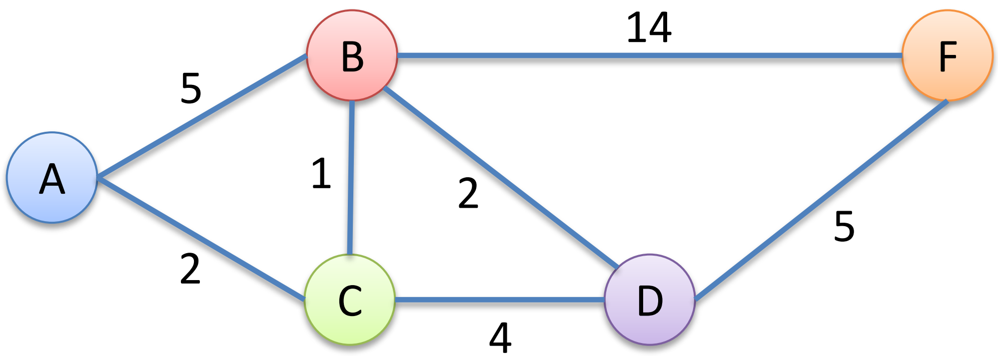
Router C

Via→ ↓ To	A	B	D
A	2	4?	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?	13?	5

Distance Vector – Round 2



This process repeats for a while…

Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	11

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		4?	2	19
F		10	7	14

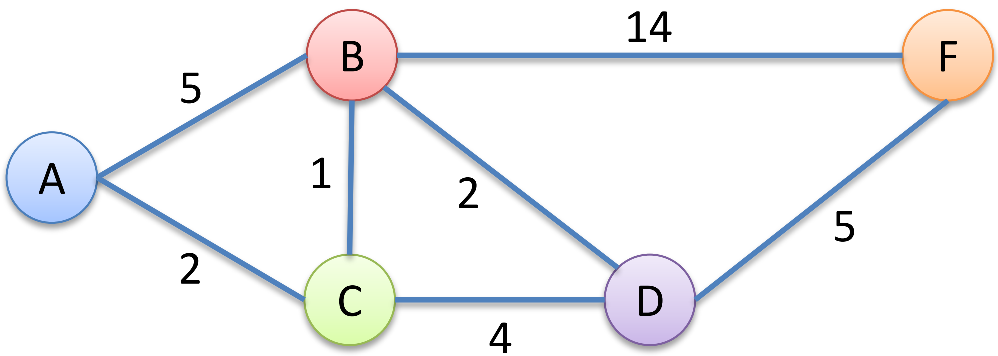
Router C

Via→ ↓ To	A	B	D
A	2	4?	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?	13?	5

Distance Vector – Convergence



Eventually, we reach a converged state.

Router F

Via→ ↓ To	B	D
A	17	10
B	14	7
C	15	8
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	10

Router B

Via→ ↓ To	A	C	D	F
A	5	3	7	24
C	7	1	4	22
D	10	4	2	19
F	15	9	7	14

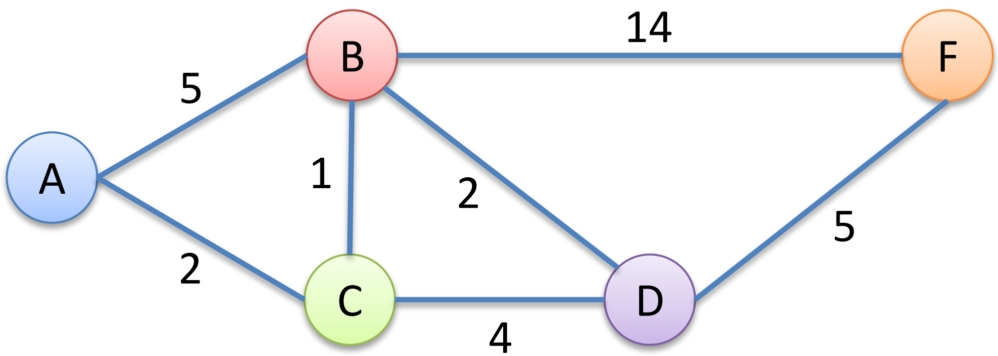
Router C

Via→ ↓ To	A	B	D
A	2	4	9
B	7	1	6
D	7	3	4
F	12	8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	15
B	2	5	12
C	3	4	13
F	9	12	5

Distance Vector – Convergence



Final forwarding tables:

Router F

Via→ ↓ To	B	D
A	17	10
B	14	7
C	15	8
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	10

Router B

Via→ ↓ To	A	C	D	F
A	5	3	7	24
C	7	1	4	22
D	10	4	2	19
F	15	9	7	14

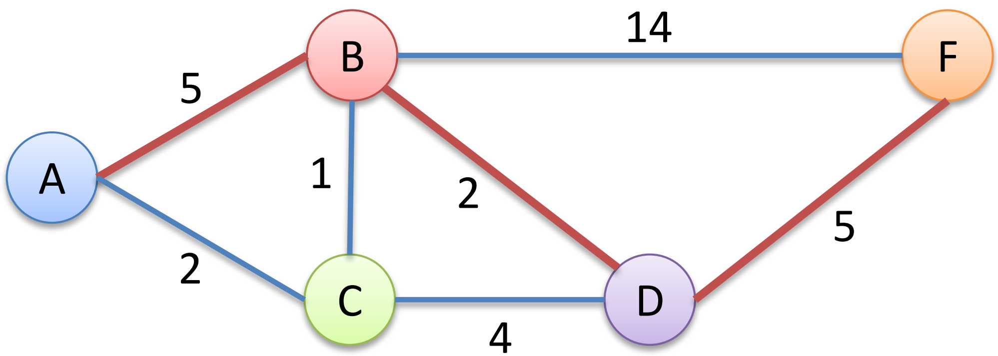
Router C

Via→ ↓ To	A	B	D
A	2	4	9
B	7	1	6
D	7	3	4
F	12	8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	15
B	2	5	12
C	3	4	13
F	9	12	5

Of the links in red below, for how many would a failure cause a loop?



A – 0, B – 1, C – 2, D – 3

Consider the failures independently (not all at the same time).

Router F

Via→ ↓ To	B	D
A	17	10
B	14	7
C	15	8
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	10

Router B

Via→ ↓ To	A	C	D	F
A	5	3	7	24
C	7	1	4	22
D	10	4	2	19
F	15	9	7	14

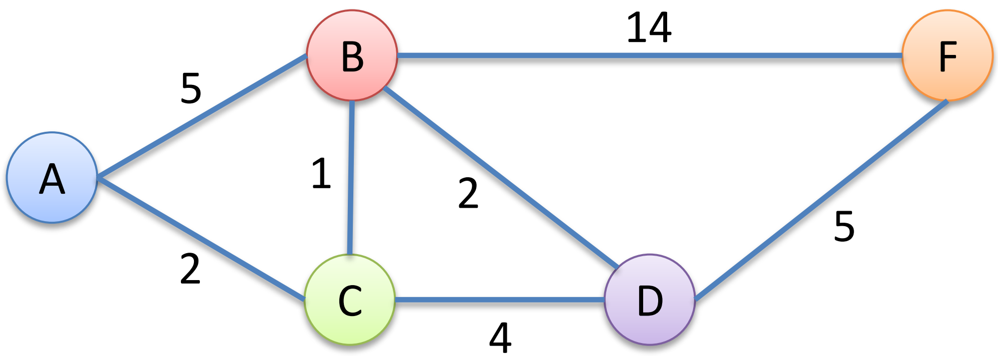
Router C

Via→ ↓ To	A	B	D
A	2	4	9
B	7	1	6
D	7	3	4
F	12	8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	15
B	2	5	12
C	3	4	13
F	9	12	5

Rewind: Distance Vector – Round 2



B will send to neighbors (A, C, D, F):
I can get to A in 3, C in 1, D in 2, and F in 7.

Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	12	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

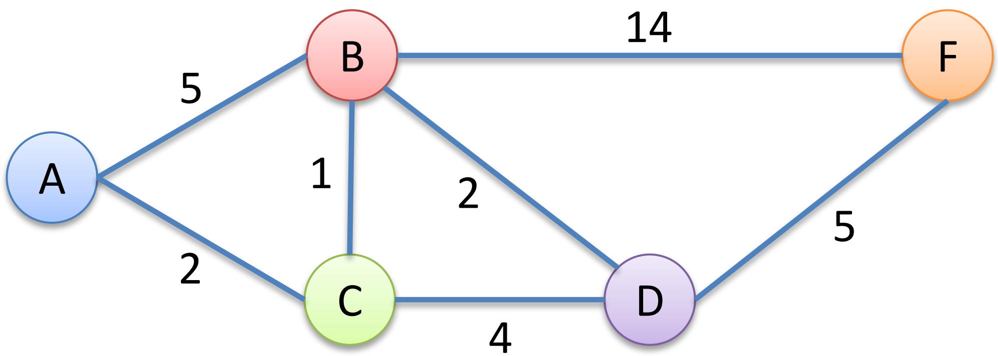
Router C

Via→ ↓ To	A	B	D
A	2	4?	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?		5

Rewind: Distance Vector – Round 2



Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

Poisoned reverse: Don't advertise a lower value to a neighbor if you go through that neighbor to get there!

No!
∞

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	12	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

Router C

Via→ ↓ To	A	B	D
A	2	~~4?~~	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?		5

Loop-prevention

- Route poisoning helps prevent loops, but doesn’t guarantee loop free.

- Other mechanisms help too

- <u>There will always be a window of vulnerability</u>

Summary

Link State

- + Fast convergence (reacts to events quickly)
- + Small window of inconsistency

- − Large number of messages sent on events
- − Large routing tables as network size grows

Distance Vector

- + Distributed (small tables)
- + No flooding (fewer messages)

- – Slower convergence
- – Larger window of inconsistency

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) (sort of)