# Routing and forwarding

Damien Saucez
Inria Sophia Antipolis

UBINET/CSSR 12/21/2012
Evolving Internet II

# Contact information


- Damien Saucez
- Office: Inria, Lagrange L.145
- Email: damien.saucez@inria.fr
- Phone: +33 4 89.73.24.18

# Outline of the course

- 12/14/2012: naming and addressing
- 12/21/2012: routing and forwarding
- 01/25/2013: interior gateway protocols
- 02/01/2013: exterior gateway protocols
- 02/08/2013: security in networks
- 02/15/2013: final examination

# Table of Content

- Motivation
- Link state routing
- Distance vector routing
- Forwarding in Ethernet networks
- Operational corner
- Research corner

# Motivation

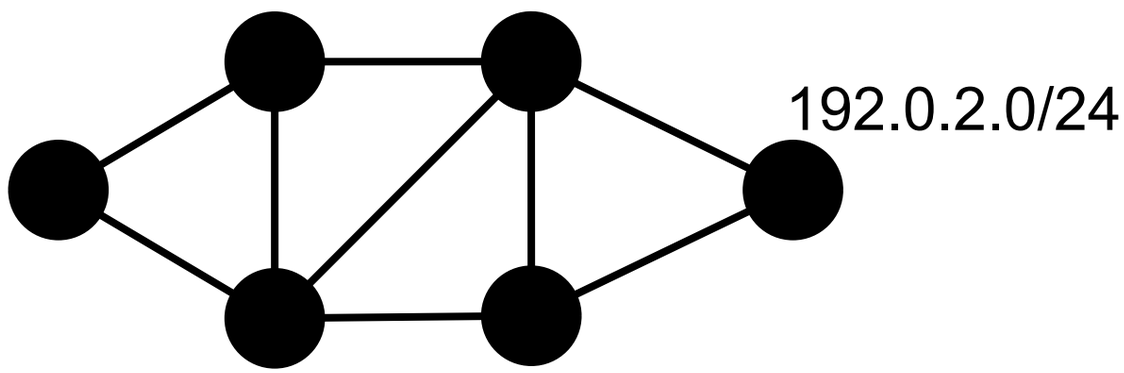
- Suppose you want to connect to a host in Sausalito, CA, with your laptop
  - is there a route to this host?
  - what route should you follow?
  - is there another (better) route?
  - what if a link or a node along the route fails?
- Routing is the mean to deal with all these questions

# Routing and forwarding

- The network layer aims at transmitting packets from their source to their destination
- Two mechanisms are combined to achieve this goal
  - **routing**
    - algorithm that determines the (best) paths between sources and destinations
      - construction of the routing table on routers
  - **forwarding**
    - for each destination, a router determines the interface(s) through which packets must be sent in order to reach their final destination, according to the routing table
      - construction of the forwarding table
- Routers make local decision based on global topology

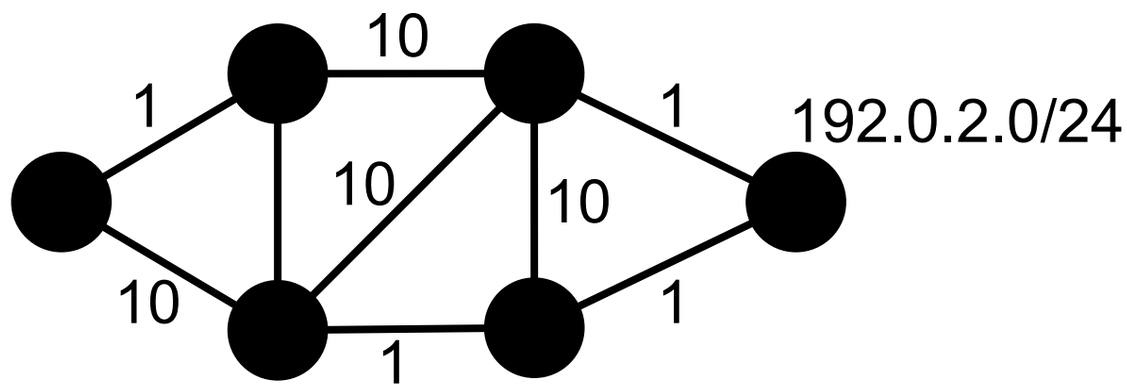
# Routing table

- Routing table: table containing the necessary informations about paths allowing a router to reach each destination
- which path to insert in the routing table?



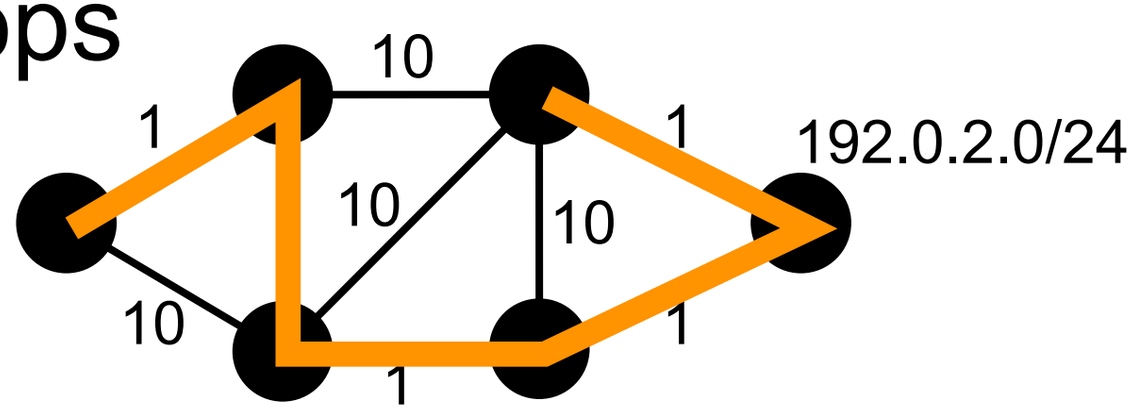
# Selection of the path

- Associate a cost to every path
  - each link receives a weight/cost
  - best path is selected, i.e., path with the lowest cost



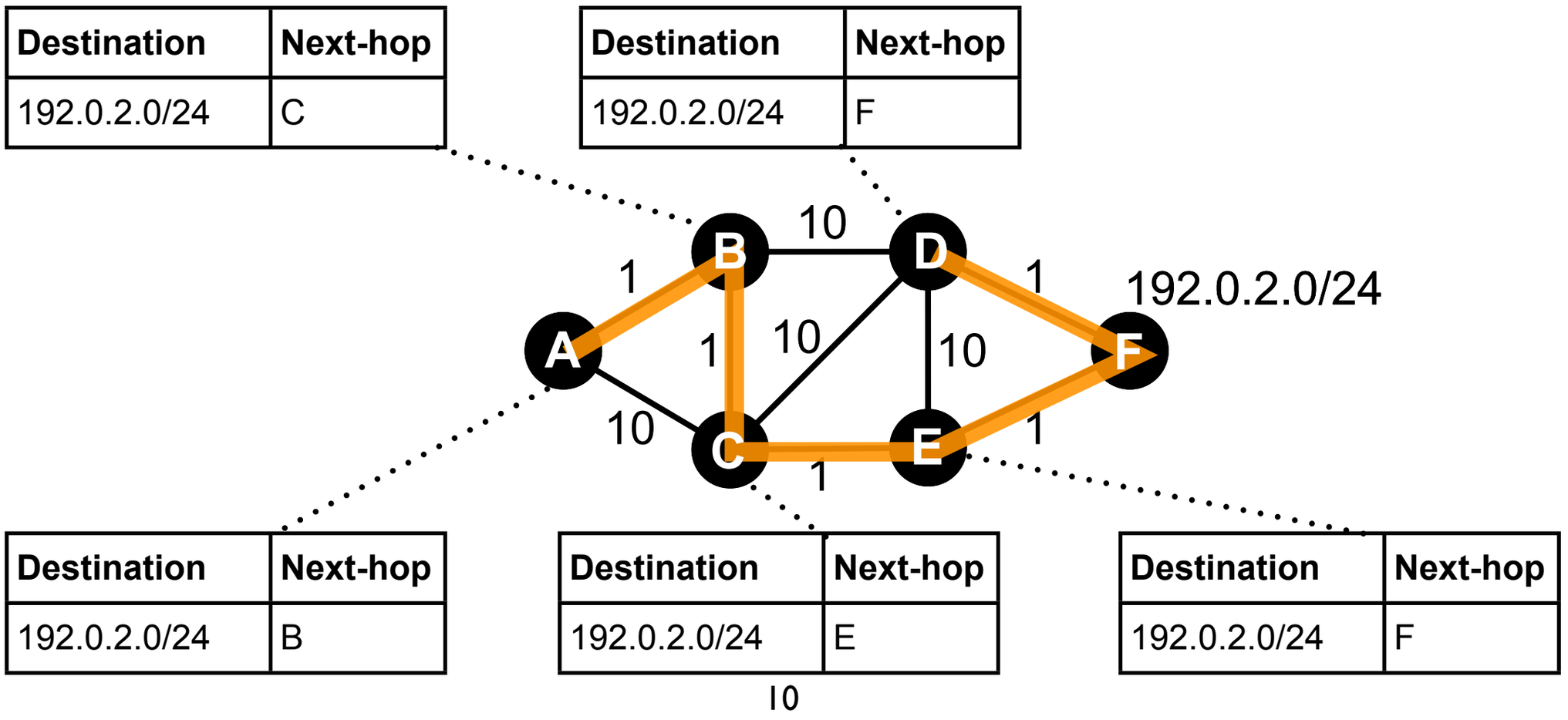
# Selection of the path (contd.)

- The selection of the best path is coherent between routers thanks to the routing algorithm
  - avoid blackholes
  - avoid loops



# Selection of the path (contd.)

- Each router maintains a routing table that summaries the path to reach the destination

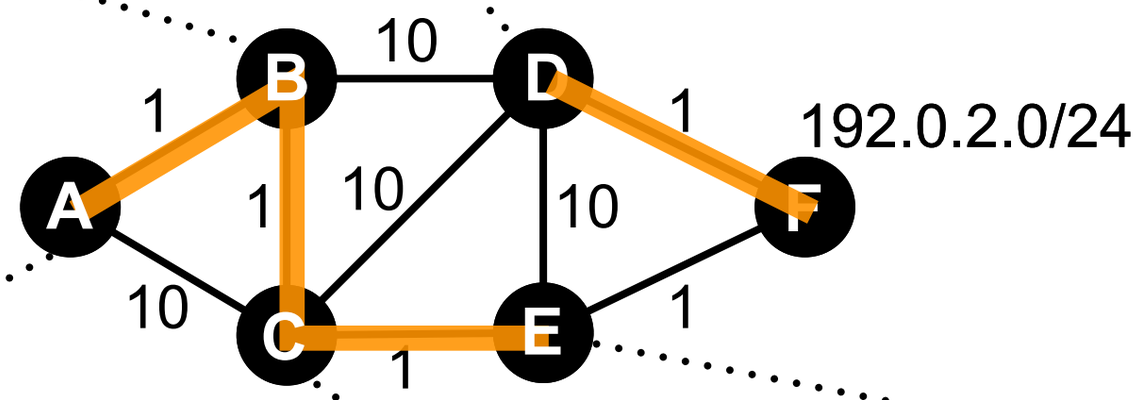


# Example of blackhole

- Blackhole: at least one path exists to reach the destination but no route allows to reach it

| Destination | Next-hop |
|---|---|
| 192.0.2.0/24 | C |

| Destination | Next-hop |
|---|---|
| 192.0.2.0/24 | F |



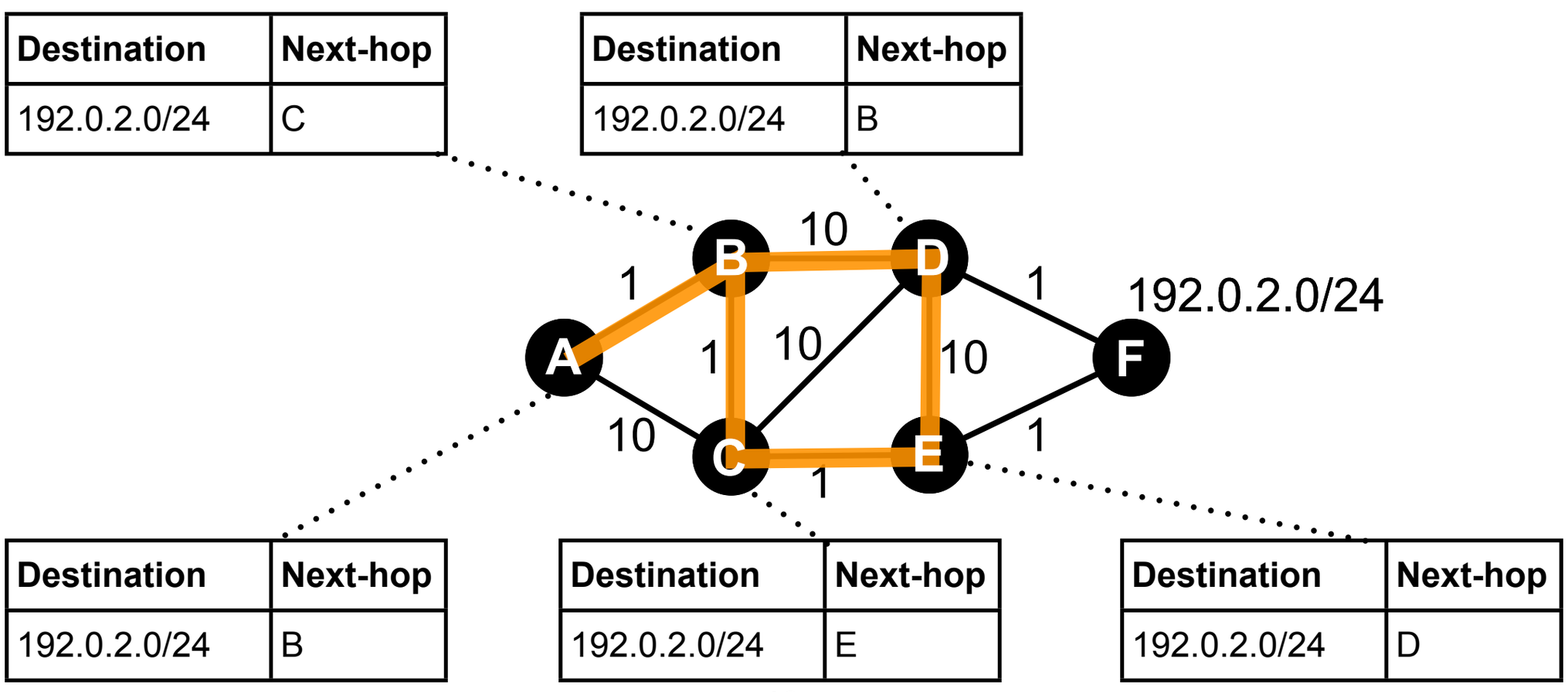

| Destination | Next-hop |
|---|---|
| 192.0.2.0/24 | B |

| Destination | Next-hop |
|---|---|
| 192.0.2.0/24 | E |

| Destination | Next-hop |
|---|---|
| | |

# Example of loop

- Loop: a packet crosses several times the same router

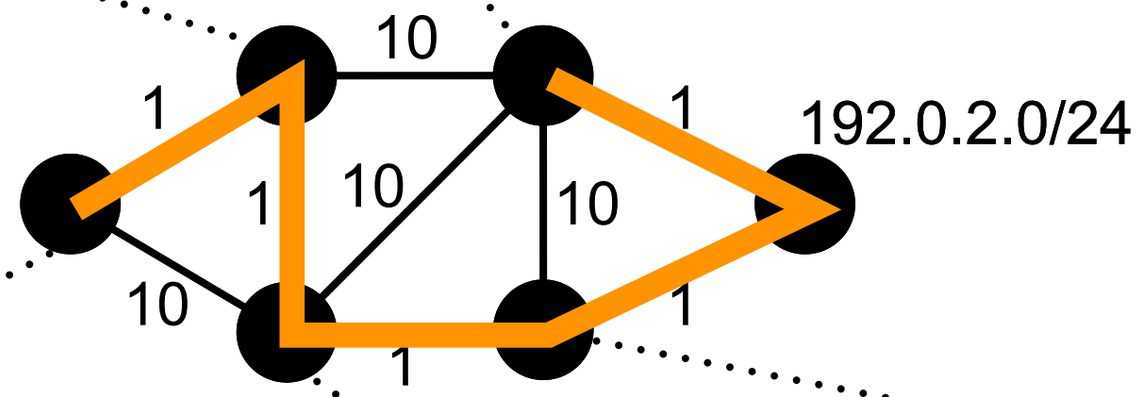


# Construction of the forwarding table

- Determines the interface to which packets must be sent to eventually reach the next-hop

| Destination | Interface |
|---|---|
| 192.0.2.0/24 | South |

| Destination | Interface |
|---|---|
| 192.0.2.0/24 | South-East |




| Destination | Interface |
|---|---|
| 192.0.2.0/24 | North-East |

| Destination | Interface |
|---|---|
| 192.0.2.0/24 | East |

| Destination | Interface |
|---|---|
| 192.0.2.0/24 | North-East |

# Key problem

- How to make correct local decisions?
  - each router must know “something” about global state
- Global state
  - hard to collect
  - inherently large
  - potentially very dynamic
- A routing protocol must intelligently summarize relevant information

# Routing protocol requirements

- Minimize routing table size
  - fast lookup (e.g., finding one route among 10 is faster than among $10^{12}$)
  - potentially less information to exchange
- Minimize number (and size) of exchanges between routers
- Robustness/consistency/stability
  - avoid black holes
  - avoid loops
  - avoid oscillations
- Use best path, for any optimality definition (e.g., cheapest, fastest, more stable, less used...)
- Tradeoffs
  - robustness vs control messages vs routing table size
  - routing table size vs path optimality

# Routing protocol alternatives

- Centralized vs distributed
  - centralization is simple but fragile (what if the central entity fails?)
- Source routing vs hop-by-hop routing
  - how much space can be consumed in on packet to indicate the route to follow?
    - loose source routing allows to define only part of the path
- Stochastic vs deterministic
  - stochastic allows spreads the load and may avoid oscillations
  - deterministic makes troubleshooting easier
- Single-path vs multiple-paths
- Dynamic vs state-independent
  - changing routes based on dynamic metrics like load or delay can cause oscillations

# Centralized routing

- In centralized (or static) routing, the network manager or the network controller computes all the routing tables and install them in each router
- Pros
  - simple to use
  - possibility to optimize routes precisely
- Cons
  - hard to adapt to sudden network changes (e.g., failure)
  - not adapted to dynamic networks (e.g., mesh networks)
- Regain interest with OpenFlow...

# Distributed routing

- In distributed routing, routers exchange message and run a distributed routing algorithm to build their routing tables
- Pros
  - easy to adapt to sudden network changes (e.g., failure)
  - adapted to dynamic networks (e.g., mesh networks)
- Cons
  - more complex to implement than static routing
  - troubleshooting can become difficult
  - risk of transient loops or blackholes
- Distributed routing is the most common technique nowadays
- Two main alternatives
  - Link state routing (LS)
  - Distance vector routing (DV)

# Link state routing

# LS principle

- Each router knows the entire topology and computes, locally, shortest paths
  - routes are computed independently
    - loop free if same view of the topology and same routing algorithm
- Two main functions
  - topology dissemination
  - route computation

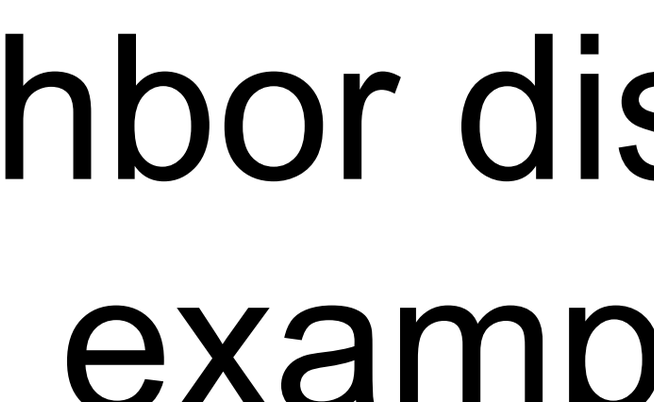
# Topology dissemination (neighbor discovery)

- Routers discover their direct neighbors
  - each link is associated a cost (aka weight)
  - periodically send a HELLO message on each link to determine if the link is up
  - upon HELLO message reception, the receiver sends a HELLO message in the reverse direction

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| |
| |
| |

| B.Neighbors |
| --- |
| B:0 |
| |




| F.Neighbors |
| --- |
| F:0 |
| |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| |
| |
| |

| B.Neighbors |
| --- |
| B:0 |
| |

D: HELLO


| F.Neighbors |
| --- |
| F:0 |
| |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| |
| |
| |

| B.Neighbors |
| --- |
| B:0 |
| |

D: HELLO

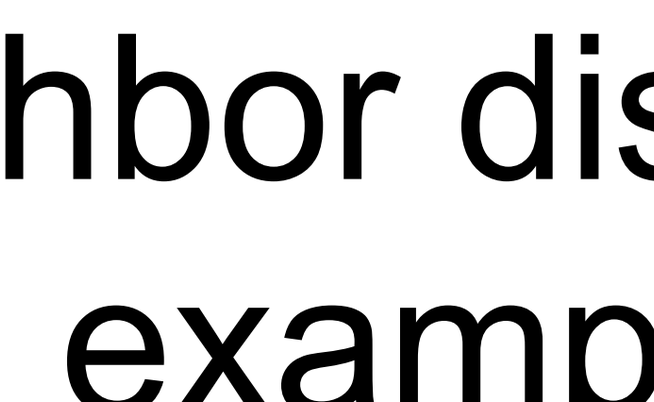

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| |
| |
| |

| B.Neighbors |
| --- |
| B:0 |
| |




| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| B.Neighbors |
| --- |
| B:0 |
| |

| D.Neighbors |
| --- |
| D:0 |
| |
| |
| |

F: HELLO


| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| B.Neighbors |
| --- |
| B:0 |
| |

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| |
| |

F: HELLO

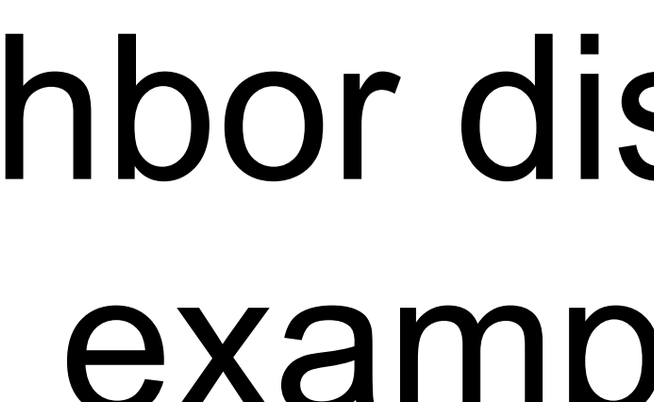

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| |
| |

| B.Neighbors |
| --- |
| B:0 |
| |



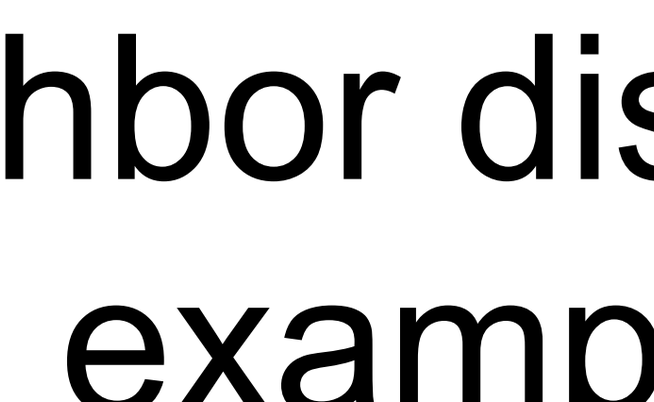

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| |
| |

| B.Neighbors |
| --- |
| B:0 |
| |



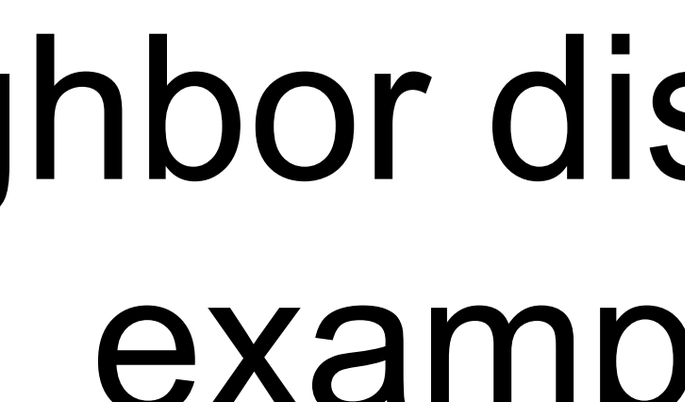

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| B.Neighbors |
| --- |
| B:0 |
| |

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |



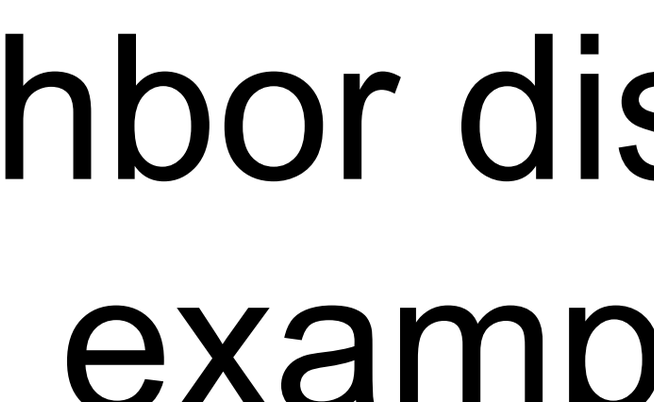

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| B.Neighbors |
| --- |
| B:0 |
| |

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |



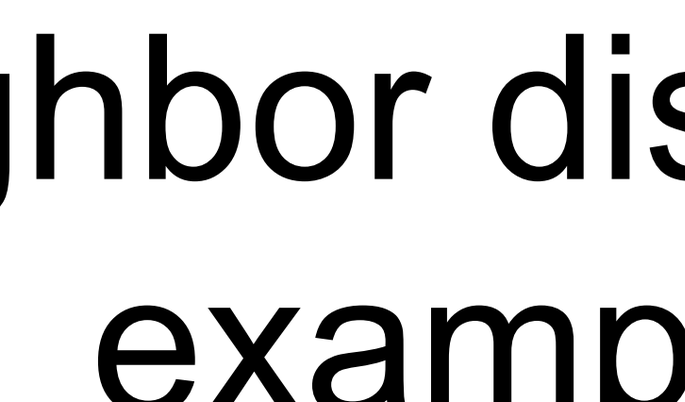

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| |
| |

# Neighbor discovery example

| D.Neighbors |
|---|
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
|---|
| B:0 |
| |



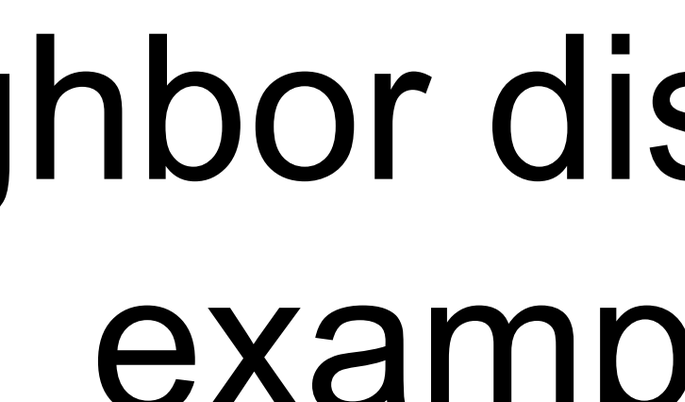

| F.Neighbors |
|---|
| F:0 |
| D:1 |
| |

| E.Neighbors |
|---|
| E:0 |
| |
| |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
| --- |
| B:0 |
| |



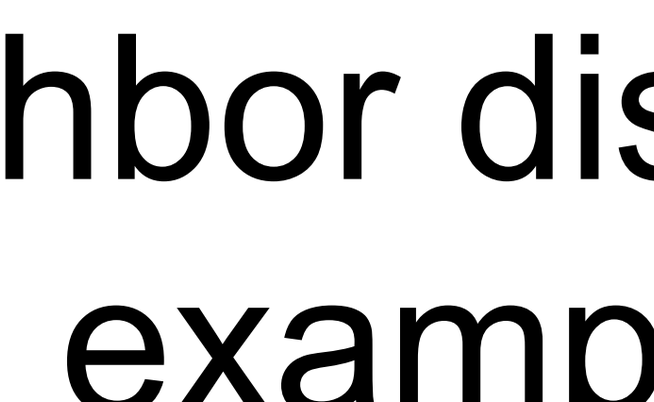

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
| --- |
| B:0 |
| |




| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| |

# Neighbor discovery example

| B.Neighbors |
| --- |
| B:0 |
| |

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

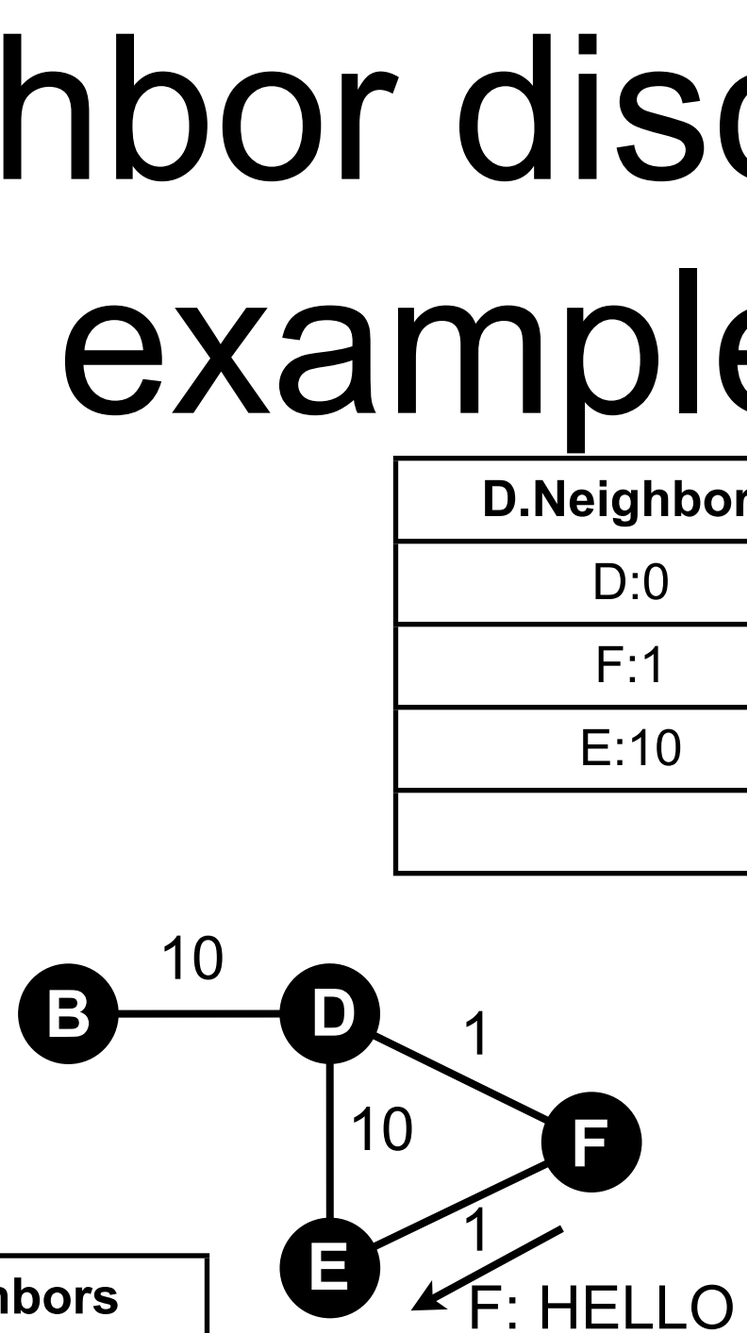

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
| --- |
| B:0 |
| |



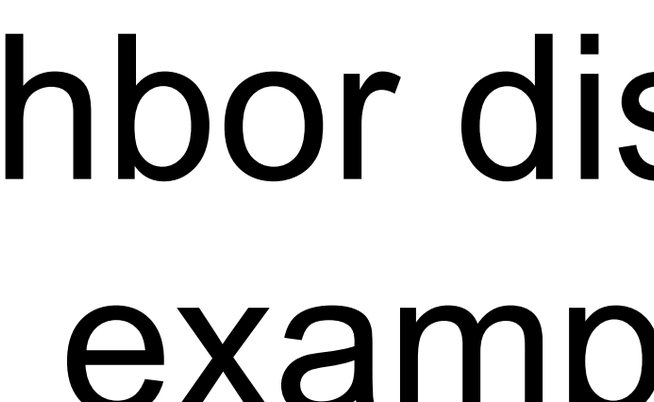

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
| --- |
| B:0 |
| |




| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| B.Neighbors |
| --- |
| B:0 |
| |

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |


| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
| --- |
| B:0 |
| |

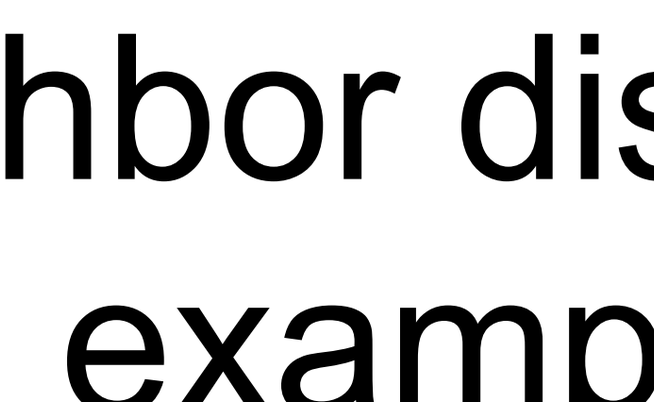

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| E:1 |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
| --- |
| B:0 |
| |



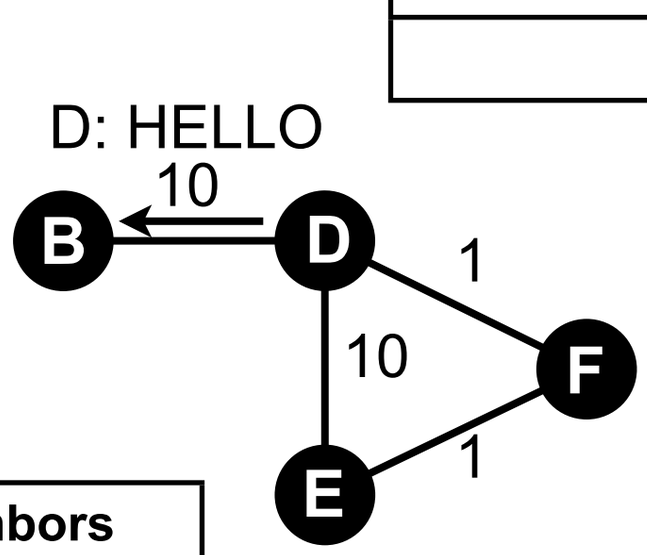

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| E:1 |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
| --- |
| B:0 |
| |



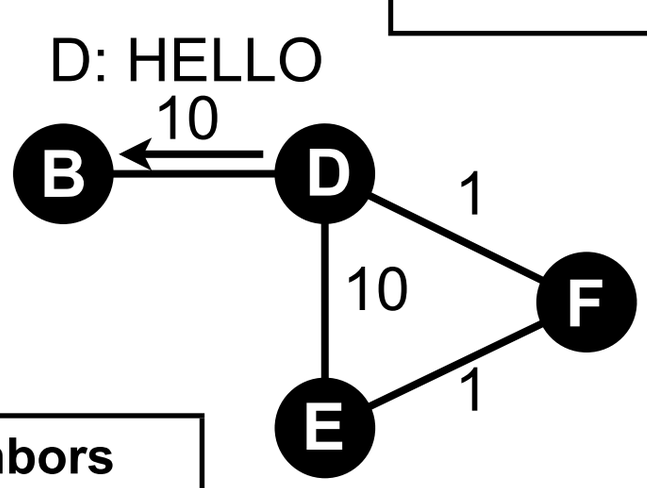

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| E:1 |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
| --- |
| B:0 |
| D:10 |



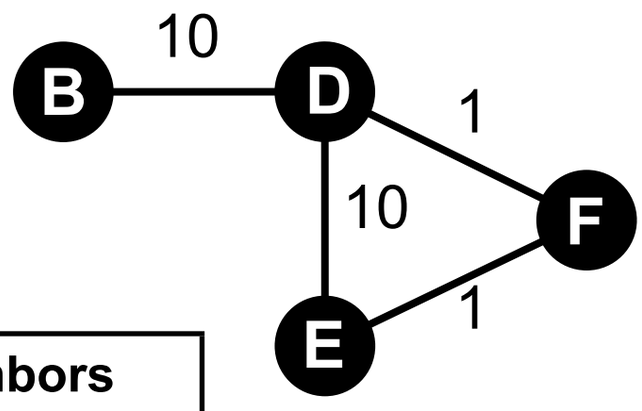

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| E:1 |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
| --- |
| B:0 |
| D:10 |



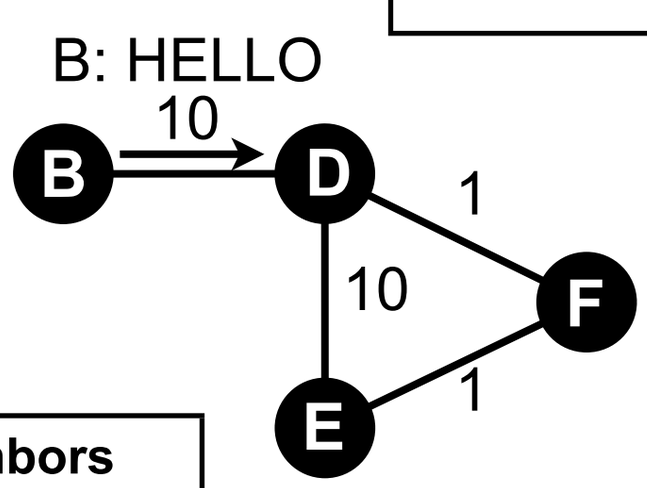

| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| E:1 |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| D.Neighbors |
|---|
| D:0 |
| F:1 |
| E:10 |
| |

| B.Neighbors |
|---|
| B:0 |
| D:10 |



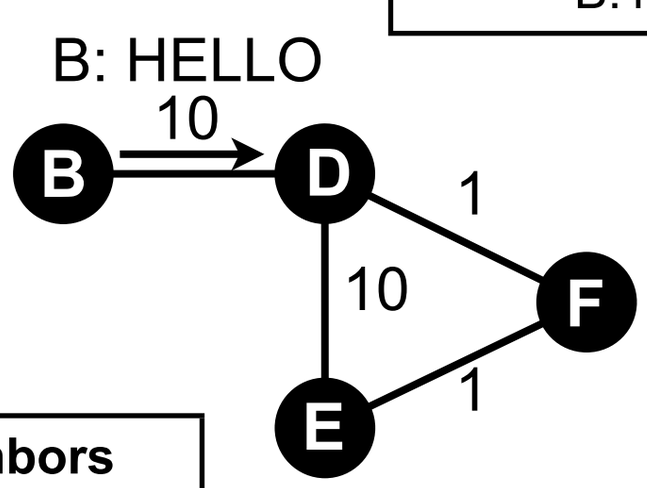

| F.Neighbors |
|---|
| F:0 |
| D:1 |
| E:1 |

| E.Neighbors |
|---|
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| D.Neighbors |
|---|
| D:0 |
| F:1 |
| E:10 |
| B:10 |

| B.Neighbors |
|---|
| B:0 |
| D:10 |



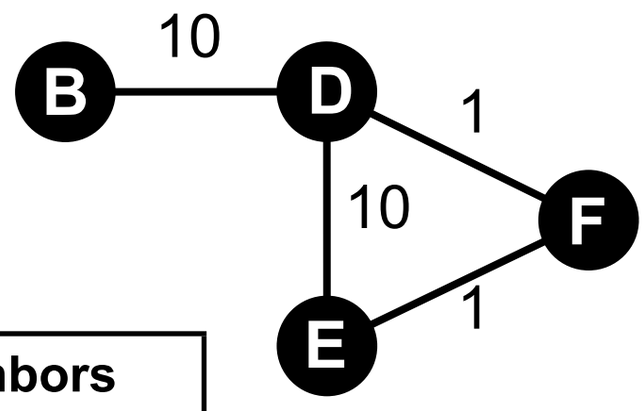

| F.Neighbors |
|---|
| F:0 |
| D:1 |
| E:1 |

| E.Neighbors |
|---|
| E:0 |
| D:10 |
| F:1 |

# Neighbor discovery example

| B.Neighbors |
| --- |
| B:0 |
| D:10 |

| D.Neighbors |
| --- |
| D:0 |
| F:1 |
| E:10 |
| B:10 |




| F.Neighbors |
| --- |
| F:0 |
| D:1 |
| E:1 |

| E.Neighbors |
| --- |
| E:0 |
| D:10 |
| F:1 |

# Topology dissemination (LSP flooding)

- Routers build their local part of the network map upon HELLO message reception
- An additional mechanism is necessary for routers to be able to construct the map of the entire topology
  - every router summarizes its local topology inside a Link State Packet (LSP) that contains
    - the router identification
    - its list of <neighbor, cost> pairs
  - the LSP is flooded to all its outgoing interfaces
  - when a router receives a LSP
    - it stores it in its Link State Packet  DataBase (LSPDB)
    - it floods it to all its outgoing interface (but the one it received the LSP from)

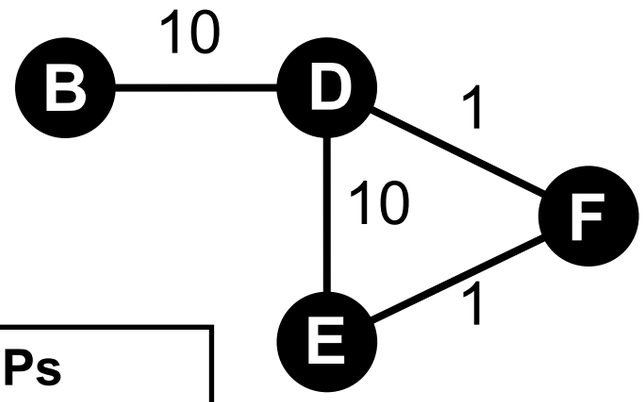
# LSP flooding loop avoidance

- How to avoid a router to flood a LSP that it already flooded?
- When a router generates a LSP, it tags it with a (local) sequence number
  - the sequence number is incremented every time the router generates a new sequence number
- When a router receives a LSP, it stores the LSP and its sequence number in its LSPDB
  - if the sequence number is higher than the sequence number in the LSPDB, then the LSP is flooded. Otherwise, it is not flooded

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | |
| F:1 | |
| E:10 | |
| B:10 | |



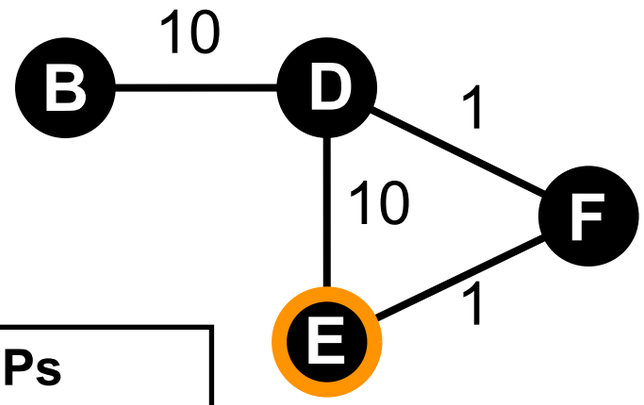

| E.Neighbors | E.LSPs |
|---|---|
| E:0 | |
| D:10 | |
| F:1 | |
| | |

| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | |
| D:1 | |
| E:1 | |
| | |

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | |
| F:1 | |
| E:10 | |
| B:10 | |



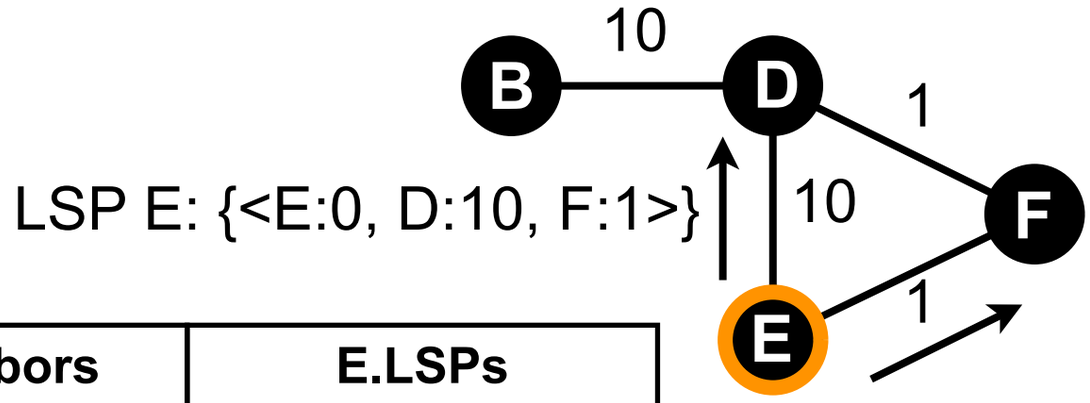

| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| F:1 | |
| | |

| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | |
| D:1 | |
| E:1 | |
| | |

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | |
| F:1 | |
| E:10 | |
| B:10 | |




| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| F:1 | |
| | |

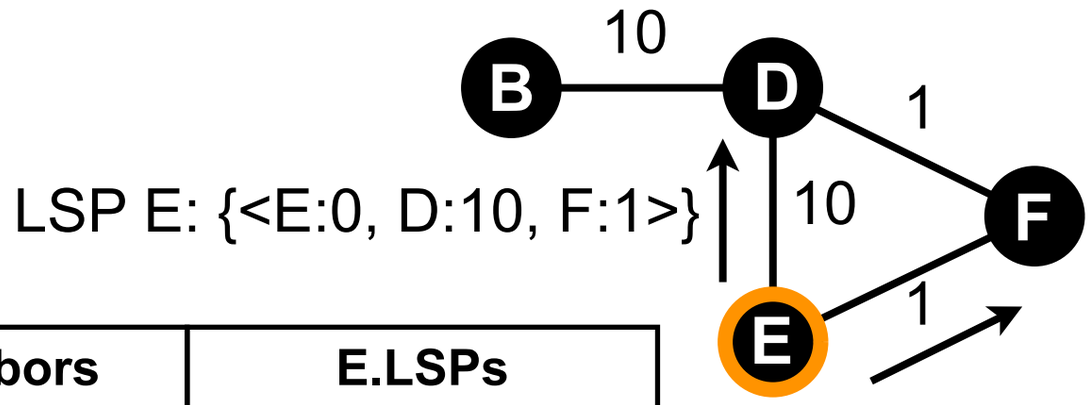
| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | |
| D:1 | |
| E:1 | |
| | |

LSP E: {<E:0, D:10, F:1>}

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | E: {<E:0, D:10, F:1>} |
| F:1 | |
| E:10 | |
| B:10 | |



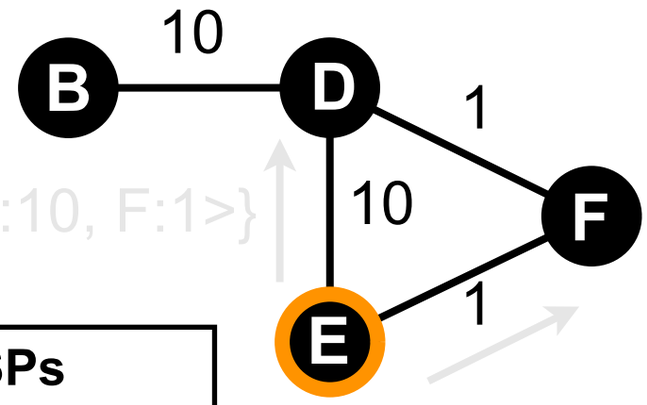

| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| F:1 | |
| | |

| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | E: {<E:0, D:10, F:1>} |
| D:1 | |
| E:1 | |
| | |

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | E: {<E:0, D:10, F:1>} |
| F:1 | |
| E:10 | |
| B:10 | |




| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| F:1 | |
| | |

| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | E: {<E:0, D:10, F:1>} |
| D:1 | |
| E:1 | |
| | |

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | E: {<E:0, D:10, F:1>} |
| F:1 | |
| E:10 | |
| B:10 | |


| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | E: {<E:0, D:10, F:1>} |
| D:1 | |
| E:1 | |
| | |

| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| F:1 | |
| | |

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | E: {<E:0, D:10, F:1>} |
| F:1 | |
| E:10 | |
| B:10 | |


| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | E: {<E:0, D:10, F:1>} |
| D:1 | |
| E:1 | |
| | |

| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| F:1 | |
| | |

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | E: {<E:0, D:10, F:1>} |
| F:1 | |
| E:10 | |
| B:10 | |

LSP E: {<E:0, D:10, F:1>}


LSP E: {<E:0, D:10, F:1>}

LSP E: {<E:0, D:10, F:1>}

| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | E: {<E:0, D:10, F:1>} |
| D:1 | |
| E:1 | |
| | |

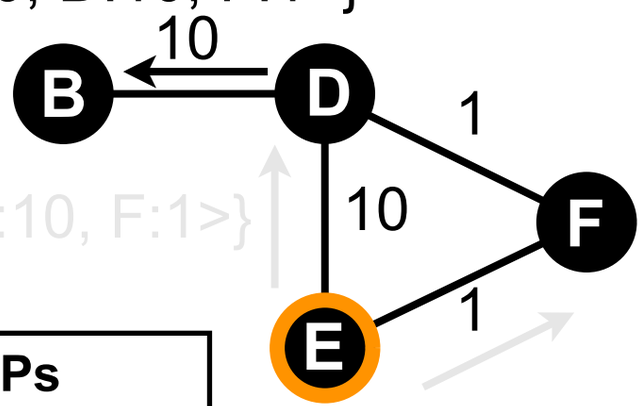
| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| F:1 | |
| | |

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | E: {<E:0, D:10, F:1>} |
| F:1 | |
| E:10 | |
| B:10 | |

LSP E: {<E:0, D:10, F:1>}




LSP E: {<E:0, D:10, F:1>}

LSP E: {<E:0, D:10, F:1>}

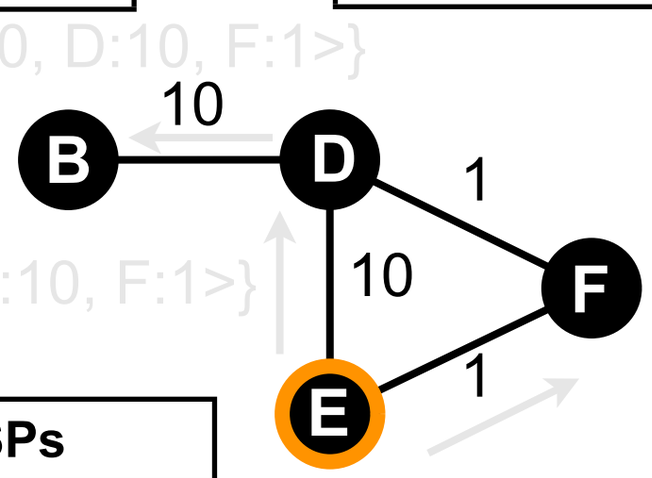
| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | E: {<E:0, D:10, F:1>} |
| D:1 | |
| E:1 | |
| | |

| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| F:1 | |
| | |

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| | |
| | |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | E: {<E:0, D:10, F:1>} |
| F:1 | |
| E:10 | |
| B:10 | |



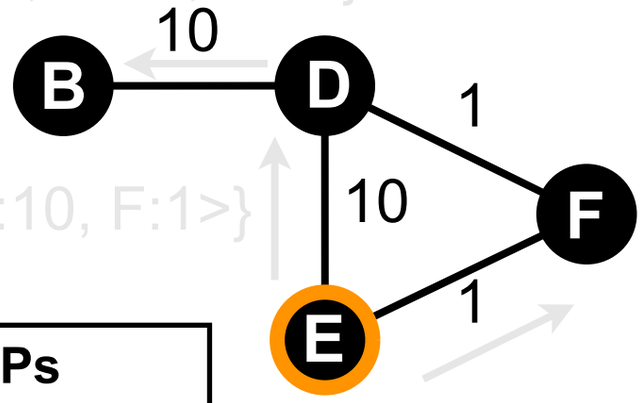

| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | |
| F:1 | |
| | |

| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | E: {<E:0, D:10, F:1>} |
| D:1 | |
| E:1 | |
| | |

# LSP flooding example

| B.Neighbors | B.LSPDB |
|---|---|
| B:0 | E: {<E:0, D:10, F:1>} |
| D:10 | B: {<B:0, D:10>} |
| | D: {<D:0, F:1, E:10,B:10>} |
| | F: {<F:0, D:1, E:1>} |

| D.Neighbors | D.LSPDB |
|---|---|
| D:0 | E: {<E:0, D:10, F:1>} |
| F:1 | B: {<B:0, D:10>} |
| E:10 | D: {<D:0, F:1, E:10,B:10>} |
| B:10 | F: {<F:0, D:1, E:1>} |




| E.Neighbors | E.LSPs |
|---|---|
| E:0 | E: {<E:0, D:10, F:1>} |
| D:10 | B: {<B:0, D:10>} |
| F:1 | D: {<D:0, F:1, E:10,B:10>} |
| | F: {<F:0, D:1, E:1>} |

| F.Neighbors | F.LSPDB |
|---|---|
| F:0 | E: {<E:0, D:10, F:1>} |
| D:1 | B: {<B:0, D:10>} |
| E:1 | D: {<D:0, R:1, E:10,B:10>} |
| | F: {<F:0, D:1, E:1>} |

# Route computation

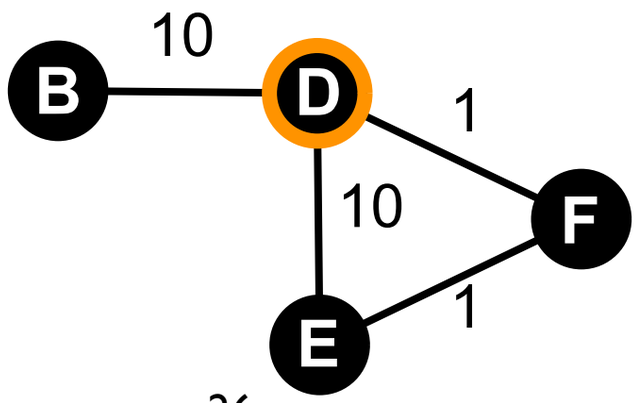
- Each router constructs a graph representation of the network by combining LSPs in its LSPDB
  - LSPs define weighted edges
- Each router computes the shortest spanning tree on that graph, rooted on itself
  - using Dijkstra algorithm
- Each router constructs its routing then forwarding tables based on the spanning tree

| D.LSPDB |
| --- |
| E: {<E:0, D:10, F:1>} |
| B: {<B:0, D:10>} |
| D: {<D:0, F:1, E:10,B:10>} |
| F: {<F:0, D:1, E:1>} |

# Route computation

- Each router constructs a graph representation of the network by combining LSPs in its LSPDB
  - LSPs define weighted edges
- Each router computes the shortest spanning tree on that graph, rooted on itself
  - using Dijkstra algorithm
- Each router constructs its routing then forwarding tables based on the spanning tree

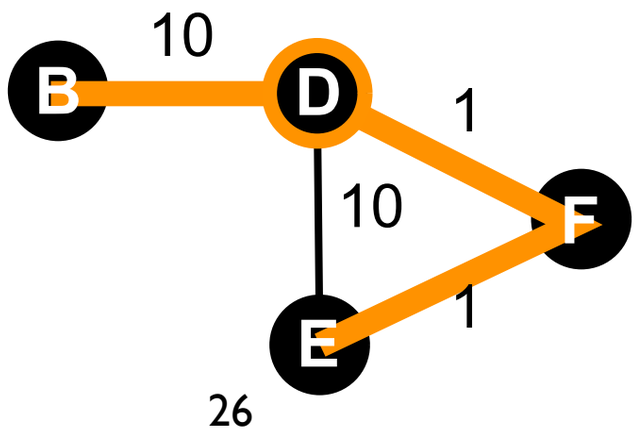
| D.LSPDB |
| --- |
| E: {<E:0, D:10, F:1>} |
| B: {<B:0, D:10>} |
| D: {<D:0, F:1, E:10,B:10>} |
| F: {<F:0, D:1, E:1>} |



# Route computation

- Each router constructs a graph representation of the network by combining LSPs in its LSPDB
  - LSPs define weighted edges
- Each router computes the shortest spanning tree on that graph, rooted on itself
  - using Dijkstra algorithm
- Each router constructs its routing then forwarding tables based on the spanning tree

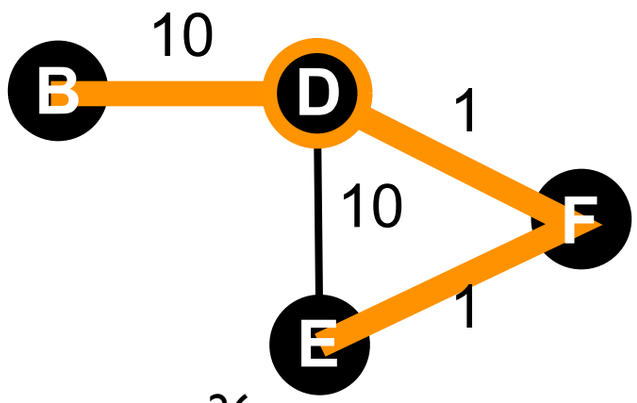
| D.LSPDB |
| --- |
| E: {<E:0, D:10, F:1>} |
| B: {<B:0, D:10>} |
| D: {<D:0, F:1, E:10,B:10>} |
| F: {<F:0, D:1, E:1>} |



# Route computation

- Each router constructs a graph representation of the network by combining LSPs in its LSPDB
  - LSPs define weighted edges
- Each router computes the shortest spanning tree on that graph, rooted on itself
  - using Dijkstra algorithm
- Each router constructs its routing then forwarding tables based on the spanning tree

| D.LSPDB |
|---|
| E: {<E:0, D:10, F:1>} |
| B: {<B:0, D:10>} |
| D: {<D:0, F:1, E:10,B:10>} |
| F: {<F:0, D:1, E:1>} |

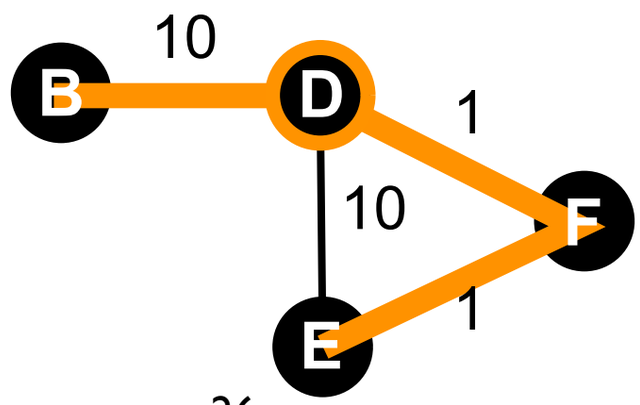



| Destination | Next-hop | Interface |
|---|---|---|
| B | | |
| D | | |
| E | | |
| F | | |

# Route computation

- Each router constructs a graph representation of the network by combining LSPs in its LSPDB
  - LSPs define weighted edges
- Each router computes the shortest spanning tree on that graph, rooted on itself
  - using Dijkstra algorithm
- Each router constructs its routing then forwarding tables based on the spanning tree

| D.LSPDB |
|---|
| E: {<E:0, D:10, F:1>} |
| B: {<B:0, D:10>} |
| D: {<D:0, F:1, E:10,B:10>} |
| F: {<F:0, D:1, E:1>} |

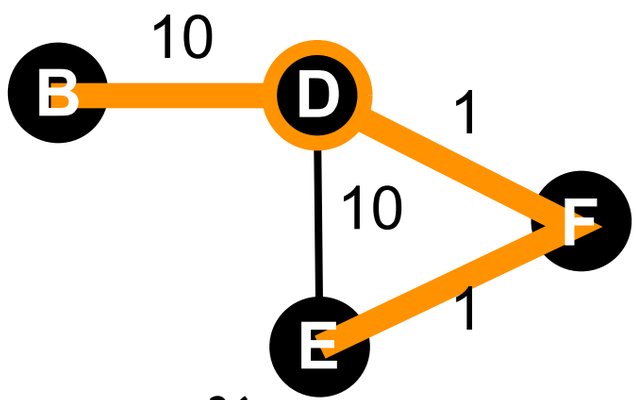



| Destination | Next-hop | Interface |
|---|---|---|
| B | B | |
| D | local | |
| E | F | |
| F | F | |

# Route computation

- Each router constructs a graph representation of the network by combining LSPs in its LSPDB
  - LSPs define weighted edges
- Each router computes the shortest spanning tree on that graph, rooted on itself
  - using Dijkstra algorithm
- Each router constructs its routing then forwarding tables based on the spanning tree

| D.LSPDB |
|---|
| E: {<E:0, D:10, F:1>} |
| B: {<B:0, D:10>} |
| D: {<D:0, F:1, E:10,B:10>} |
| F: {<F:0, D:1, E:1>} |




| Destination | Next-hop | Interface |
|---|---|---|
| B | B | West |
| D | local | local |
| E | F | South-East |
| F | F | South-East |

# Distance vector routing

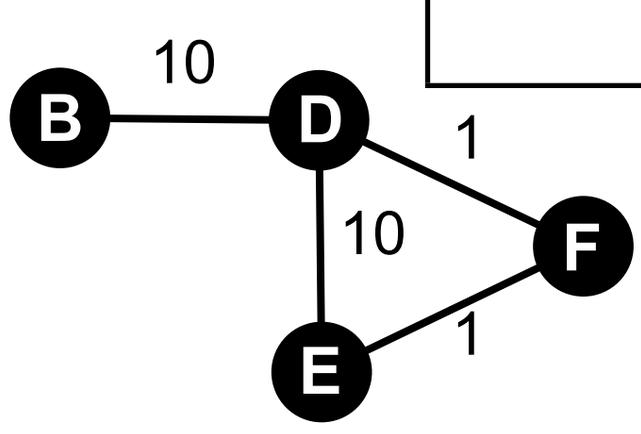
# DV principle

- At startup, a router only knows about itself (and the cost of each of its directly connected links)
- Routers learn the distance between their neighbors and every other node
  - each router sends to all its neighbors the best idea of the distance from itself to every node that it knows
    - the routing table is summarized in distance vectors that contain, for every node that the router knows,
      - the destination (i.e., node)
      - and the distance between the router and the destination
  - when a router receives a distance vector, it updates its estimation of the distance to the destinations in the distance vector
    - if distances changed since the previous estimation, it advertises the changes to its neighbors by sending updated distance vectors
- No view of the entire topology
  - route computation is entirely distributed

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| | | |
| | | |
| | | |



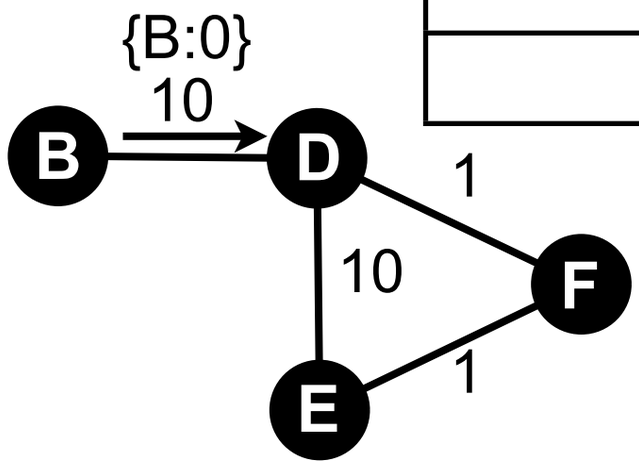

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| | | |
| | | |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| | | |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| | | |
| | | |
| | | |
| | | |



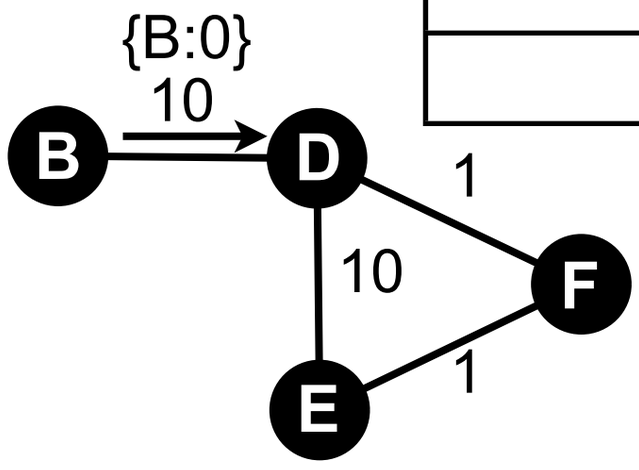

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| | | |
| | | |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| | | |
| | | |




| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| | | |
| | | |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| | | |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| | | |
| | | |

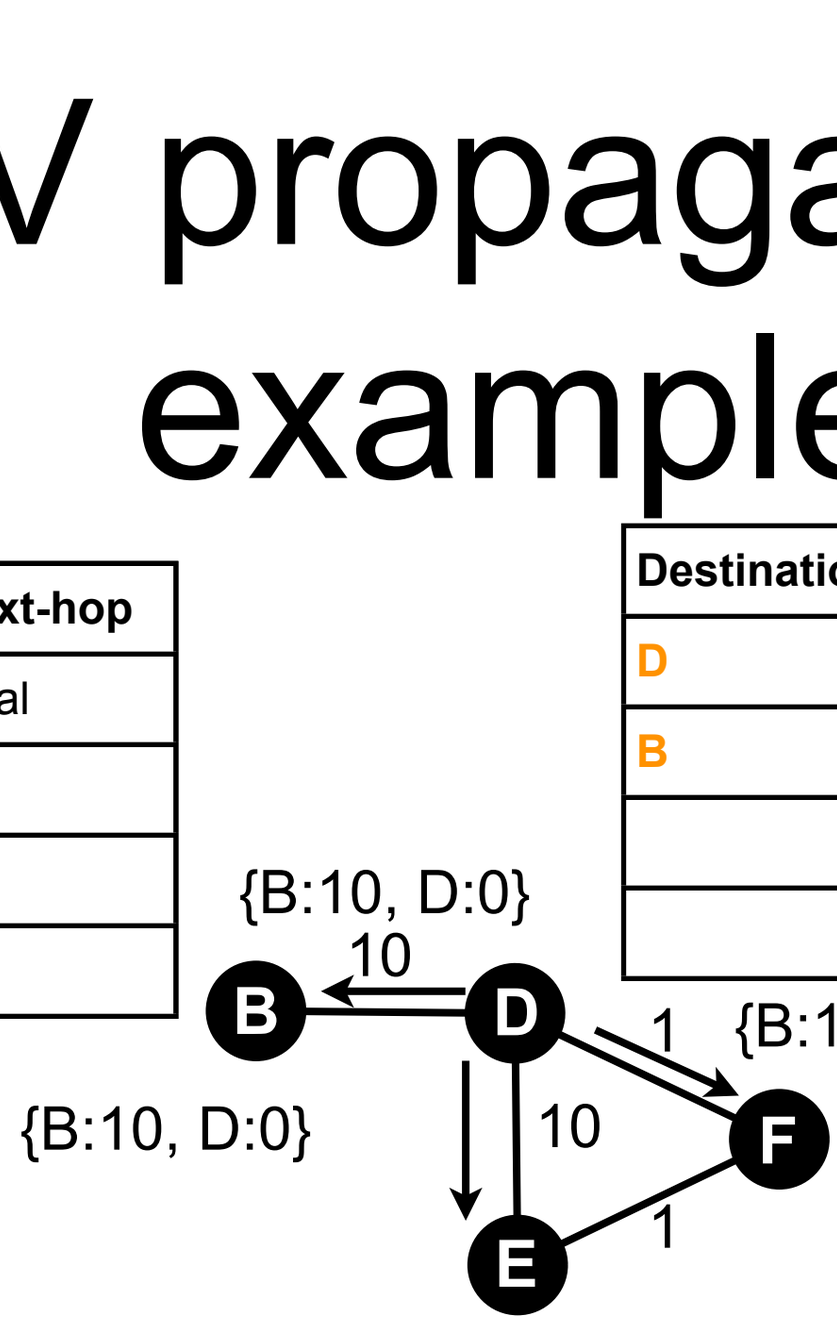

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| | | |
| | | |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| **D** | **10** | **D** |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **D** | **0** | **local** |
| **B** | **10** | **B** |
| | | |
| | | |



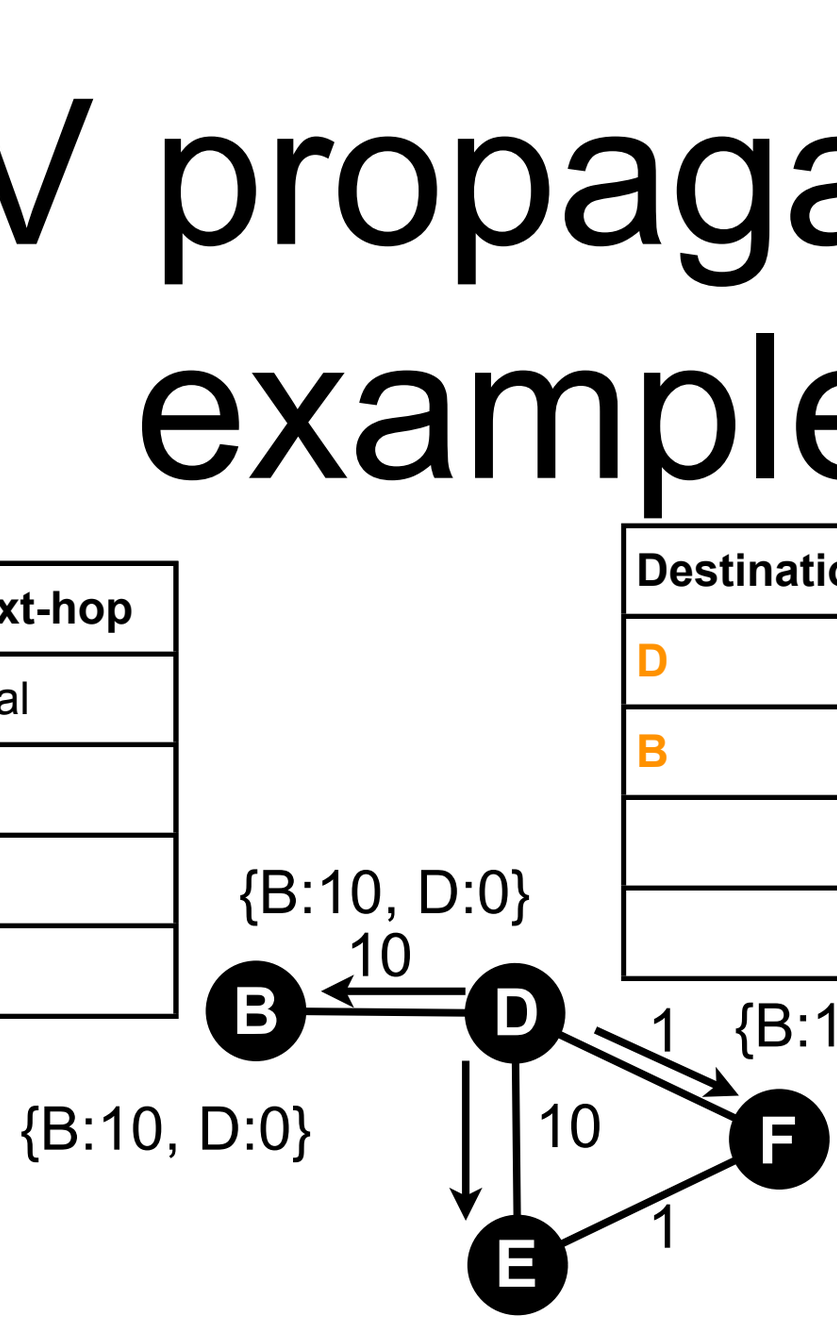

| Destination | Distance | Next-hop |
|---|---|---|
| **F** | **0** | **local** |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **E** | **0** | **local** |
| | | |
| | | |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| **D** | **10** | **D** |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **D** | **0** | **local** |
| **B** | **10** | **B** |
| | | |
| | | |



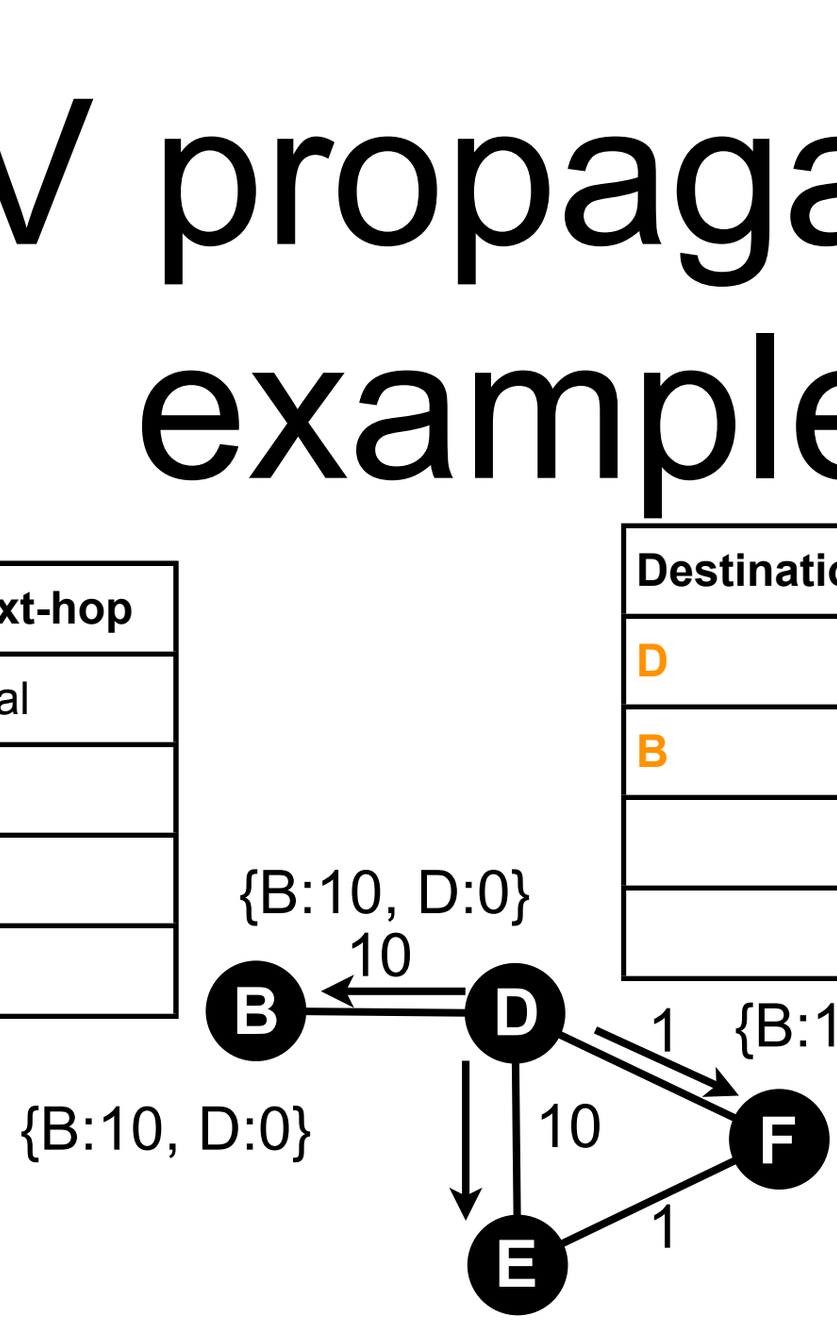

| Destination | Distance | Next-hop |
|---|---|---|
| **F** | **0** | **local** |
| | | |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **E** | **0** | **local** |
| **B** | **20** | **D** |
| **D** | **10** | **D** |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| **D** | **10** | **D** |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **D** | **0** | **local** |
| **B** | **10** | **B** |
| | | |
| | | |




| Destination | Distance | Next-hop |
|---|---|---|
| **F** | **0** | **local** |
| **B** | **11** | **D** |
| **D** | **1** | **D** |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **E** | **0** | **local** |
| **B** | **20** | **D** |
| **D** | **10** | **D** |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| **D** | **10** | **D** |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| | | |
| | | |

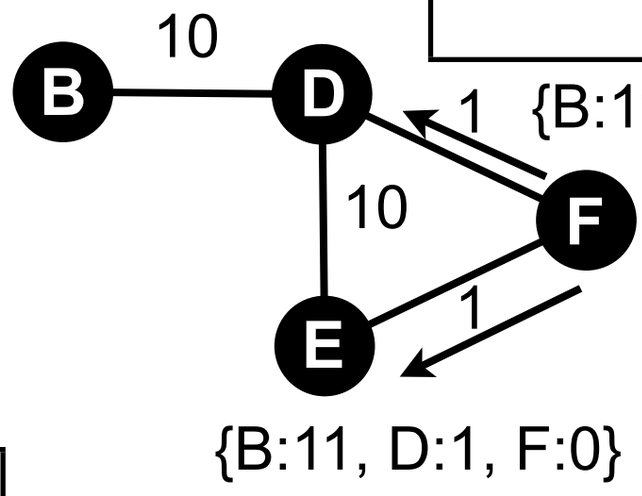

| Destination | Distance | Next-hop |
|---|---|---|
| **F** | **0** | **local** |
| **B** | **11** | **D** |
| **D** | **1** | **D** |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **E** | **0** | **local** |
| **B** | **20** | **D** |
| **D** | **10** | **D** |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| | | |
| | | |



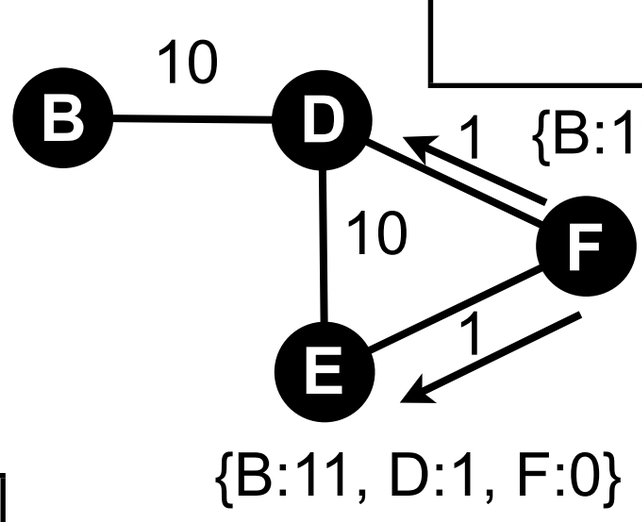

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| B | 20 | D |
| D | 10 | D |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| **F** | **1** | **F** |
| | | |



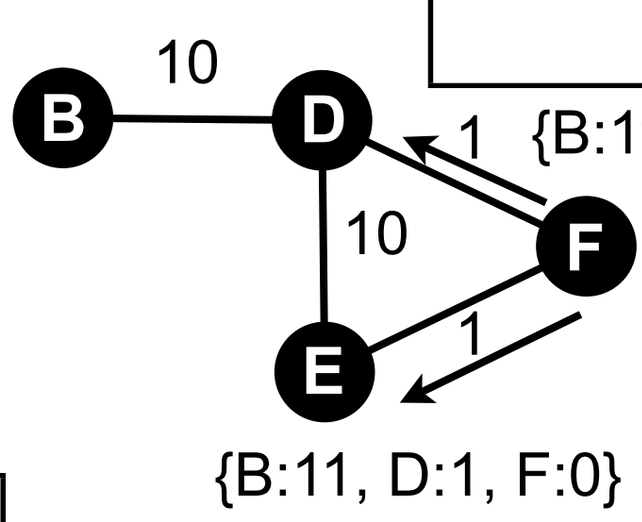

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| B | 20 | D |
| D | 10 | D |
| | | |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| **F** | **1** | **F** |
| | | |



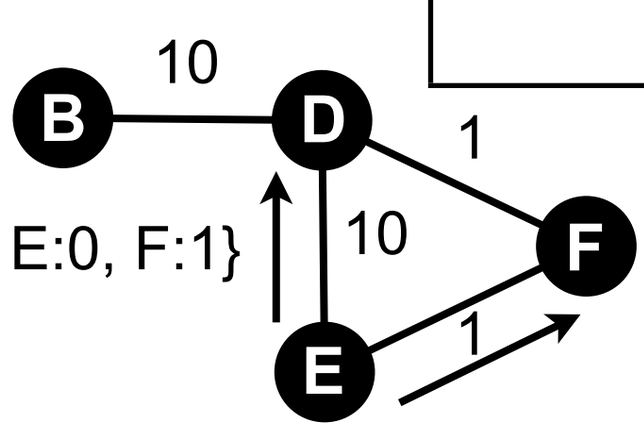

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **E** | **0** | **local** |
| **B** | **12** | **F** |
| **D** | **2** | **F** |
| **F** | **1** | **F** |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| **F** | **1** | **F** |
| | | |



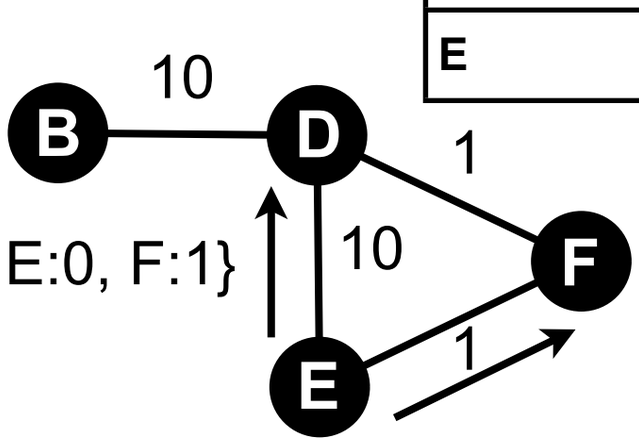

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **E** | **0** | **local** |
| **B** | **12** | **F** |
| **D** | **2** | **F** |
| **F** | **1** | **F** |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| **F** | **1** | **F** |
| **E** | **10** | **E** |



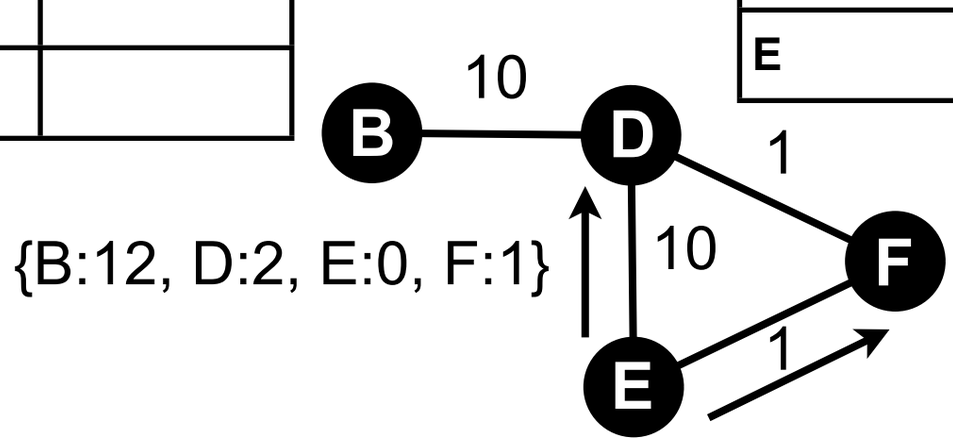

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| **E** | **0** | **local** |
| **B** | **12** | **F** |
| **D** | **2** | **F** |
| **F** | **1** | **F** |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| **F** | **1** | **F** |
| **E** | **10** | **E** |



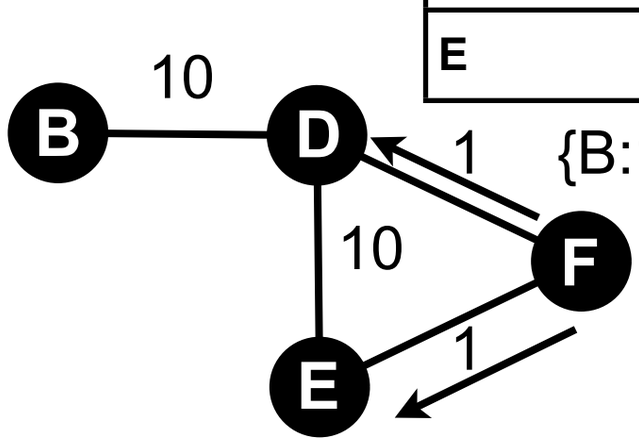

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| **E** | **1** | **E** |

| Destination | Distance | Next-hop |
|---|---|---|
| **E** | **0** | **local** |
| **B** | **12** | **F** |
| **D** | **2** | **F** |
| **F** | **1** | **F** |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| **F** | **1** | **F** |
| **E** | **10** | **E** |



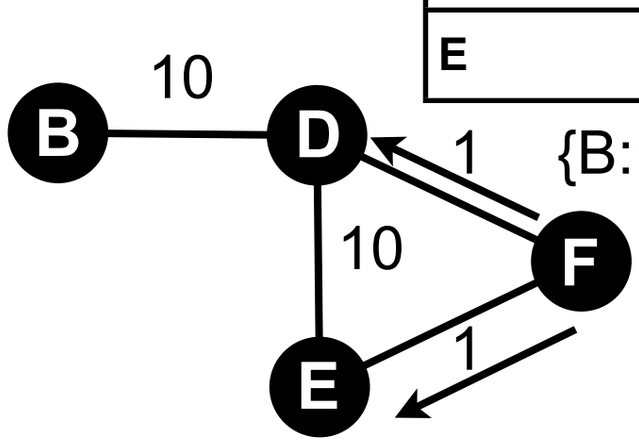

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| **E** | **1** | **E** |

| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| B | 12 | F |
| D | 2 | F |
| F | 1 | F |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| | | |
| | | |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| **F** | **1** | **F** |
| **E** | **2** | **F** |




| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| **E** | **1** | **E** |

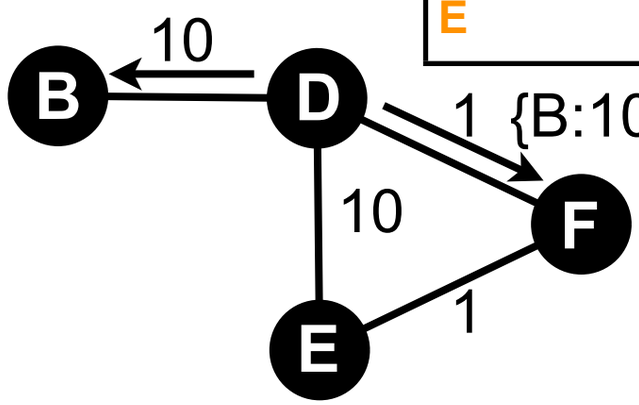
| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| B | 12 | F |
| D | 2 | F |
| F | 1 | F |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| | | |
| | | |

{B:10, D:0, E:2, F:1}

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| **F** | **1** | **F** |
| **E** | **2** | **F** |




| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| E | 1 | E |

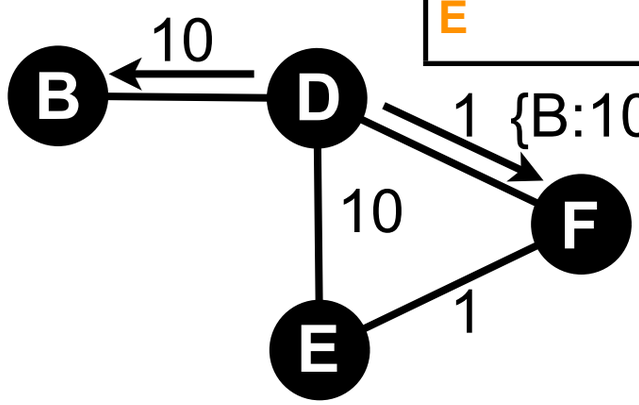
| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| B | 12 | F |
| D | 2 | F |
| F | 1 | F |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| **E** | **12** | **D** |
| **F** | **11** | **D** |

{B:10, D:0, E:2, F:1}

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| **F** | **1** | **F** |
| **E** | **2** | **F** |




| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| E | 1 | E |

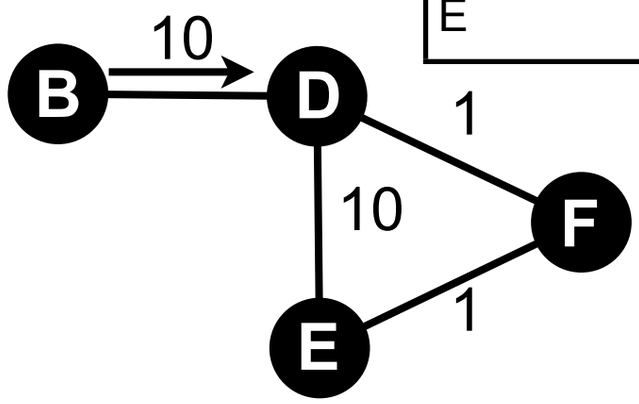
| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| B | 12 | F |
| D | 2 | F |
| F | 1 | F |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| **E** | **12** | **D** |
| **F** | **11** | **D** |

{B:0, D:10,
E:12, F:11}

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| F | 1 | F |
| E | 2 | F |



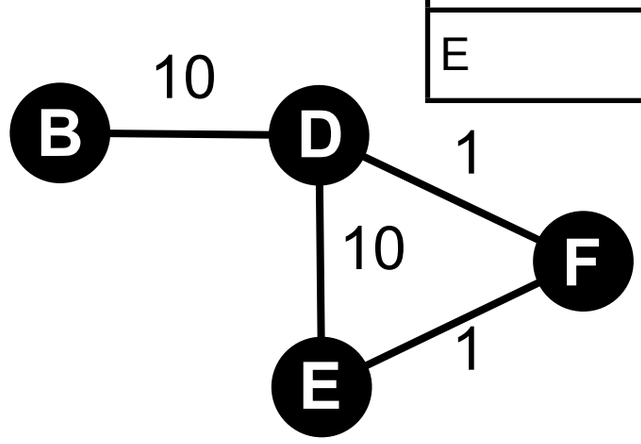

| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| E | 1 | E |

| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| B | 12 | F |
| D | 2 | F |
| F | 1 | F |

# DV propagation example

| Destination | Distance | Next-hop |
|---|---|---|
| B | 0 | local |
| D | 10 | D |
| E | 12 | D |
| F | 11 | D |

| Destination | Distance | Next-hop |
|---|---|---|
| D | 0 | local |
| B | 10 | B |
| F | 1 | F |
| E | 2 | F |




| Destination | Distance | Next-hop |
|---|---|---|
| F | 0 | local |
| B | 11 | D |
| D | 1 | D |
| E | 1 | E |

| Destination | Distance | Next-hop |
|---|---|---|
| E | 0 | local |
| B | 12 | F |
| D | 2 | F |
| F | 1 | F |

# Why does it work?

- Every node knows exactly the cost to reach its direct neighbors
- This true knowledge is propagated to the neighbors
  - and subsequently to the neighbors of the neighbors
  - until every routing table eventually incorporates the costs
- This has been proved by Bellman and Ford in 1957 (more details during next session)

# Forwarding in Ethernet networks

# Forwarding in Ethernet networks

- Large Ethernet networks are constituted of the interconnection of hubs and/or switches
  - hubs and switches are devices composed of several physical layer ports
  - hub (repeaters) operate at the physical layer
  - switches (bridges) operate at the datalink layer
- Duplication of devices and links is often used to make the network more robust

# Large Ethernet networks: hubs

- Hubs work at the physical layer (no intelligence)
  - collision domain: entire network
  - loops: frames can loop forever

# Large Ethernet networks: hubs

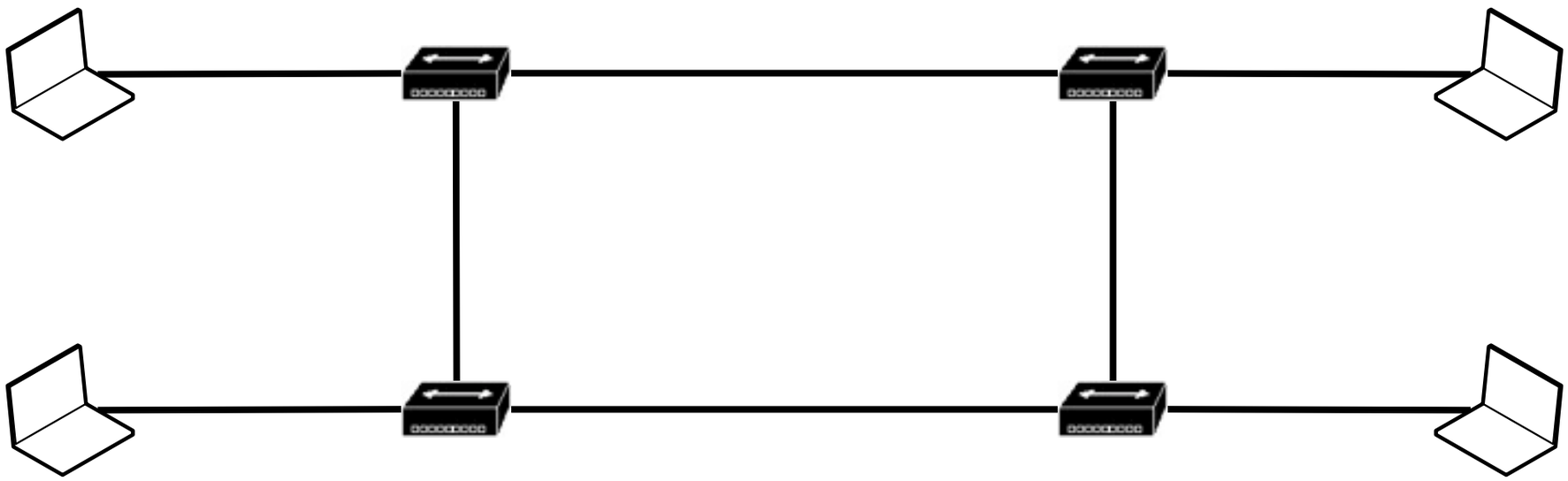
- Hubs work at the physical layer (no intelligence)
  - collision domain: entire network
  - loops: frames can loop forever

# Large Ethernet networks: hubs

- Hubs work at the physical layer (no intelligence)
  - collision domain: entire network
  - loops: frames can loop forever

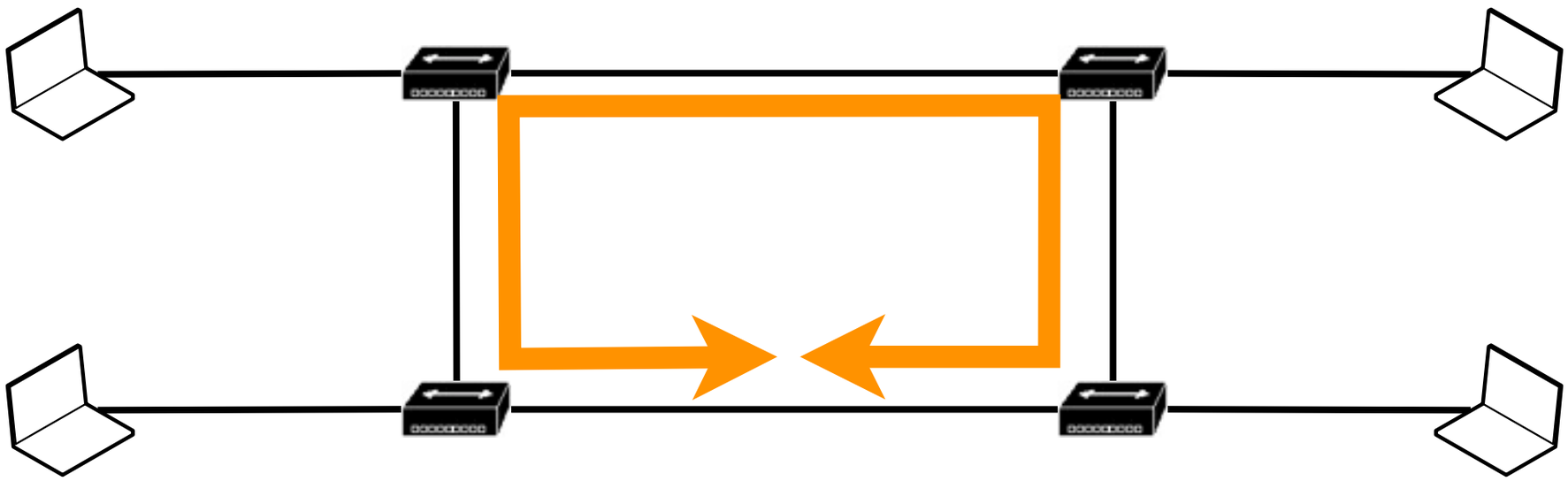
# Large Ethernet networks: hubs

- Hubs work at the physical layer (no intelligence)
  - collision domain: entire network
  - loops: frames can loop forever



# Large Ethernet networks: hubs

- Hubs work at the physical layer (no intelligence)
  - collision domain: entire network
  - loops: frames can loop forever



# Large Ethernet networks: switches

- Switches operate at the datalink layer
  - frames can be forwarded or even filtered according to their addresses
  - switches maintain a <*address, port*> table to determine to which port frames must be forwarded
    - the source address of frames traversing the switch are used to dynamically construct the <*address, port*> table
      - what if the address is not yet in the table?
      - how to forward broadcast or multicast frames?

# Frame processing

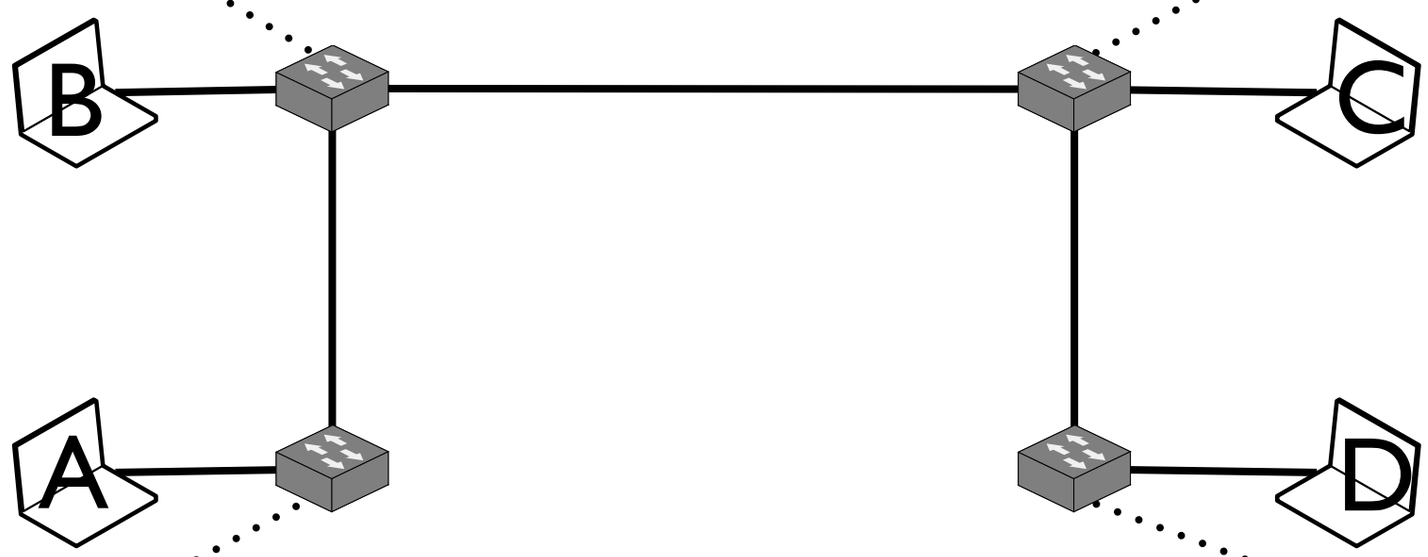
```
Upon reception of frame f on port p
table[f.source] := p    # construction of the <address, port> table on the fly
if f.destination is multicast or broadcast
        # multicast and broadcast frames are forwarded on all ports
        foreach port e != p
                forward_frame(f, e)
else
        # unicast frames are forwarded to only one port ...
        if f.destination is in table
                forward_frame(f, table[f.destination])
        # ... unless the port has not been learned yet
        else
                # then the frame is forwarded on all ports
                foreach port e != p
                        forward_frame(f, e)
```

# Large Ethernet networks: switches

- Switches work at the datalink layer (intelligence)
  - collision domain: limited
  - loops: risk of loop

| Address | Port |
|---|---|
| A | South |
| B | West |
| C | East |
| D | East |

| Address | Port |
|---|---|
| A | West |
| B | North |
| C | North |
| D | North |




| Address | Port |
|---|---|
| A | West |
| B | West |
| C | East |
| D | South |

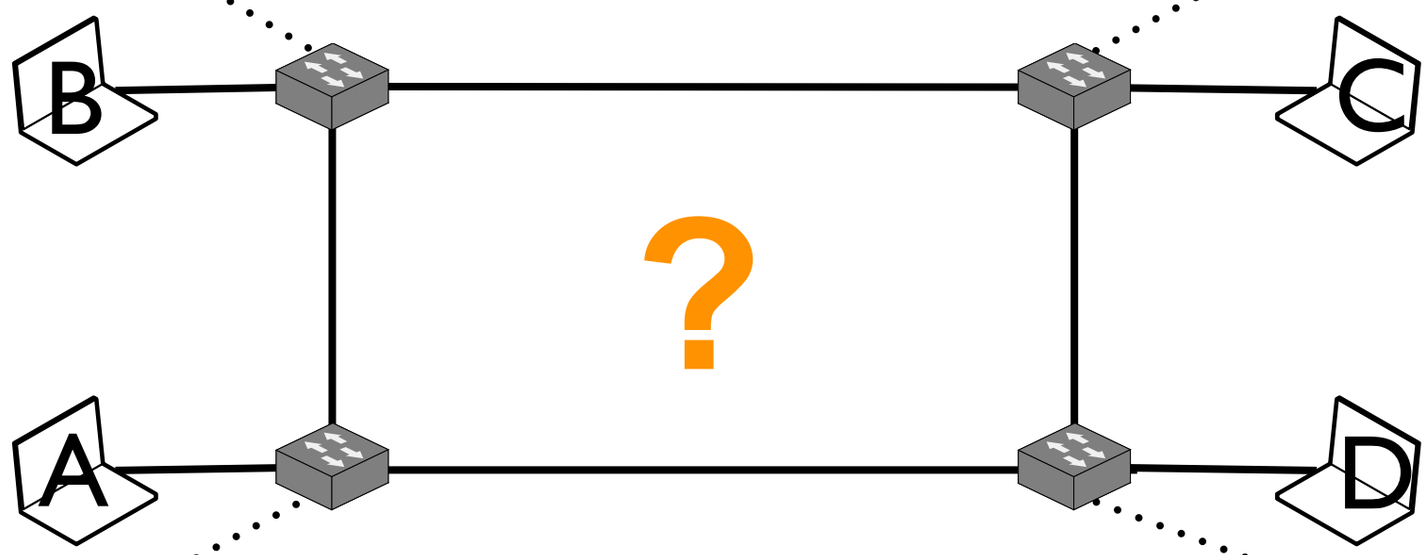
| Address | Port |
|---|---|
| A | North |
| B | North |
| C | North |
| D | East |

# Large Ethernet networks: switches

- Switches work at the datalink layer (intelligence)
  - collision domain: limited
  - loops: risk of loop

| Address | Port |
|---|---|
| A | South |
| B | West |
| C | East |
| D | East |

| Address | Port |
|---|---|
| A | West |
| B | North |
| C | North |
| D | North |




| Address | Port |
|---|---|
| A | West |
| B | West |
| C | East |
| D | South |

| Address | Port |
|---|---|
| A | North |
| B | North |
| C | North |
| D | East |

# Resilient switched Ethernet networks

- Two ways to construct a loop-free switched Ethernet topology
  - static: interconnect switches in a way they form a tree topology
    - no automatic recovery from link or switch failure
  - distributed: define a distributed algorithm that allows switches to automatically discover links causing loops and disable them

# The spanning tree protocol

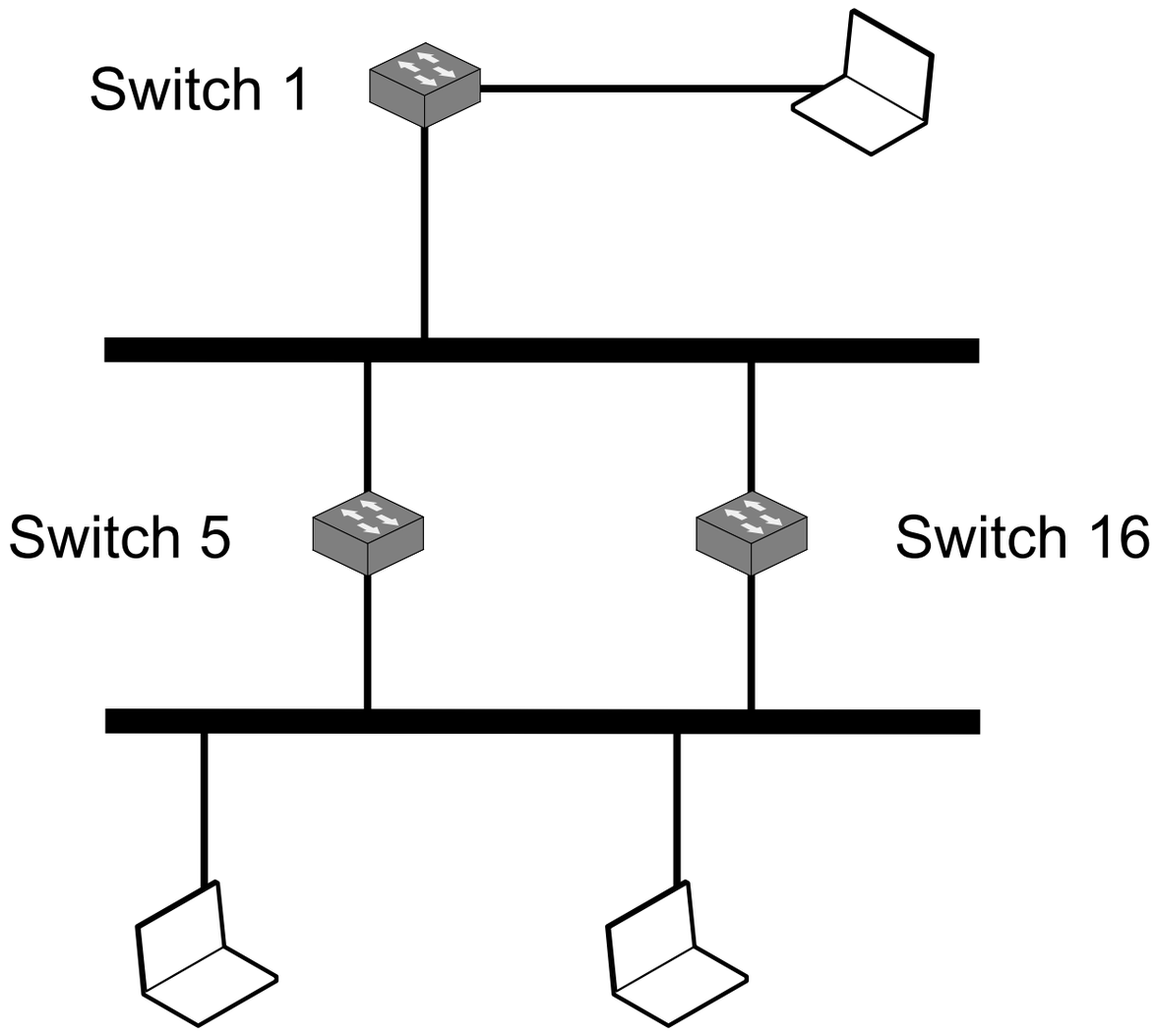
- The Spanning Tree Protocol (STP) [IEEE802.d] is a distributed protocol run by switches
  - build a spanning tree inside the network by disabling ports
  - each switch has a unique 64-bit identifier
    - 16 high order bits: a priority configured by the operator (default is 32768)
    - 48 low order bits: a unique Ethernet address from the manufacturer
  - the tree is rooted on the switch with the lowest identifier

# The spanning tree protocol (contd.)

- STP algorithm has four different features
  - root election
    - the root can change when a switch is added to the network, or removed
  - computation of the distance between switches and the root
  - when several switches are attached to the same LAN, elect one forwarder and disable the others
  - determine the ports that must be part of the spanning tree

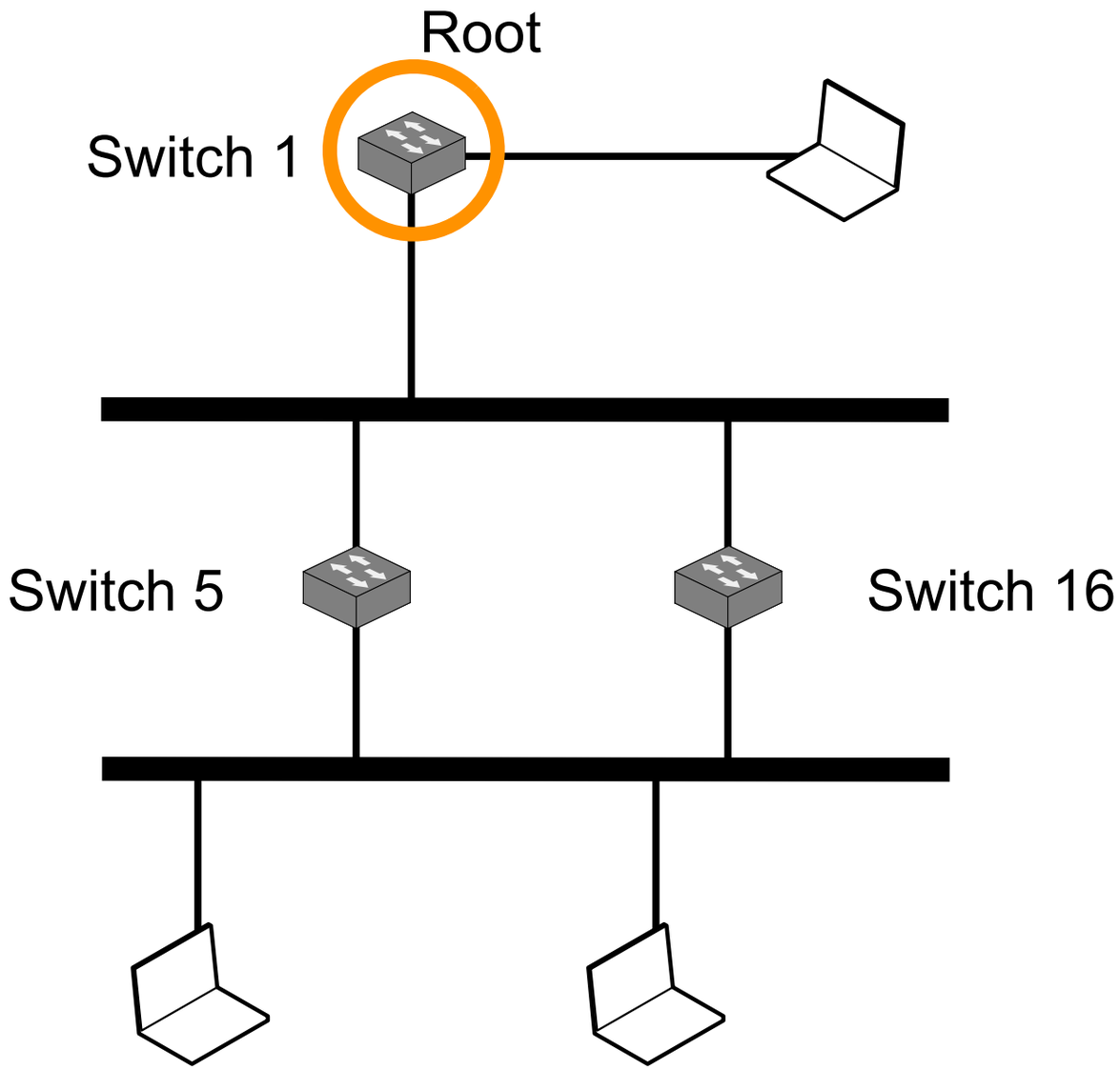
# Root and designated switches

- Root switch
  - unique, the root of the spanning tree
- Designated switch
  - the only one switch, on a given link, that can forward frames from the root
  - the root is the designated switch for all its links



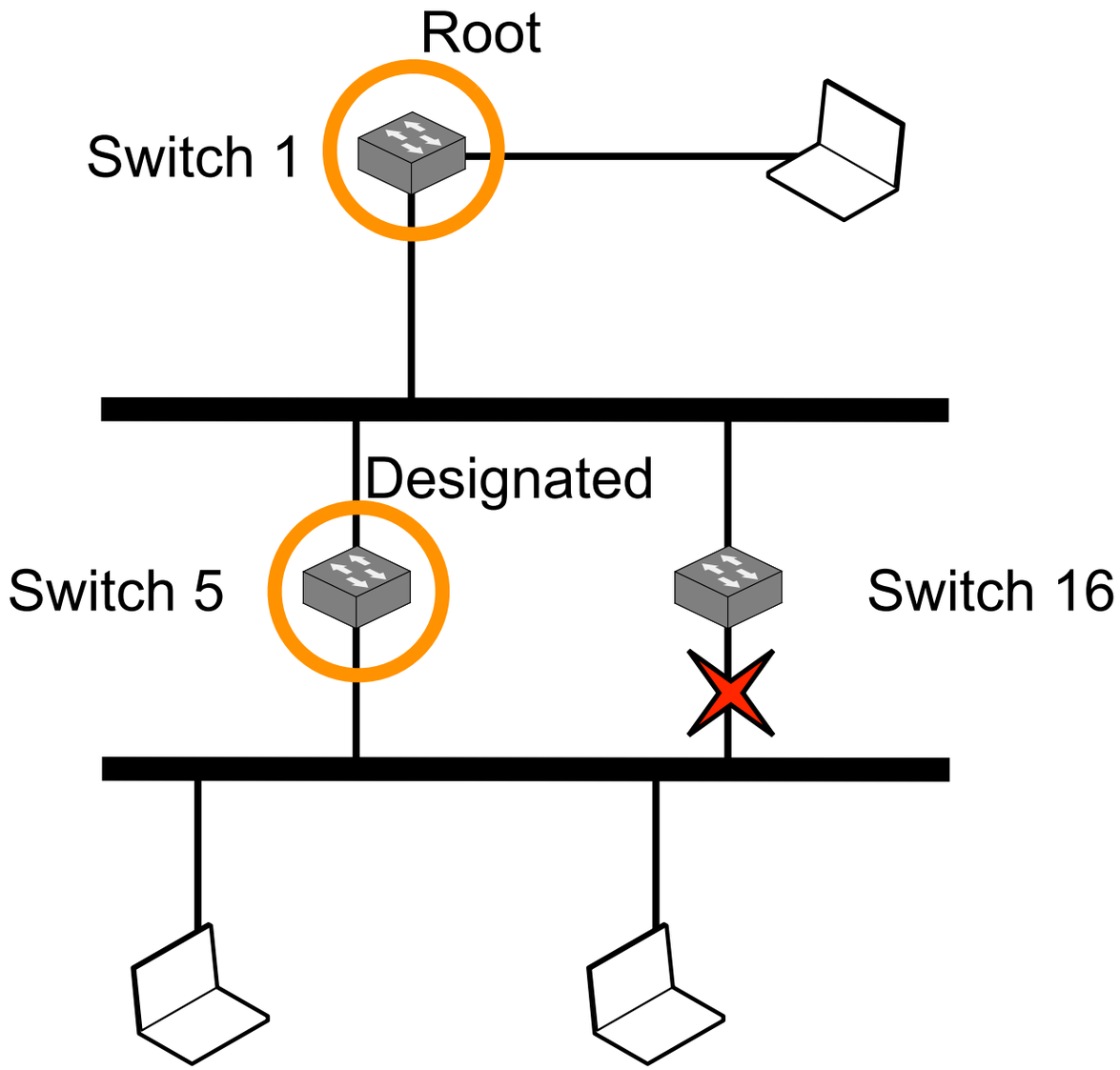
# Root and designated switches

- Root switch
  - unique, the root of the spanning tree
- Designated switch
  - the only one switch, on a given link, that can forward frames from the root
  - the root is the designated switch for all its links



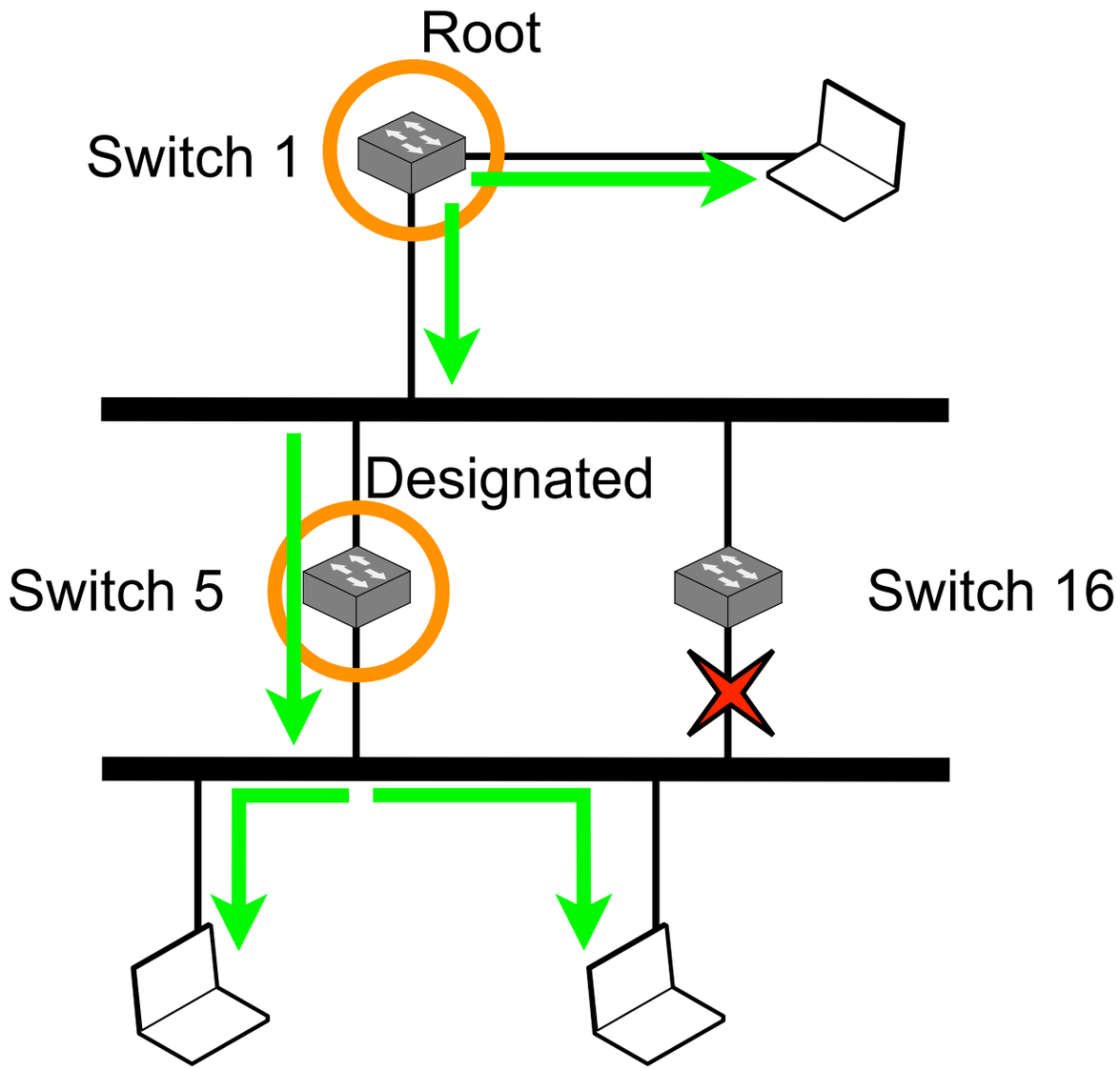
# Root and designated switches

- Root switch
  - unique, the root of the spanning tree
- Designated switch
  - the only one switch, on a given link, that can forward frames from the root
  - the root is the designated switch for all its links



# Root and designated switches

- Root switch
  - unique, the root of the spanning tree
- Designated switch
  - the only one switch, on a given link, that can forward frames from the root
  - the root is the designated switch for all its links



# Spanning tree construction

- The IEEE 802.1d protocol is used to dynamically construct the spanning tree
  - every switch periodically transmits a Bridge Protocol Data Unit (BPDU) on all its attached LANs
    - BPDUs are multicast frames
      - listened by switches implementing IEEE 802.1d
      - never forwarded by a switch
  - every BPDU contains
    - the identifier of the current root switch (e.g., root ID),
    - the cost of the shortest path between the origin of the BPDU and the root switch (e.g., cost),
    - the identifier of the switch that issued the BPDU (e.g., transmitting ID)
  - when a switch starts (or didn't received BPDUs), it considers itself as the root with a cost of 0 to reach it
  - it exists a total order relationship between BPDUs

# Switch port states

- A port can have 3 states for control plane frames (i.e., IEEE 802.1d BPDUs)
  - Root port (RP)
    - port on which the best BPDU was received
    - only one root port per switch
    - not used to transmit BPDUs
  - Designated port (DP)
    - port(s) on which BPDUs are sent upon reception of a BPDU received via the root port
    - a port is in designated mode if the switch's BPDU is better than the best BPDU received on that port
  - Blocked port (BP)
    - only receives BPDUs
- A port can have 2 activities for data plane frame
  - Active: the port can receive or send frames on the port
  - Inactive: no frame can be received or sent on the port

# Port state and activity: summary

| | **Receive BPDUs** | **Transmit BPDUs** |
|---|---|---|
| **Root** | Yes | No |
| **Designated** | Yes | Yes |
| **Blocked** | Yes | No |

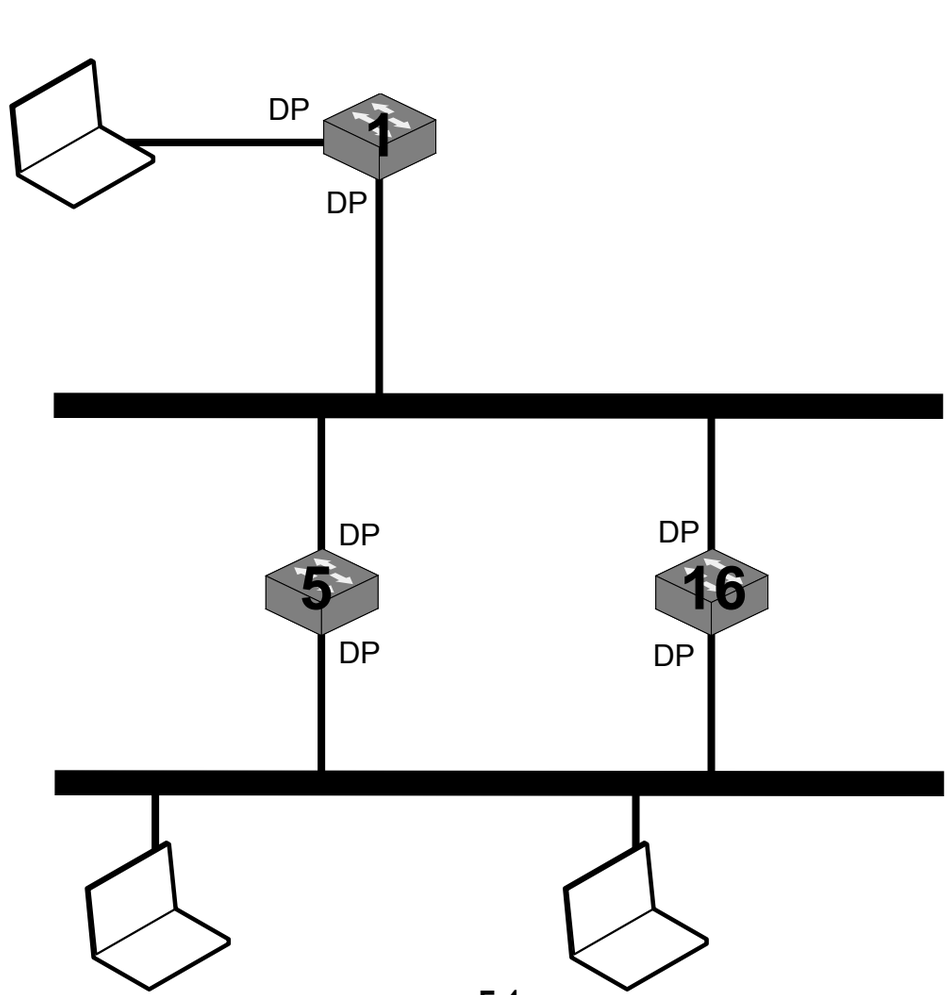
| | **Learn addresses** | **Forward Data Frames** |
|---|---|---|
| **Active** | Yes | Yes |
| **Inactive** | No | No |

# BPDUs total order relationship

- It exists a total order relationship between BPDUs
  - BPDU1 [root ID = R1, cost = C1, transmitting ID = T1]
    BPDU2 [root ID = R2, cost = C2, transmitting ID = T2]
  - BPDU1 is better than BPDU2
    - if R1 < R2
    - or if R1 = R2 and C1 < C2
    - or if R1 = R2 and C1 = C2 and T1 < T2
      - in case of equality, the port with the lowest port identifier is selected
- A switch stores the best BPDU it receives on each of its ports
- The root, as seen by a given switch, is the one indicated in the best BPDU among all the BPDUs stored in the switch
- A switch stops sending BPDUs on a port when it has received a better BPDU on that port

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1




| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | | | |

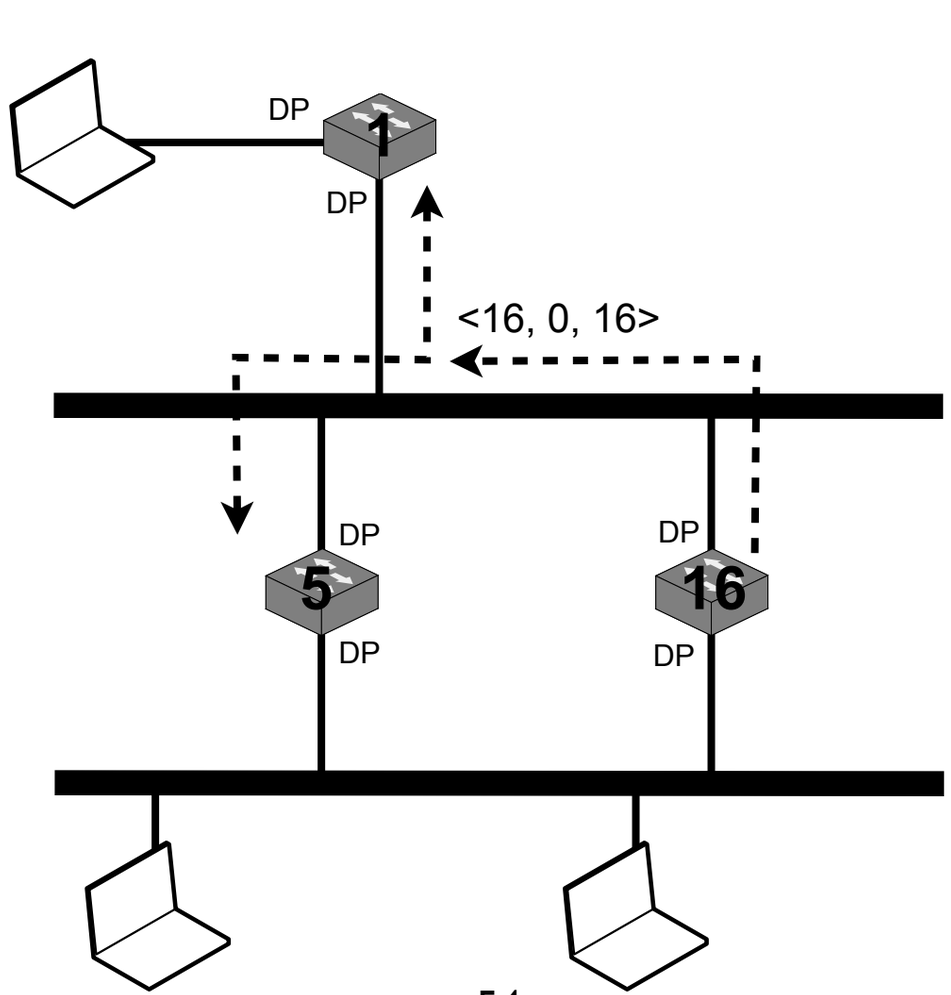
| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | | | |
| 2 | | | |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **16** | **0** | **16** |
| 1 | | | |
| 2 | | | |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | | | |

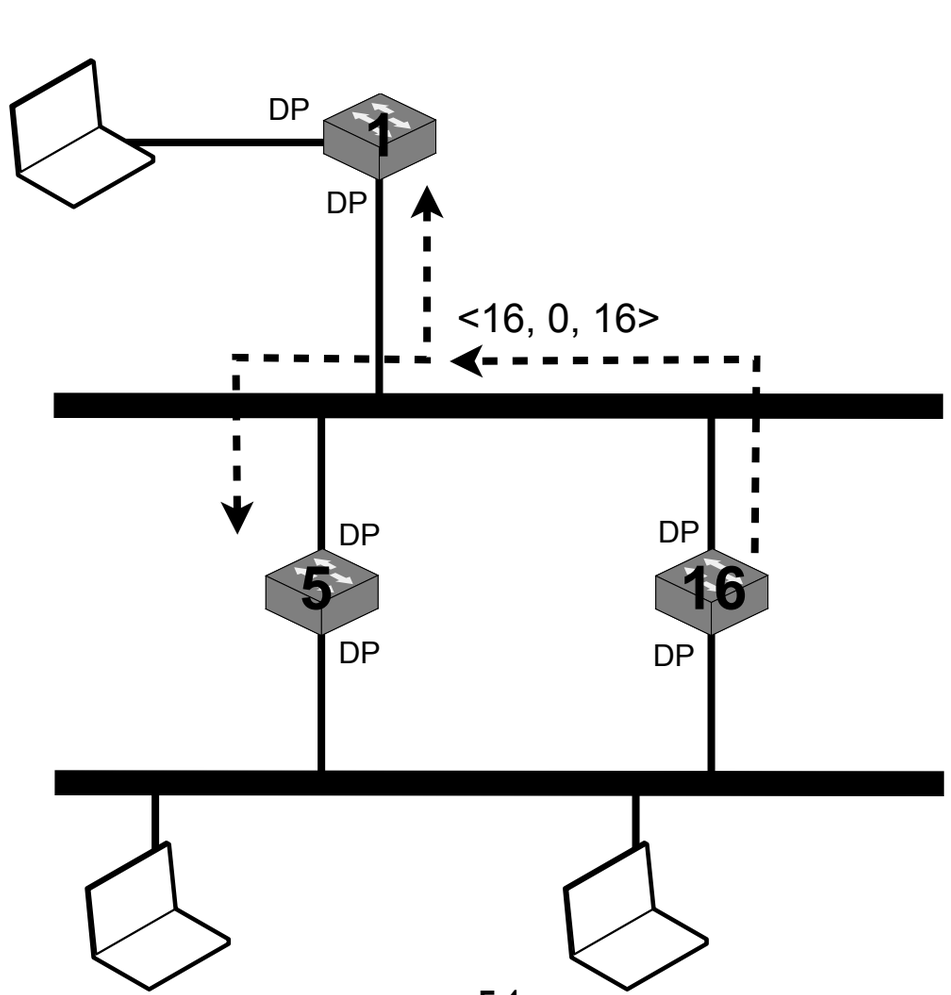



| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | | | |
| 2 | | | |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **16** | **0** | **16** |
| 1 | | | |
| 2 | | | |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1



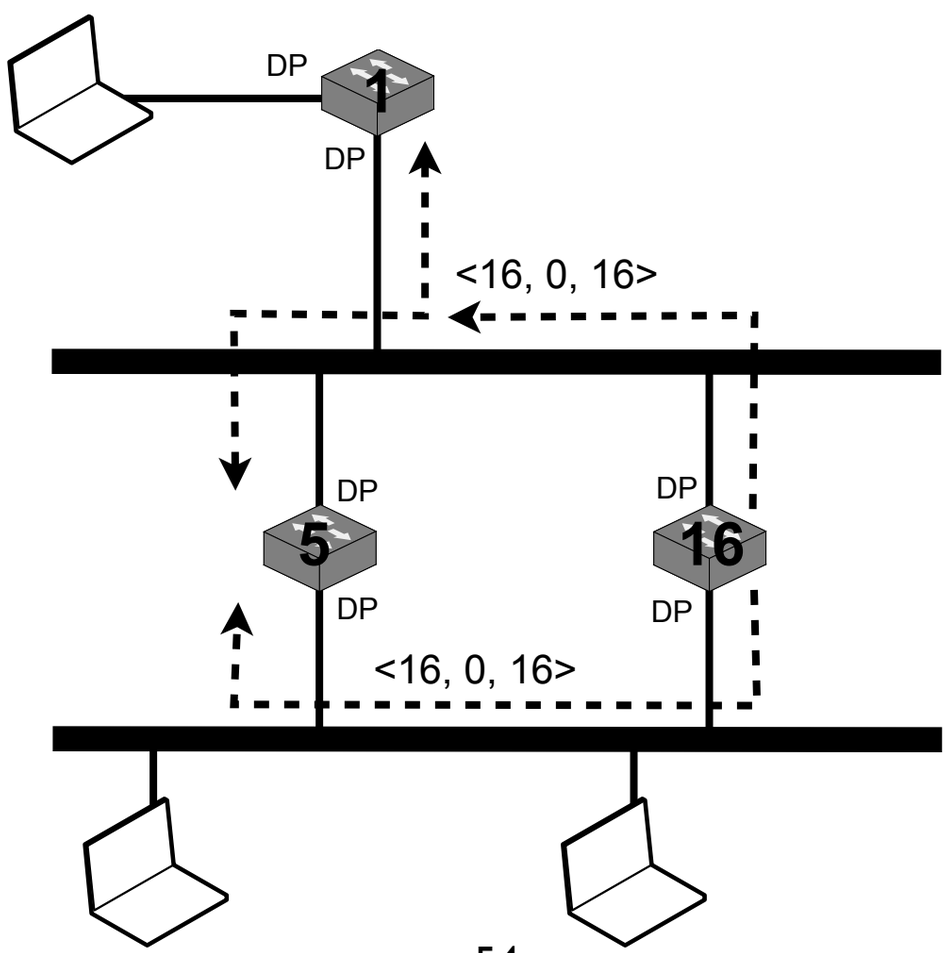

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | | | |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **16** | **0** | **16** |
| 1 | | | |
| 2 | | | |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1




| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 16 | 1 | 16 |

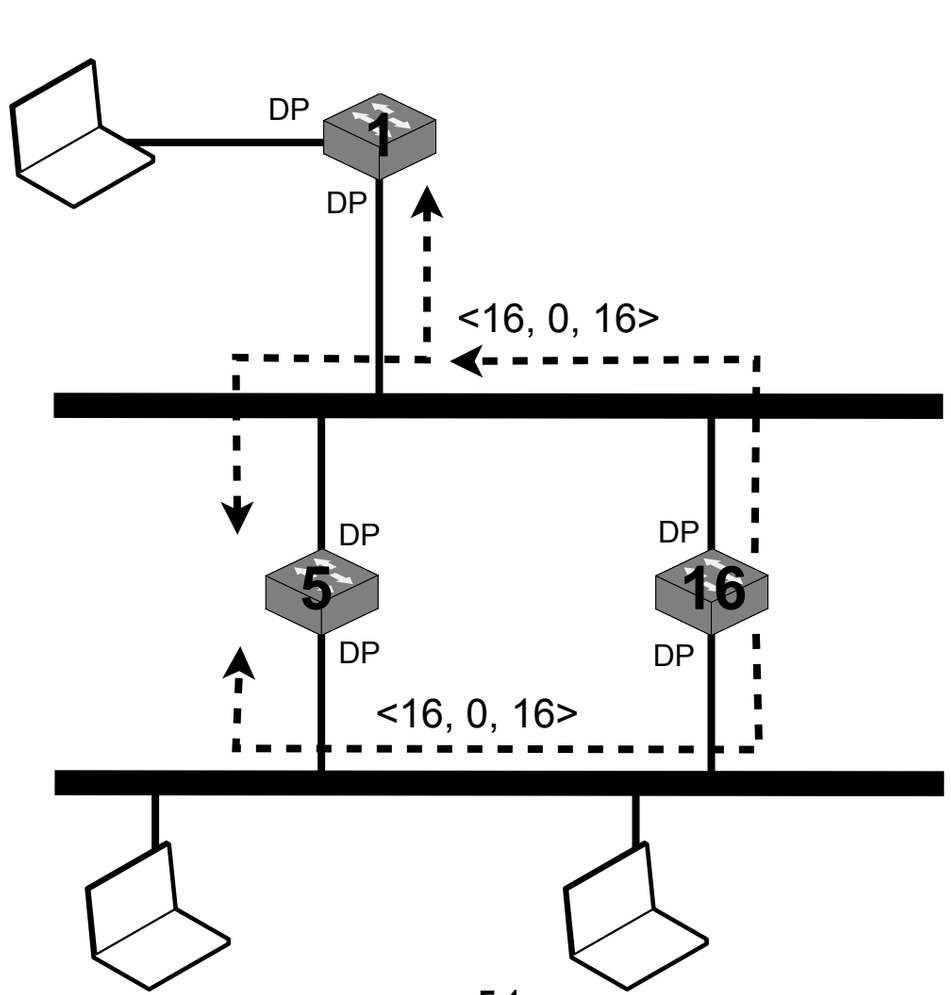
| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | | | |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **16** | **0** | **16** |
| 1 | | | |
| 2 | | | |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 16 | 1 | 16 |

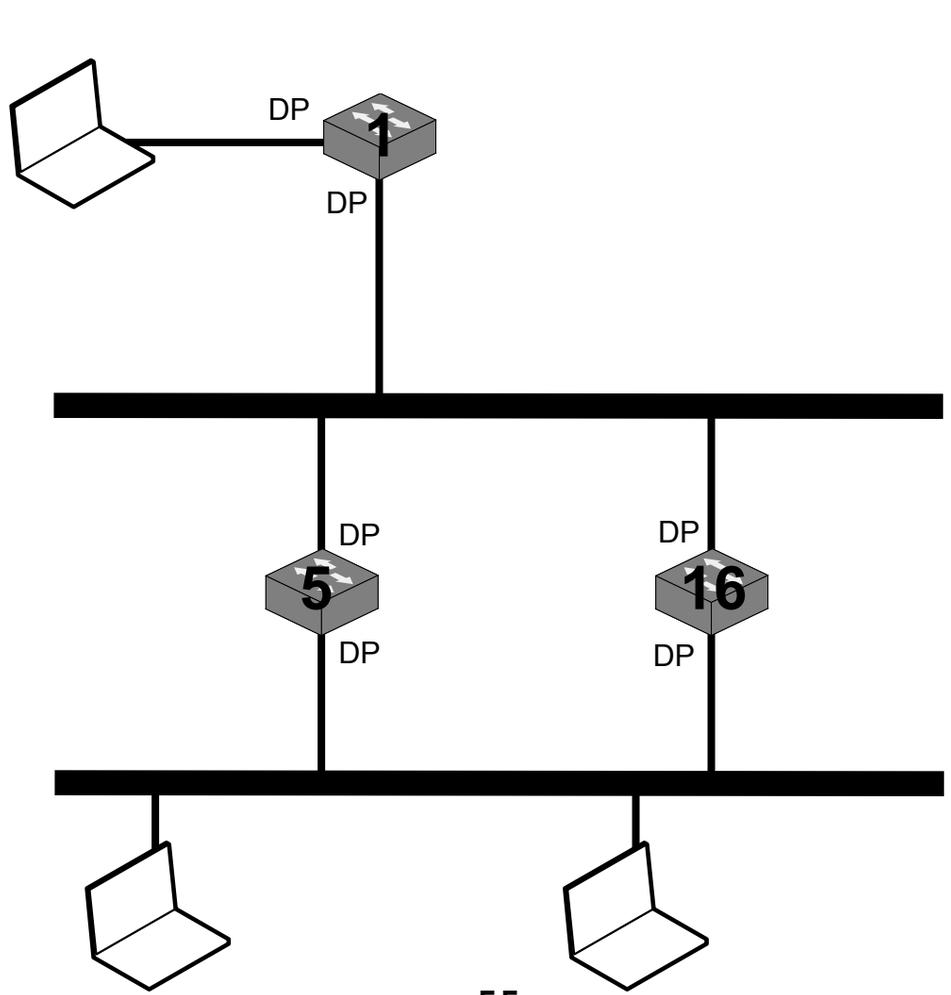



| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **16** | **0** | **16** |
| 1 | | | |
| 2 | | | |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1



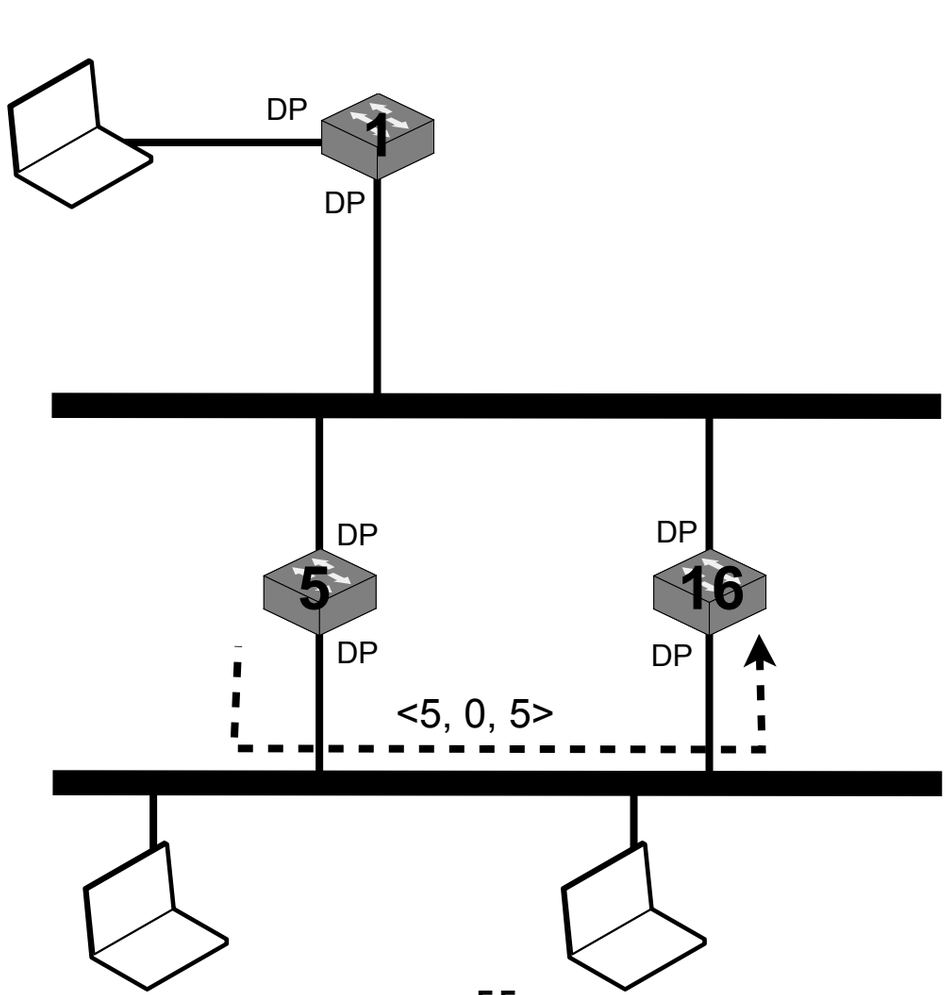

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **16** | **0** | **16** |
| 1 | | | |
| 2 | | | |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1




| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 16 | 1 | 16 |

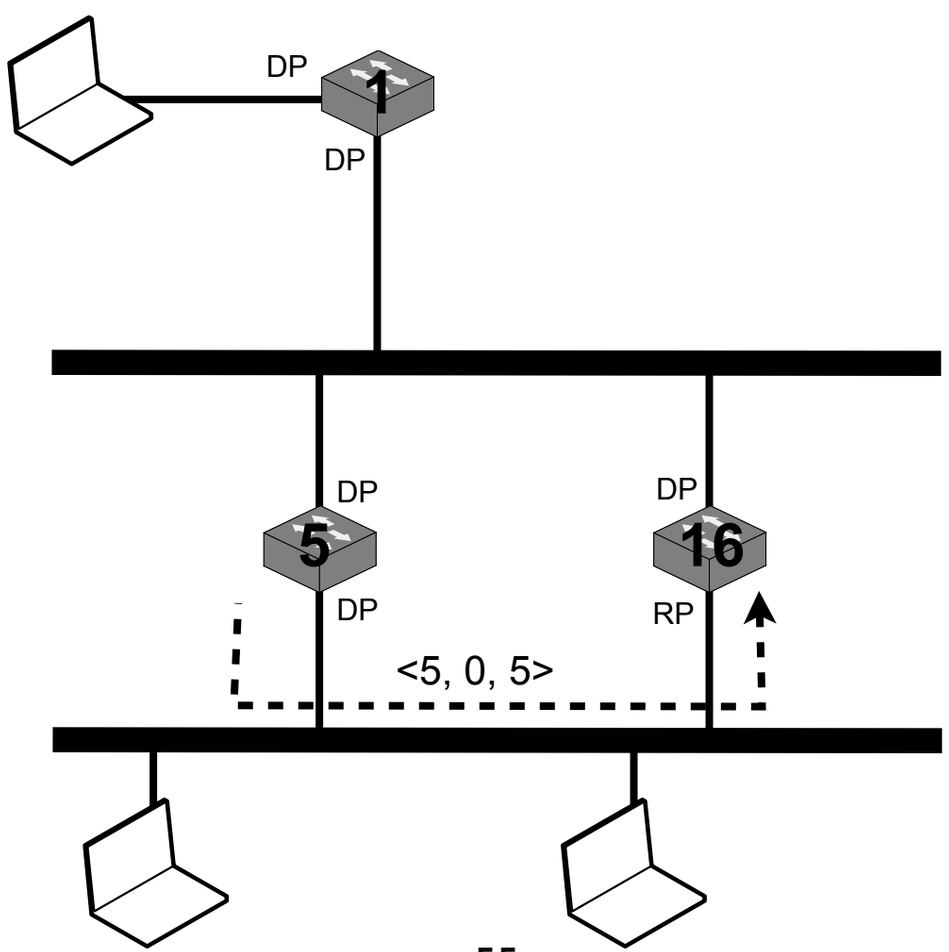
| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **16** | **0** | **16** |
| 1 | | | |
| 2 | | | |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 16 | 1 | 16 |



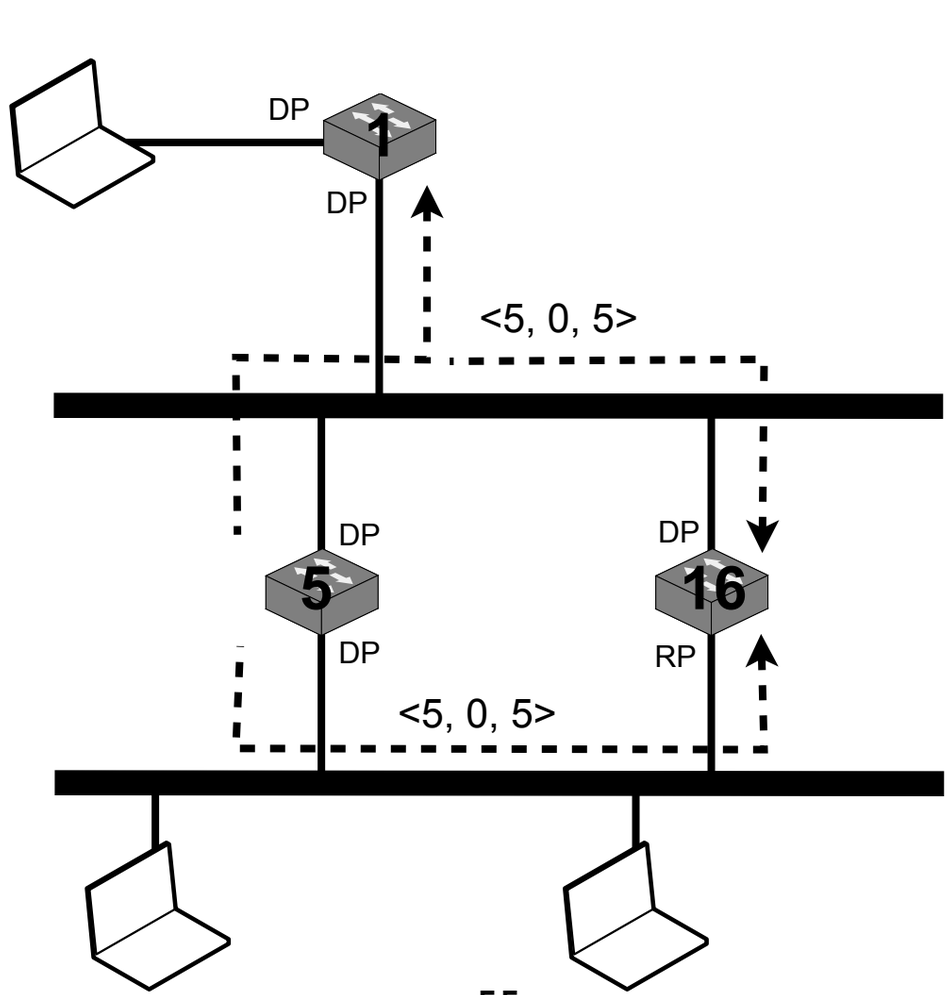

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 5 | 1 | 16 |
| 1 | | | |
| 2 | **5** | **1** | **5** |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 16 | 1 | 16 |



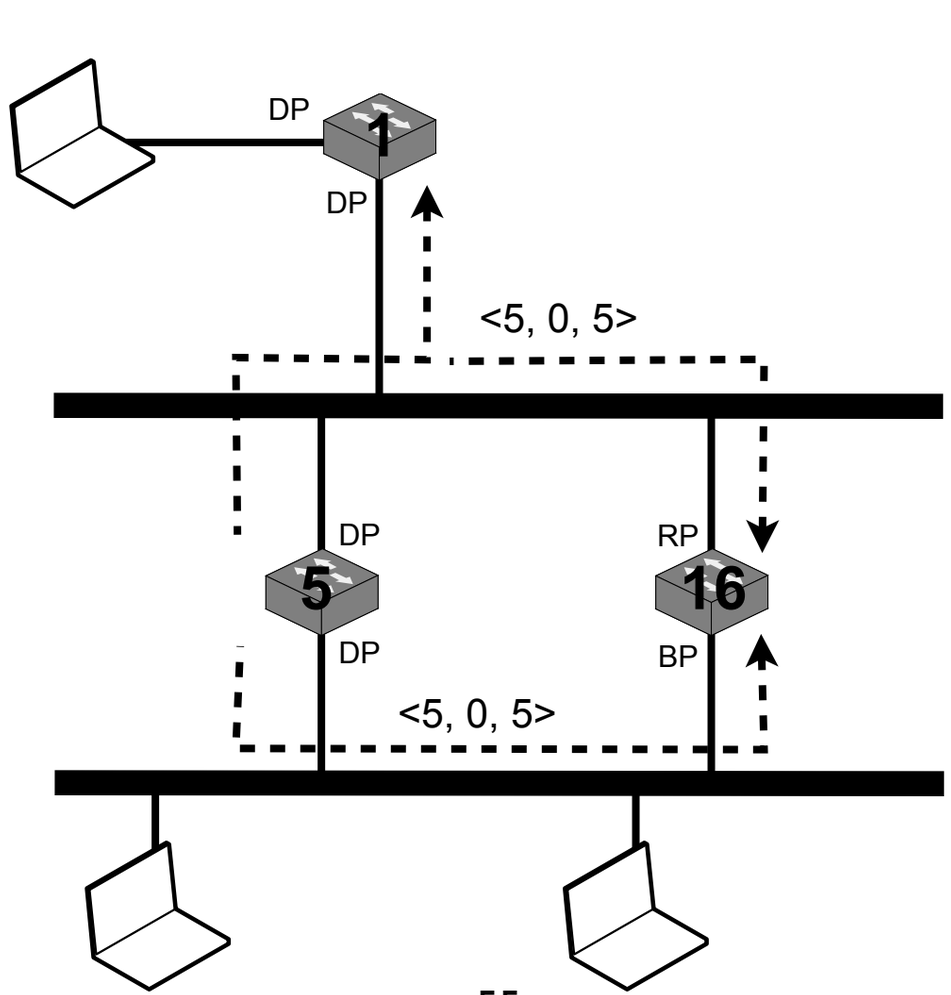

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 5 | 1 | 16 |
| 1 | | | |
| 2 | **5** | **1** | **5** |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 16 | 1 | 16 |



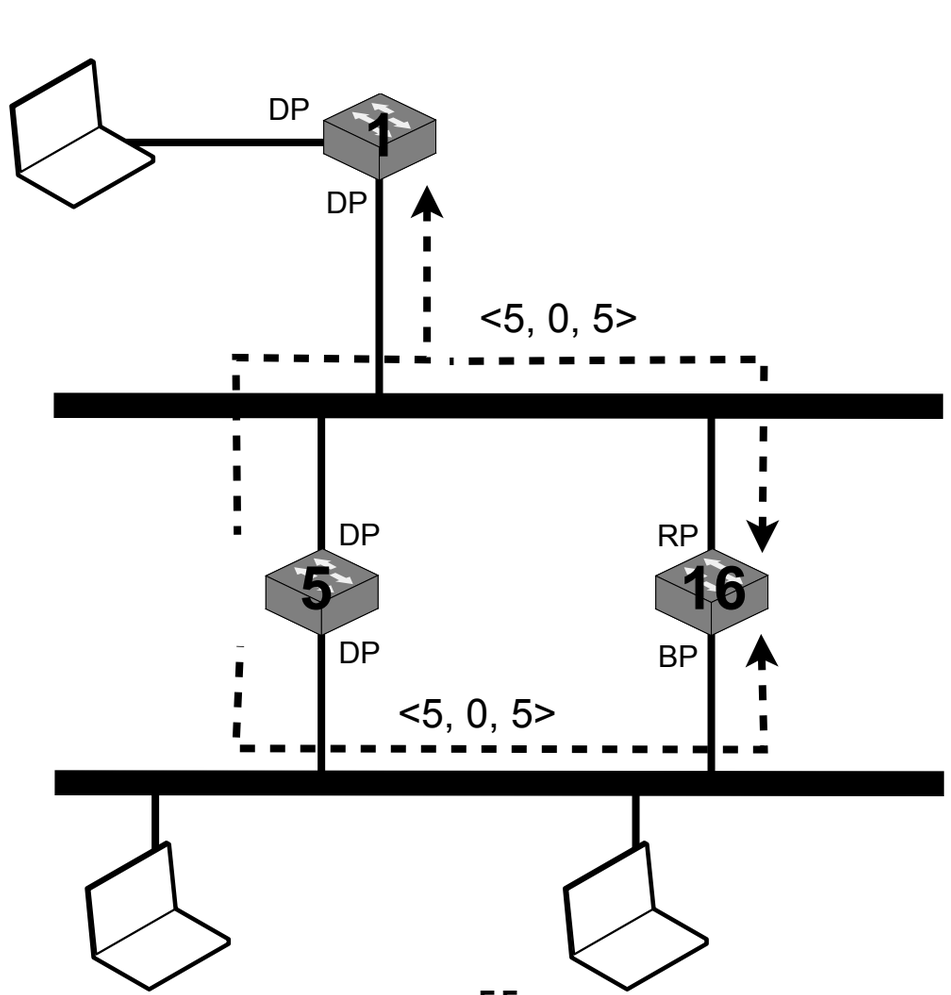

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 5 | 1 | 16 |
| 1 | **5** | **1** | **5** |
| 2 | 5 | 1 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |

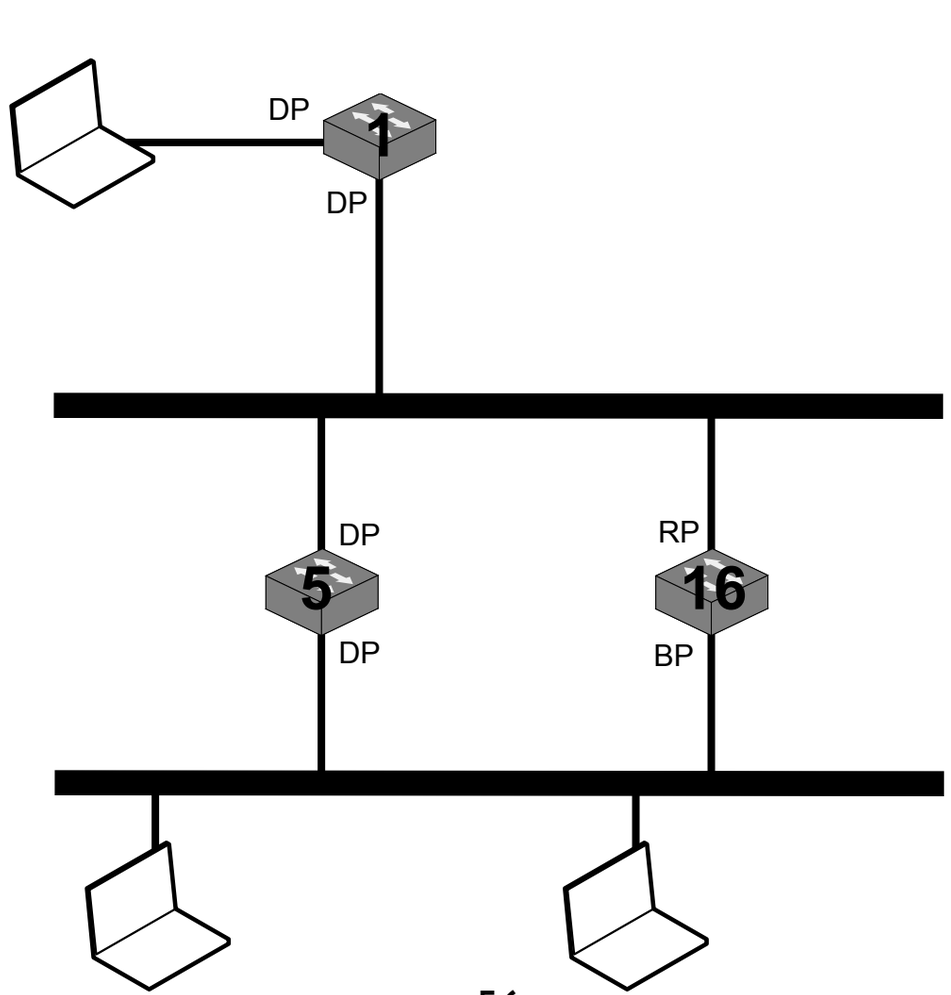



| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 5 | 1 | 16 |
| 1 | **5** | **1** | **5** |
| 2 | 5 | 1 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1



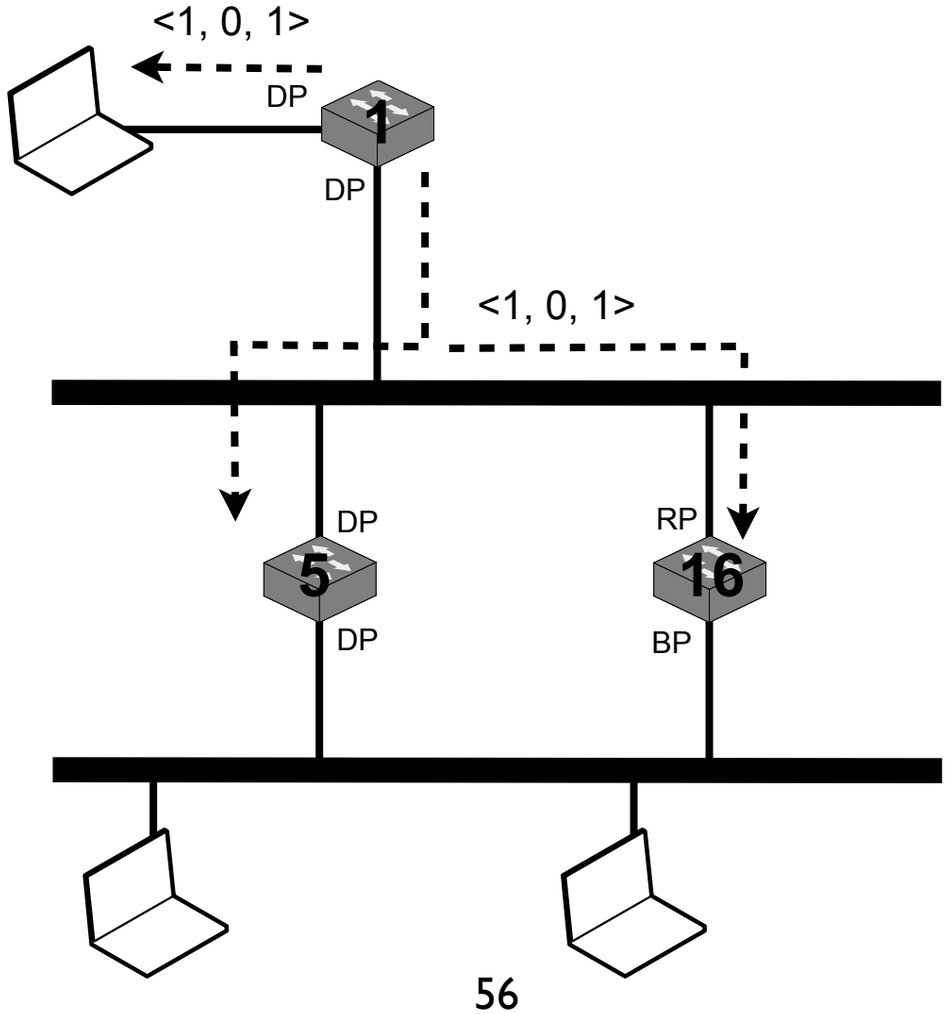

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 5 | 1 | 16 |
| 1 | **5** | **1** | **5** |
| 2 | 5 | 1 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1



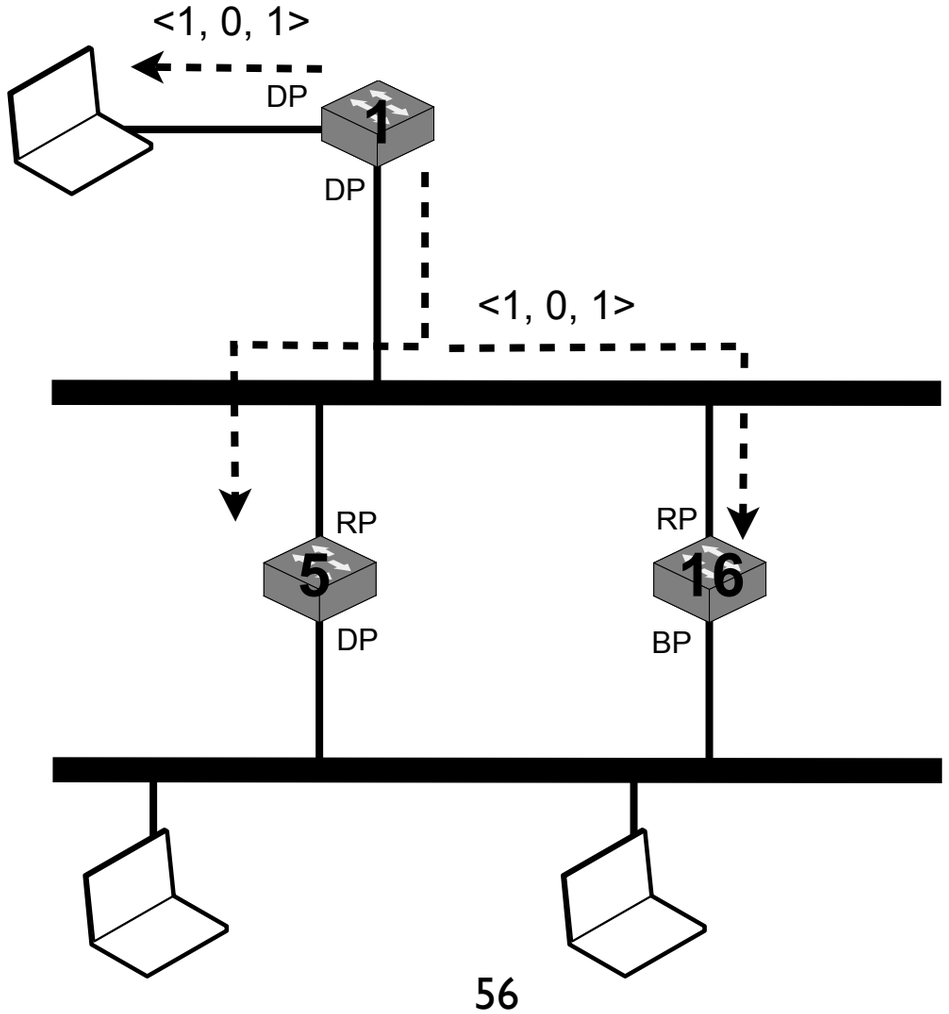

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **5** | **0** | **5** |
| 1 | 16 | 1 | 16 |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 5 | 1 | 16 |
| 1 | **5** | **1** | **5** |
| 2 | 5 | 1 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1



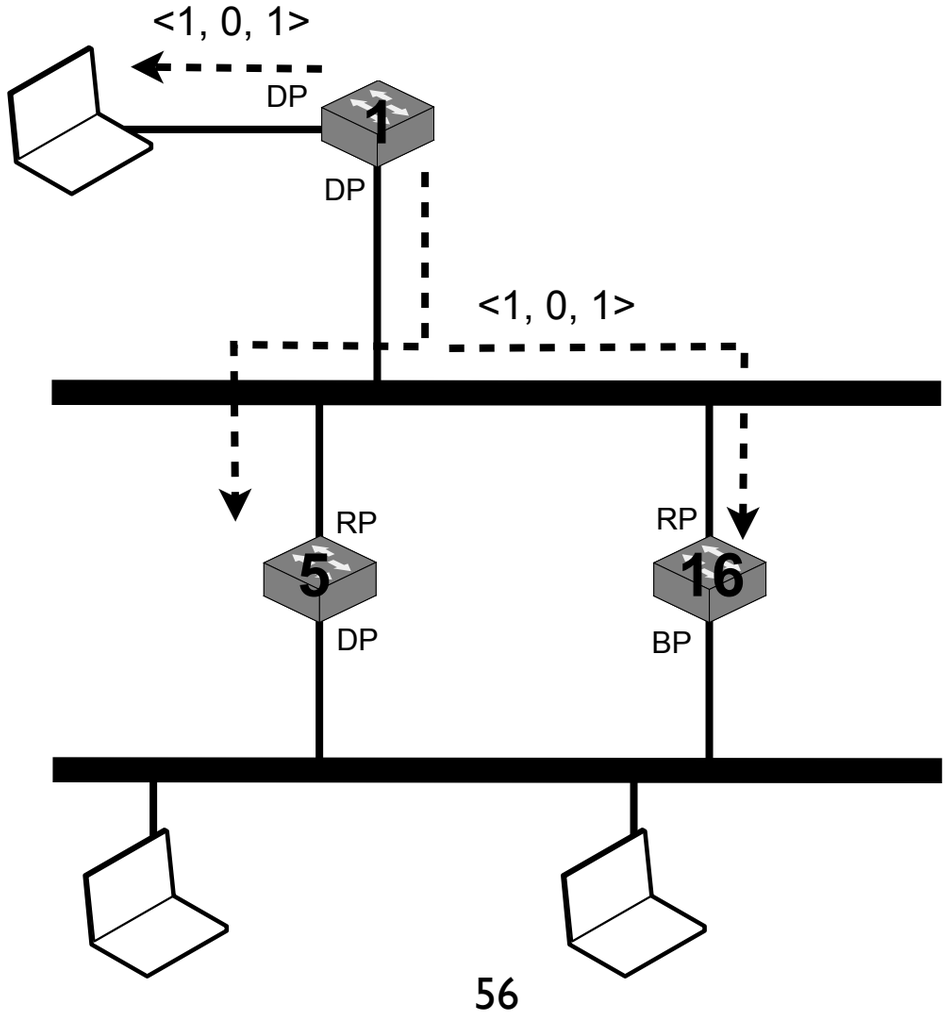

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 5 |
| 1 | **1** | **1** | **1** |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 5 | 1 | 16 |
| 1 | **5** | **1** | **5** |
| 2 | 5 | 1 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1



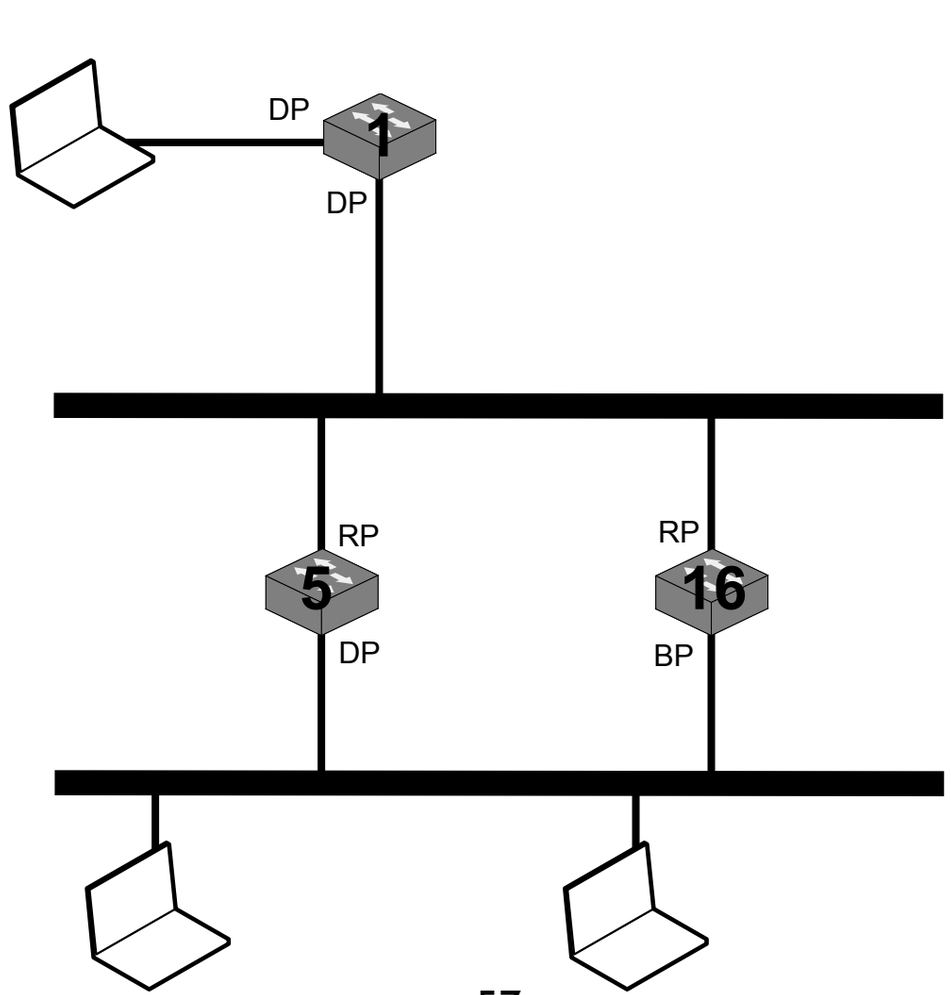

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 5 |
| 1 | **1** | **1** | **1** |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 16 |
| 1 | **1** | **1** | **1** |
| 2 | 5 | 1 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1




| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |

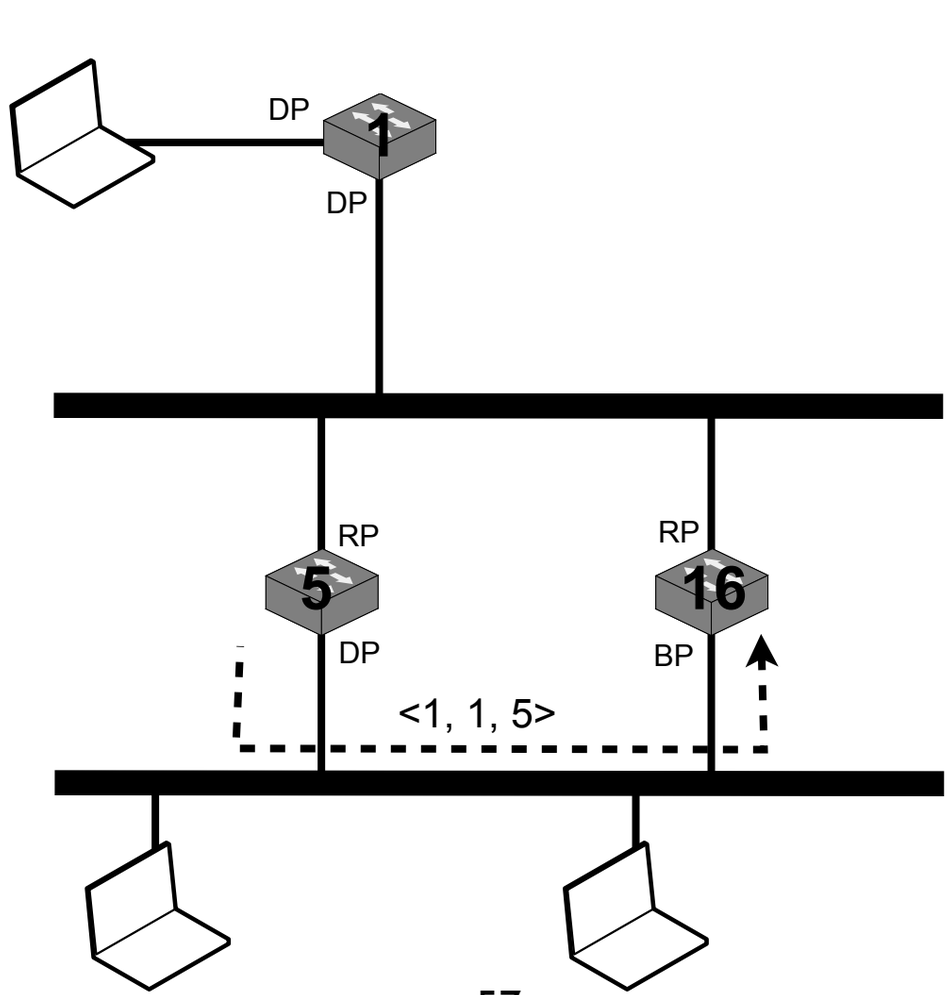
| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 5 |
| 1 | **1** | **1** | **1** |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 16 |
| 1 | **1** | **1** | **1** |
| 2 | 5 | 1 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |



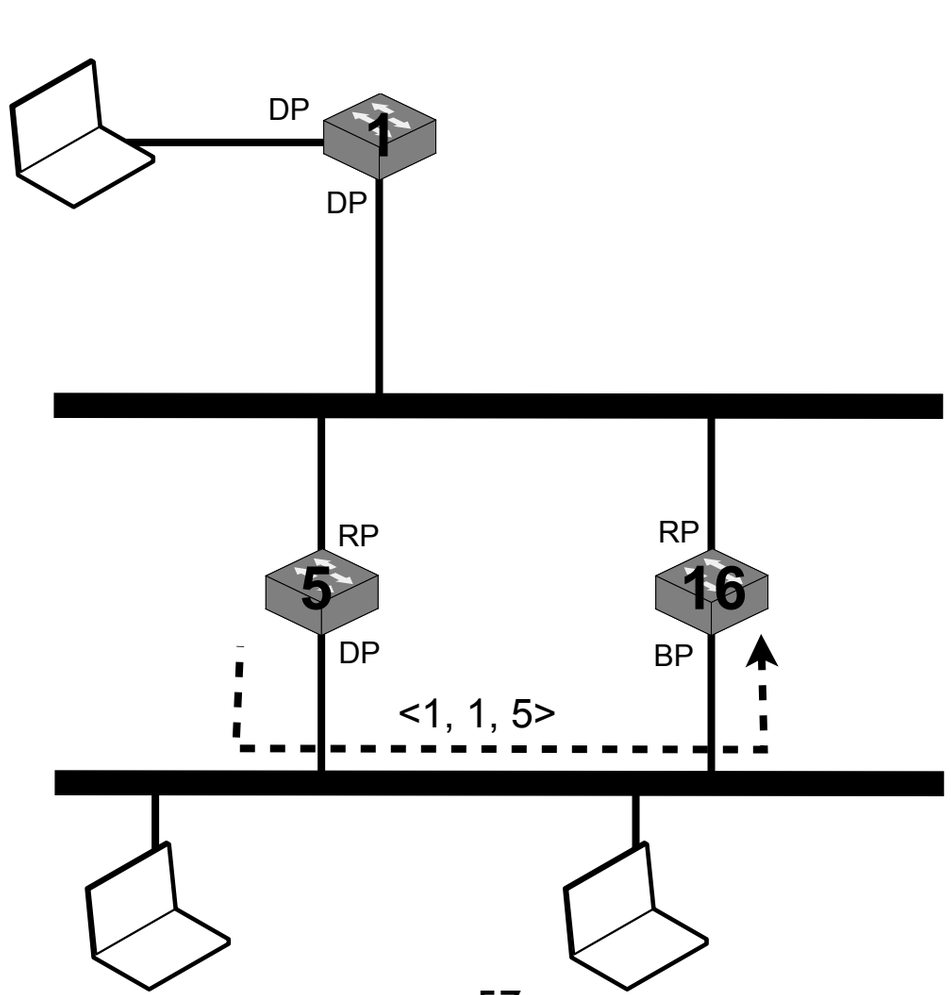

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 5 |
| 1 | **1** | **1** | **1** |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 16 |
| 1 | **1** | **1** | **1** |
| 2 | 5 | 1 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |

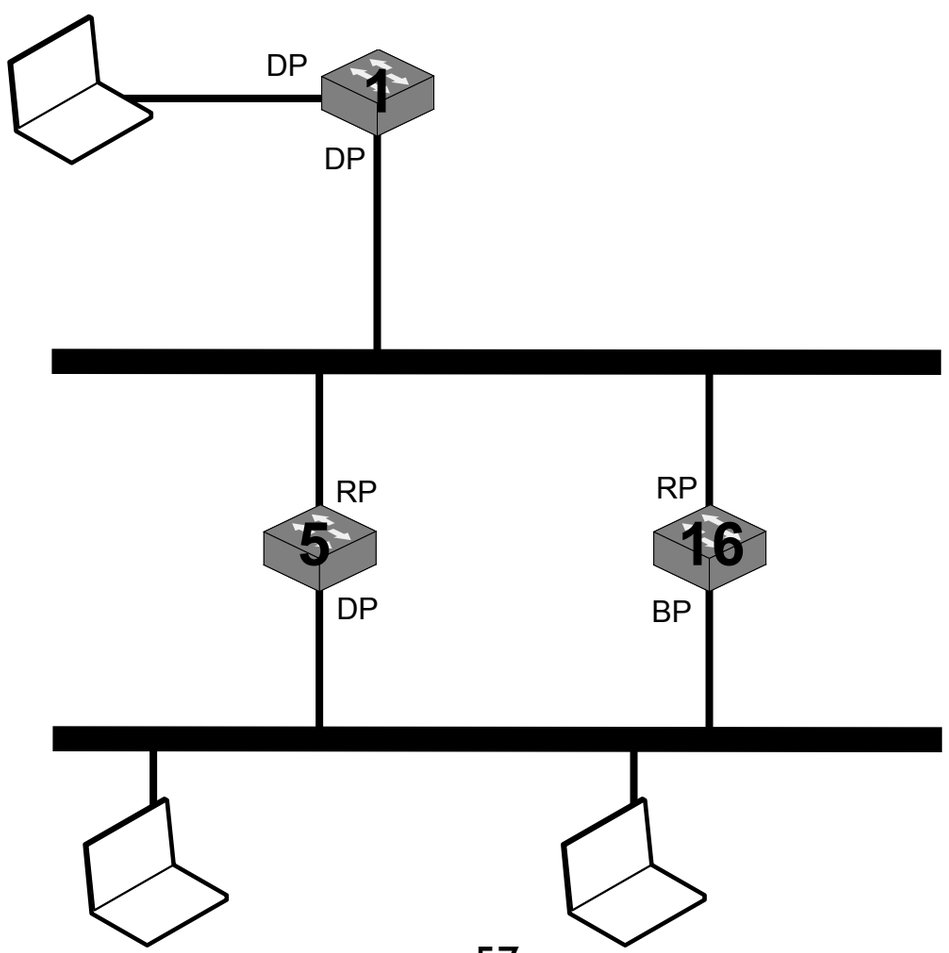



| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 5 |
| 1 | **1** | **1** | **1** |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 16 |
| 1 | **1** | **1** | **1** |
| 2 | 1 | 2 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1




| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |

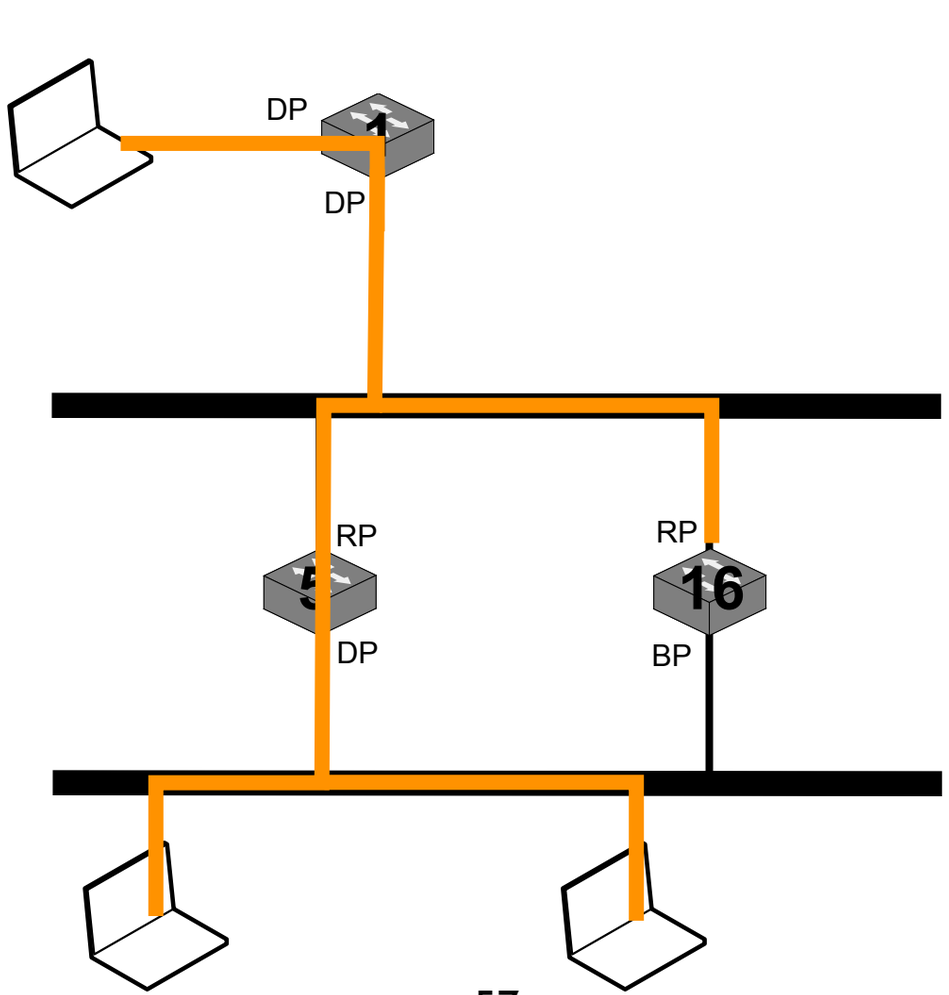
| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 5 |
| 1 | **1** | **1** | **1** |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 16 |
| 1 | **1** | **1** | **1** |
| 2 | 1 | 2 | 5 |

# Spanning tree protocol example

- Example
  - link cost = 1
  - port numbered from counter-clockwise, starting at 1

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | **1** | **0** | **1** |
| 1 | | | |
| 2 | 5 | 1 | 5 |




| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 5 |
| 1 | **1** | **1** | **1** |
| 2 | 16 | 1 | 16 |

| Port | Root | Cost | Trans. |
|---|---|---|---|
| local | 1 | 1 | 16 |
| 1 | **1** | **1** | **1** |
| 2 | 1 | 2 | 5 |

# Choosing link cost

# How to determine link cost?

- The spanning tree protocol and shortest path routing use link costs
  - one cost is associated per link and direction
  - the lower is the cost, the more is the expected traffic on the link
  - costs can be assigned
    - statically: stable with time but not adapted to network load
    - dynamically: adapted to the network load but unstable and risk of oscillations

# Common costs

- Unit cost, every link has the same cost
  - not adapted to heterogeneous networks
- Cost depending on link capacity (e.g., 1000/capacity)
  - high cost for low capacity links
  - low cost for high capacity links
- Cost depending on link delay
  - often used to avoid satellite links
- Cost dynamically determined based on measurements
  - proportional to router queue length (independent of link capacity)
  - depending on link bandwidth usage

# Why aren't dynamic costs used?

- Dynamically assigned costs are not widely used in today's Internet
  - hard to control the amount of routing updates as it depends on the traffic
  - hard to determine the right frequency at which update the costs to ignore transient spikes (e.g., bursty nature of TCP traffic) but take major traffic changes into account
  - risk of transient loops during each path re-computation
  - Risk of oscillations
    - e.g., when a link is overloaded, its cost become high and it not used anymore, lowering its cost making traffic coming back...

# Operational corner

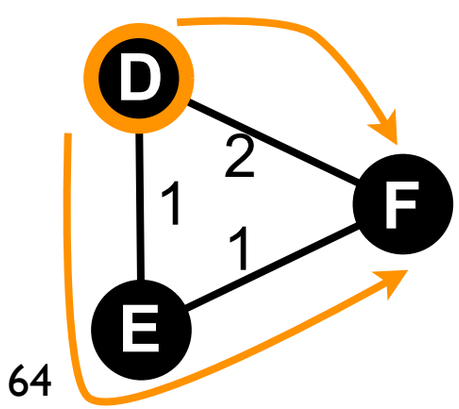
# Multipath routing

- In the examples shown earlier, there is only one best path
  - in practice it may have many paths with the same minimum cost
  - but the multiple shortest paths do not have the same performance (delay, bandwidth...)
- Equal-Cost Multi-Path routing (ECMP) is often deployed to use several path in parallel inside a network

# ECMP principle

- Several outgoing interfaces can be associated to a destination
  - for every packet, a function, local to the router, is called to determine the outgoing interface to use among the possible outgoing interfaces

| Destination | Next-hop | Interface |
|---|---|---|
| D | local | local |
| E | E | South |
| F | E,F | South, South-East |



# Outgoing interface selection

- Many different function exist to select the outgoing interface, the most common are
  - random: *f(packet)* = *random()*
  - per destination: *f(packet)* = *magic(packet.destination)*
  - per flow: *f(packet)* = *magic(packet.5_tuple)*
- If the *n* possible interfaces for the destination are numbered from *0* to *n-1*, the choice of the interface is given by
  - *outgoing_interface* = *f(packet) % n*

# What is a good function?

- A good function is a function that avoid packets of the same flow to follow different paths
  - Different paths may have different performances (e.g., bandwidth, delay, queues...), causing packets loss
    - TCP (e.g., AIMD) is very aggressive when it detects packet loss
- A good function should be fast to compute
- A good function should be stateless
- Functions used for outgoing interface selection in ECMP generally rely on hash functions
  - *f(packet) = hash(packet.5_tuple, seed)*

# Research corner

# How to build routing tables without exchanging routing information?

- fill me
- fill me
- fill me

# Homework

due date 01/25/2013

# Forwarding in Ethernet networks

- How to recover from a major failure in a large switched Ethernet network?
- How to use several links in parallel in a large switched Ethernet network?

# Link state routing

- How to deal with link failure?
- How to deal with router reboot and/or failure?
- How to deal with the loss of a LSP?
- How to avoid the same LSP to be sent twice on the same link (like between D and F in the LSP flooding example)?
- How to make LSP and LSPDB robust to
  - corruption
  - attacks
- Under which condition transient loops can be observed?

# Distance vector routing

- How to deal with link failure?
- How to deal with router failure?
- How to deal with the lost of a distance vector?
- Is that possible to have a distance vector loop?

# General questions

- Compare link state routing and distance vector routing
  - what are their respective pros?
  - what are their respective cons?
- So far we have seen unicast routing, how to construct routing tables for the multicast case?

# Dynagen

- For the next two sessions, we will need Dynagen (http://dynagen.org/) to setup network labs
  - be sure the tool can run on your machine
    - in case of problem, please contact me