Internet, Principes et Protocoles (IPP)

2. Routing and Network Layer

Table of Contents

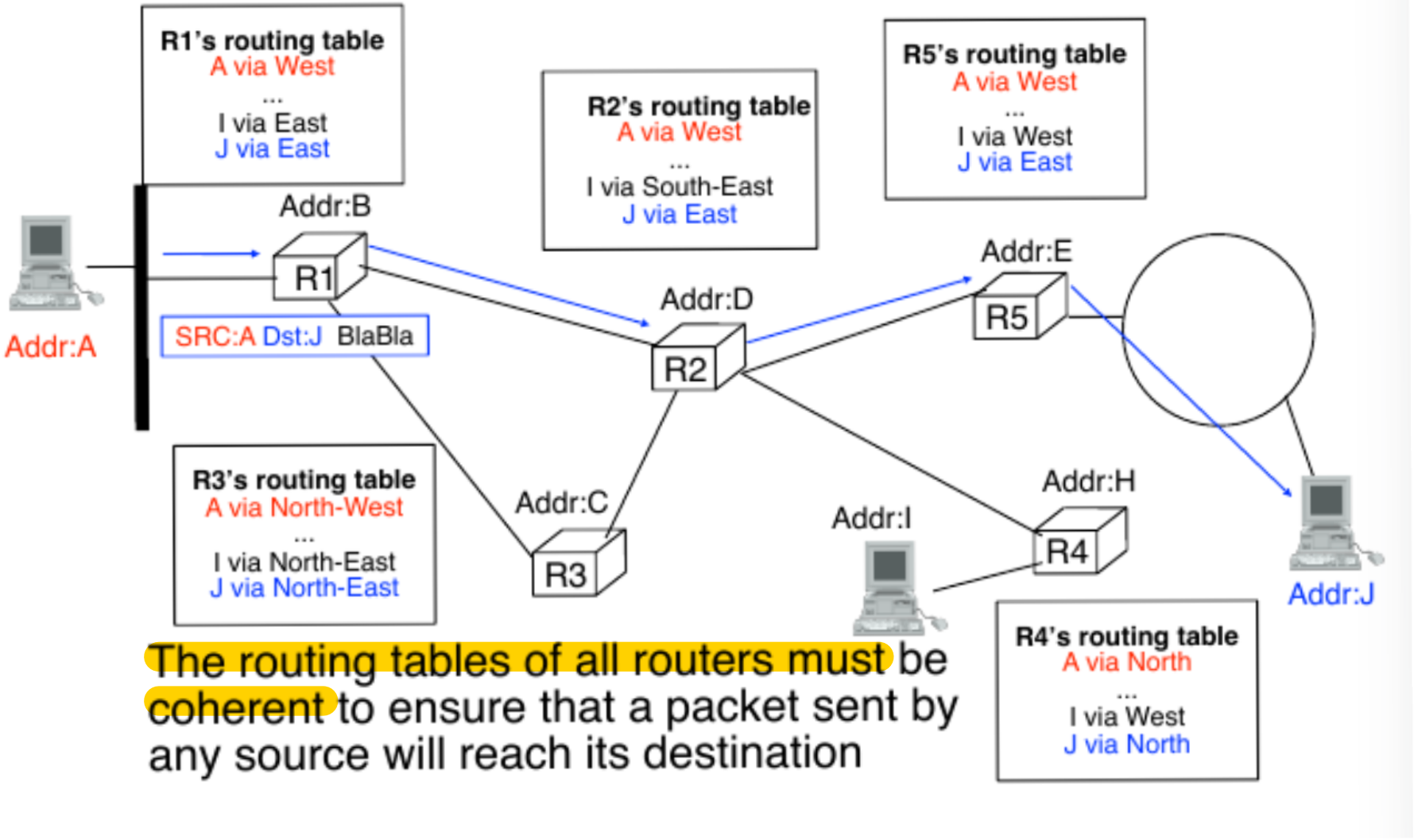
Network Layer

- Forwarding information from source to destination, through multiple routers if necessary.
- 2 possible organisations:
  - Datagrams; most often used for a connectionless service
  - Virtual Circuits; most often used for a connection oriented service

Datagram Transmision mode

- The information is divided into packets, containing:
  - Source
  - Destination
  - Payload
- The router forwards the packet after looking up the destination in the routing table. Each router in the route takes a routing decision.
- Protocol examples: IP (IPv4 & IPv6), CLNP, IPX

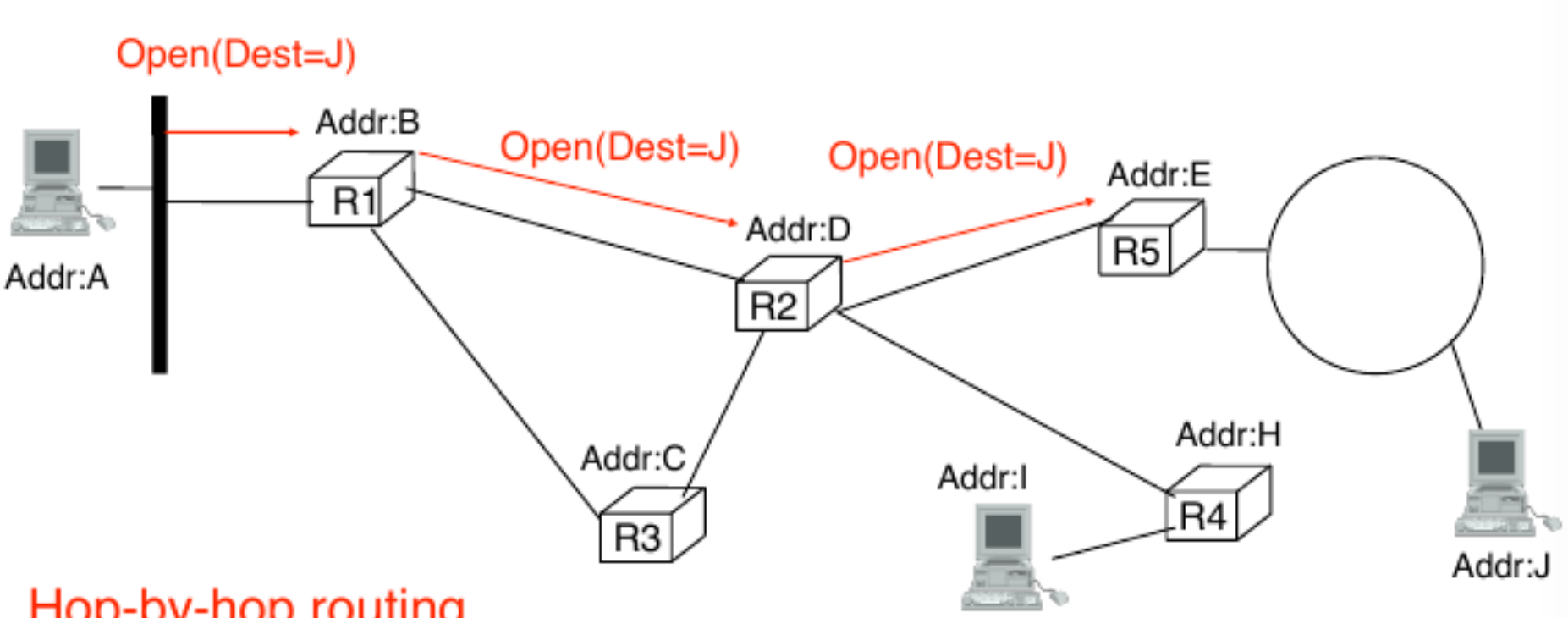
Datagram Transmision mode



Virtual Circuit Mode

- Goals
  - Keep Forwarding on routers simple. Looking up the routing table every time is costly.
- Create a virtual circuit the links source and destination through the network. All packets will follow the same path.
- Ex: ATM, X.25, MPLS, gMPLS

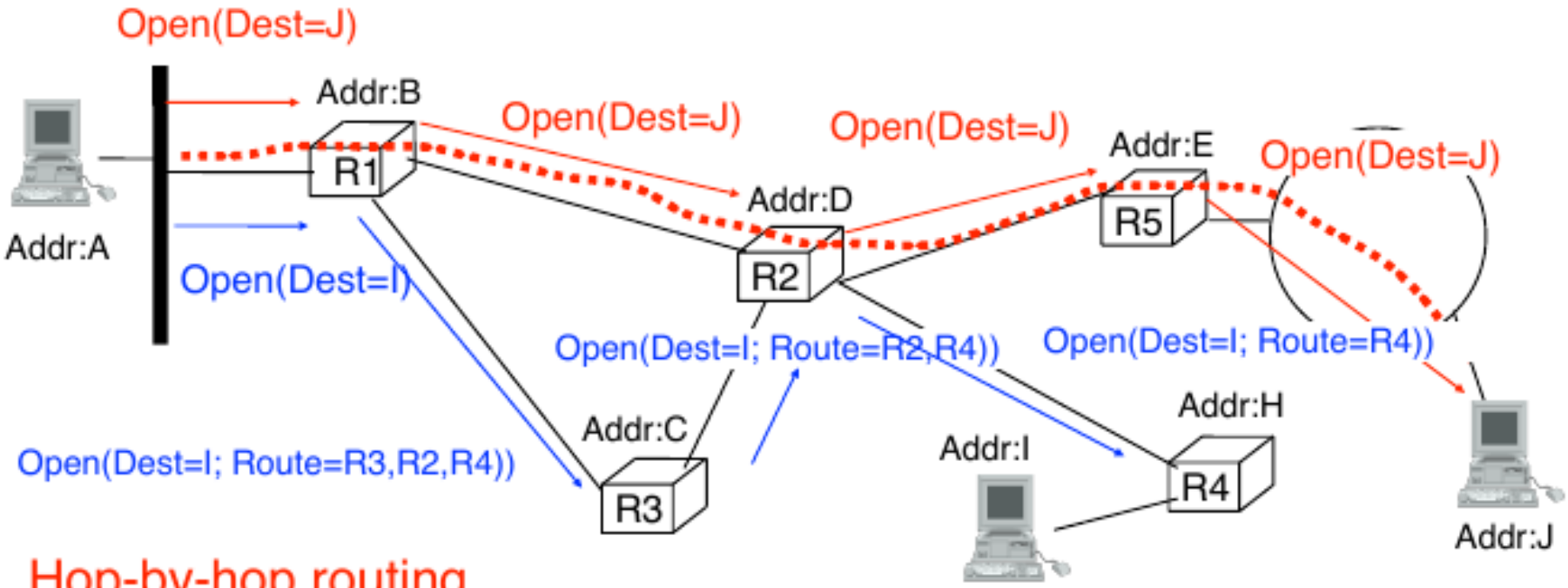
VC establishment



Hop-by-hop routing

Each router consults its routing table to forward vc establishment

VC establishment



Hop-by-hop routing

Each router consults its routing table to forward vc establishment

Source routing/ explicit routing

Source (or first hop router) indicates in vc establishment packet the path to be followed

Routing and Forwarding

Network Layer: transport packets from source to destination

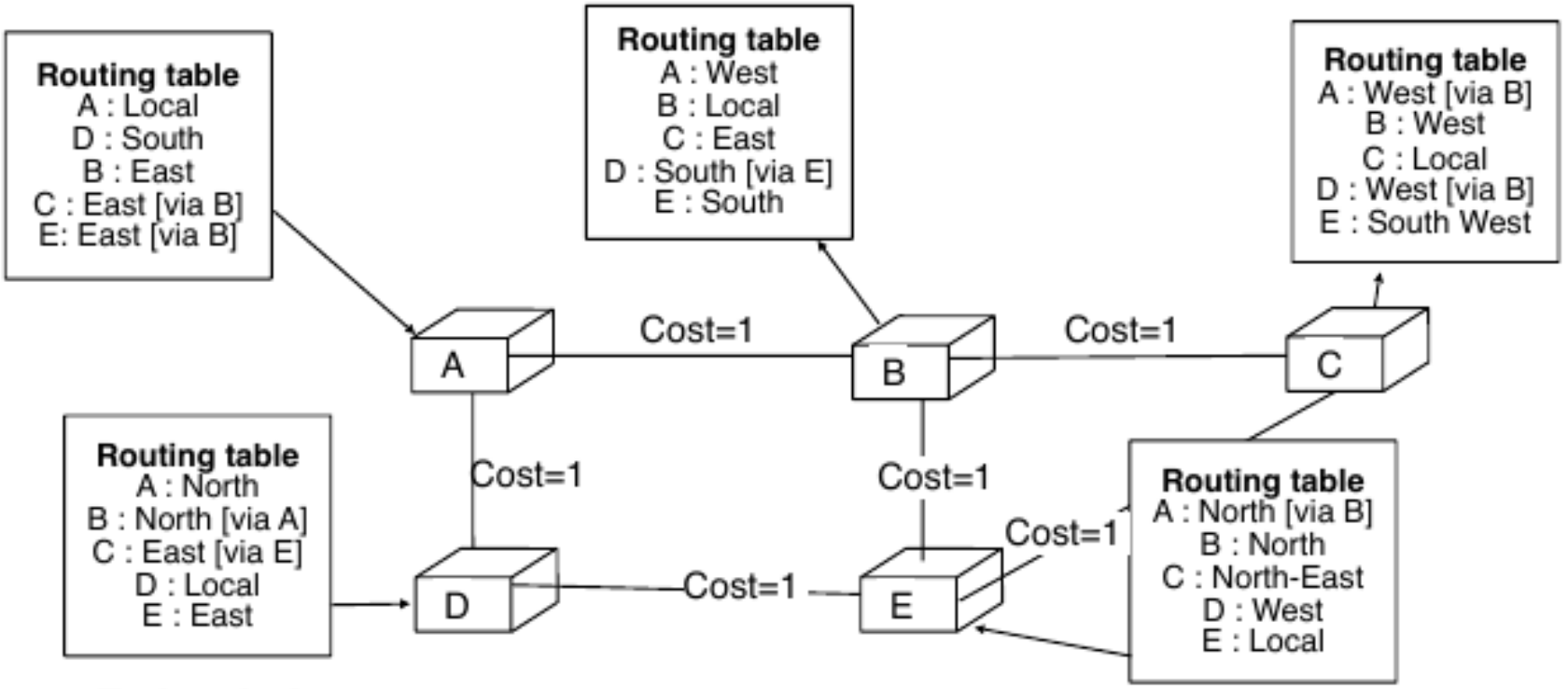
Two mechanisms for this:

- Static routing/Forwarding: sending the packet to the next hop, based on the routing table
- Dynamic Routing: algorithm that distributes to all the routers the information that allows them to build their routing tables

Static Routing

- Principle
  - Network manager or network management station computes all routing tables and downloads them on all routers
  - How to compute routing tables ?
    - shortest path algorithms
    - more complex algorithms to provide load balancing or traffic engineering
  - Advantages of static routing
    - Easy to use in a small network
    - routing tables can be optimised
  - Drawbacks of static routing
    - does not adapt dynamically to network load
    - how to deal with link and router failures ?

Static Routing



Principle

Associate a weight/cost to each link

Each router chooses the lowest cost path

How to ensure that the routing tables of all routers are coherent ?

Table of Contents

Dynamic Routing

Principle
- routers exchange messages and use a distributed algorithm to build their routing tables
- used in almost all networks

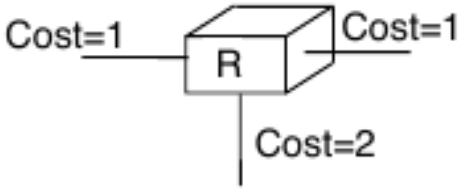
Advantages
- can easily adapt routing tables to events

Drawbacks
- more complex to implement than static routing

Most common distributed routing methods
- Distance vector routing
- Link state routing

Distance Vector routing

Basic principles
- Configuration of each router
  - Cost of each link



When it boots, a router only knows itself

Each router sends periodically to all its neighbours a vector that contains for each destination that it knows

1. Destination address
2. Distance between transmitting router and destination
   - distance vector is a summary of the router's routing table

Each router will update its routing table based on the information received from its neighbours

Distance Vector Routing

Routing table *maintained by router*

For each destination d inside routing table

R[d].cost = total cost of shortest path to reach d

R[d].link = outgoing link used to reach d via shortest path

Distance vector *sent to neighbours*

For each destination d

V[d].cost = total cost of shortest path to reach d

```
Every N seconds:
 Vector=null;
 for each destination=d in R[]
 {
  Vector=Vector+Pair(d,R[d].cost);
 }
 for each interface
 {
  Send(Vector);
 }
```

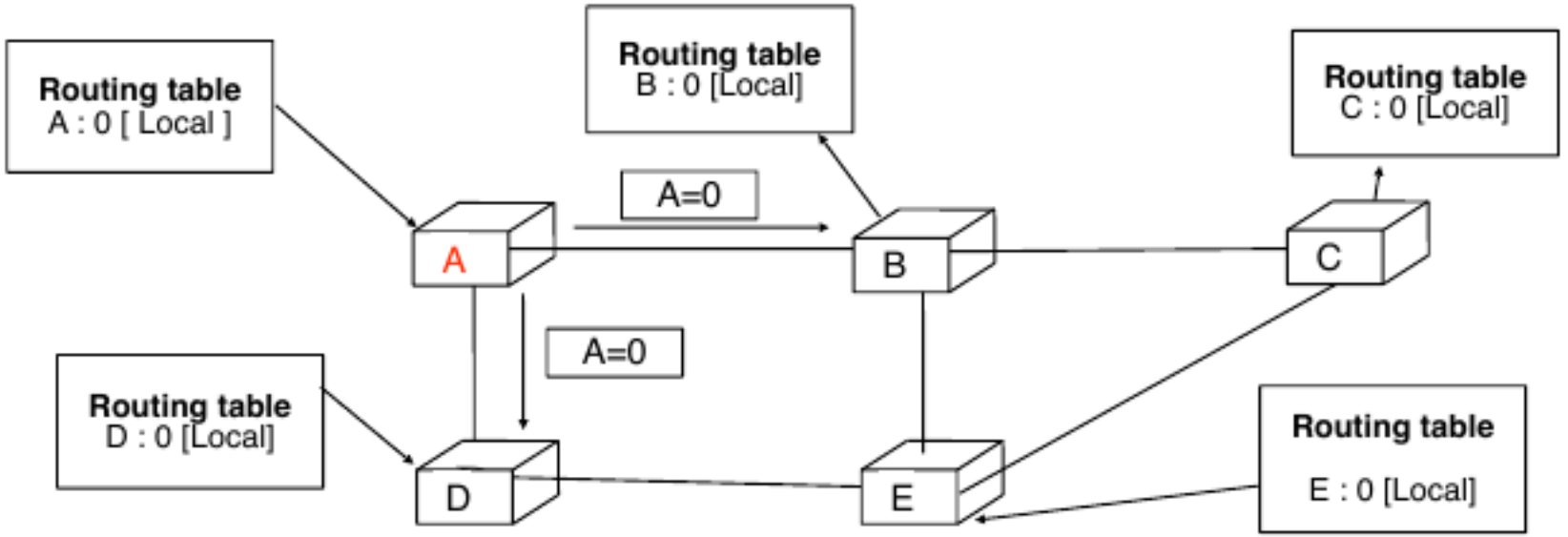
Distance Vector Routing

Processing of received distance vectors

```
Received(Vector V[],link l)
{ /* received vector from link l */
for each destination=d in V[]
{
  if (d isin R[])
  { if ((V[d].cost+l.cost) < R[d].cost)
     { /* shorter path */
       R[d].cost=V[d].cost+l.cost;
       R[d].link=l;
     }
   }
   else
   { /* new route */
     R[d].cost=V[d].cost+l.cost;
     R[d].link=l;
   }
}
```

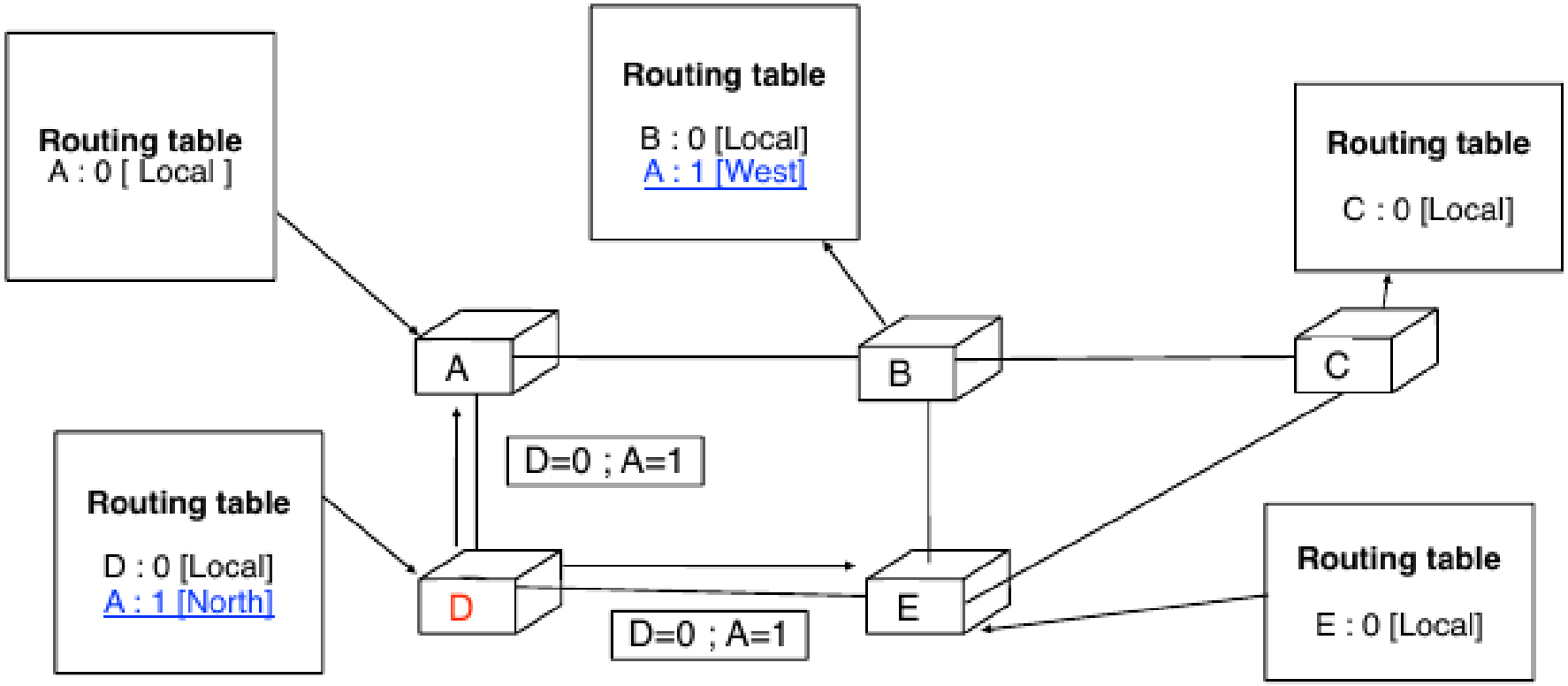
Example

All links have a unit cost

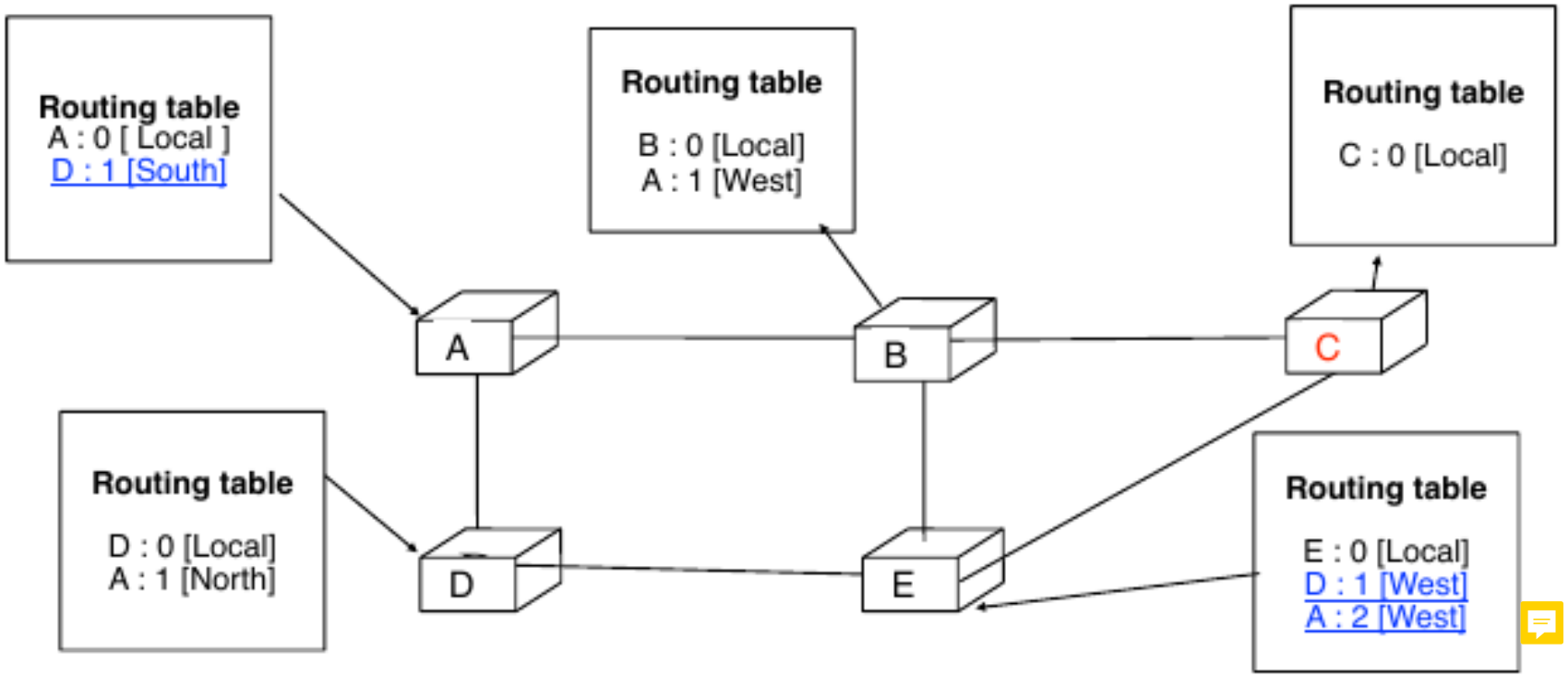


When a router boots, it only knows itself. Its distance vector thus contains only its address

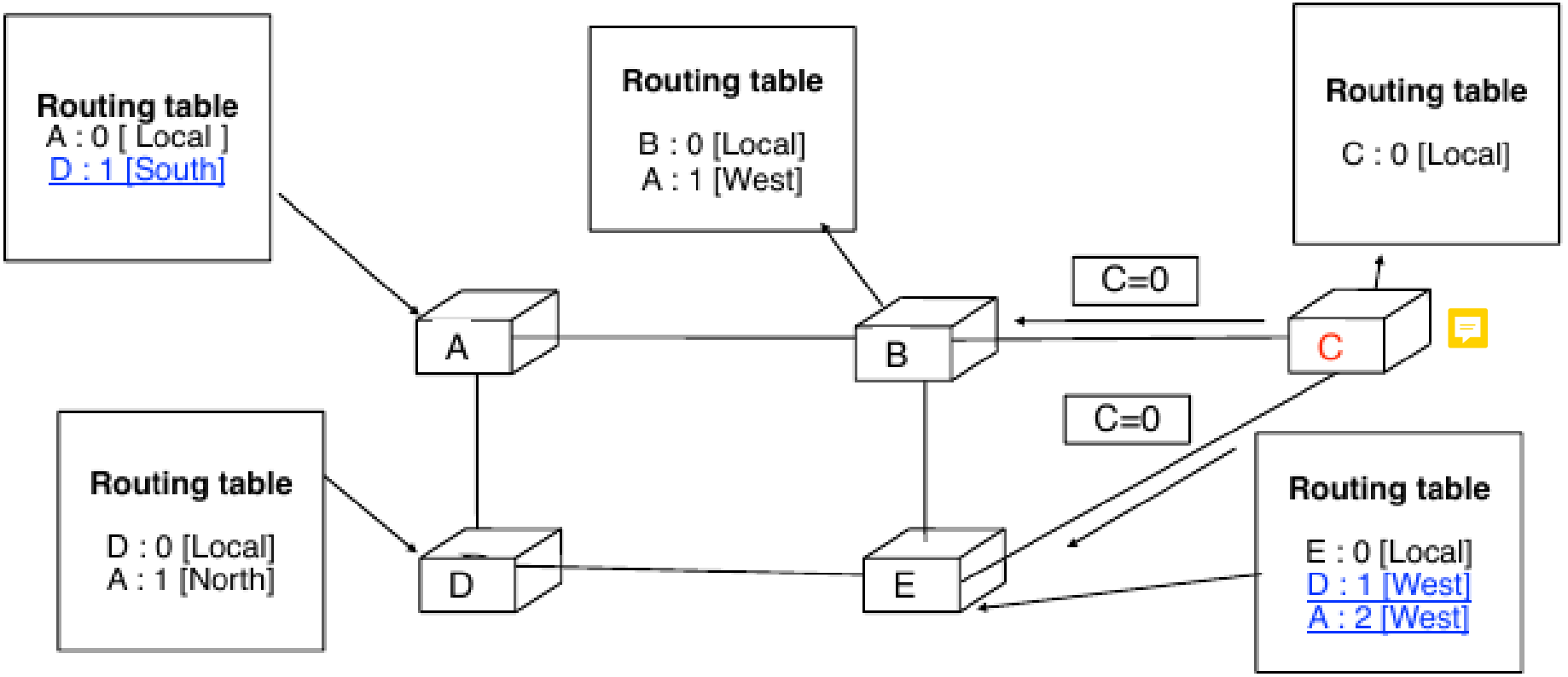
Example


Routing table
A : 0 [Local]
Routing table
B : 0 [Local]
A : 1 [West]
Routing table
C : 0 [Local]
A
B
C
D=0 ; A=1
Routing table
D : 0 [Local]
A : 1 [North]
D
E
D=0 ; A=1
Routing table
E : 0 [Local]

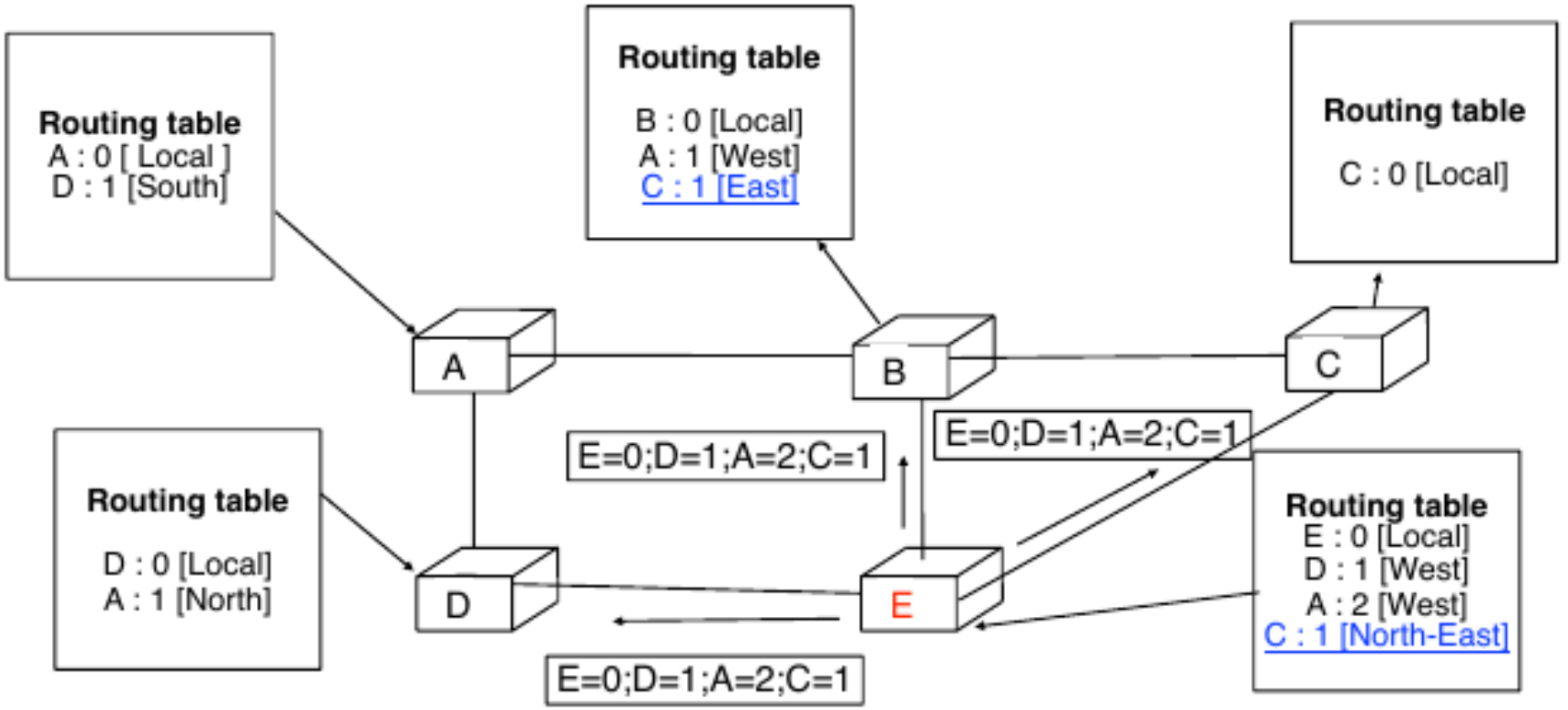
Example


Routing table
A : 0 [Local]
D : 1 [South]
Routing table
B : 0 [Local]
A : 1 [West]
Routing table
C : 0 [Local]
A
B
C
Routing table
D : 0 [Local]
A : 1 [North]
D
E
Routing table
E : 0 [Local]
D : 1 [West]
A : 2 [West]

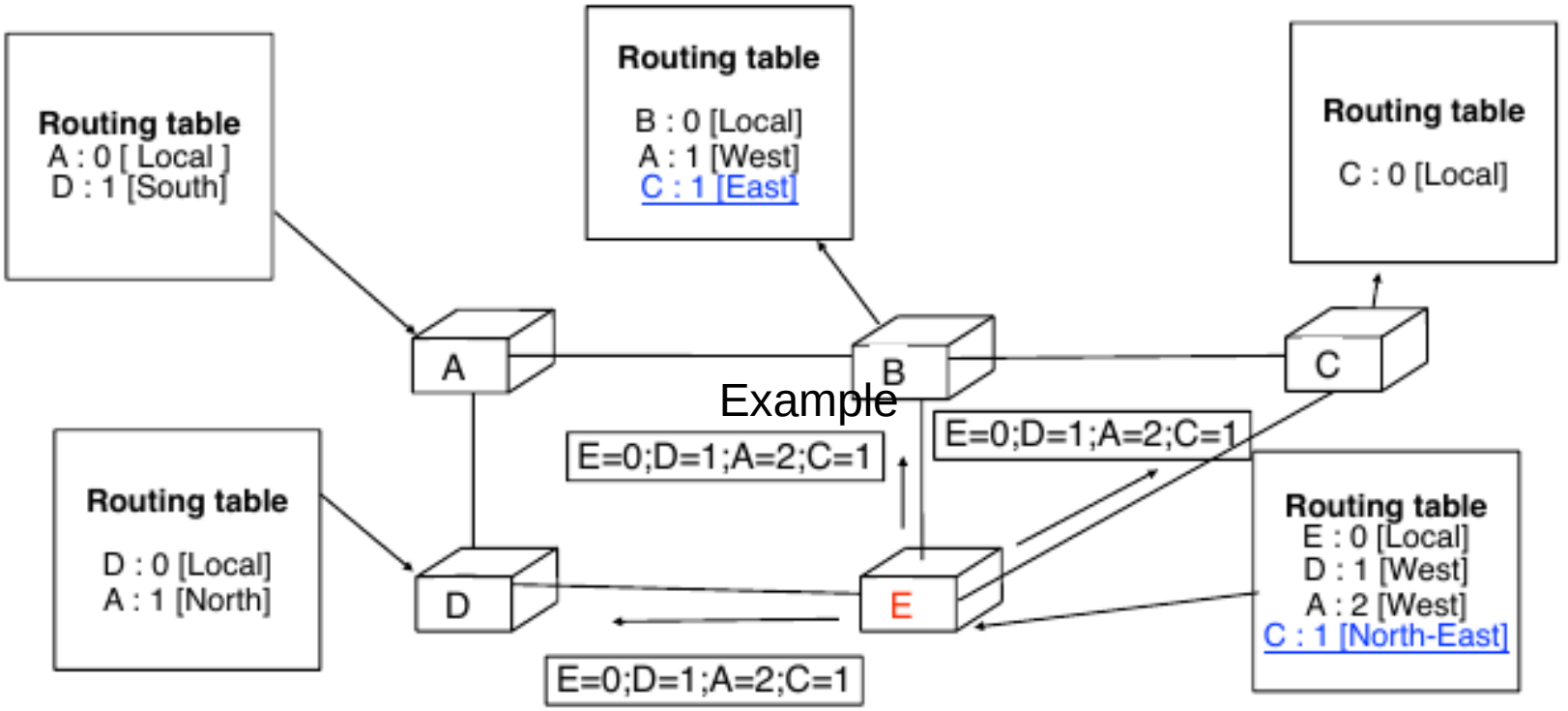
Example


Routing table
A : 0 [Local]
D : 1 [South]
Routing table
B : 0 [Local]
A : 1 [West]
Routing table
C : 0 [Local]
C=0
A
B
C
C=0
Routing table
D : 0 [Local]
A : 1 [North]
D
E
Routing table
E : 0 [Local]
D : 1 [West]
A : 2 [West]

Example


Routing table
A : 0 [Local]
D : 1 [South]
Routing table
B : 0 [Local]
A : 1 [West]
C : 1 [East]
Routing table
C : 0 [Local]
A
B
C
D
E
E=0;D=1;A=2;C=1
E=0;D=1;A=2;C=1
Routing table
D : 0 [Local]
A : 1 [North]
Routing table
E : 0 [Local]
D : 1 [West]
A : 2 [West]
C : 1 [North-East]
E=0;D=1;A=2;C=1

Example



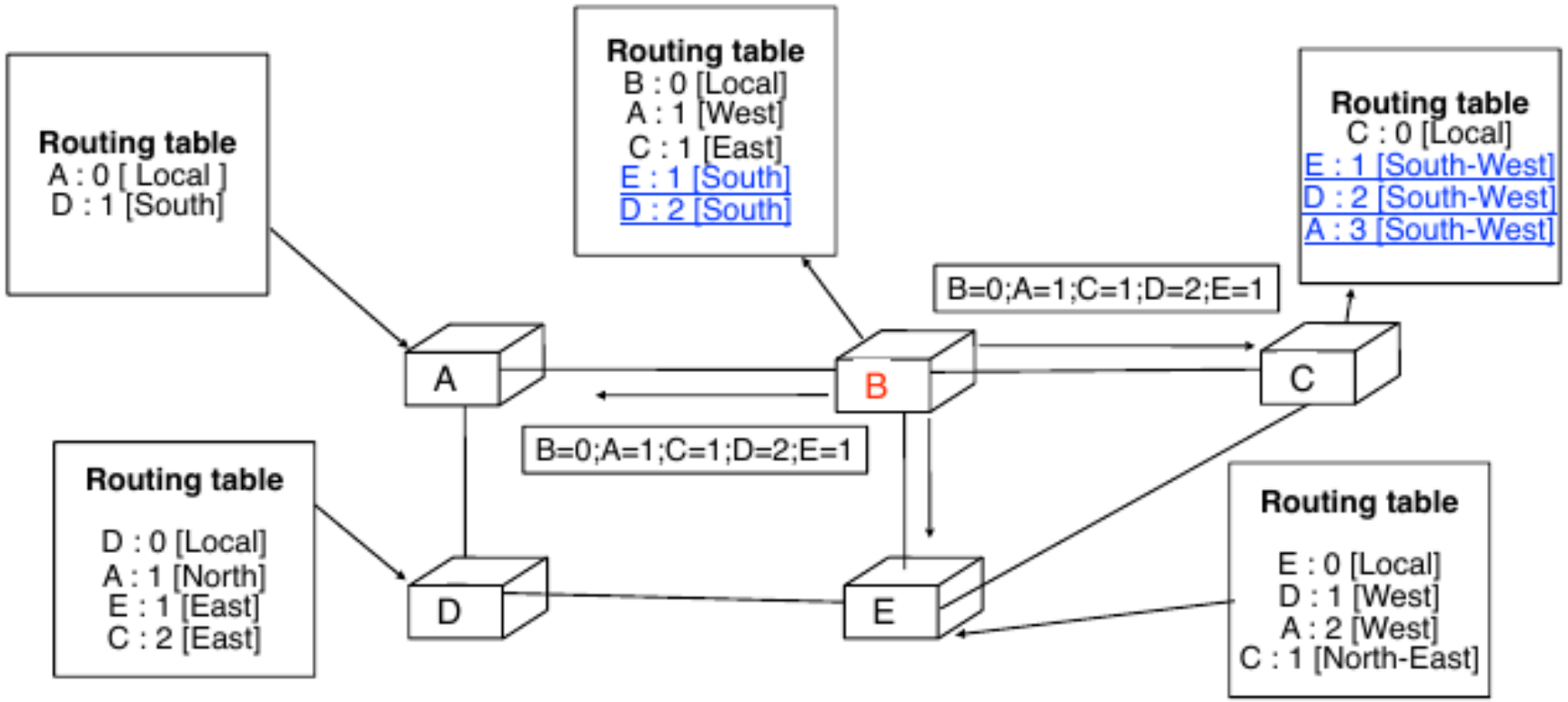
Reception of distance vector on B
- New route to reach E and D, longer route for A

Reception of distance vector on C
- New routes to reach D, A and E

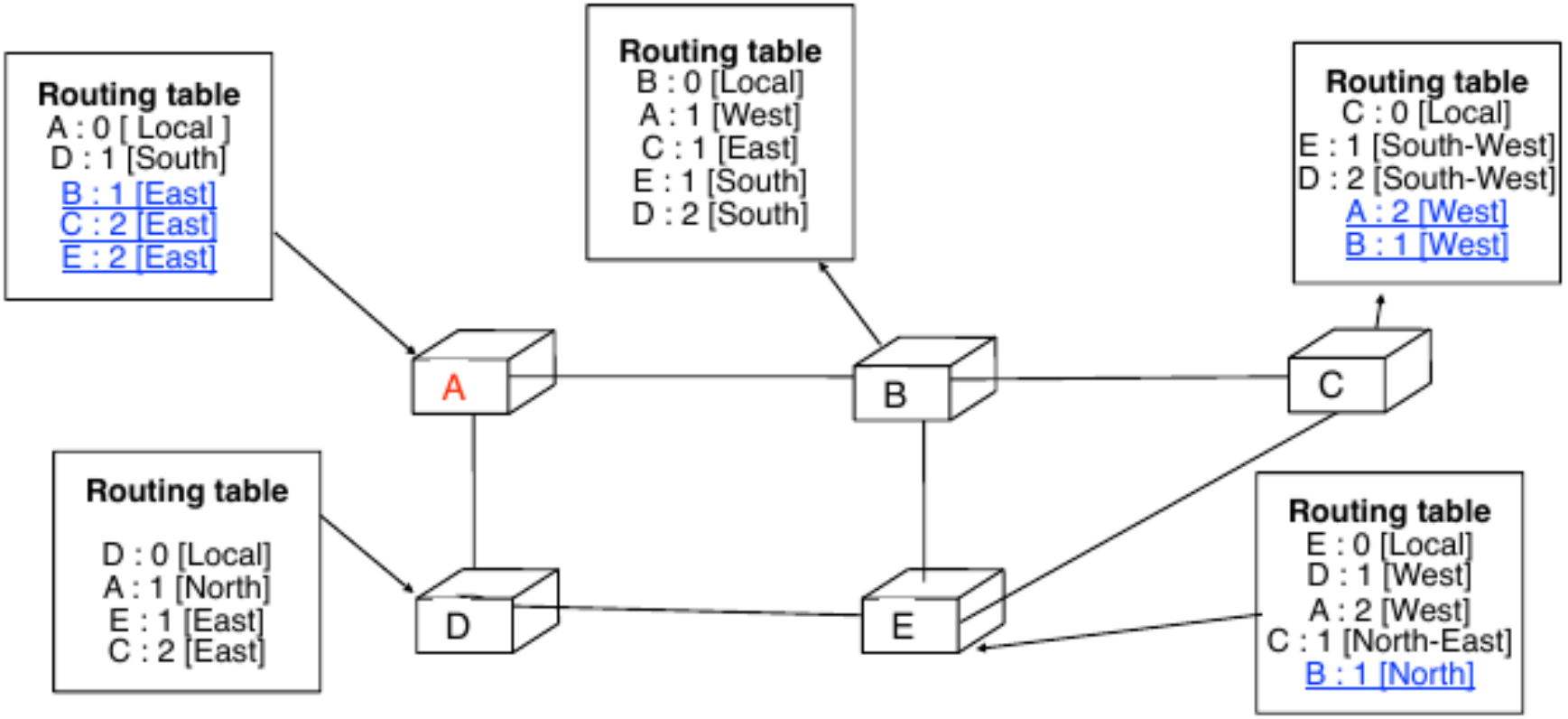
Reception of distance vector on D
- New routes to reach E and C, longer route for A

Example

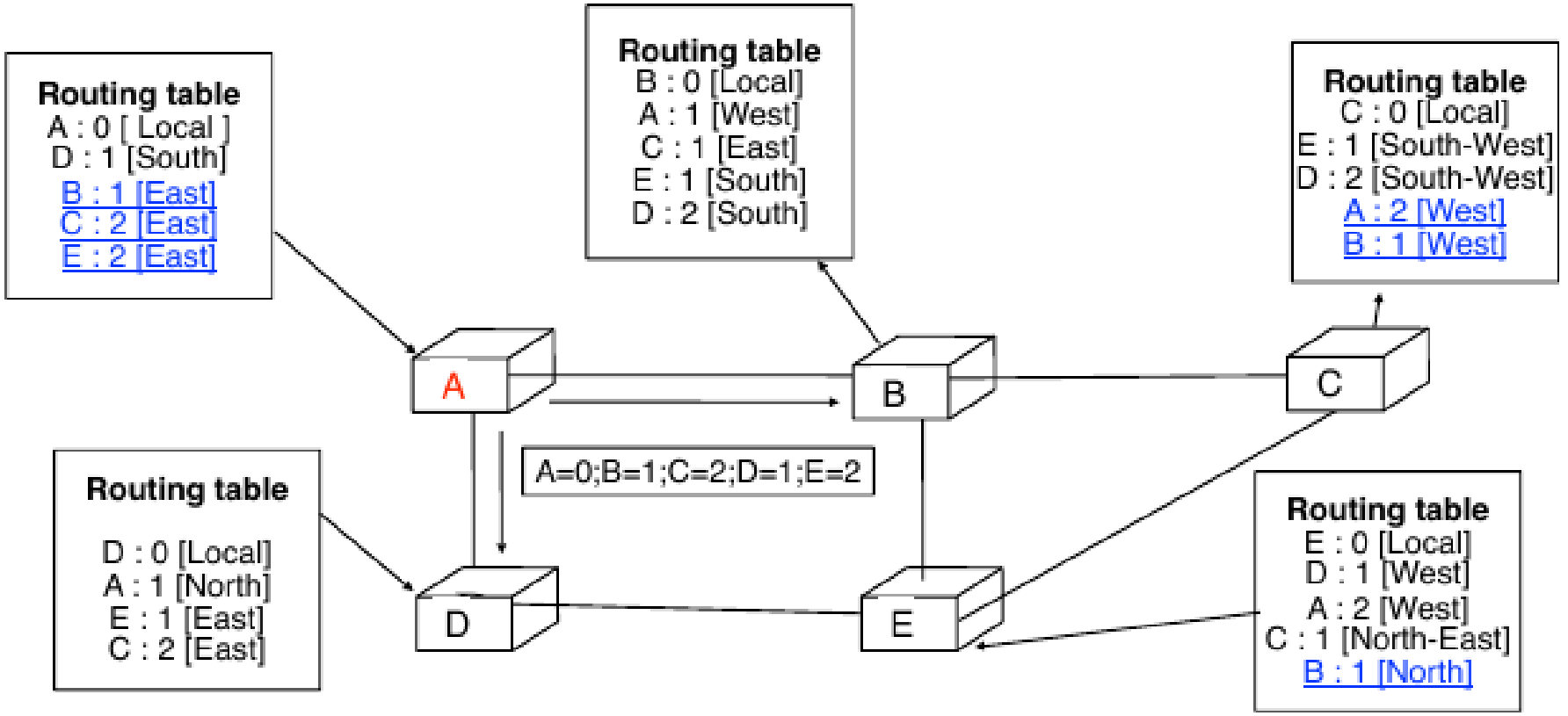
B is the first to send its vector



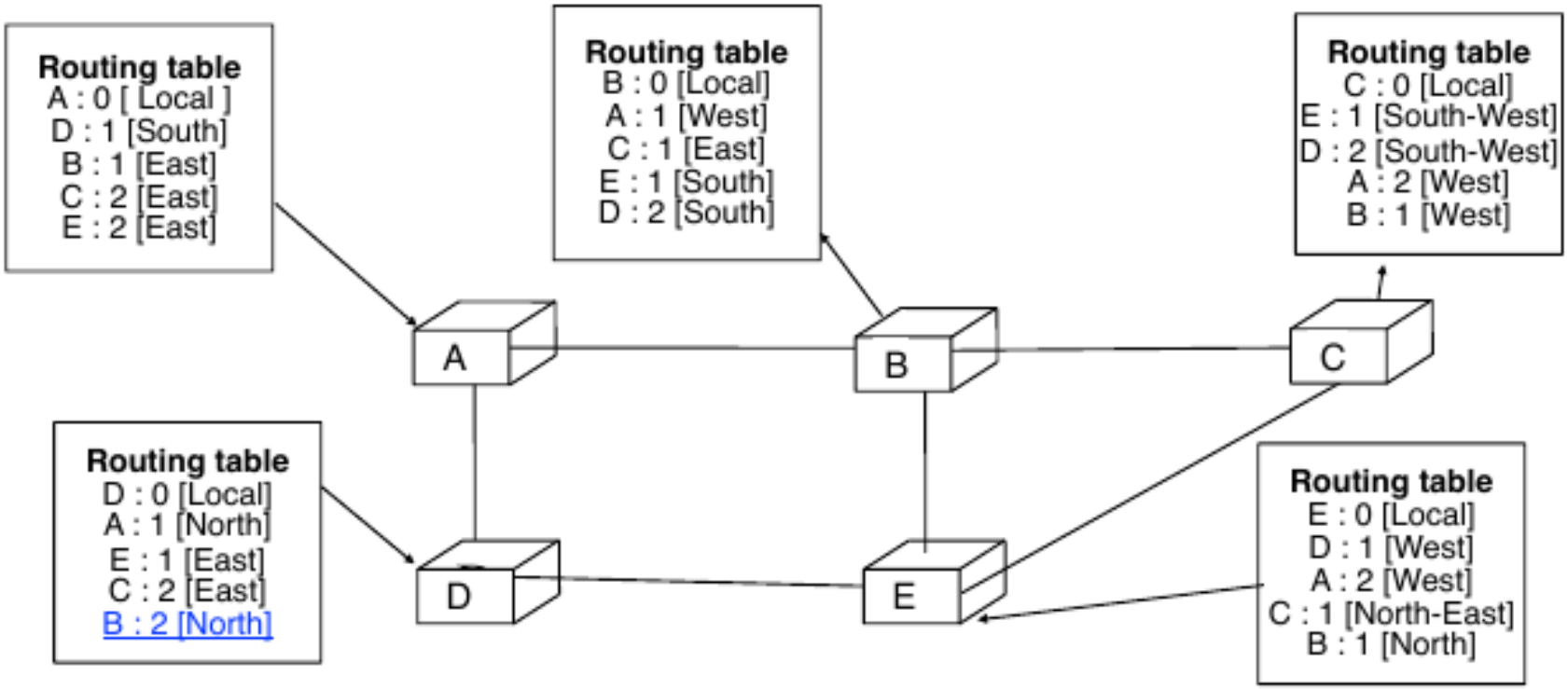
Example


Routing table
A : 0 [Local]
D : 1 [South]
B : 1 [East]
C : 2 [East]
E : 2 [East]
Routing table
B : 0 [Local]
A : 1 [West]
C : 1 [East]
E : 1 [South]
D : 2 [South]
Routing table
C : 0 [Local]
E : 1 [South-West]
D : 2 [South-West]
A : 2 [West]
B : 1 [West]
A
B
C
Routing table
D : 0 [Local]
A : 1 [North]
E : 1 [East]
C : 2 [East]
D
E
Routing table
E : 0 [Local]
D : 1 [West]
A : 2 [West]
C : 1 [North-East]
B : 1 [North]

Example


Routing table
A : 0 [Local]
D : 1 [South]
B : 1 [East]
C : 2 [East]
E : 2 [East]
Routing table
B : 0 [Local]
A : 1 [West]
C : 1 [East]
E : 1 [South]
D : 2 [South]
Routing table
C : 0 [Local]
E : 1 [South-West]
D : 2 [South-West]
A : 2 [West]
B : 1 [West]
A
B
C
A=0;B=1;C=2;D=1;E=2
Routing table
D : 0 [Local]
A : 1 [North]
E : 1 [East]
C : 2 [East]
D
E
Routing table
E : 0 [Local]
D : 1 [West]
A : 2 [West]
C : 1 [North-East]
B : 1 [North]

Example



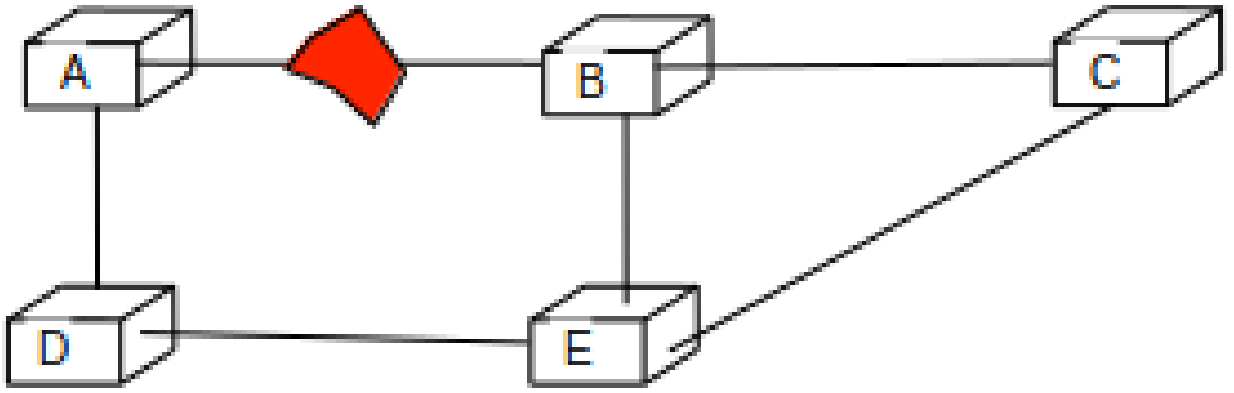
All routers know how to reach all other routers

Routing tables are stable

If a distance vector is sent by one router, it will not cause any change to the routing table of other routers in the network

Distance Vector Failures

How to deal with link failures ?



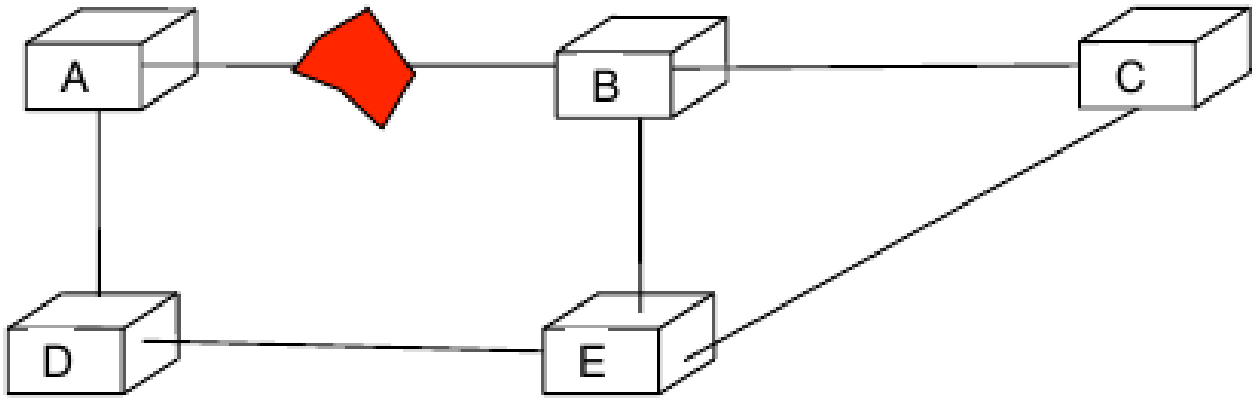
Two problems must be solved for failures

How to detect that the link has failed ?

How to indicate to all routers that they should update their routing table since the paths that use link A-B do not work anymore ?

Distance Vector Failures

How to deal with link failures (detect them, update the tables)?

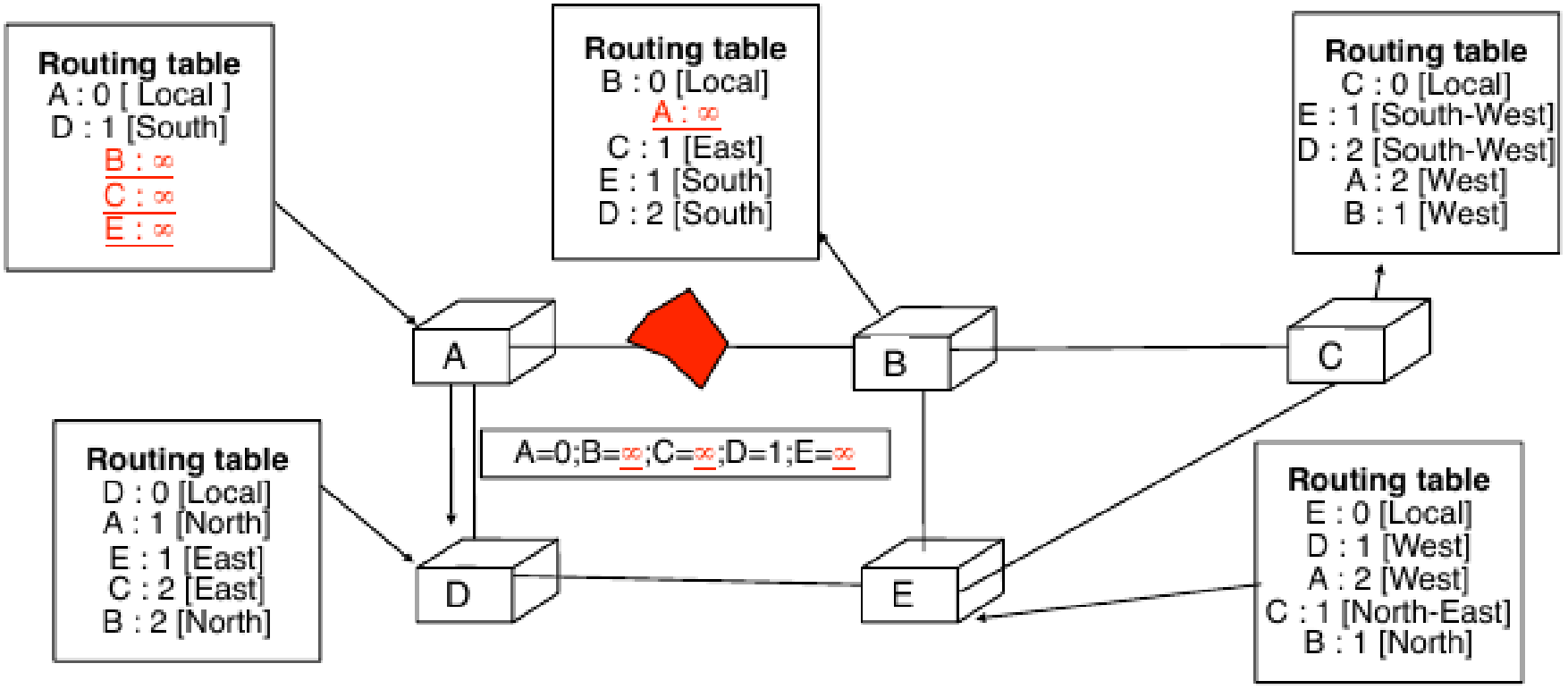


ask each router to regularly send its distance vector (e.g. every 30 seconds)

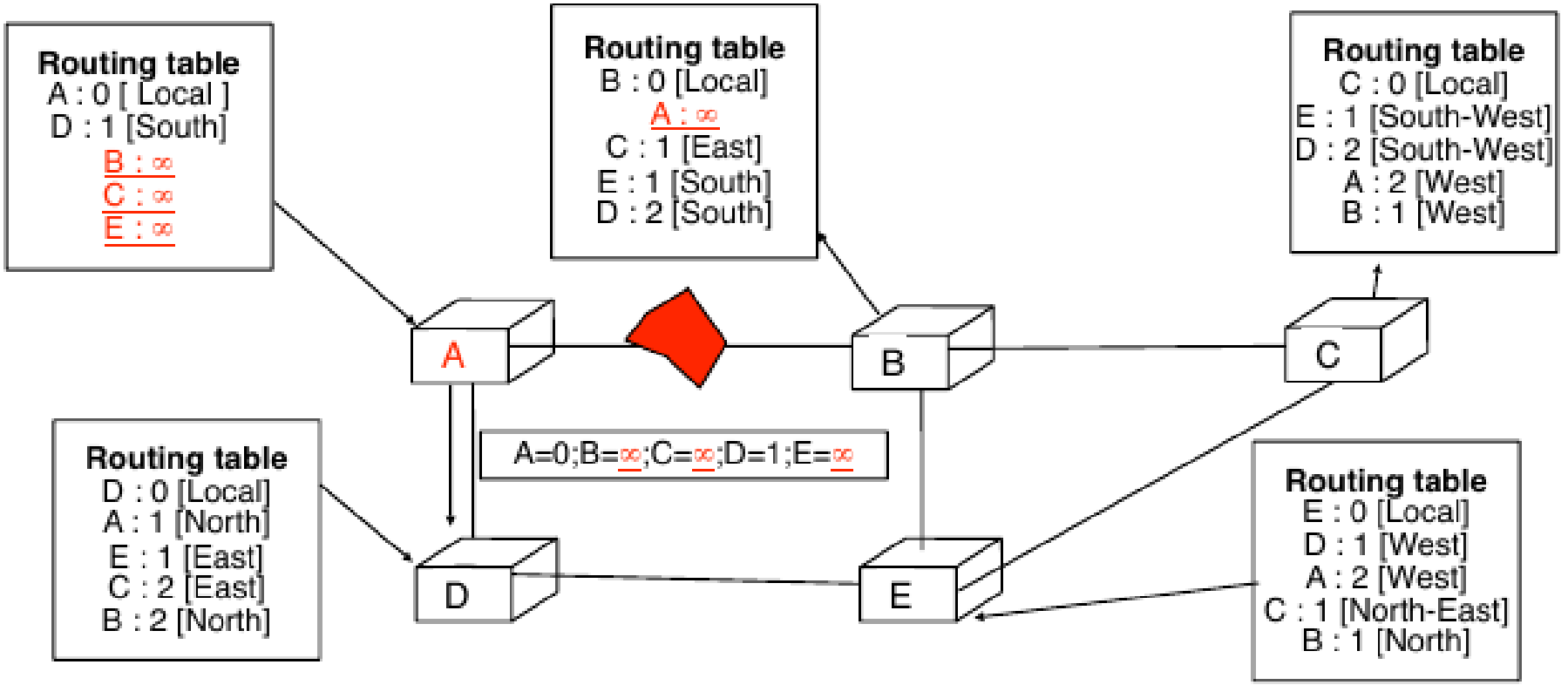
If a router does not receive a refresh for a route in a distance vector from one of its neighbours during some time (e.g. 90 seconds), it assumes that the route is not available anymore

Distance Vector Failures

All routes that use a failed link are marked with an infinite cost

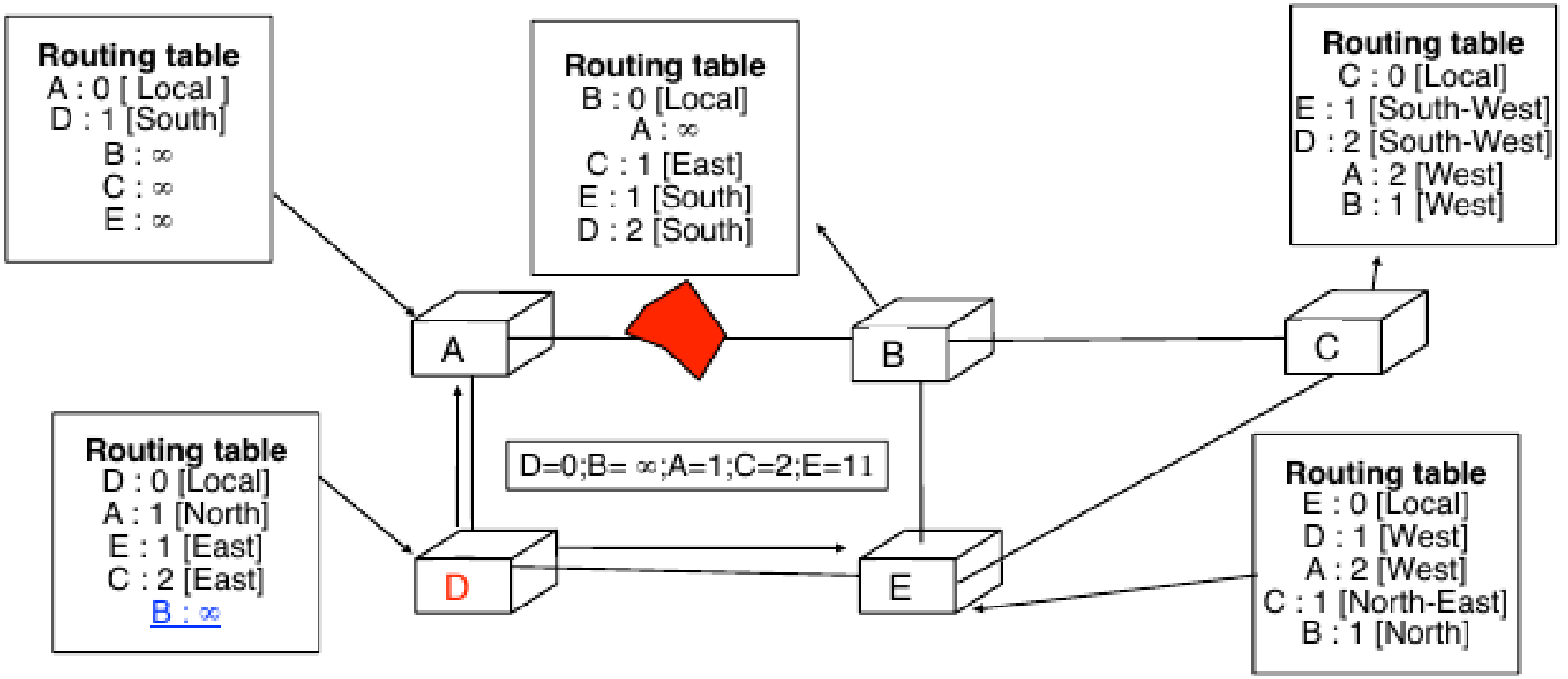


Distance Vector Failures

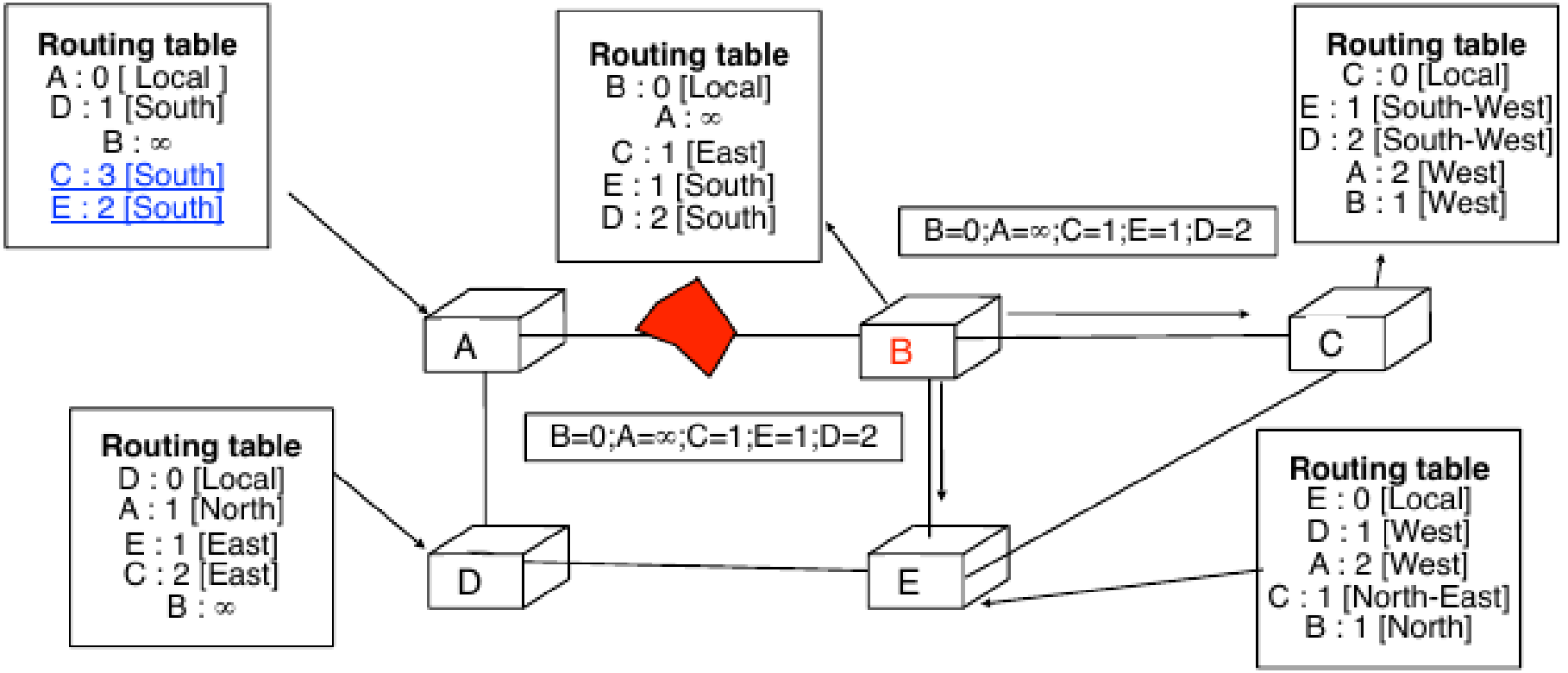


D must remove from its routing tables all the routes that it learned from its North link and are announced now with an ∞ cost

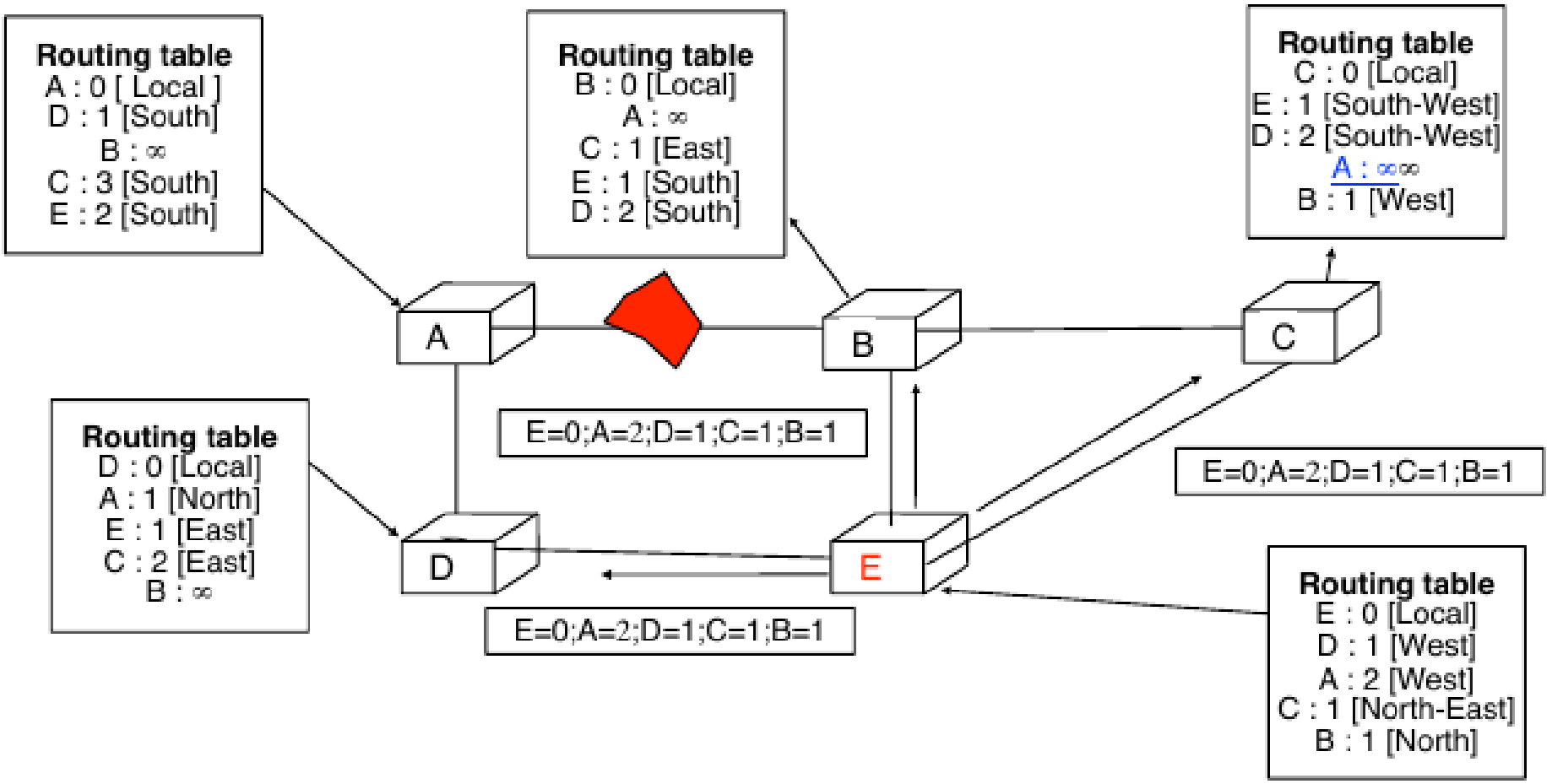
Distance Vector Failures



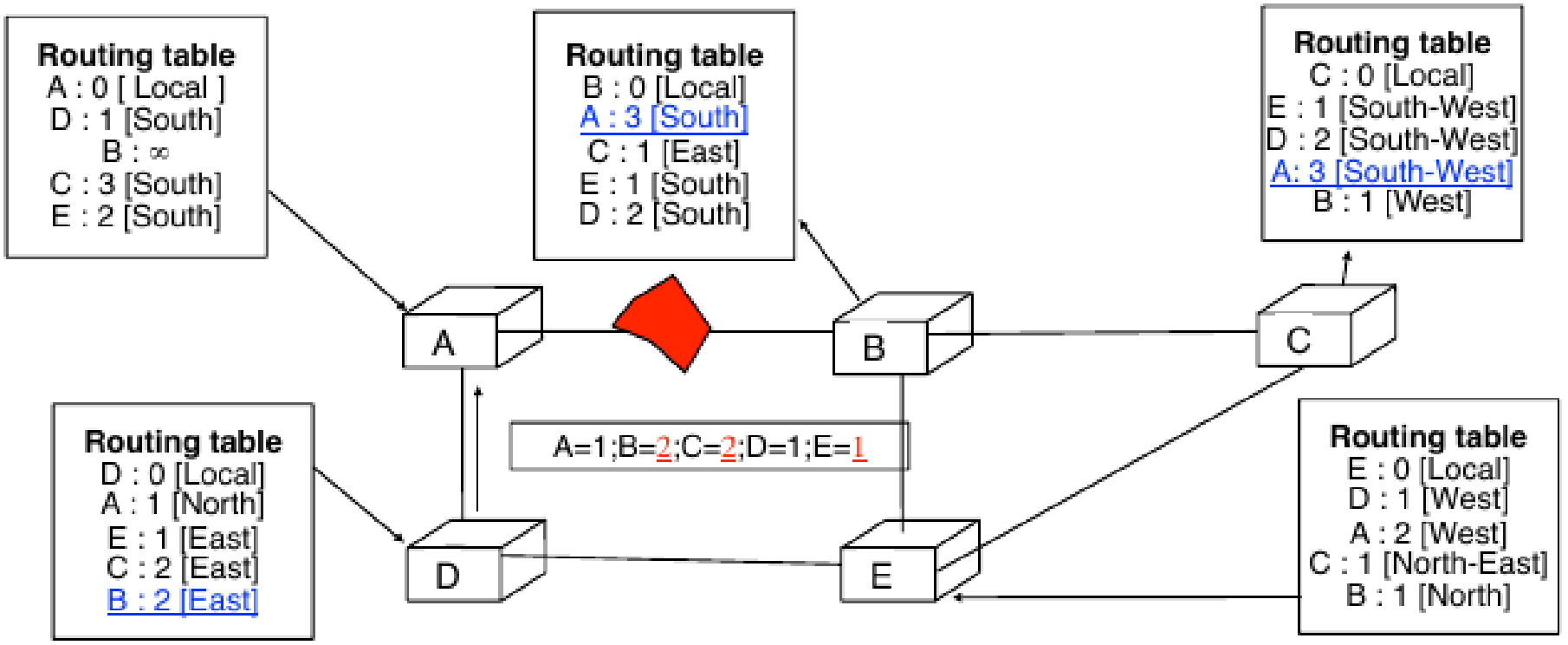
Distance Vector Failures



Distance Vector Failures

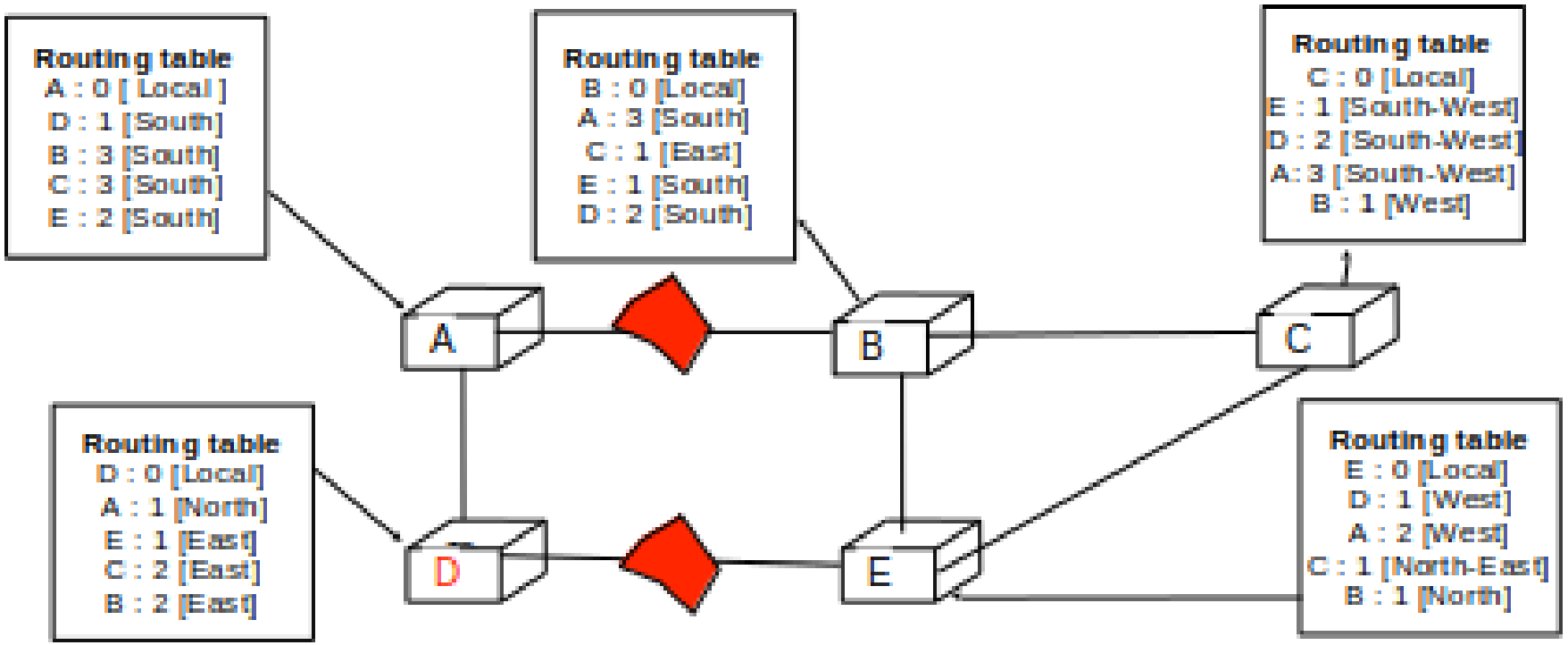


Distance Vector Failures



Failure has been recovered, all routers are now reachable again from any router

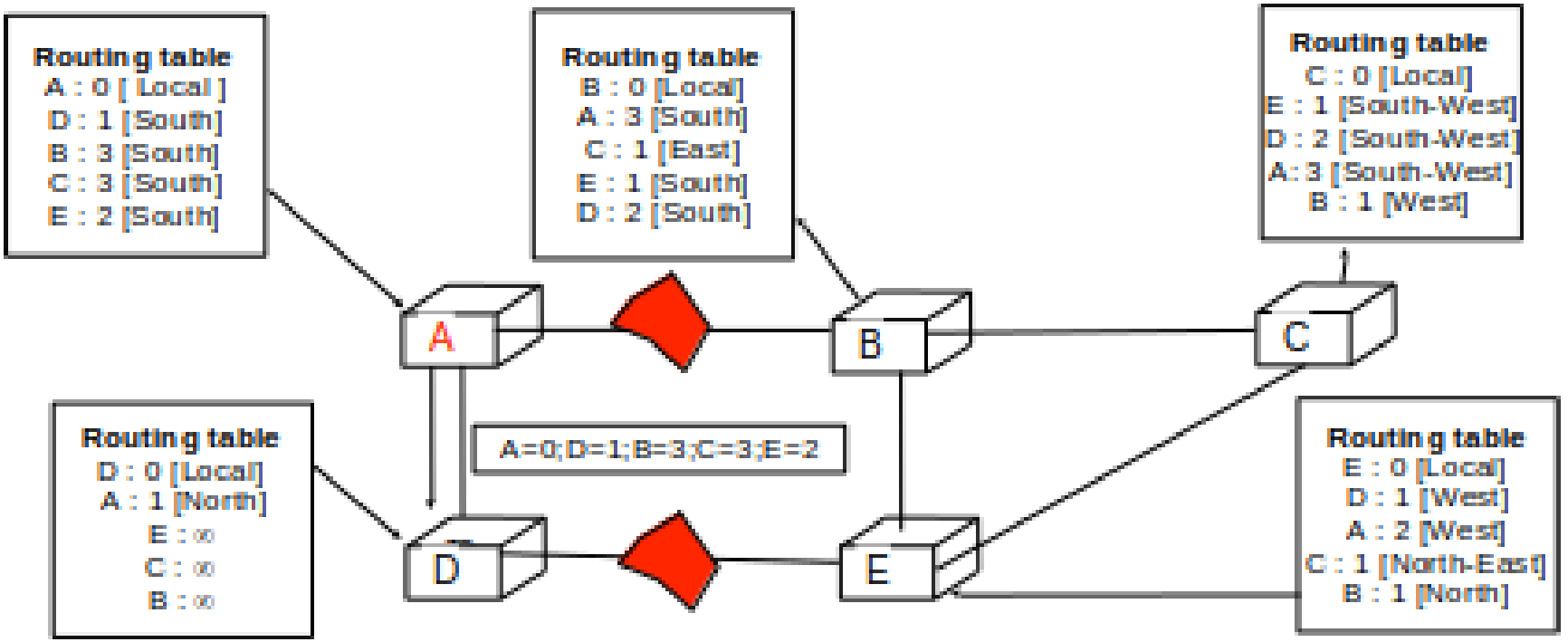
Second link failure



D detects the failure

If it is the first to send its distance vector, failure is detected and router A updates its routing table

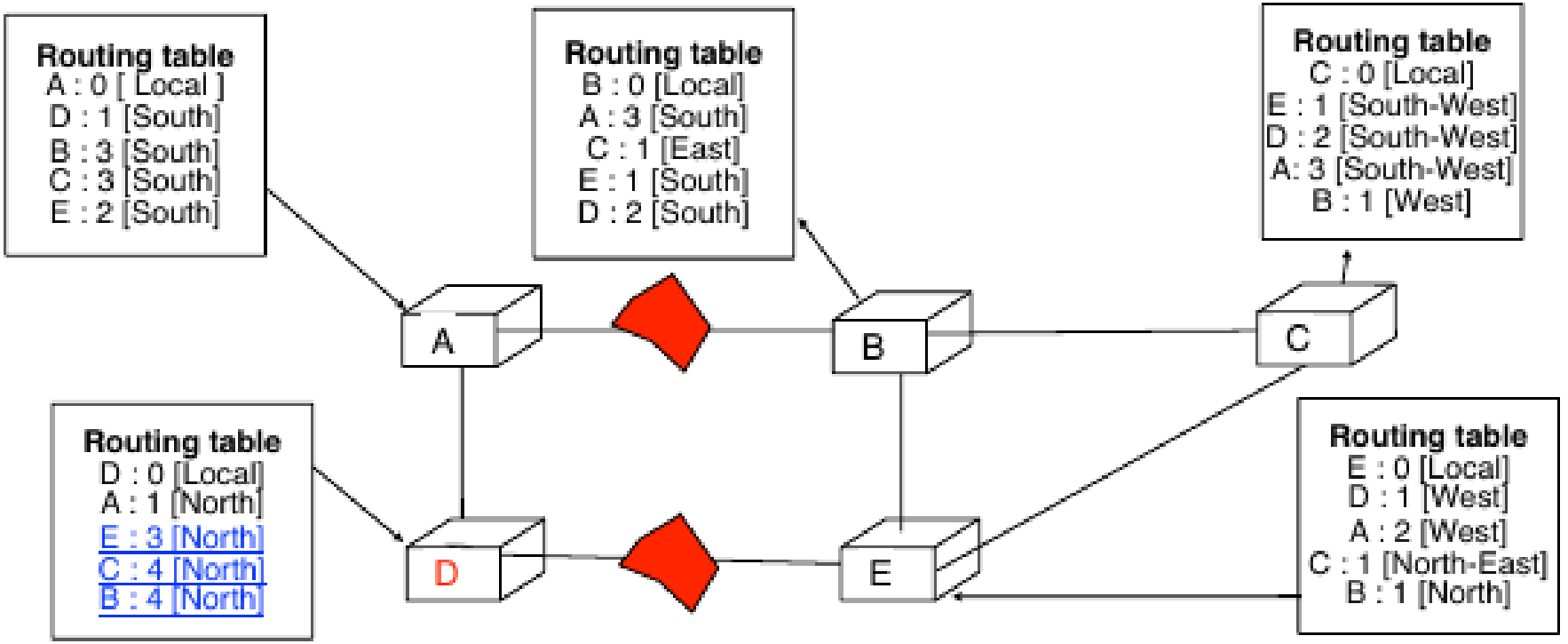
Second Link Failure



But if A sends its distance vector before having received or processed D's updated distance vector ...

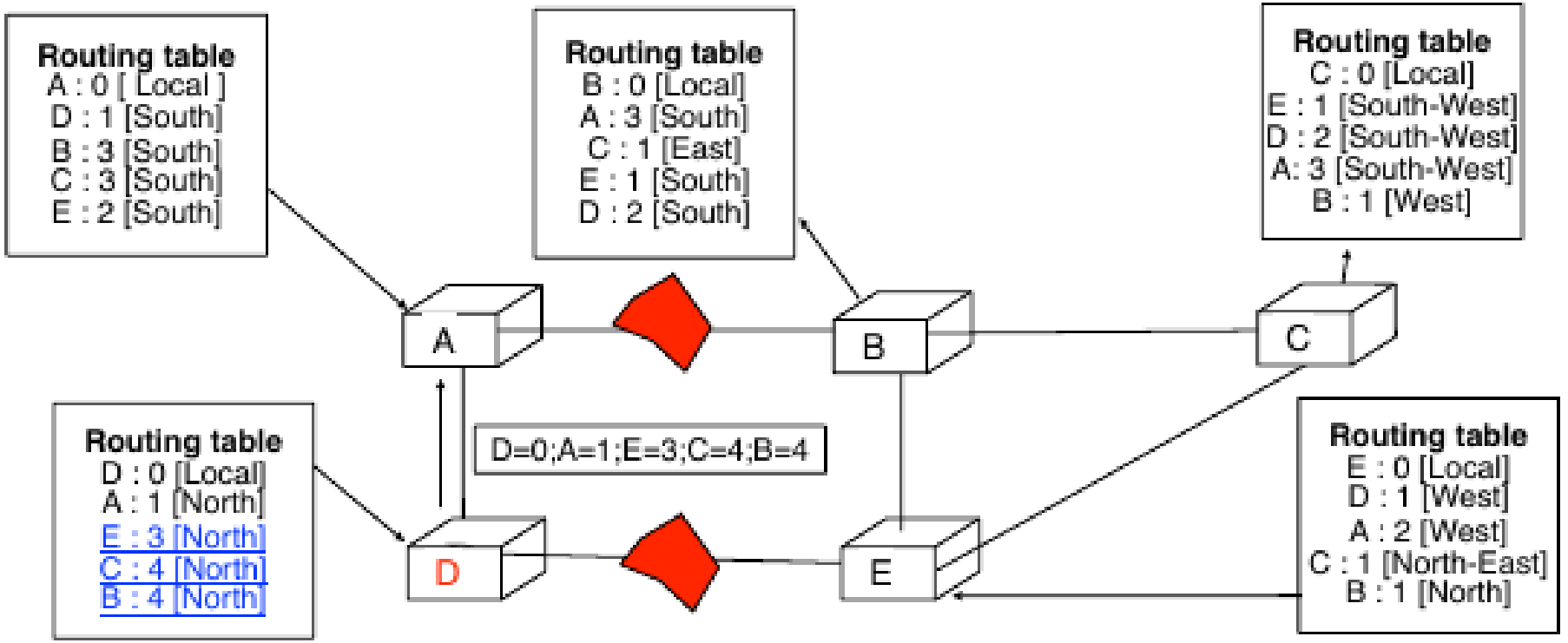
Second Link Failure

Upon reception of A's vector, D updates its routing table

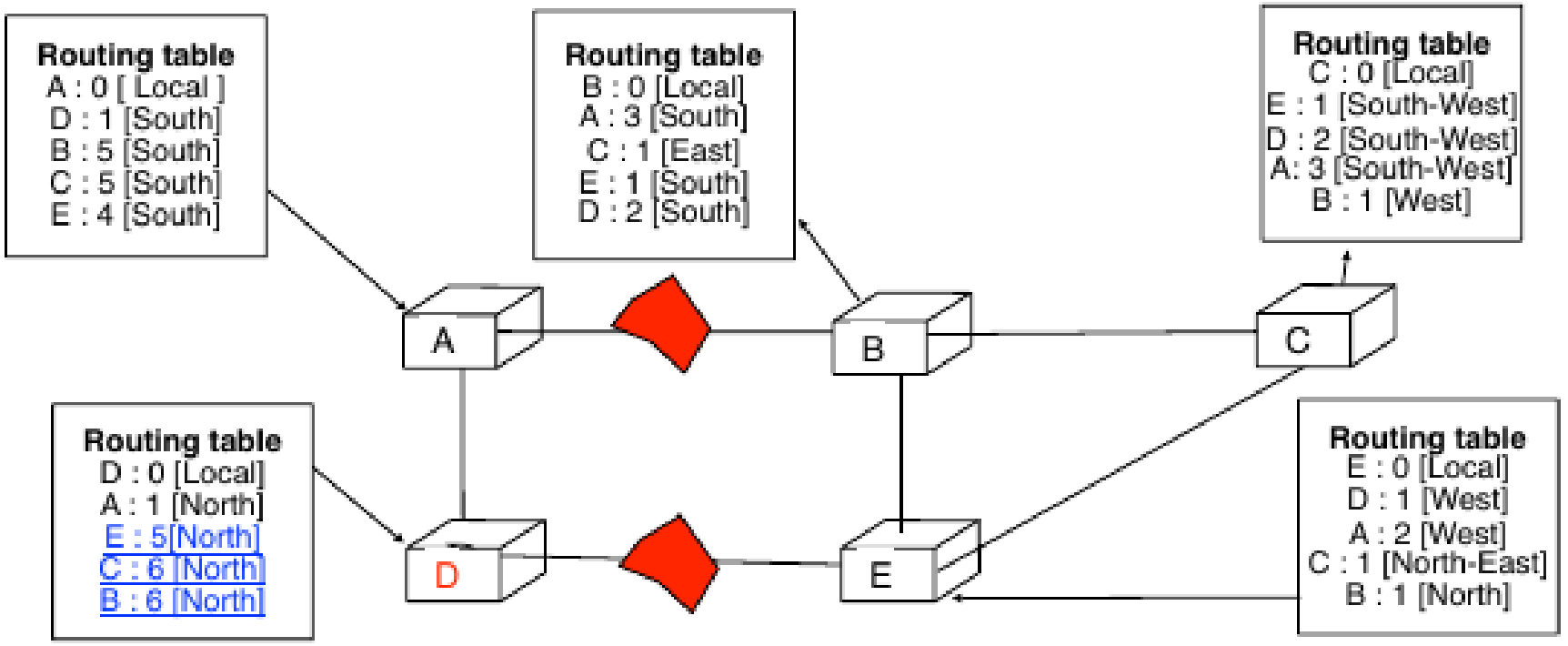


Second Link Failure

Upon reception of A's vector, D updates its routing table



Second Link Failure



This problem is called counting to infinity

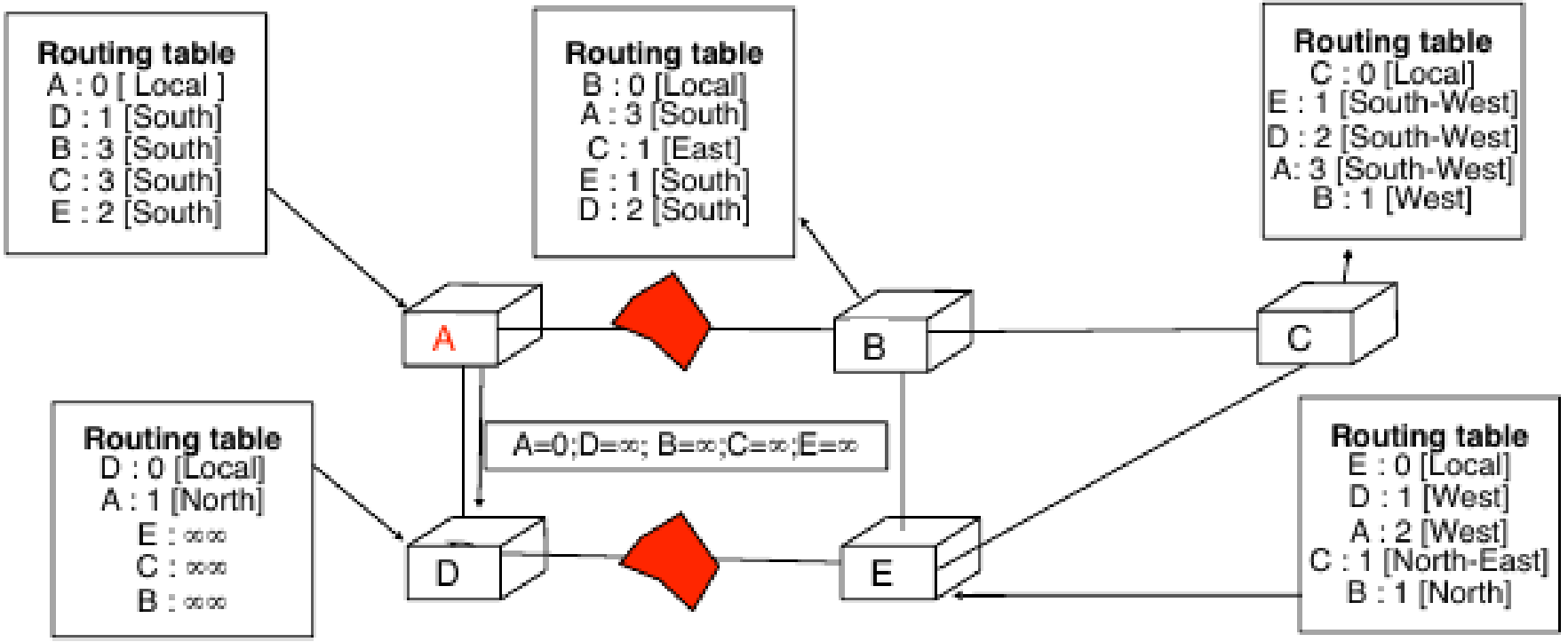
How can we avoid it ?

Split Horizon

Each router creates a distance vector for each link. On link n, the router announces the the routers learned from router N with an infinite cost.

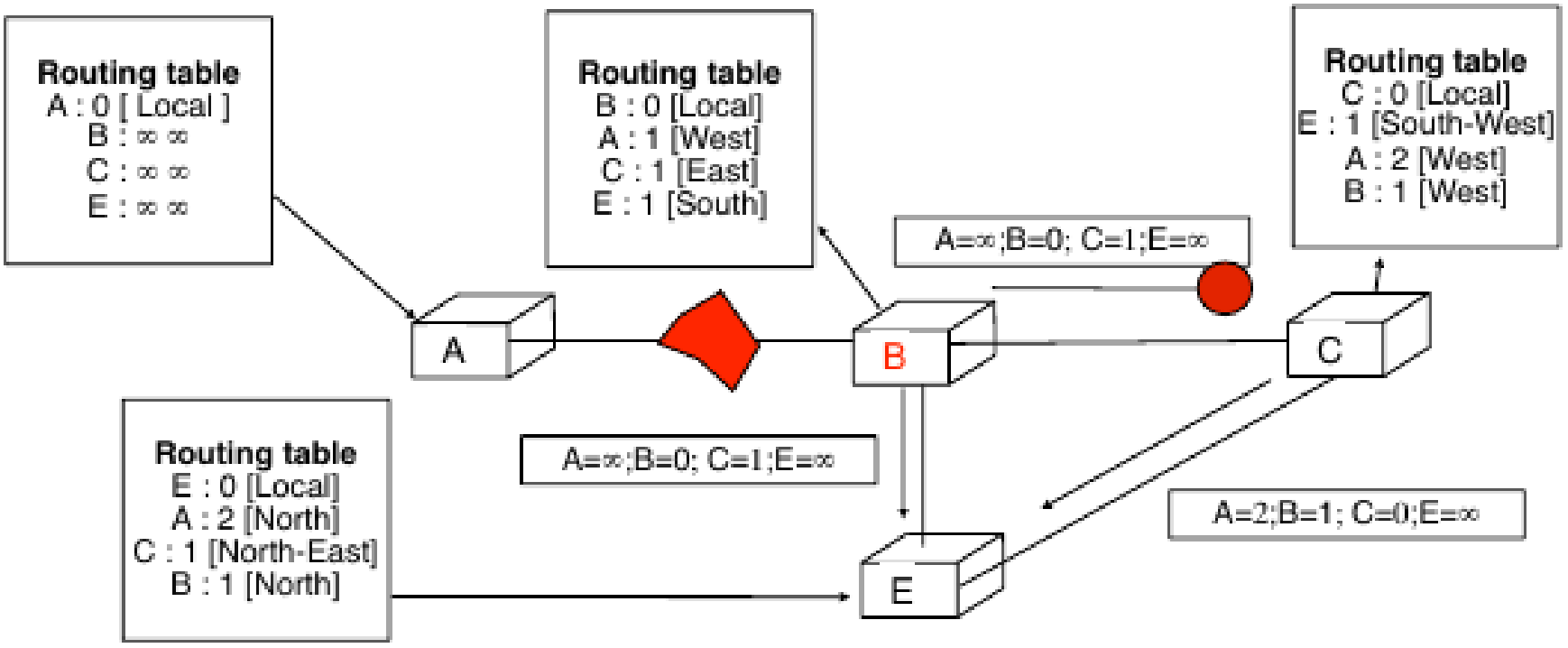
```
Pseudocode

Every N seconds:
 for each link=l
 { /* one different vector for each link */
  Vector=null;
  for each destination=d in R[]
  {
   if (R[d].link<>l)
   {
    Vector=Vector+Pair(d,R[d].cost);
   }
   else
   {
    Vector=Vector+Pair(d,∞);
   }
  }
  Send(Vector);
 }
```

Split horizon



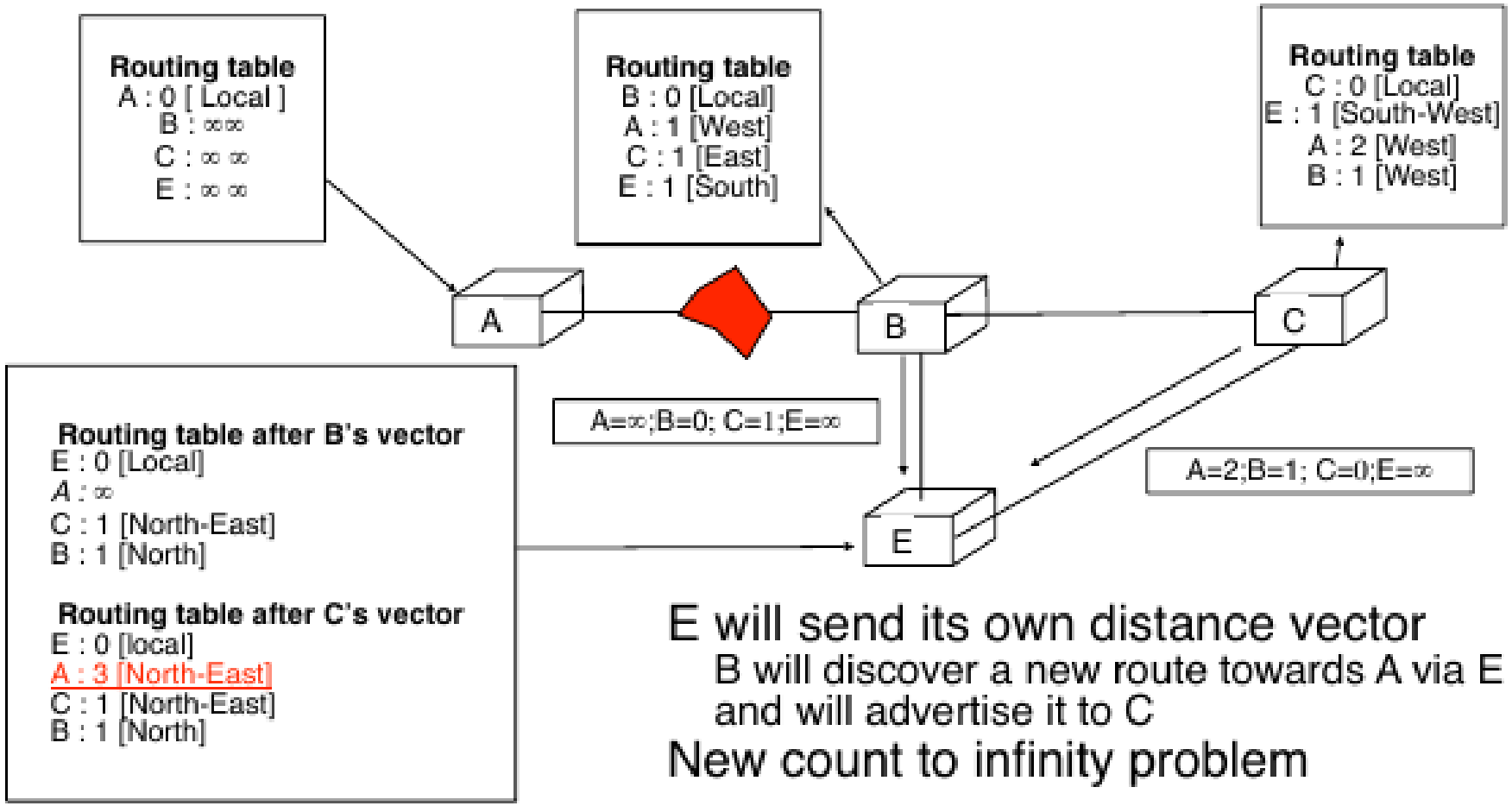
Limitations of Split Horizon?

Split horizon limitation



The vector from B to C does not arrive. Before B sends a new vector, C sends his vector, with a route to A.

Split horizon limitation



Link State Routing

Idea

Instead of distributing summaries of routing tables, wouldn't it be better to distribute network map ?

How to build such as network map ?

Each router must discover its neighbours

It should be possible to associate a cost to each link since all links are not equal

Each router sends its local topology to all routes and assembles the information received from other routers

Routers build the network graph and used Dijkstra's algorithm to compute shortest paths

Link cost

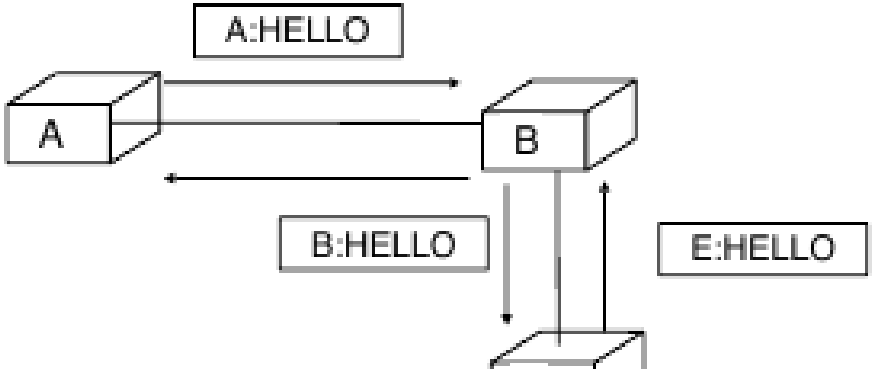
- Principle
  - one cost is associated with each link direction
- Commonly configured link costs
  - Unit cost
    - simplest solution but only suitable for homogeneous networks
  - Cost depends on link bandwidth
    - high cost for low bandwidth links
    - low cost for high bandwidth links
  - Cost depends on link delays
    - often used to avoid satellite links
- Cost based on measurements
  - Use HELLO to measure link rtt
    - allows to track link load, but be careful if the measurement is not stable enough as each delay change will cause a topology change ...

Link State Routing

How does a router discover its neighbours ?

By manual configuration
Unreliable and difficult to manage

By using HELLO packets
Every N seconds, each router sends a HELLO packet on each link with its address
Neighbours replay by sending their own address
Periodic transmission allows to verify that the link remains up and detect failures



Network topology assembly

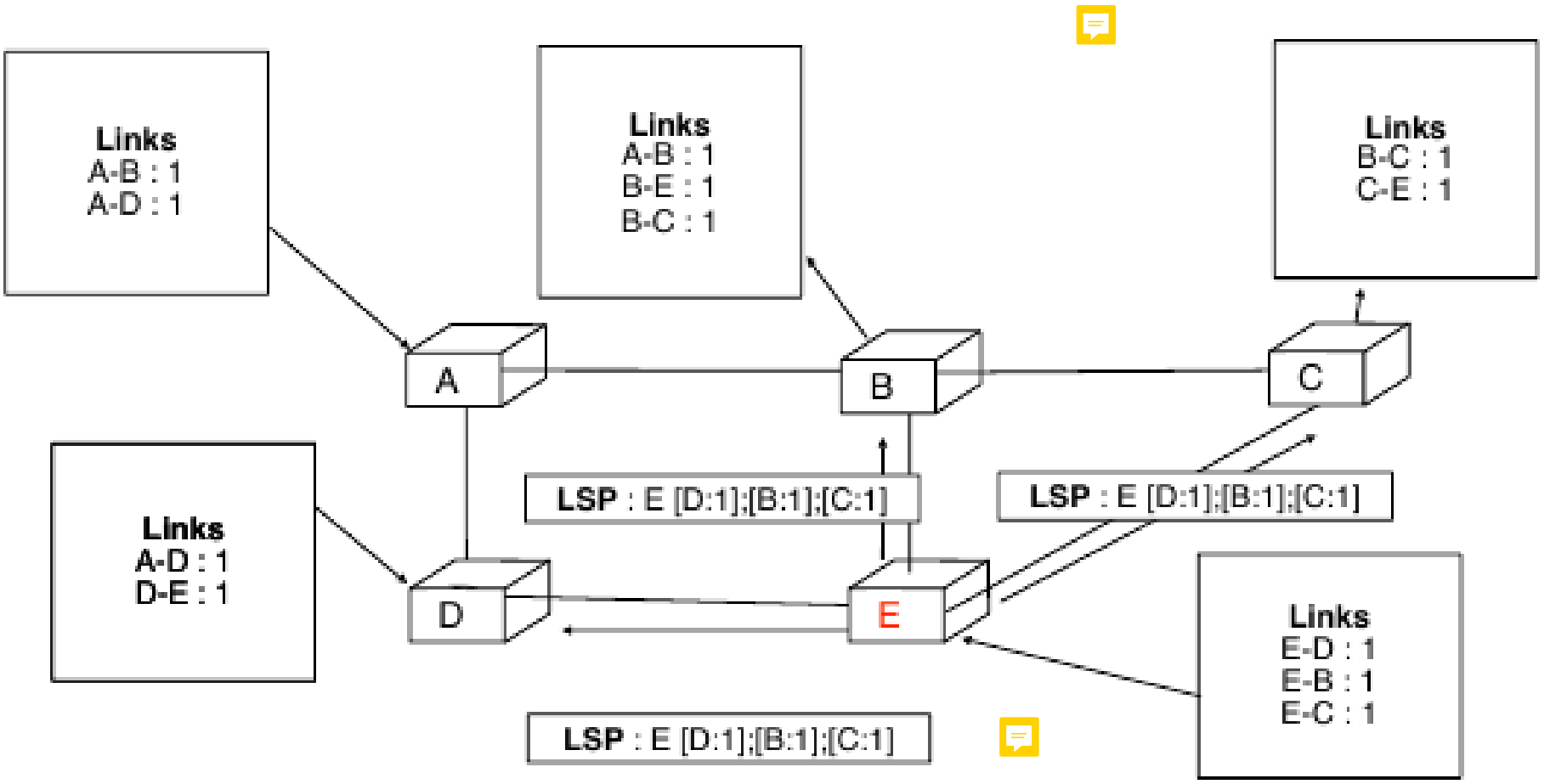
How to assemble the network topology
- By receiving HELLOs, each routers builds its local part of the network map
- Each router summarises its local topology inside one link state packet that contains
  - router identification
  - pairs (neighbour identification,cost to reach neighbour)

When should a router send its link state packet ?
- in case of modification to its local topology
  - allows to inform all other routers of the change
- Every N seconds
  - allows to refresh information in all routers and makes sure that if an invalid information was stored on a router due to memory errors it will not remain in the router forever

Link state packets distribution

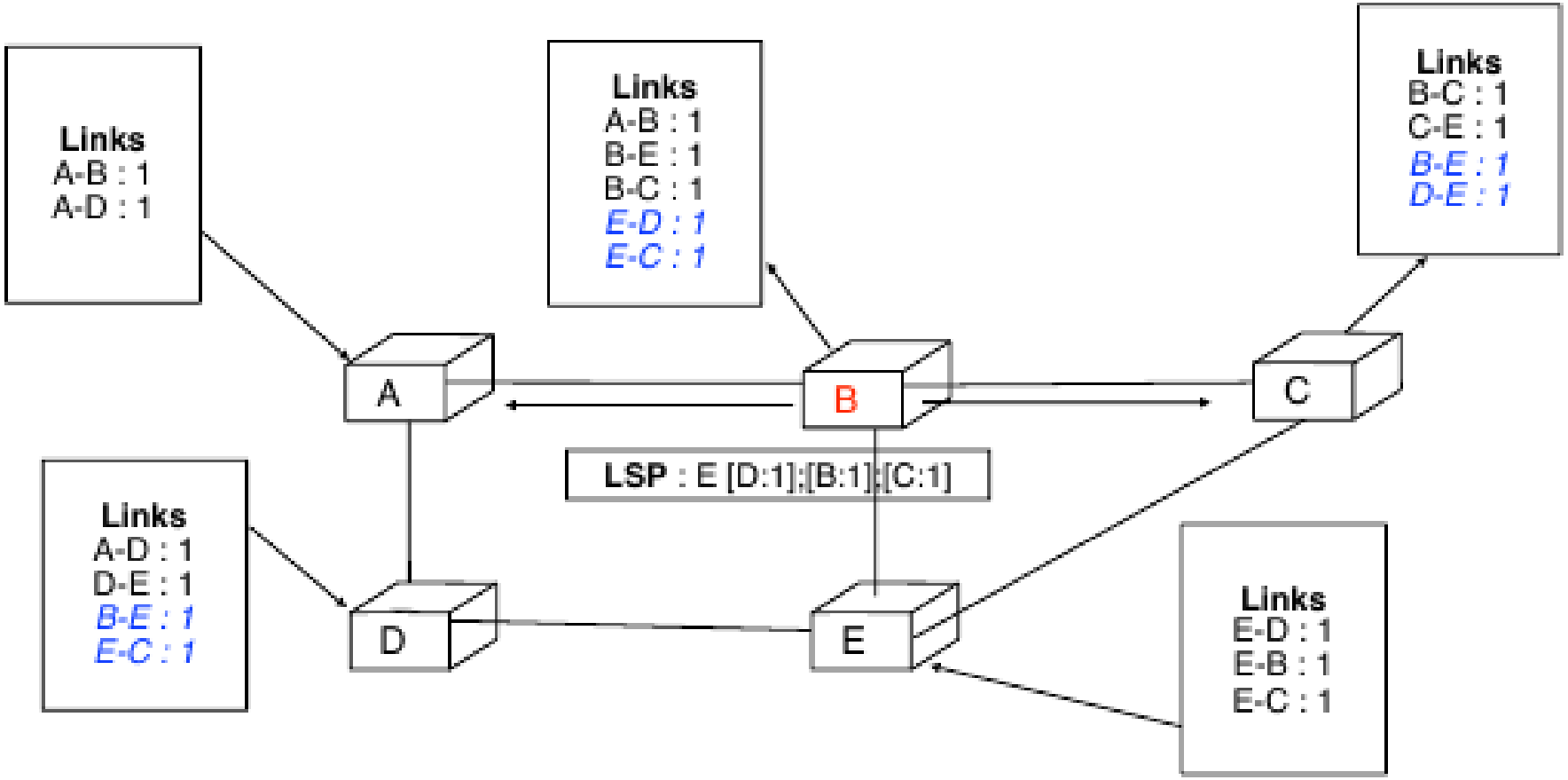
- Naive solution
  - Each router sends one packet to each other router in the network
    - This solution can only work if
      - All routers know the address of all other routers in network
      - All routers already have routing tables that allow them to forward packets to any destination
- Realistic solution
  - Does not rely on pre-existing routing tables
  - Each router must receive entire topology
  - First solution
    - Each router sends local topology in link state packet and sends it to all its outgoing links
    - When a router receives an LSP, it forwards it to all its outgoing links except the link from which it received it

LSP Flooding

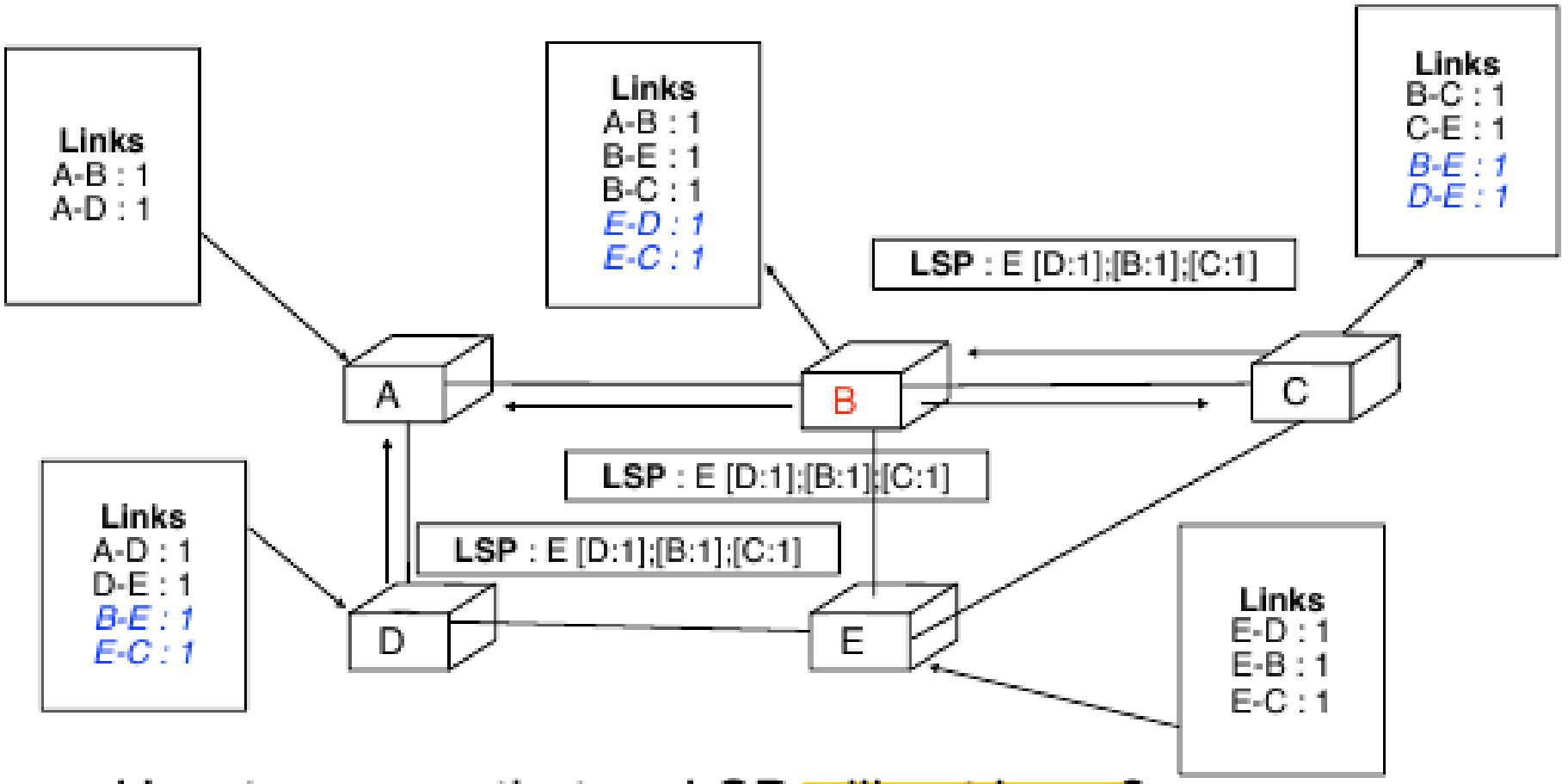


Assumes that all links have a unit cost

LSP flooding



LSP Flooding



How to ensure that an LSP will not loop ?

LSP flooding

How to avoid LSP flooding loops ?

A router should not reflood an LSP that it has already and flooded

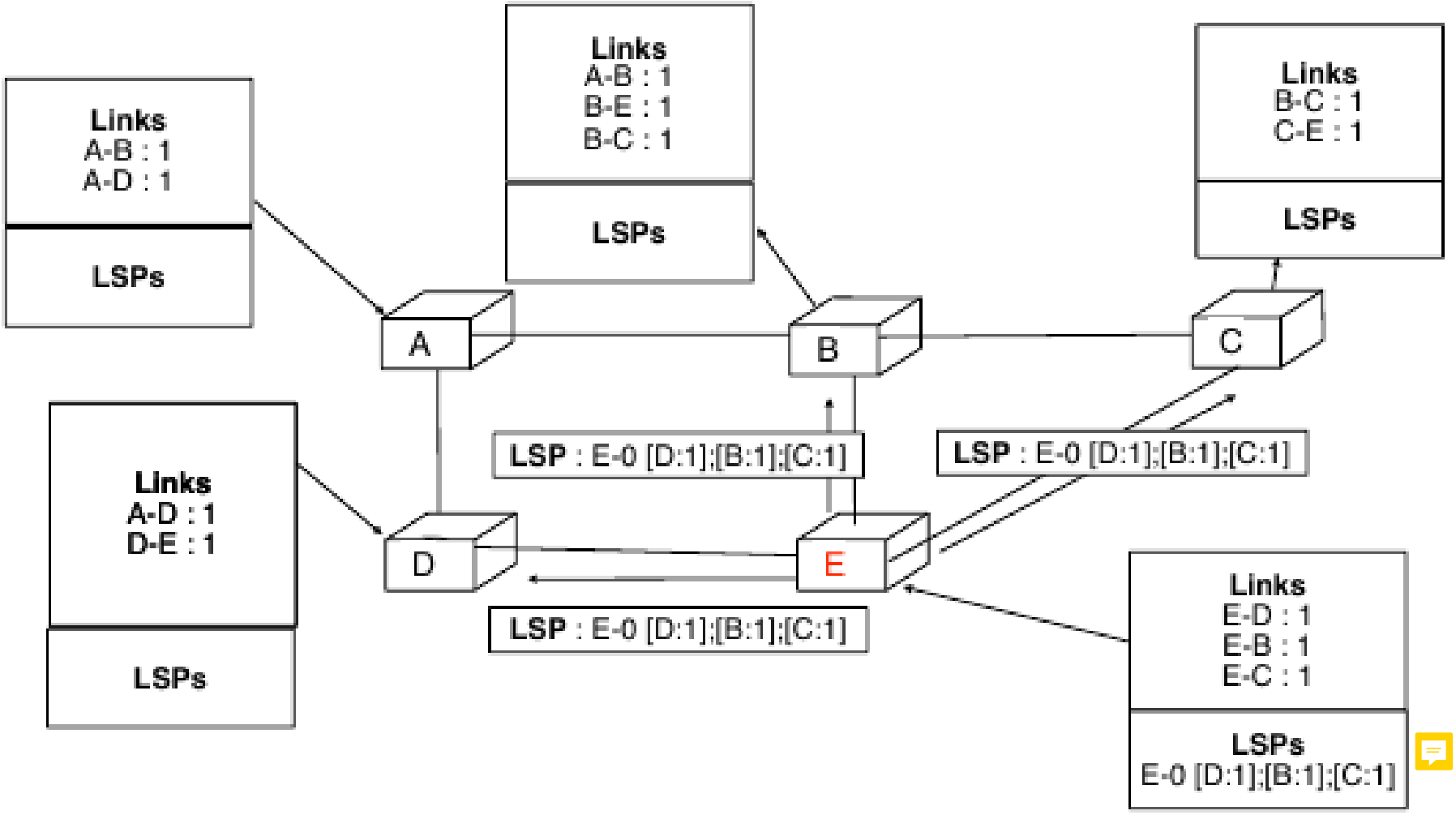
Solution

- LSP contents
  - sequence number
    - incremented every time an LSP is generated by a router
  - address of LSP originator
  - pairs address:distance for all neighbours of the originator

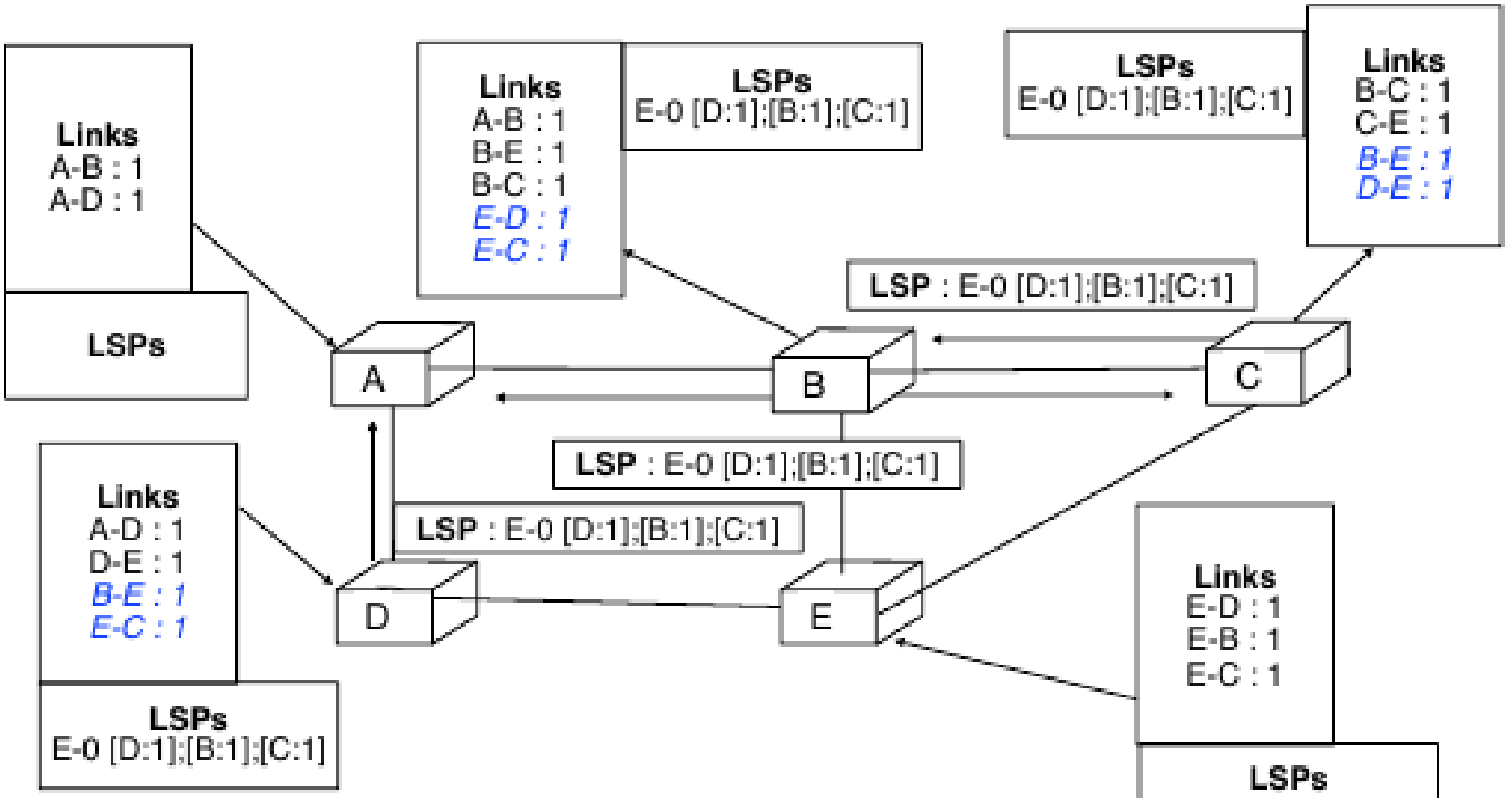
Each router must store the last LSP received from each router of the network

A received LSP is processed and flooded only if is it more recent than the LSP stored in the LSDB

LSP flooding



LSP flooding

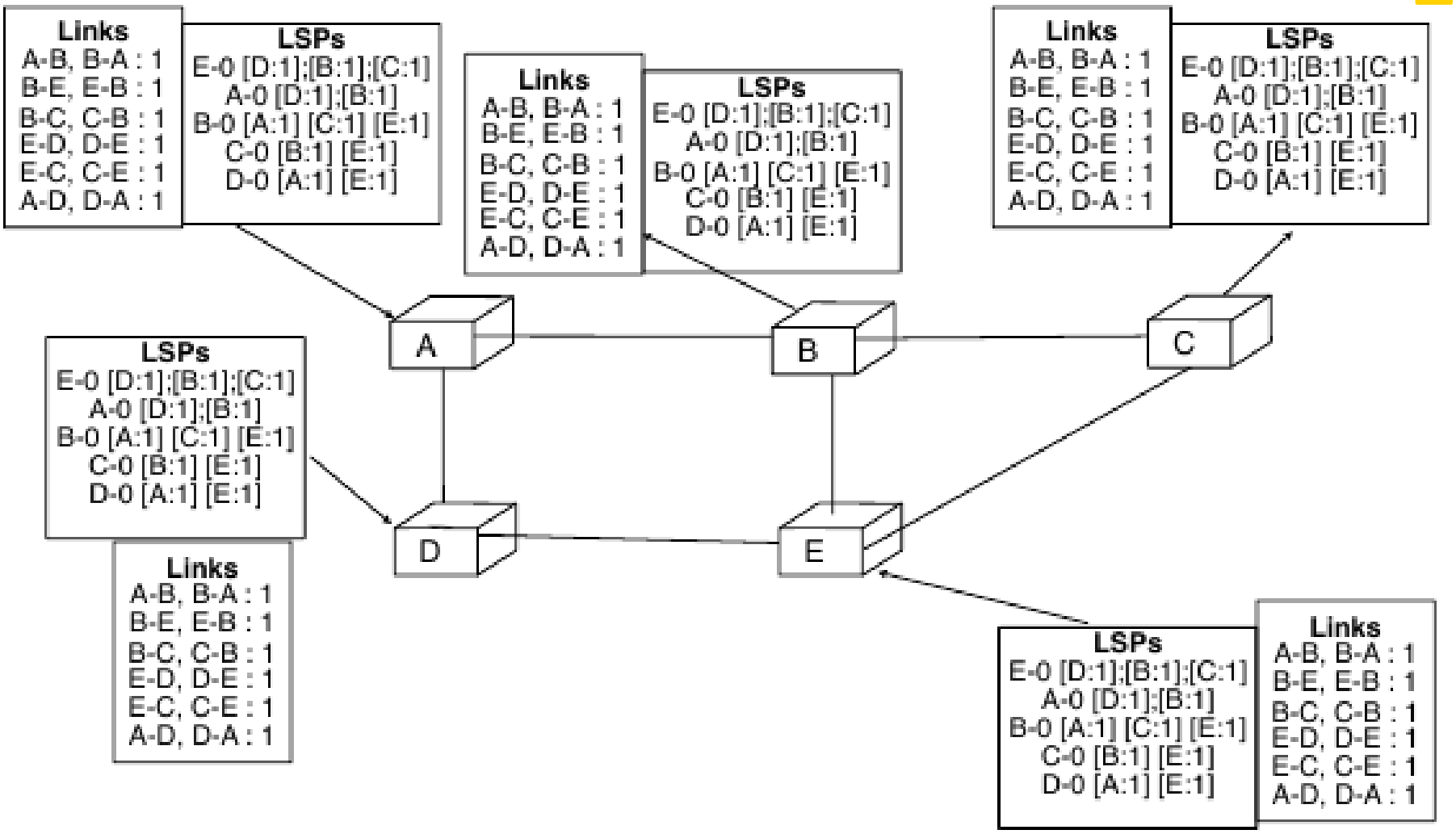


Thanks to its LSP table

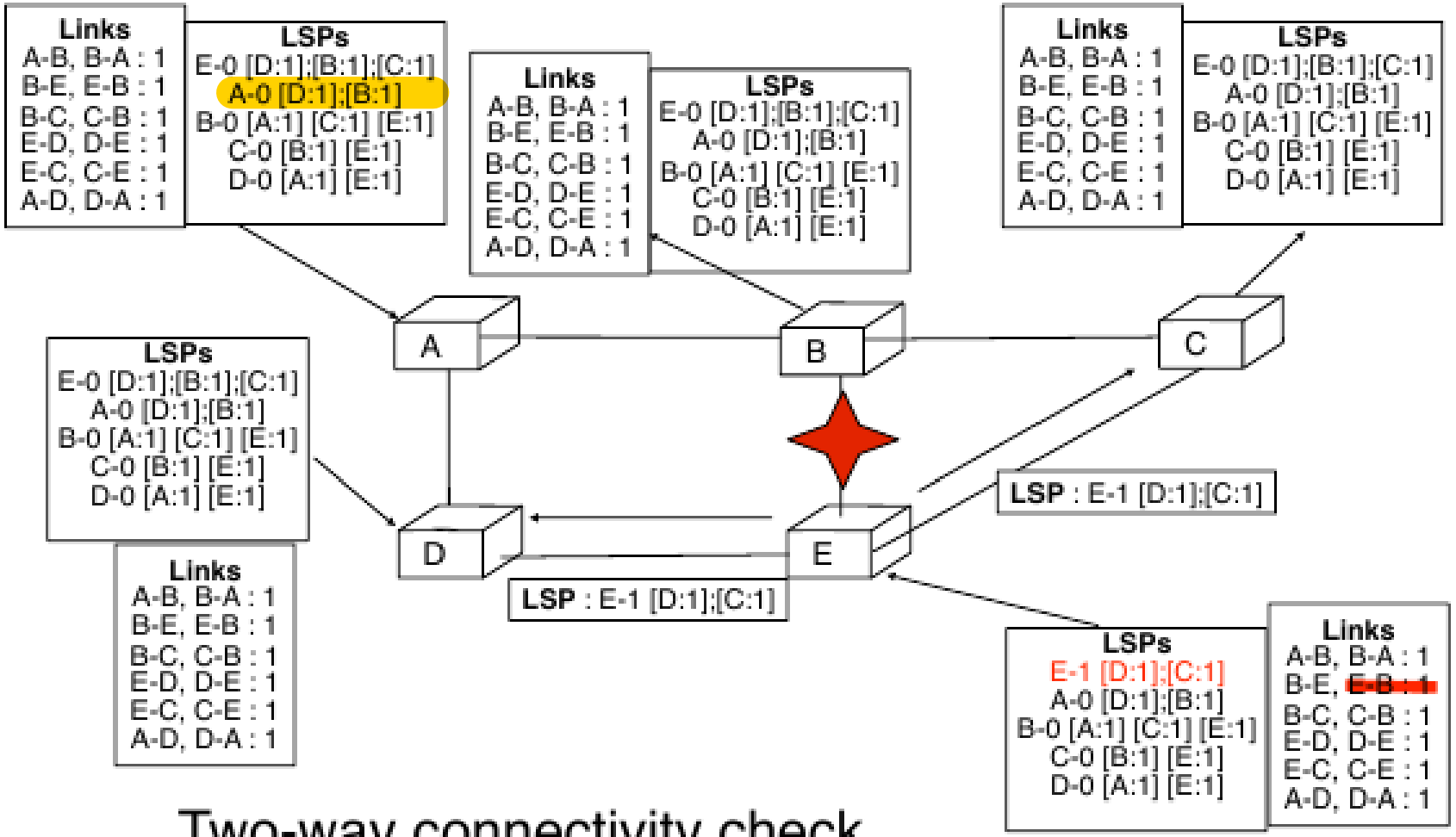
A can detect that it received same LSP via B and D

C can detect that it received same LSP via B and E

LSP flooding



Link Failures



Two-way connectivity check

A link is only considered useable if both directions have been advertised

Router Failures

- What happens if a router fails ?
  - All its interfaces become unusable and do not reply anymore to HELLO packets
- What happens when the router reboots ?
  - It will send its LSP with its sequence number set to zero
    - If older LSPs from same router were still in network, then the new LSP will not be flooded

Router Failures

- What happens if a router fails ?
  - All its interfaces become unusable and do not reply anymore to HELLO packets
- What happens when the router reboots ?
  - It will send its LSP with its sequence number set to zero
    - If older LSPs from same router were still in network, then the new LSP will not be flooded
- Solution
  - Add "age" field inside each LSP
  - Each router must decrement age regularly
    - even for the LSPs stored in its LSDB
  - LSP having age=0 is too old and must be deleted
  - Each router must flood regularly its own LSP with age>0 to ensure that it remains inside network

Building the Shortest Path

With the LSPs, and thus the full map of the network, the routers now have to find the shortes path to any given destination.

Represent the network in the form of a tree, using Dijkstra’s shortest path algorithm.

Objects used:

- A tree, holding the results
- a set of candidates (set of next routers to consider)
- a set of all the routers in the network

Dijkstra's Shortest Path

Computing the shortest path tree

- At the beginning, the tree only contains the root node
- Adjacent routers are placed with the cost of their link in the candidates list
- Candidate router with lowest cost is chosen and added to the tree
- Consider the neighbours of the chosen candidate router and update the candidate router list if
  - one of the new neighbours was not already in the candidates list
  - one of the new neighbours was already in the candidates list but with a longer path than the one in the current list
- Algorithm continues with the new candidates list and ends when all routers belong to shortest path tree

Table of Contents