CSE 3231
Computer Networks

Chapter 5
The Network Layer
part 4

William Allen, PhD

Spring 2022

Fairness vs. Efficiency

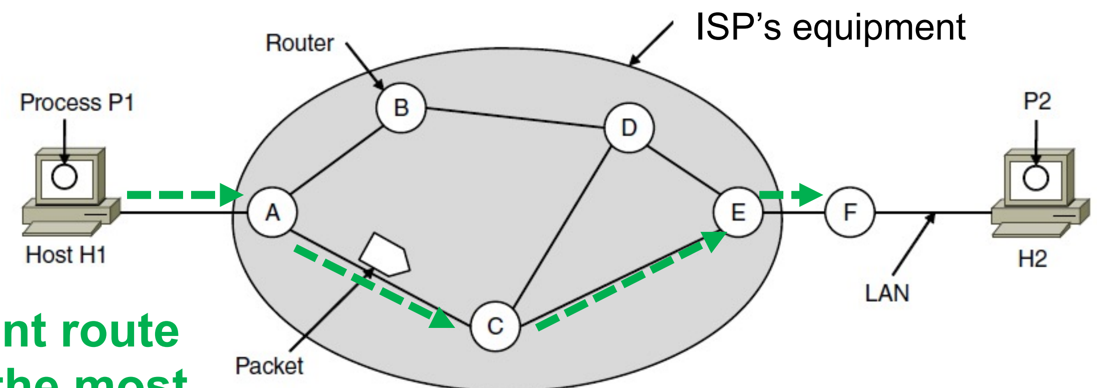
- Fairness:
  - ensure that all users/applications receive a fair share of the network's resources
    - fair queuing - using an algorithm to decide which application transmits data next
    - congestion control - prevent sources from interfering with transmissions by others
- Efficiency:
  - maximizing throughput and/or the utilization of radio channels
    - "*goodput*" - refers to % of usable data delivered

Network Layer Routing

Forwarding: sending packets along a known path

Routing: the process of discovering network paths

- Model the network as a graph of nodes and links
- Decide what to optimize (e.g., fairness vs efficiency)
- Update routes for changes in topology (e.g., failures)



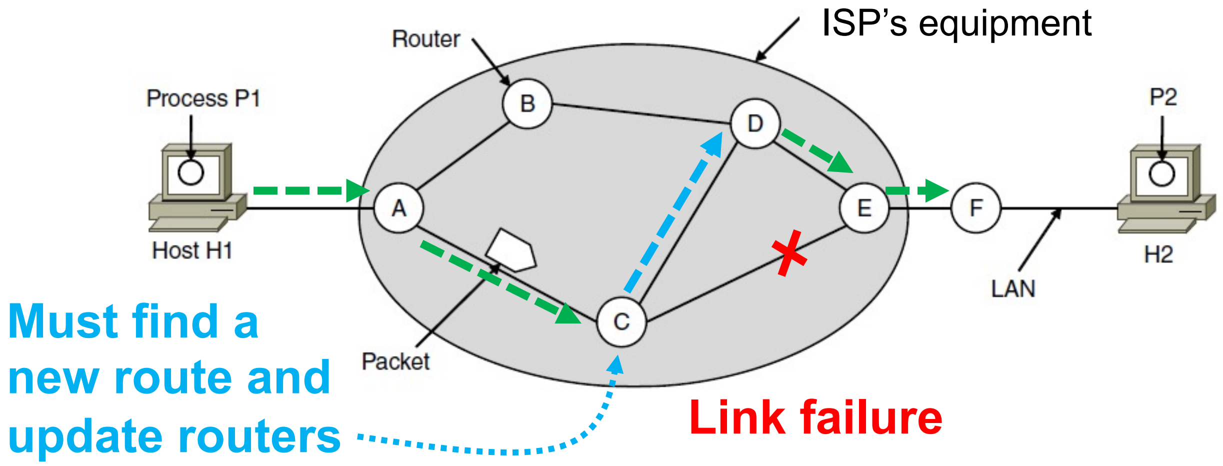
Current route over the most efficient path

Network Layer Routing

Forwarding: sending packets along a known path

Routing: the process of discovering network paths

- Model the network as a graph of nodes and links
- Decide what to optimize (e.g., fairness vs efficiency)
- Update routes for changes in topology (e.g., failures)



Routing and Forwarding

Routing:

- process by which the routing table is built

- Routing table
  - used by the routing algorithm to build the forwarding table
  - generally contains mapping from network number to next hop

Forwarding:

- selecting an output port (interface) based on the destination address and entries in the routing table

- Forwarding table
  - used when a packet is being forwarded, it must contain enough information to accomplish the forwarding function
  - forwarding table contains mapping from a network number to an outgoing interface and the Ethernet address of next host/router (to deliver packets over a link we need the physical address)

Routing and Forwarding

- Example of a *routing table* entry **(a)**
  - Routing is done at the Network Layer and the table shows the IP address of the next node in the path
- Example of a *forwarding table* entry **(b)**
  - Forwarding is done at the Data Link Layer, the table shows the output interface and the MAC address of the next node

(a)

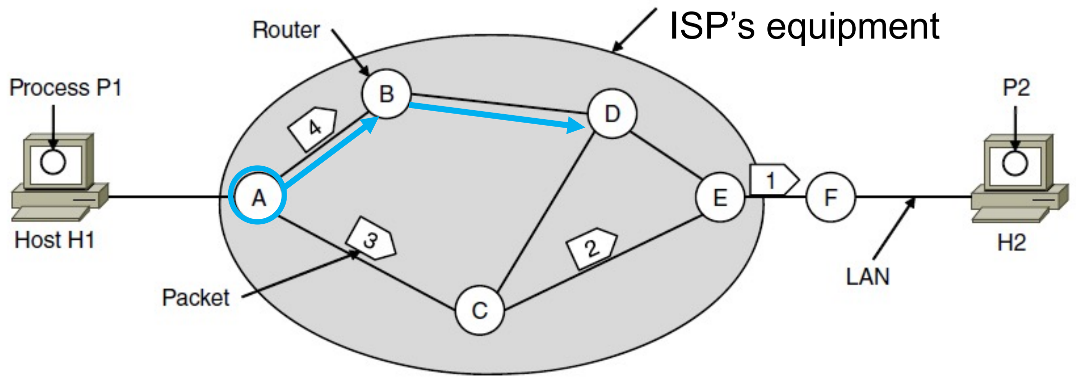
Network	Next Hop
171.69.245.0/24	171.69.245.10

(b)

Network	Interface	MAC Address
171.69.245.0/24	if0	8:0:2b:e4:b:1:2

Connectionless Service – Datagrams

- A packet is forwarded using the destination address in its header
  - Each router decides how to forward packets
  - Different packets may take different paths if the topology changes



For example:
When links from A thru C became congested, new packets to E and F are routed thru B (and D)

Each node will have a table of paths from their location

A's table

Destination	Next Node
to B	use B
to C	use C
to D	use B
to E	use B
to F	use B

What is the Best Way to Deliver Packets Across the Network?

We need a way to deliver packets from any node to any other node

1. The sending node will send the packet to its local router
2. Packets will pass from router to router across the network until they reach the router at the destination node’s network
3. That router will deliver the packet to the destination node in its local network

One Approach: Flooding

A simple method where each router sends an incoming packet out all of the router's ports

- This does not find routes, it just delivers packets

Obviously, this makes many copies of the packet and fills the network with traffic

- A *hop count* can be used to drop packets, but the number of hops is unknown so the limit must be set high enough to cross the network
- A better way has routers discard packets they have seen at least once, but adds overhead to keep track of the packets that have been seen

A Better Way

Similar to the *Spanning Tree* we saw earlier, we would like to have a pre-selected path to follow instead of producing excess traffic

However, this path is potentially temporary since IP networks have to be able to adjust to link or router failures

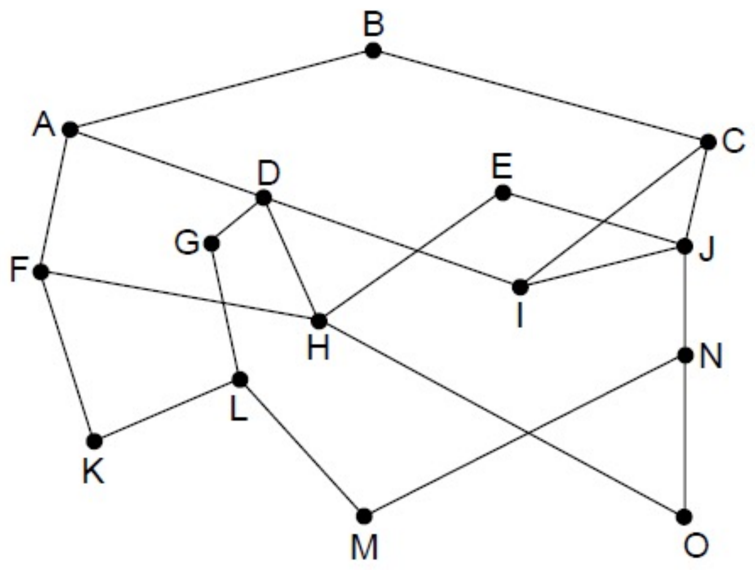
We need an algorithm that can find a new path from one router to another when the old route is no longer suitable

But, it needs to do it efficiently

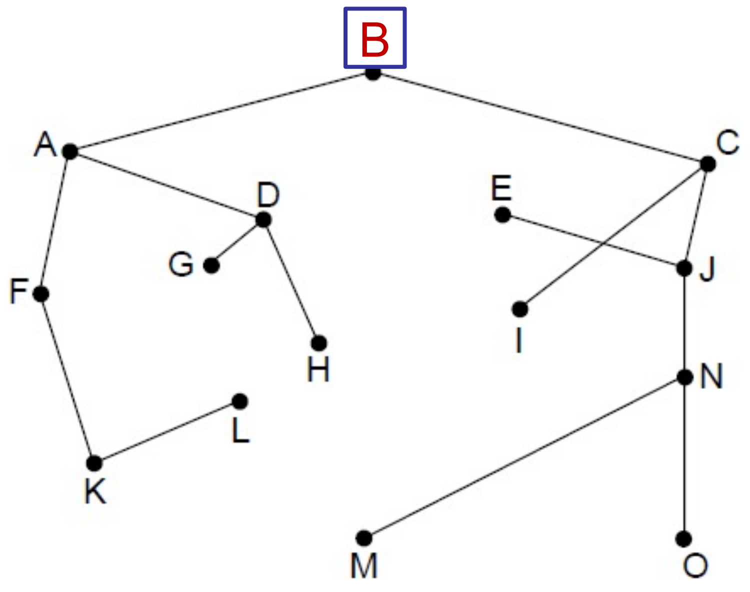
The Optimality Principle

Each portion of a best path is also a best path

- *Best* means fewest hops between nodes
- Proof: if any of the individual sub-paths is not the best possible sub-path, then replace it with a better sub-path. However, the overall path is the best path, so no better sub-path can exist.



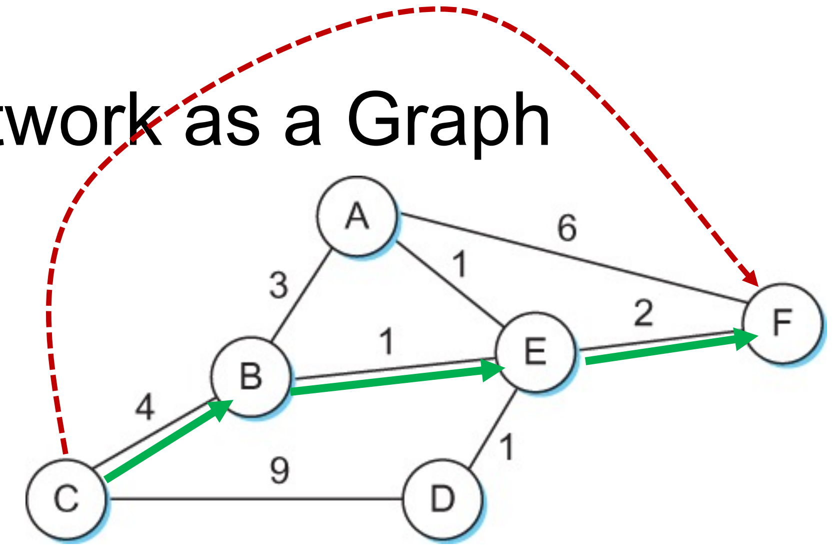
Sink Tree for the Network



Tree of best paths to router B

Routing

View the Network as a Graph



- The basic problem of routing is to find the lowest-cost path between any two nodes
  - We want to find the shortest path from C to F
  - The cost of a path equals the sum of the costs of all the edges that make up the path

    C → B (4), B → A (3), A → F (6), total cost: 13

    C → D (9), D → E (1), E → F (2), total cost: 12

    C → B (4), B → E (1), E → F (2), total cost: 7

Calculating the Shortest Path

- We could use an approach like Dijkstra’s Algorithm to calculate all shortest paths and store them in a routing table on each node
- However, this creates *static routing* tables and has several shortcomings:
  - It does not deal with node or link failures
  - It does not consider the addition of new nodes or links
  - It implies that edge costs cannot change
- We really need a *dynamic* and *distributed* approach that will update the routing tables when changes occur in the network

Dynamic Routing

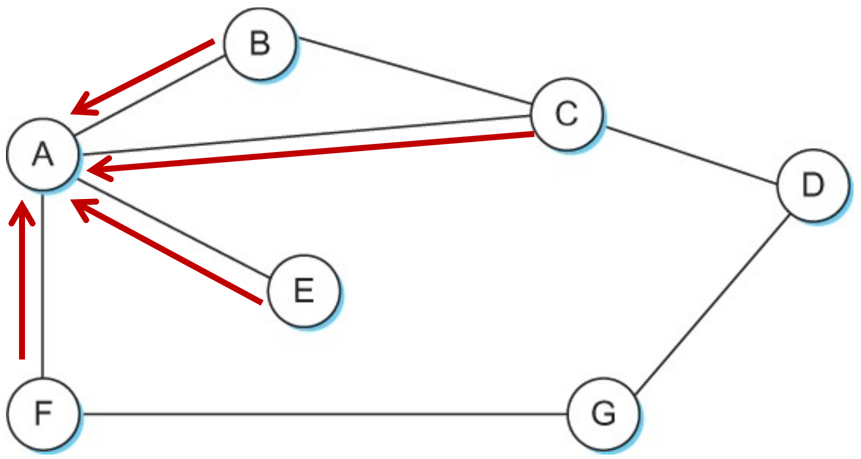
- There are two main classes of routing protocols:
  - *Distance Vector*
    - Routers have a table with the shortest distance to each destination known at that point in time
    - Routers exchange information on changes with their neighbors and use that to update their tables
  - *Link State*
    - Routers gather information on link costs to their neighbors and send that to all other routers
    - When they receive that information from the other routers, they update their routing tables with the shortest path to those other routers

Benefit of these Routing Algorithms

- These two routing approaches don't have to recalculate all paths outside of their local area to update their routing tables
  - changes "ripple" through the network over time by passing from one area to another through neighbors
- *Distance Vector*
  - While the initial table requires learning the best routes to all nodes, updates only include local changes and are faster while producing less traffic
- *Link State*
  - Tables are updated by exchanging information with neighbors, not from across the whole network

Distance Vector Routing

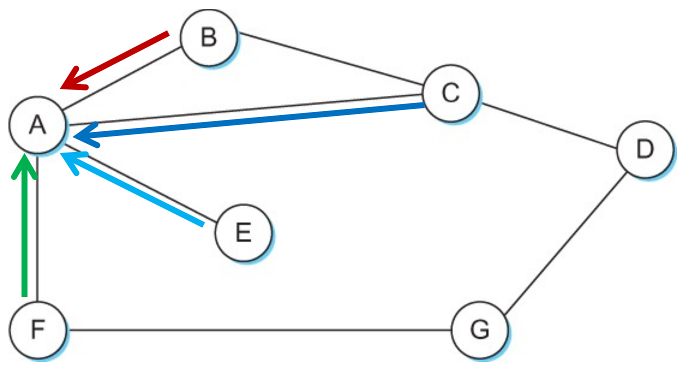
- Each node constructs a one dimensional array (a vector) containing the "distances" (costs) to all neighbor nodes and distributes that vector to its immediate neighbors
  - Starting assumption is that each node knows the cost of the link to each of its directly connected neighbors



Distance Vector Routing

- The distance vector routing algorithm is also known as the Bellman-Ford algorithm
  - Every ***t*** seconds each router sends a copy of its routing table to its immediate neighbors
  - Each router then updates its own routing table based on the new information it has received
- Disadvantages:
  - changes in network topology are slow to update since they must be passed one node at a time
  - this does not scale well in large networks because many update messages are being exchanged

Distance Vector Routing



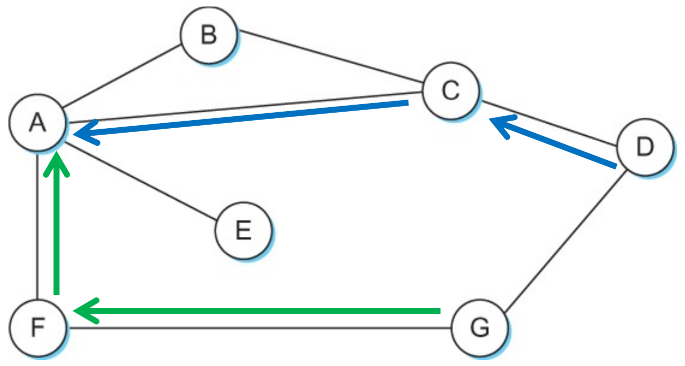
Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

A only "knows" the cost to its direct neighbors

∞ indicates 'not a neighbor'

Initial routing table at node A

Distance Vector Routing



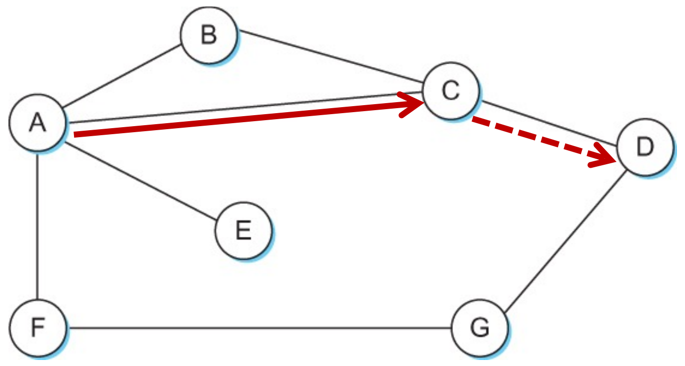
Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

C "tells" A how to get to D

F "tells" A how to get to G

Final routing table at node A

Distance Vector Routing



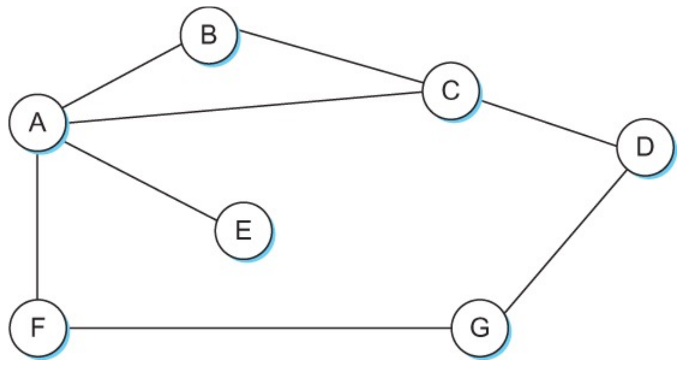
Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

If A wants to send a packet to D, it only "knows" that it must deliver it to C and C will take care of it from there on.

Similarly, to get to G, A must go through F

Final routing table at node A

Distance Vector Routing



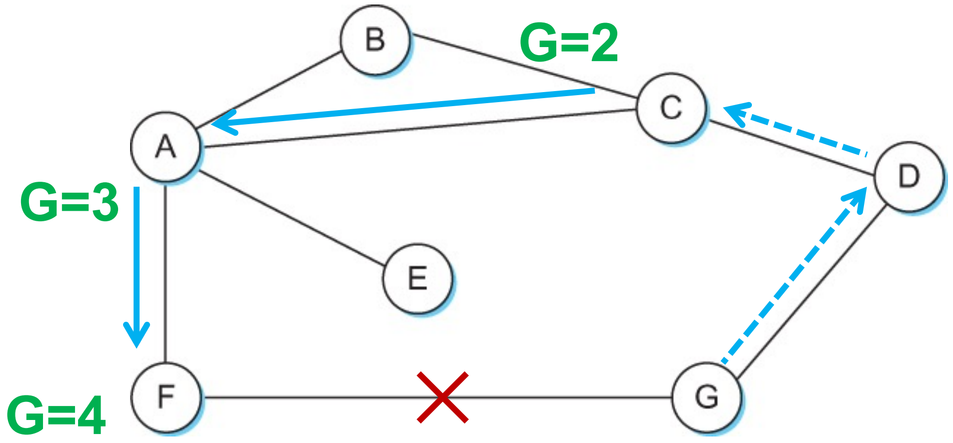
Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Final distances stored at each node

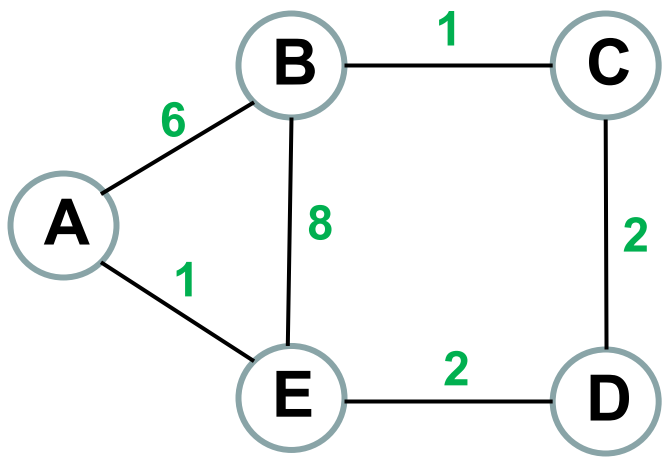
(this is the global view, nodes only store their own local table)

Distance Vector Routing

- When a node detects a link failure
  - **F** detects that link to **G** has failed
  - **F** sets distance to **G** to infinity and sends update to **A**
  - **A** sets distance to **G** to infinity since it uses **F** to reach **G**
  - **A** receives periodic update from **C** with 2-hop path to **G**
  - **A** sets distance to **G** to 3 and sends update to **F**
  - **F** determines that it can reach **G** in 4 hops via **A**



Distance Vector: example



In this example, we will calculate the distances from nodes A and E to the other nodes

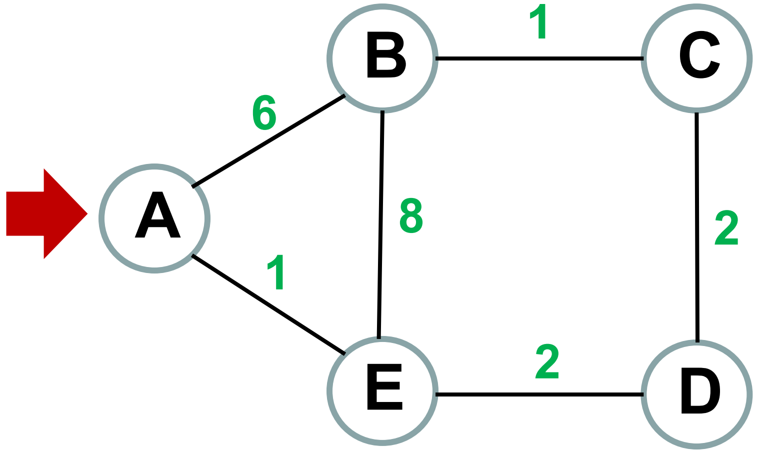
$D^{X}(Y,Z)$ = distance *from* **X** *to* **Y**, *via* **Z** as next hop

$= D(X,Z) + \min_{i} \{D^{Z}(Y,i)\}$

To find the distance from X to Y, via Z

- we start with the distance from X to Z,
- then we find the minimum distance from Z to Y by trying all possible paths from Z to Y

Distance Vector: example



Only neighbors of A

Calculate Distances:
From A to:

Via ⇓	B	C	D	E
B				
E				

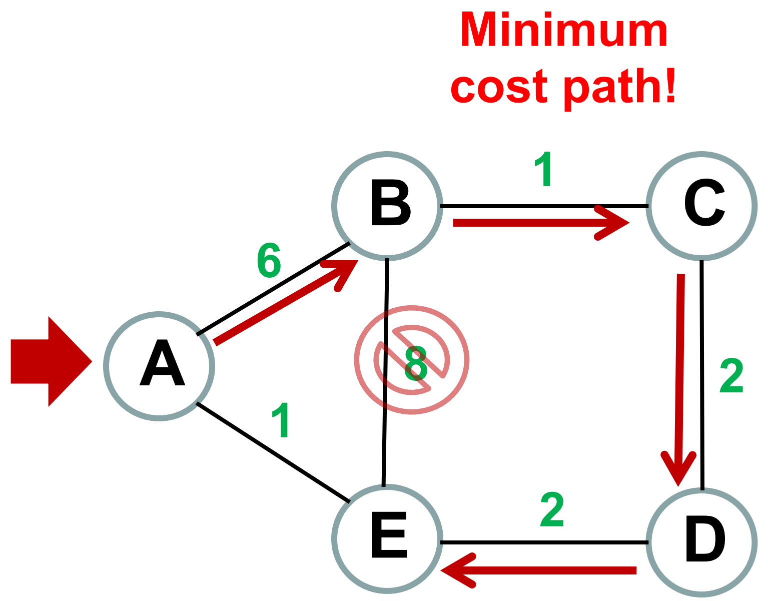
$D^X(Y,Z)$ = distance *from* **X** *to* **Y**, *via* **Z** as next hop

$= D(X,Z) + \min_i \{D^Z(Y,i)\}$

To find the distance from X to Y, via Z

- we start with the distance from X to Z,
- then we find the minimum distance from Z to Y by trying all possible paths from Z to Y

Distance Vector: example



Calculate Distances:
From A to:

Via ⇩	B	C	D	E
B	6	7	9	11
E				

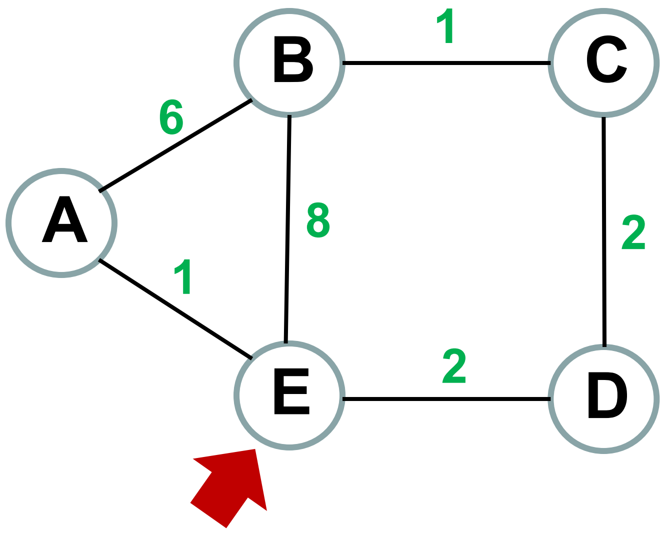
$D^{X}(Y,Z)$ = distance *from* X *to* Y, *via* Z as next hop

$$= D(X,Z) + \min_{i} \{D^{Z}(Y,i)\}$$

To find the distance from X to Y, via Z

- we start with the distance from X to Z,
- then we find the minimum distance from Z to Y by trying all possible paths from Z to Y

Distance Vector: example



$D^{X}(Y,Z)$ = distance *from* **X** *to* **Y**, *via* **Z** as next hop

$= D(X,Z) + \min_i \{D^{Z}(Y,i)\}$

Calculate Distances:

From A to:

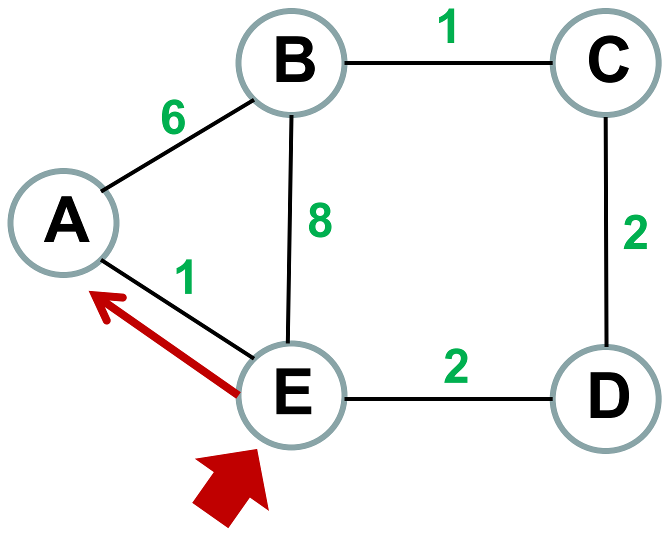
Via ⇩	B	C	D	E
B	6	7	9	11
E	6	5	3	1

From E to:

Via ⇩	A	B	C	D
A				
B				
D				

Only neighbors of E

Distance Vector: example



Calculate Distances:

From A to:

Via ⇩	B	C	D	E
B	6	7	9	11
E	6	5	3	1

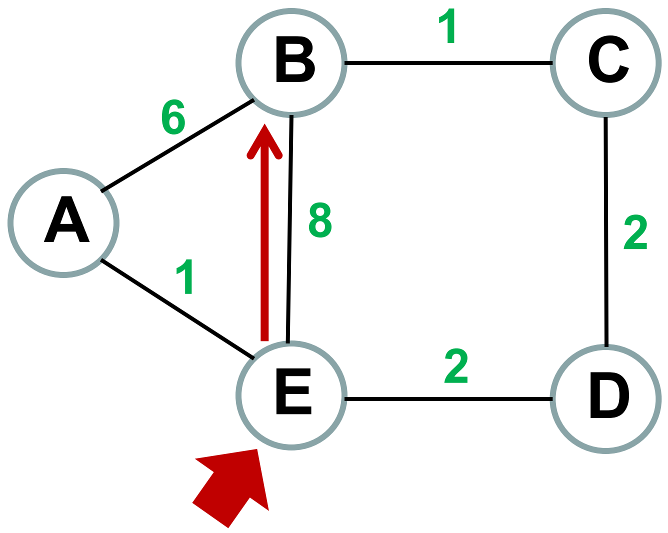
From E to:

Via ⇩	A	B	C	D
A	1	7	6	4
B				
D				

$D^X(Y,Z)$ = distance *from* **X** *to* **Y**, *via* **Z** as next hop

$= D(X,Z) + \min_i \{D^Z(Y,i)\}$

Distance Vector: example



Calculate Distances:

From A to:

Via ⇩	B	C	D	E
B	6	7	9	11
E	6	5	3	1

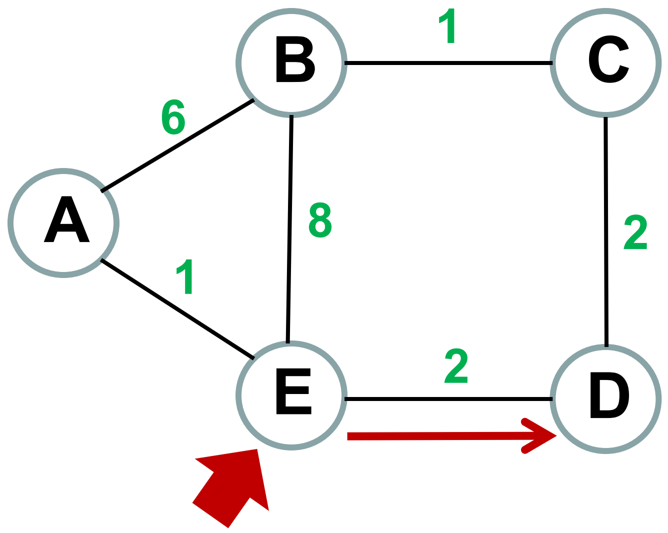
From E to:

Via ⇩	A	B	C	D
A	1	7	6	4
B	14	8	9	11
D				

$D^{X}(Y,Z)$ = distance *from* **X** *to* **Y**, *via* **Z** as next hop

$= D(X,Z) + \min_{i} \{D^{Z}(Y,i)\}$

Distance Vector: example



Calculate Distances:

	From A to:			
Via ⇩	B	C	D	E
B	6	7	9	11
E	6	5	3	1

	From E to:			
Via ⇩	A	B	C	D
A	1	7	6	4
B	14	8	9	11
D	5	5	4	2

$D^X(Y,Z)$ = distance *from* X *to* Y, *via* Z as next hop

$= D(X,Z) + \min_i \{D^Z(Y,i)\}$

Creating the Routing and Forwarding Tables

- After we have the distance table, we can use that information to create the Routing Table and the Forwarding Table
  - the Routing Table shows the lowest cost path to each destination node
  - the Forwarding Table shows the interface (port) to use to reach each destination node

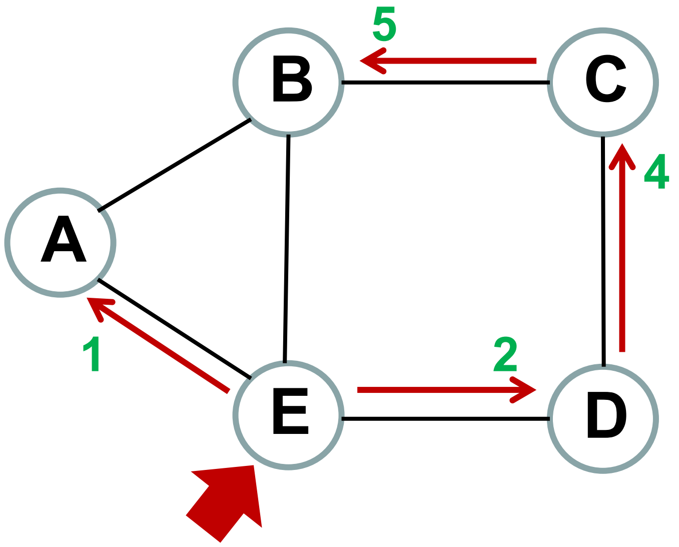
Distance Table to Routing Table:

	From E to:			
Via:	A	B	C	D
A	1	7	6	4
B	14	8	9	11
D	5	5	4	2

Routing Table at Node E:

Destination	Link to:	Cost:
A	A	1
B	D	5
C	D	4
D	D	2

In this example, we will calculate the routing table for node E



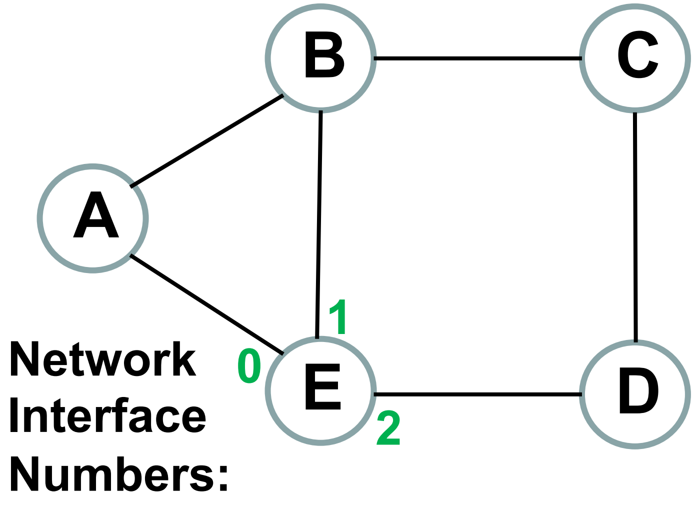
Routing Table to Forwarding Table:

Routing Table at Node E:

Destination	Link to:	Cost:
A	A	1
B	D	5
C	D	4
D	D	2

Forwarding Table at Node E:

Destination	Output:
A	0
B	2
C	2
D	2



In this example, we calculated the forwarding table for node E