Network Layer - Count-to-Infinity and Link State Routing

COMPUTER NETWORKS A.A. 24/25

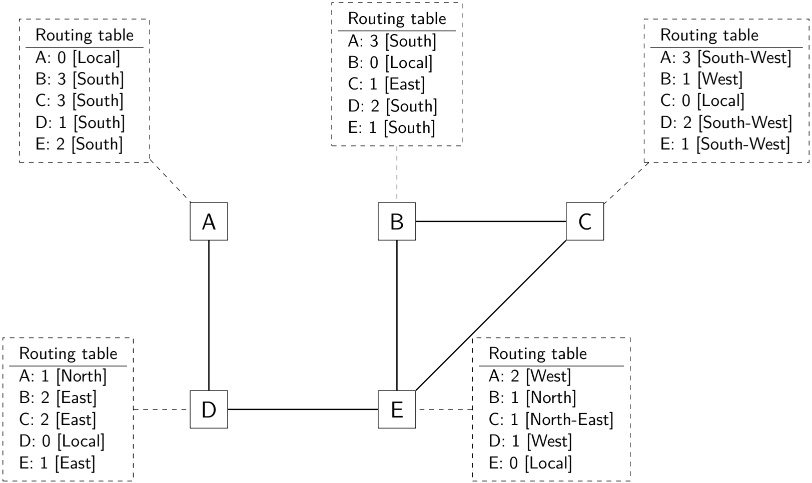
Leonardo Maccari, DAIS: Ca' Foscari University of Venice,
leonardo.maccari@unive.it

Venice, fall 2024

License

Sect. 1 Count-to-Infinity and Posion Reverse

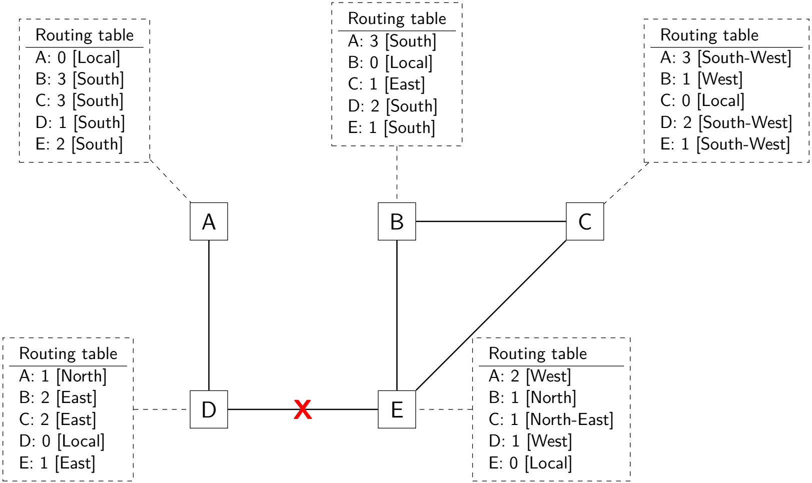
Back to our Toy Network, after Convergence



Network Failure

- So far we assumed all the packets are correctly delivered.
- But we know that this is not always the case.
- Let's introduce another link failure, followed by the loss of a packet containing the DV.
- Let's assume link D-E breaks...

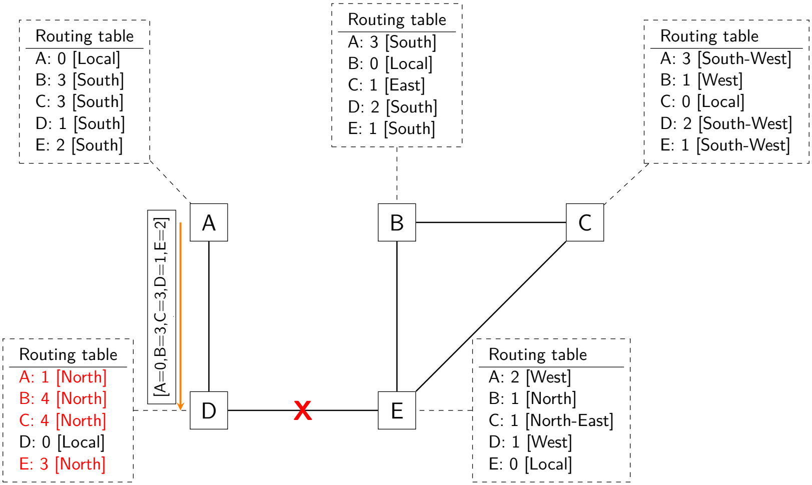
Link D-E fails



Partitioned network

- Now the network is partitioned, (A,D) can not communicate anymore with (B,C,D)
- Node D is the first to detect the failure and sends sets its cost to B/C//E to infinite
- However, node A generates a DV before D generates its own DV.

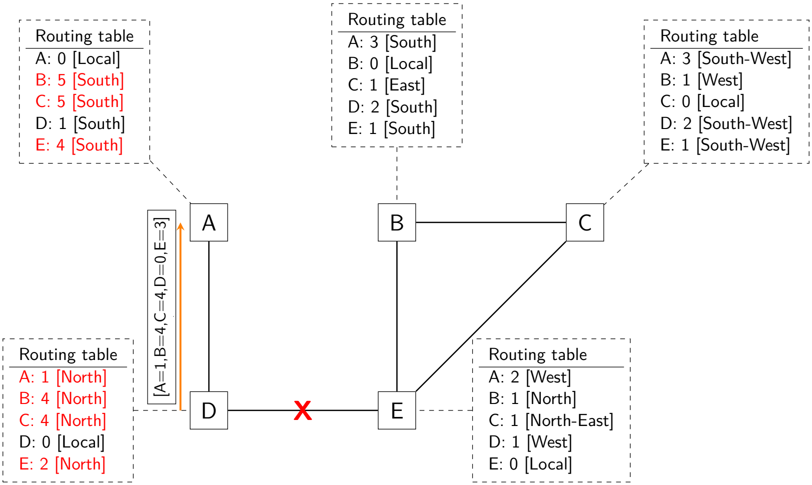
Router A sends the DV



Router D receives the DV

- Router D updates its routing table, and concludes that it can use router A as a next hop to E, B, and C
- Note that when router D receives the DV from A it can not know that the route that router A is exporting are going exactly through router D.
- Then after a while, router D will send its own DV to A

Router D sends the DV



Count-to-infinity Problem

- Router A updates its tables increasing the distance to (B,C,E), and it will send another DV
- Router D will receive the DV, update it tables increasing the distance. . .
- The result is that routers A and D will at some point reach the maximum possible distance, overflow the number and keep doing the same over and over

Easy Fix: Triggered Updates

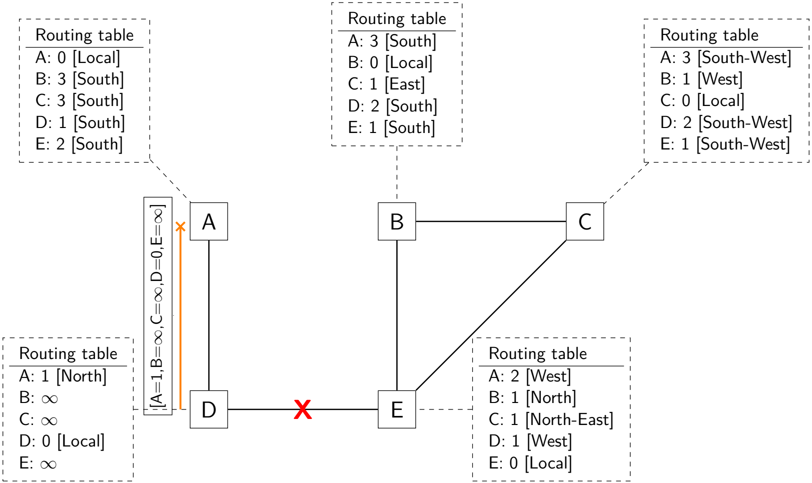
- An easy fix is to trigger a DV generation when the link fails.
- When D sets the cost to infinity (or in general, when a certain route changes its cost), it will generate a new DV immediately.
- These are called triggered updates in RIP[1]

[1] See `https://datatracker.ietf.org/doc/html/rfc2453#section-3.4.4`

Triggered Updates: Problem Solved?

- Triggered updates may seem to be a solution to the problem, however this is not the case for two reasons.
- The first is that for as much quick as a router may react, there is always the chance that D receives the update in between sensing the link failure and sending the DV
- The second is that a DV packet can get lost.
- Note that triggered updates will create a storm of packets, as each router that is affected will generate a new DV.

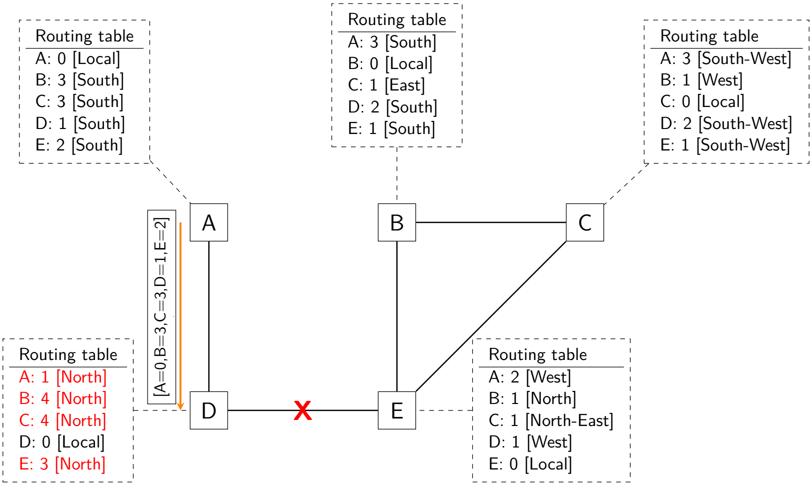
Link D-E fails: DV is lost



Packet Loss

- Let's assume the packet never reaches A, this could be due to any link failure reason we analysed before.
- After some time A will send its own DV to D

Router A sends the DV. . . loop is created



Count-to-infinity With Triggered Updates

- The problem is made possible by some events:
  - There must be a loop and a single full-duplex link already creates a one-hop loop
  - There must be a change in the topology
  - A packet is lost
- Note that it is not important how the (B,C,E) network is made. It could be just one router.

Count-to-infinity

- Note also that in the DV message routes do not contain the path used to reach the destination.
- For every route only the distance is included (from this the name "*distance vector*")
- However, the receiving router does not know that the next-hop used by the neighbors, is actually itself
- This leads to the repeated increase in the distance and the divergence of the routing tables metrics.

Split-Horizon with Poison Reverse

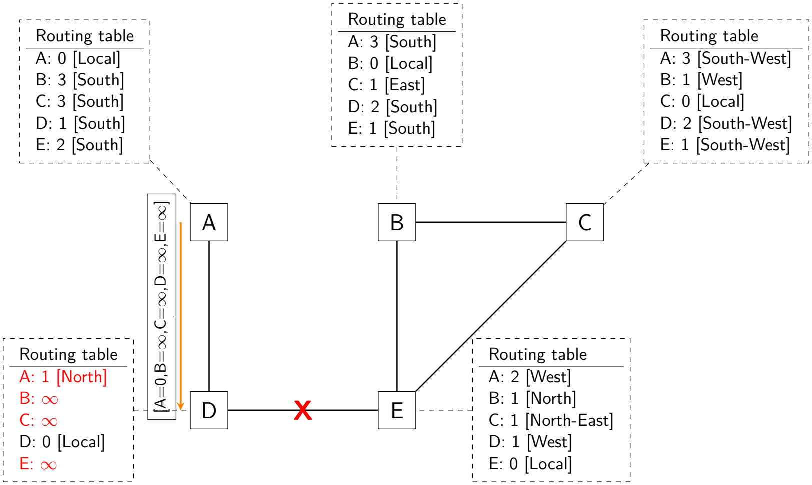
A variant of pure DV is given by the following pseudo-code for generating DVs:

```
Every N seconds:
    for ifc in interfaces:
        # one vector for each interface
        v = Vector()
        for d in R[]:
            if (R[d].link != ifc):
                v.add(Pair(d, R[d].cost))
            else:
                v.add(Pair(d, infinity))
        send(v, ifc)
```

Split-Horizon with Poison Reverse

- This is called split-horizon with poison-reverse.
- Split horizon tells that for a router its *horizon* is not all the same, it is split between those routers that it uses as next-hop, and all the others
- Plus, the router is *poisoning* the routing table of its neighbor, telling it that it does not have a route to some destination, if the route is passing through the neighbor itself.
- This way, the count-to-infinity can be solved.

Router A sends the DV (using Poison Reverse)



Poison Reverse

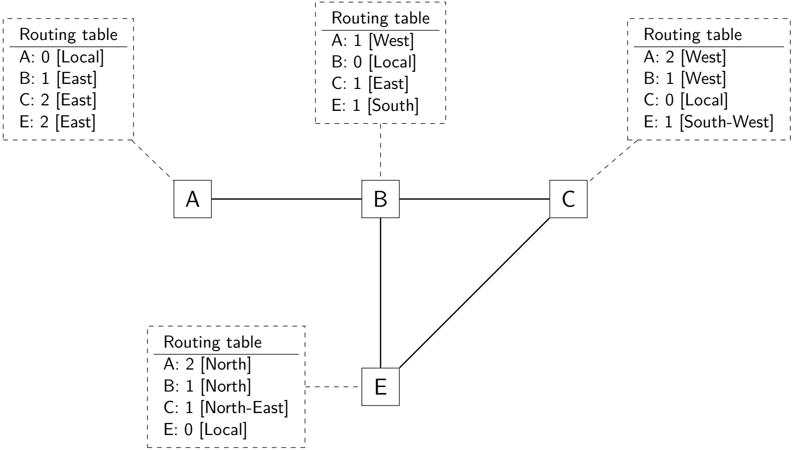
- If Poison Reverse is used, router D will not update its routing table
- Router A still has broken tables
- At the next time interval, router D will send the DV again
- As soon as one DV reaches router A, router A will update its routing table correctly

Quoting myself. . .

"*This way, the count-to-infinity can be solved*"

Unfortunately, this was too optimistic, there are other situations in which count-to-infinity can happen. . .

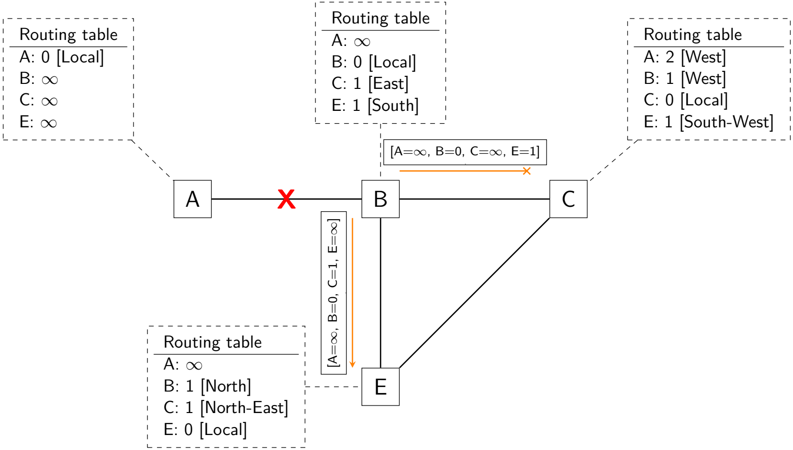
Another Topology



Link Failure

- Now assume link A-B breaks
- Router B will send a DV to C and E
- all routers use poison reverse.
- However, router E correctly receives the DV, while router C fails to receive it.

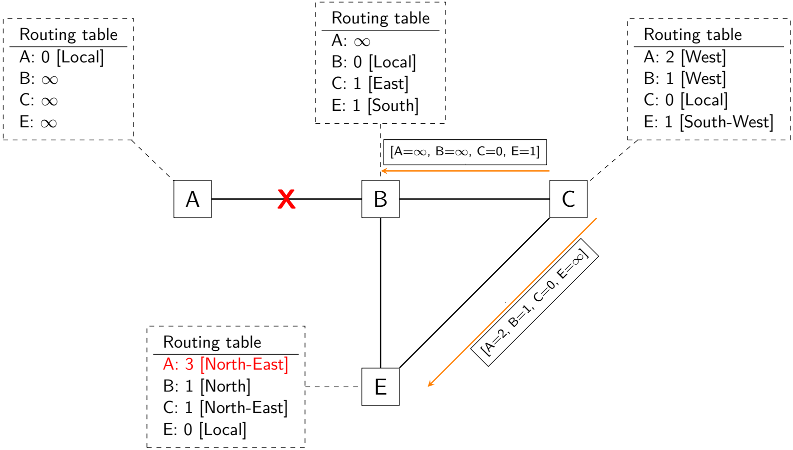
Link A-B Fails: B sends a DV



Router C Sends its DV

- Assume that now Router C sends its own DV to B and E
- Without poison reverse, this would have created a loop between Router B and C, but this does not happen
- Router E will update is table

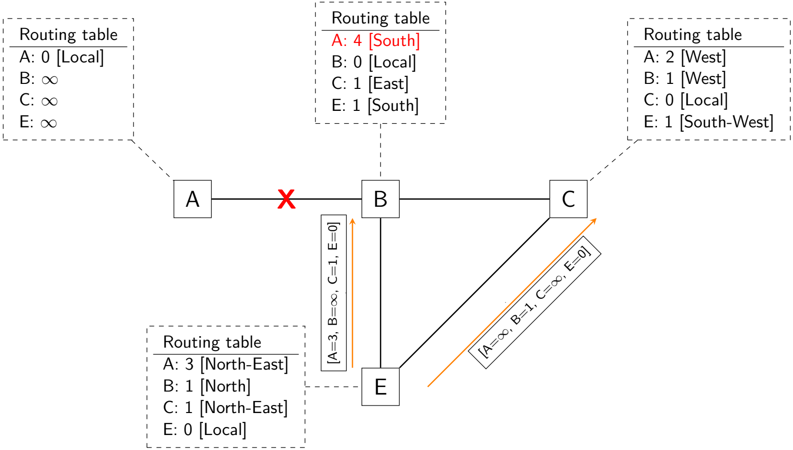
Link A-B Fails: C sends a DV



Router E sends its DV

- Assume that now Router C sends its own DV to B and E
- Router B will not update its table as C uses poison reverse
- However, router E will update its table
- There is a black-hole in B

Link A-B Fails: E sends a DV



A new Loop

- No more black-hole, a new loop is formed by E-B-C
- When B will send its DV, C will update and so on. . .
- A new count to infinity is created

Timer Management

- One way to avoid this is to prevent route updates for a certain time after.
- That means that when a route cost is set to ∞, the router should not accept an update for a certain time.
- This makes it possible to avoid single events of packet loss, and rely on the fact that the information will propagate to neighbors

Timer Management

- RFC1058 specifies the following timers[2]:
  - 30s: generation of DVs
  - 180s: route timeout, after that the route is set to ∞ cost
  - 240s: garbage collection. 60s after the timer expires, the route is removed
- Cisco implementations added a hold down timer, that is, when a route is set to ∞, for 180s no update is accepted for that route[3]
- That of course makes convergence extremely slow.

[2] See `https://datatracker.ietf.org/doc/html/rfc1058#section-3.3`

[3] See `https://community.cisco.com/t5/switching/rip-and-hold-down-timer/td-p/1175544`

Count-to-infinty: Bottom Line

Count-to-infinity can always happen when there are loops and loss of packets. Split Horizon and Poison Reverse only fix the problem when the loop is made of 2 routers, but not when the loop is larger than 2 routers. Slow down the convergence can fix most of the practical cases, but at a high cost.

To fix count-to-infinity routers should not export only the distance, but the whole path to the destination. This is done in path-vector routing protocols, like BGP, that we will see in future lessons.

Sect. 2 Link State Routing

Link-State (LS) Routing

- Link state routing is the second family of routing protocols.
- While distance vector routers use a distributed algorithm to compute their routing tables, link-state routers exchange messages to allow each router to learn the entire network topology.
- Based on this learned topology, each router is then able to compute its routing table by using a shortest path computation such as Dijkstra's algorithm

Assumptions on the Network Graph

- Again, the network is represented as a graph, with each router being a node and each edge being a communication link
- Edges are directional, i.e.: there are two edges between every neighbor, so each direction can be characterized independently
- Every link has a weight: the highest, the worse
- The algorithm chooses the path with the lowest weight based on some function applied to the weights.

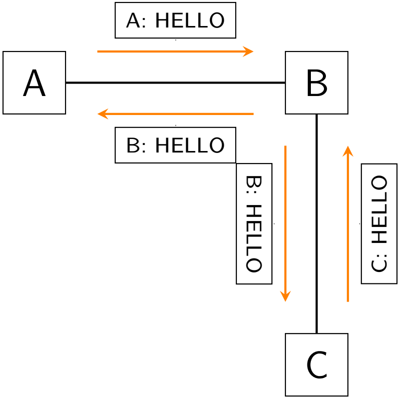
Link Weights

Links can use different kinds of weights, some are:

- Unit weight, same for all links
- weight proportional to the propagation delay on the link. The shortest path routing uses the paths with the smallest propagation delay.
- $\frac{C}{\texttt{link\_capacity}}$ where C is a constant. The higher the capacity of a link, the lower the weight.

LS Routing: Working Principles

- Every router has a unique address
- A router sends a `HELLO` message every N seconds on all its interfaces. `HELLO` contain the router's address.
- As its neighboring routers also send `HELLO` messages, the router automatically discovers to which neighbors it is connected.
- `HELLO` messages are also used to detect link and router failures. A link is considered to have failed if no `HELLO` message has been received from a neighboring router for a period of $k \times N$ seconds for some constant k.



LSP Messages

- Once a router has discovered its neighbours, it must reliably distribute its local links to all routers.
- Each router builds a link-state packet (LSP) containing the following information :

`LSP.Router`: identification (address) of the sender of the LSP
`LSP.age`: age or remaining lifetime of the LSP
`LSP.seq`: sequence number of the LSP (incremented at each LSP)
`LSP.Links[]`: links advertised in the LSP, with two fields:
  `LSP.Links[i].Id`: identification of the neighbour
  `LSP.Links[i].cost`: cost of the link

LSP Flooding

- These LSPs must be reliably distributed before the routers have a working routing table.
- A Flooding algorithm is used to efficiently distribute the LSPs of all routers.
- Each router that implements flooding maintains a link state database (LSDB) containing the most recent LSP received by each other router.

LSP Flooding

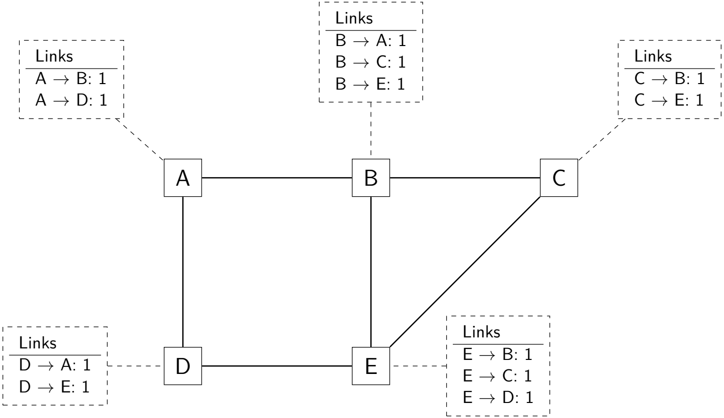
- When a router receives an LSP, it first verifies whether this LSP is already stored inside its LSDB using the sequence number.
- If so, the router has already distributed the LSP earlier it does not need to forward it.
- Otherwise, the router forwards the LSP on all links except the link over which the LSP was received.

LSP Flooding

```
# links is the set of all links on the router
# Router R LSP arrival on link l
if newer(LSP, LSDB(LSP.Router)) : # get last LSP from the DB, compare with current
    LSDB.add(LSP)  # implicitly removes older LSP from same router
    for i in links:
        if i!=l:
            send(LSP,i)
# else, LSP has already been flooded
```

Note that a sequence number is not a time-stamp. At some point it will wrap and the code needs to take care of it.

A Larger Network after Receiving HELLO messages

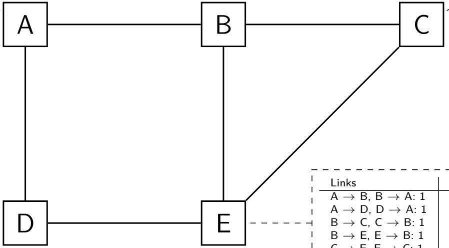


A Larger Network after Flooding LSP messages

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
B → E, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-0 [B:1];[C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
B → E, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-0 [B:1];[C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
B → E, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-0 [B:1];[C:1];[D:1]
D → E, E → D: 1	



Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
B → E, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-0 [B:1];[C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
B → E, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-0 [B:1];[C:1];[D:1]
D → E, E → D: 1	

Link Failure

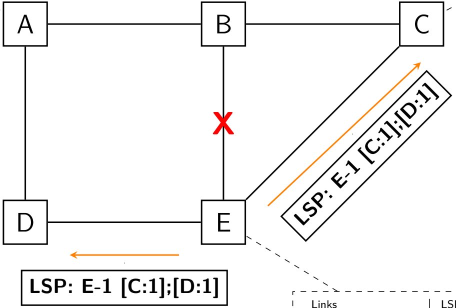
- When a link fails, the two routers attached to the link detect the failure by the absence of `HELLO` messages.
- We know links are lossy, so failing to receive one `HELLO` dos not mean the link has broken
- If no `HELLO`s are received during the last $k \times N$ seconds, a router detects the failure of one of its local links.
- It generates and floods a new `LSP` that no longer contains the failed link.
- This new `LSP` replaces the previous `LSP` in the network

1-hop Propagation of the LSP

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
B → E, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-0 [B:1];[C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
B → E, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-0 [B:1];[C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	



Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

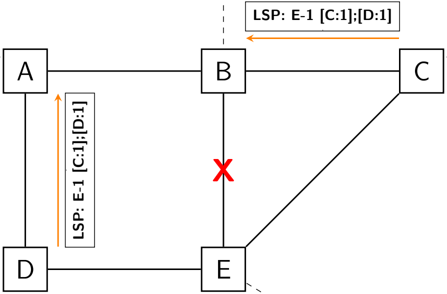
Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E: 1~~, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

2-hop Propagation of the LSP

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	



Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

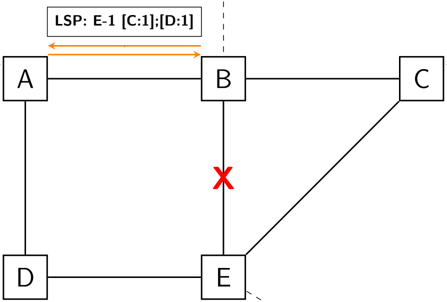
Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E: 1~~, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

3-hop Propagation of the LSP

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	



Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E, E → B: 1~~	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

Links	LSPs
A → B, B → A: 1	A-0 [B:1];[D:1]
A → D, D → A: 1	B-0 [A:1];[C:1];[E:1]
B → C, C → B: 1	C-0 [B:1];[E:1]
~~B → E: 1~~, E → B: 1	D-0 [A:1];[E:1]
C → E, E → C: 1	E-1 [C:1];[D:1]
D → E, E → D: 1	

Important Notes

1. After the two-hop propagation the network has converged. The flooding mechanisms makes it possible that a router receives more than one copy of the same message.
2. The time at which router B and E detect the failure may not be the same. So router B can receive the *bad news* before it senses that the link failed
3. In general, a link working in only one direction is not functional, so even if the LSP from router B has not been received, other routers will remove the broken link when they receive the update from router E.
4. On the other way around, if the link is fixed, routers will wait till they receive the LSP from both B and E, to be sure the link works in both directions.

Important Notes

5. In most of the cases both router E and B will start the update process, and the two floods go in parallel
6. In the transitory phase, loops or blackholes can be created, for instance in a ring network.
7. If some message gets lost, the convergence may not be reached. `LSP` generation routing should be coupled with acknowledges from the receiving end, or, it must be periodic.

LS: Conclusions

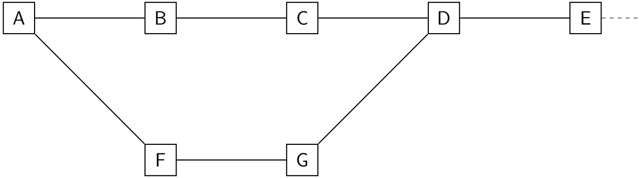
1. LS protocols provide more information to the routers. Complex algorithms can be used to build the routing tables, not only shortest path ones (for instance: multiplicative metrics can be used for the choice of the path, or flow can be distributed based on link-capacity).
2. When there is a change in the network topology the `LSP` is flooded quickly, and convergence is fast.
3. `H` messages are not propagated so they can be generated at a high frequency, link failure detection is faster

LS: Conclusions

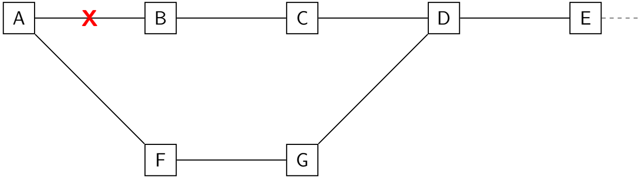
5. LS routing is more computationally demanding than DV routing.
6. On the Internet there are about hundreds of thousands of destination networks. Running Djikstra's algorithm on such a big network requires powerful hardware
7. Plus, a large network implies a high probability that something breaks at any second. This information must be propagated to the whole Internet, which has two consequences:
   - A lot of `LSP` flooding compared to DV. DV may repair a failure locally.
   - Constant recomputation of the shortest paths

DV Vs LS: Example

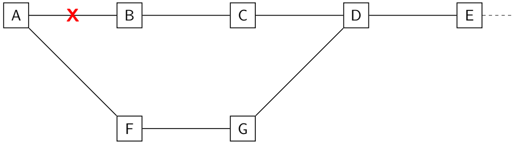
- LS protocols tend to be faster but more *talktative*: they produce more overhead in terms of control messages sent.
- Consider the following network, after convergence.
- Assume the path from D to A is D→G→F→A. Assume more routers are at the right of E.



Failure Example



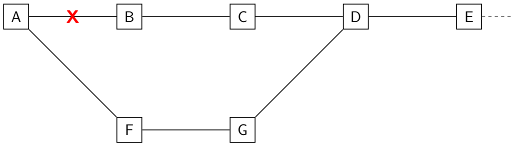
Failure Example: DV



- B and A will send a DV and, F and C are updated
- C and F will send a DV, D now knows that the path to A is broken
- G will send the DV. The RT of D does not change.

- Since the link A-B is not used by other routers, the information on the broken link does not propagate beyond D
- D will generate a DV, and this will fix the RT of C, and later on, the one of B
- Then B and C have a route to A, the network converged.

Failure Example: LS



- B and A will quickly detect the failure
- They will start an LSP flood

- As soon a router receives one LSP, it will recompute the whole graph and its own routing table
- This happens to *every* router.

LS scalability

In LS routing even a single modification that doesn't have any practical effect further than one hop will be propagated to all the network. All routers will re-apply Dijkstras' algorithm for nothing.

- The Internet has millions of routers and links.
- Probably every second some link break
- If the Internet routing was using LS, every second all the routers would recompute the shortest paths on a graph made of millions of routers $\rightarrow$ totally impractical

Concluding Remarks

- Irrespectively of the routing protocol we know that when there is a change in the topology:
  - temporary loops can be created: a loop delivers copies of packets
  - packets will go through different paths: it can be that one that was sent later on takes a faster path
  - black holes can be created: packets are lost.
- These effects are temporary, but inevitable
- So the network layer can not guarantee the correct, ordered delivery of packets.

! **At the network layer packets can be lost, corrupted, duplicated or received our of order. The upper layers must consider this.**