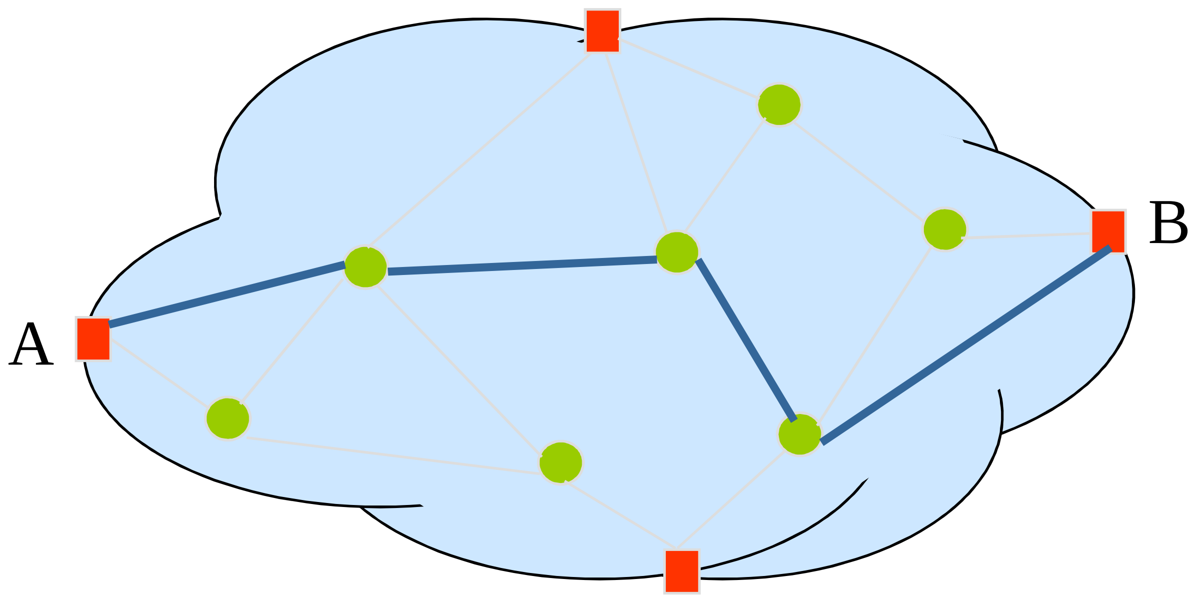
How To Route Packets

- Unicast Routing
  - Distance Vector Basics
  - Link State Basics

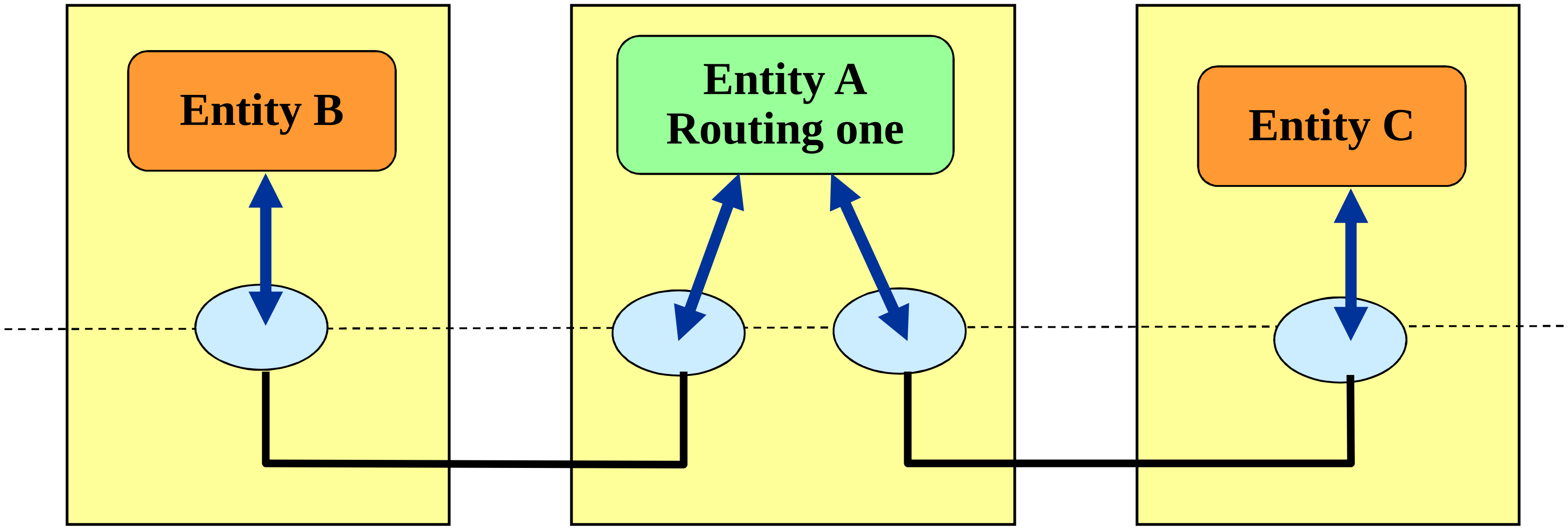
Unicast Routing

- *Routing* functionalities are fundamental for internetworking
- In TCP/IP networks:
  - Routing allows the communication of two nodes A and B not directly connected



Unicast Routing

- Layer 3 entities along the path route (choose the exit SAP) packets according to the destination address
- The correspondence Exit SAP – destination address is stored in the routing table



Routing Protocol

- Comprises two different functionalities
  - Info exchange on network topology, traffic, etc. (1)
  - *routing table creation and maintenance (2)*
- Formally, (1) is the routing protocol
- Practically, (1) and (2) are joint phases. The way the routing tables are created depends on the routing message exchange and viceversa

Routing Algorithms

- A routing algorithm defines the criteria on how to choose a path between a source and a destination…
- …and builds the routing tables
- The choice criteria depend on the type of network (*datagram*, virtual circuit)

Routing and Network Capacity

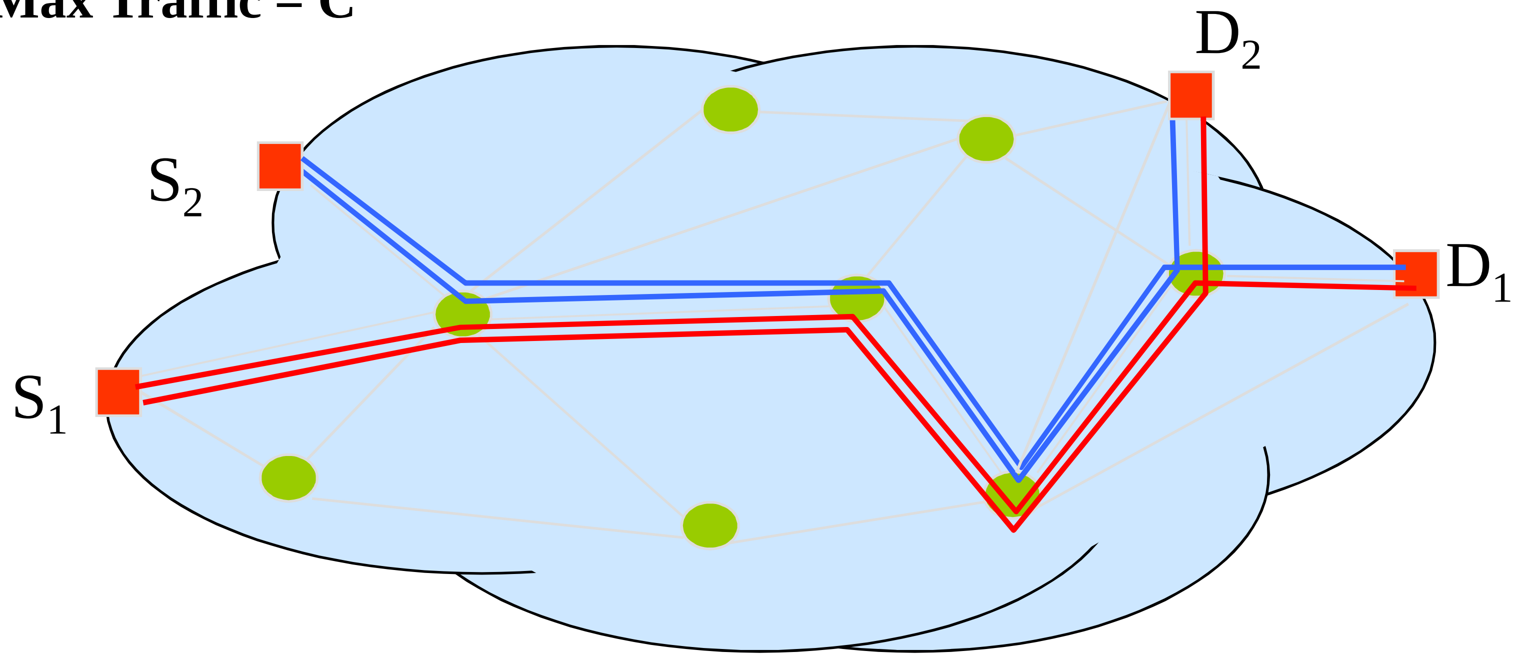
- In broadcast networks no need of routing
- Thus the maximum supported traffic depends on the capacity of the channel
- In meshed IP networks, multiple links can be used at the same time
- Thus, WHICH links are used has a deep impact on the network capacity

Routing and Capacity

- Dumb Routing Planning

Link Capacity = C

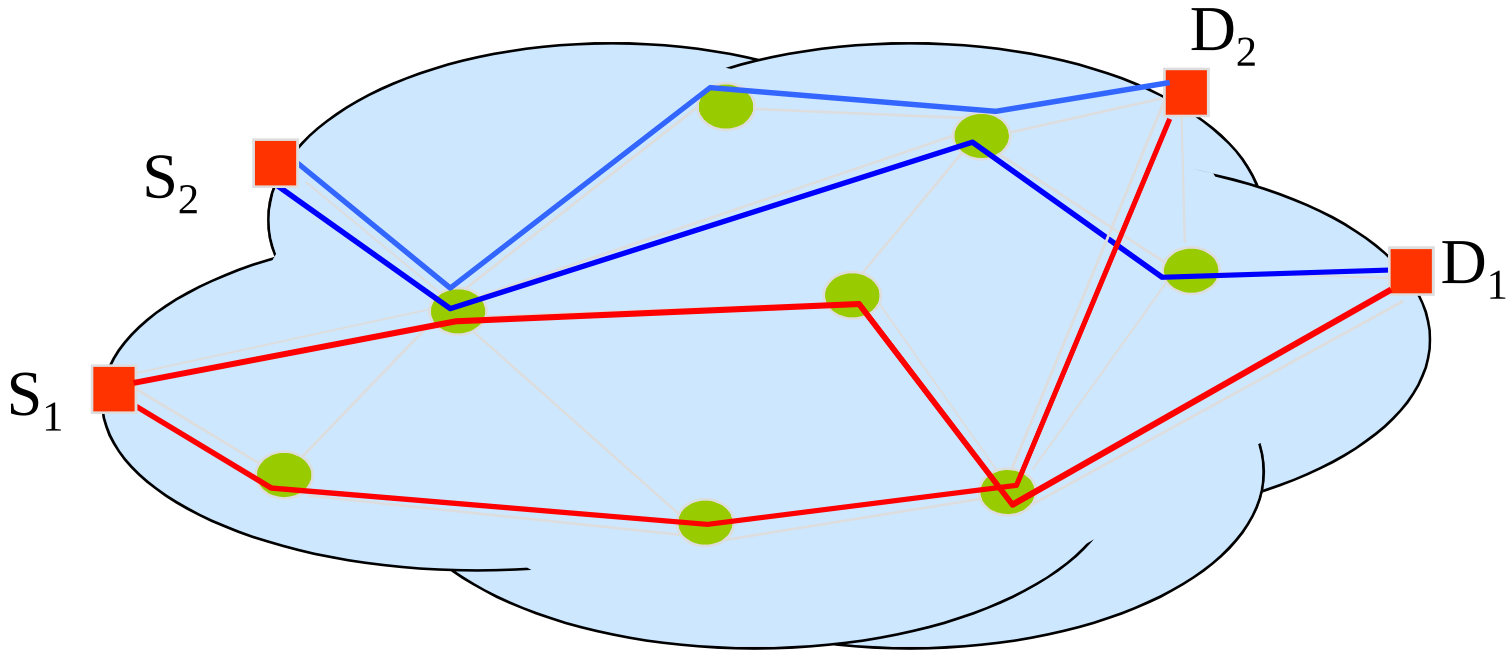
Max Traffic = C



Routing and Capacity

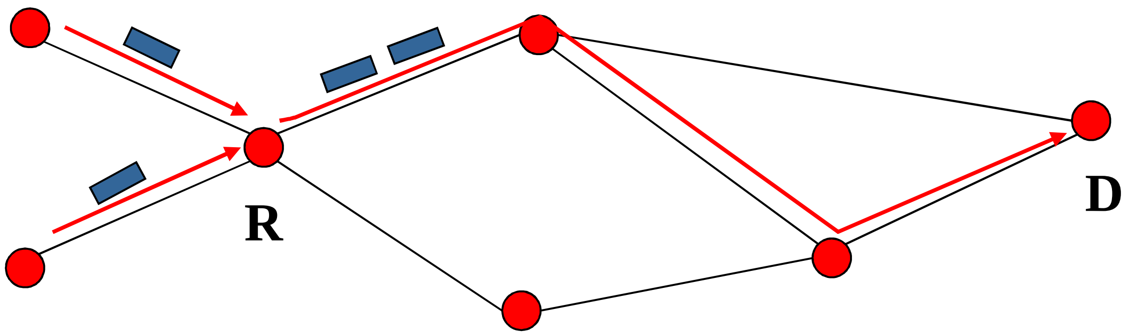
- Wise routing planning

◆ **Link Capacity = C**
◆ **Max Traffic = 3C**



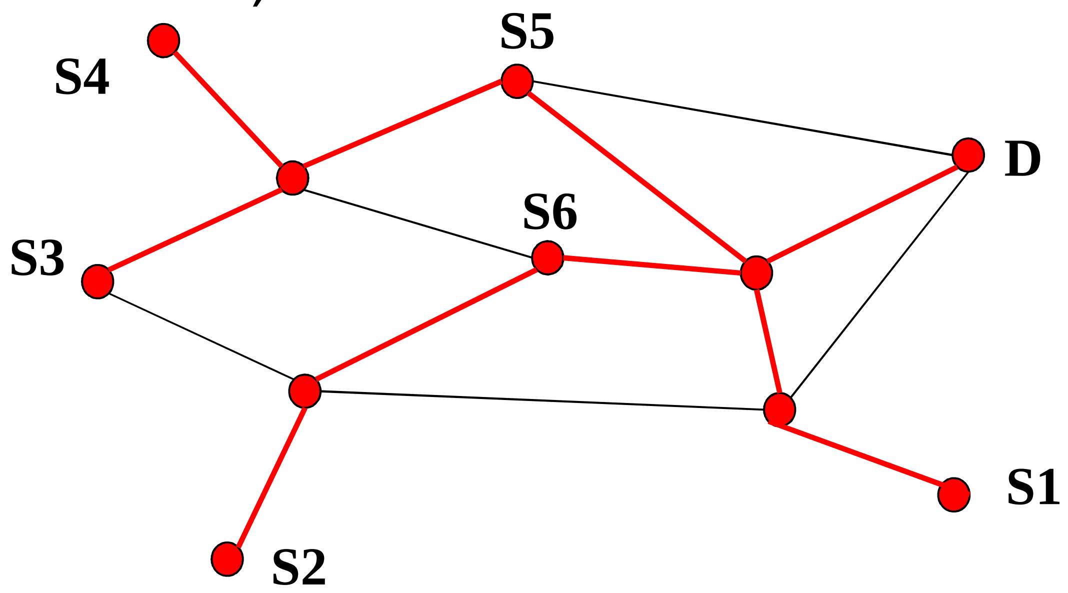
Routing in the Internet

- The type of *forwarding* impacts the routing policy
- *IP forwarding* is:
  - *destination-based*
  - *next-hop based*
- Consequence:
  - *All the packs destined to D arriving at router R follow the same path after R*



Routing in the Internet

- Thus, we have the following constraints on the routing:
  - *All the paths from all the sources to a destination D must form a tree, for each D*



  - *Source-Destination Couples cannot be routed independently from other couples.*

Shortest Path Routing

- TCP/IP Routing: the shortest path to a destination is chosen
- The computation of the shortest path is performed on the graph representing the network (device=vertex, link=edge, edge weight=metrics)
- Shortest Path properties:
  - All the paths to a destination form a tree
  - Easy and simple algorithms (polynomial complexity, even distributed)

Routing Algorithms

A Flavor of Graph Theory

Bellman-Ford algorithm

Dyijkstra algorithm

Some Definition on Graphs

- **digraph** $G(N,A)$
  - N nodes
  - $A=\{(i,j), i \in N, j \in N\}$ edges (ordered couple of nodes)
- **path**: $(n_1, n_2, \ldots, n_l)$ set of nodes with $(n_i, n_{i+1}) \in A$, *without repeated nodes*
- **cycle**: route with $n_1 = n_l$
- **Connected digraph**: for each couple i and j at least one path from i to j exists
- **Weighted digraph**: d_{ij} *weights associated to* the edge $(i,j) \in A$
- **Path** $(n_1, n_2, \ldots, n_l)$ **length** :

$$d_{n1,\, n2} + d_{n2,n3} + \ldots + d_{n(l-1),\, nl}$$

Finding the Shortest Path

Given *G(N,A)* and two nodes *i* and *j*, find the path with minimum length

- The problem has polynomial complexity in the number of nodes

Property:

If node k is traversed by the shortest path from i to j, also the path from i to k is the shortest

Bellman-Ford Algorithm

- Assumptions:
  - Positive-negative weights
  - No negative cycles
- Target:
  - Find the shortest paths from a source to all the other nodes
  - Find the shortest paths from all the nodes to a destination

Bellman-Ford Algorithm

- Variables:
  - $D_i^{(h)}$: length of the shortest path from the source (assumed to be node 1) to node *i* with a number of hops $\leq h$
- Initialization:

$$D_1^{(h)} = 0 \qquad \forall h$$

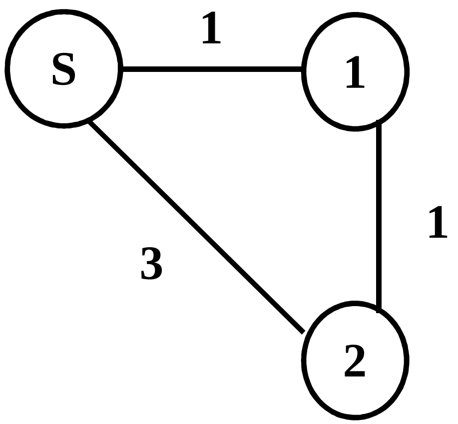
$$D_i^{(0)} = \infty \qquad \forall i \neq 1$$

- Iterations:

$$D_i^{(h+1)} = \min\left[D_i^{(h)}, \min_j \left(D_j^{(h)} + d_{ji} \right) \right]$$

- The algorithm stops after N-1 iterations

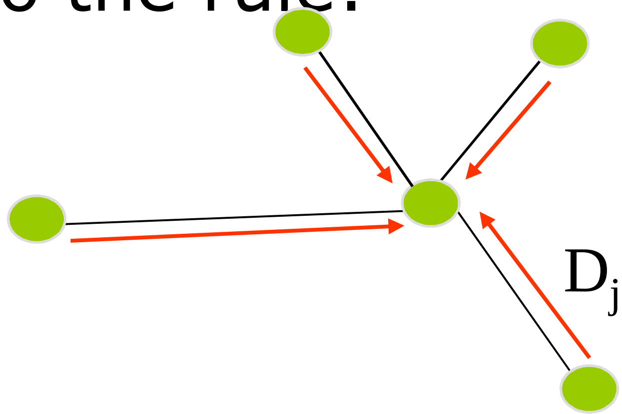
An Example



- Initialization
  - $D_s^h=0$
  - $D_1^0=inf$
  - $D_2^0=inf$
- First Iteration
  - $D_1^1=min\ (D_1^0\ ,\ D_s^0+1)=1$, NH:S
  - $D_2^1=min\ (D_2^0\ ,\ D_s^0+3)=3$, NH:S
- Second Iteration
  - $D_1^2=min\ (D_1^1\ ,\ D_2^1+1)=1$, NH:S
  - $D_2^2=min\ (D_2^1\ ,\ D_1^1+1)=2$, NH:1

Distributed *Bellman-Ford*

- It can be shown that the algorithm does converge in a *finite number* of iterations, even in its distributed form
- Nodes periodically send out their estimation of the shortest path and update such estimation according to the rule:

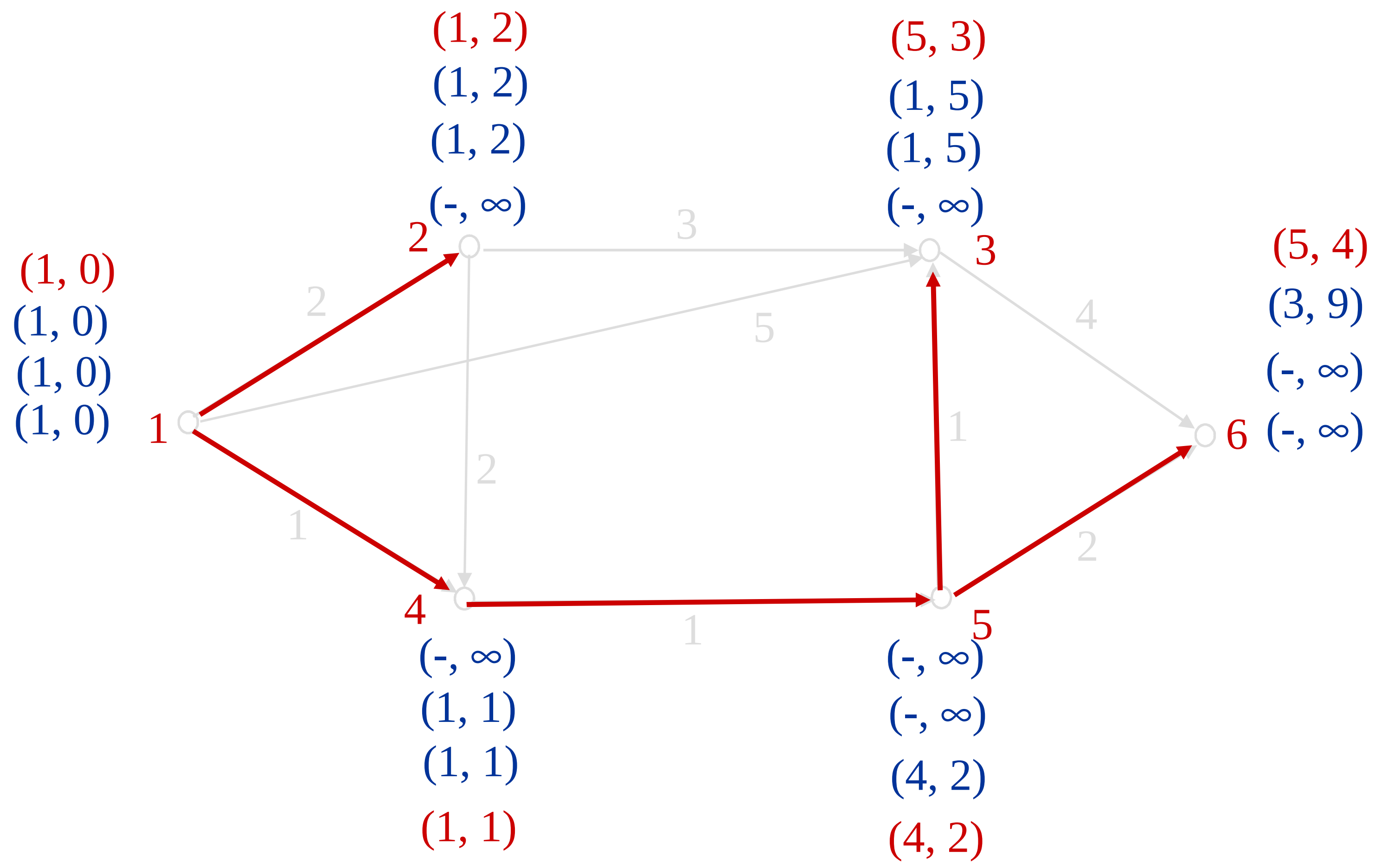


$$D_i := \min\left[D_i \,, \min_j \left(D_j + d_{ji} \right) \right]$$

Bellman-Ford **in practice**

- Each node is assigned a label *(n, L)* where *n* is the next hop on the path and *L* is the path length
- Each node updates its label looking at its neighbors' labels
- When the labels do not change any longer the shortest path tree can be built

Example : Bellman-Ford



Dijkstra Algorithm

- Assumptions:
  - Positive weighted edges
- Target:
  - Find out the shortest paths form a source node (1) and all the other nodes
- Initialization:

$$P = \{1\},$$

$$D_1 = 0, \quad D_j^{(0)} = d_{1j} \quad \forall j \neq 1$$

  - $d_{ij} = \infty$ if the edge i-j does not exist

Dijkstra Algorithms

1. find $i \in (N\text{-}P)$ *:*

$$D_i = \min_{j \in (N-P)} D_j$$

and set

$$P := P \cup \{i\}. \ \textit{If } P = N, \textit{then STOP.}$$

2. for each $j \in (N\text{-}P)$ *neighbor of any node in P set :*

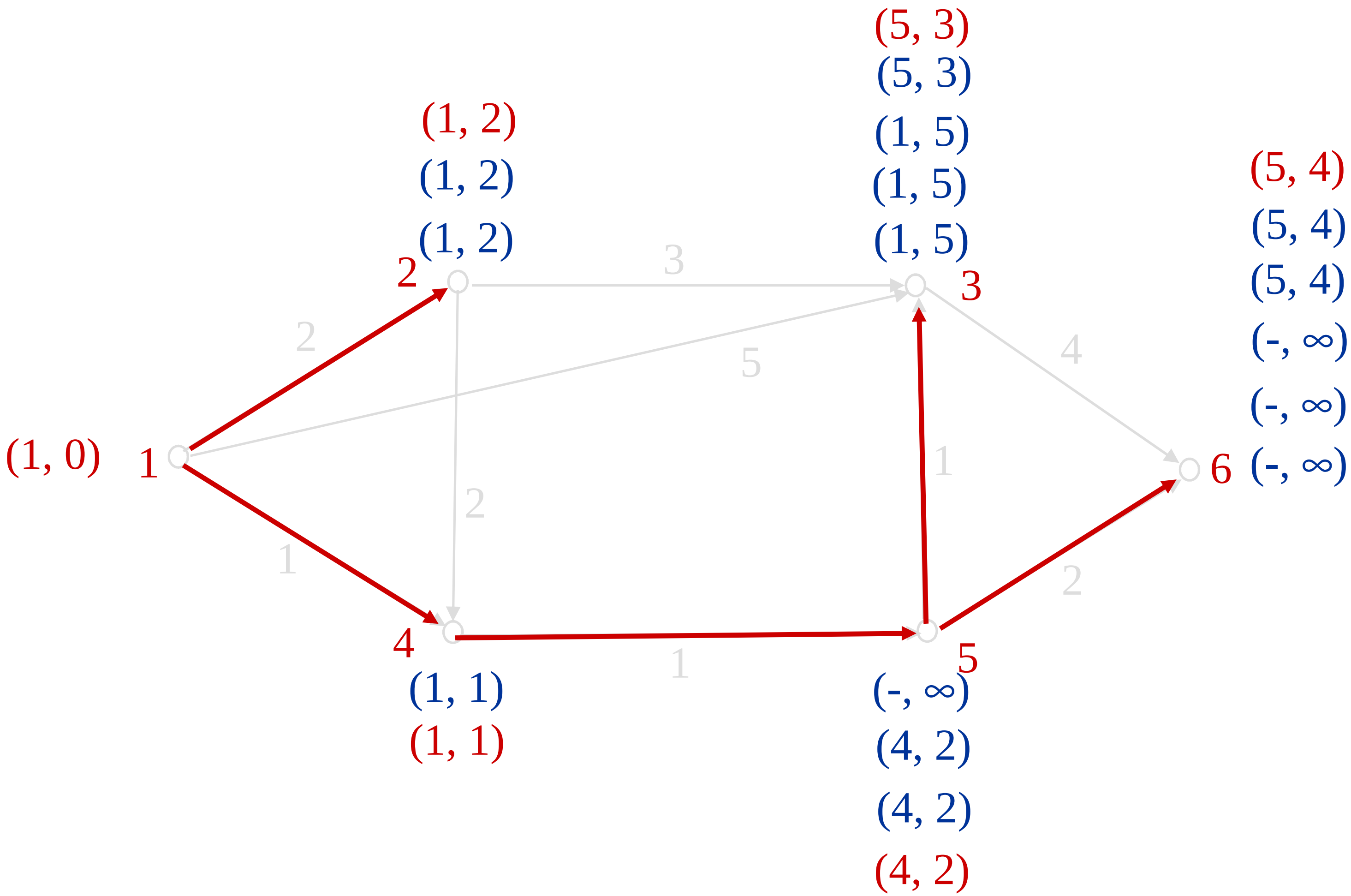
$$D_j = \min\left[D_j, \min_k \left(D_k + d_{kj}\right)\right]$$

3. Go To 1.

Dijkstra in practice

- Same label criteria as Bellman-Ford
- Label can be *temporary* or *permanent*
- In the beginning, the only permanent label is the one of the source
- At each iteration, the temporary label with the lowest cost of the path is made permanent

Example: Dijkstra



On Complexity

- Bellman-Ford:
  - N-1 iterations
  - N-1 nodes to be checked each iteration
  - N-1 comparisons per node
    - Complexity: $O(N^3)$
- Dijkstra:
  - N-1 iterations
  - N operations each iteration on average
    - Complexity: $O(N^2)$
- Dijkstra is generally more convenient

Routing IP

- Sends packet on the shortest path to the destination
- The length of the path is measured according to a given metrics
- The shortest path computation is implemented in a distributed way through a *routing protocol*
- In the routing table, *only* the next hop is stored, thanks to the property that sub-paths of a shortest path are shortest themselves.

Routing Protocols

- Handle the message exchange among routers to compute the paths to a destination
- Two classes
  - Distance Vector (RIP, IGRP)
  - Link State (OSPF,IS-IS)
- Differences
  - Type of metrics
  - Type of messages exchanged
  - Type of procedures used to exchange messages

POLYTECH
PARIS-SUD

Distance Vector Routing Protocols

Distance Vector Protocols

- Routers exchange specific connectivity information: the Distance Vector *(DV)*:

  [destination address, distance]

- DV is sent *only* to directly connected routers
- DV is sent periodically and/or whenever the network topology changes
- Distance estimation is performed using *Bellman-Ford* distributed algorithm

Distance Vector: Algorithm

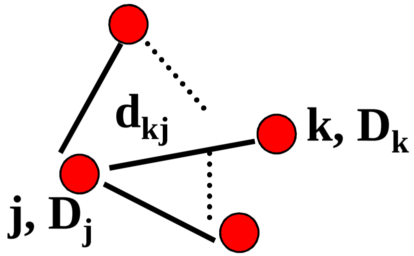
- DV reception

1. Increase the distance to the specified destination of the current link cost
2. For each specified destination
   - If the destination is not in the routing table
     - Add destination/distance
   - Otherwise
     - If the next hop in the routing table is the DV sender
       - Update the stored information with the new one
     - Otherwise
       - If the stored distance to the destination is bigger to the one specified in the DV
         - Update the stored info with the new one
3. End

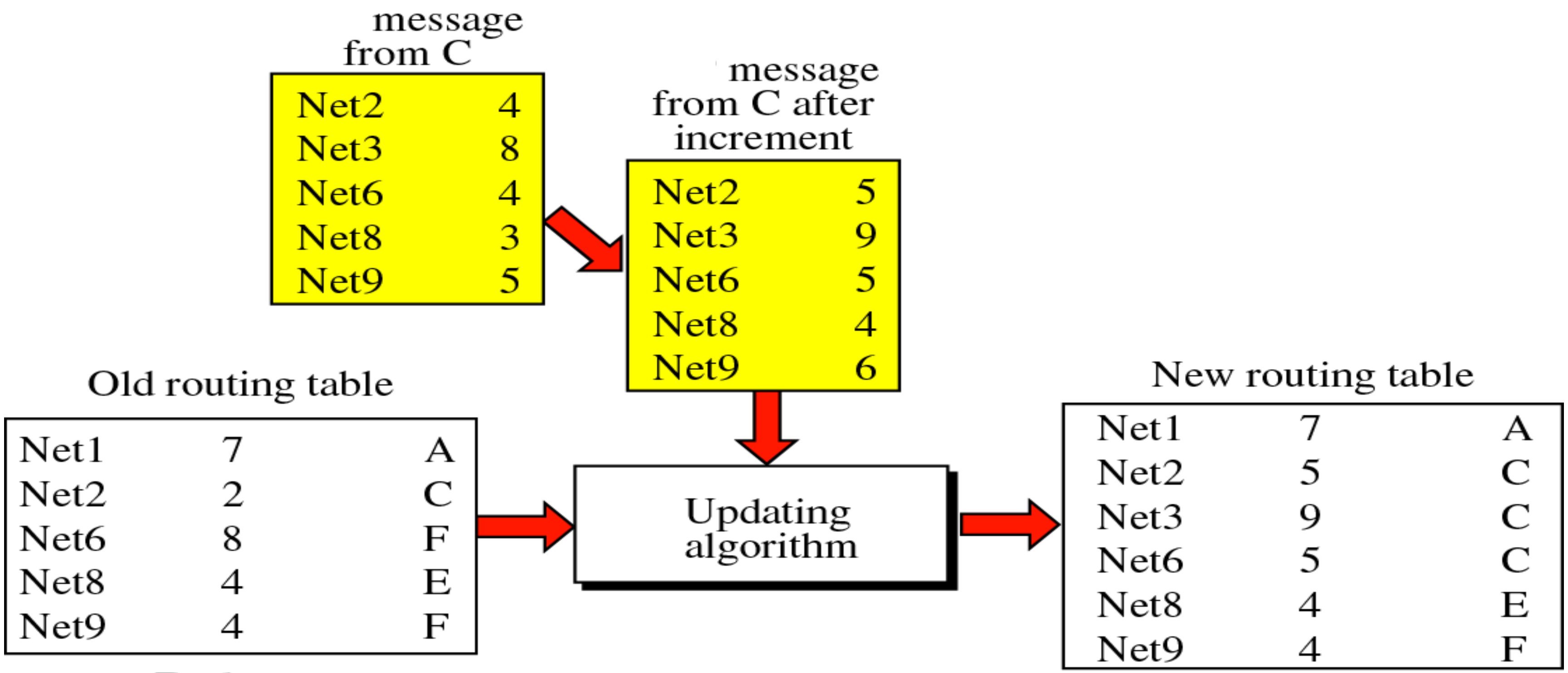
Distance Vector

- DV is sent
  - periodically
  - Whenever something changes upon the reception of another DV
- Routers calculate distances if:
  - A new DV is received
  - Something changes in the local network topology (local link failure)

Computation: $D_j' = \min_k [\ D_k + d_{kj}\ ]$



Routing Tables Update

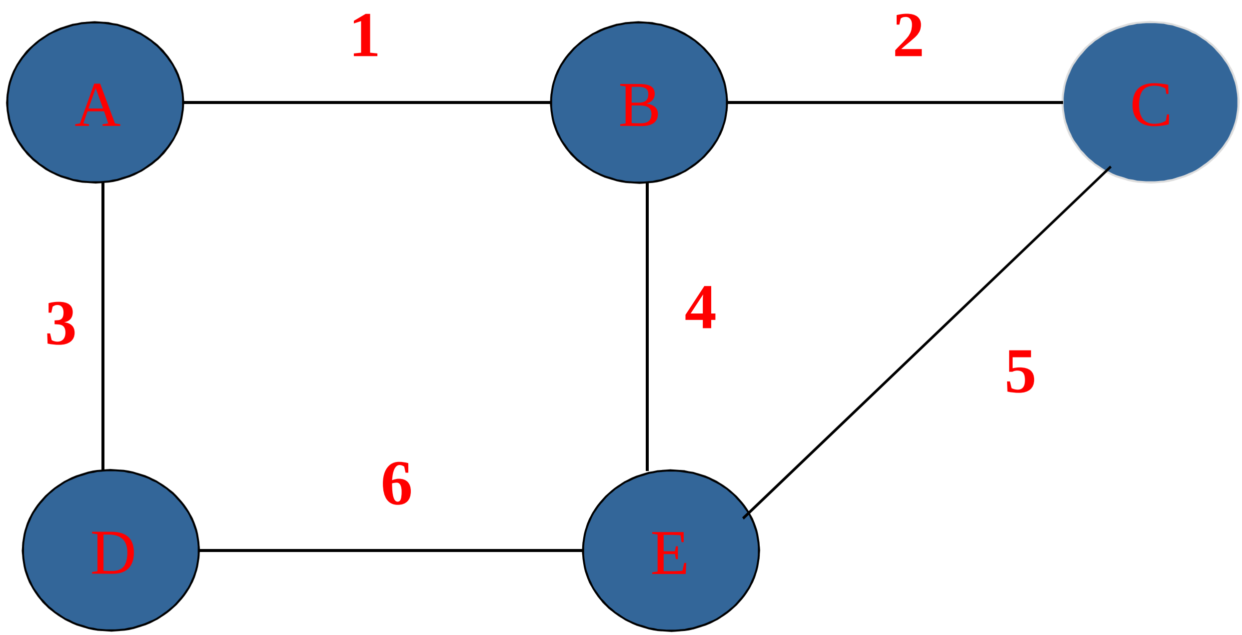


Rules

Net1: No news, don't change
Net2: Same next hop, replace
Net3: A new router, add
Net6: Different next hop, new hop count smaller, replace
Net8: Different next hop, new hop count the same, don't change
Net9: Different next hop, new hop count larger, don't change

Distance Vector Example (1)

- Simple Network Topology:



- Assume each link has cost = 1

Distance Vector Example (2)

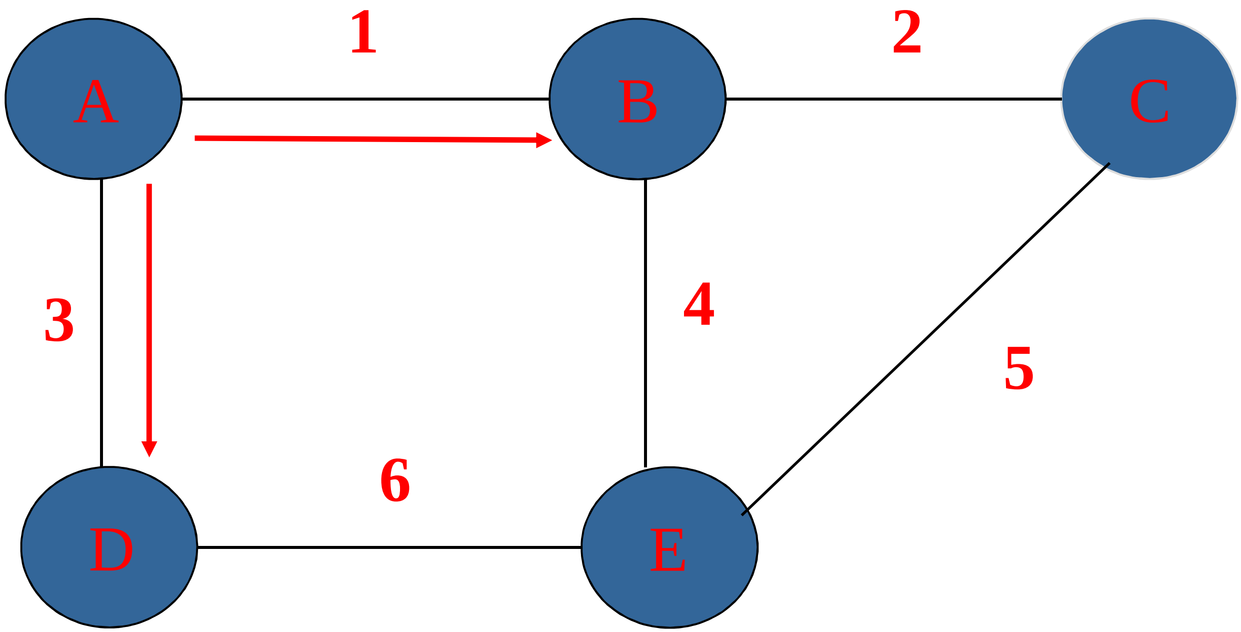
- Assume all the nodes wake up at the same time
  - ☞ *cold start procedure*
- Each node knows its local connectivity situation (directly connected links and interfaces)
- Start Up routing table for node A:

From A To	Link	Cost
A	local	0

Distance Vector Example (3)

- A sets up its *Distance Vector*

  A=0 and sends it out to all of its neighbors (on local links)
- B and D receive the DV and enlarge their knowledge of the network



Distance Vector Example (4)

- Node B, upon reception of the *Distance Vector*, updates the distance adding the link cost (A=1) and checks the DV against its routing table. A is still unknown, thus routing table update

From B To	Link	Cost
B	local	0
A	1	1

- The same thing for node D

From D To	Link	Cost
D	local	0
A	3	1

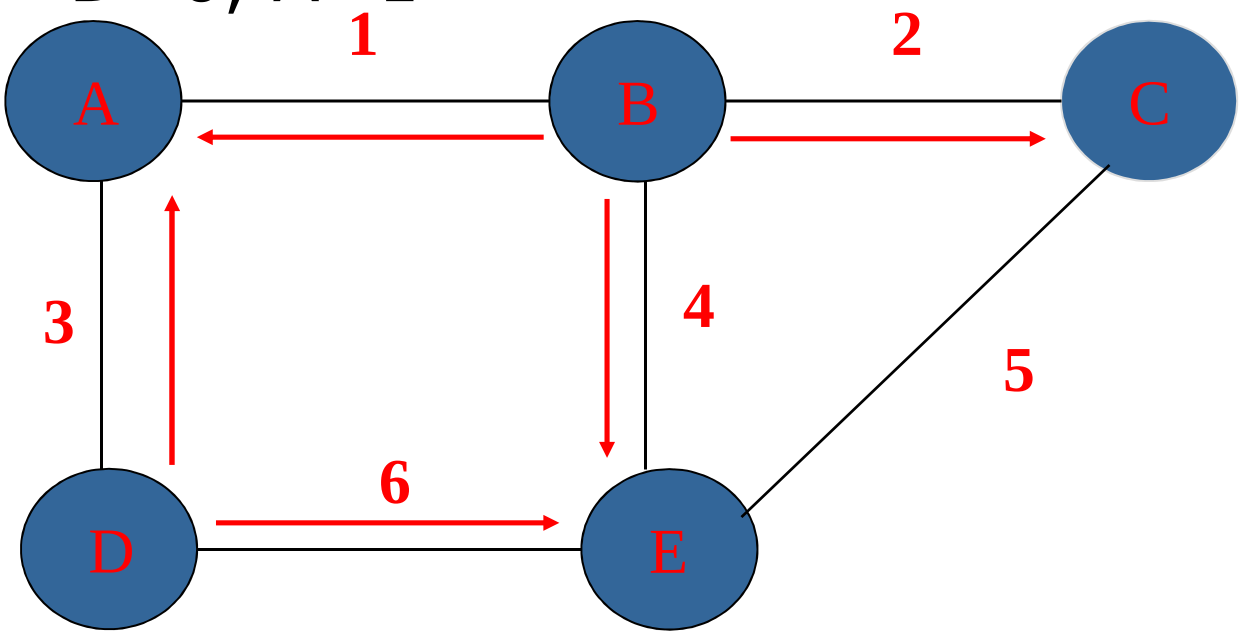
Distance Vector Example (5)

- Node B prepares its DV …

  B=0, A=1

  … and fires it through its local links
- The same for node D:

  D=0, A=1



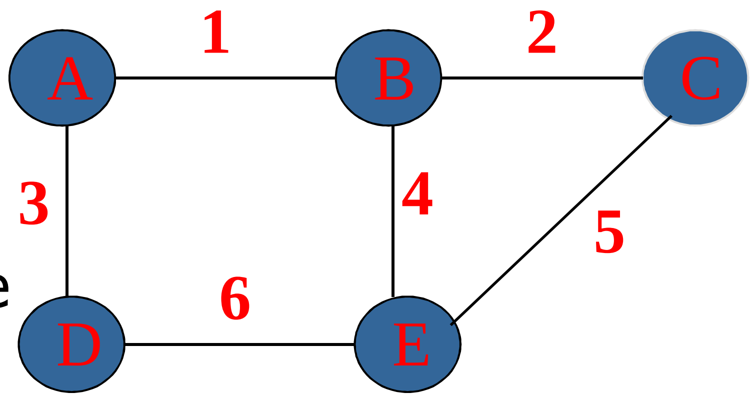
Distance Vector Example (6)

- The DV from B is received by A,C and E whilst that from D is received by A and E
- A receives the two DVs

  From B: B=0, A=1

  From D: D=0, A=1

  … and updates its routing table



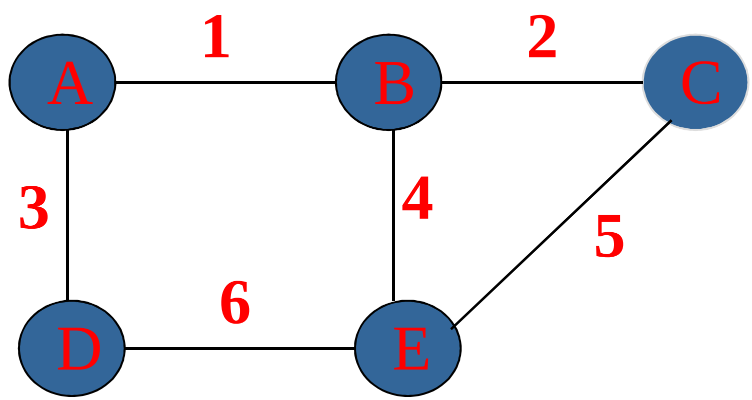
From A to	Link	Cost
A	local	0
B	1	1
D	3	1

Distance Vector Example (7)

- C receives from B on *link* 2

  B=0, A=1

  … and updates its routing table :



From C to	Link	Cost
C	local	0
B	2	1
A	2	2

Distance Vector Example (8)

- Node E receives from B on link 4

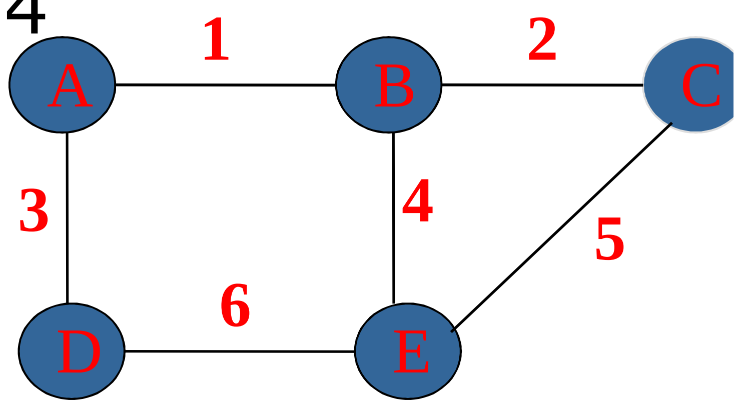
  B=0, A=1

  and from D on link 6

  D=0, A=1

  … and updates its routing table

  - Note that the distance to A is the same through links 4 and 6



From E To	Link	Cost
E	local	0
B	4	1
A	4	2
D	6	1

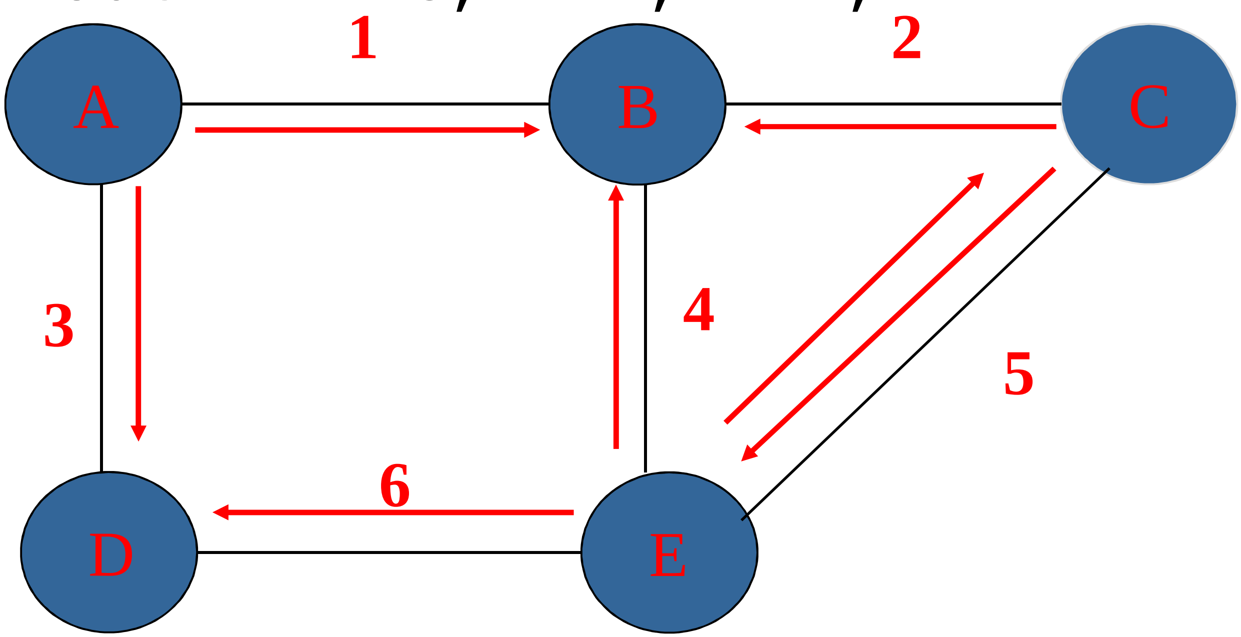
Distance Vector Example (9)

- The nodes A, C and E have updated their routing tables, hence they transmit their own DVs:

  node A: A=0, B=1, D=1

  node C: C=0, B=1, A=2

  node E: E=0, B=1, A=2, D=1



Distance Vector Example (10)

- Node B:

B	local	0
A	1	1

A: A=0, B=1, D=1
C: C=0, B=1, A=2
E: E=0, B=1, A=2, D=1

From B To	Link	Cost
B	local	0
A	1	1
D	1	2
C	2	1
E	4	1

- Node D:

D	local	0
A	3	1

A: A=0, B=1, D=1
E: E=0, B=1, A=2, D=1

From D To	Link	Cost
D	local	0
A	3	1
B	3	2
E	6	1

- Node E

E	local	0
B	4	1
A	4	2
D	6	1

C: C=0, B=1, A=2

From E verso	Link	Cost
E	local	0
B	4	1
A	4	2
D	6	1
C	5	1

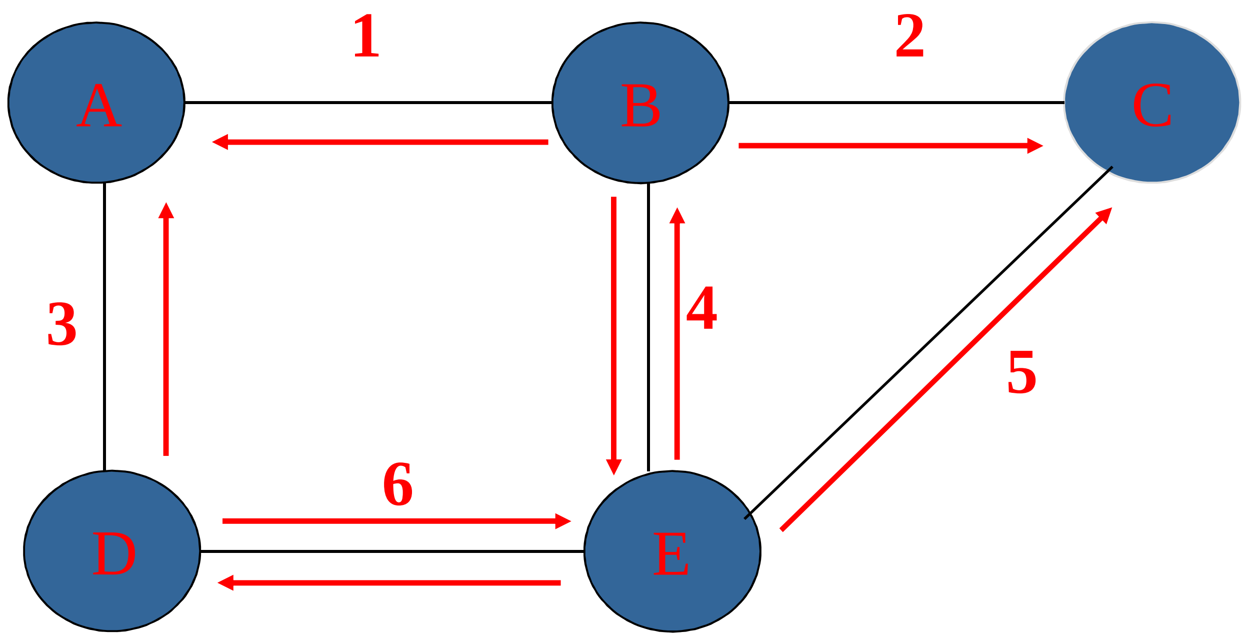
Distance Vector Example (11)

- The nodes B, D and E transmit their own DVs:

  node B: B=0, A=1, D=2, C=1, E=1

  node D: D=0, A=1, B=2, E=1

  node E: E=0, B=1, A=2, D=1, C=1



Distance Vector Example (12)

- Node A:

A	local	0
B	1	1
D	3	1

B=0, A=1, D=2, C=1, E=1

D: D=0, A=1, B=2, E=1

From A To	Link	Cost
A	local	0
B	1	1
D	3	1
C	1	2
E	1	2

- Node C:

C	local	0
B	2	1
A	2	2

B=0, A=1, D=2, C=1, E=1

E=0, B=1, A=2, D=1, C=1

From C To	Link	Cost
C	local	0
B	2	1
A	2	2
E	5	1
D	5	2

- Node D

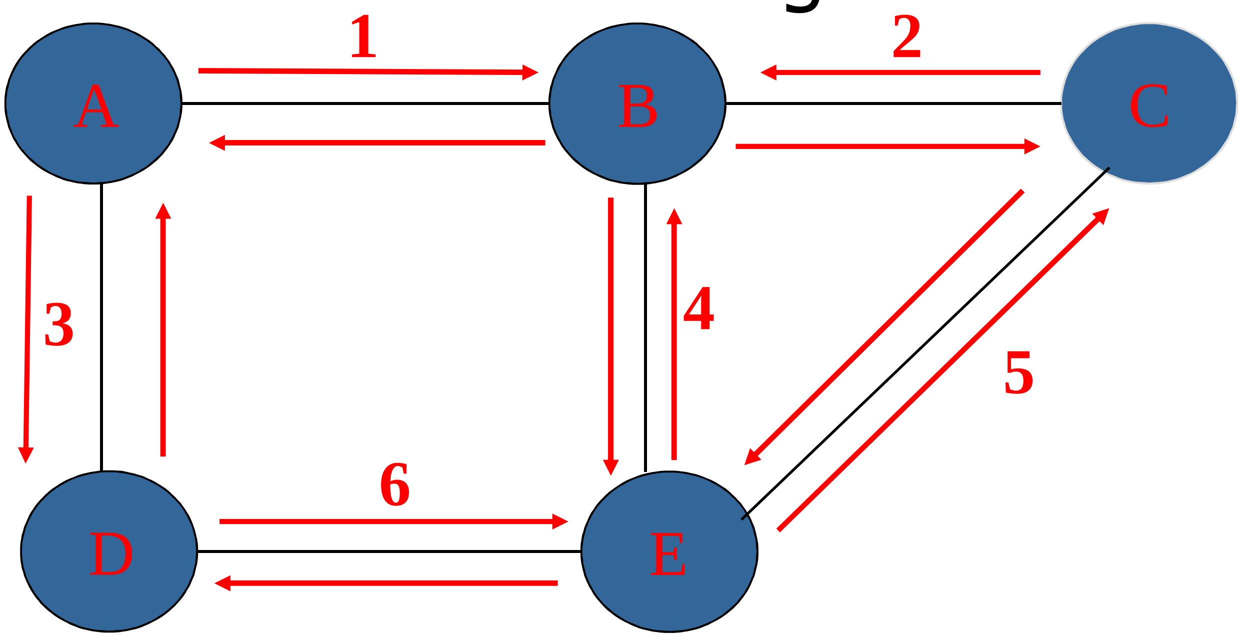
D	Local	0
A	3	1
B	3	2
E	6	1

E=0, B=1, A=2, D=1, C=1

From D To	Link	Cost
D	local	0
A	3	1
B	3	2
E	6	1
C	6	2

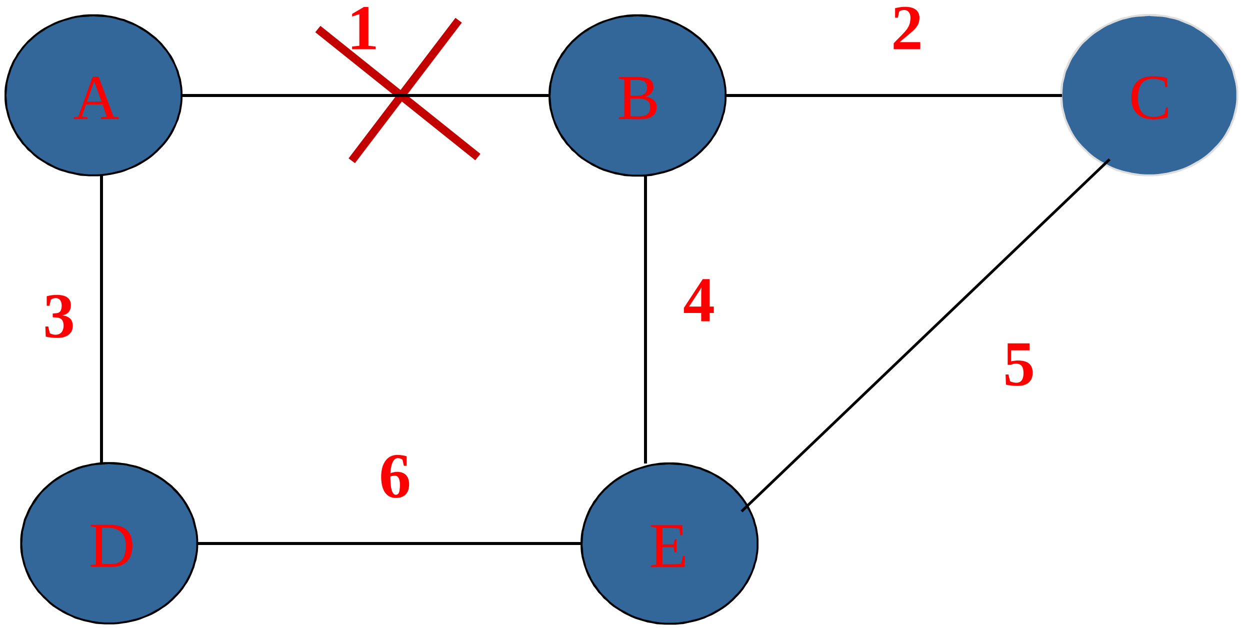
Distance Vector Example (13)

- The algorithm has reached *convergence*
- The nodes keep transmitting their DVs periodically but the routing tables do not change



Distance Vector: Link 1 Failure

- Link 1 goes down



- Nodes A and B get aware of the link failure
- … and update their routing table assigning cost = infinity to link 1

Distance Vector: Link 1 Failure

From A To	Link	Cost
A	local	0
B	1	1⇒inf
D	3	1
C	1	2⇒inf
E	1	2⇒inf

From B To	Link	Cost
B	local	0
A	1	1⇒inf
D	1	2⇒inf
C	2	1
E	4	1

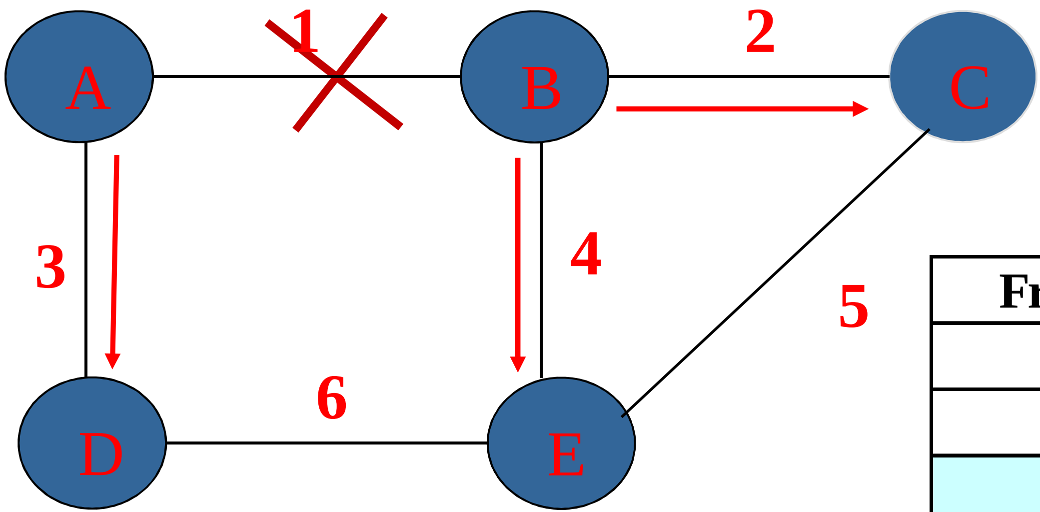
- New DVs are sent:

  node A: A=0, B=inf, D=1, C=inf, E=inf

  node B: B=0, A=inf, D=inf, C=1, E=1

Distance Vector: Link 1 Failure

- The DV from A is received by D, which compares it against its routing table
- All the costs specified in the DV are greater or equal than the ones stored in the routing table, <u>but node D updates its routing table since the link it receives the DV from is the one it uses to reach all the destinations</u>



From D to	Link	Cost
D	local	0
A	3	1
B	3	2 ⇒ inf
E	6	1
C	6	2

Distance Vector: Link 1 Failure

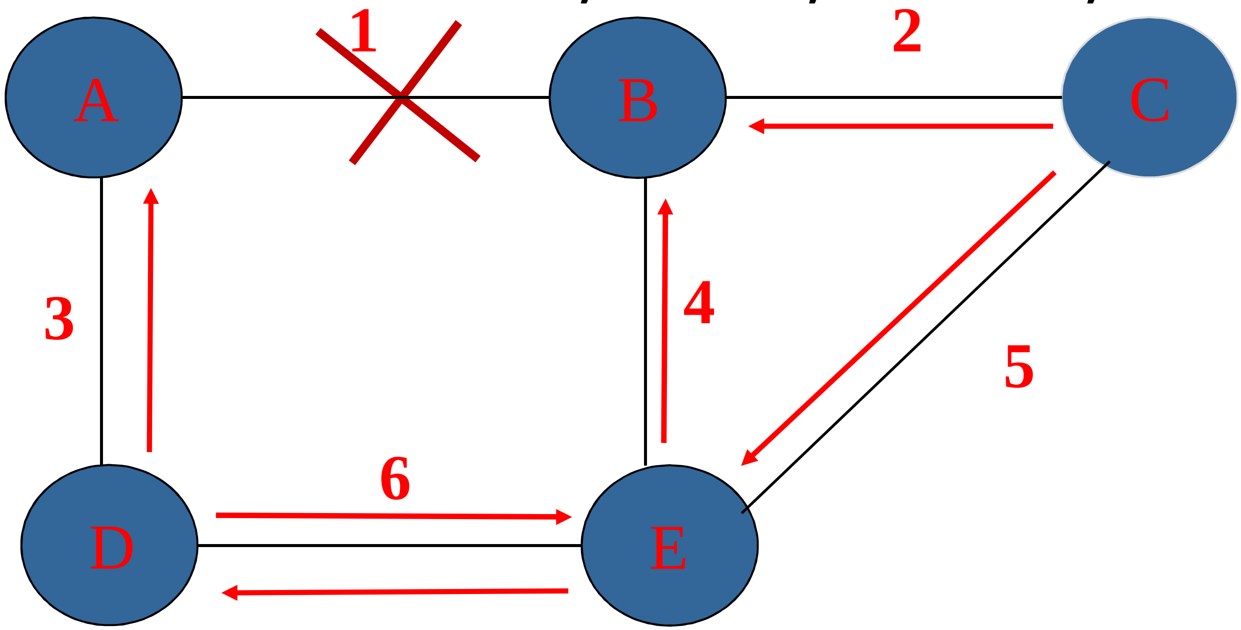
- Also nodes C and E update their tables

From C to	Link	Cost
C	local	0
B	2	1
A	2	2⇒inf
E	5	1
D	5	2

From E to	Link	Cost
E	local	0
B	4	1
A	4	2⇒inf
D	6	1
C	5	1

Distance Vector: Link 1 Failure

- nodes D, C and E transmit their DVs
  - node D: D=0, A=1, B=inf, E=1, C=2
  - node C: C=0, B=1, A=inf, E=1, D=2
  - node E: E=0, B=1, A=inf, D=1, C=1



Distance Vector: Link 1 Failure

- These DVs update the tables of A,B,D and E

From A to	Link	Cost
A	local	0
B	1	inf
D	3	1
C	1⇒3	inf⇒3
E	1⇒3	inf⇒2

From B To	Link	Cost
B	local	0
A	1	inf
D	1⇒4	inf⇒2
C	2	1
E	4	1

From D To	Link	Cost
D	local	0
A	3	1
B	3⇒6	inf⇒2
E	6	1
C	6	2

From E To	Link	Cost
E	local	0
B	4	1
A	4⇒6	inf⇒2
D	6	1
C	5	1

Distance Vector: Link 1 Failure

- Nodes A,B,D and E transmit the new DVs
  - node A: A=0, B=inf, D=1, C=3, E=2
  - node B: B=0, A=inf, D=2, C=1, E=1
  - node D: D=0, A=1, B=2, E=1, C=2
  - node E: E=0, B=1, A=2, D=1, C=1
- A, B and C update their tables

From A To	Link	Cost
A	local	0
B	1⇒3	inf⇒3
D	3	1
C	3	3
E	3	2

From B To	Link	Cost
B	local	0
A	1⇒4	inf⇒3
D	4	2
C	2	1
E	4	1

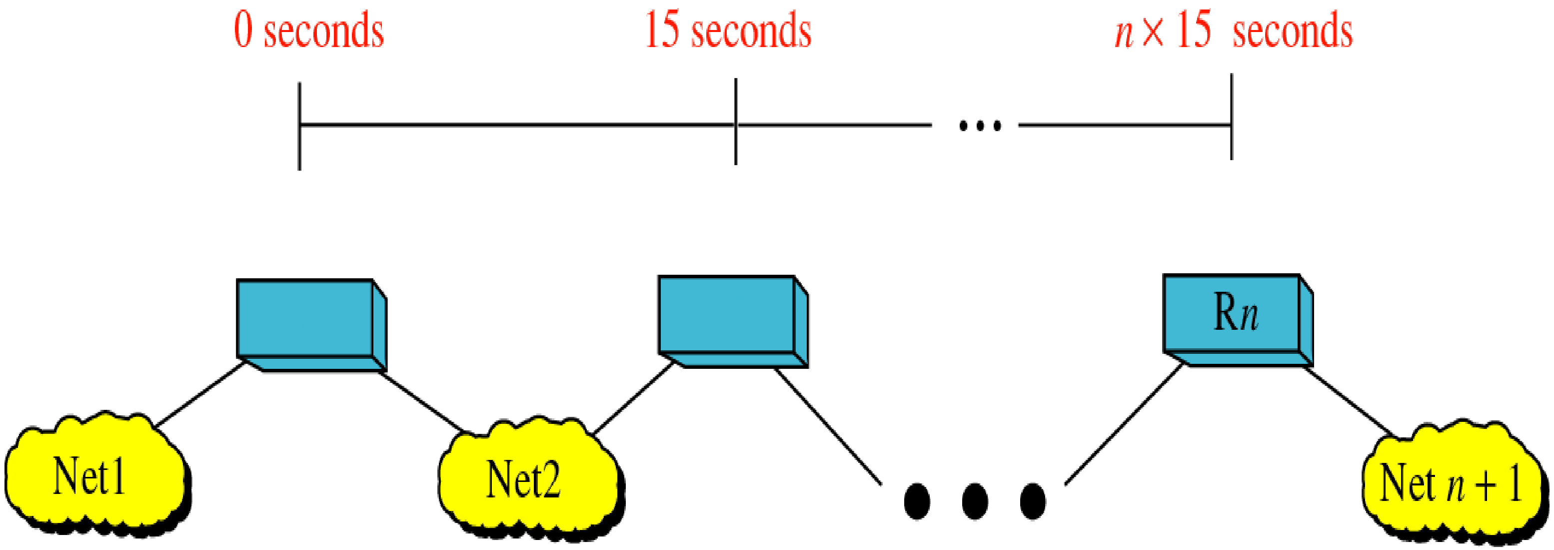
From C To	Link	Cost
C	local	0
B	2	1
A	2⇒5	inf⇒3
E	5	1
D	5	2

- The algorithm has reached a new steady state !!!

Distance Vector: Main Features

- PROs:
  - Very easy
- CONs:
  - High time to convergence
  - Limited by the lowest node
  - Possible loops
  - Instability in big networks
    (*counting to infinity)*

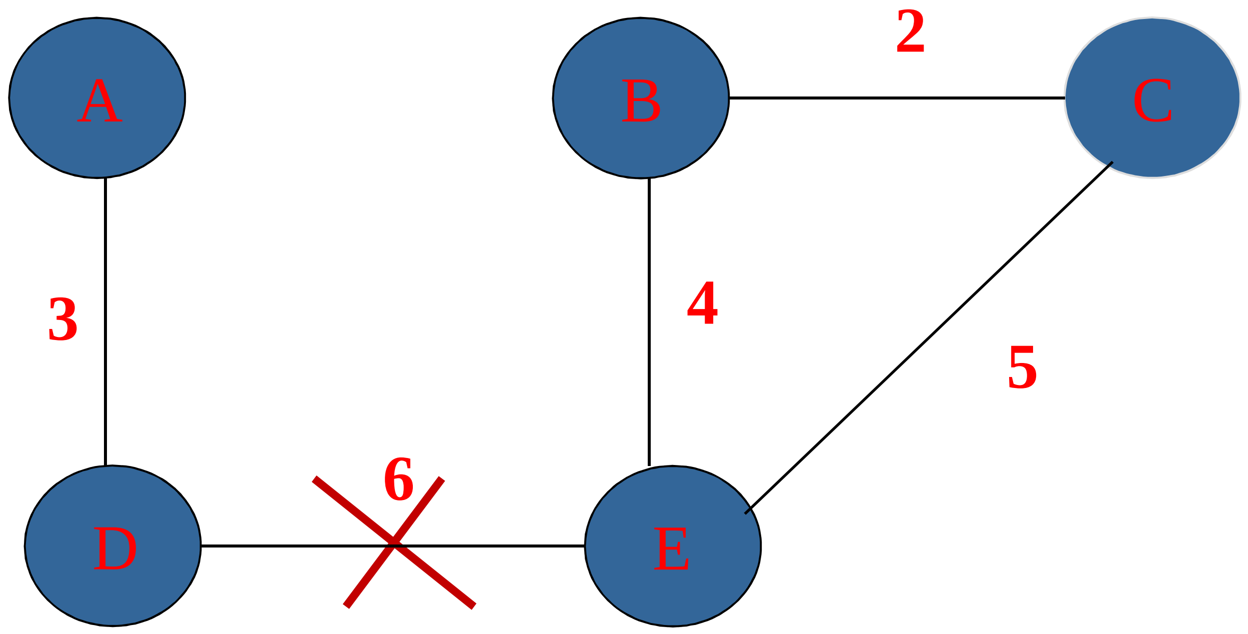
Convergence Time



- Grows proportionally with the number of nodes (Low Scalability)

Distance Vector: *counting to infinity*

- Suppose *link* 6 goes down



Distance Vector: counting to infinity

- Node D detects *link* 6 failure and updates its *routing table*

From D To	Link	Cost
D	local	0
A	3	1
B	6	2⇒inf
E	6	1⇒inf
C	6	2⇒inf

- if D immediately transmits the new DV, node A updates its routing table (the only reachable node is D)

Distance Vector: counting to infinity

- Buf if node A transmits its DV *before* D; what happens?

  node A: A=0, B=3, D=1, C=3, E=2

➡ node D updates its routing table !!!

From D To	Link	Cost
D	local	0
A	3	1
B	6⇒3	inf⇒4
E	6⇒3	inf⇒3
C	6⇒3	inf⇒4

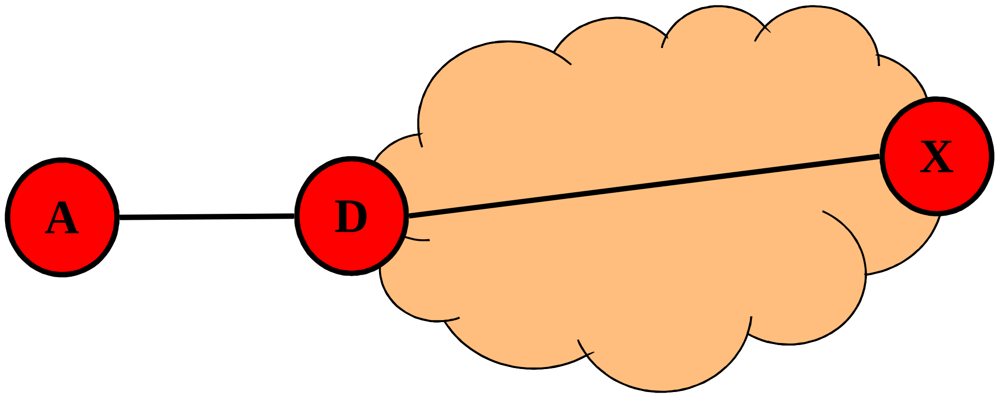
- A *loop* is created between nodes A and D
- The algorithm does *not* reach convergence
- At each step the distances to B, C and E grows by 2 ☛counting to infinity

Counting to infinity: Remedies

- Hop Count Limit:
  - The *counting to infinity* is broken if infinity is represented by a finite value
  - Such value must be bigger than the length of the longest path in the network
  - When any distance reaches such value the corresponding node is declared unreachable
  - During the *counting to infinity* :
    - Packets *loop*
    - Congested links
    - High packet loss probability (including routing packets)
  - ☛ Convergence may be very slow

Counting to infinity: Remedies

- Split-Horizon:
  - if node A sends to D the packets meant for X, it's *pointless* for A to announce X in its own DV to D

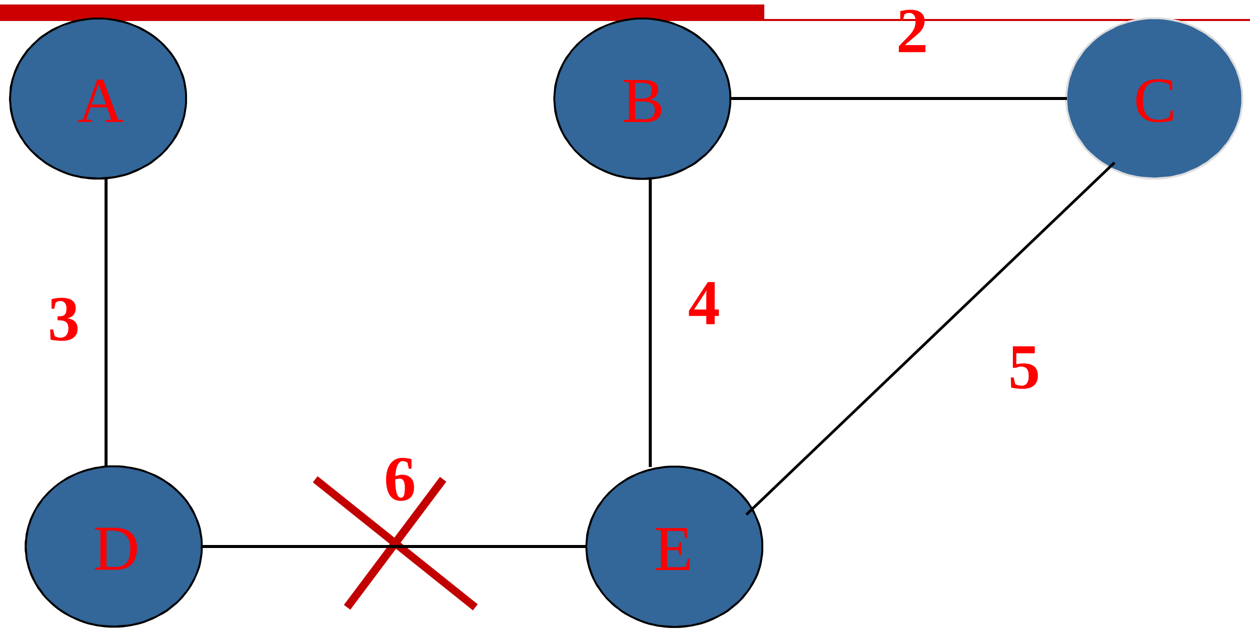


  - node A does *not* advertise to D the destination X

Distance Vector: Split Horizon

- Node A sends different DV on different local *links*
- *Two Flavors of Split Horizon:*
  - *Basic:* the node omits any information on the destination which it reaches through the link it is using
  - *Poisonous Reverse*: the node includes all the destinations, setting to infinity the distance to those reachable through the link it is using
- Split Horizon does not work with some topologies

Distance Vector: Split Horizon



- when *link* 6 goes down this is the situation of nodes B,C and E

From	Link	Cost
B to D	4	2
C to D	5	2
E to D	6	1⇒ inf

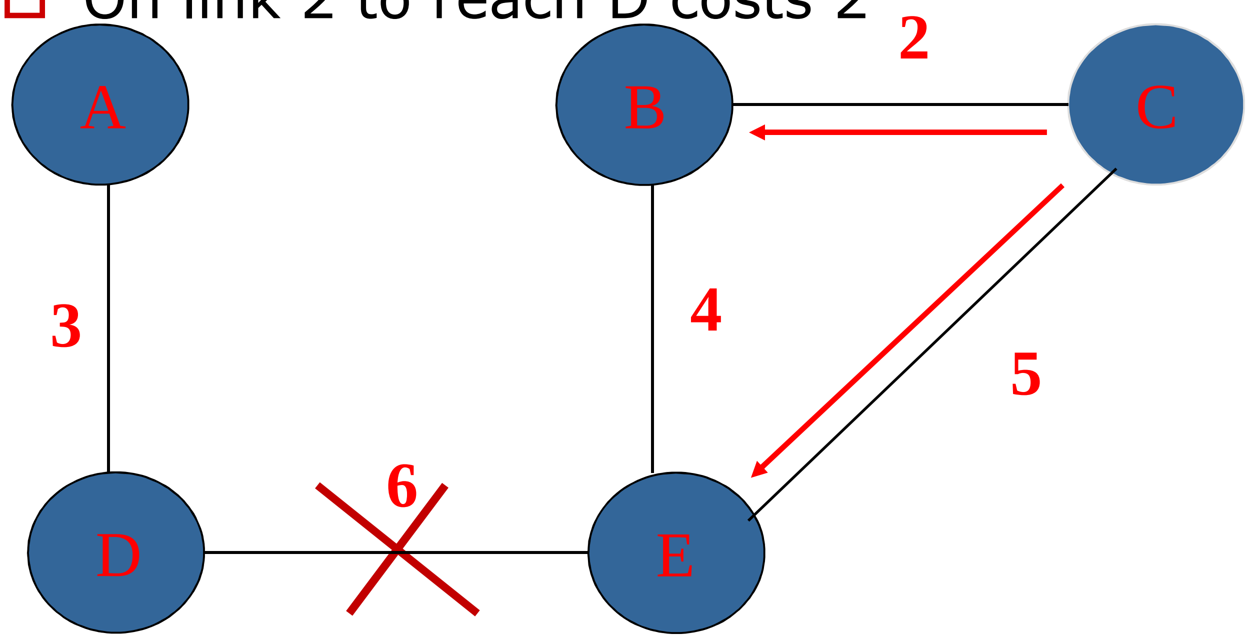
Distance Vector: Split Horizon

- Node E advertises on links 4 and 5 that the distance to D is infinity
- Suppose that such message is received by B but not by C (for example, due to an error on such routing message/packet)

From	Link	Cost
B to D	4	2 ⇒ inf
C to D	5	2
E to D	6	inf

Distance Vector: Split Horizon

- Node C fires its DV (*Split Horizon* with *Poisonous Reverse On)*
  - To node E: C=0, B=1, A=inf, E=inf, D=inf
    - On link 5 to reach D costs infinity
  - to node B: C=0, B=inf, A=3, E=1, D=2
    - On link 2 to reach D costs 2



Distance Vector: Split Horizon

- B updates its routing table and sends its DV (*Split Horizon Poisonous Reverse On)*:
  - on *link* 2 D is reachable with cost = infinity
  - on *link* 4 D is reachable with cost 3
- nodes B,C and E:

From	Link	Cost
B to D	**4⇒2**	**inf⇒3**
C to D	**5**	**2**
E to D	**6⇒4**	**inf⇒4**

- *loop* among nodes B,C and E until the cost threshold is reached
- AGAIN counting to infinity

Counting to infinity: remedies

- Use of Counters/Timers (Hold down)
  - If for $T_{invalid}$ no info from the first hop to a specific destination, destination is no longer valid (not advertised in the DVs, DVs from other nodes skipped)
  - after T_{flush} the route is flushed
  - $T_{invalid}$ - T_{flush} must be set so that the new information propagate within the whole network
  - Invalid routes advertised with distance = infinity
  - Nodes receiving an invalid route set the route as invalid themselves

Counting to infinity: remedies

- *Triggered Update*
  - Explicit advertisement of the changes in the topology
  - Speed up convergence
  - Prompt failures discovery

Link State Routing Protocols

Link State Routing Protocols

- Each node knows neighboring nodes and the relative costs to reach them
- Each node sends to ALL the other nodes such information (*flooding*) through *Link State Packet* (*LSP*)
- All the nodes keep a LSP data base and a complete map of the network topology (graph)
- On the complete graph shortest paths are computed using Dijkstra

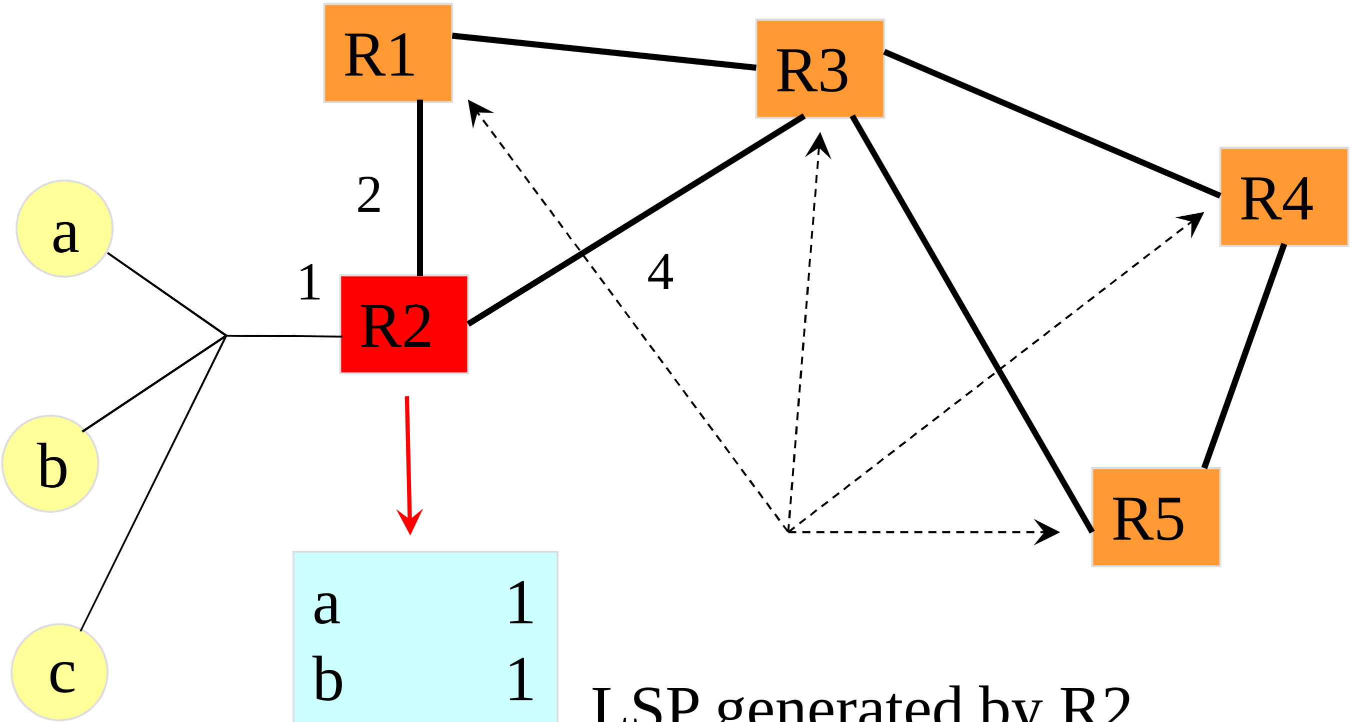
Link State: PROs

- Flexibility and Optimality in the path definition (complete map of the network topology)
- LSP information is not sent periodically but only when something changes
- All the nodes get promptly aware of any change in the network topology

Link State: CONs

- Signaling protocol required to keep the topological information (*Hello*)
- *flooding* needed
- LSP must be acknowledged
- Difficult to implement

Link State: example



a	1
b	1
c	1
R2	0
R1	2
R3	4

LSP generated by R2

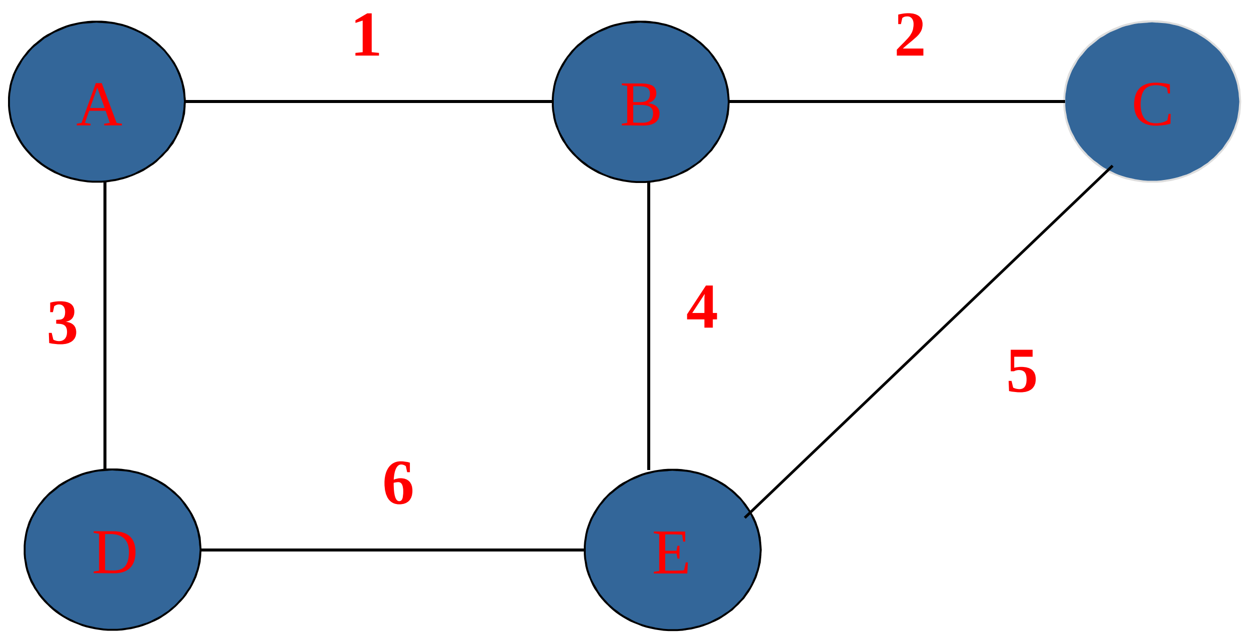
Flooding

- Each entering packet is transmitted through all the interfaces except the incoming one
- possible *loops* and consequent traffic congestion

- Sequence number (SN) + SN database in each node to avoid multiple transmissions of the same packet
- *Hop counter* (same as TTL in IP)

Example

- Each node owns a LSP data base



Example

- The LSP data base represents the network topology

From	To	Link	Cost	Sequence Number
A	B	1	1	1
A	D	3	1	1
B	A	1	1	1
B	C	2	1	1
B	E	4	1	1
C	B	2	1	1
C	E	5	1	1
D	A	3	1	1
D	E	6	1	1
E	B	4	1	1
E	C	5	1	1
E	D	6	1	1

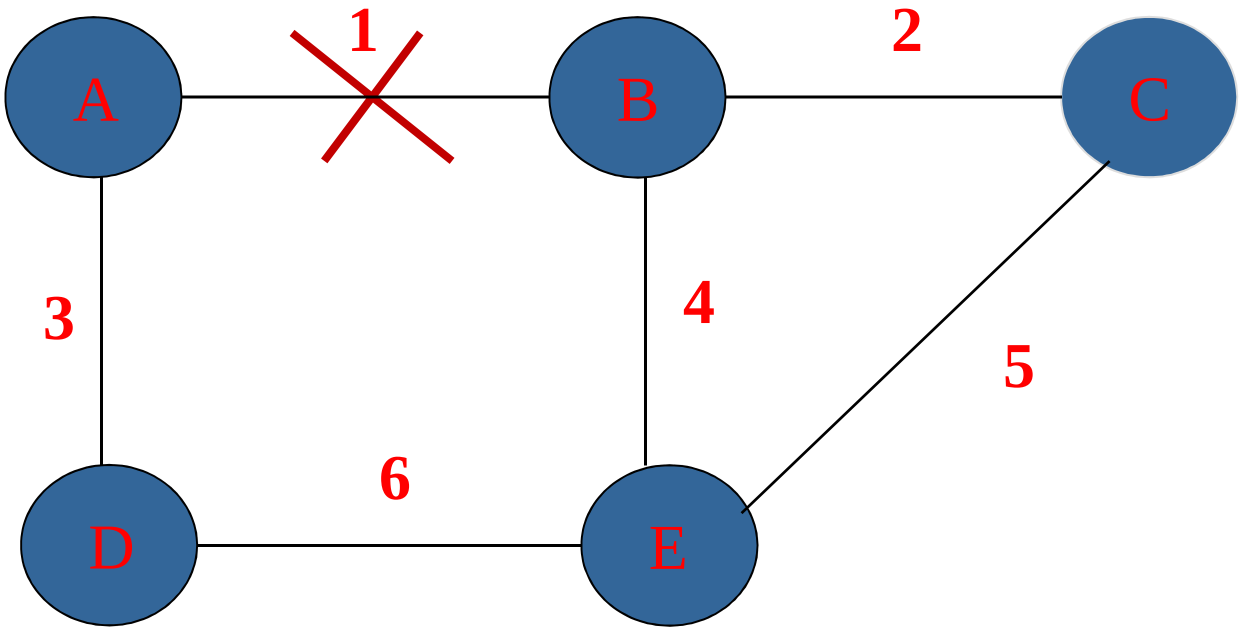
- Each node can easily calculate the shortest path to all the other nodes in the network

Upon reception of an LSP

- If the LSP has not been received yet or if the SN is greater than the one already stored:
  - Store the new LSP
  - *Apply the flooding*
- If the LSP has the same SN of the one stored
  - Do nothing
- If the LSP is older than the one stored
  - Transmit the newer one to the sender

Link State: Example

- The routing protocol must update the network topology whenever something changes



- *link* 1 failure is detected by nodes A and B which send an LS update packet on *links* 3, 2 and 4
  - node A: From A, To B, Link 1, Cost=inf, Number=2
  - node B: From B, To A, link 1, Cost= inf, Number=2

Link State: Example

- The messages are received by nodes D,E and C which update their data base and flood on the local links
- The new data base after *flooding is:*

From	To	Link	Cost	Sequence Number
A	B	1	1⇒inf	1⇒2
A	D	3	1	1
B	A	1	1⇒inf	1⇒2
B	C	2	1	1
B	E	4	1	1
C	B	2	1	1
C	E	5	1	1
D	A	3	1	1
D	E	6	1	1
E	B	4	1	1
E	C	5	1	1
E	D	6	1	1