Network Layer:
Distance Vector Protocols Variations
Link-State Protocol

Qiao Xiang

https://qiaoxiang.me/courses/cnns-xmuf22/index.shtml

11/22/2022

This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

Outline

- Admin and recap
- Network overview
- Network control plane
    - Routing
        - Link weights assignment
        - Routing computation
            - Distance vector protocols (distributed computing)
            - Link state protocols (distributed state synchronization)

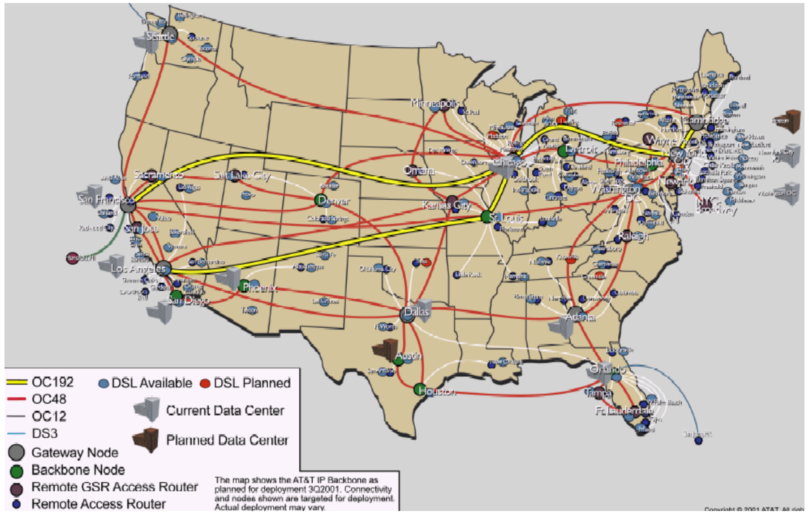
Admin

- Guest lectures are important ☺
- Upcoming guest lectures
  - December 1, Dr. Linghe Kong@SJTU, Internet of Things
  - December 8, Dr. Zaoxing (Alan) Liu@Boston Univ., Programmable Networks
  - December 15, Dr. Zhenhua Li@THU, 5G Network
- Schedule is tentative

Recap: Routing Context

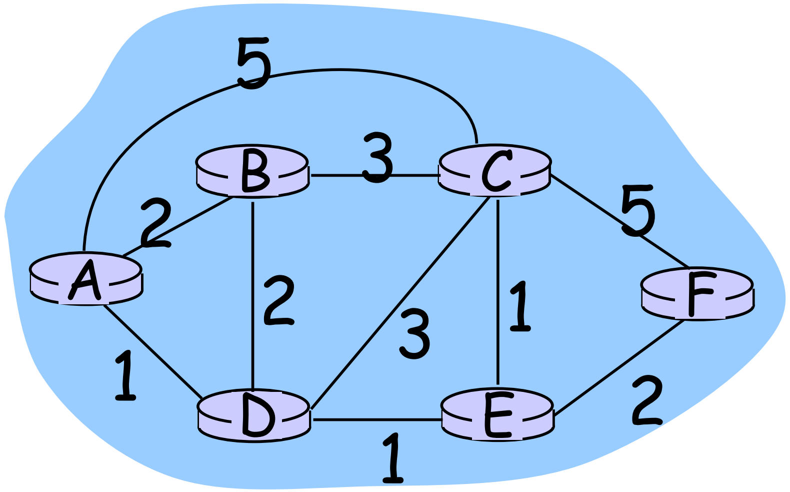
Routing

Goal: determine "good" paths (sequences of routers) thru networks from source to dest.



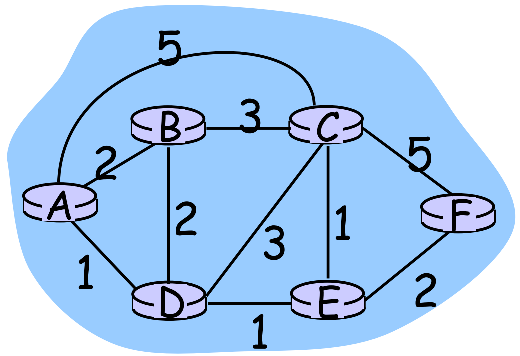
Often depends on a graph abstraction:

- graph nodes are routers
- graph edges are physical links
  - links have properties: delay, capacity, $ cost, policy



Recap: Routing Design Space

- Routing has a large design space
  - who decides routing?
    - source routing: end hosts make decision
    - network routing: networks make decision
  - how many paths from source s to destination d?
    - multi-path routing
    - single path routing
  - what does routing compute?
    - network cost minimization (shortest path routing)
    - QoS aware
  - will routing adapt to network traffic demand?
    - adaptive routing
    - static routing
  - ...



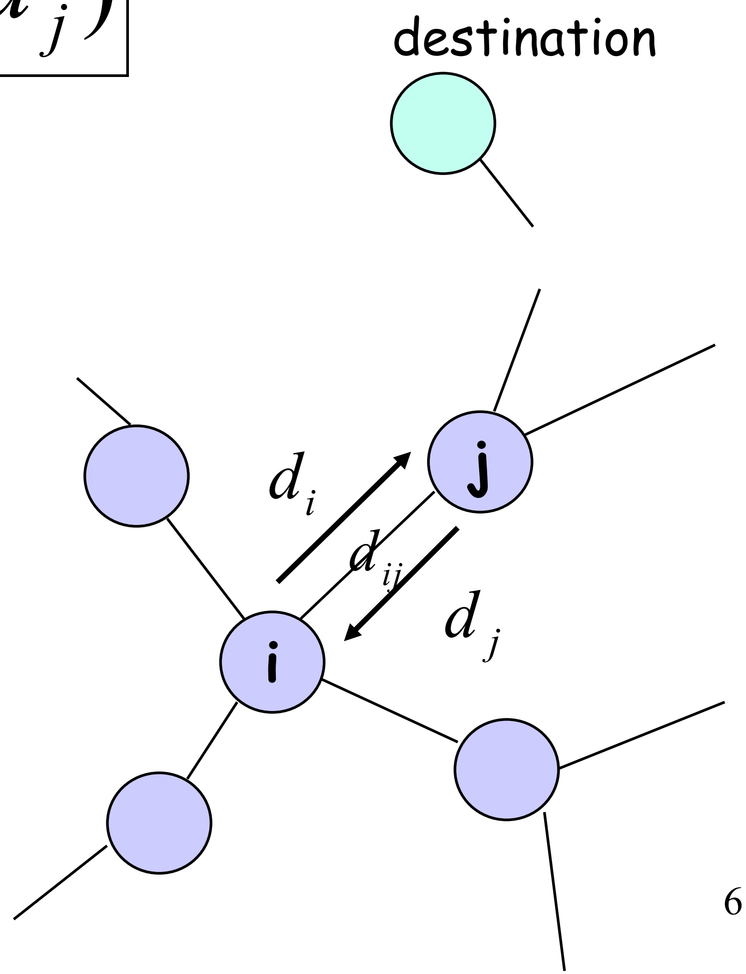
Recap: Distance Vector Routing: Basic Idea (Bellman-Ford Alg)

- At node i, the basic update rule

$$\boxed{d_i = \min_{j \in N(i)} (d_{ij} + d_j)}$$

where

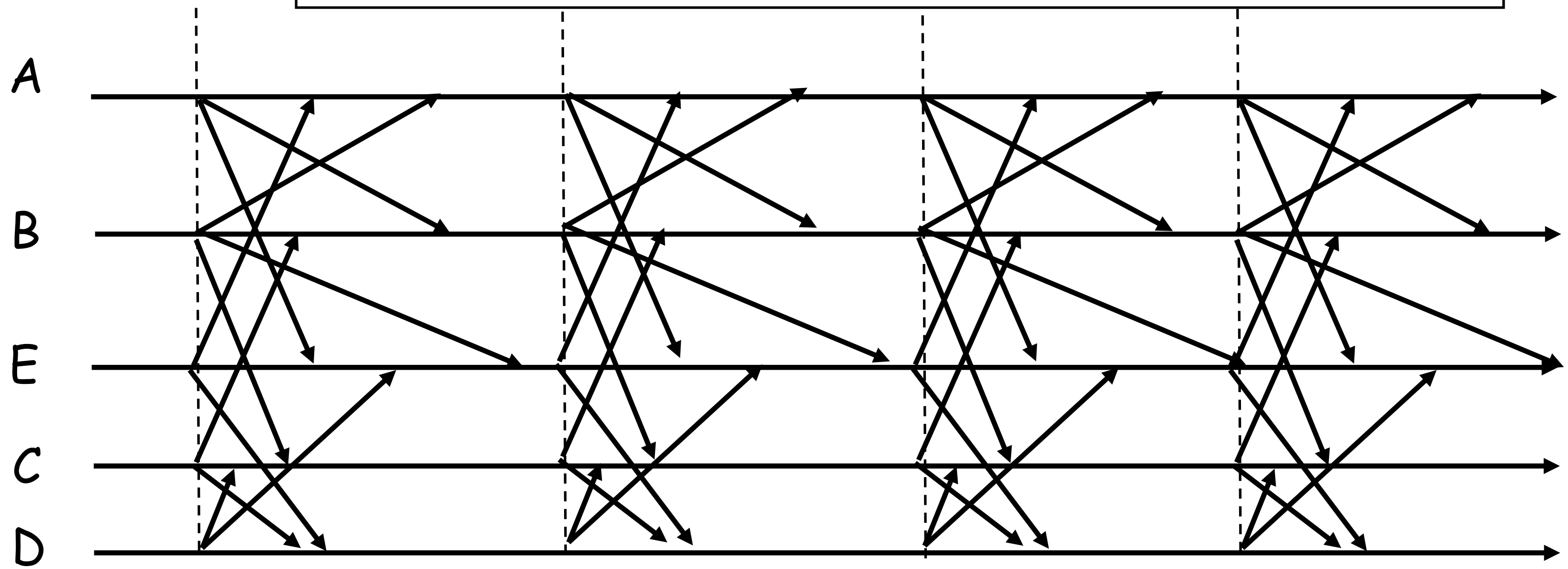
- d_i denotes the distance estimation from i to the destination,
- N(i) is set of neighbors of node i, and
- d_{ij} is the distance of the direct link from i to j



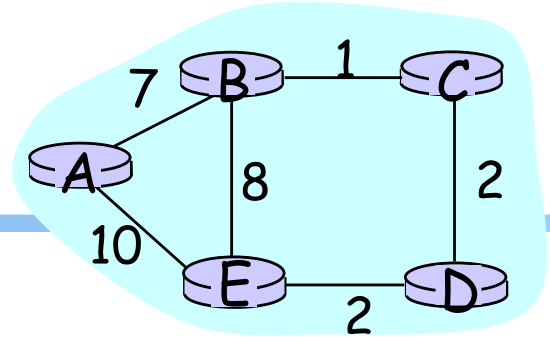
Recap: Synchronous Bellman-Ford (SBF)

- Nodes update in rounds:
  - there is a global clock;
  - at the beginning of each round, each node sends its estimate to all of its neighbors;
  - at the end of the round, updates its estimation

$$\boxed{d_i(h+1) = \min_{j \in N(i)} (d_{ij} + d_j(h))}$$



Recap: SBF/∞

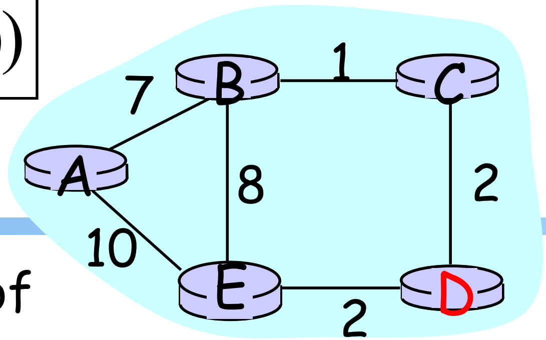


- Initialization (time 0):

$$d_i(0) = \begin{cases} 0 & \mathrm{i = dest} \\ \infty & \mathrm{otherwise} \end{cases}$$

$$d_i(h+1) = \min_{j \in N(i)} (d_{ij} + d_j(h))$$

Example



Consider D as destination; d(t) is a vector consisting of estimation of each node at round t

	A	B	C	E	D
d(0)	∞	∞	∞	∞	0
d(1)	∞	∞	2	2	0
d(2)	12	3	2	2	0
d(3)	10	3	2	2	0
d(4)	10	3	2	2	0

Observation: $d(0) \geq d(1) \geq d(2) \geq d(3) \geq d(4) = d^*$

$$d_i(h+1) = \min_{j \in N(i)} (d_{ij} + d_j(h))$$

A Nice Property of SBF: Monotonicity

- Consider two configurations d(t) and d'(t)

- If d(t) $\geq$ d'(t)
  - i.e., each node has a higher estimate in one scenario (d) than in another scenario (d'),

- then d(t+1) $\geq$ d'(t+1)
  - i.e., each node has a higher estimate in d than in d' after one round of synchronous update.

$$d_i(h+1) = \min_{j \in N(i)} (d_{ij} + d_j(h))$$

Correctness of SBF/∞

- Claim: $d_i(h)$ is the length $L_i(h)$ of a shortest path from i to the destination using $\leq h$ hops
  - base case: h = 0 is trivially true
  - assume true for $\leq$ h,
    i.e., $L_i(h) = d_i(h)$, $L_i(h-1) = d_i(h-1)$, ...

$$\boxed{d_i(h+1) = \min_{j \in N(i)}(d_{ij} + d_j(h))}$$

Correctness of SBF/∞

- consider ≤ h+1 hops:

$$L_i(h+1) = \min(L_i(h), \min_{j \in N(i)}(d_{ij} + L_j(h)))$$

$$= \min(d_i(h), \min_{j \in N(i)}(d_{ij} + d_j(h)))$$

$$= \min(d_i(h), d_i(h+1))$$

since $d_i(h) \le d_i(h-1)$

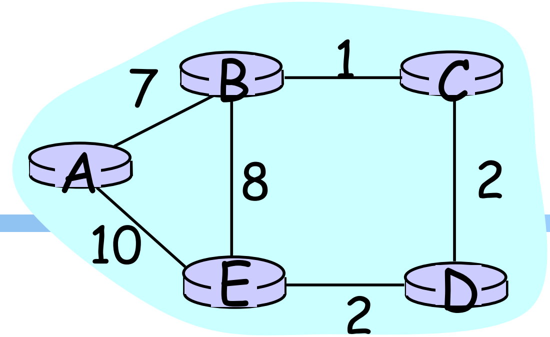
$$\boxed{d_i(h+1) = \min_{j \in N(i)}(d_{ij} + d_j(h)) \le \min_{j \in N(i)}(d_{ij} + d_j(h-1)) = d_i(h)}$$

$$L_i(h+1) = d_i(h+1)$$

Outline

- Admin and recap
- Network overview
- Network control plane
  - Routing
    - Link weights assignment
    - Routing computation
      - *Distributed distance vector protocols*
        - *synchronous Bellman-Ford (SBF)*
          - SBF/∞
          - SBF/-1 SBF/∞

SBF at another Initial Configuration: SBF/-1

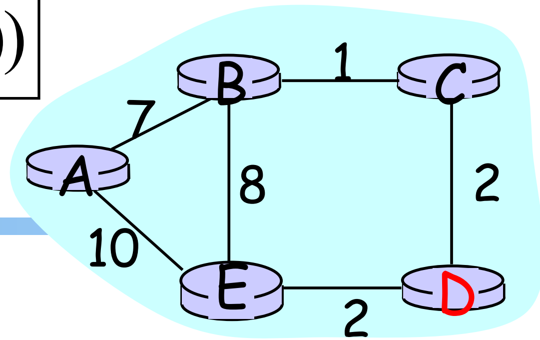


- Initialization (time 0):

$$d_i(0) = \begin{cases} 0 & \text{i} = \text{dest} \\ -1 & \text{otherwise} \end{cases}$$

$$d_i(h+1) = \min_{j \in N(i)} (d_{ij} + d_j(h))$$

Example



Consider D as destination

	A	B	C	E	D
d(0)	-1	-1	-1	-1	0
d(1)	6	0	0	2	0
d(2)	7	1	1	2	0
d(3)	8	2	2	2	0
d(4)	9	3	3	2	0
d(5)	10	3	3	2	0
d(6)	10	3	3	2	0

Observation: d(0) ≤ d(1) ≤ d(2) ≤ d(3) ≤ d(4) ≤ d(5) = d(6) = d*

Correctness of SBF/-1

- SBF/-1 converges due to monotonicity

- Remaining question:
  - Can we guarantee that SBF/-1 converges to shortest path?

Correctness of SBF/-1

- Common between SBF/∞ and SBF/-1: they solve the Bellman equation

$$d_i = \min_{j \in N(i)} (d_{ij} + d_j)$$

where $d_D = 0$.

- We have proven SBF/∞ is the shortest path solution.
- SBF/-1 computes shortest path if Bellman equation has a unique solution.

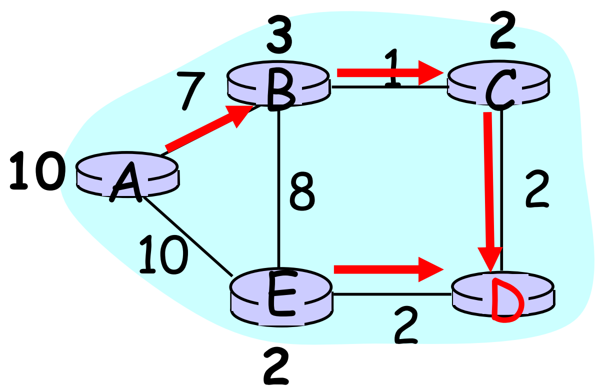
$$d_i = \min_{j \in N(i)} (d_{ij} + d_j)$$

Uniqueness of Solution to BE

- Assume another solution d, we will show that d = d*

case 1: we show d ≥ d*

Since d is a solution to BE, we can construct paths as follows: for each i, pick a j which satisfies the equation; since d* is shortest, d ≥ d*



$$d_i = \min_{j \in N(i)} (d_{ij} + d_j)$$

Uniqueness of Solution to BE

Case 2: we show $d \leq d^*$

assume we run SBF with two initial configurations:

- one is d
- another is SBF/∞ (d^∞),

-> monotonicity and convergence of SBF/∞ imply that $d \leq d^*$

Discussion

- Will SBF converge under other non-negative initial conditions?

- Problems of running *synchronous* BF?

Outline

- Admin and recap
- Network overview
- Network control plane
  - Routing
    - Link weights assignment
    - Routing computation
      - *Distributed distance vector protocols*
        - synchronous Bellman-Ford (SBF)
        - *asynchronous Bellman-Ford (ABF)*

Asynchronous Bellman-Ford (ABF)

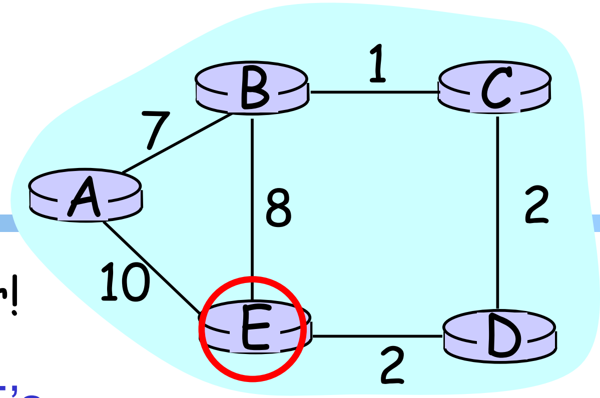
- No notion of global iterations
  - each node updates at its own pace
- Asynchronously each node i computes

$$d_i = \min_{j \in N(i)} (d_{ij} + d_j^i)$$

using last received value d^i_j from neighbor j.

- Asynchronously node j sends its estimate to its neighbor i:
  - We assume that there is an upper bound on the delay of estimate packet

ABF: Example



Below is just one step! The protocol repeats forever!

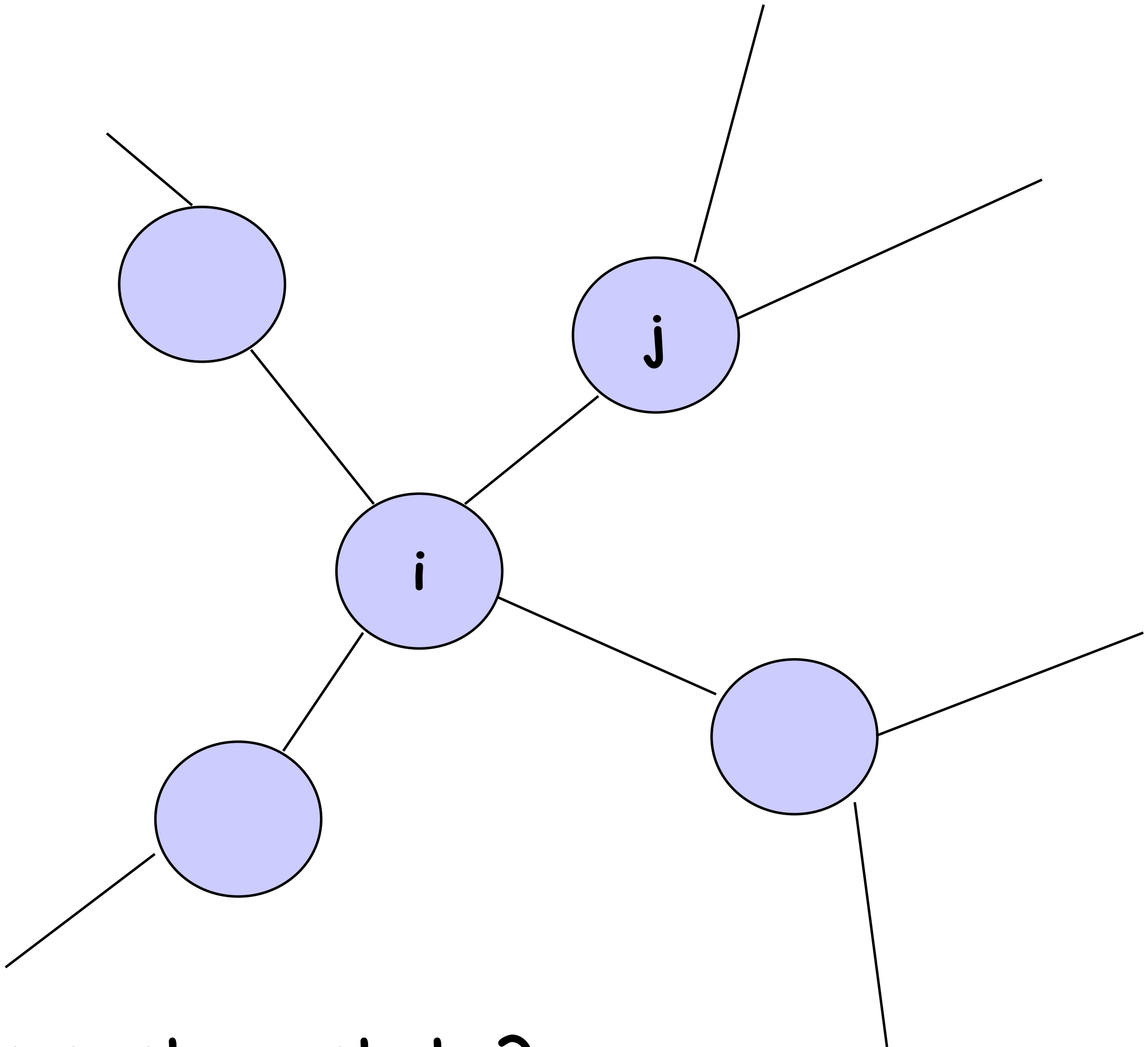
	distance tables from neighbors			computation			E's distance table	distance table E sends to its neighbors
$d_E()$	A	B	D	A	B	D		
A	0	7	∞	10	15	∞	A: 10	A: 10
B	7	0	∞	17	8	∞	B: 8	B: 8
C	∞	1	2	∞	9	4	D: 4	C: 4
D	∞	∞	0	∞	∞	2	D: 2	D: 2
	10	8	2					E: 0

destinations

Asynchronous Bellman-Ford (ABF)

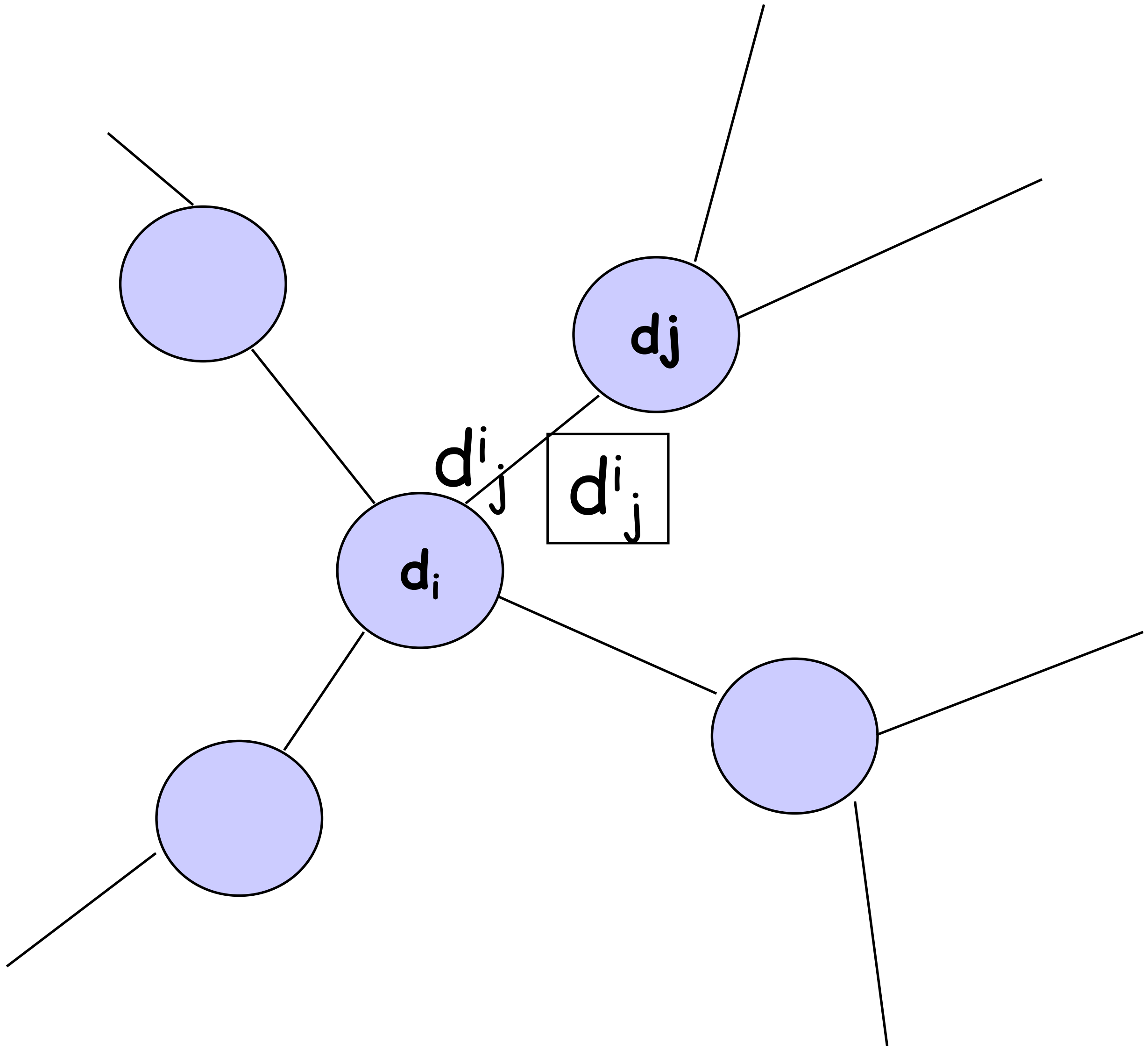
- ABF will eventually converge to the shortest path
  - links can go down and come up – but if topology is stabilized after some time t and connected, ABF will eventually converge to the shortest path !

ABF Convergence Proof Complexity: Complex System State



What is system state?

System State



three types of distance state from node j:

- d_j: current distance estimate state at node j

- d^i_j: last d_j that neighbor i received

- d^i_j: those d_j that are still in transit to neighbor i

ABF Convergence Proof: The Sandwich Technique

- Basic idea:
  - bound system state using extreme states
- Extreme states:
  - SBF/∞; call the sequence U()
  - SBF/-1; call the sequence L()

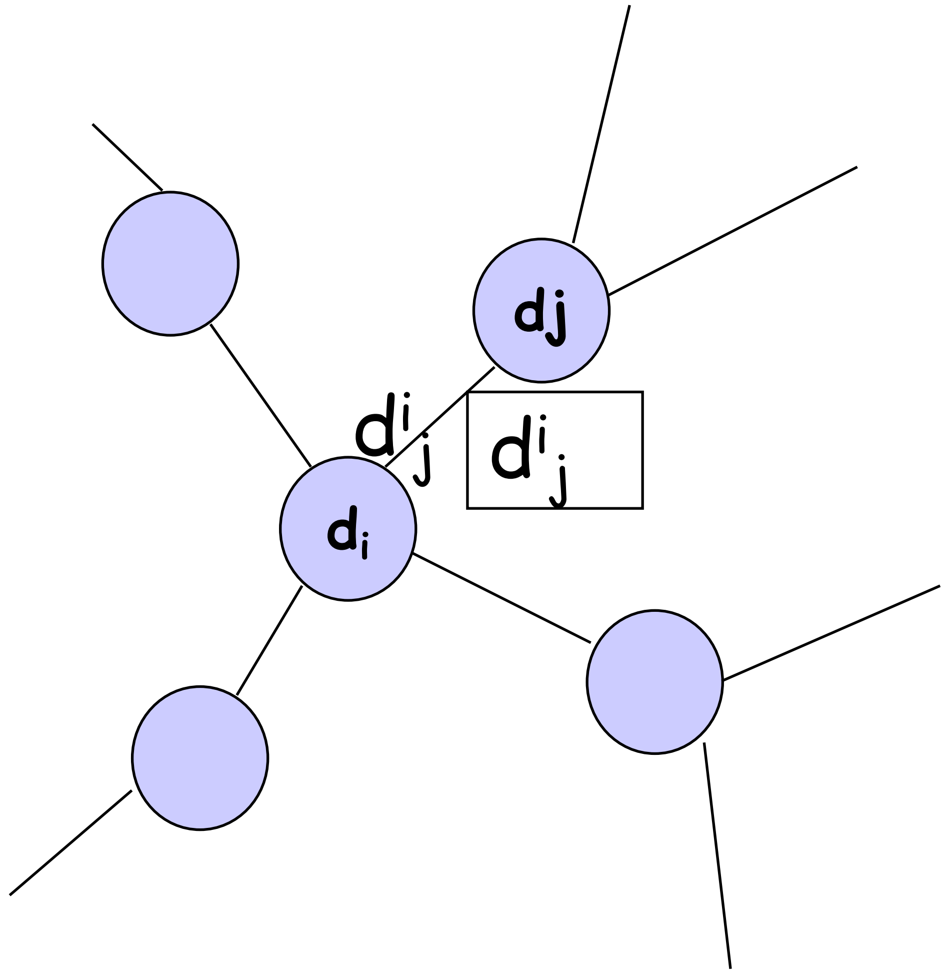
ABF Convergence

- Consider the time when the topology is stabilized as time 0

- U(0) and L(0) provide upper and lower bounds at time 0 on all corresponding elements of states
  - $L_j (0) \le d_j \le U_j (0)$ for all d_j state at node j
  - $L_j (0) \le d^i_j \le U_j (0)$
  - $L_j (0) \le$ `update messages` $d^i_j \le U_j (0)$

ABF Convergence

- d_j
  - after at least one update at node j: d_j falls between $L_j(1) \le d_j \le U_j(1)$
- d^i_j :
  - eventually all d^i_j that are only bounded by $L_j(0)$ and $U_j(0)$ are replaced with in $L_j(1)$ and $U_j(1)$



Asynchronous Bellman-Ford: Summary

- Distributed
  - each node communicates its routing table to its directly-attached neighbors
- Iterative
  - continues periodically or when link changes, e.g. detects a link failure
- Asynchronous
  - nodes need *not* exchange info/iterate in lock step!
- Convergence
  - in finite steps, independent of initial condition if network is connected

Summary: Distributed Distance-Vector

- Tool box: a key technique for proving convergence (liveness) of distributed protocols: monotonicity and bounding-box (sandwich) design
  - Consider two configurations d(t) and d'(t):
    - if d(t) <= d'(t), then d(t+1) <= d'(t+1)
  - Identify two extreme configurations to sandwich any real configurations

Outline

- Admin and recap
- Network control plane
  - Routing
    - Link weights assignment
    - Routing computation
      - Distance vector protocols (distributed computing)
        - synchronous Bellman-Ford (SBF)
        - asynchronous Bellman-Ford (ABF)
        - *properties of DV*

Properties of Distance-Vector Algorithms

- Good news propagate fast

A	B	C	D	E	
	∞	∞	∞	∞	Initially
	1	∞	∞	∞	After 1 exchange
	1	2	∞	∞	After 2 exchanges
	1	2	3	∞	After 3 exchanges
	1	2	3	4	After 4 exchanges

Properties of Distance-Vector Algorithms

- Bad news propagate slowly

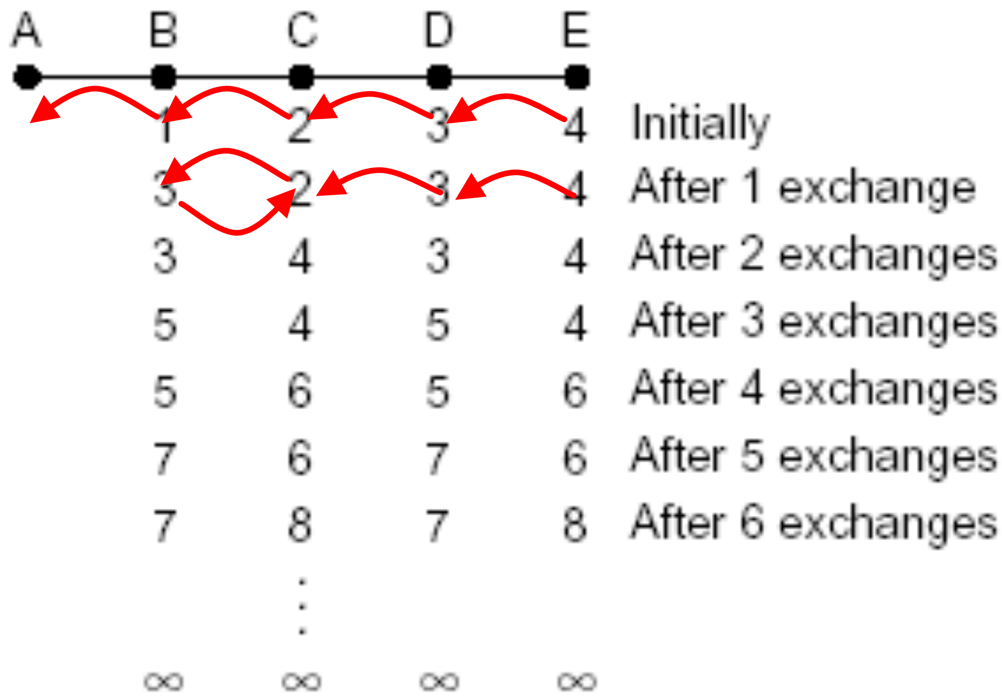
A-B link down

A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	∞	∞	∞	∞	

- This is called the *counting-to-infinity* problem
- Q: what causes counting-to-infinity?

Counting-To-Infinity is Because of Routing Loop

- Counting-to-infinity is caused by a routing loop, which is a global state (consisting of the nodes' local states) at a global moment (observed by an oracle) such that there exist nodes A, B, C, … E such that A (locally) thinks B as next hop, B thinks C as next hop, … E thinks A as next hop



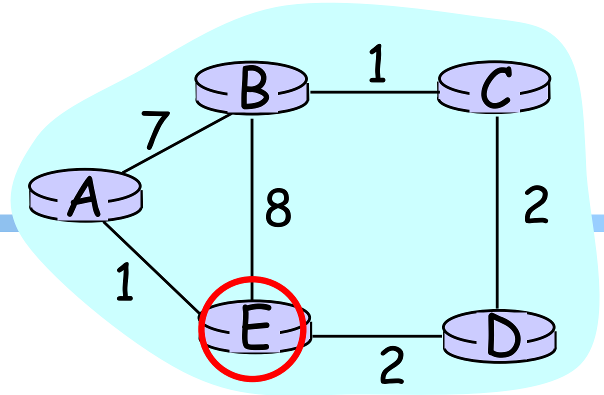
Discussion

- Why avoid routing loops is hard?
- Any proposals to avoid routing loops?

Outline

- Admin and recap
- Network control plane
  - Routing
    - Link weights assignment
    - Routing computation
      - Distance vector protocols (distributed computing)
        - synchronous Bellman-Ford (SBF)
        - asynchronous Bellman-Ford (ABF)
        - properties of DV
          - DV w/ loop prevention
            - *reverse poison*

The Reverse-Poison (Split-horizon) Hack



If the path to dest is through neighbor h, report ∞ to neighbor h for dest.

$D^E()$	distance tables from neighbors: A	B	D	computation: A	B	D	E's distance table
A	0	7	∞	1	15	∞	1, A
B	7	0	∞	8	8	∞	8, B
C	∞	1	2	∞	9	4	4, D
D	∞	∞	0	∞	∞	2	2, D
	1	8	2				
	c(E,A)	c(E,B)	c(E,D)				

destinations

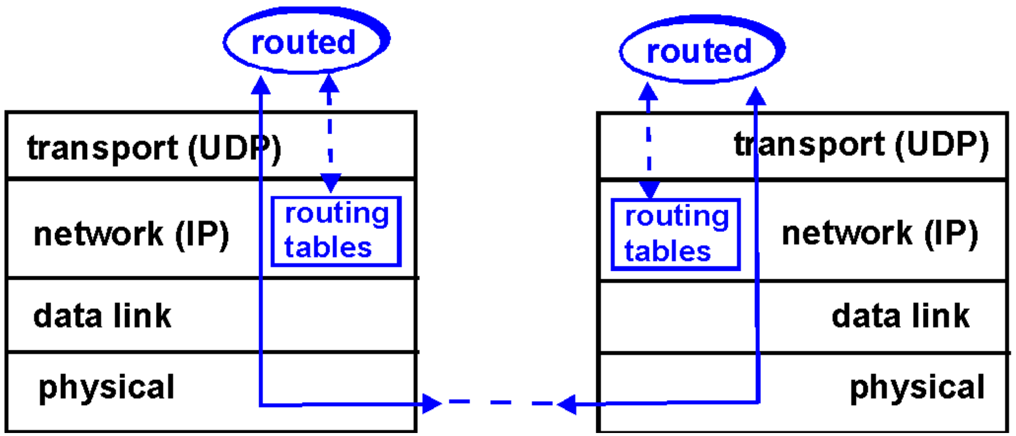
distance through neighbor

distance table E sends to its neighbors

To A	To B	To D
A: ∞	A: 1	A: 1
B: 8	B: ∞	B: 8
C: 4	C: 4	C: ∞
D: 2	D: 2	D: ∞
E: 0	E: 0	E: 0

DV+RP => RIP
(Routing Information Protocol)

- Included in BSD-UNIX Distribution in 1982
- Link cost: 1
- Distance metric: # of hops
- Distance vectors
  - exchanged every 30 sec via Response Message (also called **advertisement**) using UDP
  - each advertisement: route to up to 25 destination nets



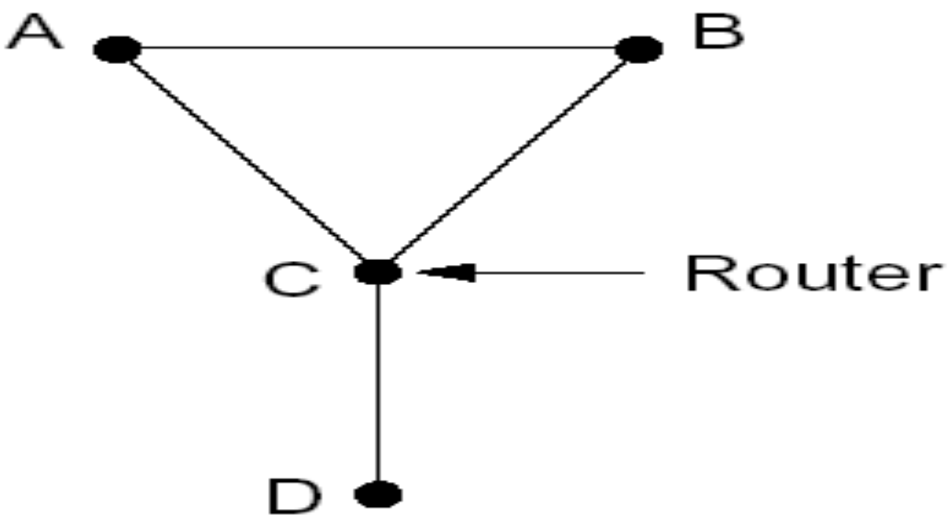
RIP: Link Failure and Recovery

If no advertisement heard after 180 sec --> neighbor/link declared dead

- routes via neighbor invalidated
- new advertisements sent to neighbors
- neighbors in turn send out new advertisements (if tables changed)
- link failure info quickly propagates to entire net
- reverse-poison used to prevent ping-pong loops
- set infinite distance = 16 hops (why?)

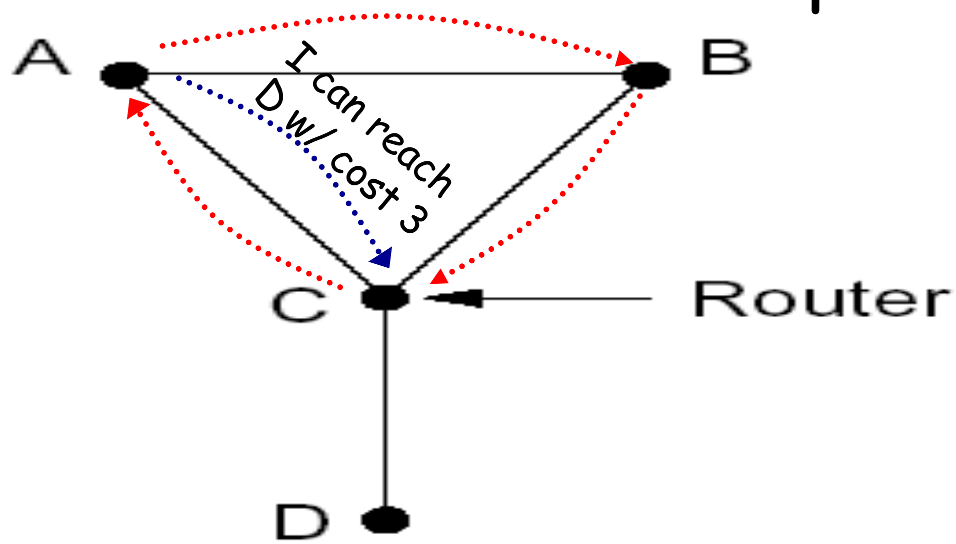
General Routing Loops and Reverse-poison

- Exercise: Can Reverse-poison guarantee no loop for this network?



General Routing Loops and Reverse-poison

- Reverse-poison removes two-node loops but may not remove more-node loops



- Unfortunate timing can lead to a loop
  - When the link between C and D fails, C will set its distance to D as ∞
  - A receives the bad news (∞) from C, A will use B to go to D
  - A sends the news to C
  - C sends the news to B

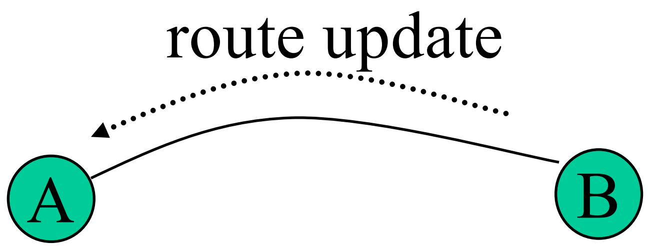
Outline

- Admin and recap
- Network control plane
    - Routing
        - Link weights assignment
        - Routing computation
            - Distance vector protocols (distributed computing)
                - synchronous Bellman-Ford (SBF)
                - asynchronous Bellman-Ford (ABF)
                - properties of DV
                    - DV w/ loop prevention
                        - reverse poison
                        - *destination-sequenced DV (DSDV)*

Destination-Sequenced Distance Vector protocol (DSDV)

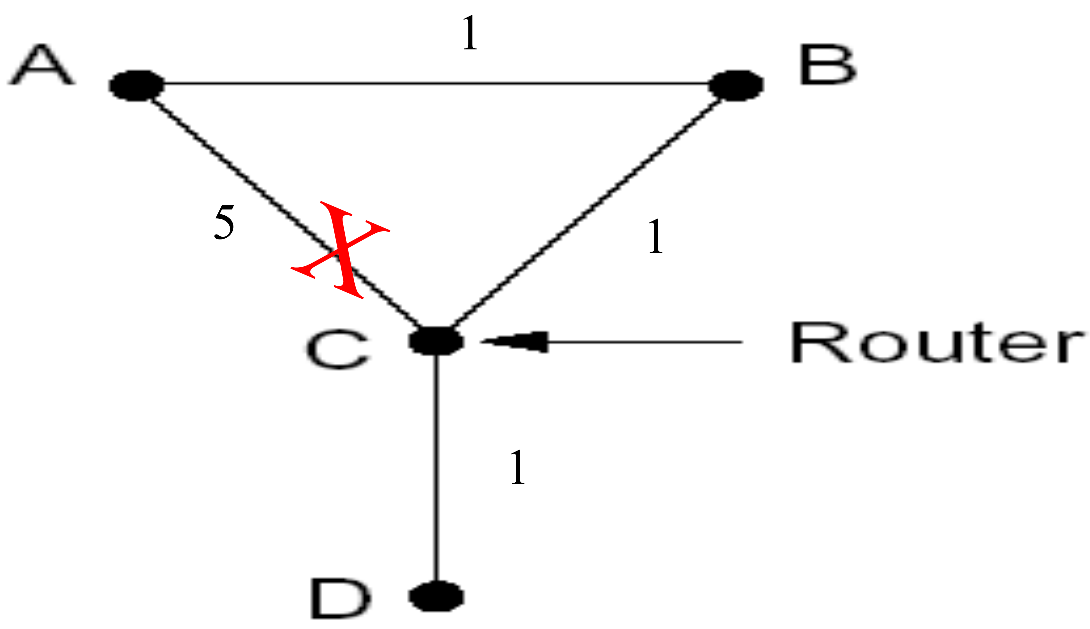
- Basic idea: use sequence numbers to partition computation
  - tags each route with a sequence number
  - each destination node D periodically advertises monotonically increasing even-numbered sequence numbers
  - when a node realizes that the link it uses to reach destination D is broken, it advertises an infinite metric and a sequence number which is one greater than the previous route (i.e., an odd seq. number)
    - the route is repaired by a later even-number advertisement from the destination

DSDV: More Detail



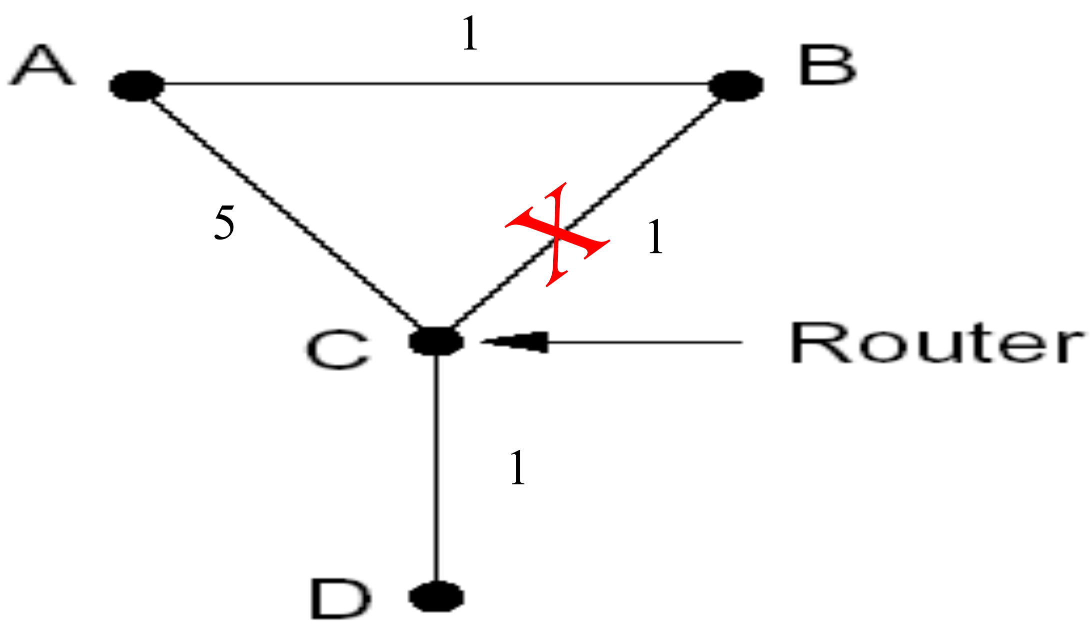
- Let's assume the destination node is D
- There are optimizations but we present a simple version:
  - each node B maintains (S^B, d^B), where S^B is the sequence number at B for destination D and d^B is the best distance using a neighbor from B to D
- Both periodical and triggered updates
  - periodically: D increases its seq. by 2 and broadcasts with (S^D, 0)
  - if B is using C as next hop to D and B discovers that C is no longer reachable
    - B increases its sequence number S^B by 1, sets d^B to ∞, and sends (S^B, d^B) to all neighbors

Example



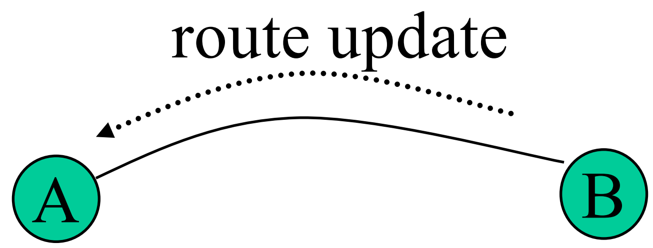
Will this trigger an update?

Example



Will this trigger an update?

DSDV: Update Alg.



- Consider simple version, no optimization
- Update after receiving a message
  - assume B sends to A its current state (S^B, d^B)
  - when A receives (S^B, d^B)
    - if $S^B > S^A$, then
      // always update if a higher seq#
      - » $S^A = S^B$
      - » if $(d^B == \infty)$ $d^A = \infty$; else $d^A = d^B + d(A,B)$
    - else if $S^A == S^B$, then
      - » if $d^A > d^B + d(A,B)$
        // update for the same seq# only if better route
        $d^A = d^B + d(A,B)$ and uses B as next hop