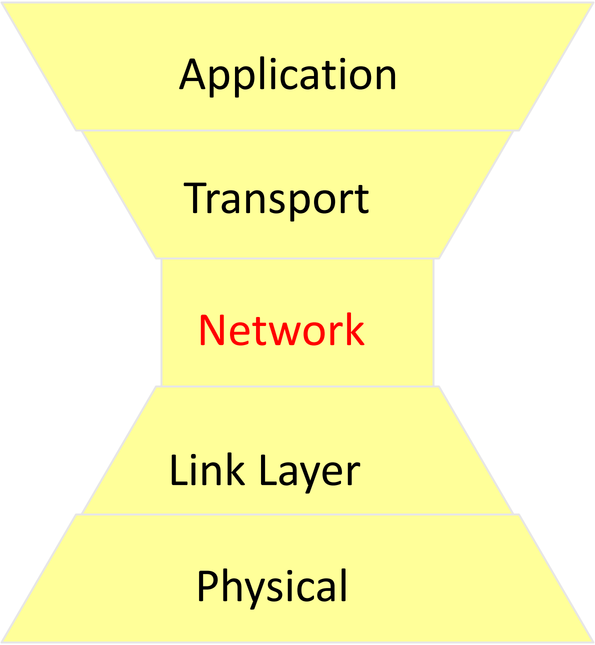
IP Routing: Intradomain

CS/ECE 438: Spring 2014

Instructor: Matthew Caesar

http://courses.engr.illinois.edu/cs438/

Today

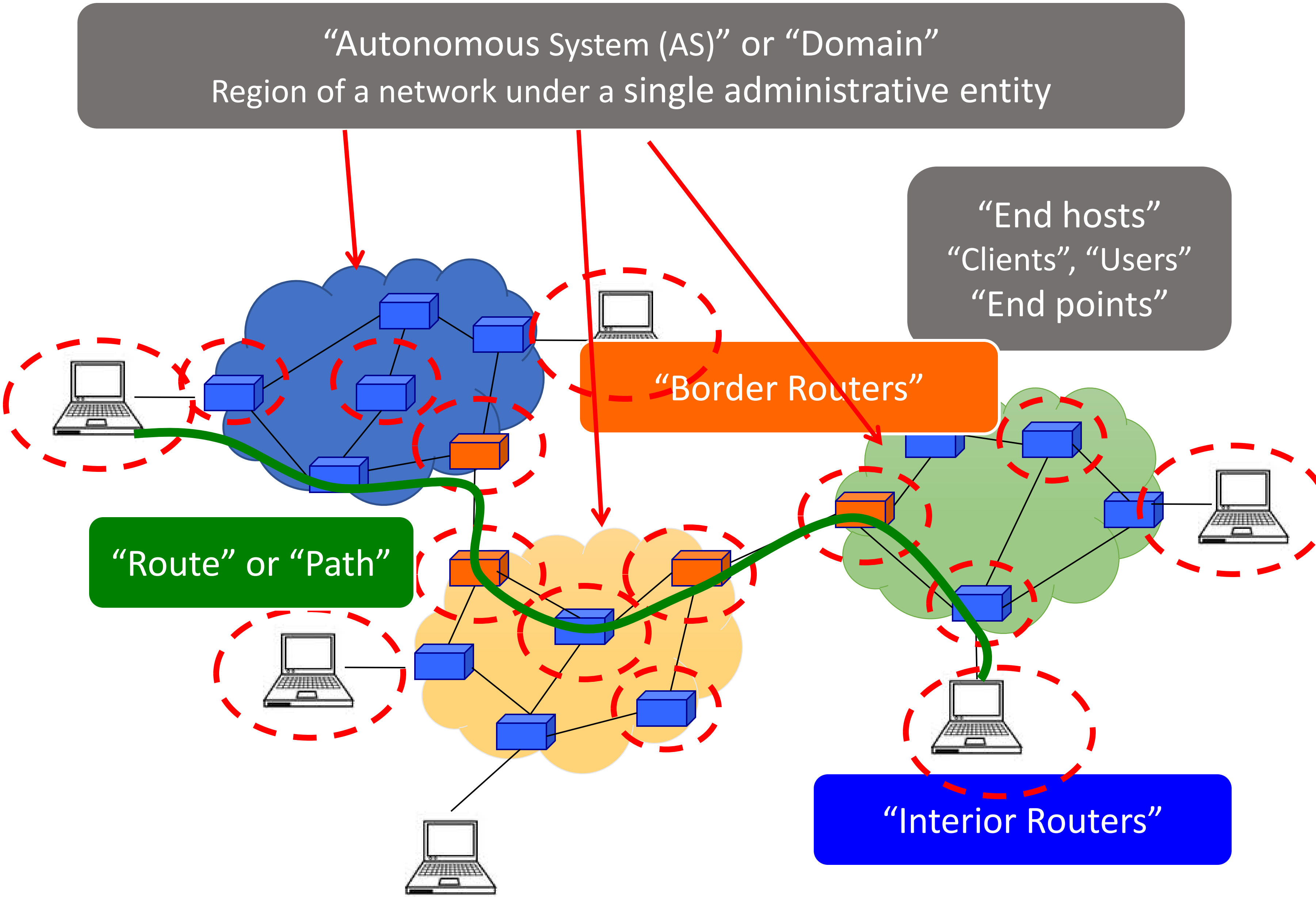


Starting on the internals of the network layer

Many pieces to the network layer

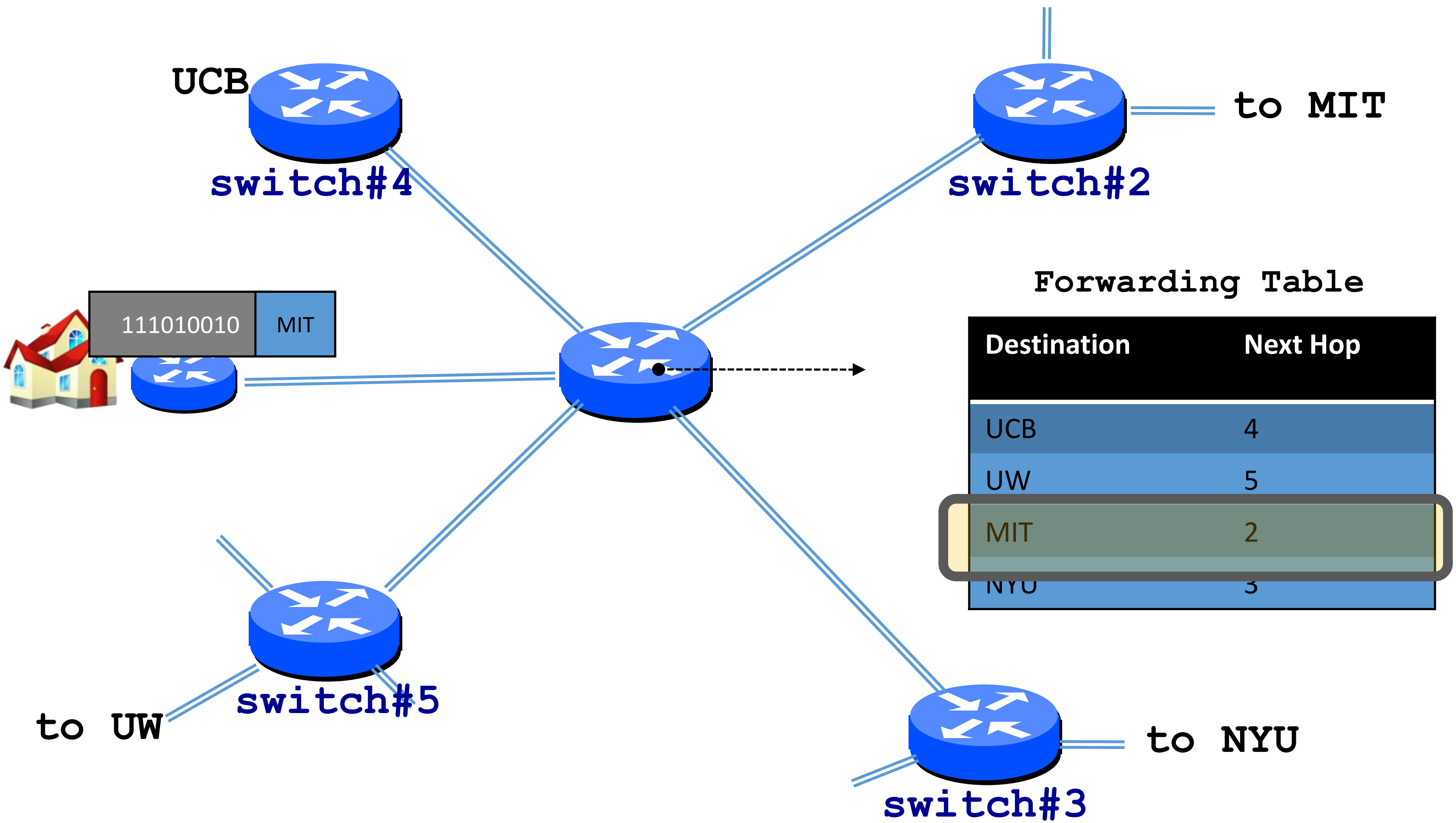
- Addressing
- Routing
- Forwarding
- Policy and management
- IP protocol details
- …

Today + next 1-2 lectures: Routing



“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”

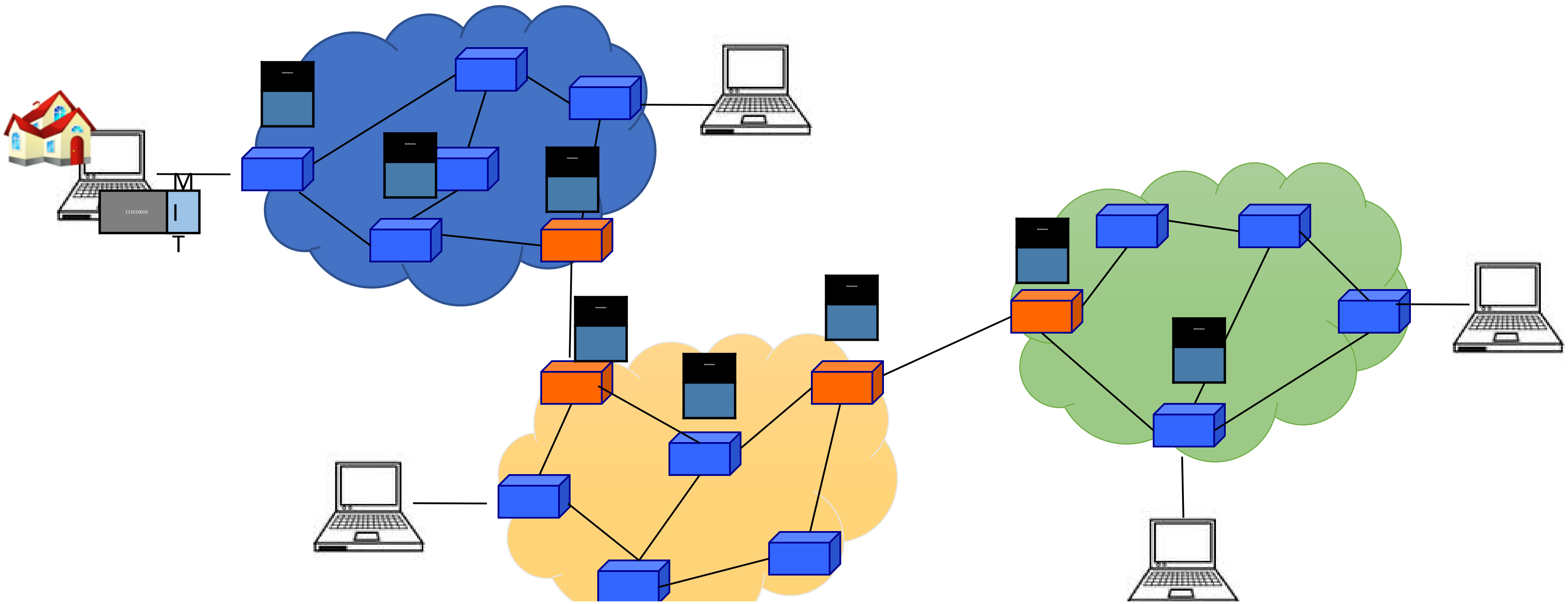
Lecture#2: Routers Forward Packets



Forwarding Table

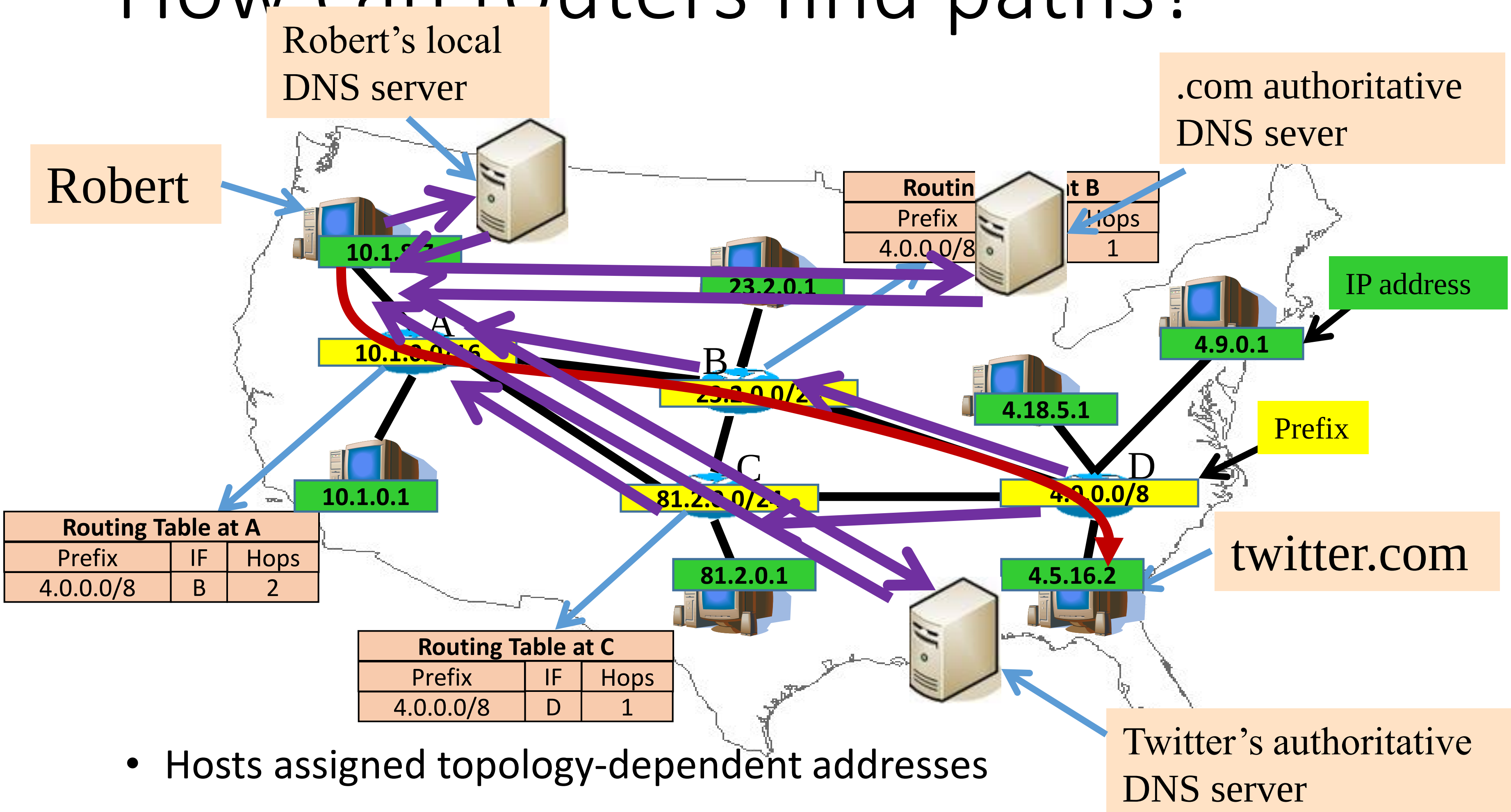
Destination	Next Hop
UCB	4
UW	5
MIT	2
NYU	3

Context and Terminology



Internet routing protocols are responsible for constructing and updating the forwarding tables at routers

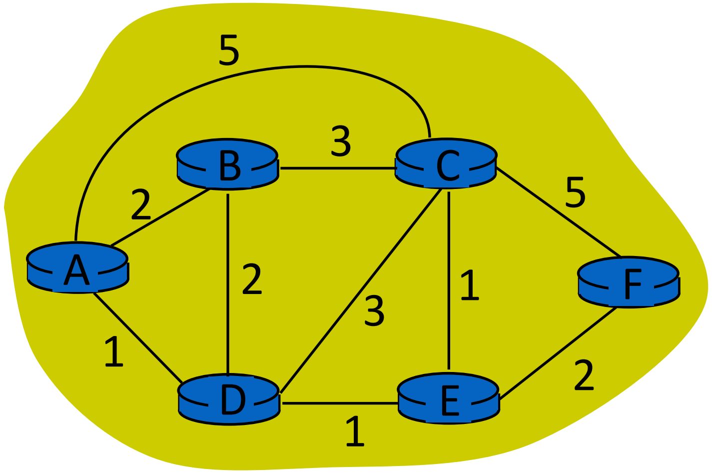
How can routers find paths?



- Hosts assigned topology-dependent addresses
- Routers advertise address blocks ("prefixes")
- Routers compute "shortest" paths to prefixes
- Map IP addresses to names with DNS

Routing Protocols

- Routing protocols implement the core function of a network
  - Establish paths between nodes
  - Part of the network's "control plane"
- Network modeled as a graph
  - Routers are graph vertices
  - Links are edges
  - Edges have an associated "cost"
    - e.g., distance, loss

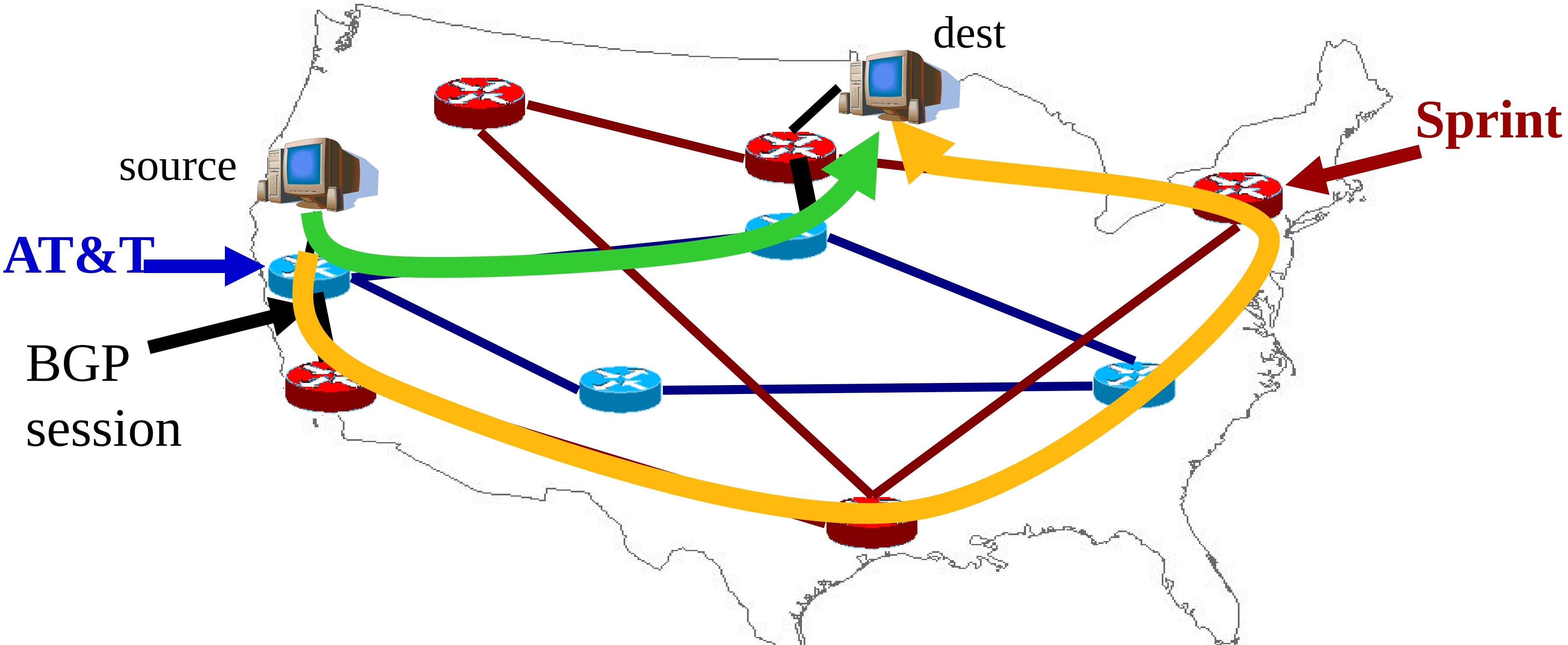


- Goal: compute a "good" path from source to destination
  - "good" usually means the shortest (least cost) path

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)

Intra- vs. Inter-domain routing



- Run “Interior Gateway Protocol” (IGP) within ISPs
  - OSPF, IS-IS, RIP
- Use “Border Gateway Protocol” (BGP) to connect ISPs
  - To reduce costs, peer at exchange points (AMS-IX, MAE-EAST)

Complete Network Assets : XO Communications

Addressing (for now)

- Assume each host has a unique ID (address)
- No particular structure to those IDs
- Later in course will talk about real IP addressing

Outline

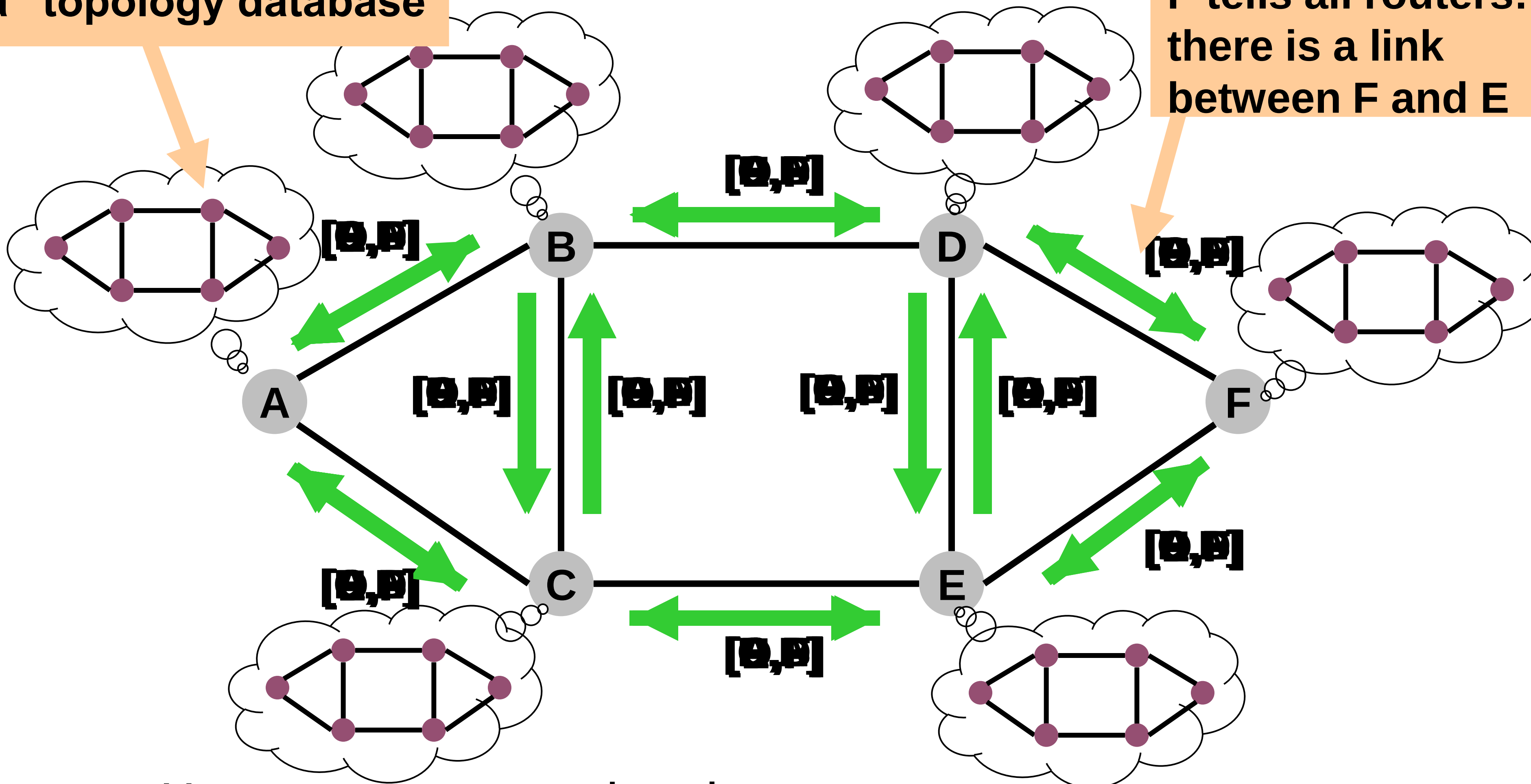
- Link State
- Distance Vector
- Routing: goals and metrics (if time)

Link-State Routing

: update propagation

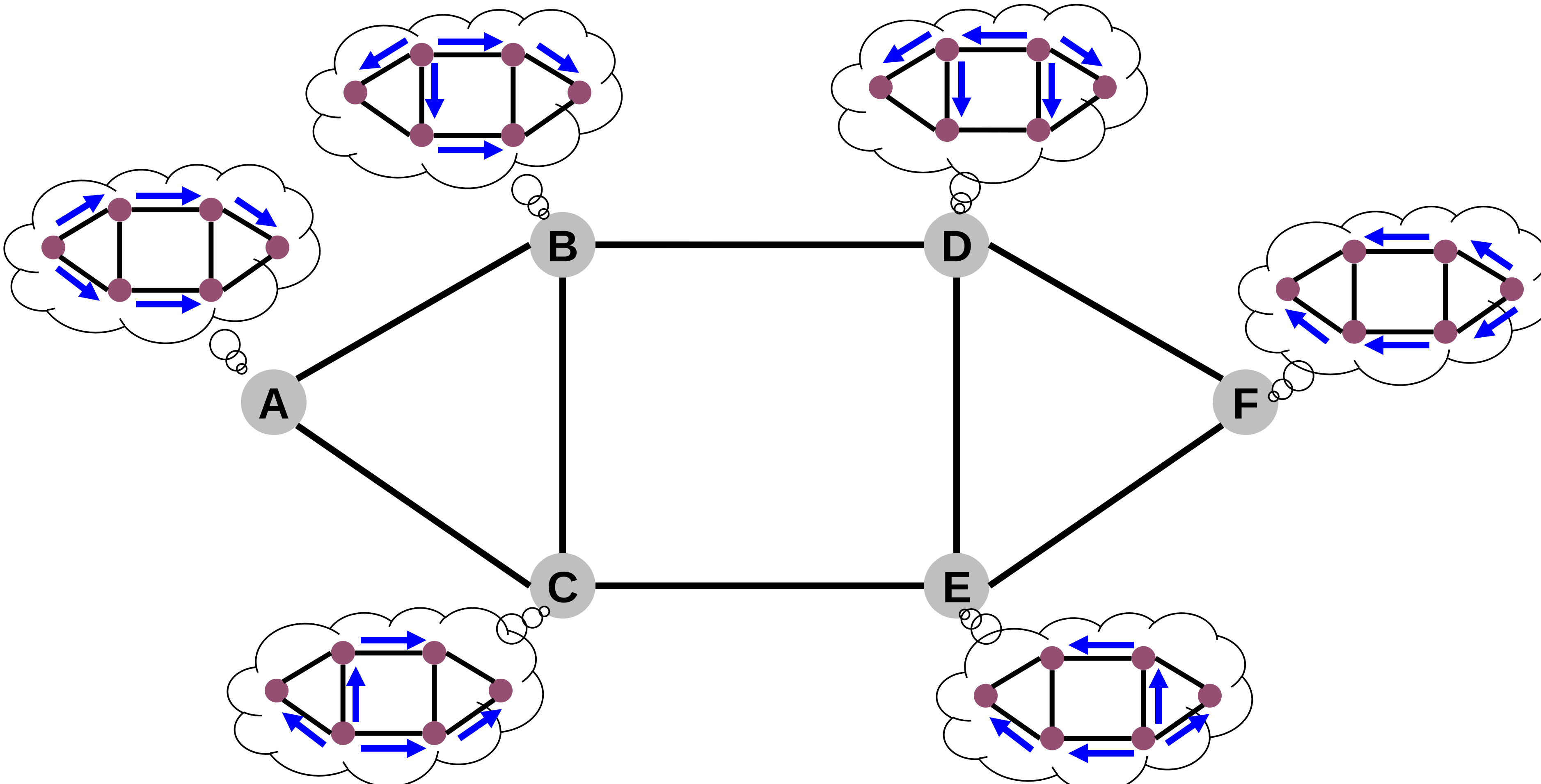
Each node maintains a "topology database"

F tells all routers: there is a link between F and E



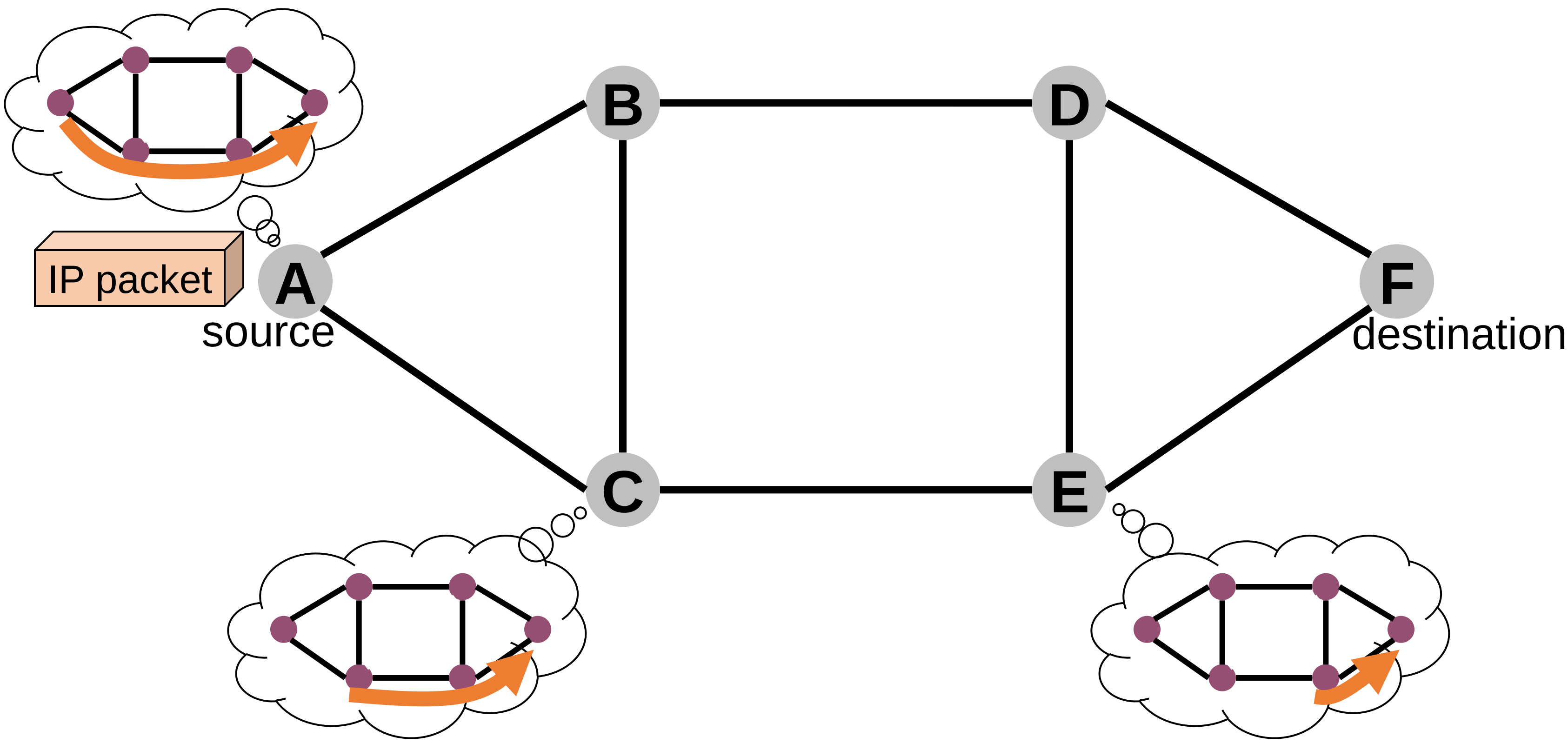
- How to prevent update loops:
- How to bring up new node:

Link state: route computation



- Each router computes shortest path tree, rooted at that router
- Determines next-hop to each dest, publish to forwarding table
- Operators can assign link costs to control path selection

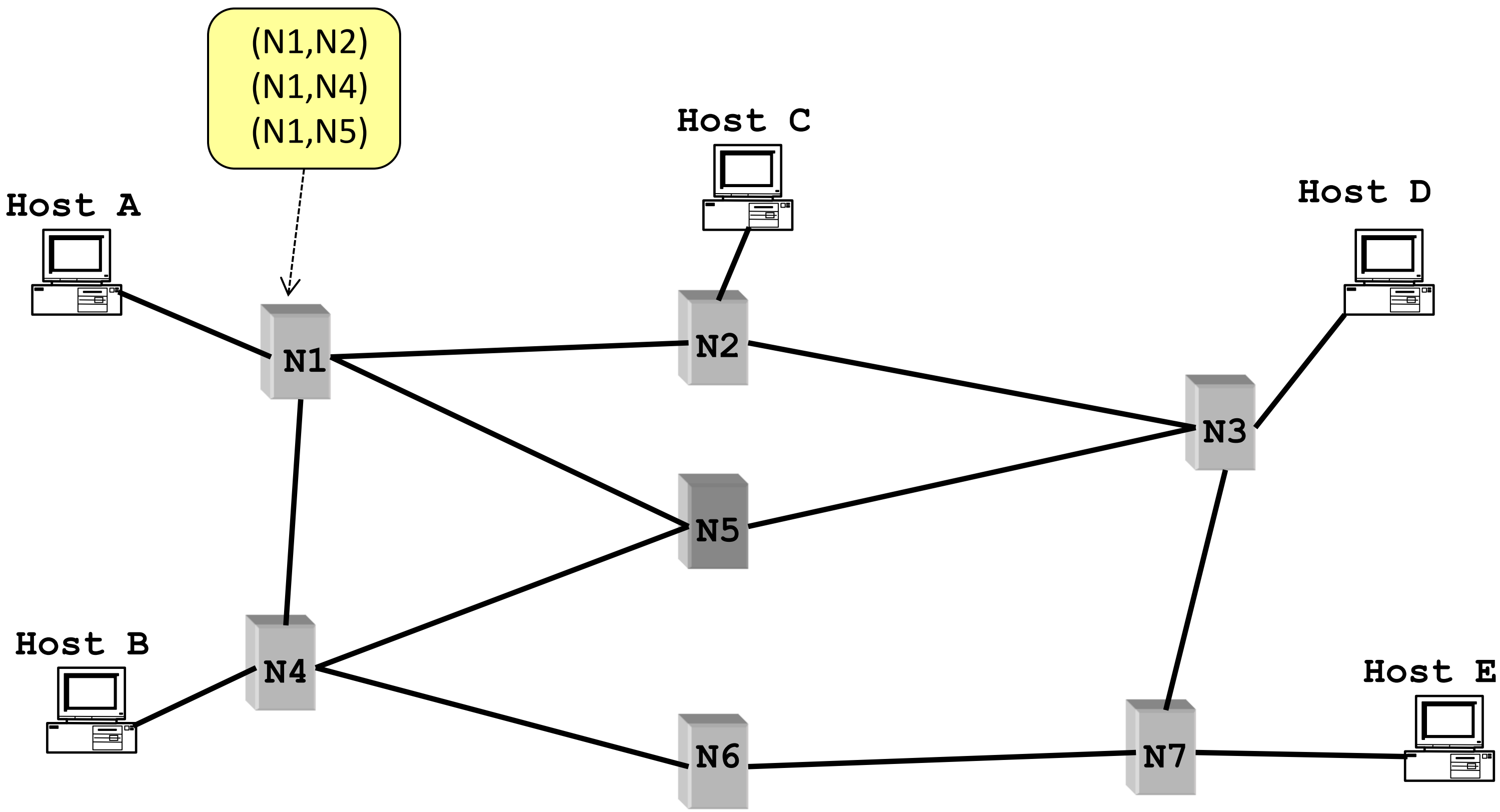
Link-state: packet forwarding



- In practice: shortest path precomputed, next-hops stored in forwarding table
- Downsides of link-state:
  - Lesser control on policy (certain routes can't be filtered), more cpu
  - Increased visibility (bad for privacy, but good for diagnostics)

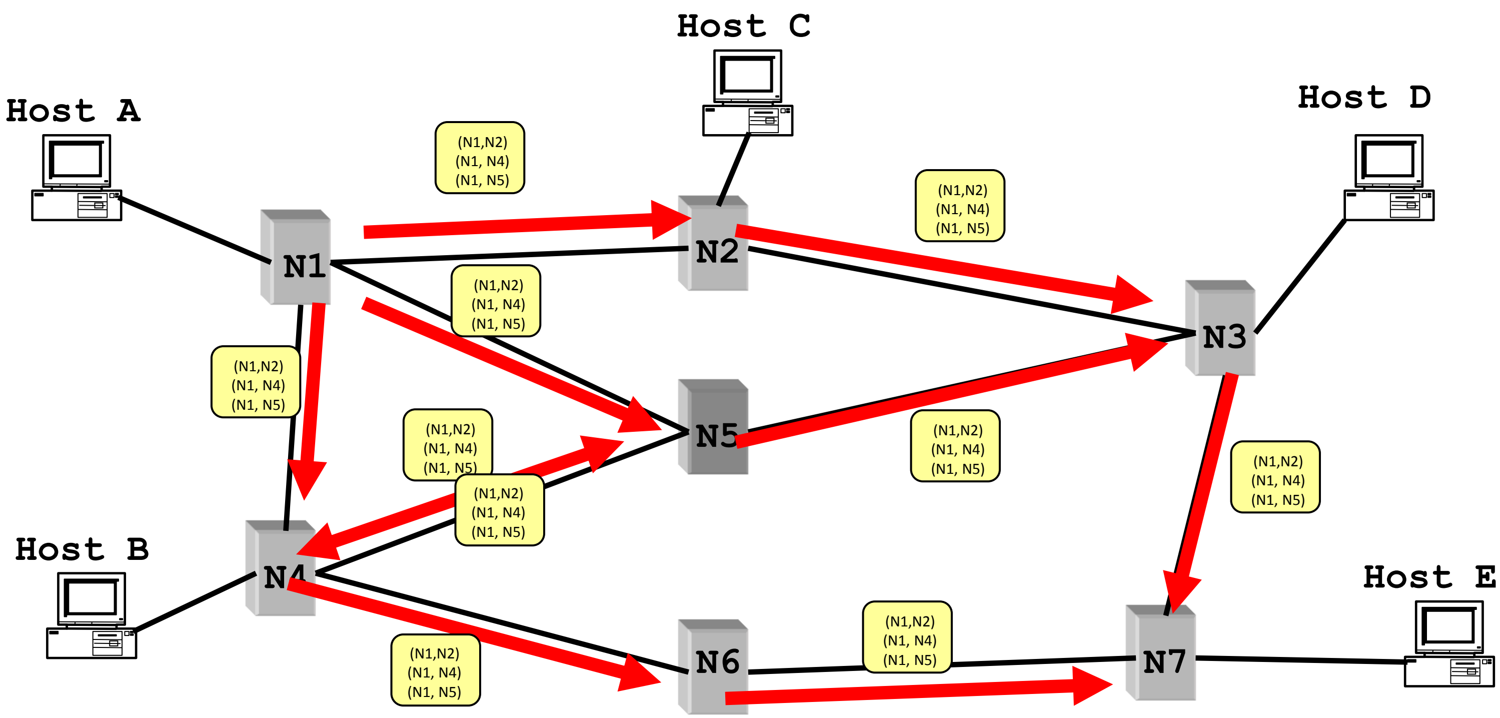
Link State Routing

- Each node maintains its local "link state" (LS)
  - i.e., a list of its directly attached links and their costs



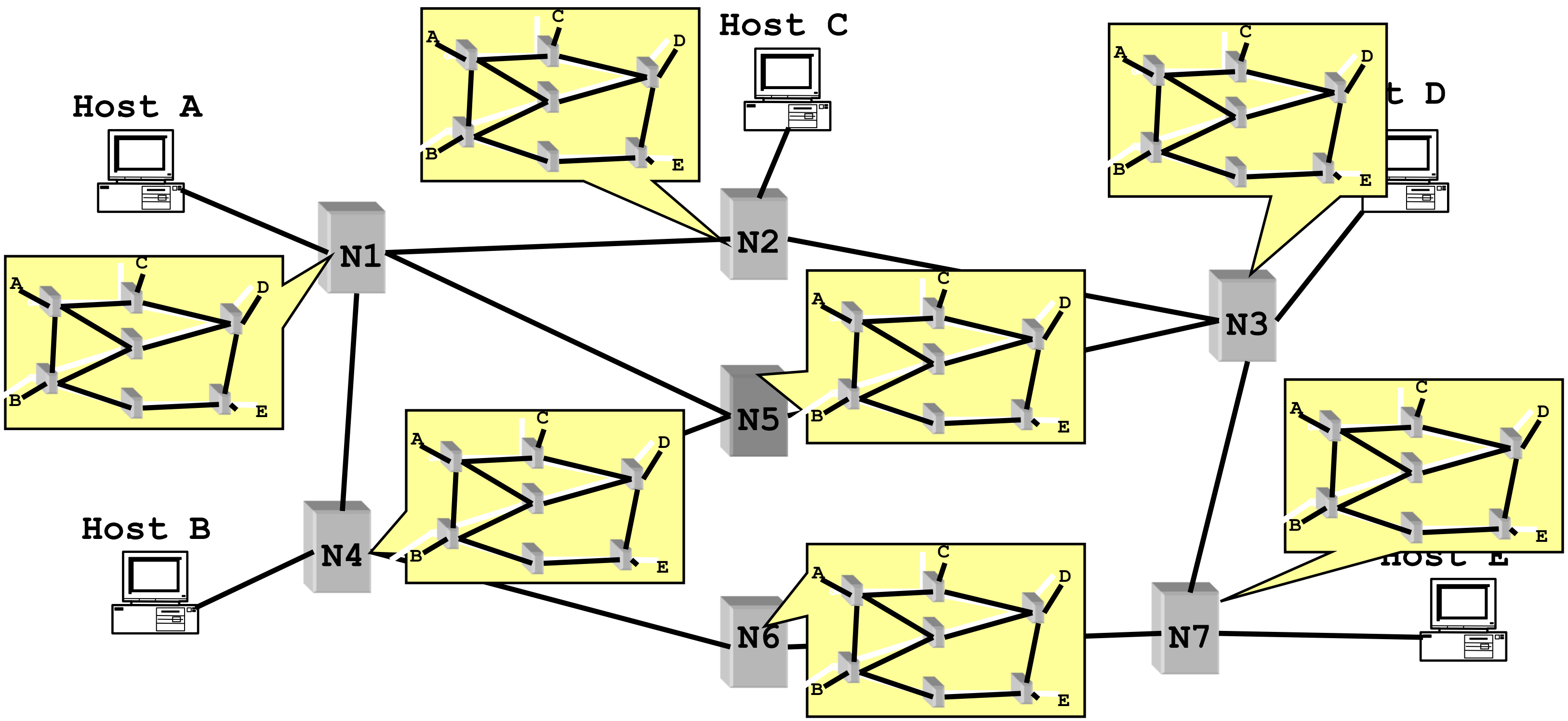
Link State Routing

- 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



Link State Routing

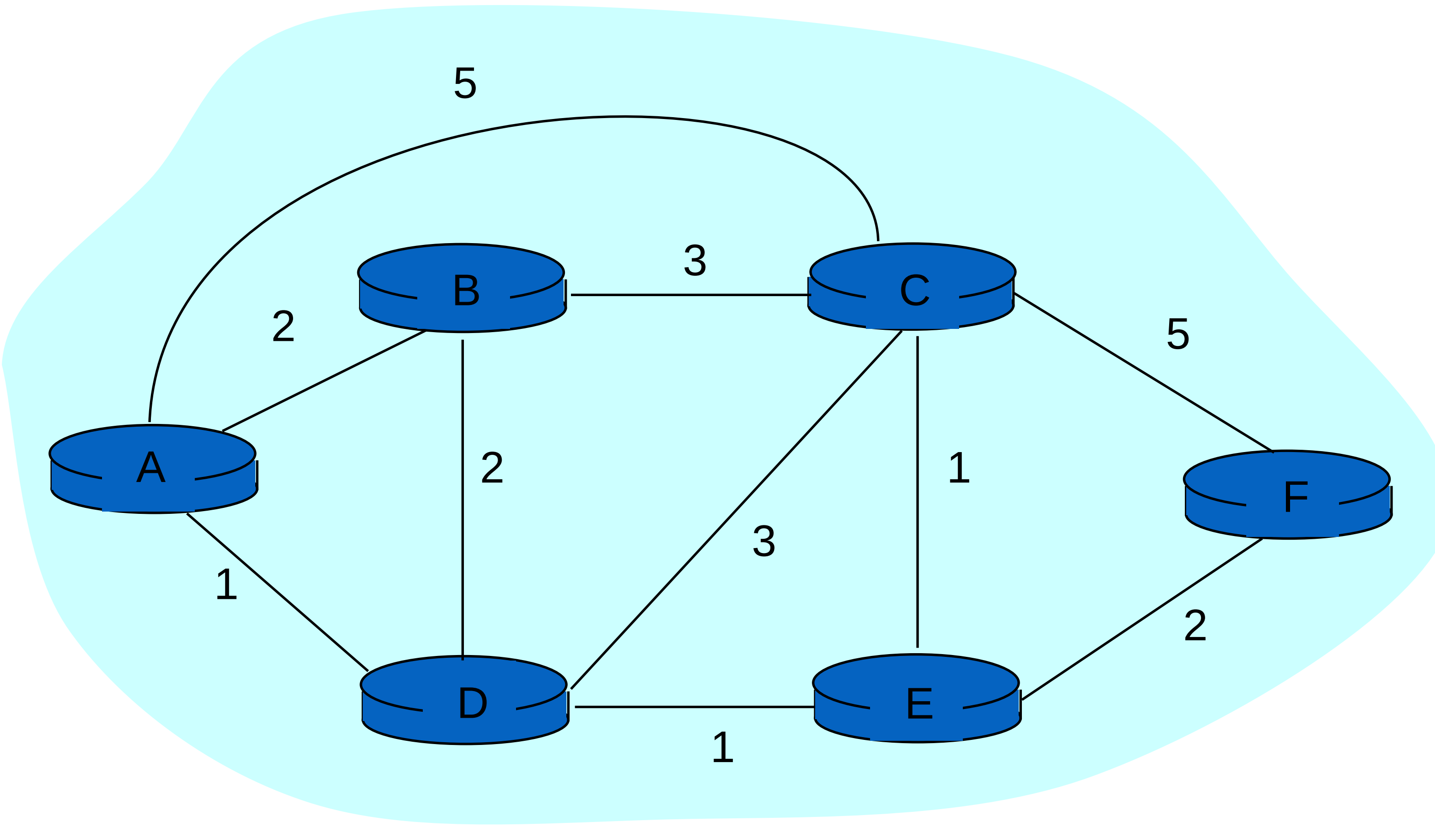
- Each node maintains its local "link state" (LS)
- Each node floods its local link state
- Hence, each node learns the entire network topology
  - Can use Dijkstra's to compute the shortest paths between nodes



Dijkstra’s Shortest Path Algorithm

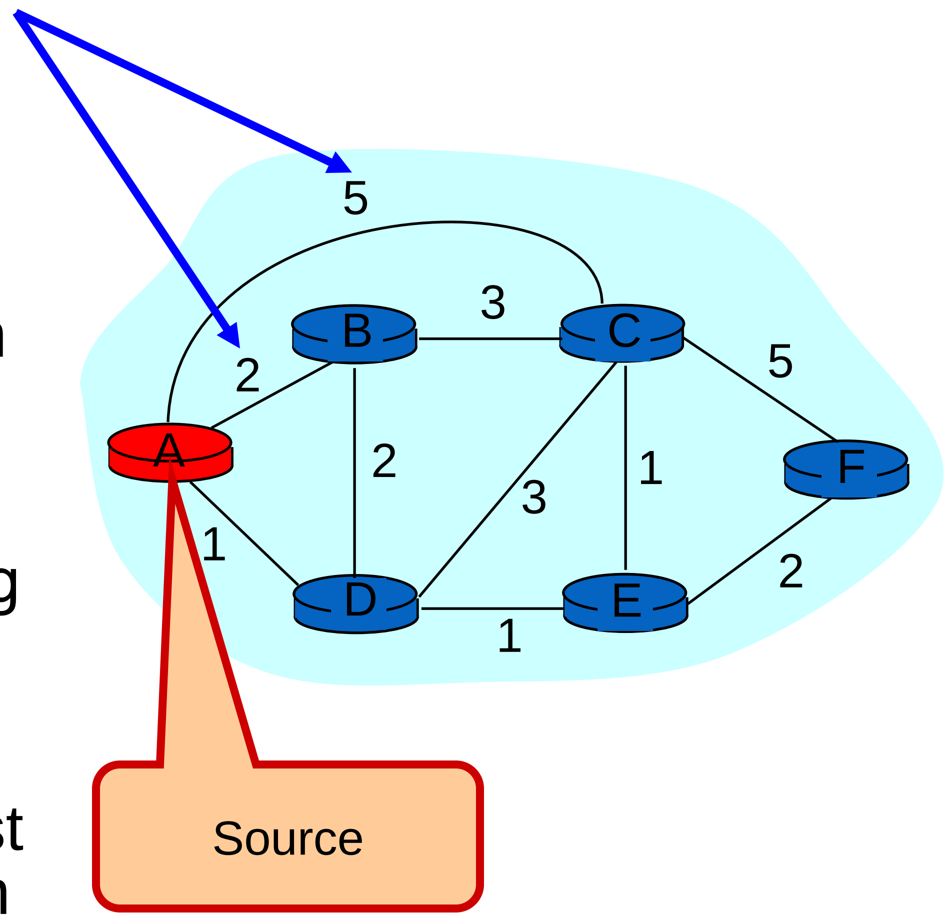
- INPUT:
  - Network topology (graph), with link costs
- OUTPUT:
  - Least cost paths from one node to all other nodes
- Iterative: after *k* iterations, a node knows the least cost path to its *k* closest neighbors

Example



Notation

- c(i,j): link cost from node *i* to *j*; cost is infinite if not direct neighbors; **≥ 0**
- D(v): total cost of the current least cost path from source to destination *v*
- p(v): *v*'s predecessor along path from source to *v*
- S: set of nodes whose least cost path definitively known



Dijkstra's Algorithm

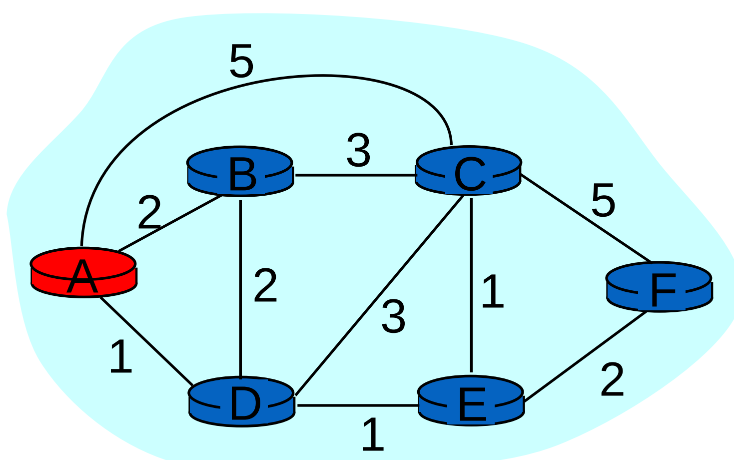
- c(i,j): link cost from node *i* to *j*
- D(v): current cost source → *v*
- p(v): *v*'s predecessor along path from source to *v*
- S: set of nodes whose least cost path definitively known

```
Initialization:
  S = {A};
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v);
      else D(v) = ¥;

Loop
    find w not in S such that D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent to w and not in S:
       if  D(w) + c(w,v) < D(v) then
         // w gives us a shorter path to v than we've found so far
         D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra’s Algorithm

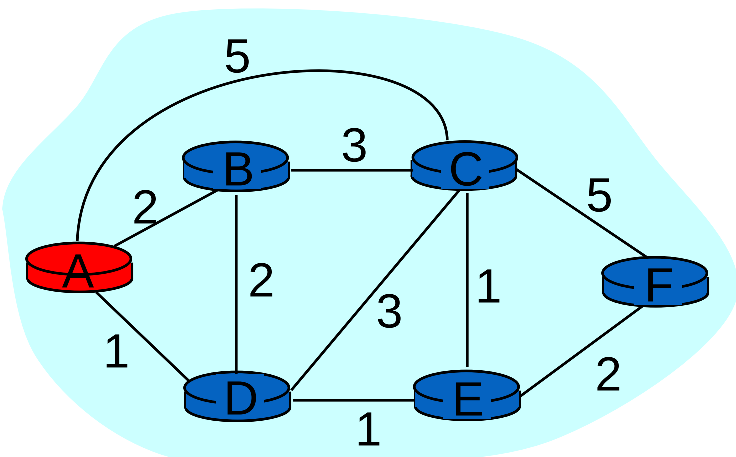
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	¥	¥
1						
2						
3						
4						
5						



Initialization:
  S = {A};
  for all nodes ***v***
    if ***v*** adjacent to ***A***
      then D(v) = c(A,v);
      else D(v) = **¥**;
…

Example: Dijkstra's Algorithm

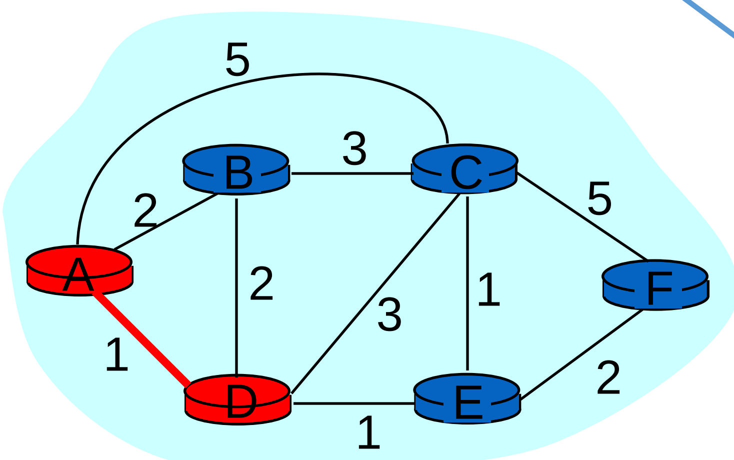
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	¥	¥
1						
2						
3						
4						
5						




```
…
Loop
  find w not in S s.t. D(w) is a minimum;
  add w to S;
  update D(v) for all v adjacent
    to w and not in S:
  If D(w) + c(w,v) < D(v) then
     D(v) = D(w) + c(w,v); p(v) = w;
 until all nodes in S;
```

Example: Dijkstra’s Algorithm

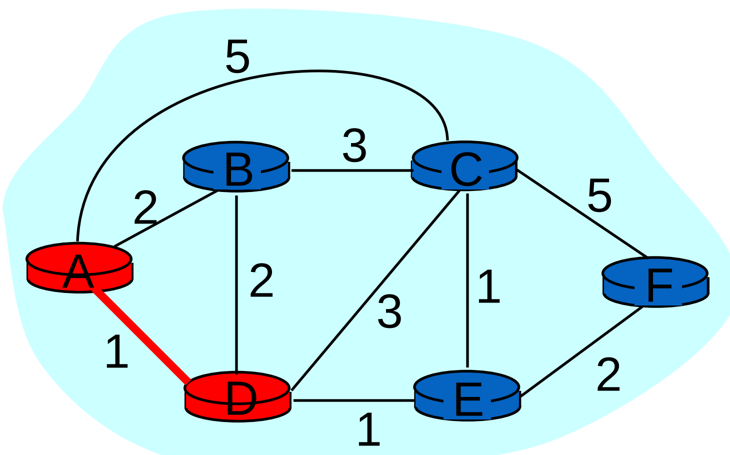
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	¥	¥
1	AD					
2						
3						
4						
5						




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
    until all nodes in S;
```

Example: Dijkstra’s Algorithm

Step	set S	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						



…

Loop

find **w** not in **S** s.t. D(w) is a minimum;

add **w** to **S**;

update D(v) for all **v** adjacent to **w** and not in **S**:

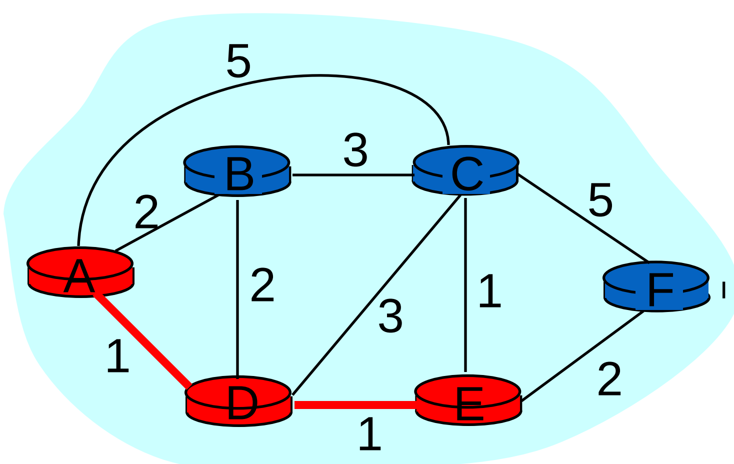
If D(w) + c(w,v) < D(v) then

D(v) = D(w) + c(w,v); p(v) = w;

until all nodes in S;

Example: Dijkstra's Algorithm

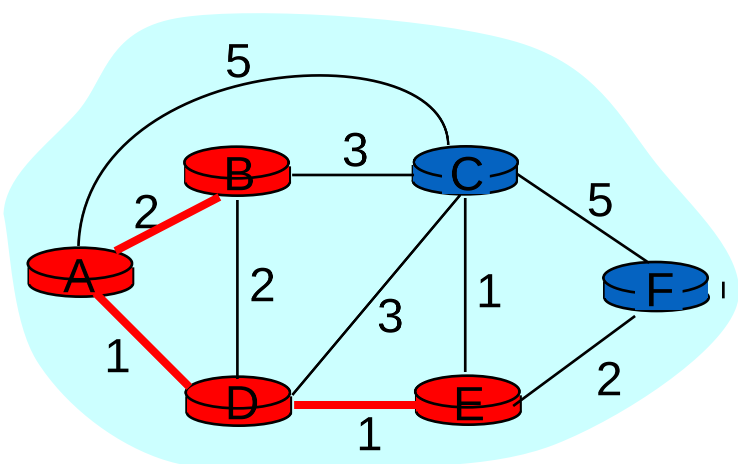
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	¥	¥
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3						
4						
5						




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
    until all nodes in S;
```

Example: Dijkstra’s Algorithm

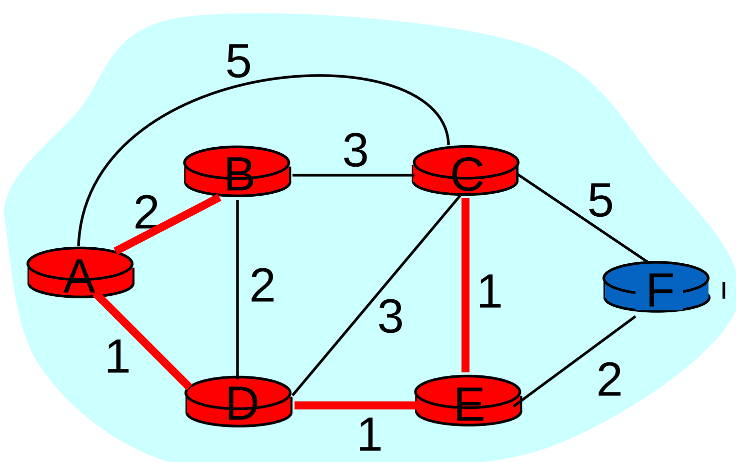
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	¥	¥
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4						
5						




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
    until all nodes in S;
```

Example: Dijkstra's Algorithm

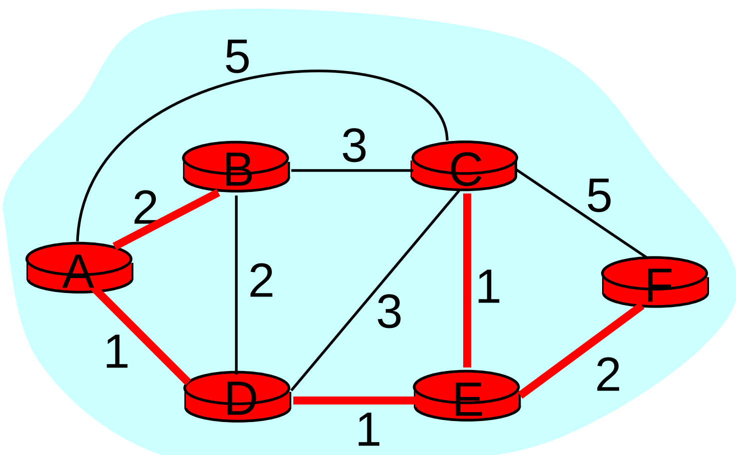
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	¥	¥
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5						




```
…
Loop
   find w not in S s.t. D(w) is a minimum;
   add w to S;
  update D(v) for all v adjacent
     to w and not in S:
  If D(w) + c(w,v) < D(v) then
      D(v) = D(w) + c(w,v); p(v) = w;
   until all nodes in S;
```

Example: Dijkstra’s Algorithm

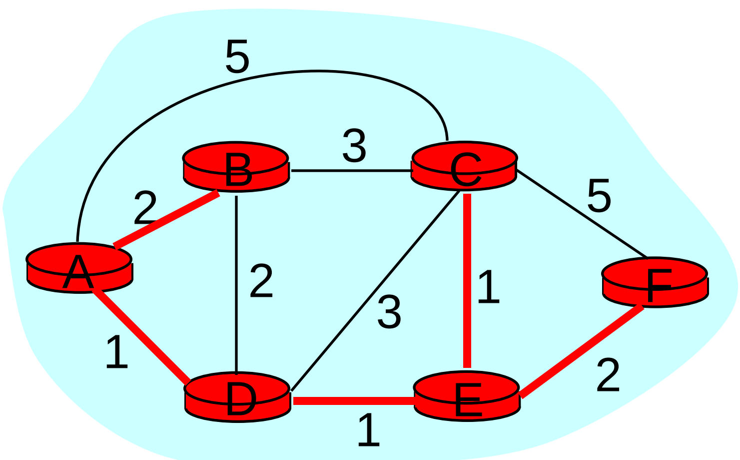
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	¥	¥
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
      D(v) = D(w) + c(w,v); p(v) = w;
    until all nodes in S;
```

Example: Dijkstra’s Algorithm

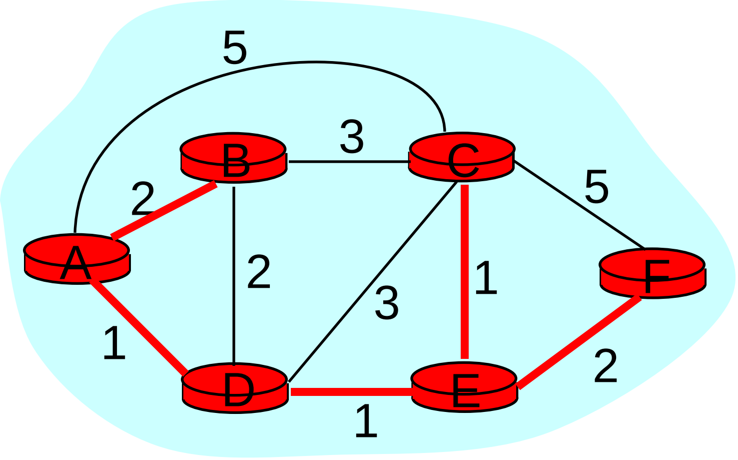
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	¥	¥
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					



To determine path A $\rightarrow$ 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*



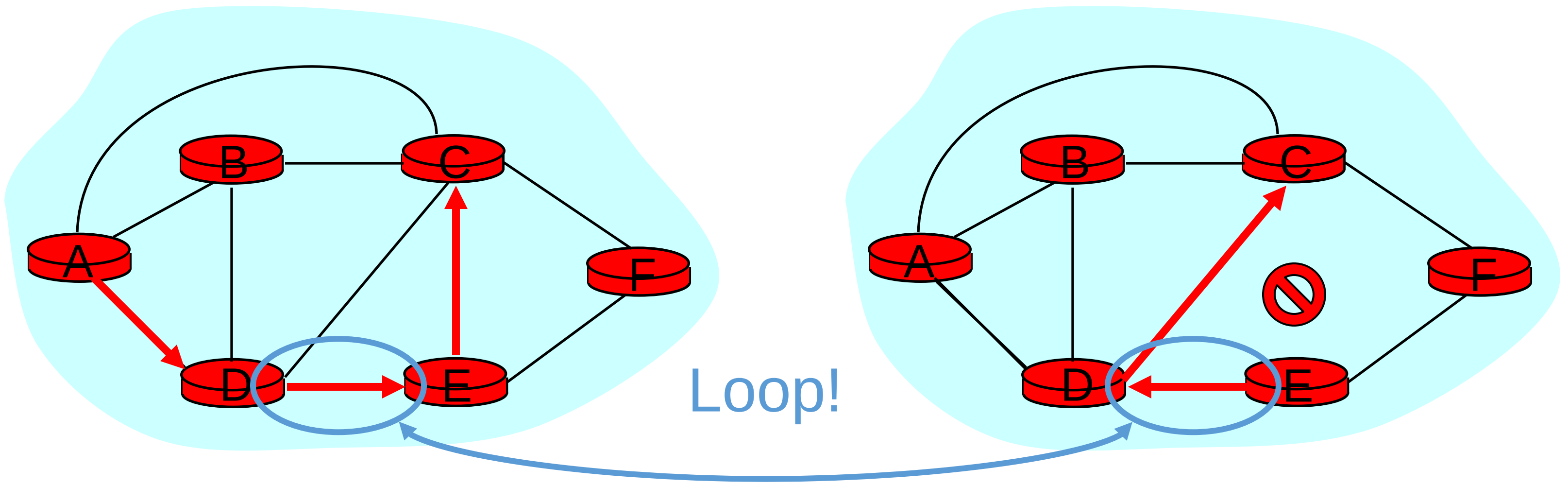
Destination	**Link**
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

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 S at each iteration and we have O(N) iterations
  - more efficient implementations: O(N log(N))
- How many entries in the LS topology database? O(E)
- How many entries in the forwarding table? O(N)

Issue#2: Transient Disruptions

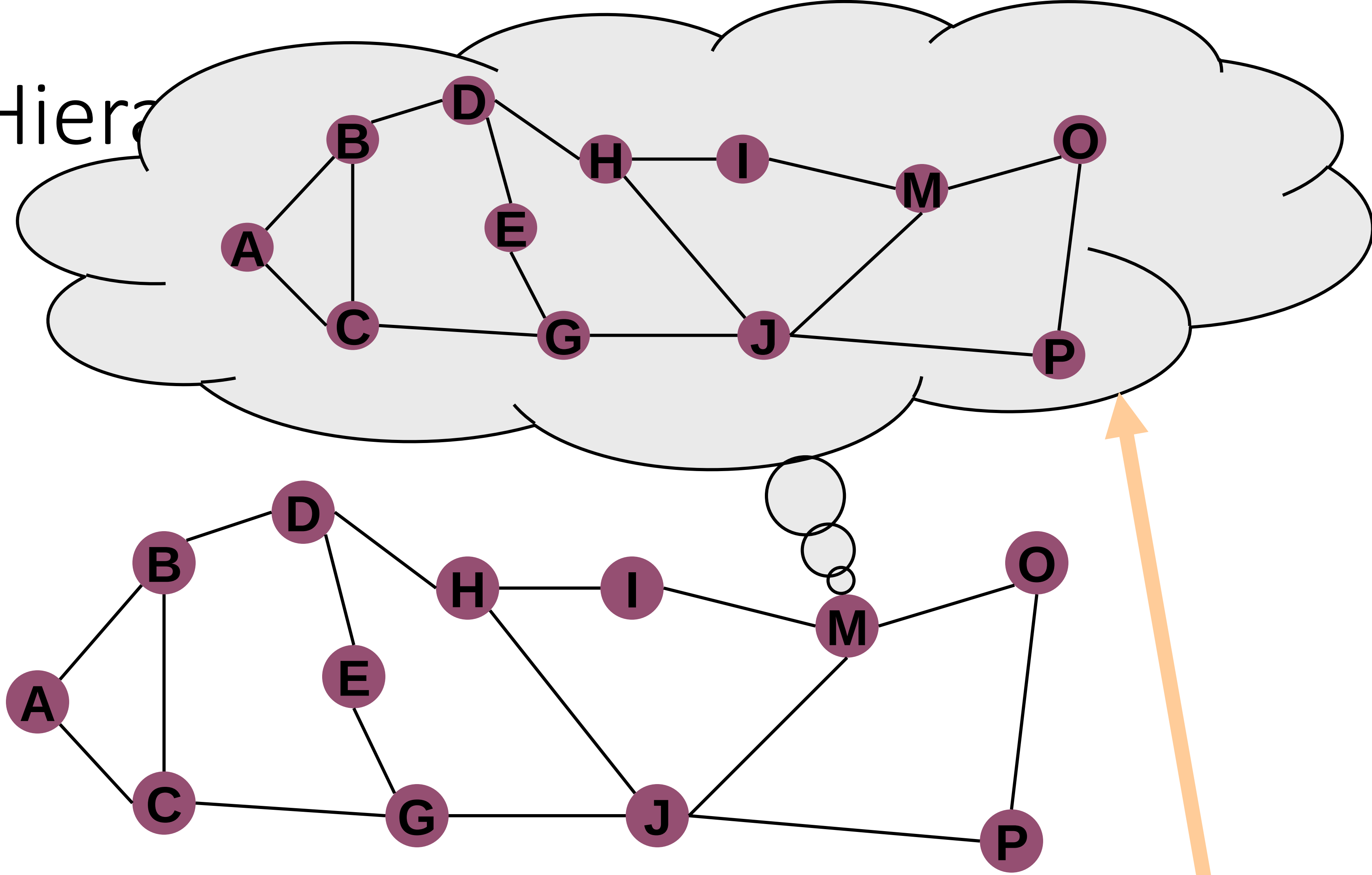
- Inconsistent link-state database
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - Can cause transient forwarding loops



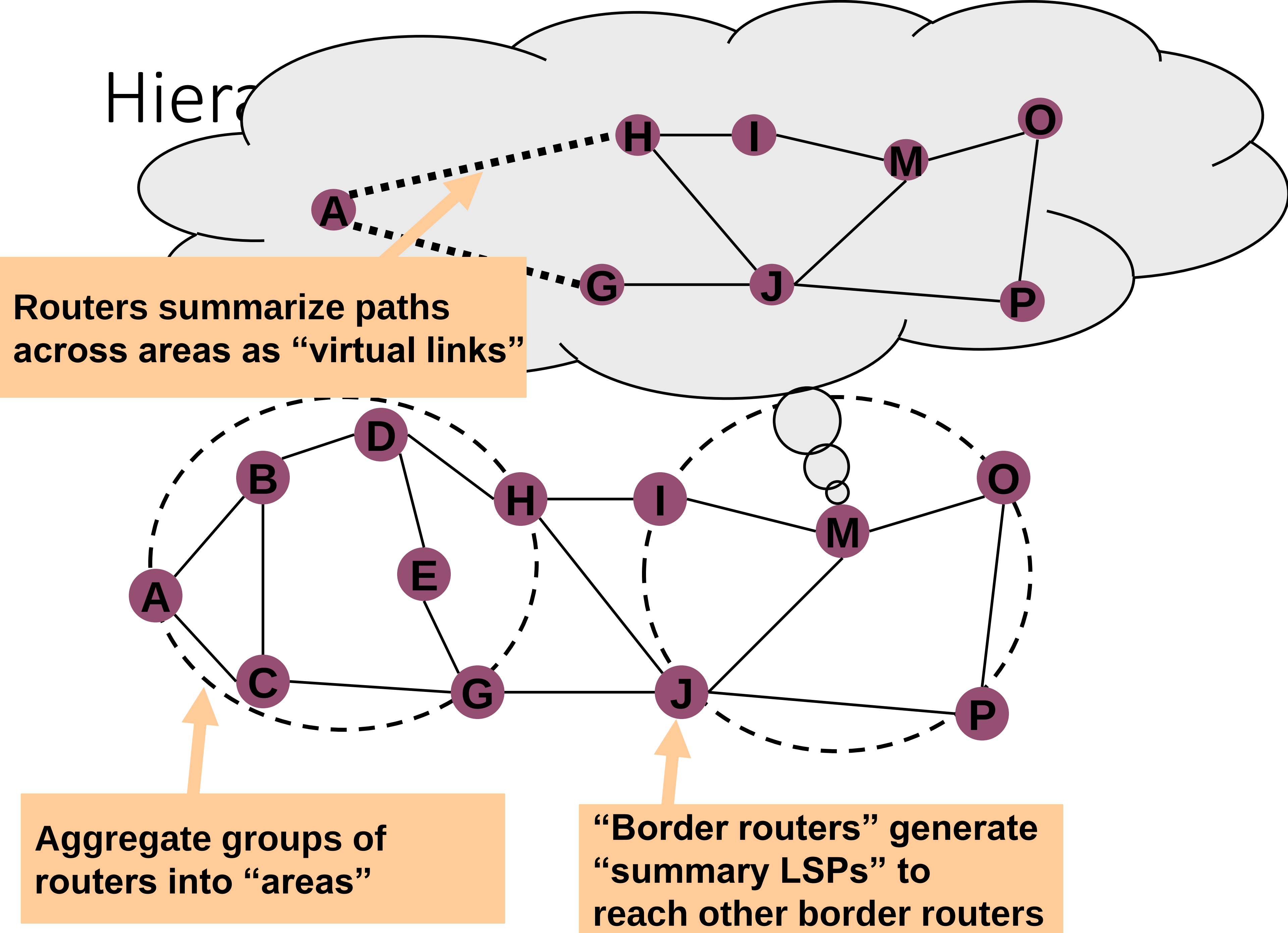
A and D think that this is the path to C

E thinks that this is the path to C

Hiera



In regular link-state, routers maintain map of entire topology



Hiera
H
I
O
M
A
G
J
P
Routers summarize paths across areas as “virtual links”
D
B
H
I
O
M
E
A
C
G
J
P
Aggregate groups of routers into “areas”
“Border routers” generate “summary LSPs” to reach other border routers

Distance Vector

Learn-By-Doing

Let’s try to collectively develop
distance-vector routing from first principles

Experiment

- Your job: find the youngest person in the room

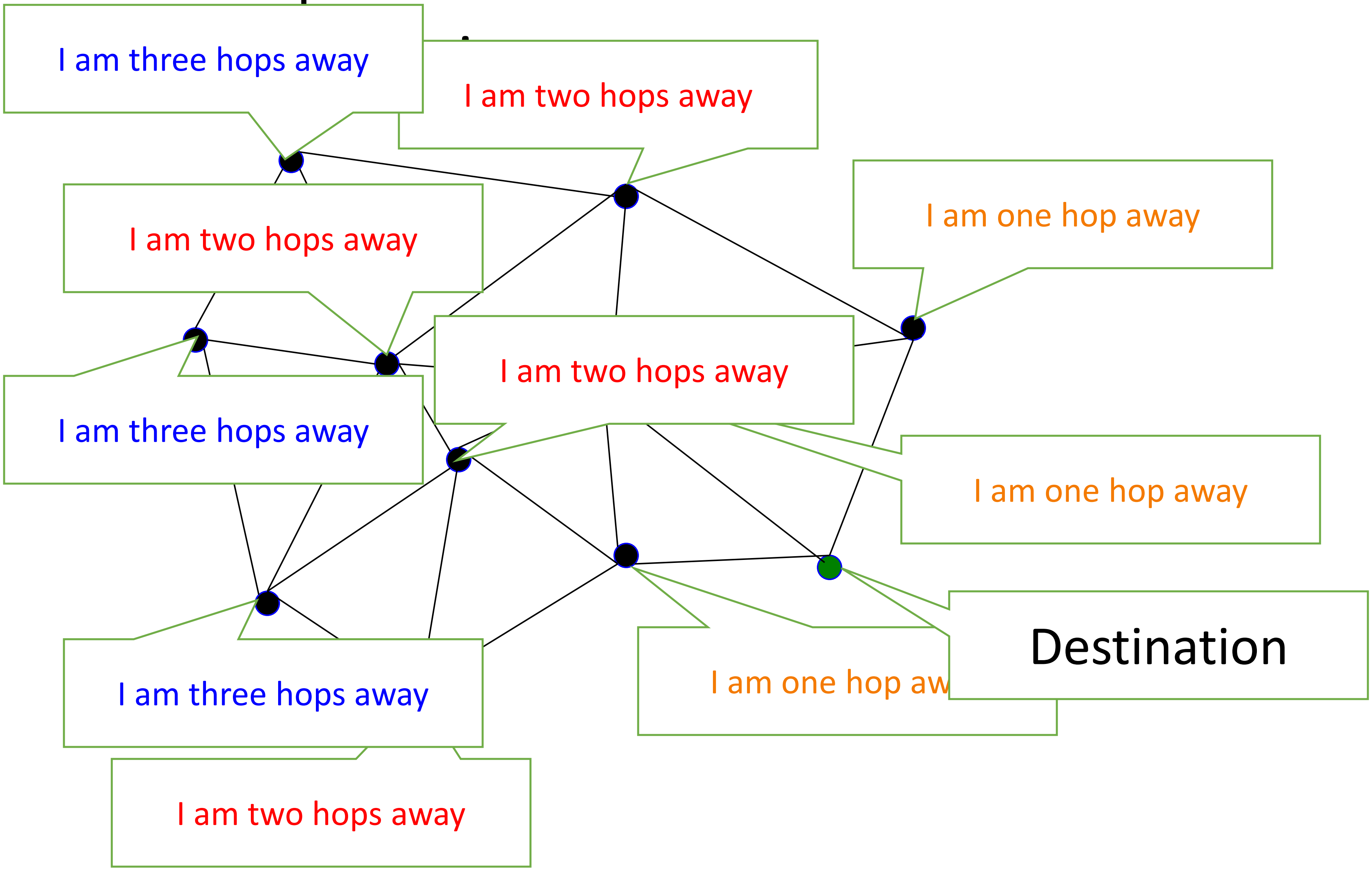
- Ground Rules
  - **You may not** leave your seat, nor shout loudly across the class
  - **You may** talk with your immediate neighbors (hint: “exchange updates” with them)

- At the end of 5 minutes, I will pick a victim and ask:
  - who is the youngest person in the room? (name, date)
  - which one of your neighbors first told you this info.?

Go!

Distance-Vector

Example of Distributed



Distance vector: update propagation

D tells B: I am D, and I can reach F via 1 hop

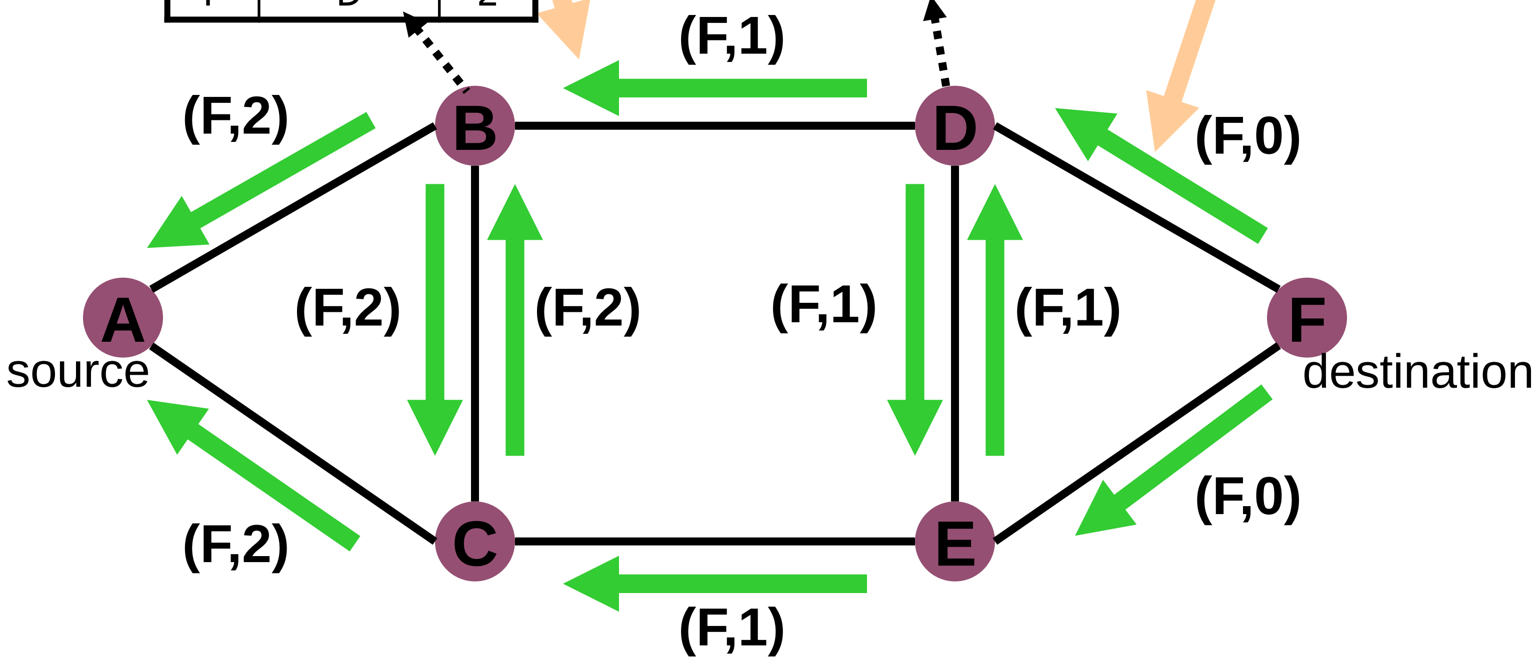
F tells D: I am F, and I can reach F via 0 hops

B's forwarding table

Dest	NextHop	Dist
F	D	2

D's forwarding table

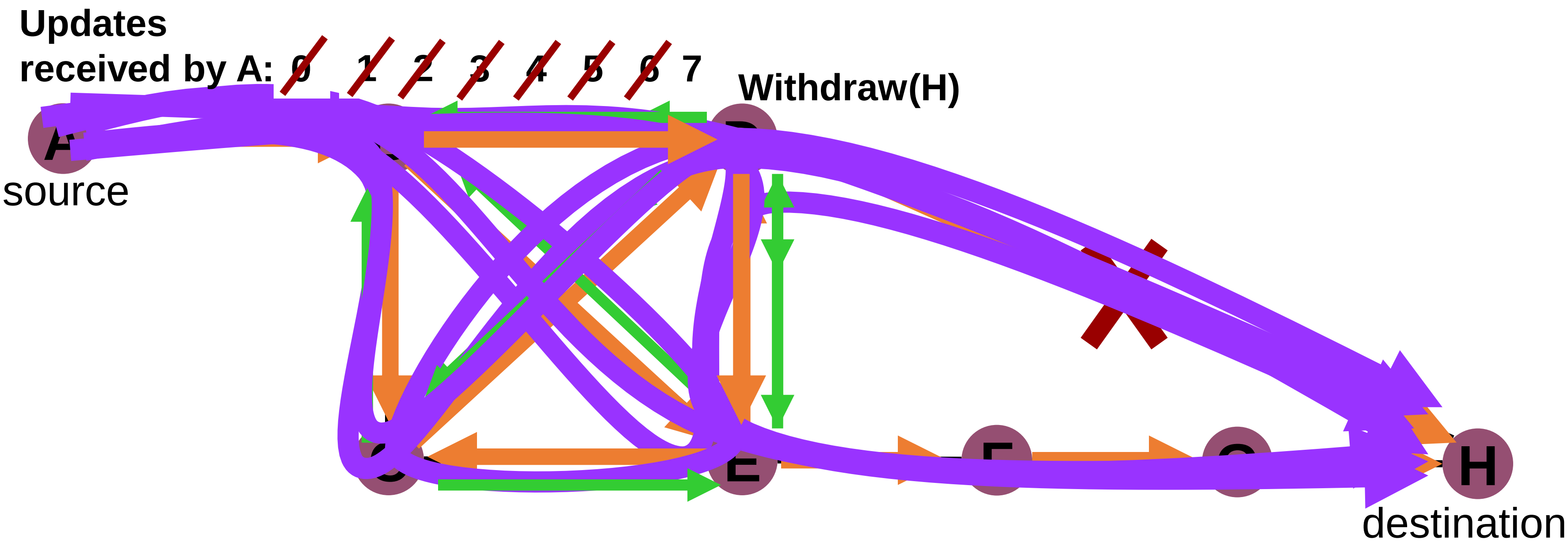
Dest	NextHop	Dist
F	F	1



Distance Vector Routing

- Each router knows the links to its neighbors
  - Does *not* flood this information to the whole network
- Each router has provisional “shortest path” to every other router
  - E.g.: Router A: “I can get to router B with cost 11”
- Routers exchange this distance vector information with their neighboring routers
  - Vector because one entry per destination
- 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: convergence



- How many updates would link-state require?
- Is link-state better or worse than distance vector?
- Which should be used for intra-domain routing? What about inter-domain routing?

Bellman-Ford Algorithm

- INPUT:
  - Link costs to each neighbor
    (*Not full topology*)
- OUTPUT:
  - Next hop to each destination and the corresponding cost
    (*Not the complete path to the destination*)

- My neighbors tell me how far they are from dest’n
  - Compute: (cost to nbr) plus (nbr’s cost to destination)
  - Pick minimum as my choice
  - Advertise that cost to my neighbors

Bellman-Ford Overview

- Each router maintains a table
  - Best known distance from X to Y, via Z as next hop = $D_Z(X,Y)$
- Each local iteration caused by:
  - Local link cost change
  - Message from neighbor
- Notify neighbors *only* if least cost path to any destination changes
  - Neighbors then notify their neighbors if necessary

Each node:

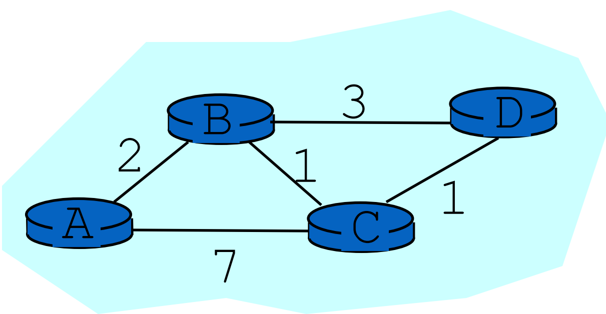
wait for (change in local link cost or msg from neighbor)

recompute distance table

if least cost path to any dest has changed, *notify* neighbors

Bellman-Ford Overview

- Each router maintains a table
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
  - Entry in row Y and column Z of node X $\Rightarrow$ best known distance from X to Y, via Z as next hop = $D_Z(X,Y)$



Node A

	B	C
B	2	8
C	3	7
D	4	8

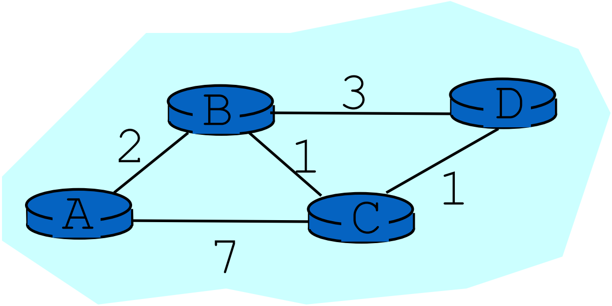
Neighbor (next-hop)

Destinations

$D_C(A, D)$

Bellman-Ford Overview

- Each router maintains a table
  - Row for each possible destination
  - Column for each directly-attached neighbor to node
  - Entry in row Y and column Z of node X $\Rightarrow$ best known distance from X to Y, via Z as next hop = $D_Z(X,Y)$



Node A

	B	C
B	2	8
C	3	7
D	4	8

Smallest distance in row Y = shortest Distance of A to Y, D(A, Y)

Distance Vector Algorithm (cont’d)

- c(i,j): link cost from node *i* to *j*
- $D_Z(A,V)$: cost from A to V via Z
- D(A,V): cost of A’s best path to V

```
Initialization:
  for all neighbors V do
      if V adjacent to A
          D(A, V) = c(A,V);
      else
          D(A, V) = ∞;
    send D(A, Y) to all neighbors
loop:
  wait (until A sees a link cost change to neighbor V  /* case 1 */
          or until A receives update from neighbor V)    /* case 2 */
   if (c(A,V) changes by ±d)  /* ⇐ case 1 */
          for all destinations Y that go through V do
              D_V(A,Y) = D_V(A,Y) ± d
   else if (update D(V, Y) received from V) /* ⇐ case 2 */
          /* shortest path from V to some Y has changed  */
          D_V(A,Y) = D_V(A,V) + D(V, Y);    /* may also change D(A,Y) */
   if (there is a new minimum for destination Y)
          send D(A, Y) to all neighbors
  forever
```

Distance Vector Algorithm (cont’d)

Each node: initialize, then

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

recompute distance table

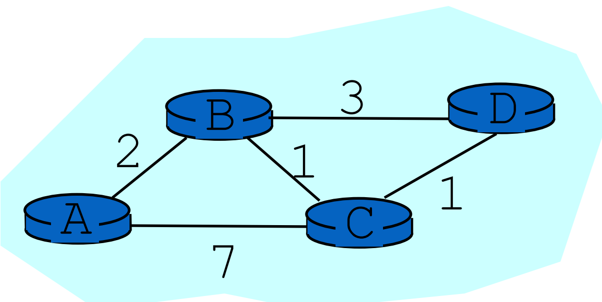
if least cost path to any dest has changed, *notify* neighbors

Distance Vector Algorithm (cont’d)

- c(i,j): link cost from node *i* to *j*
- $D_Z(A,V)$: cost from A to V via Z
- D(A,V): cost of A’s best path to V

Initialization:
  for all neighbors *V* **do**
    if *V* adjacent to *A*
      $D(A, V) = c(A,V)$;
    else
      $D(A, V) = \infty$;
  send $D(A, Y)$ to all neighbors
loop:
  wait (until *A* sees a link cost change to neighbor *V* /* **case 1** */
      or until *A* receives update from neighbor *V*) /* **case 2** */
  if ($c(A,V)$ changes by $\pm d$) /* $\Leftarrow$ **case 1** */
    for all destinations *Y* that go through *V* **do**
      $D_V(A,Y) = D_V(A,Y) \pm d$
  else if (update $D(V, Y)$ received from *V*) /* $\Leftarrow$ **case 2** */
    / shortest path from V to some Y has changed */*
    $D_V(A,Y) = D_V(A,V) + D(V, Y)$; /* ***may*** also change D(A,Y) */
  if (there is a new minimum for destination Y)
    send $D(A, Y)$ to all neighbors
  forever

Example: Initialization



Node A

	B	C
B	2	∞
C	∞	7
D	∞	∞

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

Initialization:

for all neighbors *V* **do**

  if *V* adjacent to *A*

    D(*A*, *V*) = c(*A*,*V*);

  else

    D(*A*, *V*) = ∞;

send D(*A*, *Y*) to all neighbors

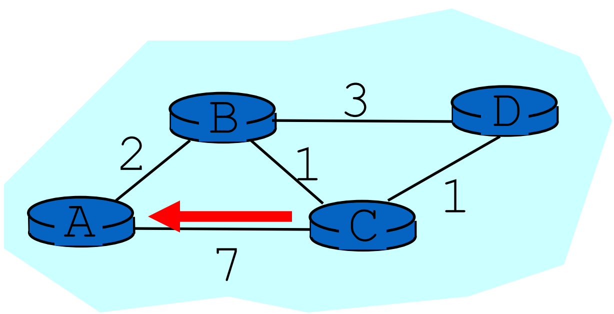
Node C

	A	B	D
A	7	∞	∞
B	∞	1	∞
D	∞	∞	1

Node D

	B	C
A	∞	∞
B	3	∞
C	∞	1

Example: C sends update to A



Node A

	B	C
B	2	8
C	∞	7
D	∞	8

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

$D_C(A, B) = D_C(A,C) + D(C, B) = 7 + 1 = 8$

$D_C(A, D) = D_C(A,C) + D(C, D) = 7 + 1 = 8$

Node C

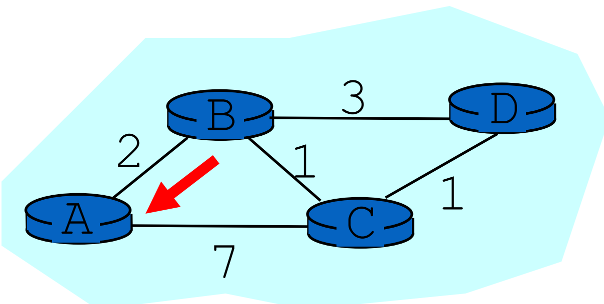
	A	B	D
A	7	∞	∞
B	∞	1	∞
D	∞	∞	1

Node D

	B	C
A	∞	∞
B	3	∞
C	∞	1

```
loop:
  …
  else if (update D(A, Y) from C)
    D_C(A,Y) = D_C(A,C) + D(C, Y);
  if (new min. for destination Y)
    send D(A, Y) to all neighbors
forever
```

Example: Now B sends update to A



Node A

	B	C
B	2	8
C	3	7
D	5	8

Node B

	A	C	D
A	2	∞	∞
C	∞	1	∞
D	∞	∞	3

C) = D_B(A,B) + D(B, C) = 2 + 1 = 3

(A,B) + D(B, D) = 2 + 3 = 5

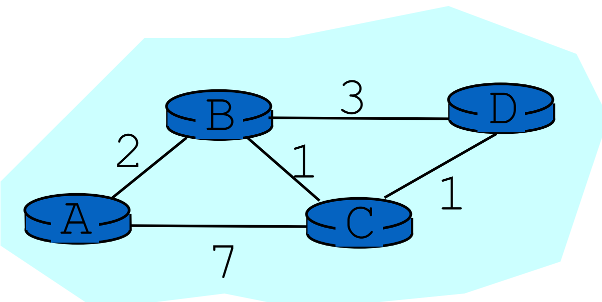
Make sure you know why this is 5, not 4!

```
else if (update D(A, Y) from B)
  D_B(A,Y) = D_B(A,B) + D(B, Y);
if (new min. for destination Y)
  send D(A, Y) to all neighbors
forever
```

A	7	∞	∞
B	∞	1	
D	∞	∞	1

		C
A	∞	∞
B	3	∞
C	∞	1

Example: After 1st Full Exchange



Node A

	B	C
B	2	8
C	3	7
D	5	8

Node B

	A	C	D
A	2	8	∞
C	9	1	4
D	∞	2	3

Make sure you know why this is 3

Assume all send messages at same time

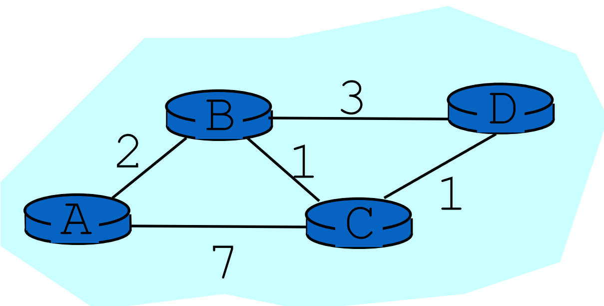
		B	D
A	7	3	∞
B	9	1	4
D	∞	4	1

Node D

	B	C
A	5	8
B	3	2
C	4	1

Example: Now A

What harm does this cause?



Node A

	B	C
B	2	8
C	3	7
D	5	8

		C	D
A	2	8	∞
C	5	1	4
D	7	2	3

$D_A(B, C) = D_A(B,A) + D(A, C) = 2 + 3 = 5$

$D_A(B, D) = D_A(B,A) + D(A, D) = 2 + 5 = 7$

```
loop:
  …
  else if (update D(B, Y) from A)
    D_A(B,Y) = D_A(B,A) + D(A, Y);
  if (new min. for destination Y)
    send D(B, Y) to all neighbors
forever
```

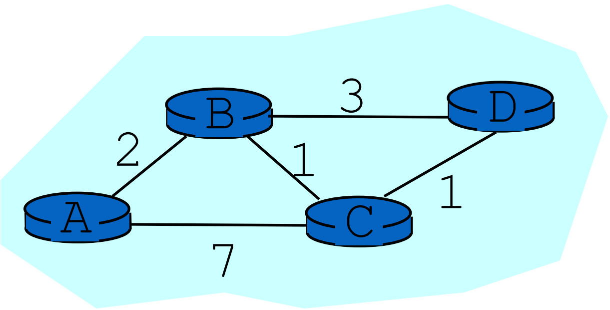
Node C

	A	B	D
A	7	3	∞
B	9	1	4
D	∞	4	1

Node D

	B	C
A	5	8
B	3	2
C	4	1

Example: End of 2nd Full Exchange



Node A

	B	C
B	2	8
C	3	7
D	4	8

Node B

	A	C	D
A	2	4	8
C	5	1	4
D	7	2	3

Node C

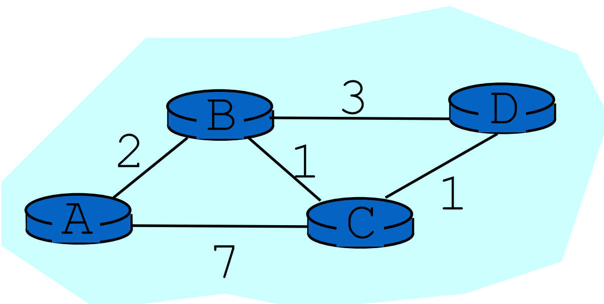
	A	B	D
A	7	3	6
B	9	1	3
D	12	3	1

Node D

	B	C
A	5	4
B	3	2
C	4	1

Assume all send messages at same time

Example: End of 3rd Full Exchange



Node A

	B	C
B	2	8
C	3	7
D	4	8

Node B

	A	C	D
A	2	4	7
C	5	1	4
D	6	2	3

Node C

	A	B	D
A	7	3	5
B	9	1	3
D	11	3	1

Node D

	B	C
A	5	4
B	3	2
C	4	1

Assume all send messages at same time

What route does this 11 represent?

Intuition

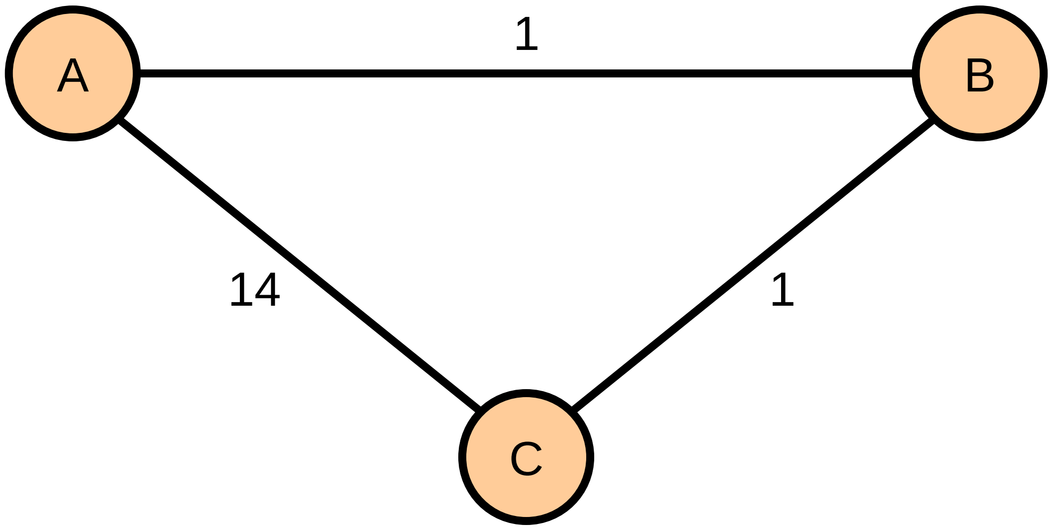
- Initial state: best one-hop paths
- One simultaneous round: best two-hop paths
- Two simultaneous rounds: best three-hop paths

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

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

Count-to-Infinity Problem

dest	cost
B	1
C	2

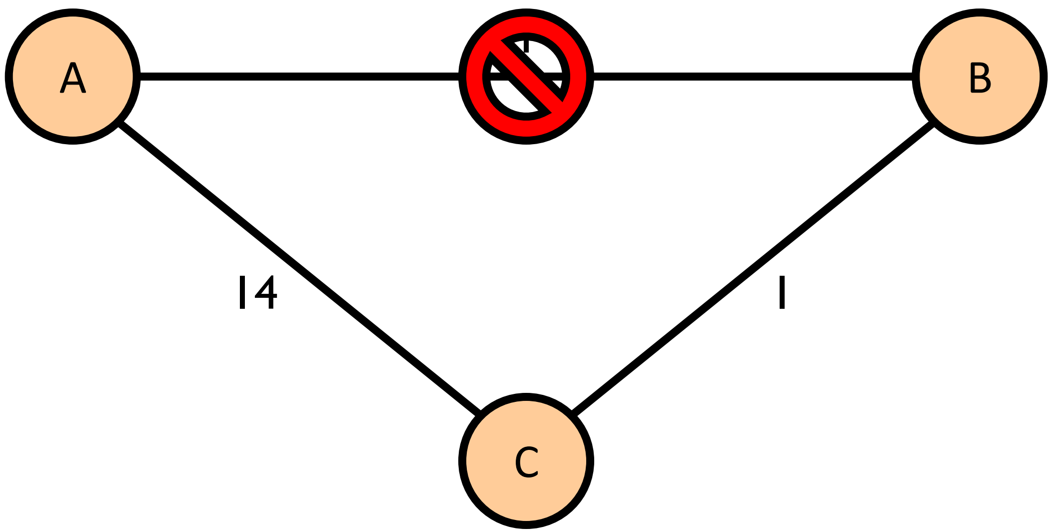


dest	cost
A	1
C	1

dest	cost
A	2
B	1

Count-to-Infinity Problem

dest	cost
B	1
C	2

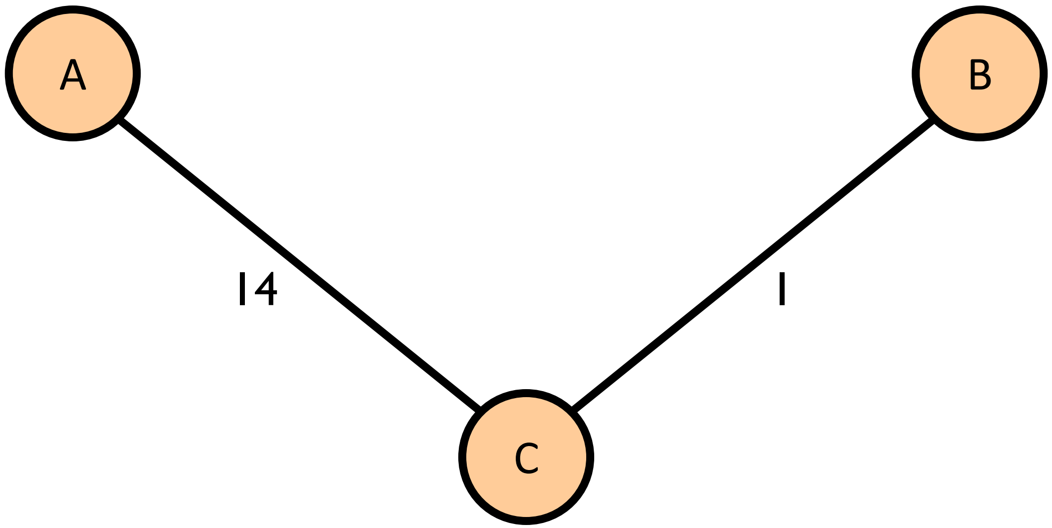


dest	cost
A	1
C	1

dest	cost
A	2
B	1

Count-to-Infinity Problem

dest	cost
B	1
C	2

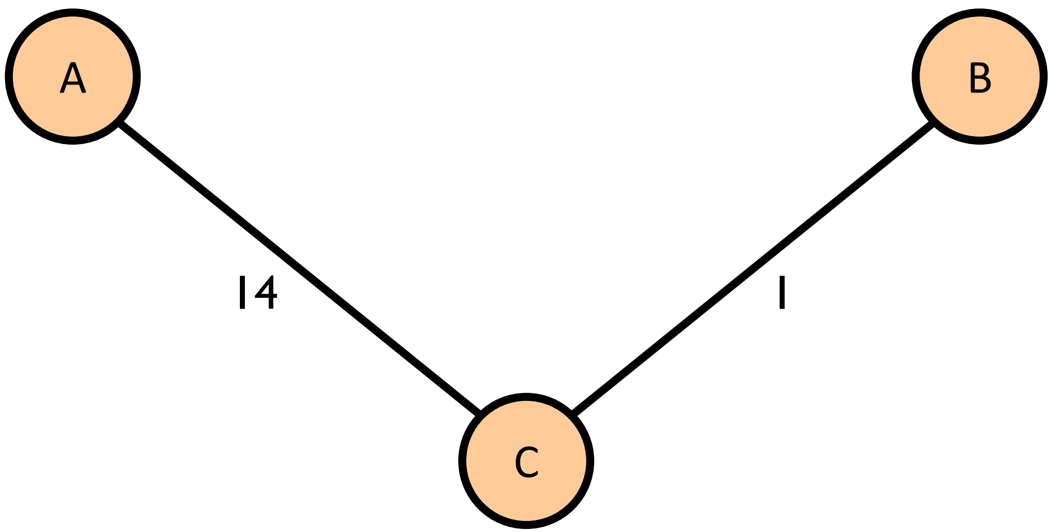


dest	cost
A	1
C	1

dest	cost
A	2
B	1

Count-to-Infinity Problem

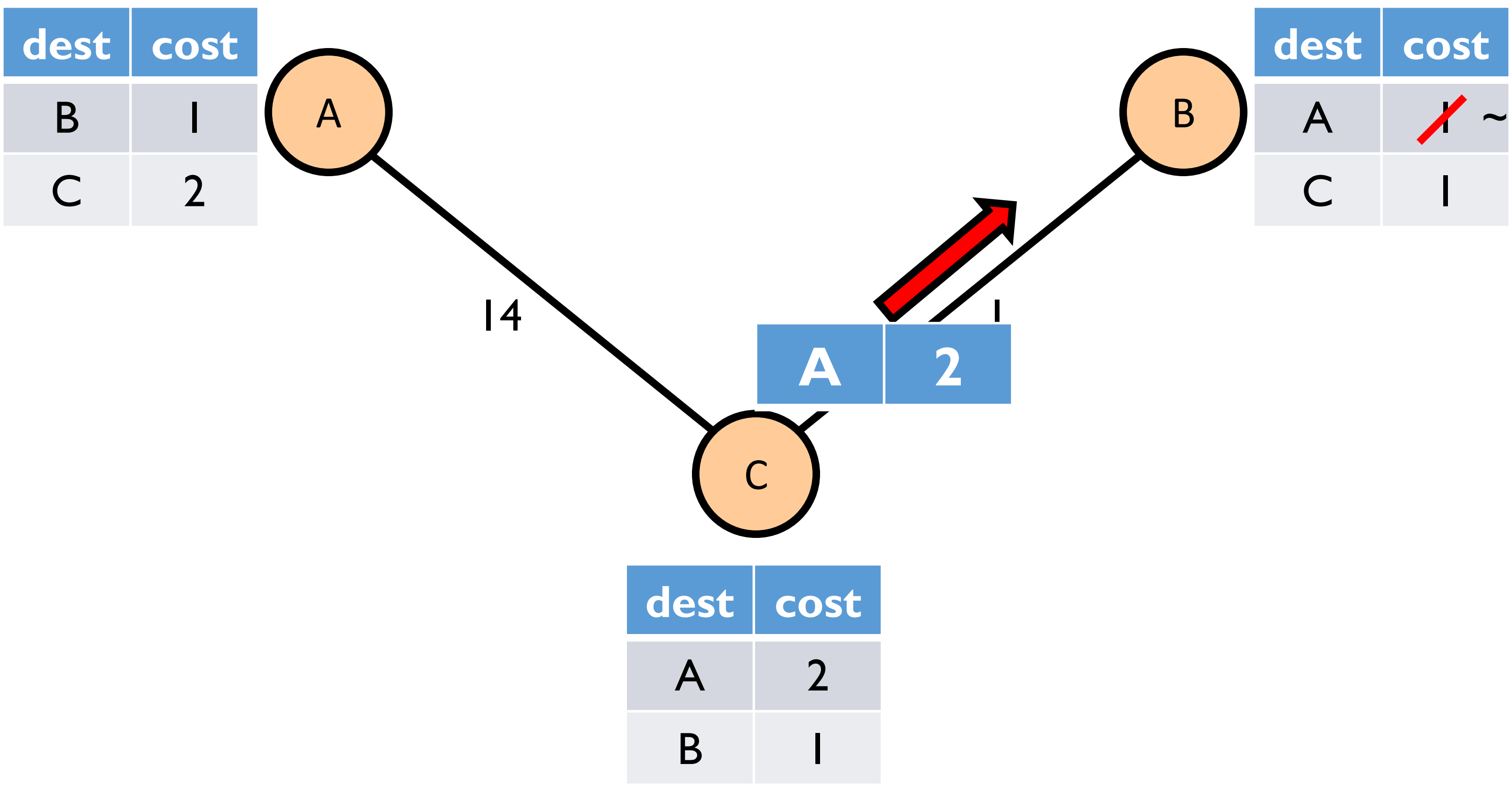
dest	cost
B	1
C	2



dest	cost
A	~~1~~ ~
C	1

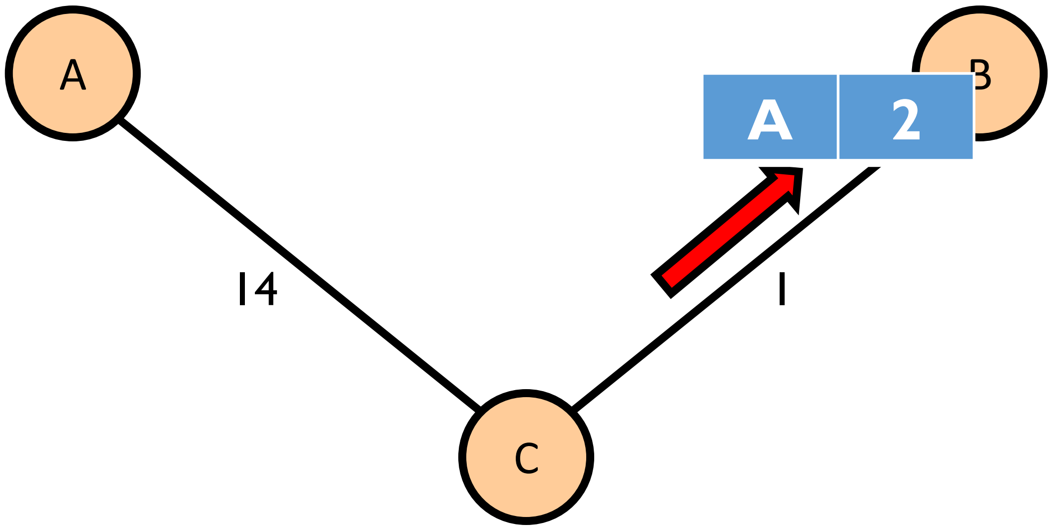
dest	cost
A	2
B	1

Count-to-Infinity Problem



Count-to-Infinity Problem

dest	cost
B	1
C	2

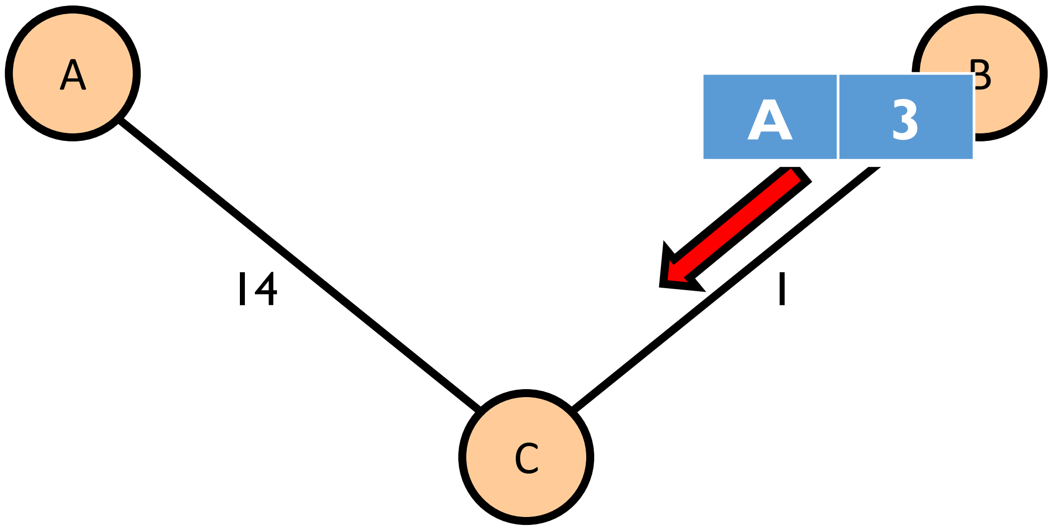


dest	cost
A	~~1~~ ~~2~~ 3
C	1

dest	cost
A	2
B	1

Count-to-Infinity Problem

dest	cost
B	1
C	2

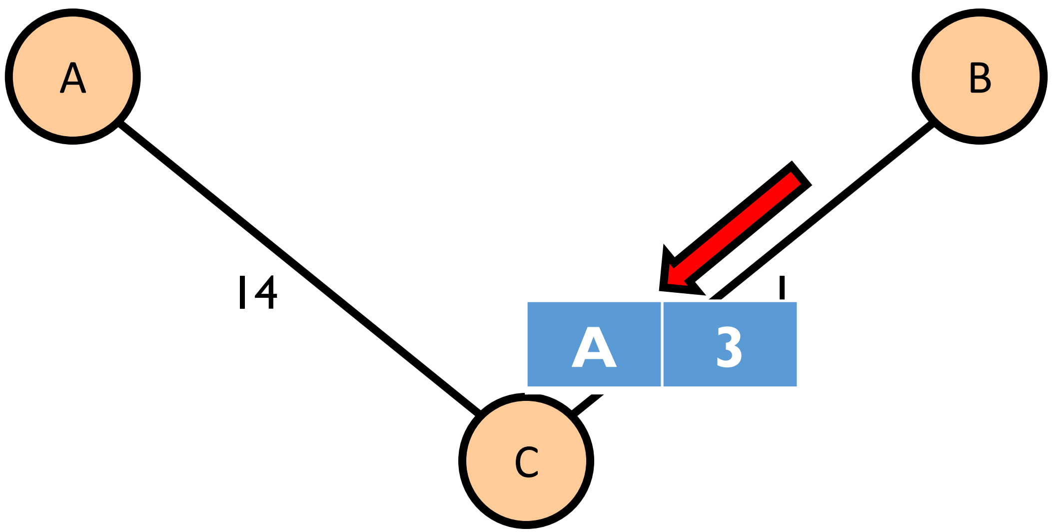


dest	cost
A	~~1~~ ~~∞~~ 3
C	1

dest	cost
A	2
B	1

Count-to-Infinity Problem

dest	cost
B	1
C	2

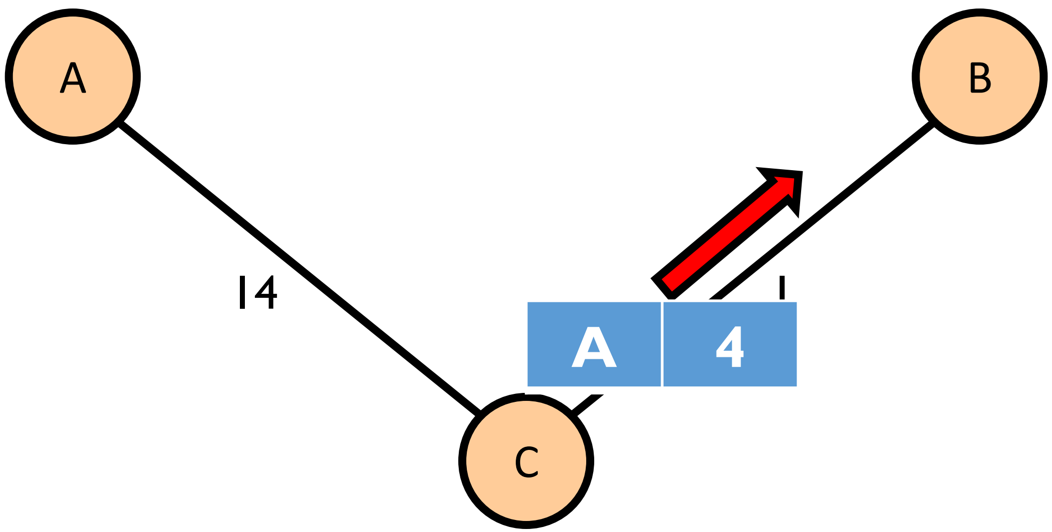


dest	cost
A	~~1~~ ~~2~~ 3
C	1

dest	cost
A	~~2~~ 4
B	1

Count-to-Infinity Problem

dest	cost
B	1
C	2

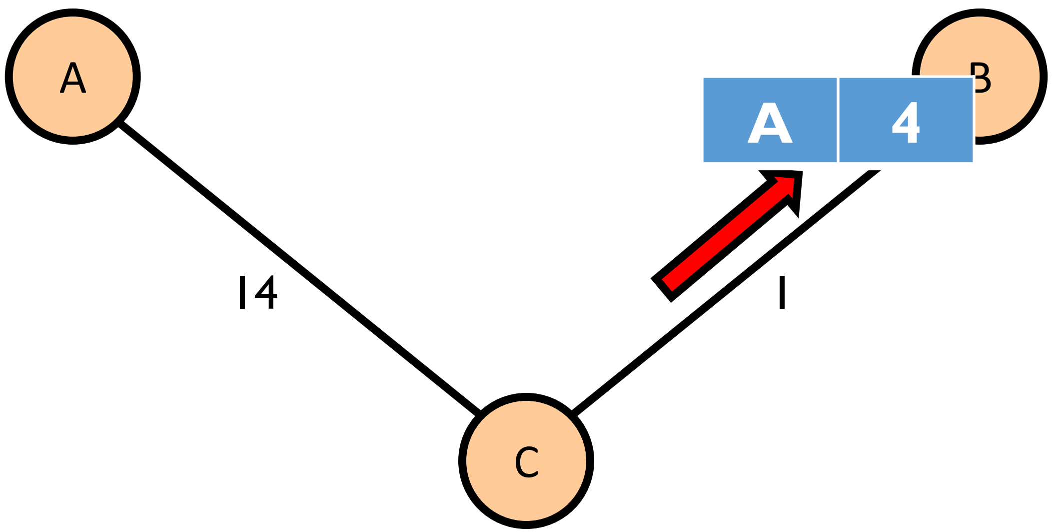


dest	cost
A	~~1~~ ~~2~~ 3
C	1

dest	cost
A	~~2~~ 4
B	1

Count-to-Infinity Problem

dest	cost
B	1
C	2



dest	cost
A	~~1~~ ~~2~~ ~~3~~ 5
C	1

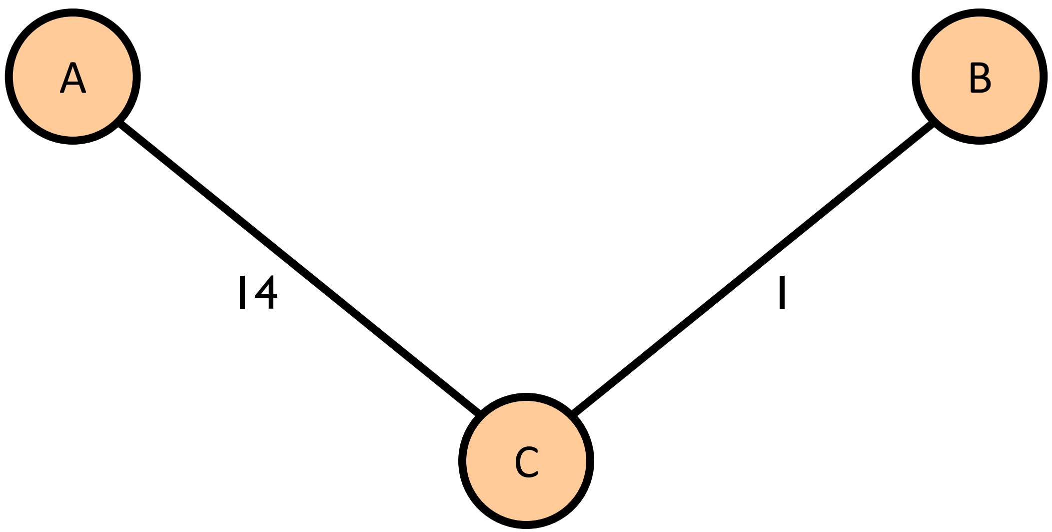
dest	cost
A	~~2~~ 4
B	1

How to deal with count-to infinity problem?

- Option 1: if router X advertises router Y a route, Y should not advertise that route back to X
  - Called split horizon
  - Can be fooled by 3-node loops
- Option 2: if one of my routes is disappeared, I should advertise that route to all my neighbors with infinite costs
  - Called poison reverse
  - Useful for networks where routes only disappear after a timeout (so it’s not necessary if you’re using a protocol with triggered updates)
  - Or, can alternatively have an explicit “withdrawal” message

Split Horizon

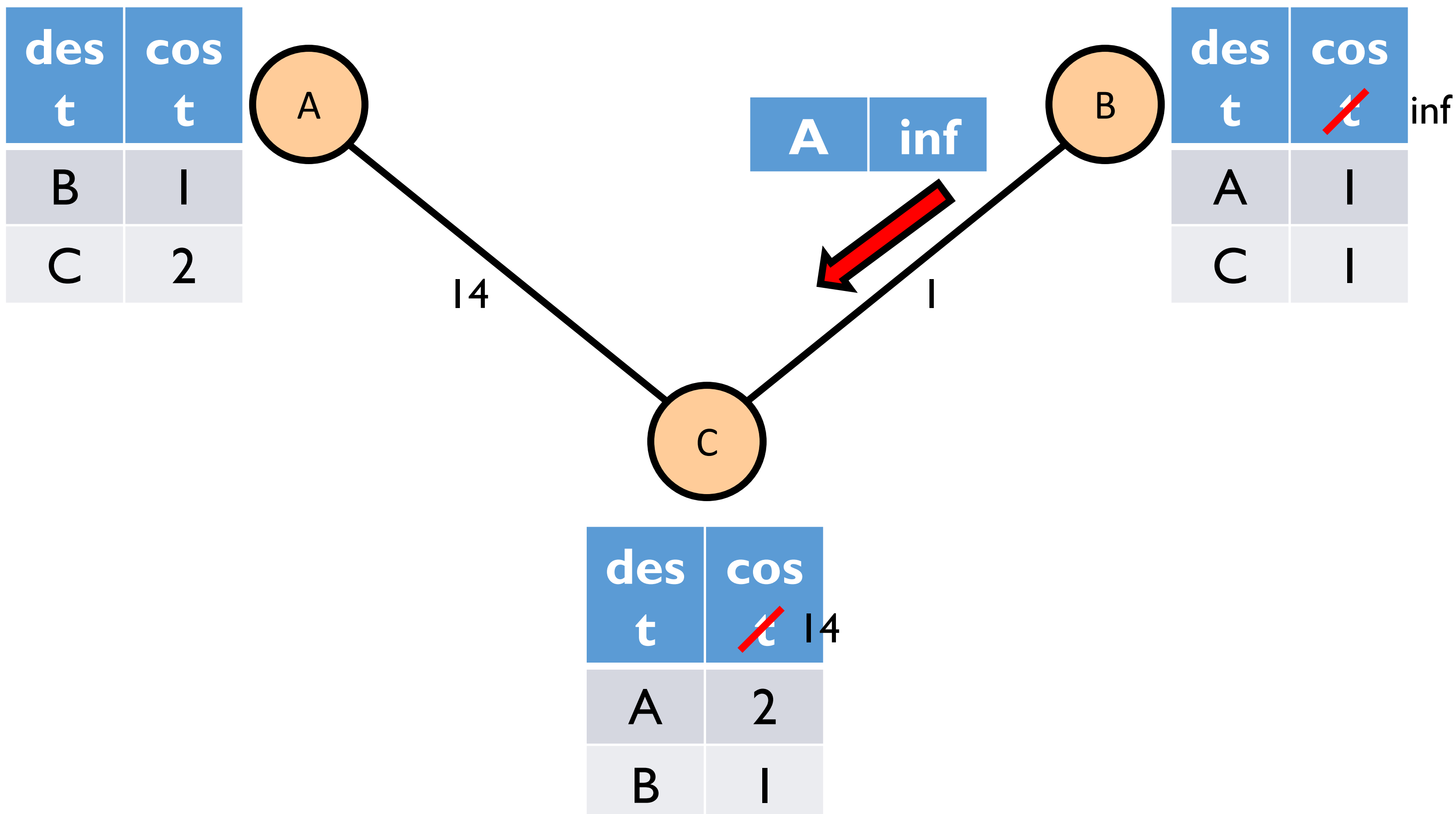
dest	cost
B	1
C	2



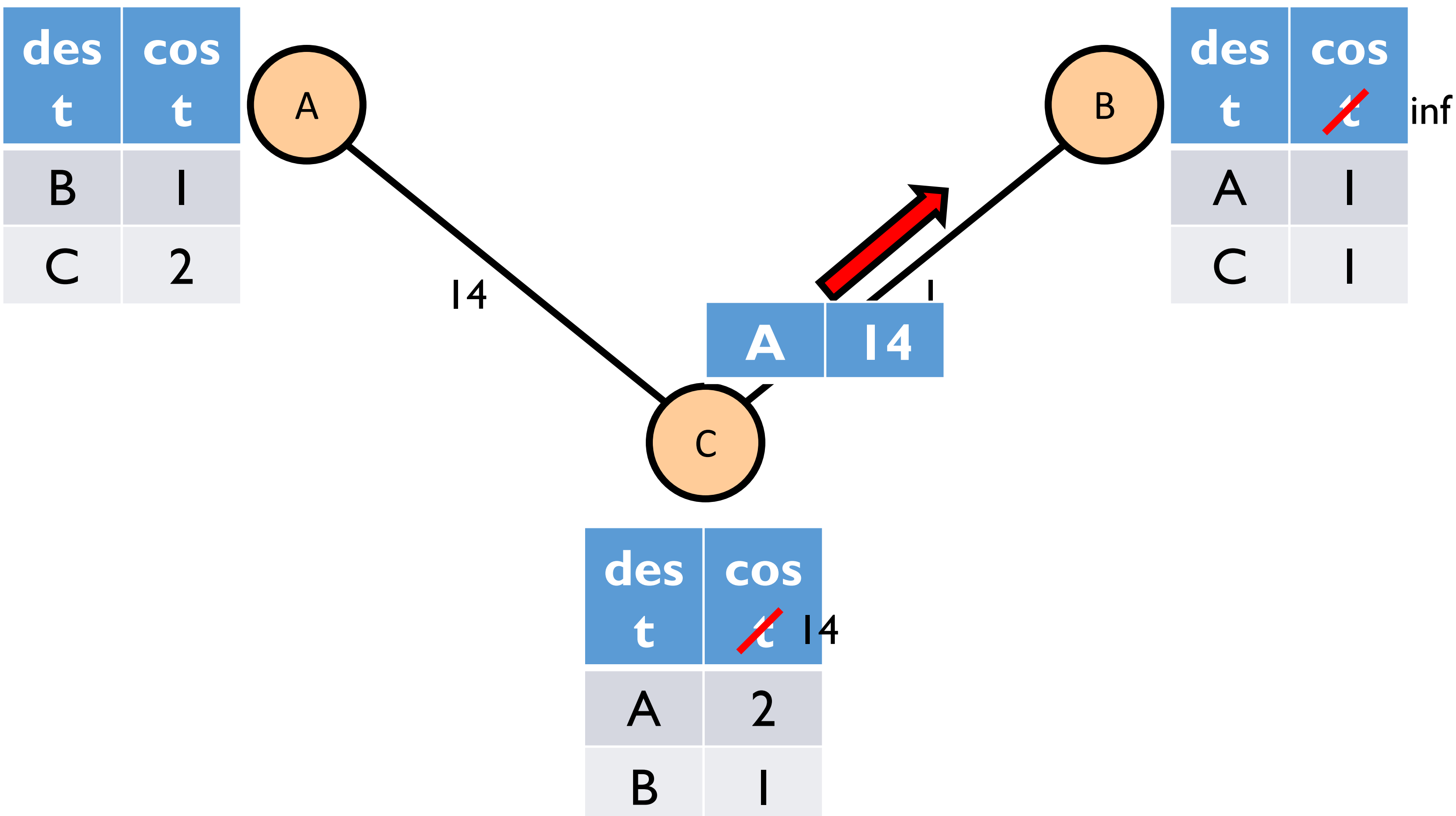
dest	~~cost~~ inf
A	1
C	1

dest	cost
A	2
B	1

Split Horizon

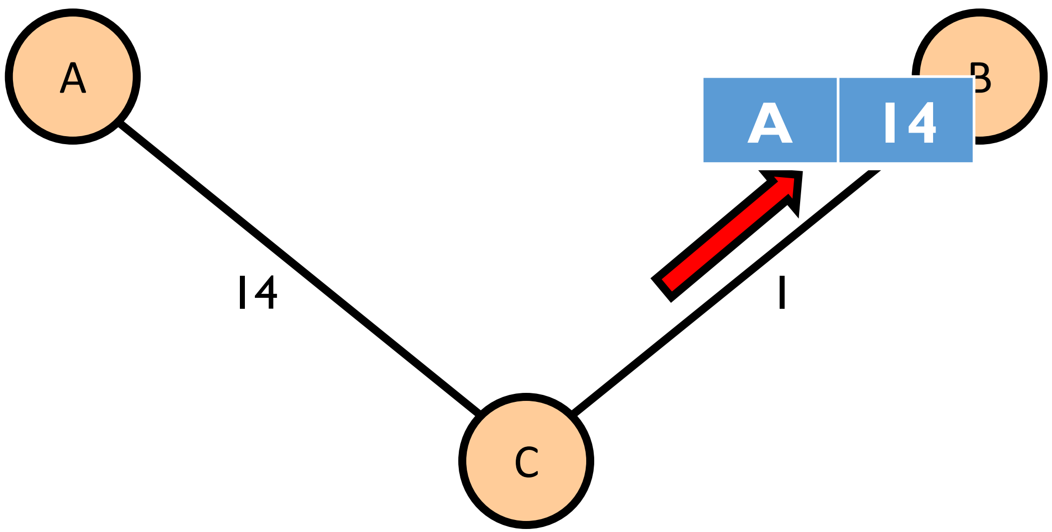


Split Horizon



Split Horizon

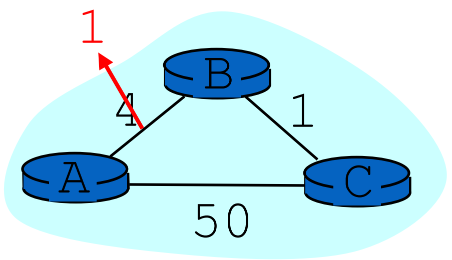
dest	cost
B	1
C	2



dest	cos~~t~~ ~~inf~~ 15
A	1
C	1

dest	cos~~t~~ 14
A	2
B	1

DV: Link Cost Changes



Stable state

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	1	51
C	2	50

Node B

	A	C
A	1	6
C	6	1

Node C

	A	B
A	50	5
B	54	1

A sends tables to B, C

Node A

	B	C
B	1	51
C	2	50

Node B

	A	C
A	1	6
C	3	1

Node C

	A	B
A	50	5
B	51	1

B sends tables to C

Node A

	B	C
B	1	51
C	2	50

Node B

	A	C
A	1	6
C	3	1

Node C

	A	B
A	50	2
B	51	1

C sends tables to B

Node A

	B	C
B	1	51
C	2	50

Node B

	A	C
A	1	3
C	3	1

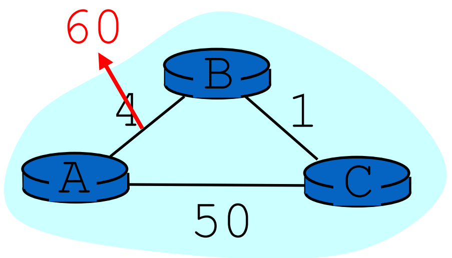
Node C

	A	B
A	50	2
B	51	1

Link cost changes here

"good news travels fast"

DV: *Count to Infinity* Problem



Stable state

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

A sends tables to B, C

Node A

	B	C
B	60	51
C	61	50

Node B

	A	C
A	60	6
C	110	1

Node C

	A	B
A	50	5
B	101	1

B sends tables to C

Node A

	B	C
B	60	51
C	61	50

Node B

	A	C
A	60	6
C	110	1

Node C

	A	B
A	50	7
B	101	1

C sends tables to B

Node A

	B	C
B	60	51
C	61	50

Node B

	A	C
A	60	8
C	110	1

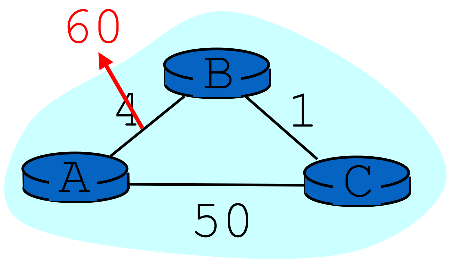
Node C

	A	B
A	50	7
B	101	1

Link cost changes here

"bad news travels slowly"
(not yet converged)

DV: *Poisoned Reverse*



- If B routes through 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)

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 tables to B, C

	B	C
B	60	51
C	61	50

B sends tables to C

	B	C
B	60	51
C	61	50

C sends tables to B

	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

A sends tables to B, C

	A	C
A	60	∞
C	110	1

B sends tables to C

	A	C
A	60	∞
C	110	1

C sends tables to B

	A	C
A	60	51
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 tables to B, C

	A	B
A	50	5
B	∞	1

B sends tables to C

	A	B
A	50	61
B	∞	1

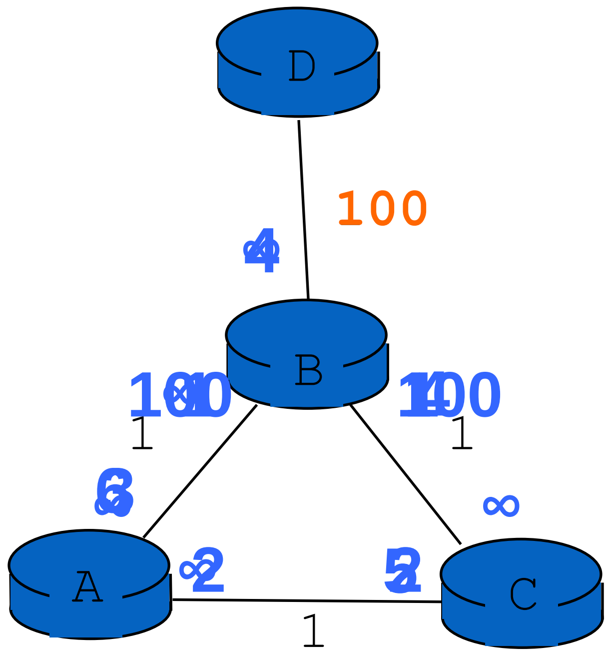
C sends tables to B

	A	B
A	50	61
B	∞	1

Link cost changes here

Note: this converges after C receives another update from B

Will Poison-Reverse Completely Solve the Count-to-Infinity Problem?



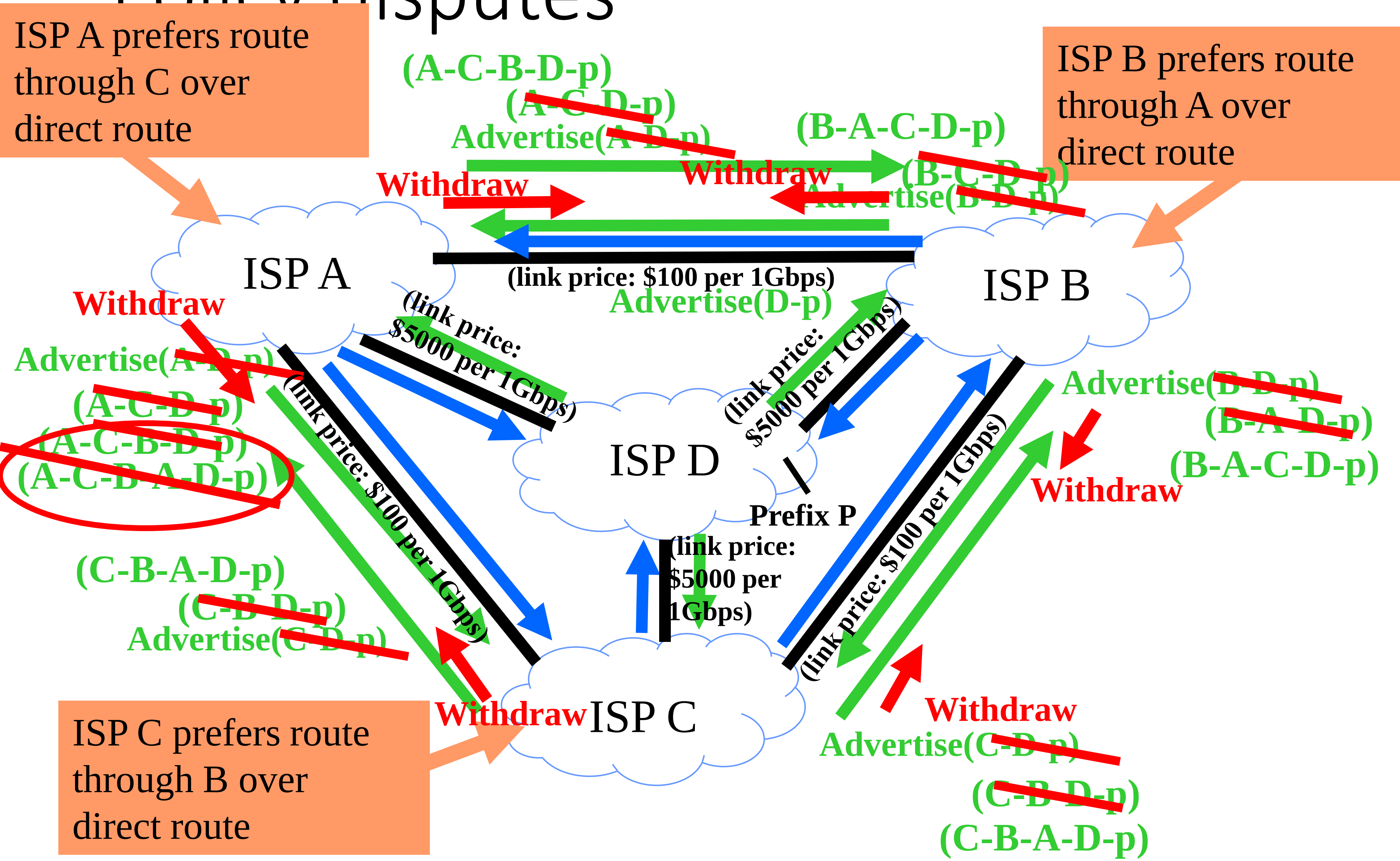
A few other inconvenient aspects

- What if we use a non-additive metric?
  - E.g., maximal capacity
- What if routers don’t use the same metric?
  - I want low delay, you want low loss rate?
- What happens if nodes lie?

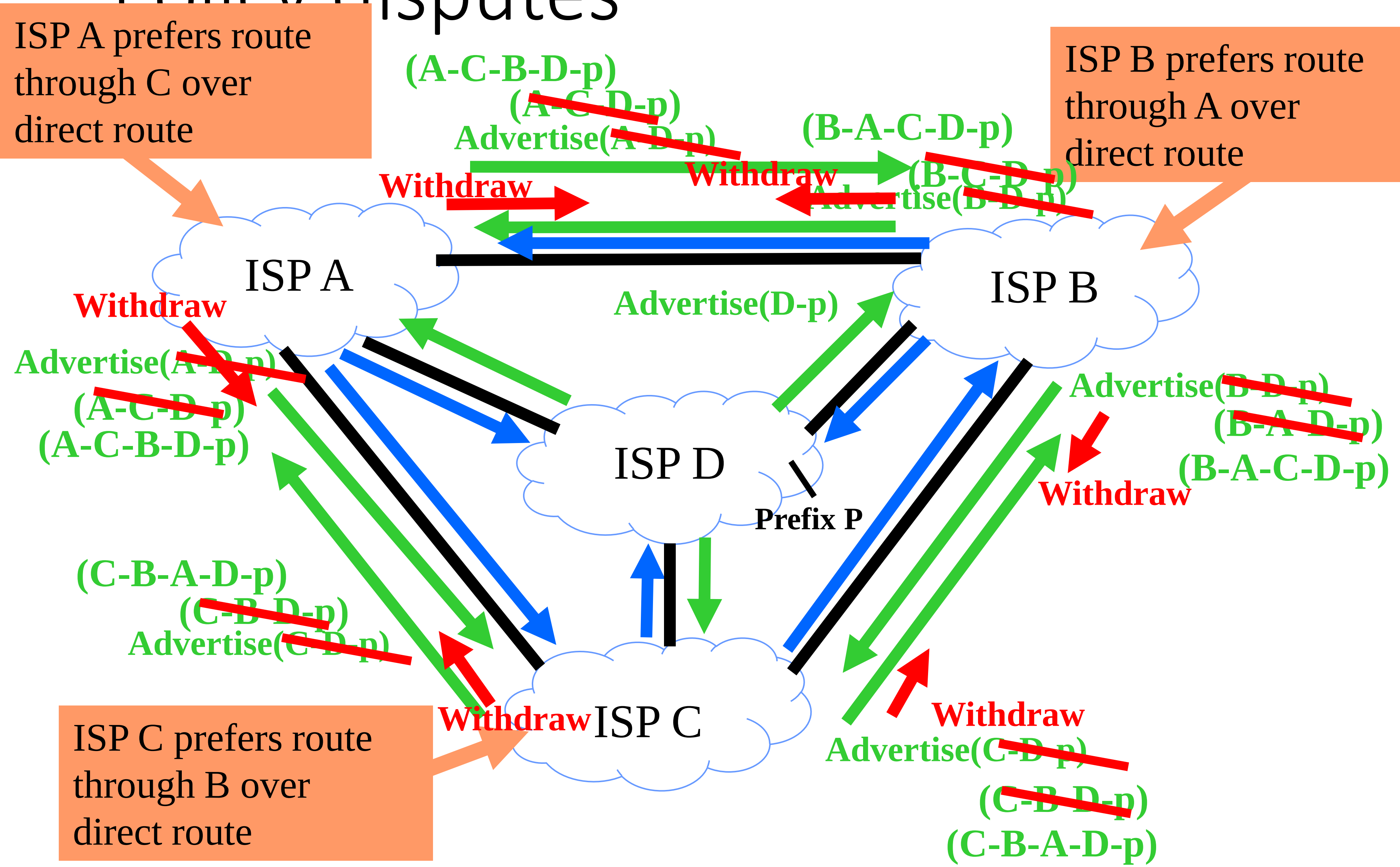
Can You Use Any Metric?

- I said that we can pick any metric. Really?
- What about maximizing capacity?

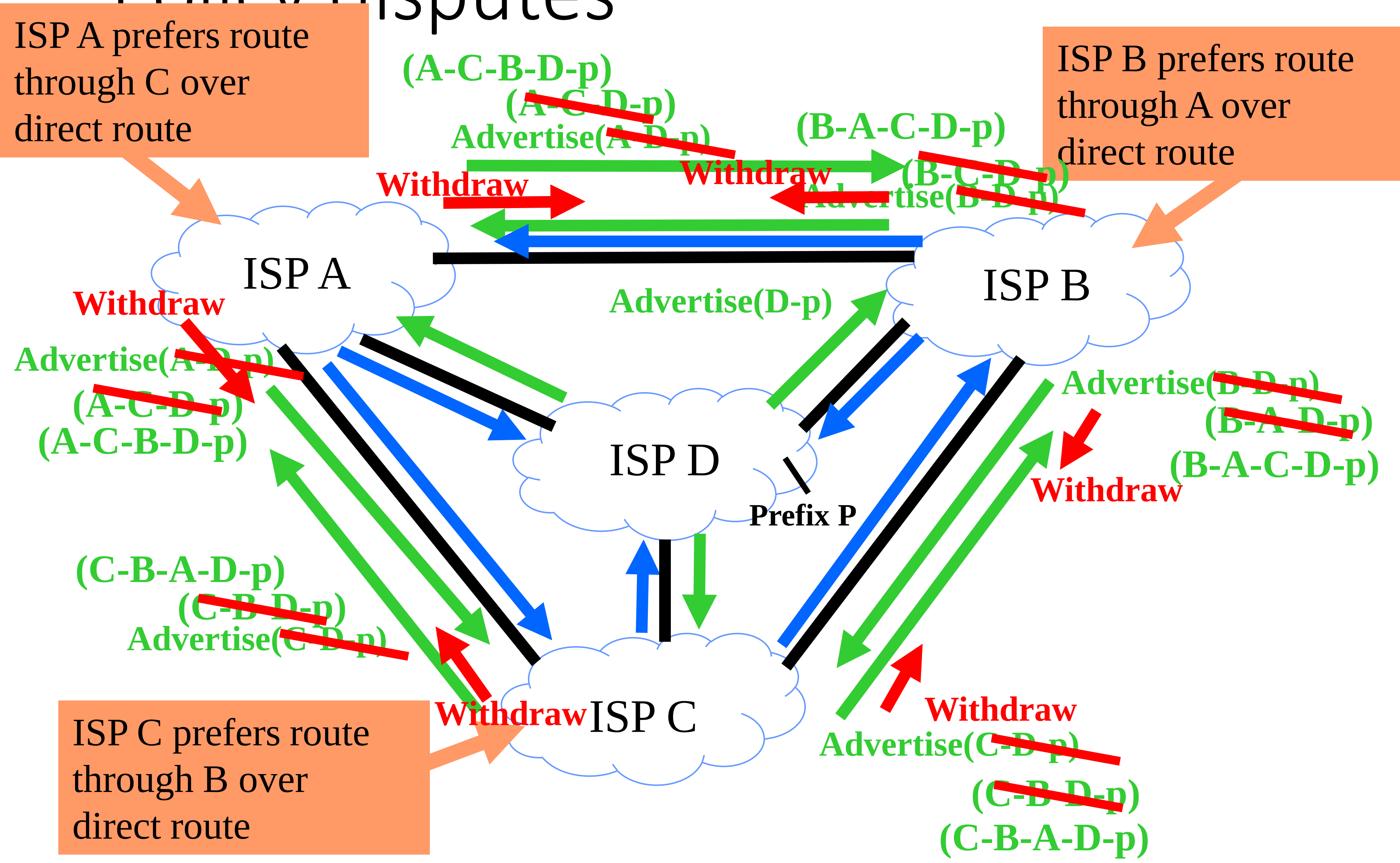
Policy disputes



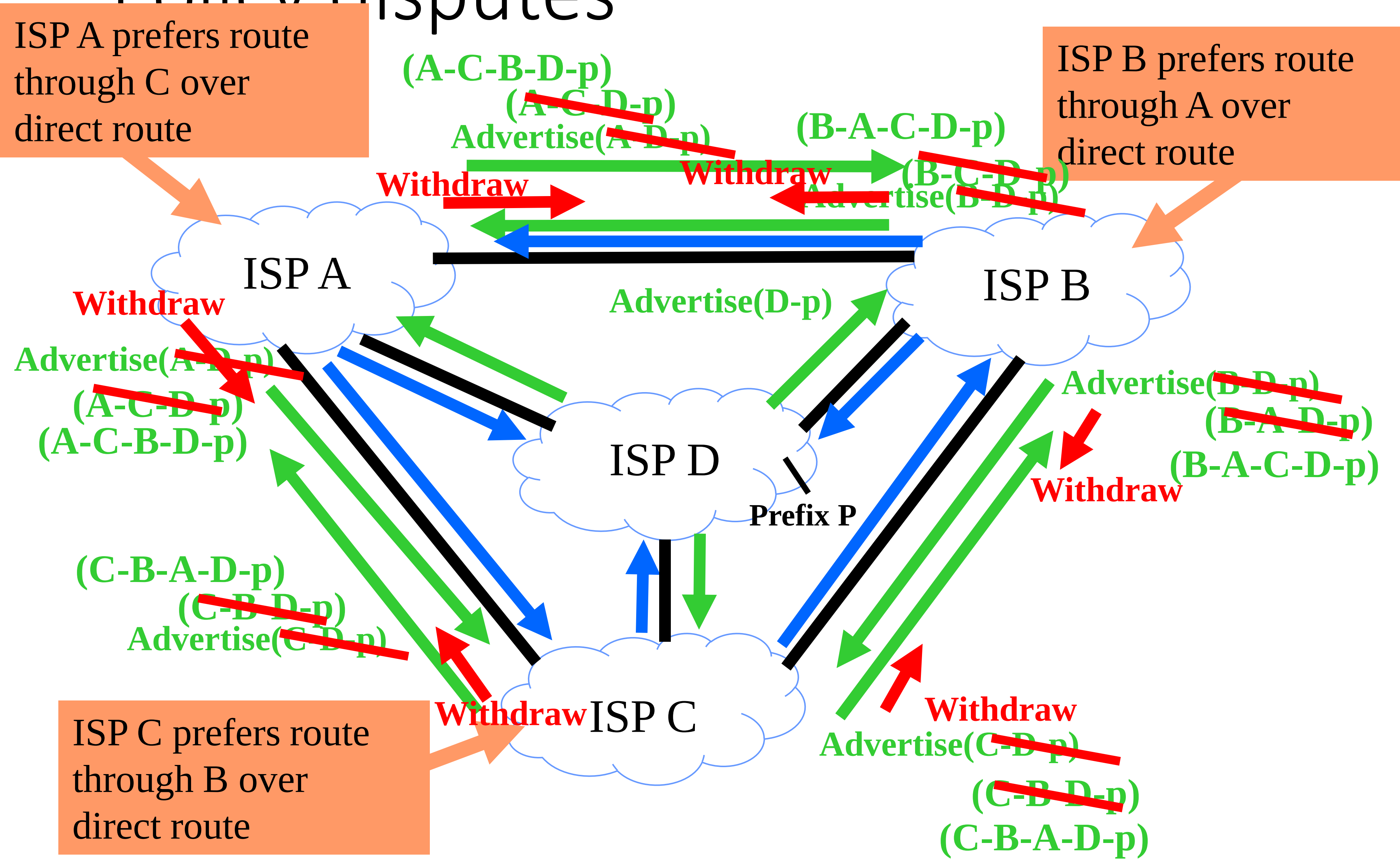
Policy disputes



Policy disputes

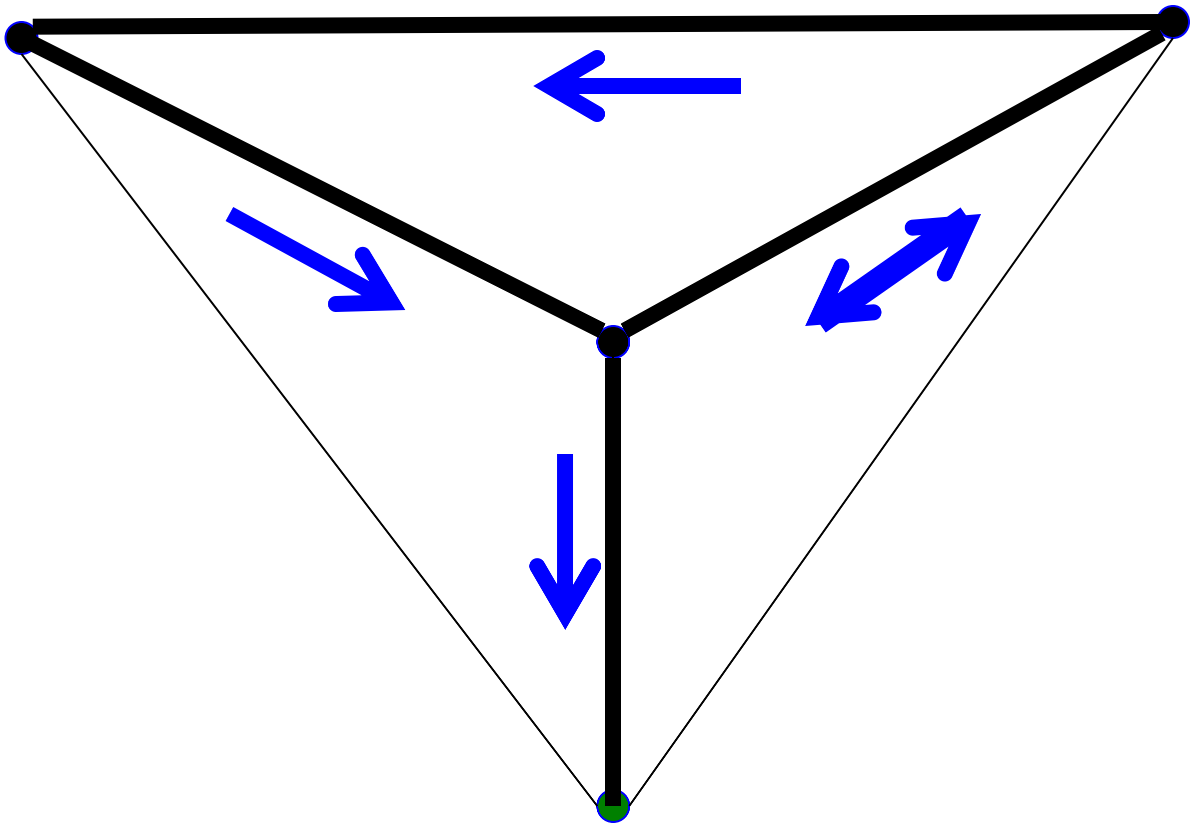


Policy disputes



What Happens Here?

Problem: "cost" does not change around loop

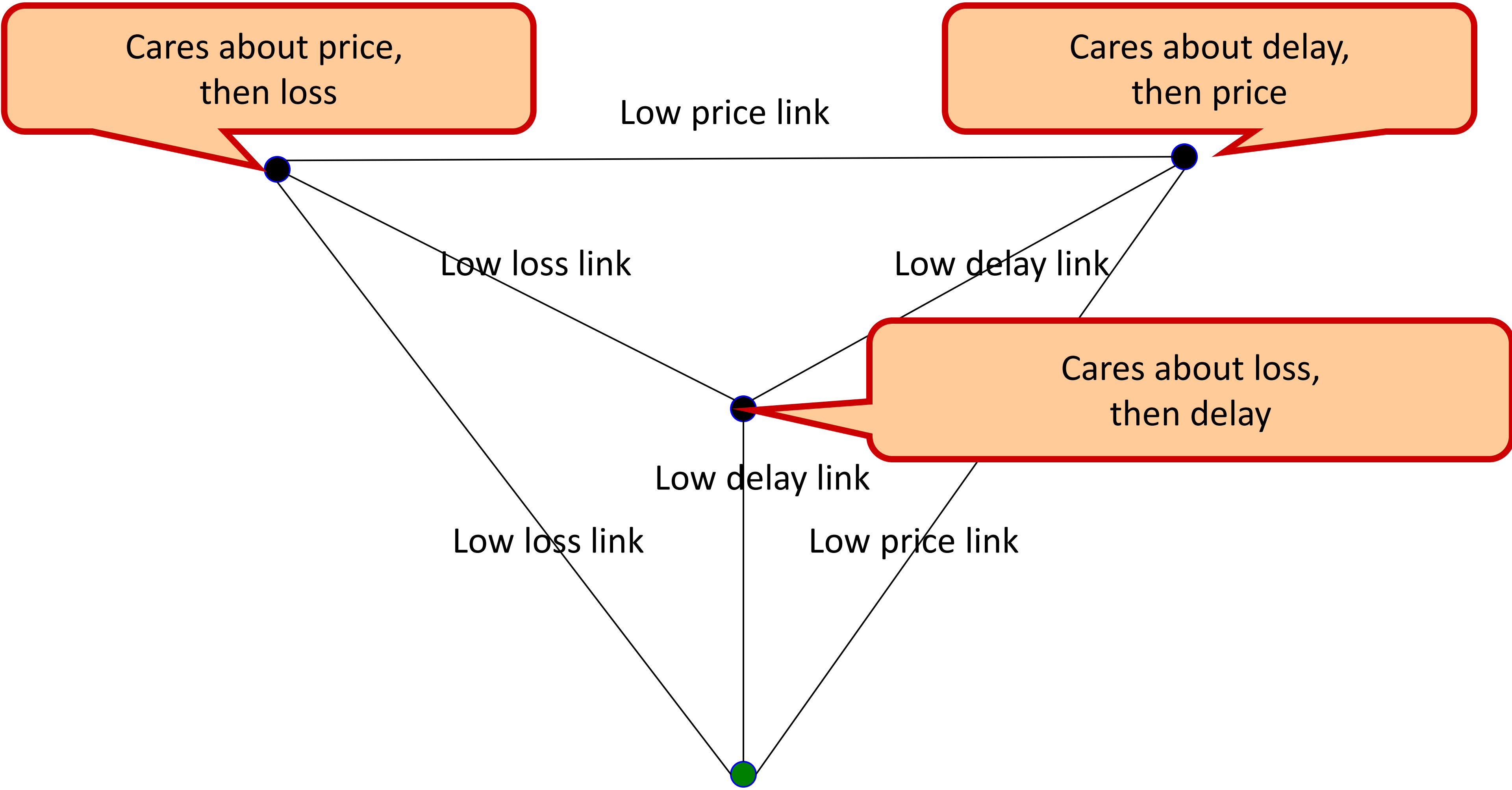


Additive measures avoid this problem!

No agreement on metrics?

- If the nodes choose their paths according to different criteria, then bad things might happen
- Example
  - Node A is minimizing latency
  - Node B is minimizing loss rate
  - Node C is minimizing price
- Any of those goals are fine, if globally adopted
  - Only a problem when nodes use different criteria
- Consider a routing algorithm where paths are described by delay, cost, loss

What Happens Here?



Must agree on loop-avoiding metric

- When all nodes minimize same metric
- And that metric increases around loops
- Then process is guaranteed to converge

What happens when routers lie?

- What if a router claims a 1-hop path to everywhere?
- All traffic from nearby routers gets sent there
- How can you tell if they are lying?
- Can this happen in real life?
  - It has, several times….

Link State vs. Distance Vector

- Core idea
  - LS: tell all nodes about your immediate neighbors
  - DV: tell your immediate neighbors about (your least cost distance to) all nodes

Link State vs. Distance Vector

- LS: each node learns the complete network map; each node computes shortest paths independently and in parallel
- DV: no node has the complete picture; nodes cooperate to compute shortest paths in a distributed manner

→LS has higher messaging overhead
→LS has higher processing complexity
→LS is less vulnerable to looping

Link State vs. Distance Vector

Message complexity

- LS: O(NxE) messages;
  - N is #nodes; E is #edges
- DV: O(#Iterations x E)
  - where #Iterations is ideally O(network diameter) but varies due to routing loops or the count-to-infinity problem

Processing complexity

- LS: $O(N^2)$
- DV: O(#Iterations x N)

Robustness: what happens if router malfunctions?

- LS:
  - node can advertise incorrect *link* cost
  - each node computes only its *own* table
- DV:
  - node can advertise incorrect *path* cost
  - each node's table used by others; error propagates through network

Routing: Just the Beginning

- Link state and distance-vector are the deployed routing paradigms for intra-domain routing
- Next lecture: inter-domain routing (BGP)
  - new constraints: policy, privacy
  - new solutions: path vector routing
  - new pitfalls: truly ugly ones

What are desirable goals for a routing solution?

- “Good” paths (least cost)
- Fast convergence after change/failures
  - no/rare loops
- Scalable
  - #messages
  - table size
  - processing complexity
- Secure
- Policy
- Rich metrics (more later)

Delivery models

- What if a node wants to send to more than one destination?
  - broadcast: send to all
  - multicast: send to all members of a group
  - anycast: send to any member of a group
- What if a node wants to send along more than one path?

Metrics

- Propagation delay
- Congestion
- Load balance
- Bandwidth (available, capacity, maximal, bbw)
- Price
- Reliability
- Loss rate
- Combinations of the above

In practice, operators set abstract "weights" (much like our costs); how exactly is a bit of a black art