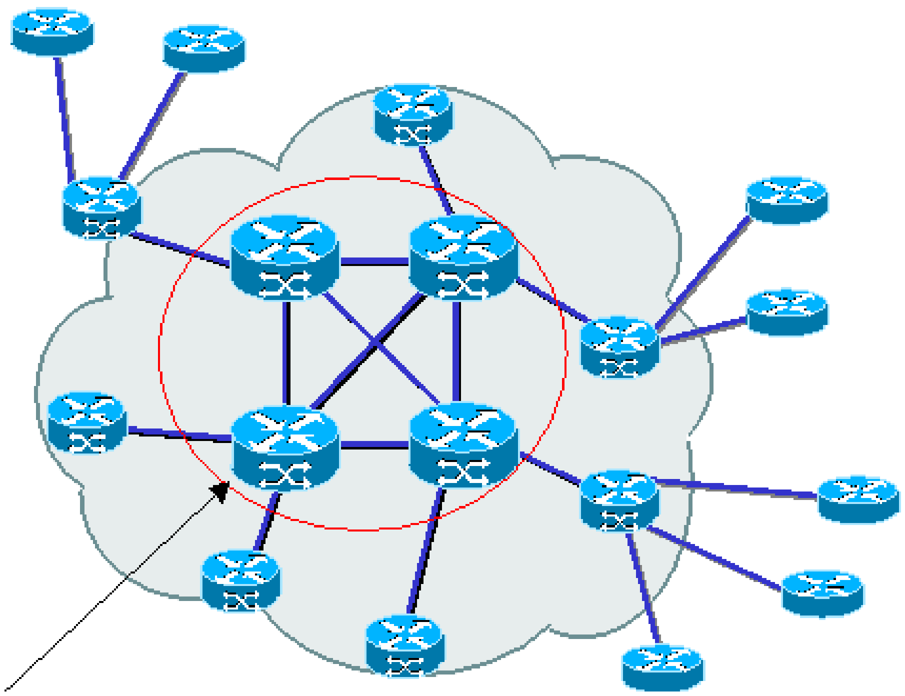
Network Layer (5): Unicast Routing

Required reading:
Kurose 4.2, 5.1, 5.2, 5.3, 5.4

EECS 3214, Fall 2020
Instructor: N. Vlajic

Core-Internet Routers



Core-Internet Routers

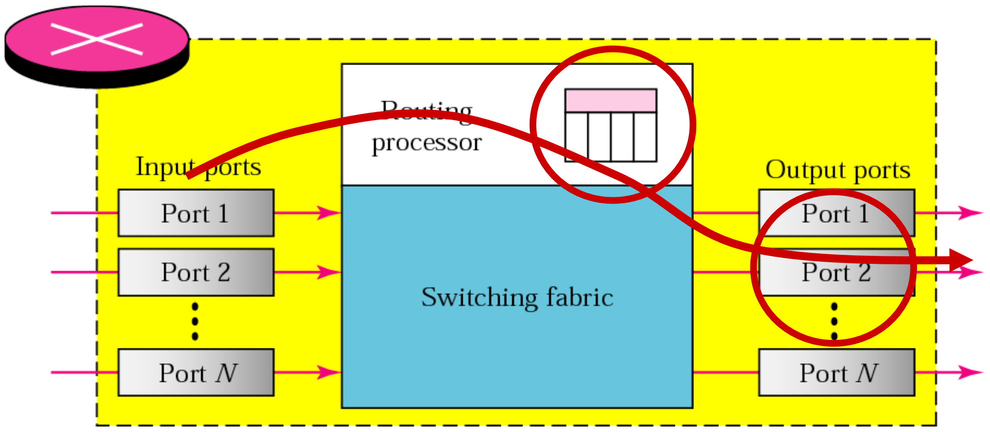
Internet Router Architecture

Router – **3-layer (physical, data-link, network) device, with 3 key functions:**

- **forward/switch IP packets from incoming to proper outgoing links**
- **run routing algorithms/protocols (RIP, OSPF, BGP) to build routing tables**
- **manage congestion**

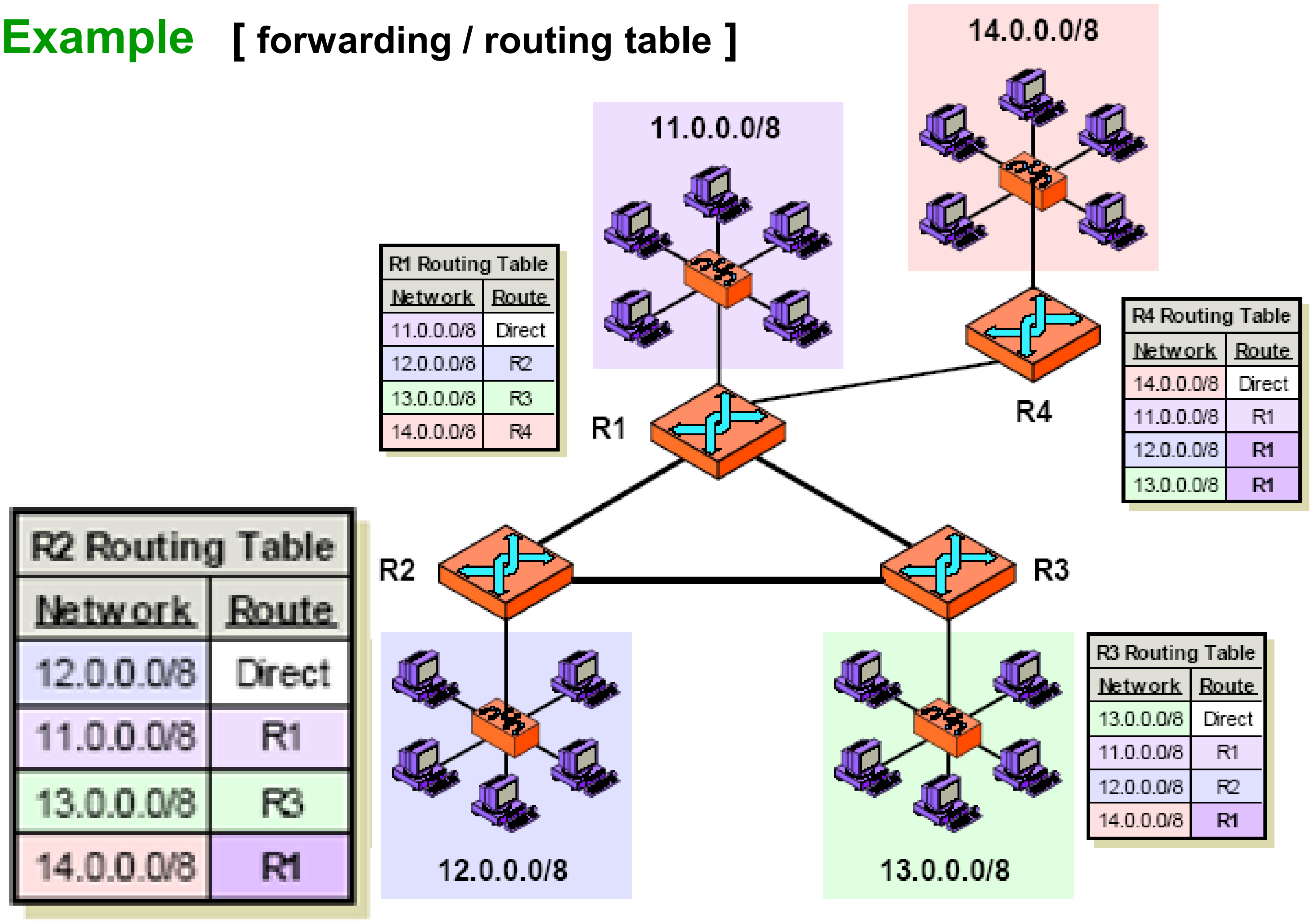
Router Architecture

- **input ports / interfaces** (see pp. 6)
- **switching fabric** (see pp. 7)
- **output ports / interfaces** (see pp. 8)
- **routing processor** – **in charge of**
  1) **executing routing protocol**
  2) **maintaining routing information and <u>forwarding tables</u>, etc.**



Internet Router Architecture (cont.)

Example **[forwarding / routing table]**



R1 Routing Table	
Network	Route
11.0.0.0/8	Direct
12.0.0.0/8	R2
13.0.0.0/8	R3
14.0.0.0/8	R4

R4 Routing Table	
Network	Route
14.0.0.0/8	Direct
11.0.0.0/8	R1
12.0.0.0/8	R1
13.0.0.0/8	R1

R2 Routing Table	
Network	Route
12.0.0.0/8	Direct
11.0.0.0/8	R1
13.0.0.0/8	R3
14.0.0.0/8	R1

R3 Routing Table	
Network	Route
13.0.0.0/8	Direct
11.0.0.0/8	R1
12.0.0.0/8	R2
14.0.0.0/8	R1

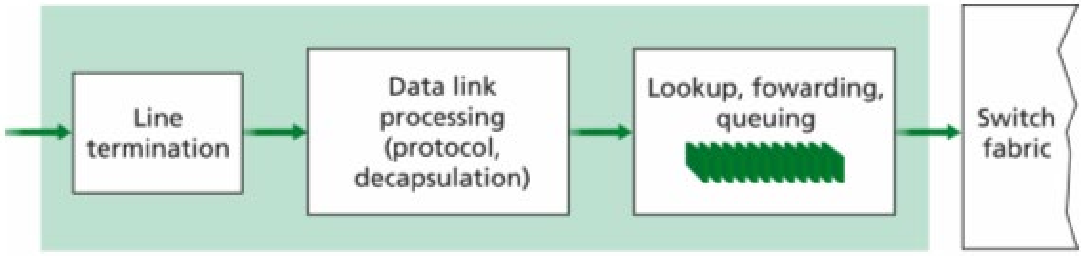
Figure 93: IP Routing and Routing Tables
This diagram shows a small, simple internetwork consisting of four LANs each served by a router. The routing table for each lists the router to which datagrams for each destination network should be sent, and is color coded to match the colors of the networks. Notice that due to the "triangle", each of R1, R2 and R3 can send to each other. However, R2 and R3 must send through R1 to deliver to R4, and R4 must use R1 to reach either of the others.

Internet Router Architecture (cont.)

Input Port – has an associated line card (NIC) which implements physical and data-link layer functions, as well as certain network layer functions

Input Line Card Functions

- physical layer: bit-level reception
- data-link layer: decapsulation, error checking, etc.
- network layer: decentralized switching / packet forwarding = decide to which output line to forward each packet based on packet header
  - forwarding table able is created and updated by routing processor, but a shadow copy of the table is typically stored at each input port and updated

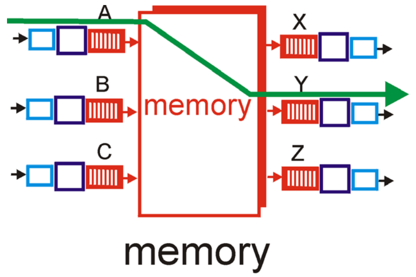

Line termination
Data link processing (protocol, decapsulation)
Lookup, fowarding, queuing
Switch fabric

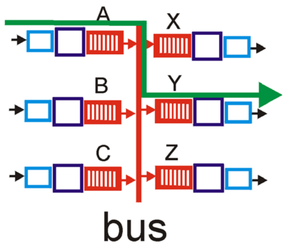
Decentralized switching prevents creating a processing bottleneck at a single point within the router.

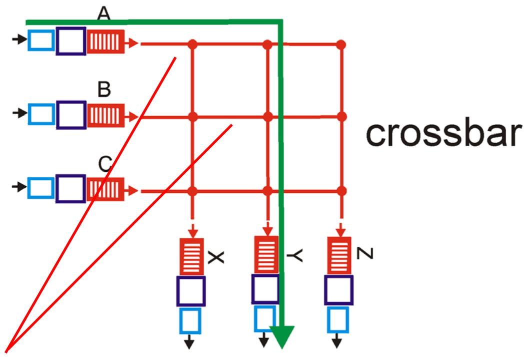
Internet Router Architecture (cont.)

Switching Fabric Function – **(physically) transfer packets between input and output line cards**

Types of Switching Fabric



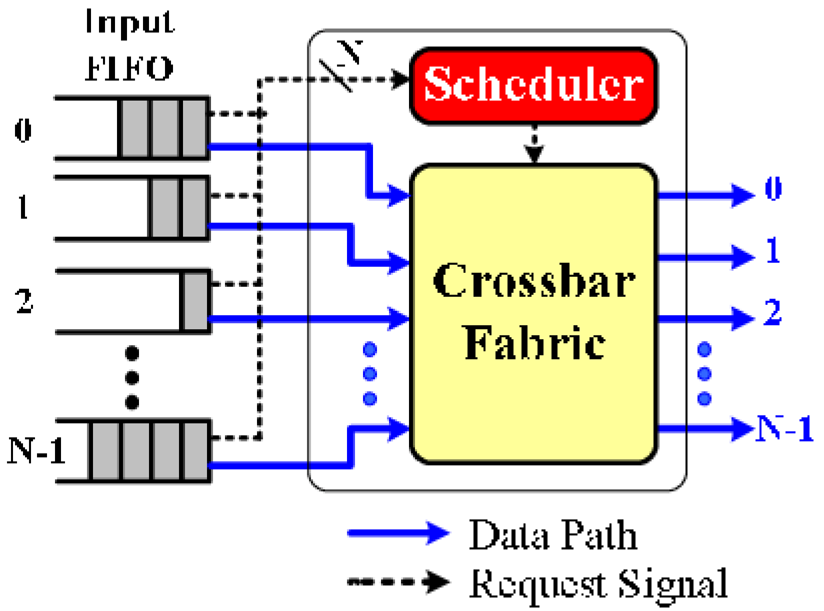


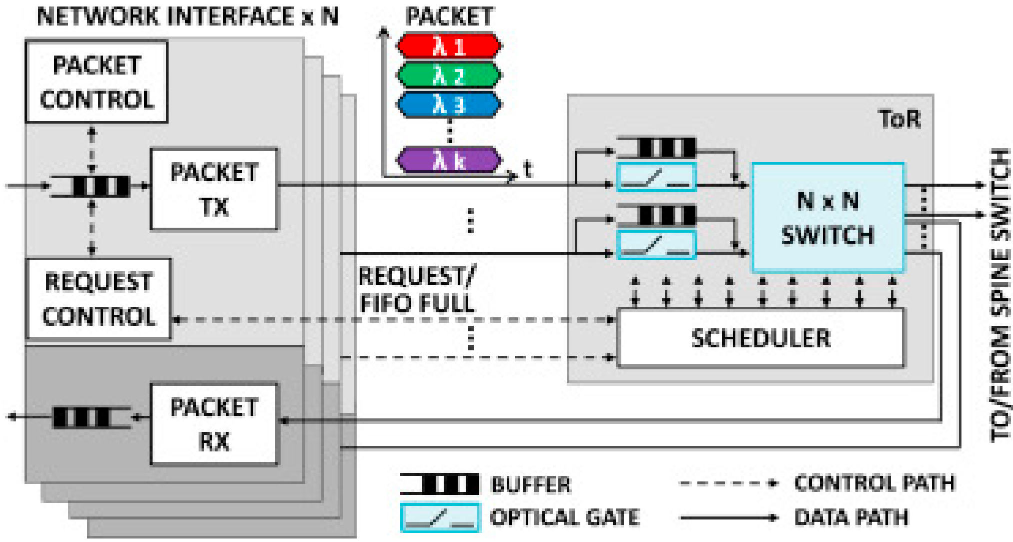


Multiple packets can be sent in parallel. E.g., a packet can be sent from A to Y at the same time as a packet from B to X

- **via memory: datagram is received through input port, stored in memory, then send to appropriate output port – slow ☹**
- **via a bus: datagram is sent directly from input port to output port via a <u>shared bus</u> ⇒ does not scale well ☹**

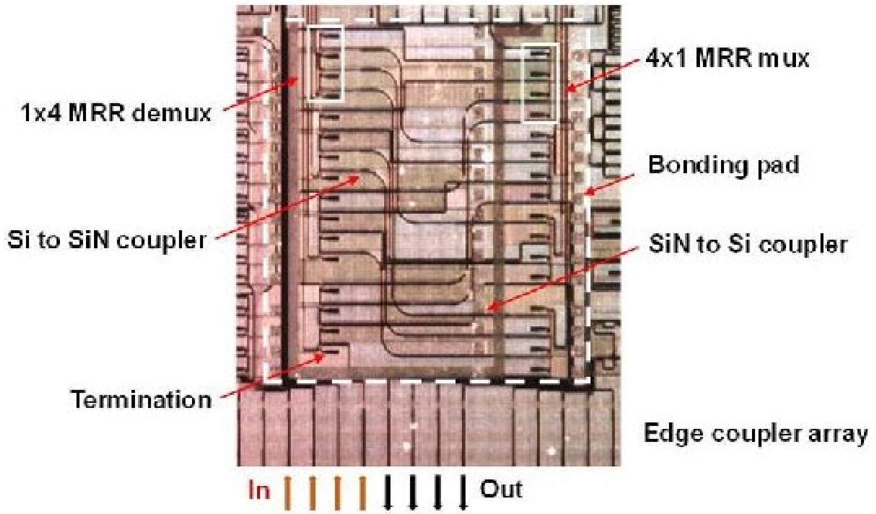
  (packets are send serially so buss speed needs to be N-times input line speed)
- **via a crossbar: interconnection network consisting of 2N busses that interconnect N input and N output**
  - **packet travels along horizontal bus until it intersects with vertical bus leading to desired output port – if vertical bus is busy, queueing at input port is needed**
  - **Cisco Catalyst 6500: 720 Gbps routers**





https://www.sciencedirect.com/science/article/pii/S1573427718301577

https://www.semanticscholar.org/paper/Fast-Fair-Crossbar-Scheduler-for-On-chip-Router-Yang-Sheu/3a59ad5be56468e9bebedfb3ff1d0e371830e7ac/figure/0

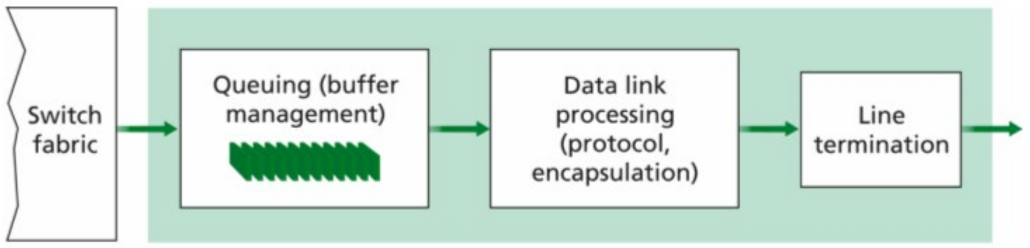


https://www.researchgate.net/publication/328280143_SiSiN_Microring-Based_Optical_Router_in_Switch-and-Select_Topology/figures?lo=1

Internet Router Architecture (cont.)

Output Line Card Functions

- **network layer:**
  1) **buffering** – required when datagrams arrive from fabric at rate faster than output line transmission rate
  2) **buffer management** – decide when and which packets to drop if there is not enough memory to store all incoming packets
  3) **scheduling / packet classification** – decide which packet, of those queued, to send out next/first
     - packet scheduling plays crucial role in providing quality-of-service (QoS)
- **data-link layer:** encapsulation, address mapping, etc.
- **physical layer:** bit-level forwarding


Switch fabric
Queuing (buffer management)
Data link processing (protocol, encapsulation)
Line termination

Routing

Routing in the Internet – combination of rules and procedures that allow routers to

routing protocol

- inform one another of 'status of' or 'changes in the network (gather/exchange information)

routing algorithm

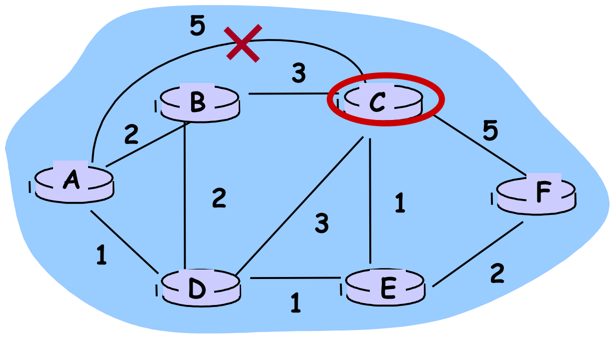
- calculate '**best**' routing paths in the network

packet forwarding

- transfer packets from a source host to a destination host along the '**best path**'

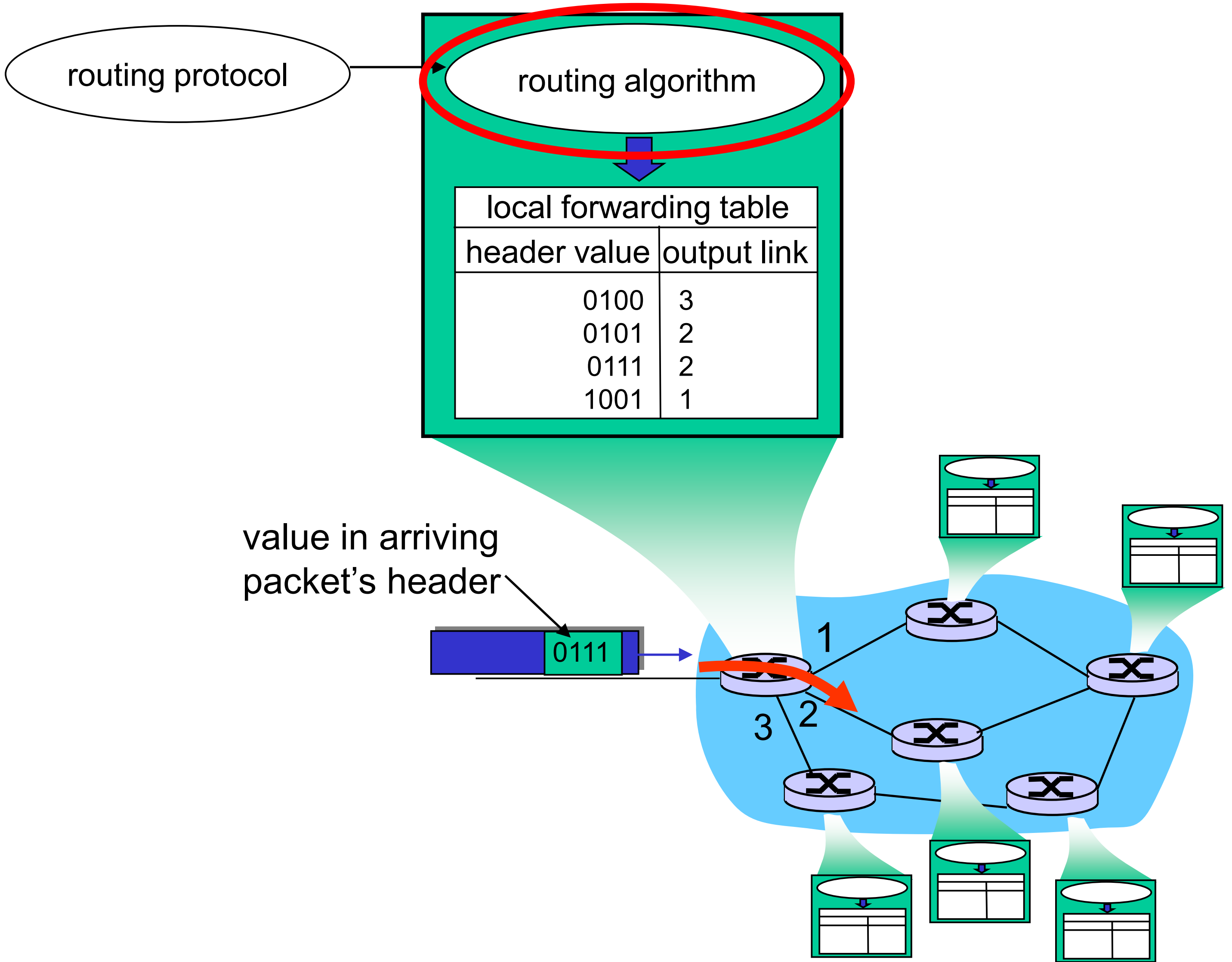
Additional Routing Goals: accurate, rapid, low-cost delivery of packets that should also …

- route packets away from failed and temporarily congested nodes or links
- avoid routing loops
- adapt to varying traffic loads
  → adjust paths based on the current traffic loads
- low overhead
  → minimize the overhead caused by control messages



Routing (cont.)

Example [interplay between routing protocol, algorithm and forwarding]



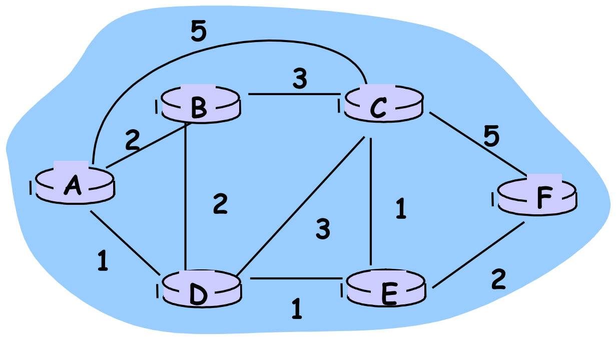
local forwarding table

header value	output link
0100	3
0101	2
0111	2
1001	1

Routing Algorithms

Routing Algorithm – heart of routing protocol - determines the best routing path between any two hosts in the network

- best path = path that minimizes the objective function that the network operator tries to optimize
- possible objective functions:
  - (1) number of hops
  - (2) end to end delay
  - (3) ISP cost
  - …



Classification of Routing Algorithms/Protocols

(1) Static vs. Dynamic – frequency of routing-info update?
(2) Centralized vs. Global vs. Distributed – 'best path' calculation?
(3) Source-based vs. Hop-by-Hop – place of routing decision?
(4) Interior AS vs. Exterior AS – routing area?

Routing Algorithms (cont.)

Static Routing

- **in static routing:**
  - **paths are recomputed offline and manually entered**
  - **paths are kept fixed for a relatively long period of time**
- **static routing may suffice if:**
  - **network size is small**
  - **network topology is relatively fixed / stable**
  - **traffic load does not change appreciably**

Dynamic Routing

- **in dynamic (adaptive) routing:**
  - **routers continuously learns the state of the network by communicating with each other**
  - **based on the information collected, routers compute the best paths to desired destinations**
- **dynamic routing may suffice if:**
  - **network loads or topology frequently change**

While dynamic algorithms are more responsive to network changes, they add complexity to router design and operation !!!

Routing Algorithms (cont.)

Centralized Routing – a network control centre computes all paths and uploads this information to the nodes in the network

- central node – single point of failure

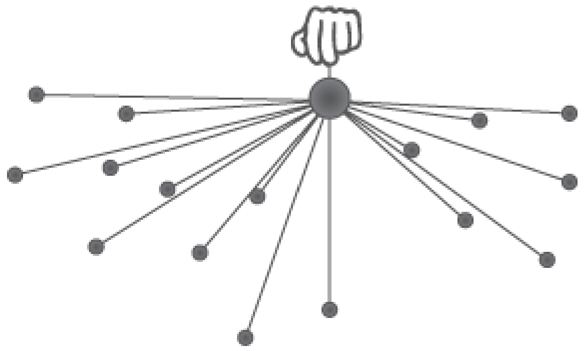
Global Routing – each node computes the least-cost paths between every possible source and destination using complete, global knowledge about the network

- also known as **link state algorithms**
- algorithm example: **Dijkstra**
- respective protocol: **OSPF**

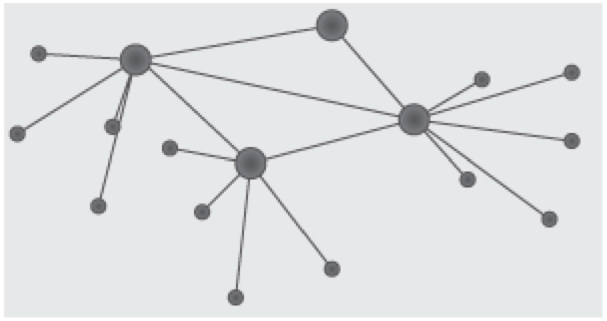
Distributed Routing – in an iterative process of calculation and exchange of information with its neighbours, a node gradually calculates the least-cost paths

- no node has complete information about the cost of all network links
- also known as **distance-vector algorithms**
- algorithm example: **Distributed Bellman-Ford**
- example of respective protocol: **RIP**

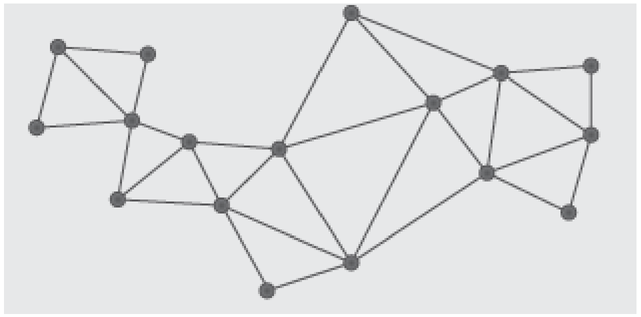
Routing Algorithms (cont.)



Centralized
Knowledge &
Computing



Global Knowledge
(aka Link-State)
e.g. Dijkstra
protocol: OSPF



Distributed Computing
(aka Distance-Vector)
e.g. Bellman-Ford
protocol: RIP

Link State Routing: Dijkstra

Dijkstra Algorithm – solves single-source shortest path problem – finds the shortest path from a given source node to all other nodes in the network (result: TREE)

- all link costs must be positive and known to all nodes
- the algorithm is iterative – by kth iteration the shortest paths to k nodes closest to the source node have been determined

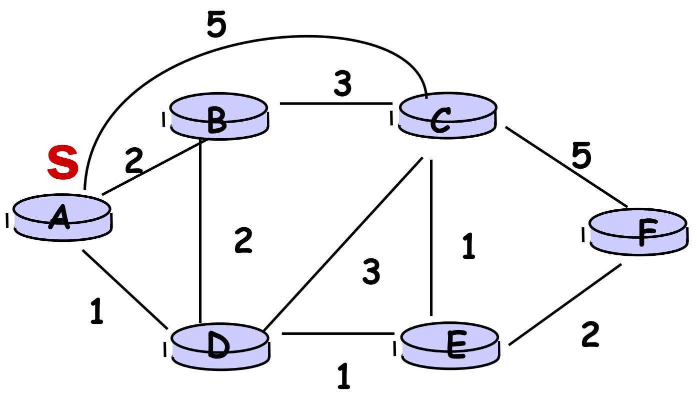
N = set of nodes in the network

w(i, j) = link cost from node i to node j

w(i, i) = 0

w(i, j) = ∞ if nodes not directly connected

w(i, j) ≥ 0 if nodes directly connected



S = source node

T = set of nodes so far incorporated

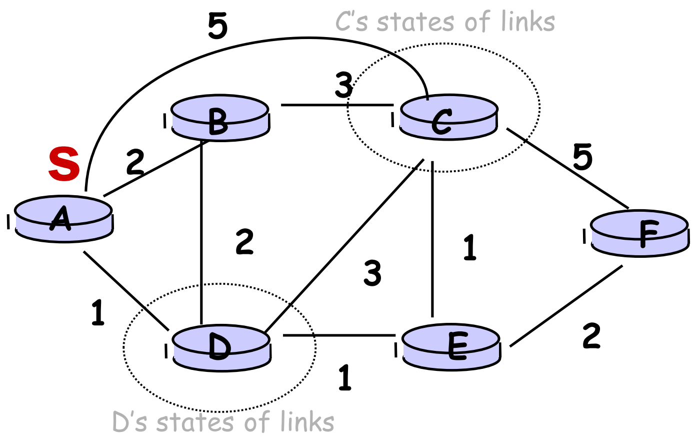
L(n) = cost of least-cost path from node S to node n that is currently known

at termination, cost of least-cost path in graph from s to n

Link State Routing: Dijkstra (cont.)

1. [Initialization]

T = { S } – set of nodes so far incorporated consists of only the source node
L(n) = w(S,n) – initial path costs to neighbouring nodes are simply the link costs



w(A, A) = 0 = L(A, A)

w(A, B) = 2 = L(A, B)

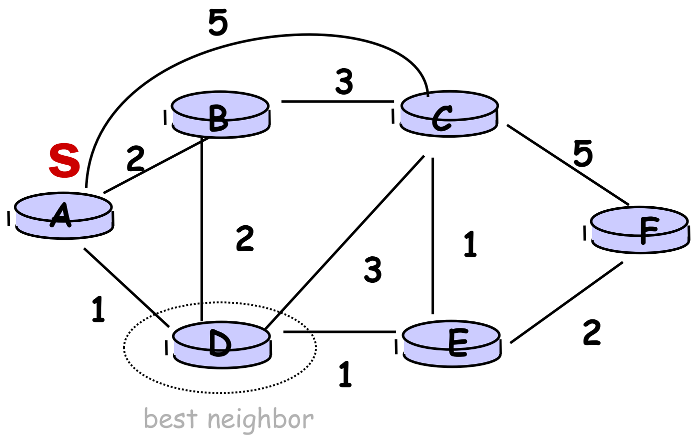
w(A, C) = 5 = L(A, C)

w(A, D) = 1 = L(A, D)

w(A, E) = ∞ = L(A, E)

w(A, F) = ∞ = L(A, F)

T = { A }



2. [Get Next Node]

Find the neighbouring node x currently NOT in T that has the least-cost path from node S and incorporate that node into T.

$$\text{find } x \notin T \text{ such that } L(x) = \min_{j \notin T} L(j)$$

3. [Update Least-Cost Paths for all Nodes n Adjacent to x, not in T]

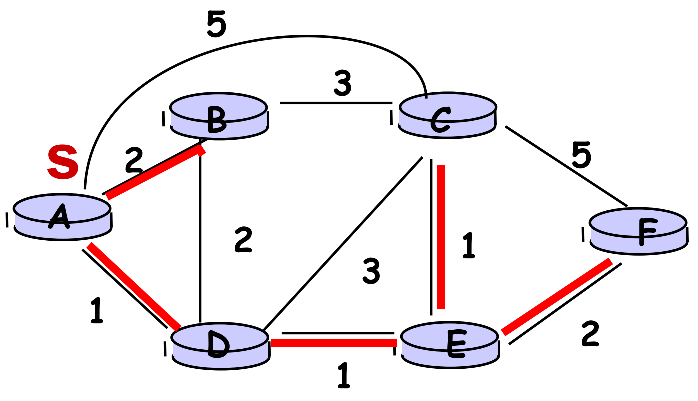
$$L(n) = \min\left[L(n), L(x) + w(x,n)\right], \text{ for all } n \notin T$$

If the latter term is the minimum, the path from S to n is now the path from S to x concatenated with the edge from x to n.

Algorithm terminates when all nodes have been added to T.

Link State Routing: Dijkstra (cont.)

Example **[Dijkstra Algorithm on Undirected Graph]**



Iteration	T	L(B)	path	L(C)	path	L(D)	path	L(E)	path	L(F)	path
1	{A}	2	A-B	5	A-C	1	A-D	∞	-	∞	-
2	{A,D}	2	A-B	4	A-D-C	1	A-D	2	A-D-E	∞	-
3	{A,D,B}	2	A-B	4	A-D-C	1	A-D	2	A-D-E	∞	-
4	{A,D,B,E}	2	A-B	3	A-D-E-C	1	A-D	2	A-D-E	4	A-D-E-F
5	{A,D,B,E,C}	2	A-B	3	A-D-E-C	1	A-D	2	A-D-E	4	A-D-E-F
6	{A,D,B,E,C,F}	2	A-B	3	A-D-E-C	1	A-D	2	A-D-E	4	A-D-E-F

Link State Routing: Dijkstra (cont.)

Assumption: All link costs in the network must be known to all nodes!

```
Initialization
  T = {S}
  for all nodes n
    if n adjacent to S
      then L(n) = w(S,n)
      else L(n) = ∞

Loop
  find x not in T such that L(x) is a minimum
  add x to T
  update L(n) for all n adjacent to x and not in T:
     L(n) = min( L(n), L(x) + w(x,n) )
until all nodes in T
```

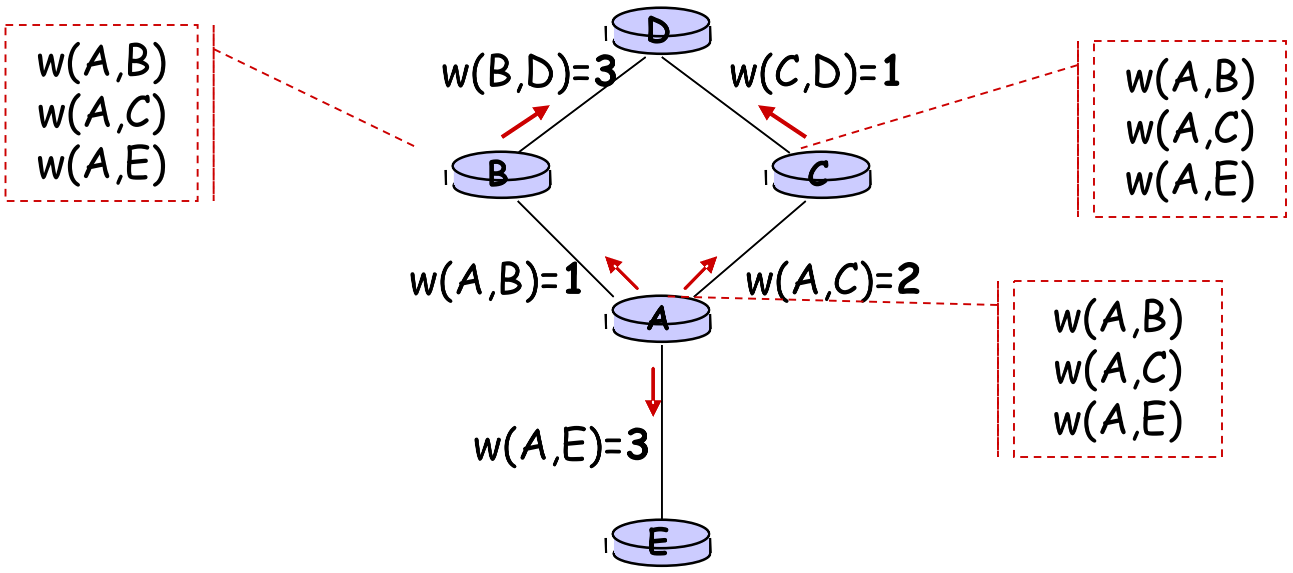
Dijkstra Algorithm RT Complexity – assume n nodes / routers in the network

- at kth iteration all remaining (n-k) nodes not currently in T should be checked
- overall $n(n-1)/2$ comparison $\Rightarrow$ $\boxed{RT = O(n^2)}$ per source node
- more efficient implementation of RT = O(nlog(n)) on sparse graphs is possible [using heaps]

Link State Routing: Dijkstra (cont.)

Example **[Dijkstra Algorithm on Undirected Graph]**

Phase 0: flooding from A

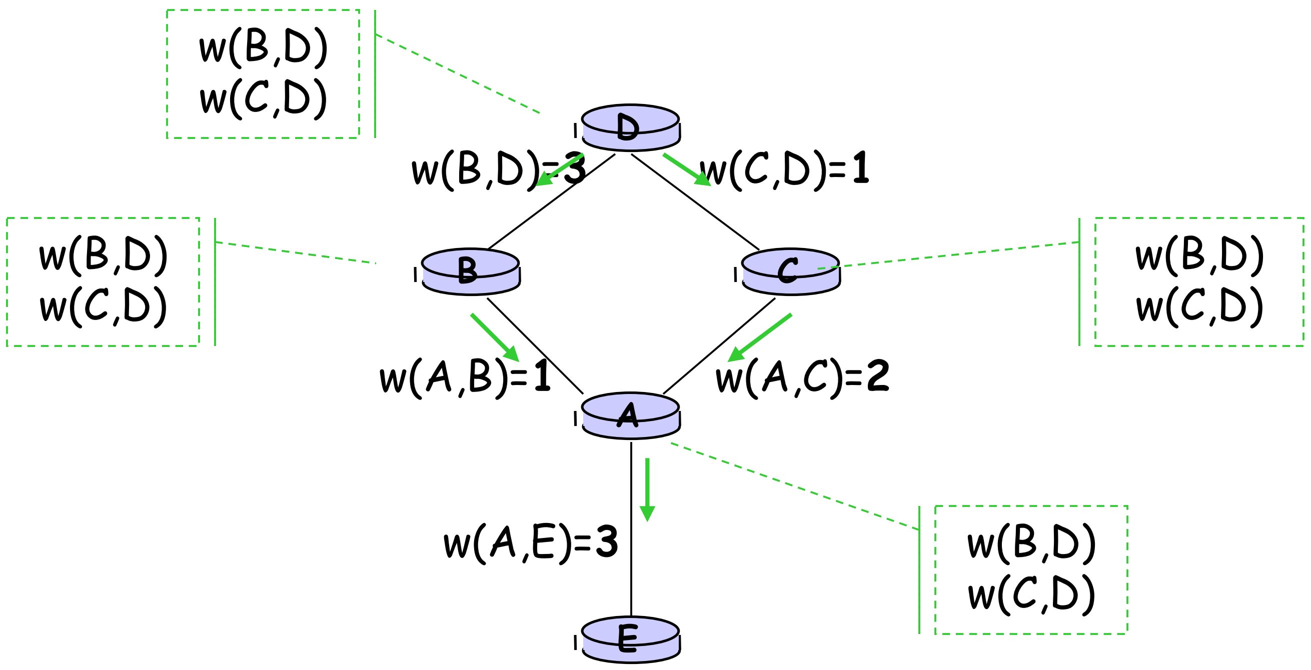


The cost of links adjacent to A must be sent to everybody.

Link State Routing: Dijkstra (cont.)

Example **[Dijkstra Algorithm on Undirected Graph cont.]**

Phase 0: flooding from D

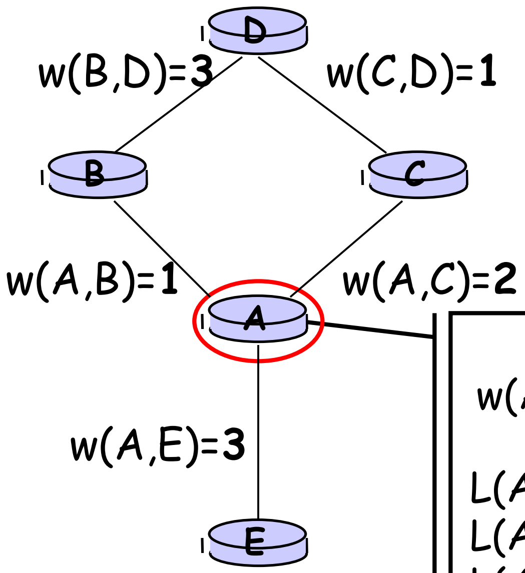


The same must be repeated for B, C, E …

Link State Routing: Dijkstra (cont.)

Example **[Dijkstra Algorithm on Undirected Graph cont.]**

Phase 1: Initialization



A knows all w-s:
w(A,B), w(A,C), w(A,E), w(B,D), w(C,D)

L(A,B)=w(A,B)=1
L(A,C)=w(A,C)=2
L(A,E)=w(A,E)=3
L(A,D)=∞

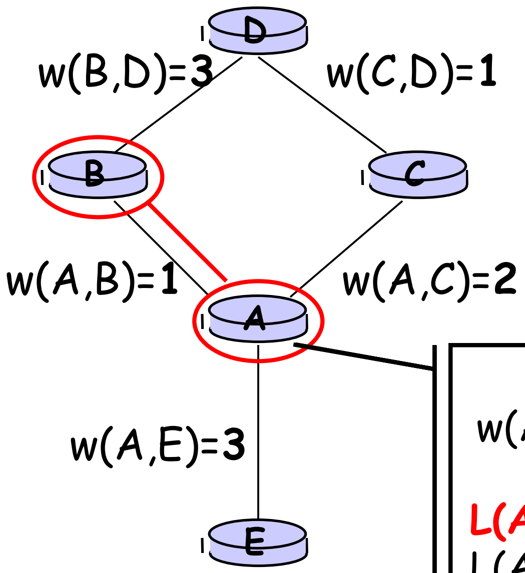
T={A}

Information stored at A. Initialize L-s and T.

Link State Routing: Dijkstra (cont.)

Example **[Dijkstra Algorithm on Undirected Graph cont.]**

Phase 2 – 1st iteration: Get Next Node



Find node, not in T, with min L to A.

A knows all w-s:
w(A,B), w(A,C), w(A,E), w(B,D), w(C,D)

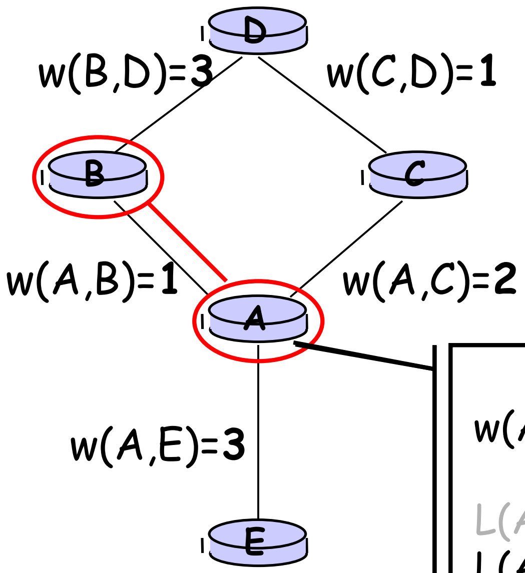
L(A,B)=w(A,B)=1
L(A,C)=w(A,C)=2
L(A,E)=w(A,E)=3
L(A,D)=∞

T={A,**B**}

Link State Routing: Dijkstra (cont.)

Example **[Dijkstra Algorithm on Undirected Graph cont.]**

Phase 2 – 1st iteration: Update Least-Cost Paths



Update L-s of all adjacent nodes.

A knows all w-s:
w(A,B), w(A,C), w(A,E), **w(B,D)**, w(C,D)

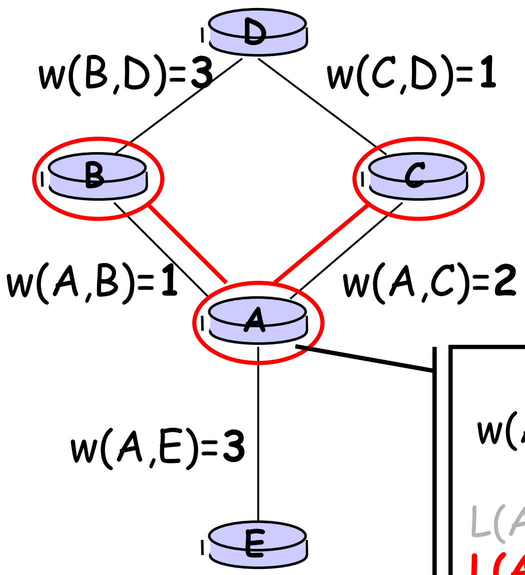
L(A,B)=w(A,B)=1
L(A,C)=w(A,C)=2
L(A,E)=w(A,E)=3
L(A,D)=4

T={A,B}

Link State Routing: Dijkstra (cont.)

Example **[Dijkstra Algorithm on Undirected Graph cont.]**

Phase 2 – 2nd iteration: Get Next Node



Find node, not in T, with min L to A.

A knows all w-s:
w(A,B), w(A,C), w(A,E), w(B,D), w(C,D)

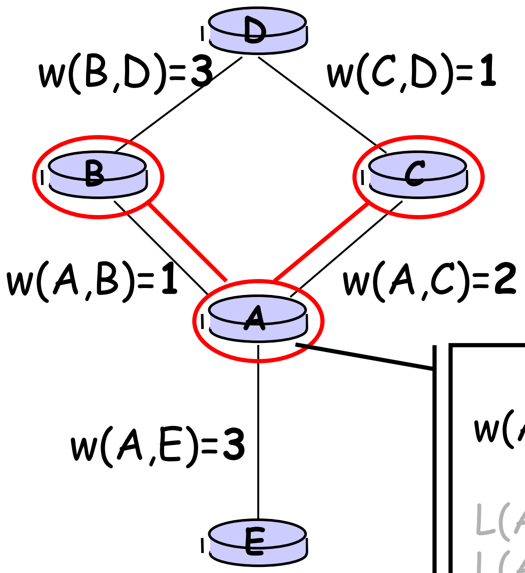
L(A,B)=w(A,B)=1
L(A,C)=w(A,C)=2
L(A,E)=w(A,E)=3
L(A,D)=4

T={A,B,C}

Link State Routing: Dijkstra (cont.)

Example **[Dijkstra Algorithm on Undirected Graph cont.]**

Phase 2 – 2nd iteration: Update Least-Cost Paths



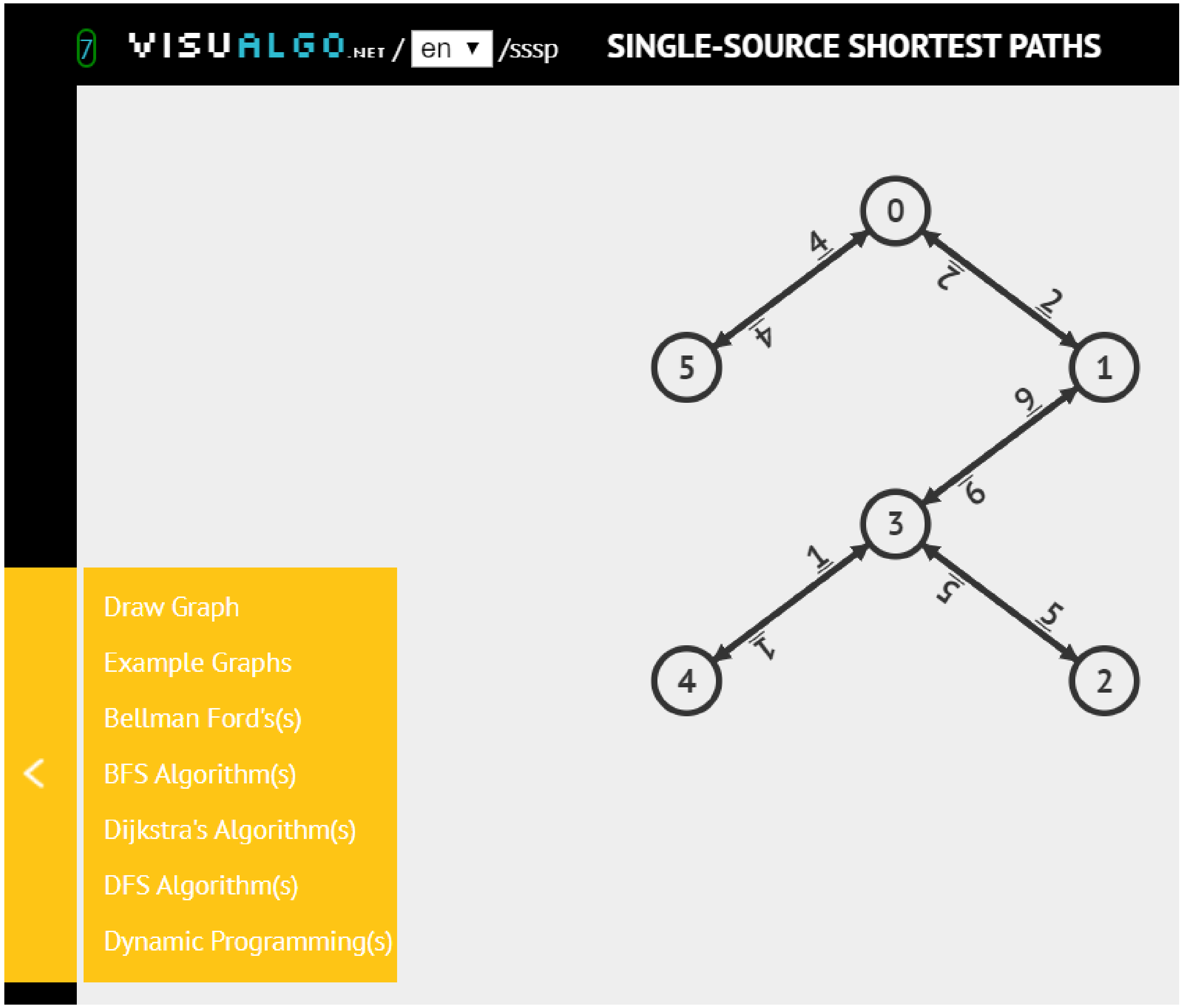
Update L-s of all adjacent nodes.

A knows all w-s:
w(A,B), w(A,C), w(A,E), w(B,D), w(C,D)

L(A,B)=w(A,B)=1
L(A,C)=w(A,C)=2
L(A,E)=w(A,E)=3
L(A,D)=3

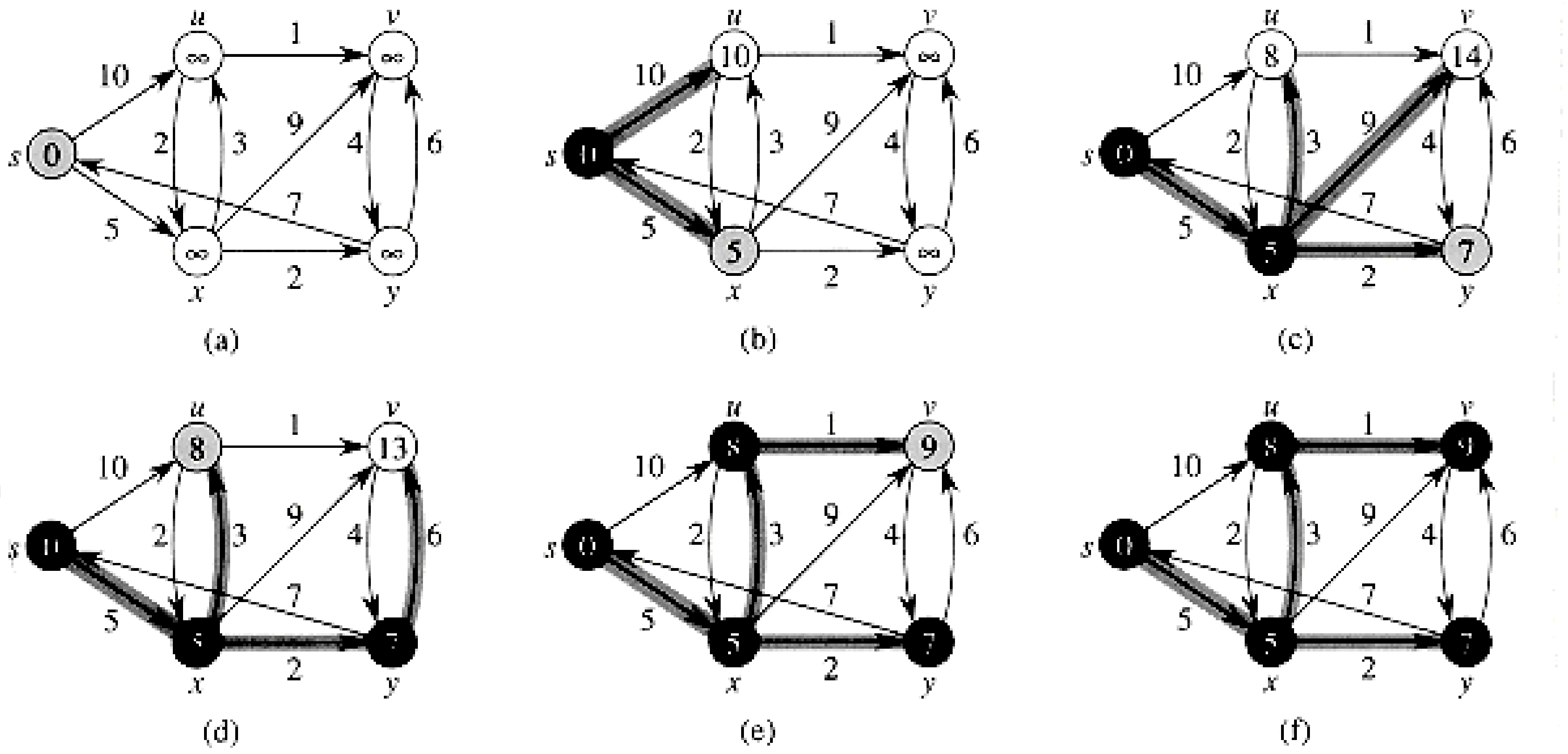
T={A,B,C}

Etc.



https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

Link State Routing: Dijkstra (cont.)



https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

Distance-Vector Routing: Distributed B-F

Distributed Bellman-Ford Algorithm – **iterative, asynchronous, distributed algorithm based on the following principle:**

If each neighbour of node Z knows the shortest path to node C, then node Z can determine its shortest path to node C by calculating the cost/distance to node C through each of its neighbours and picking the minimum.

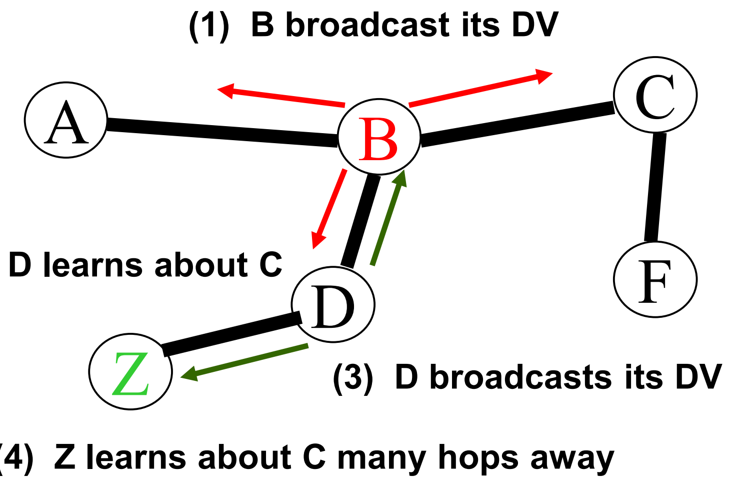
wait for (**change in local link cost** or **DV** from neighbor)

↓

recompute distance table

↓

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



Distance-Vector Routing: Distributed B-F (cont.)

<u>Each node</u> maintains three vectors:

$$W_x = \begin{bmatrix} w(x,1) \\ w(x,2) \\ \ldots \\ w(x,M) \end{bmatrix} \quad L_x = \begin{bmatrix} L(x,1) \\ L(x,2) \\ \ldots \\ L(x,N) \end{bmatrix} \quad R_x = \begin{bmatrix} R(x,1) \\ R(x,2) \\ \ldots \\ R(x,N) \end{bmatrix}$$

portion of routing tables advertised to neighbors!!!

Destination Network/Host	Metric L(X,j)	Next Router R(X,j)
A	1	B
C	2	B
D	2	B
E	3	D
F	3	C

Routing Table

M – number of nodes to which node x is directly attached
N – overall number of nodes in the network

Node does NOT know entire path to another node – only the next hop!

W_x = vector of outgoing link costs for node x

$w(x, j)$ **= cost of link from node x to <u>neighboring</u> node j (j=1,..,M)**

L_x = distance vector for node x

$L(x,i)$ **= current estimate of minimum cost from node x to network/node i (i=1,..,N)**

R_x = next-hop vector for node x

$R(x,i)$ **= next router in current minimum-cost route from node x to network/node i (i=1,..,N)**

Distance-Vector Routing: Distributed B-F (cont.)

1. [**Initialization**]

Each vector starts with a vector of distances to all directly attached networks.

2. [**Send Step**]

Periodically (every 30 sec) each router advertises its distance vector (destination + metric) to all neighbouring routers.

3. [**Receive Step**]

Upon receiving distance vectors from each of its neighbours, or detecting a change in one of its adjacent link-cost, router (re)computes its distance vector. For every host Z, router finds its neighbour Y who is closer to host Z than any other neighbour. Router updates its cost to Z and informs neighbours of any possible change.

$$\textit{find} \quad L(X,Z) = \min_{y \in A} \left[w(X,Y) + L(Y,Z) \right]$$

$$\textit{adjust} \ \ R(X,Z) \ \ \textit{accordingly}$$

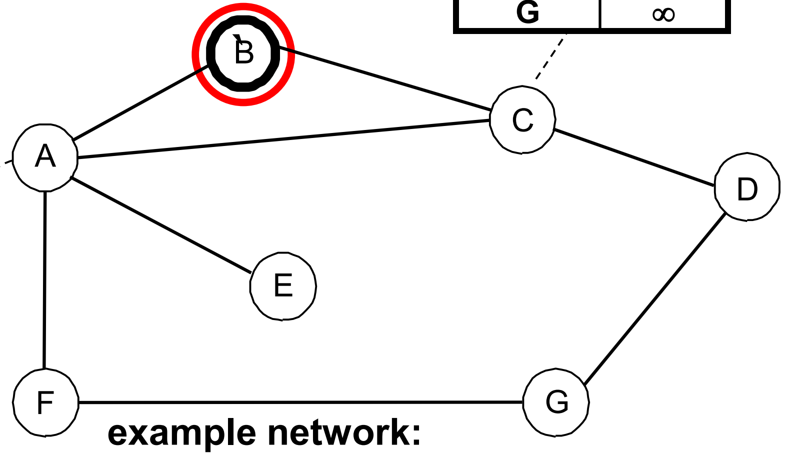
where A = set of neighbour nodes for node x.

After Step 3, router goes to send step (2.) again.

Distance-Vector Routing: Distributed B-F (cont.)

Example **[Distributed Bellman-Ford Algorithm – all links have w=1]**

Dest.	Cost
A	1
B	1
D	1
E	∞
F	∞
G	∞



example network: cost of each link = 1

Dest.	Cost
B	1
C	1
D	∞
E	1
F	1
G	∞

portion of routing tables advertised to neighbors!!!

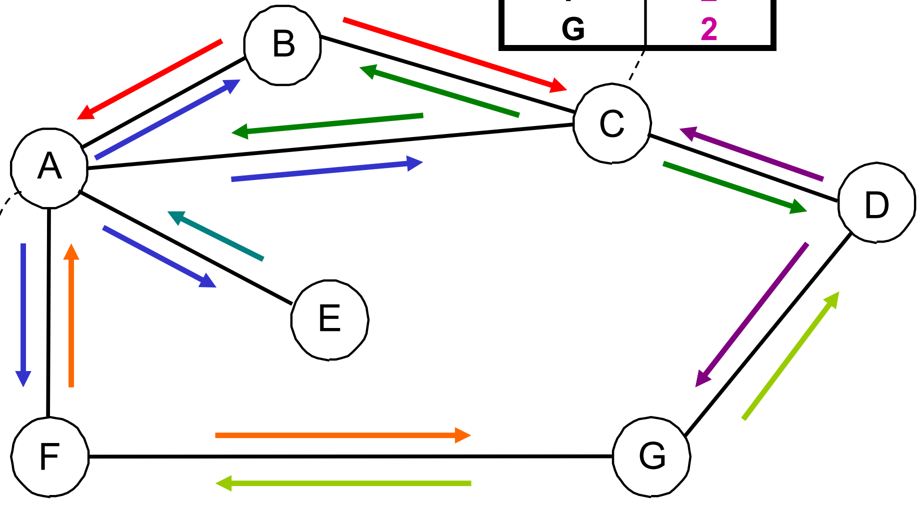
Destination	Cost	NextHop
A	1	A
C	1	C
D	∞	–
E	∞	–
F	∞	–
G	∞	–

initial routing table for B

Distance-Vector Routing: Distributed B-F (cont.)

B exchanges updates with its direct neighbours A and C – after 1st update!

Dest.	Cost
A	1
B	1
D	1
E	2
F	2
G	2



Dest.	Cost
B	1
C	1
D	2
E	1
F	1
G	2

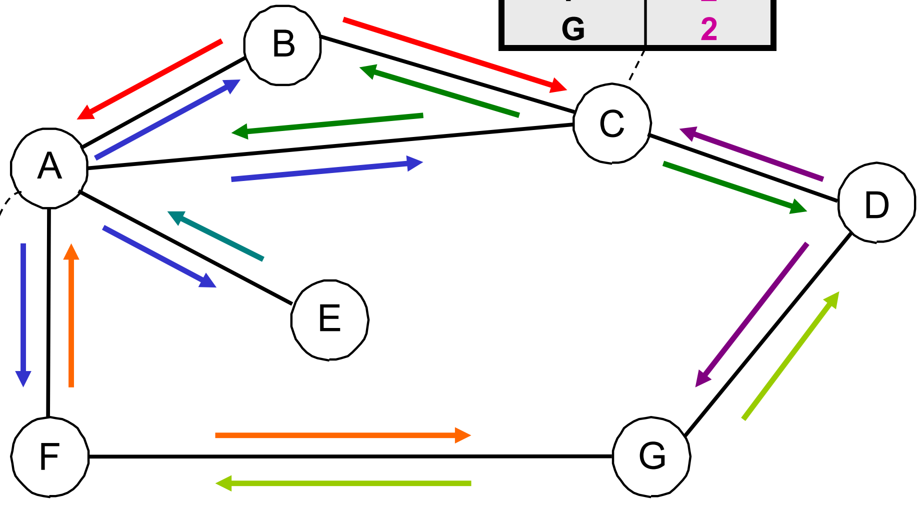
Destination	Cost	Next Hop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	∞	–

routing table for B

Distance-Vector Routing: Distributed B-F (cont.)

B exchanges updates with its direct neighbours A and C – after 2nd update!

Dest.	Cost
A	1
B	1
D	1
E	2
F	2
G	2



Destination	Cost	Next Hop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	**3**	**A**

routing table for B

Dest.	Cost
B	1
C	1
D	2
E	1
F	1
G	2

In the absence of topology changes, it only takes 2 updates to build complete routing tables for all nodes (this is known as *convergence*).

Distance-Vector Routing: Distributed B-F (cont.)

Distributed BF Algorithm RT Complexity

Destination Host	Next Router R(X,j)	Metric R(X,j)
B	B	1
C	B	2
D	B	2
E	B	3
F	B	3

- on each iteration router receives O(K) Distance Vectors (L(x,z), z=1,..,K), where K is the number of neighbours
- for each of n rows in its own DV, router finds the minimum of w(x,y) + L(y,z) over K neighbours (n – overall number of routers in the network)
- on each iteration router expands its network neighbourhood by 1 – the algorithm stabilizes when each router has expanded its horizon to the diameter of the network D

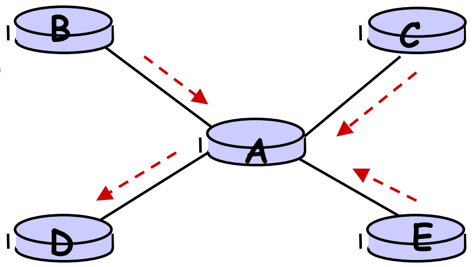
Number of iterations.

Number of iterations.

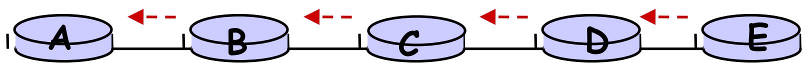
$RT = O(n * K * \boxed{D})$ max # of edges on the shortest path

$RT_{worst\text{-}case} = O(n * m)$, when K=2
m – overall edges

A initiates update of its DV only upon receiving DVs from all of its neighbours



best case

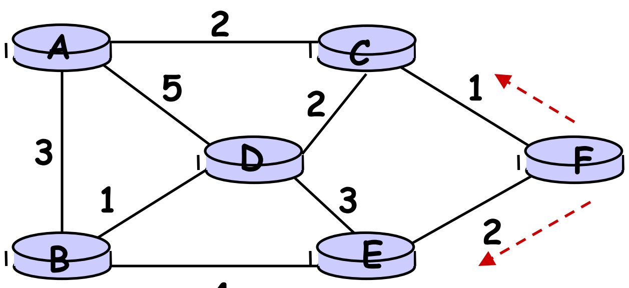


worst case

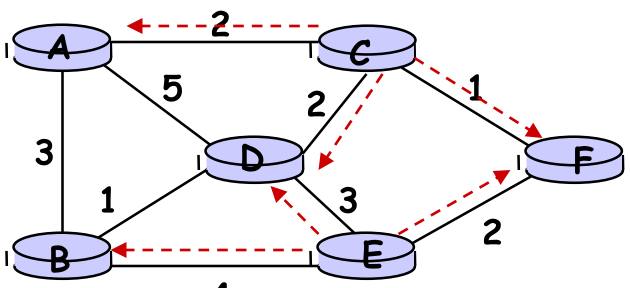
Distance-Vector Routing: Distributed B-F (cont.)

Example [Distributed Bellman-Ford Algorithm]

Assume each node i maintains an entry (R(i,x), L(i,x)), where R(i,x) is the next node along the current shortest path and L(i,x) is the current minimum cost from node i to the destination x. If the next node is not defined, we set R(i,x) to –1.



iteration 1



iteration 2

Execution of the Bellman-Ford algorithm for destination node F

Initially all nodes, other than the destination node F, are at infinite cost (distance) to node F.

***Iteration 1*: Node F informs its neighbours it is distance 0 from itself, with a timestamp.**

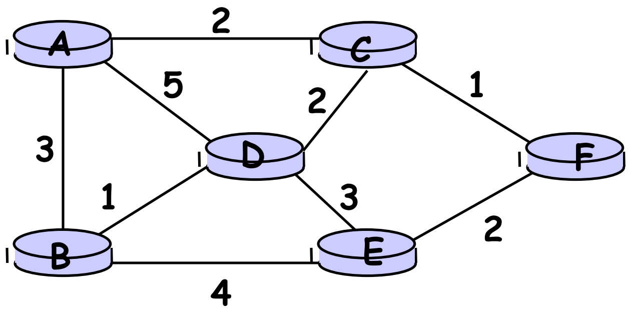
***Iteration 2*: Node C finds that it is connected to node F with cost 1.**
Node E finds that it is connected to node F at a cost of 2.
Nodes C and E update their entries and inform their neighbours.

Distance-Vector Routing: Distributed B-F (cont.)

Iteration 3: Node A finds that it can reach node F via node C with cost 3.
Node B finds that it can reach node F via node E with cost 6.
Node D finds it has paths via nodes C and E with costs 3 and 5 respectively, and it selects the path via node C.
Nodes A, B, and D update their entries and inform their neighbours.

Iteration 4: Node B finds that it can reach node F via node A, D, and E with distance 6, 4, and 6 respectively, and it selects the path via node D.
Node B changes its entry to (D,4) and informs its neighbours.

Iteration 5: Nodes A, D, and E process the new entry from node B but do not find any new shortest paths. The algorithm has converged.



routing-table entries for F only

Iteration	Node A	Node B	Node C	Node D	Node E
1	(F, ∞, -1)	(F, ∞, -1)	(F, ∞, -1)	(F, ∞, -1)	(F, ∞, -1)
2	(F, ∞, -1)	(F, ∞, -1)	(F, 1, F)	(F, ∞, -1)	(F, 2, F)
3	(F, 3, C)	(F, 6, E)	(F, 1, F)	(F, 3, C)	(F, 2, F)
4	(F, 3, C)	(F, 4, D)	(F, 1, F)	(F, 3, C)	(F, 2, F)
5	(F, 3, C)	(F, 4, D)	(F, 1, F)	(F, 3, C)	(F, 2, F)

When a node running the DV algorithm detects a change in its distance table, it updates its neighbours.