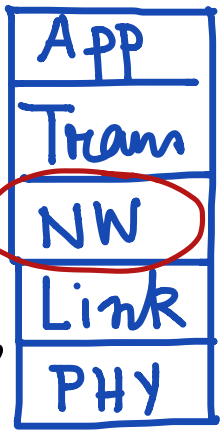
Routing

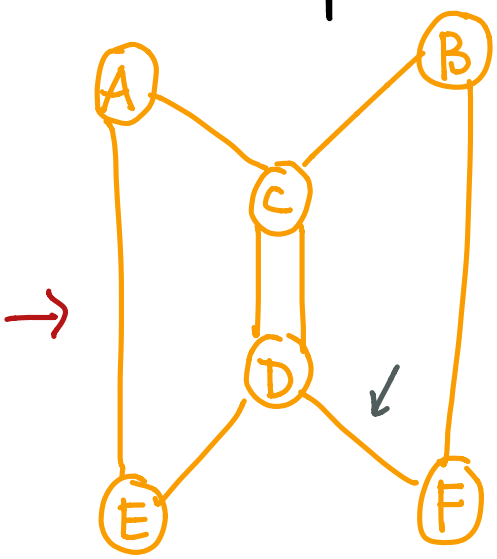


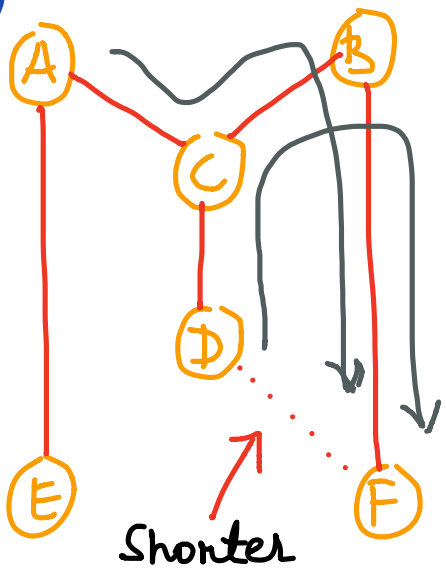
- So far discussed about packet forwarding
- Today: routing, i.e.
  How is that forwarding table created?

Forwarding is a local process

Routing is a global process - needs agreement among all nodes that are involved in it.

Past Example: Spanning Tree





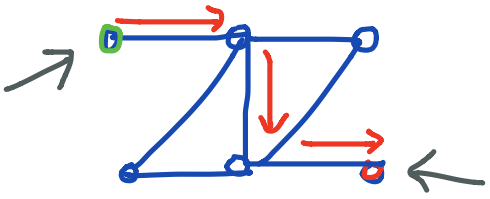
Shorter path available.

Obs: Spanning Tree algorithm provided a route, not necessarily an efficient route

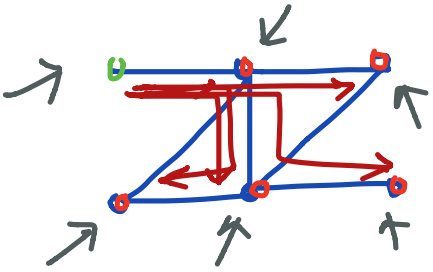
- Our goal in routing is to find "best" paths - e.g., least costly path.

Delivery models:

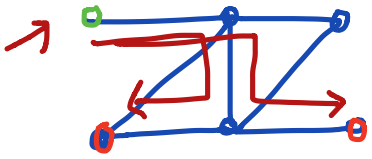
① Unicast



② Broadcast

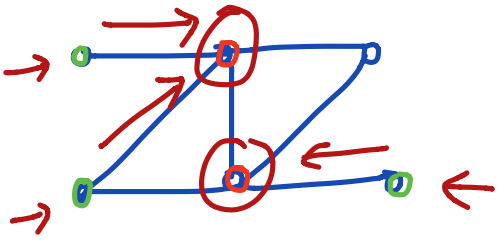


③ Multicast



④ Anycast

e.g., DNS, CDN



Objectives of Routing Algorithms

1. Correctness – from source to destination
2. Efficient path finding – least cost
3. Fast convergence – recovers quickly after topology changes
4. Scalability – complexity of the algorithm not too high.

Usually,

Routing is done in a decentralized, distributed setting

[these two are independent properties]

- no central controller
- nodes can locally exchange messages with their neighbors
- nodes work in parallel
- they are susceptible to failures

Topics to cover

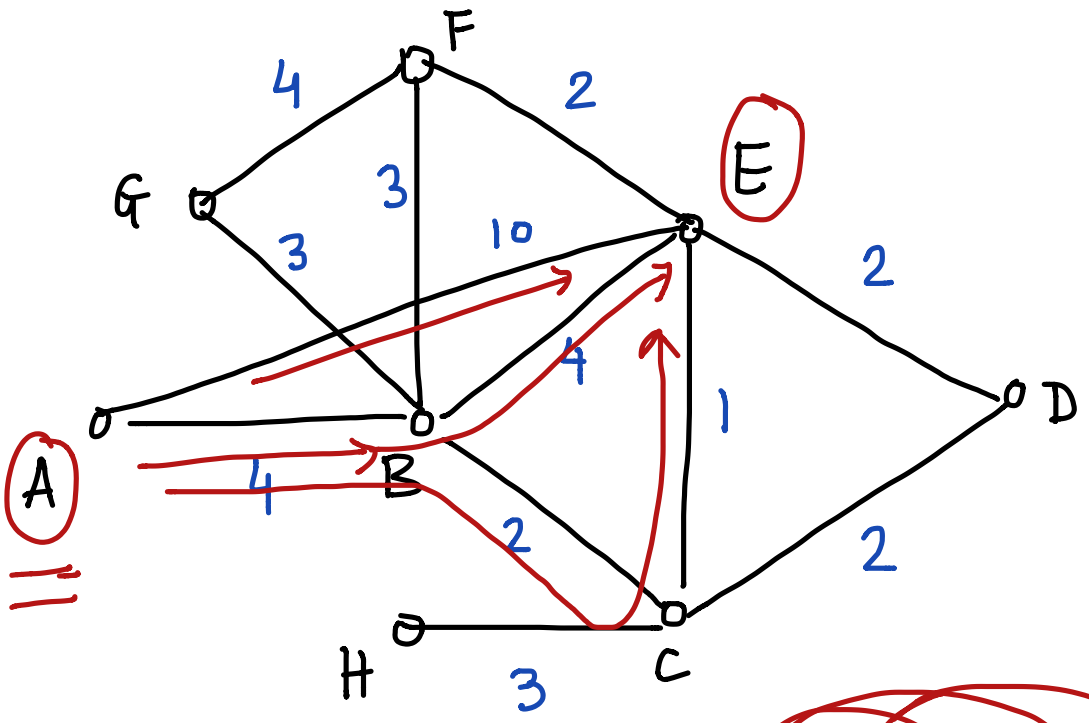
- Shortest Path Routing
  - Dijkstra's algorithm
- ✓ Distance Vector Routing
- ✓ Link-state routing
- ✓ Equal-cost multi path
- ✓ Inter-domain routing
  - Border Gateway Protocol (BGP)

Shortest Path Routing

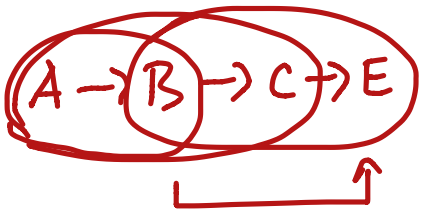
- one way of quantifying the "efficient" path
- "cost" reflects lot of factors, e.g., latency, expense of a link.
- ☆ - but only static factors topology is fixed
  - we won't consider dynamic factors like the load of a link.
- costs are additive over links ↓ ↓

$$\text{cost}(A \to B \to C) = \text{cost}(A \to B) + \text{cost}(B \to C)$$

Ex.



- Find shortest path A → E
- all links are bidirectional.



Observation:

Every subpath of a shortest path is also a shortest path

Comes as a corollary of the additive cost property

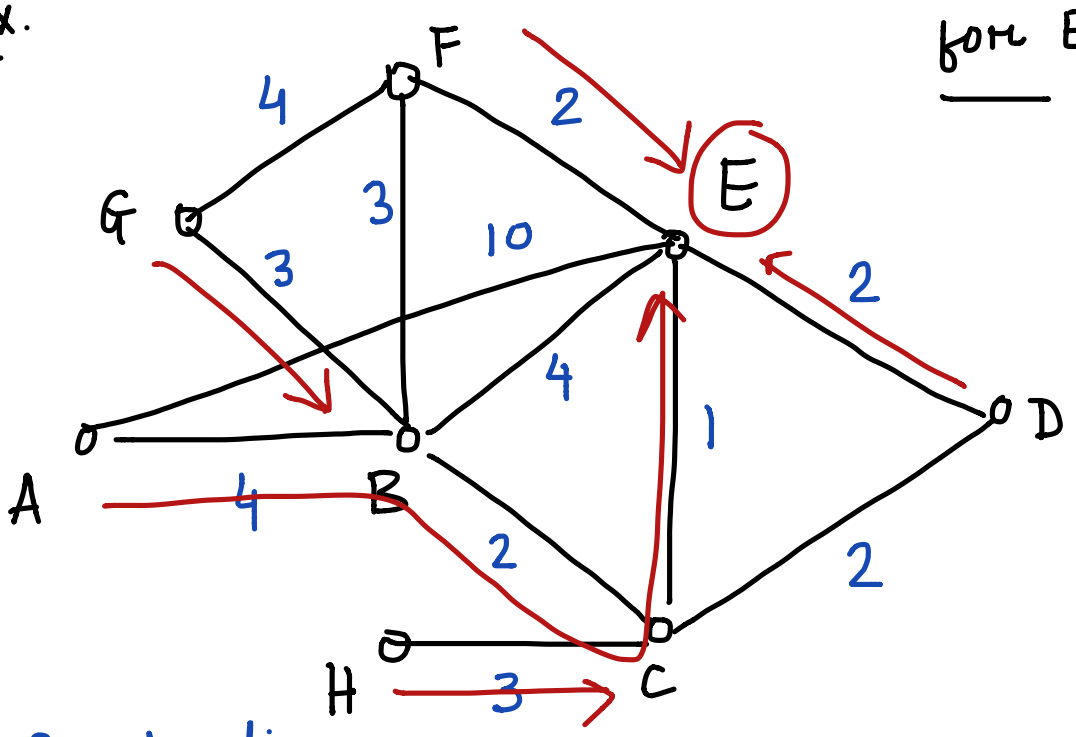
— Proof idea: if the subpath wasn't the shortest the original path wouldn't be shortest either.

Sink Trees / Source Trees

- Sink tree for a destination is the union of all shortest paths towards that destination
- Similarly source tree ✓

Ex.

Sink tree for E



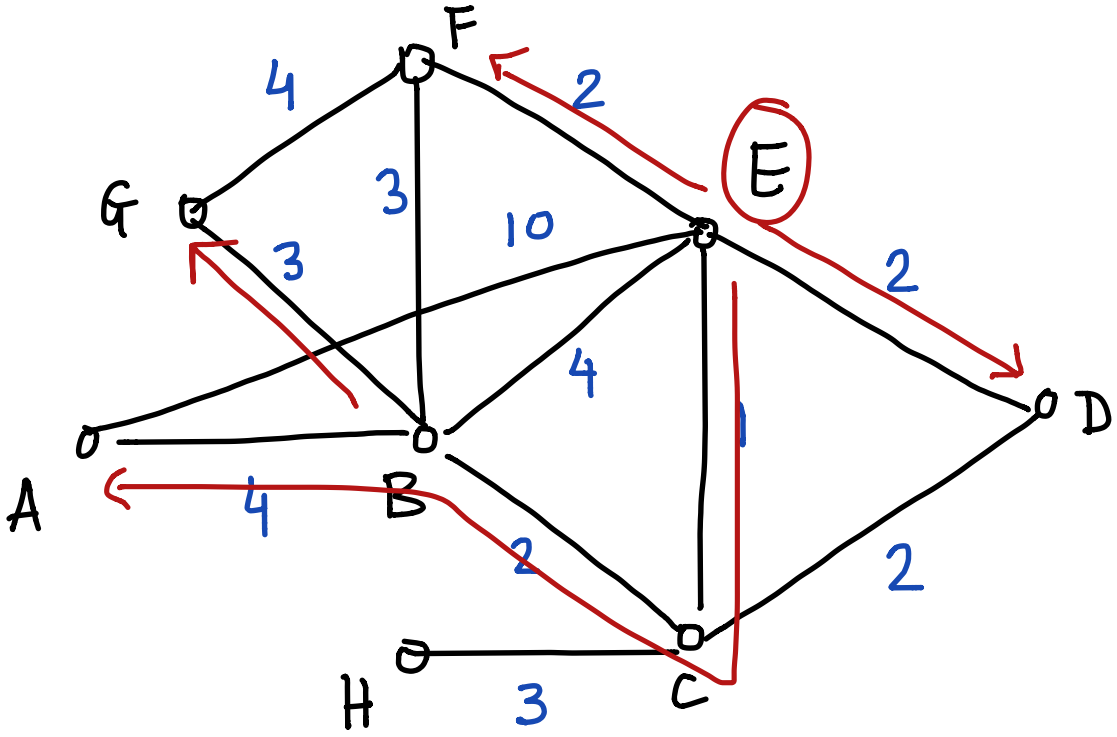
Implications

- Only need the destination info
- Each node only needs to send to the next hop node in the sink tree
- Forwarding tables are enough.

Algorithms to compute shortest paths

Q: How to compute shortest paths from a source to all destinations?

– Dijkstra's algorithm

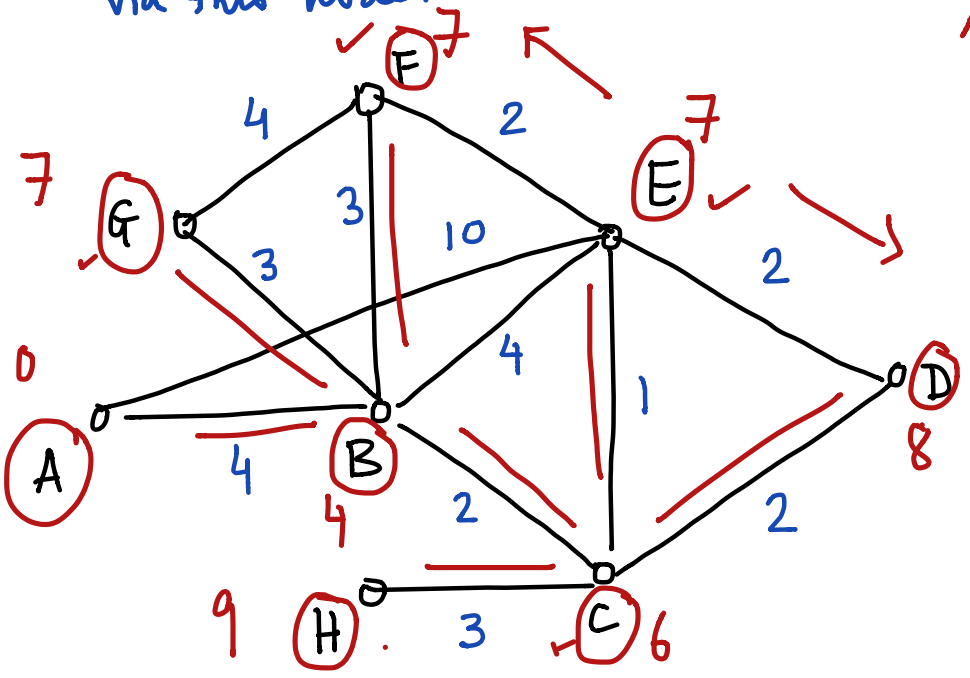


Ex: Create the forwarding table for E.

Algorithm (Dijkstra)

① Initialization: all nodes tentative
distance = 0 for source, ∞ for all others

② While ∃ tentative node
- find the node with lowest distance from the frozen nodes (break ties arbitrarily)
- add link to that node into the shortest path tree
- relax the distances of the neighbors of this newly added node, i.e., find the shortest paths to those nodes via this node.



A		
A	0	A
B	4	B
C	6	B
D	8	B
E	7	B
F	7	B
G	7	B
H	9	B

Dijkstra endnotes

- to find the shortest paths from every node to every other node, we need to run this for different sources
  - can be done carefully to minimize repetitions
- Runtime depends on the implementation of the FIND_MIN_COST_NODE function
  e.g., using fibonacci heap min-priority queue it is $O(|E| + |V| \log |V|)$ ✓
- It is a centralized algorithm
- ⊖ Does not work with negative edge weights.

  Dijkstra is a greedy algorithm, fails when the costly path has an eventual negative edge later.

Distance Vector Algorithm ✓

✓ – simple, decentralized, distributed version of the Bellman-Ford alg.

[Bellman-Ford fixes the negative edge problem of Dijkstra, but computationally expensive]

– works, but has slow convergence among other limitations

– Link-state algorithms are used instead in practice

Unlike the Dijkstra setting

(1) Nodes only know the costs to their neighbors, not the topology

(2) can talk only to neighbors

(3) can run algorithm concurrently

(4) can fail and messages may be lost.

local info

Algorithm

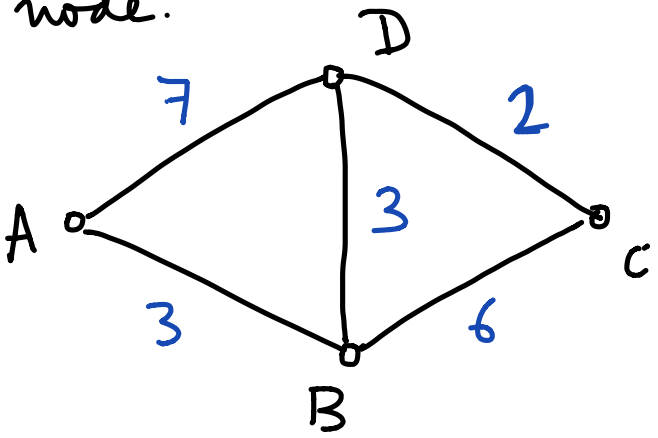
- Each node maintains a distance vector (distances, via nodes) to all destinations

(1) Initialize with cost = 0 to self, ∞ to all others

(2) Send the updated distance vector to neighbors

(3) Update the vector for each destination by selecting the min (distance from neighbors + cost to neighbors)
   - that min-cost neighbor is the via node.

Ex.

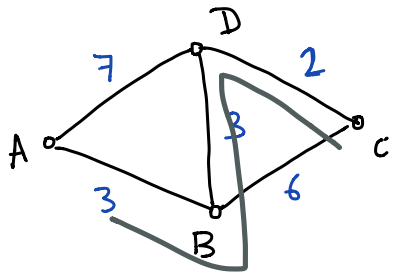


Distance vectors

	A		B		C		D	
	cost	via	cost	via	cost	via	cost	via
A	0	A	3	A	8	D	6	B
B	3	B	0	B	6	B	3	B
C	8	B	5	D	0	C	2	C
D	7	D	3	D	2	D	0	D

Announcements

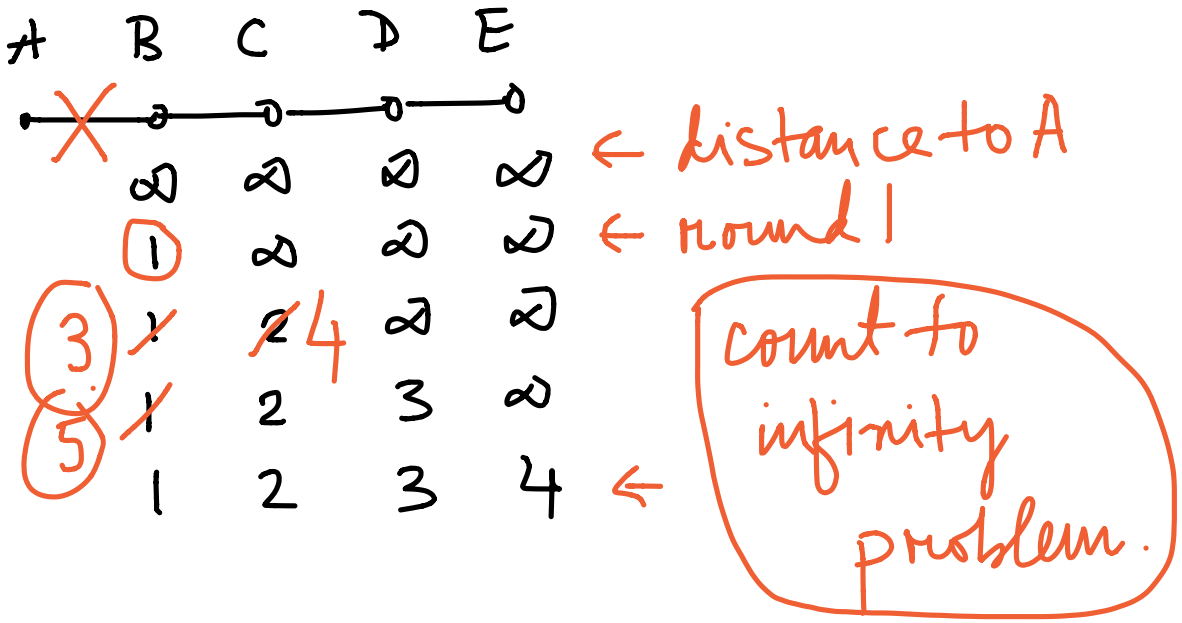
to	A	B	C	D
A	0	3	8	6
B	3	0	6	3
C	8	5	0	2
D	7	3	2	0



Dynamics of DV

- Distance vector does local update after link failure, forgets the failed node [sets distance ∞]
- One hop per exchange

Partitions are a problem



Heuristics are used to mitigate them
e.g., "split horizon, poison reverse"
don't send updates to the node where it is learned from

not very effective ✓

- instead link state algorithms are favored.

Used in: RIP (Routing Information Protocol)