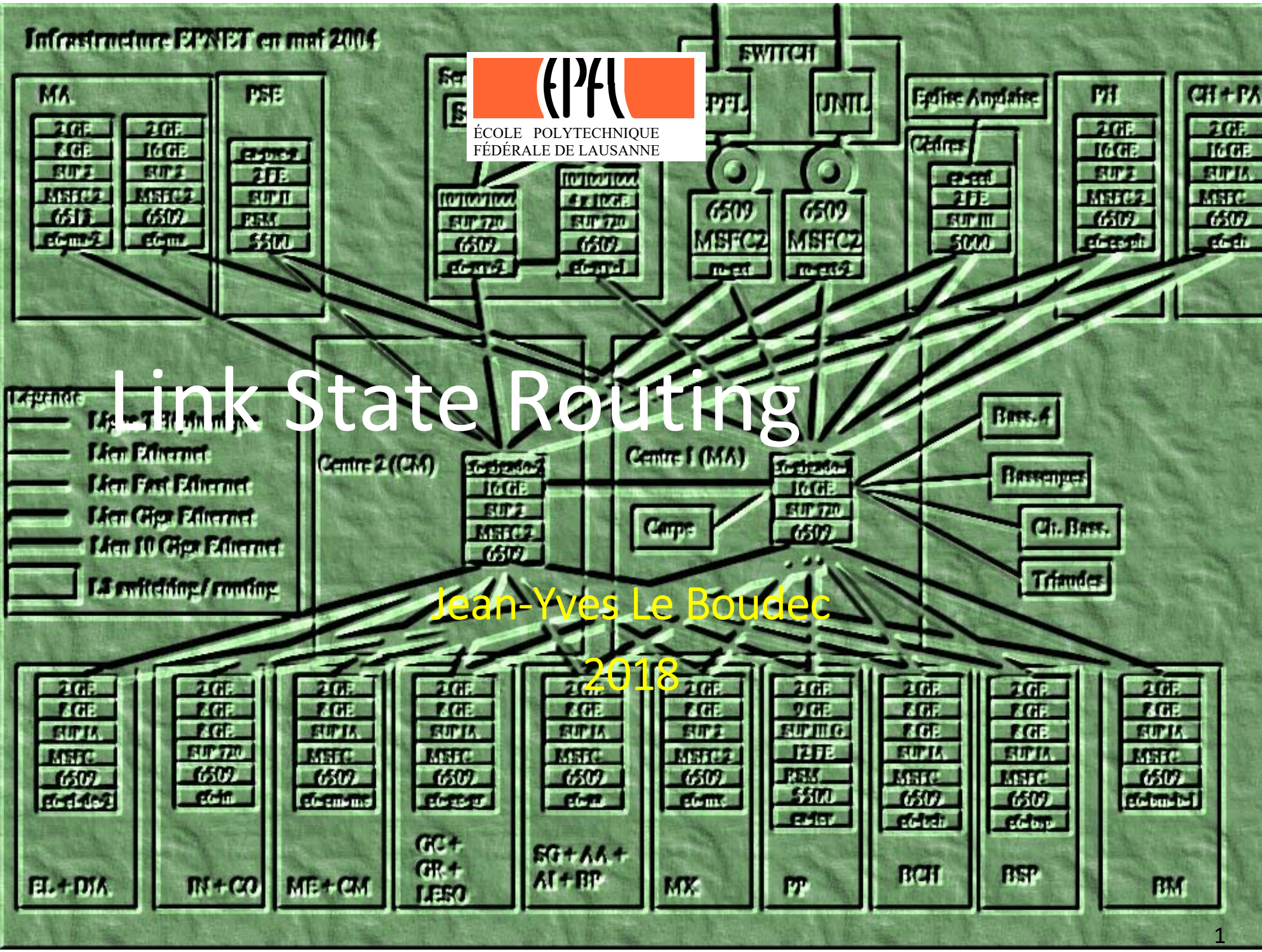



# Link State Routing

Jean-Yves Le Boudec

2018

# Contents

1. Routing in General
2. Link state routing, OSPF Single Area
3. Dijkstra's algorithm
4. Equal Cost Multipath
5. Topology Changes
6. Security of OSPF
7. OSPF, multiple areas
8. Other uses of Link State
9. Software Defined Networking (SDN)

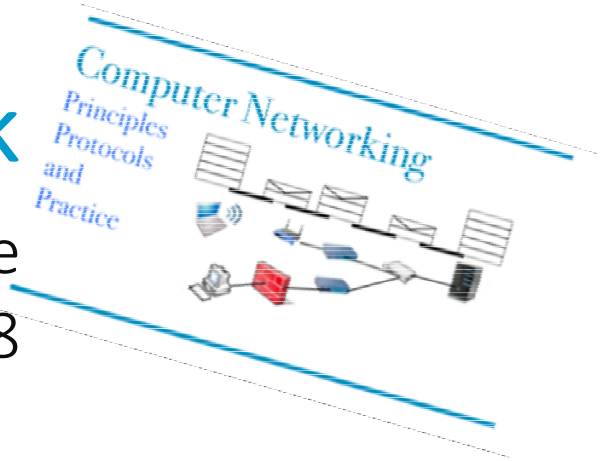
## Song

https://youtu.be/aPtr43KHBGk


## Textbook

Section 5.1.1, The control plane

RFC 2328



# Why were routing protocols invented?

Routing is often taken as synonym for “IP packet forwarding” : “this packet was routed to destination”.

recall that IP paket forwarding uses a routing table (also called “packet forwarding table”)

routing tables can be set manually (as in the lab) but this is time-consuming and error-prone

A routing protocol is a means to automatically compute the routing tables in a number of routers.

# Taxonomy of Routing Protocol Methods

## Link State

all routers in one domain (e.g. a campus, an ISP) know a map of the entire domain – obtained by gossiping (= flooding map information) with other routers

every link on the map has a cost e.g.cost(1 Gb/s link)=1; Cost(100 Mb/s link)=10.

routers compute next hop to destination by computing shortest paths based on their maps

(other algorithms are possible, e.g. paths with smallest latency, best with largest available bit rate, etc.)

used with interior routing (within one domain- OSPF, IS-IS) and in advanced bridging methods (TRILL, SPB Shortest Path Bridging)

## Distance Vector

all routers in one domain (e.g. a campus, an ISP) know only their neighbours + distance to all destinations (no global map)

routers inform their neighbours of their own list of distances to all destination (the vector of all distances)

used for interior routing (within one domain – RIP, EIGRP)

## Path Vector

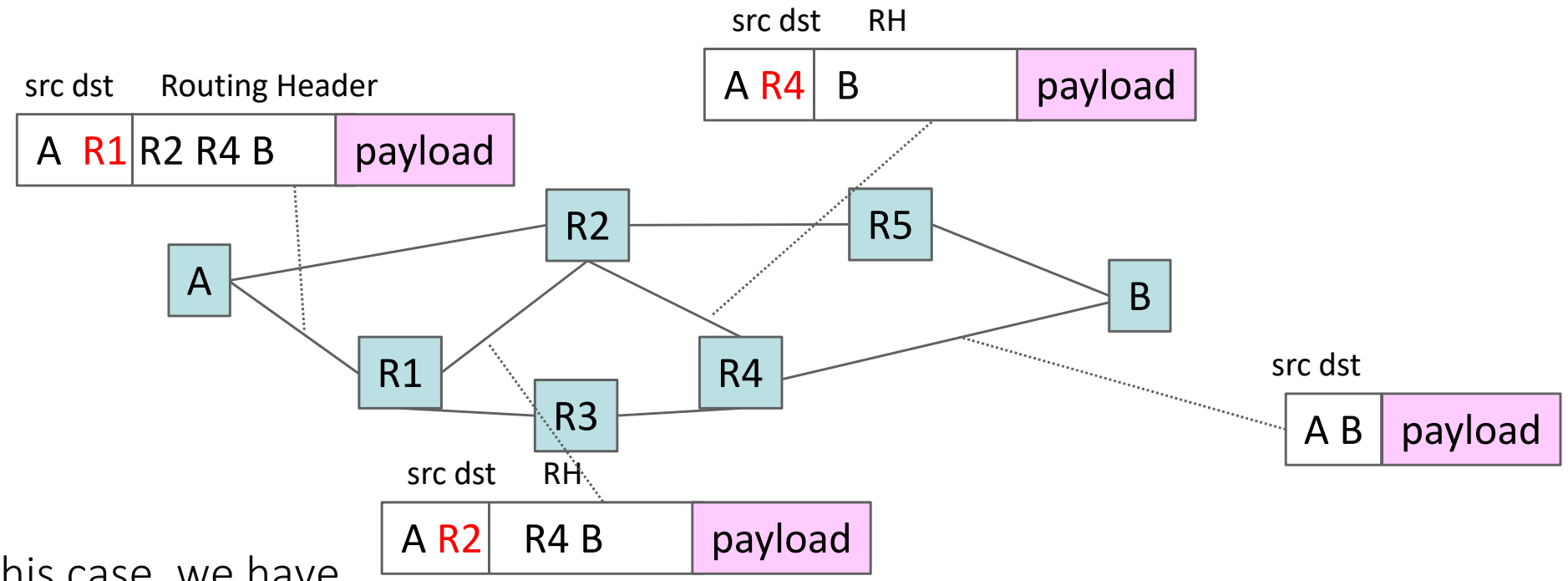
Every router knows only: its neighbours + explicit paths to all destinations

used with BGP -- for exterior routing (between domains).

# Source Routing

Paths are computed by the source host and put into packet headers. With IPv6, routing header is an extension header – contains intermediate hops and ultimate destination. When present, Destination Address is next intermediate hop.

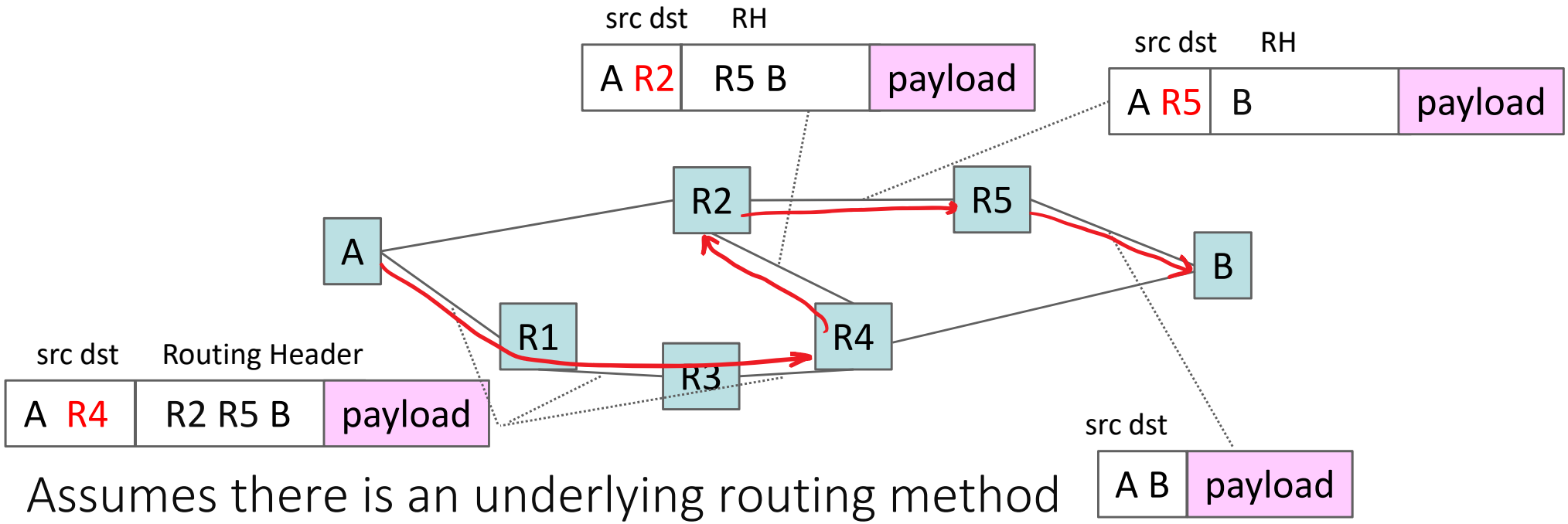
Used in ad-hoc networks (DSR) and in (old) Token Ring bridging. Here, route computation is done by a control application or by discovery e.g.: Source discovers path by flooding explorer packets that accumulate the path taken.




In this case, we have strict source routing, i.e. the path is the sequence of all intermediate hops. Intermediates systems are “dumb”.

# Loose Source Routing

= force some intermediate hops, which need not be on-link




Assumes there is an underlying routing method such as link state routing, e.g., to go from A to R2. Allows fine grained control of traffic (traffic engineering, separation of customers).

# Segment routing

Generalizes loose source routing by allowing the RH to contain indications for *processing* by intermediate hop; for example, instruct R2 to perform security function (screening, traffic separation). Used notably in data centers.

# 2. OSPF with a Single Area

OSPF (Open Shortest Path First) is a very widespread link state routing protocol. We first study it in its simplest form (single area).

Every router has

- an interface database (describing its physical connections, learnt by configuration)
- an adjacency database (describing the neighbour states, learnt by the hello protocol)
- a link state database (the network map, learnt by flooding)

Hello protocol is used to discover neighbouring routers – and to detect failures.

When two routers become new neighbours they first synchronize their link state databases. Typically, one router is new and copies what the other already knows.

# Link State Database and LSAs

Once synchronized, a router sends and accepts link state advertisements (LSAs)

- Every router sends one LSA describing its attached networks and neighbouring routers
- LSAs are flooded to the entire area and stored by all routers in their link state database
- LSAs contains a sequence number and age; only messages with new sequence number are accepted and re-flooded to all neighbours. Sequence number prevents loops. Age field is used to periodically resend LSA (eg every 30 mn) and to flush invalid LSAs.

# A Toy Example

showing interface databases

At B

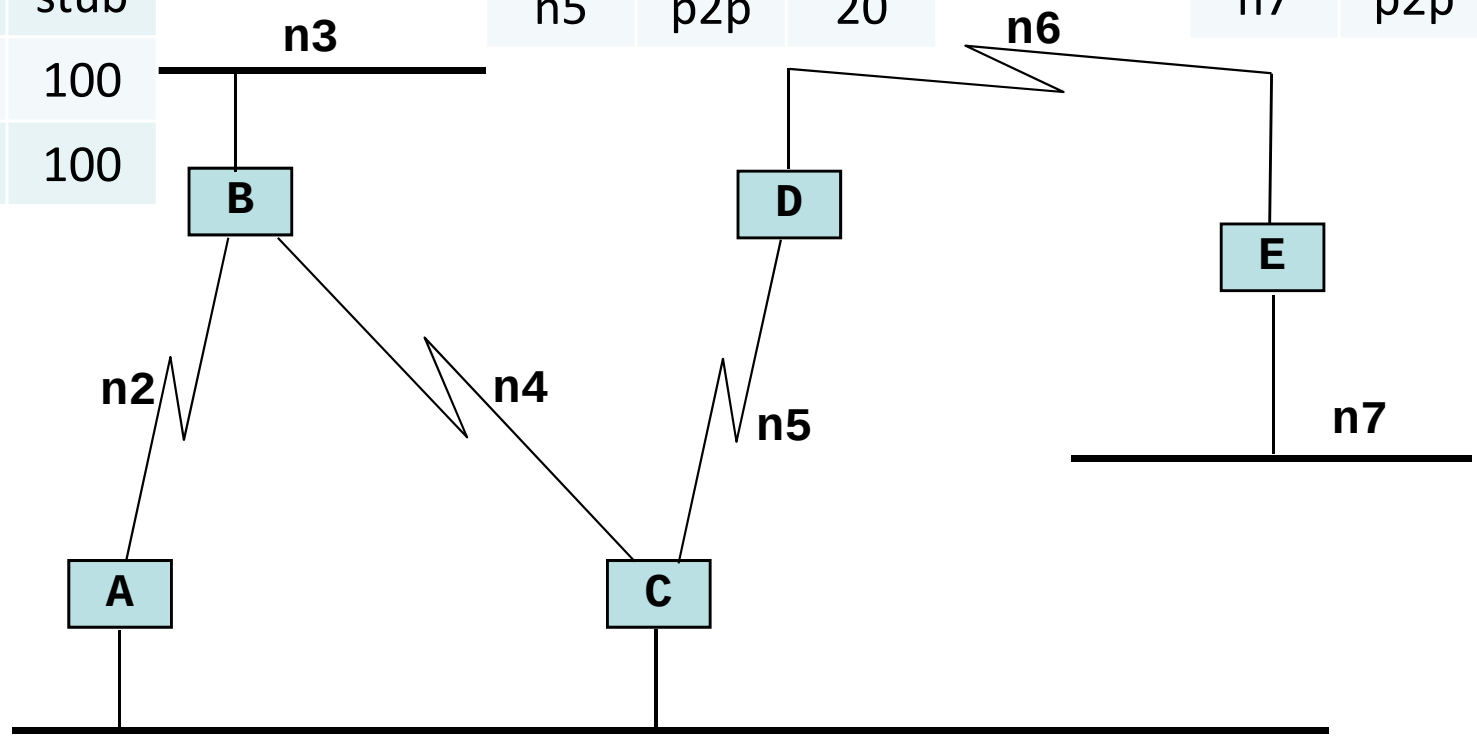
| Net | Type | cost |
|---|---|---|
| n3 | Eth | stub |
| n2 | p2p | 100 |
| n4 | p2p | 100 |

At D

| Net | Type | cost |
|---|---|---|
| n6 | p2p | 10 |
| n5 | p2p | 20 |

At E

| Net | Type | cost |
|---|---|---|
| n6 | p2p | 10 |
| n7 | p2p | 100 |



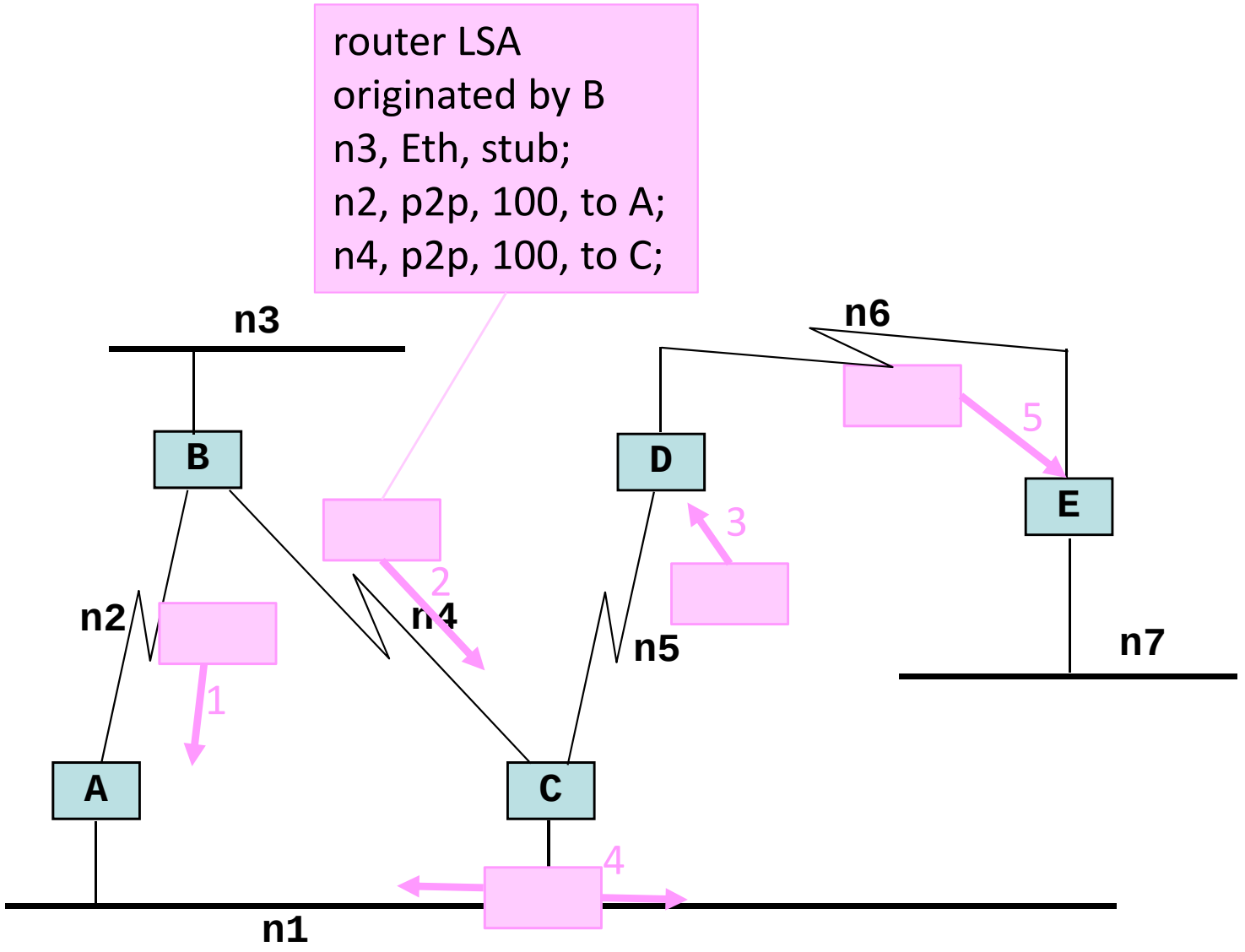

At A

| Net | Type | cost |
|---|---|---|
| n1 | Eth | 10 |
| n2 | p2p | 100 |

At C

| Net | Type | cost |
|---|---|---|
| n1 | Eth | 10 |
| n4 | p2p | 100 |

# Routers flood their LSAs throughout area



1, 2 B sends the LSA shown on the picture to A and C

the LSA describes all the networks attached to B and their costs, as well as the adjacent routers

“stub” network means non transit, ie there is no other router on this network

a stub network can be reached by only one router; all you need to know is how to reach this router so there is no need to allocate a cost to a stub network

3 C repeats the LSA (unmodified) to D

4 C also repeats the LSA to n1. Since n1 is Ethernet, the LSA is multicast to all OSPF routers on n1.
A receives the LSA but does not repeat the LSA on n1 because it received it on n1 from C

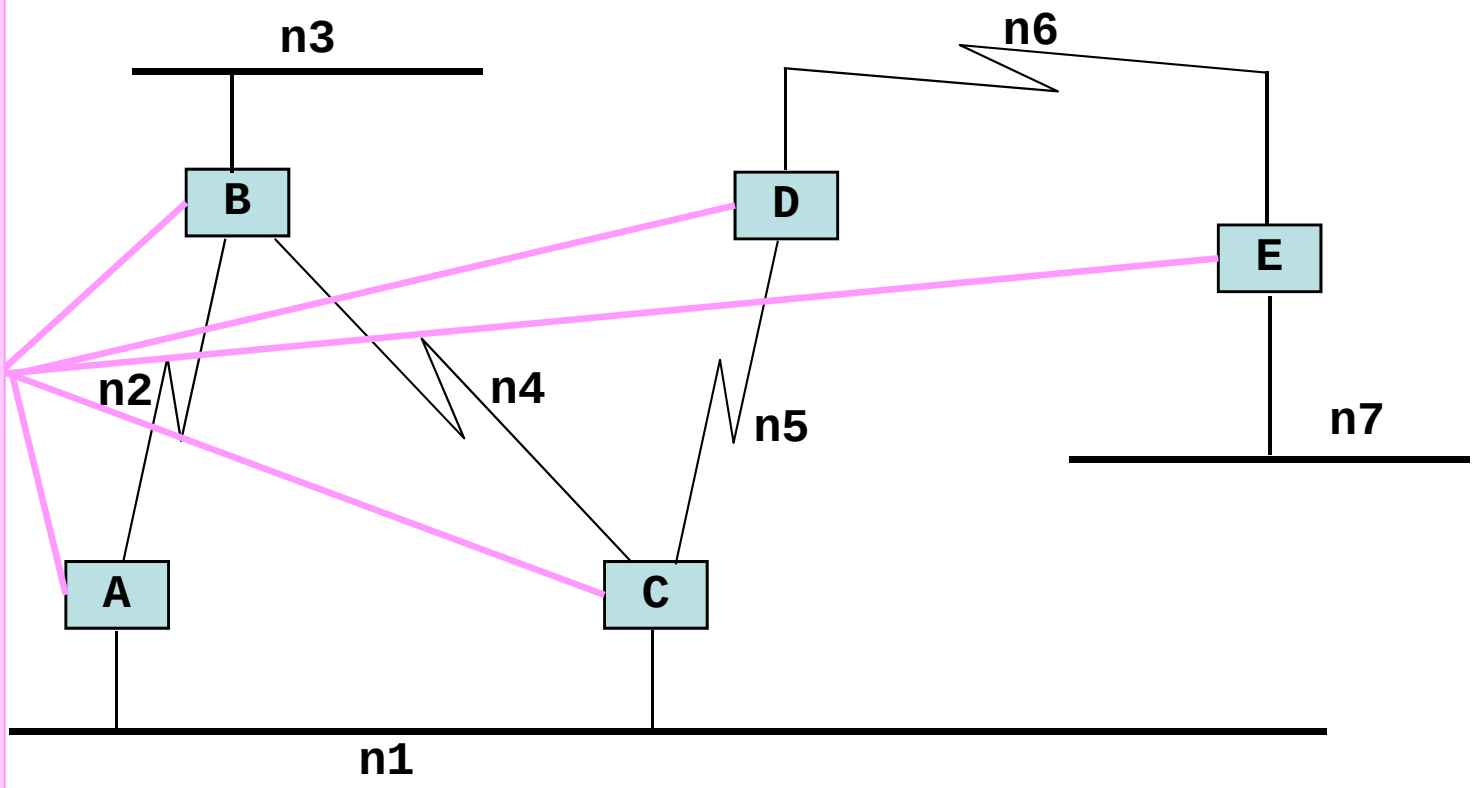
5 D repeats LSA to E

# After Flooding

After convergence, all routers have received all LSAs and store them in database.

All have the same database.

***router LSA from B***
n3, Eth, stub;
n2, p2p, 100, to A;
n4, p2p, 100, to C
***router LSA from A***
n2, p2p, 100, to B;
n1, eth, 10, DR=C
***router LSA from C***
n4, p2p, 100, to B;
n5, p2p, 20, to D;
n1, eth, 10, DR=C
***router LSA from D***
n5, p2p, 20, to C;
n6, p2p, 10, to E;
***router LSA from E***
n6, p2p, 10, to D;
n7, eth, stub
***network LSA from C***
n1, eth, 0, A, C




Link State Database at all routers

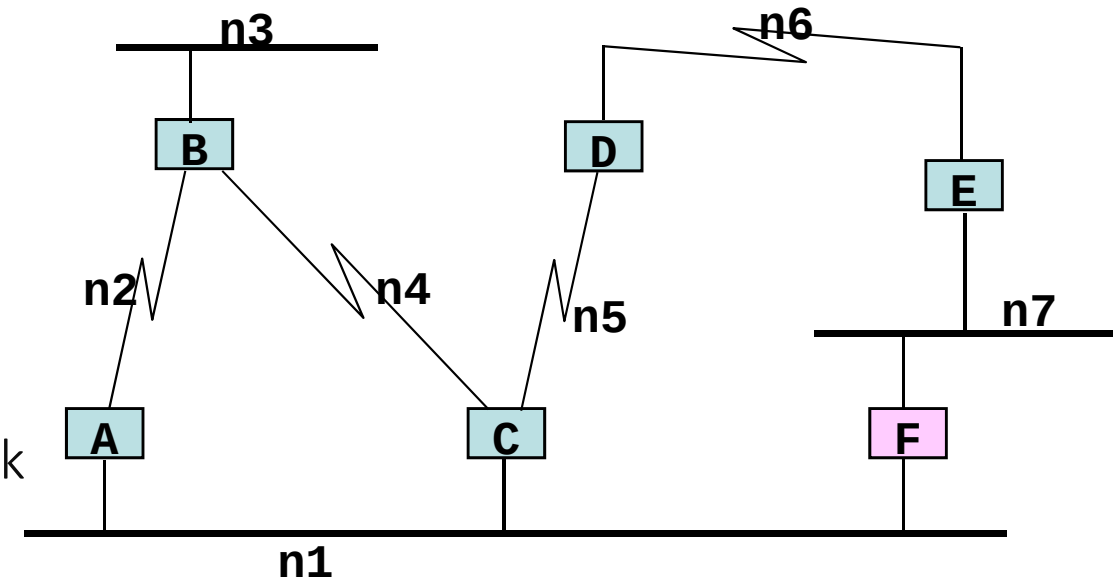
Ethernet LANs are treated in a special way. In order to avoid that every router on n1, for example, speaks to every other router on n1, the routers elect one designated router per LAN (and one backup designated router). Assume here the designated router is C.

Every router that is connected to an Ethernet LAN floods a "router LSA" indicating its connection to this LAN. The designated router "speaks for the switch" and sends a "network LSA" which gives the list of all routers connected to the LAN.

There are (at the time of writing) 11 types of LSAs. In addition to router and network LSAs, the other types are used in the multi-area case (later section) and with external routes (see BGP module). There are also other LSA types, called "opaque" that are used for purposes other than shortest path routing: opaque LSAs are not used by Dijkstra's algorithm. They can be used by OSPF extensions that make use of the link-state database for other purposes (e.g. type 10 LSAs carry information about reservable bandwidth, to be used by QoS routing).

# Toy example (cont'd): Router F boots

F discovers neighbours with the hello protocol; assume F discovers C first (C is designated router for n1): F and C establish adjacency (going through a sequence of 8 states, Down to Full). During this process, F and C synchronize their Link State Data Bases (i.e. F copies its LSDB from C). When the state is Full, synchronization is complete and F can now flood a router LSA saying that it is attached to n1; C (as designated router) sends a network LSA to say that F is now on n1.

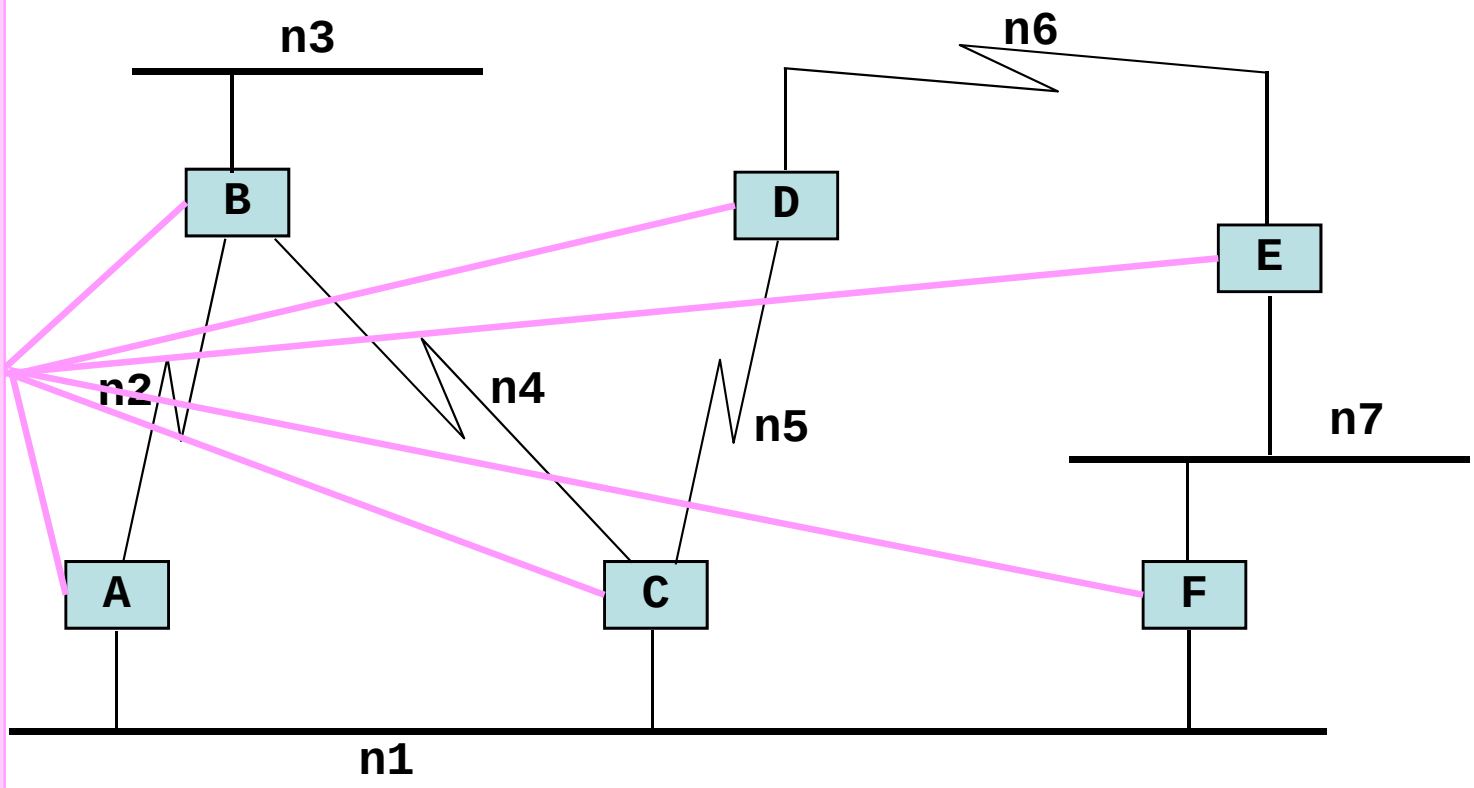



Then a similar process occurs between F and E, but now the synchronization is very fast since F already has a synchronized link-state database

# After Flooding

After convergence, all routers have received all new and modified LSAs (in red).

***router LSA from B***
n3, Eth, stub;
n2, p2p, 100, to A;
n4, p2p, 100, to C
***router LSA from A***
n2, p2p, 100, to B;
n1, eth, 10, DR=C
***router LSA from C***
n4, p2p, 100, to B;
n5, p2p, 20, to D;
n1, eth, 10, DR=C
***router LSA from F***
n7, eth, 10;
n1, eth, 10, DR=C
***router LSA from D***
n5, p2p, 20, to C;
n6, p2p, 10, to E;
***router LSA from E***
n6, p2p, 10;
n7, eth, 10, DR=E
***network LSA from C***
n1, eth, 0, A, C, F
***network LSA from E***
n7, eth, 0, E, F

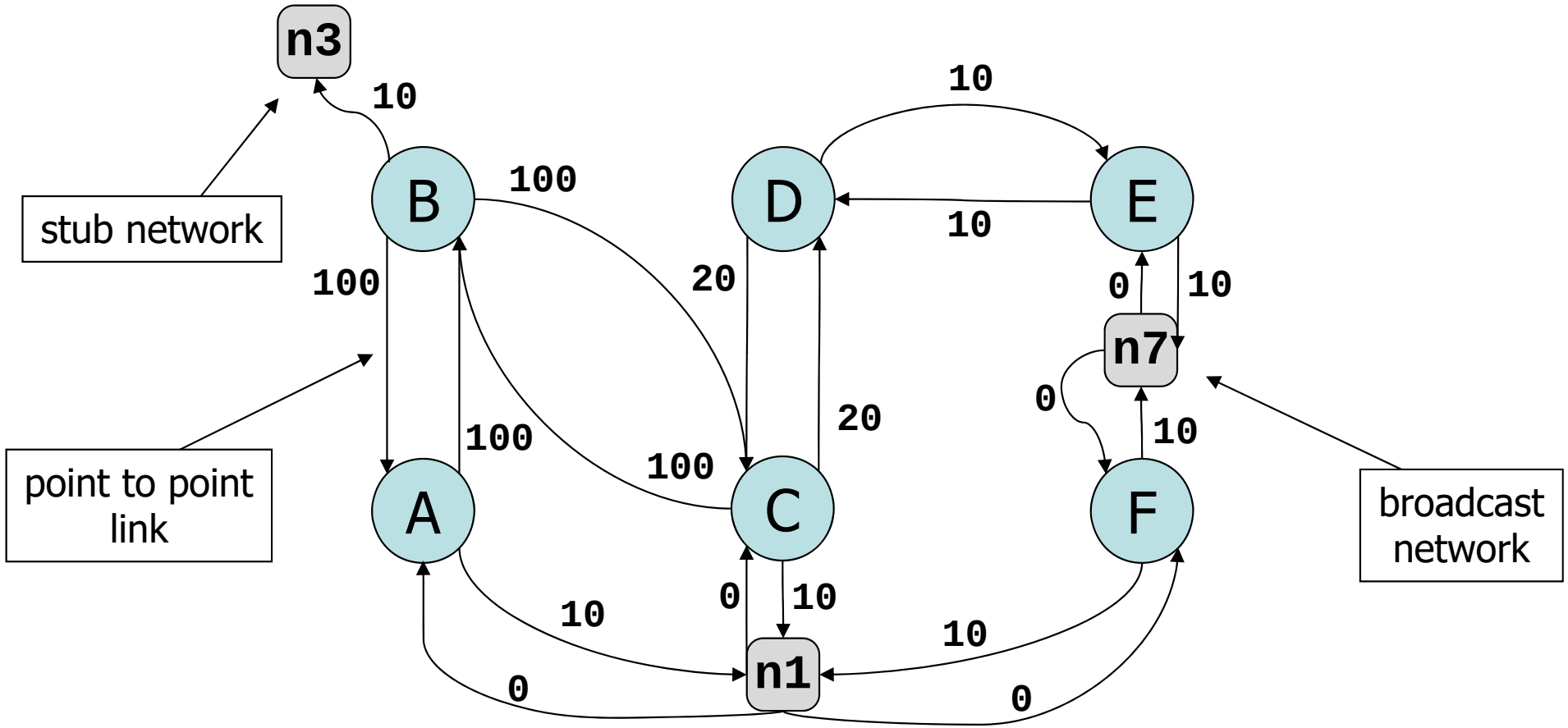



Link State Database at all routers

# How OSPF views the topology graph

The link state database describes an oriented graph, with outgoing edge cost = cost given in LSA.

Every router and every Ethernet network corresponds to one node in the graph. Cost from network node to router node is 0, by default.



# Practical Aspects

OSPF packet are sent directly over IP (OSPF=protocol 89 (0x59)).

Reliable transmission is managed by OSPF with OSPF Acks and timers.

OSPFv2 supports IPv4 only

OSPFv3 supports IPv6 and dual-stack networks

OSPF routers are identified by a 32 bit number

OSPF areas are identified by a 32 bit number

# 3. Path Computation Uses Dijkstra's Algorithm

Performed at every router, based on link state database

Router computes one or several shortest paths to every destination from self

OSPF uses Dijkstra's shortest path

- the best known algorithm for centralized operation

Paths are computed independently at every node

- link state database is same at all routers, but every router performs a different computation as it computes paths from self
- synchronization of databases guarantees absence of persistent loops

# Dijkstra's Shortest Path Algorithm

$m(0) = 0;\ m(i) = \infty\ \forall\ i \neq 0; V = \emptyset\ ; pred(i) = \emptyset\ \forall i;$

for $k = 0: N$ do

find $i \notin V$ that minimizes $m(i)$

if $m(i)$ is finite

add $i$ to $V$

for all neighbours $j \notin V$ of $i$

if $m(i) + c(i,j) < m(j)$

$m(j) = m(i) + c(i,j)$

$pred(j) = \{i\}$

else if $m(i) + c(i,j) = m(j)$

$m(j) = m(i) + c(i,j)$

$pred(j) = pred(j) \cup \{i\}$

The nodes are $0 \ldots N$ ; the algorithm computes shortest paths from node 0.

$c(i,j)$: cost of link $(i,j)$.

$V$: set of nodes visited so far.

$pred(i)$: estimated set of predecessors of node $i$ along a shortest path (multiple shortest paths are possible).

$m(j)$: estimated distance from node 0 to node $j$.

At completion, $m(i)$ is the true distance from $0$ to $i$.

# Dijkstra's Shortest Path Algorithm

$m(0) = 0;\ m(i) = \infty\ \forall\ i \neq 0; V = \emptyset\ ; pred(i) = \emptyset\ \forall i;$

for $k = 0: N$ do

find $i \notin V$ that minimizes $m(i)$

if $m(i)$ is finite

add $i$ to $V$

for all neighbours $j \notin V$ of $i$

if $m(i) + c(i,j) < m(j)$

$m(j) = m(i) + c(i,j)$

$pred(j) = \{i\}$

else if $m(i) + c(i,j) = m(j)$

$m(j) = m(i) + c(i,j)$

$pred(j) = pred(j) \cup \{i\}$

Builds the shortest path tree from this node to all nodes.

Adds one node at a time to the working set $V$, by picking the node that is closest in the sense of the best estimation of the distance that we have at this time

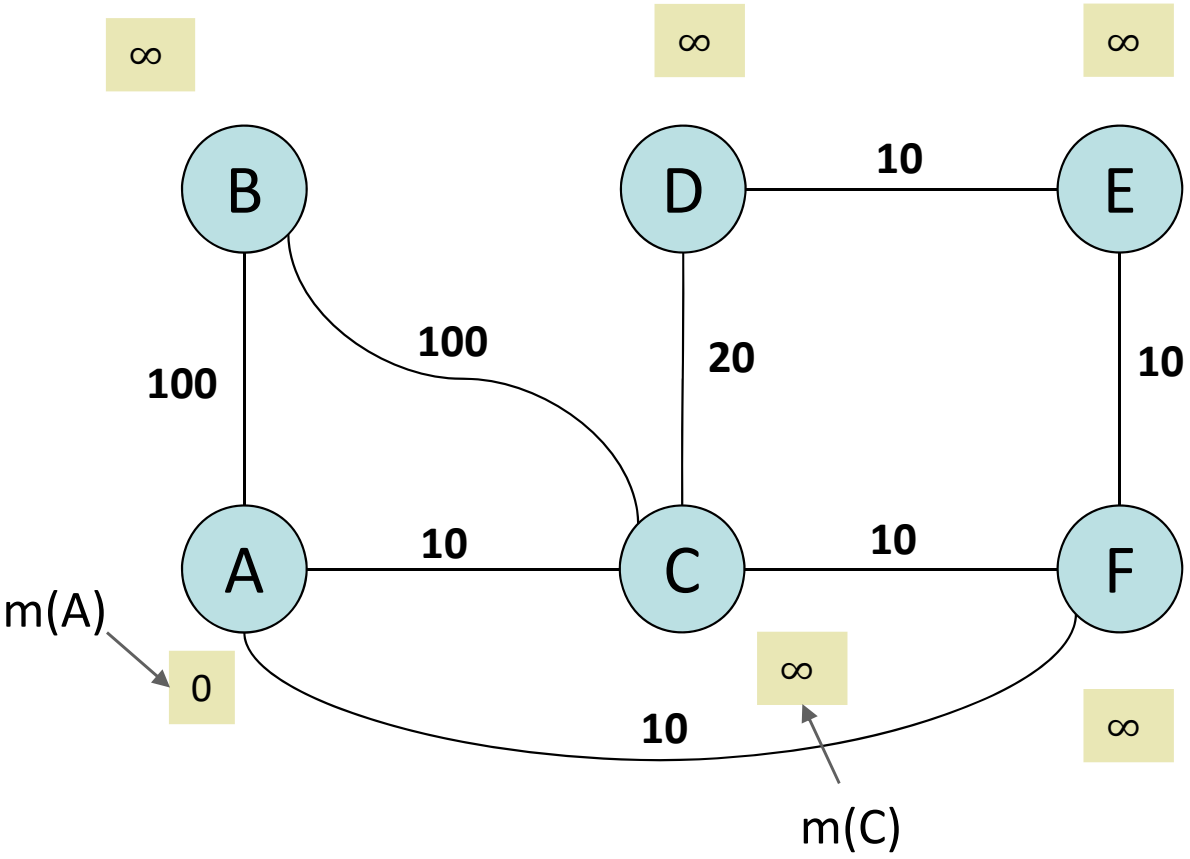
There are multiple versions of Dijkstra's algorithm. The presented version finds all shortest paths, other versions find only one shortest path to every destination. The version presented is very close to what is really implemented in OSPF (with a difference, next-hop versus pred(), see later).

The worst-case complexity of this version is $O(N^2)$ where $N$ is the number of nodes. More efficient versions of the algorithm have a smaller complexity, $O(N \log N + E)$ where $E$ is the number of links.

The algorithm adds nodes to the visited set by increasing distances from node $0$. It is greedy in the sense that at every step it adds one node to the set of visited nodes; the state of this node (distance from node $0$ and set of predecessors) is the final value and will not change in later steps of the algorithm.

The last 3 lines are for handling equal cost shortest paths. If one is interested in finding only one shortest path per destination, these 3 lines are deleted.

# Example: Dijkstra at A Initially

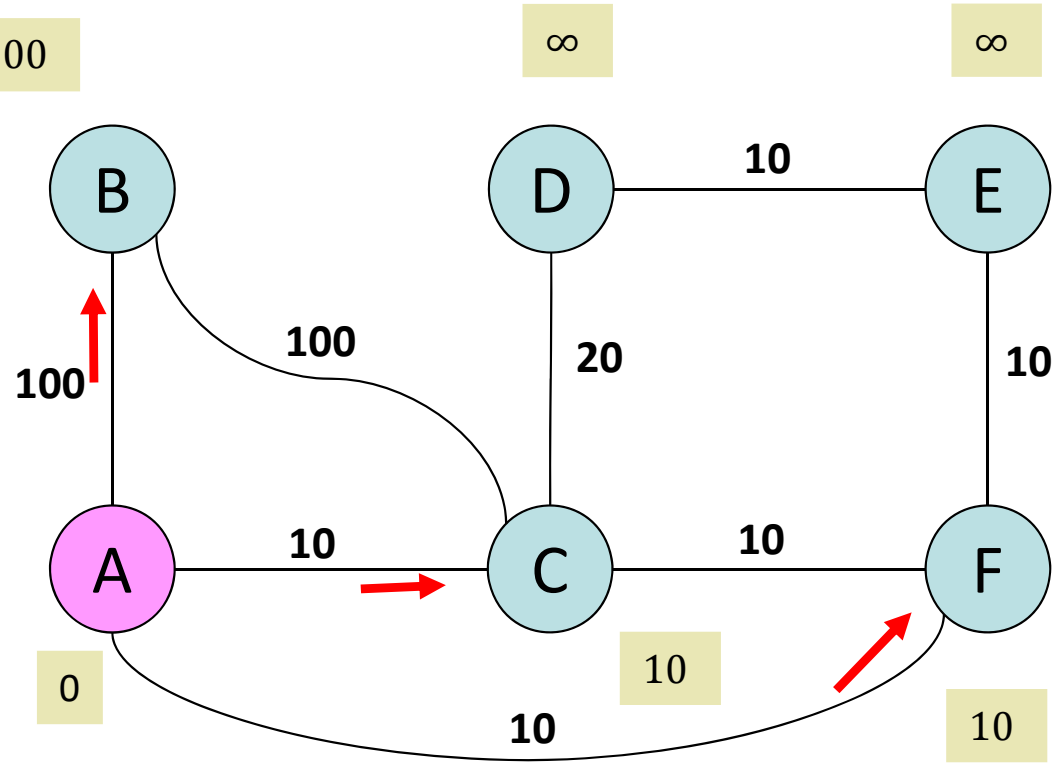



init: V = Ø

m(A)=0
m(i)=∞, $i \neq A$

pred(i)=Ø

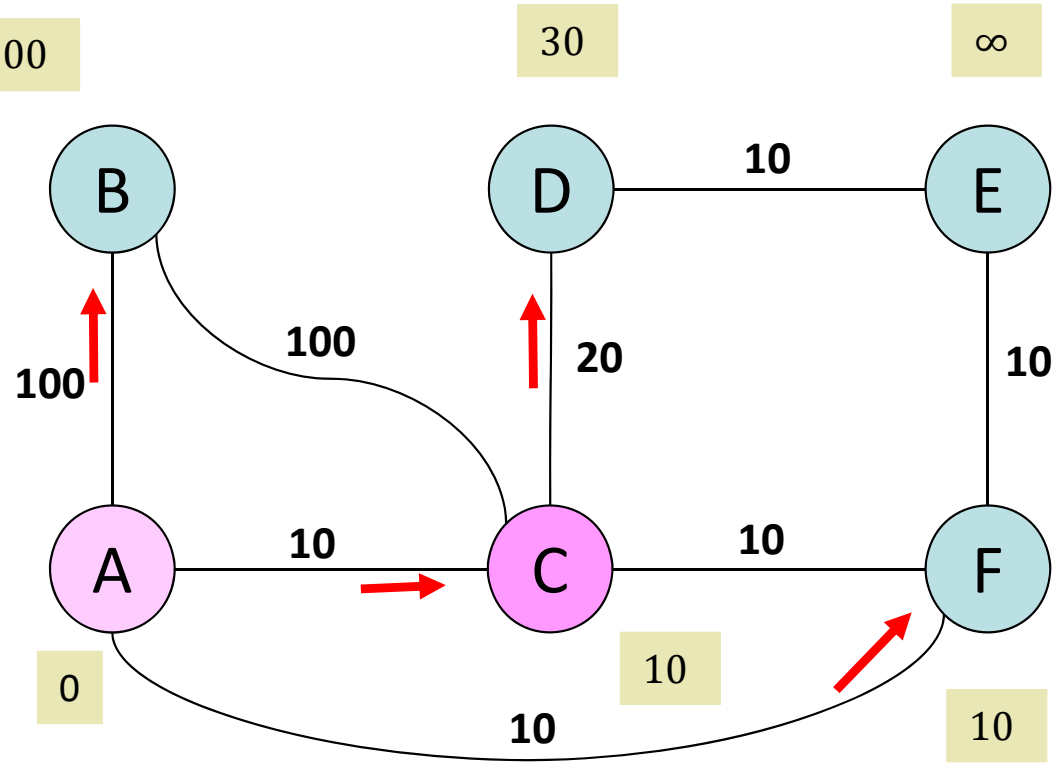
# Example: Dijkstra at A
# After step 1




step 1:
i=A
V={A}
m(B)=100
pred(B)={A}
m(C)=10
pred(C)={A}
m(F)=10
pred(F)={A}

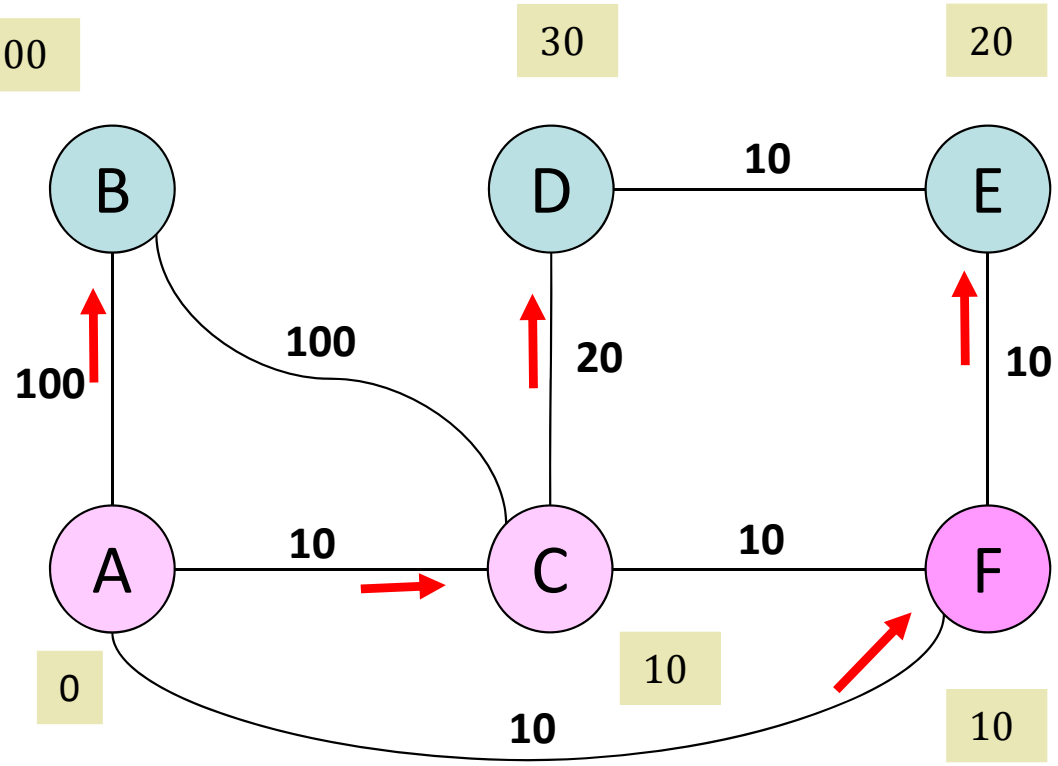

red arrow from A to B means pred(B)=A

# Example: Dijkstra at A
# After step 2



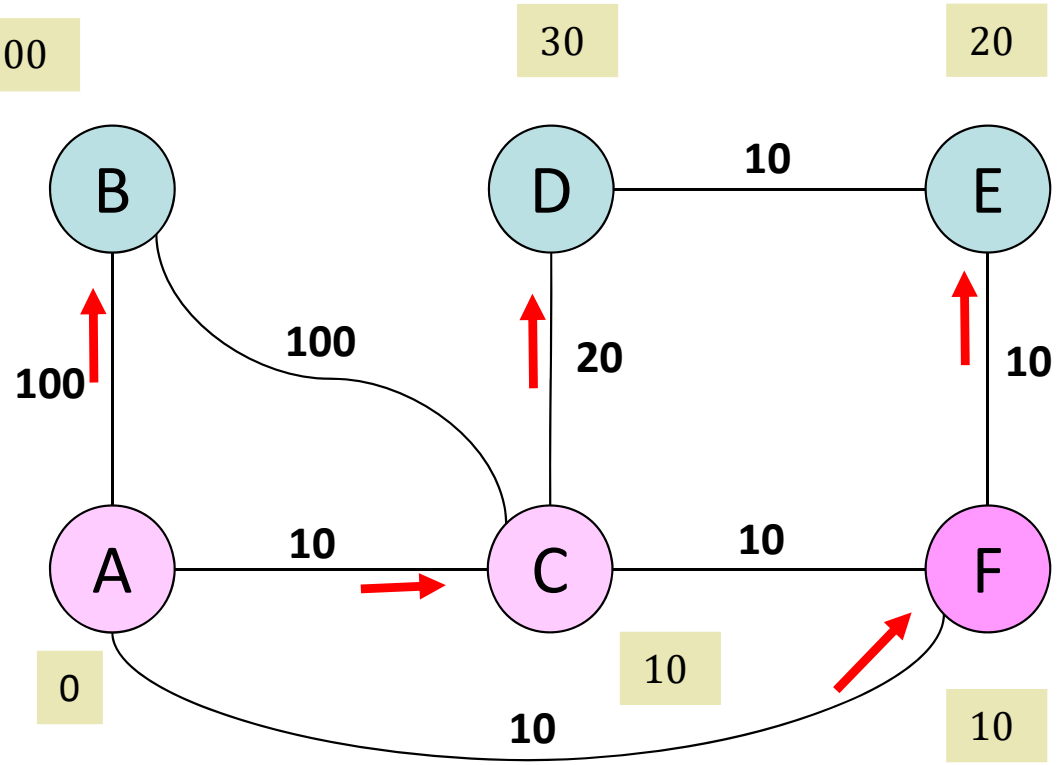

step 2:
i=C
V={A,C}
B, F unchanged
m(D)=30
pred(D)={C}

# Example: Dijkstra at A
# After step 3



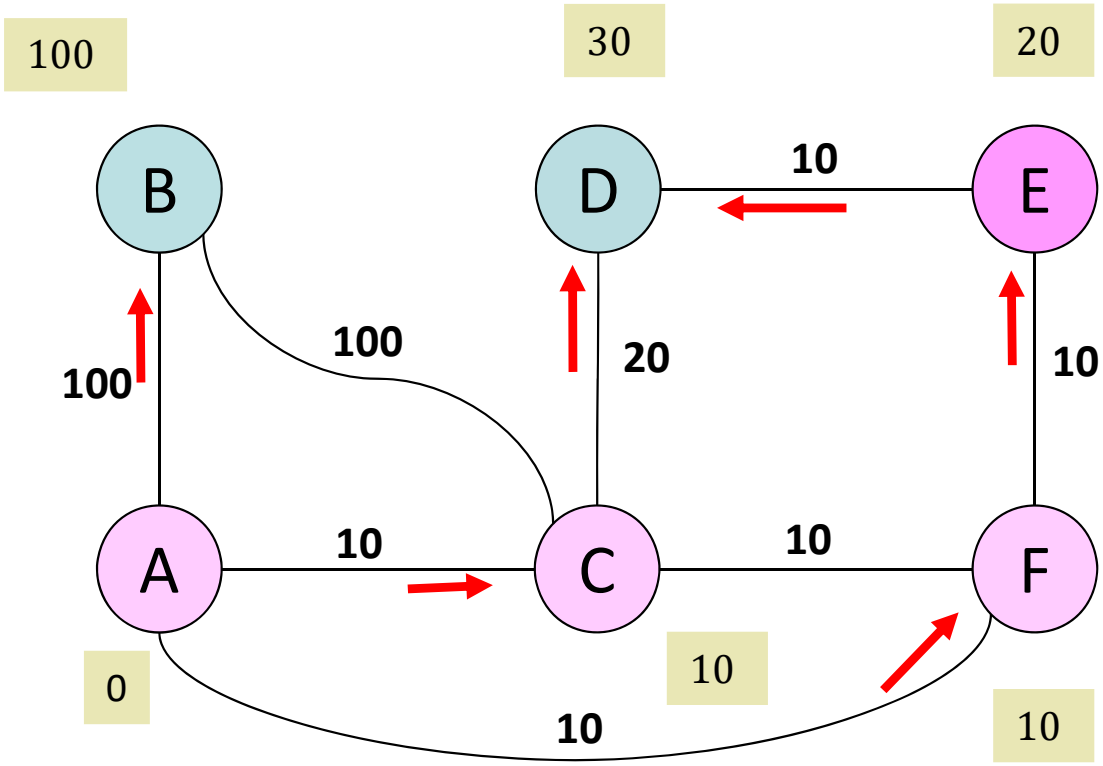

step 3:
i=F
V={A,C,F}
m(E)=20
pred(E)={F}

# At next step, which node will be added to the working set $V$?




A. B
B. D
C. E
D. I don't know

# Solution: Dijkstra at A
# After step 4




step 4: (Answer C)

i=E

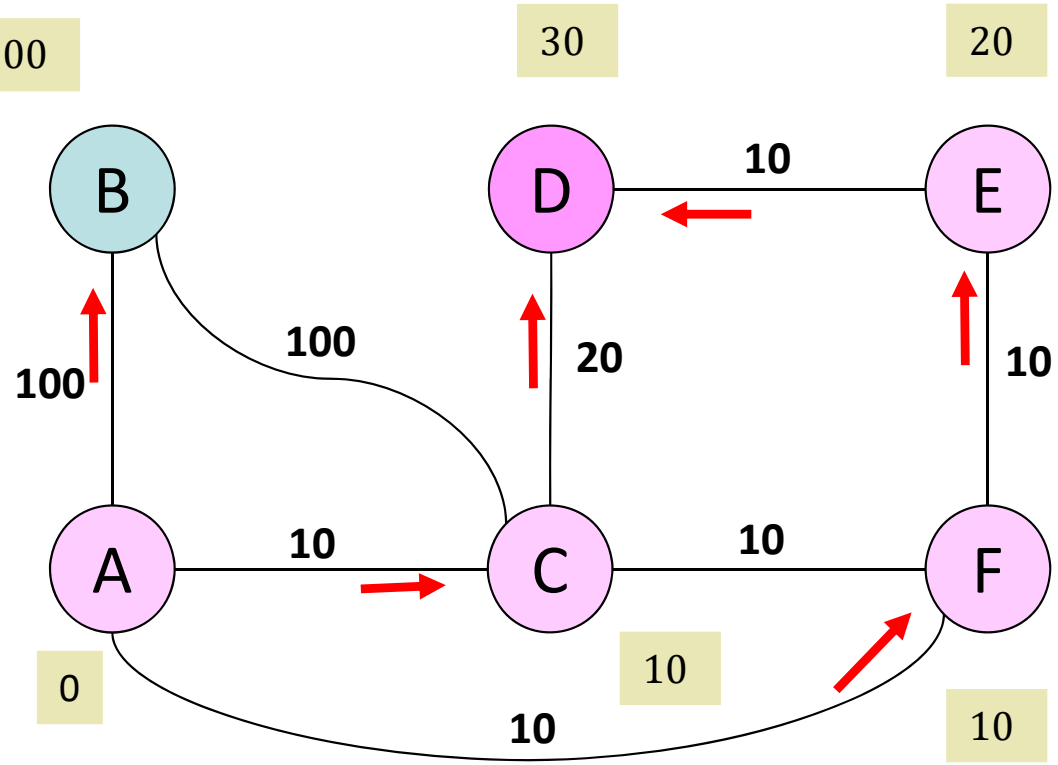
V={A,C,E,F}

m( ) unchanged

pred(D)={C,E}

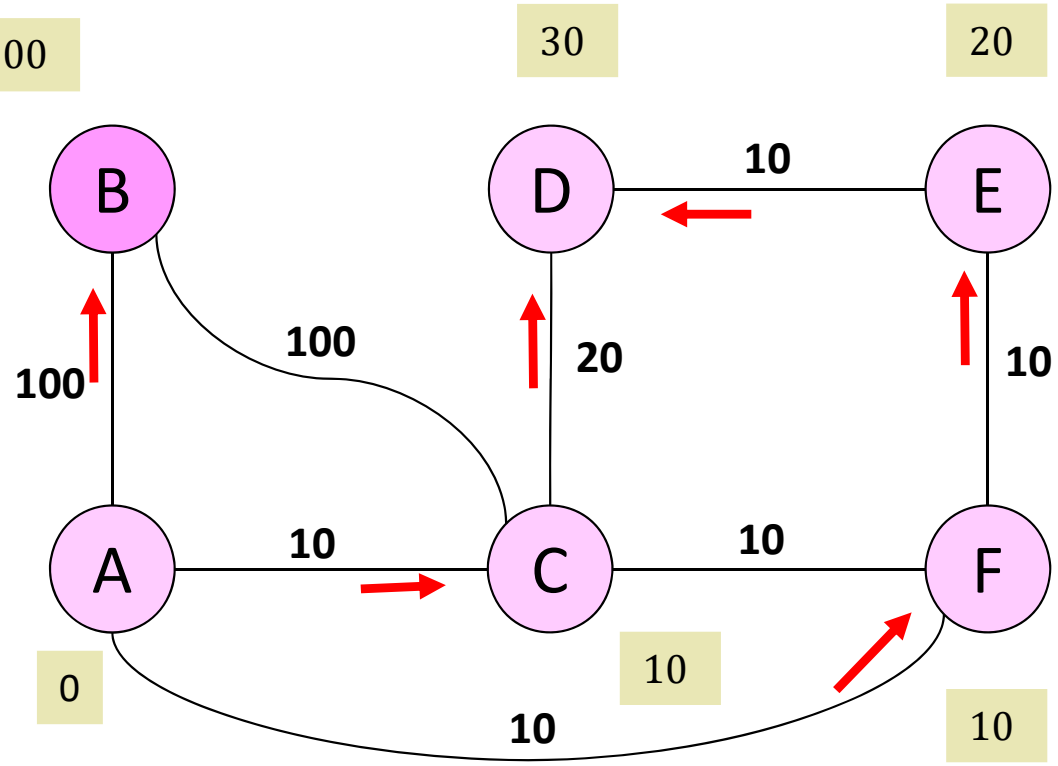
There are two equal-cost paths to D, both are recorded.

# Example: Dijkstra at A
# After step 5




step 5:
i=D
V={A,C,D,E,F}

# Example: Dijkstra at A
# After step 6




step 6:

i=B

V={A,B,C,D,E,F}

this is the final state

# Path Computation

$pred(i)$ gives the set of predecessors of node $i$ on all shortest paths from source to $i$

Shortest paths can be computed backwards, using pred(), starting from destination.

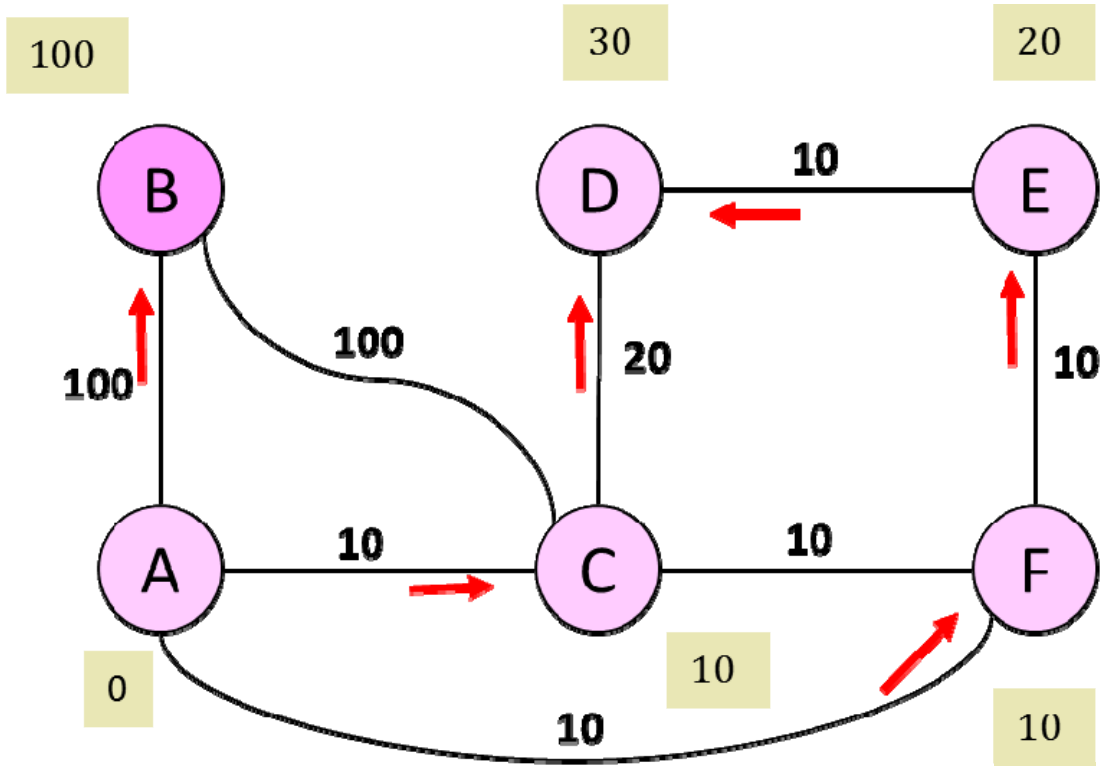
Shortest paths from A to D :

A-C-D

A-F-E-D

to E :

A-F-E

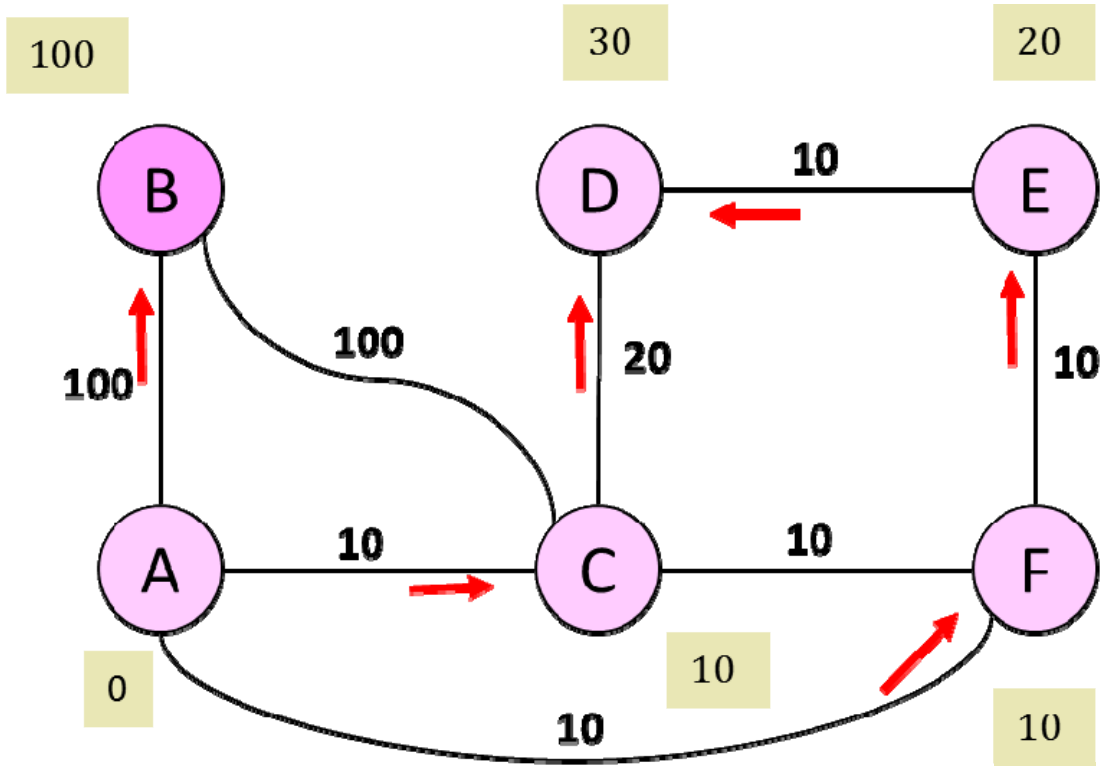


# Routing Table

Router A keeps in its routing table the next-hop and the distance to every destination (not the entire path):

At A

| Dest | Next-hop | cost |
|---|---|---|
| B | B | 100 |
| C | C | 10 |
| D | C or F | 30 |
| E | F | 20 |
| F | F | 10 |



The version of Dijkstra used in OSPF differs from is presented above in that pred() is not used. Instead, the next hop is directly computed during the main loop of the algorithm. This is faster than computing the paths separately, but makes the algorithm more difficult to understand:

$m(0) = 0;\ m(i) = \infty\ \forall\ i \neq 0; V = \emptyset\ ; nextHopTo(i) = \emptyset\ \forall i;$

for $k = 0{:}\,N$ do

find $i \in V$ that minimizes $m(i)$

if $m(i)$ is finite

add $i$ to $V$

for all neighbours $j \in V$ of $i$

if $m(i) + c(i,j) < m(j)$

$m(j) = m(i) + c(i,j)$

derive $nextHopTo(j)$ from $i$

else if $m(i) + c(i,j) = m(j)$

$m(j) = m(i) + c(i,j)$

augment $nextHopTo(j)$ from $i$

derive $nextHopTo(j)$ from $i$:

if $i == 0$

$nextHopTo(j) = \{j\}$ // $j$ is directly connected to $0$

else

$nextHopTo(j) = nextHopTo(i)$ // shortest path to $j$ is via $i$

augment $nextHopTo(j)$ from $i$:

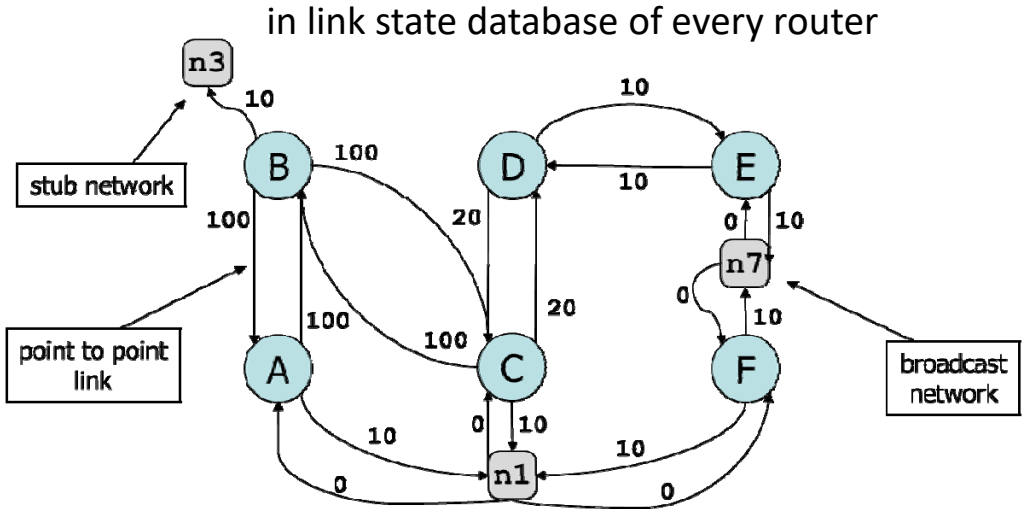
if $i == 0$

$nextHopTo(j) = \{j\}$ // $j$ is directly connected to $0$

else

$nextHopTo(j) = nextHopTo(j) \cup nextHopTo(i)$ // add shortest path to $j$ via $i$

# OSPF Shortest Path Computation




Routing table at A

| Dest | Next-hop | cost |
|---|---|---|
| B | On-link | 100 |
| C | On-link | 10 |
| n1 | On-link | 10 |
| D | F | 30 |
| D | C | 30 |
| n7 | F | 30 |
| E | F | 20 |
| F | On-link | 10 |
| n3 | B | 110 |

The previous slides showed a very simple graph. In practice, OSPF adds to the graphs nodes to networks, which makes the graph bigger.

To optimize the computation, stub network are removed before applying Dijkstra. Then Dijkstra is run and the routing table contains distances and next hop to routers such as B. Then stub networks such as n3 are added to the routing table one by one, using the information on how to reach the routers such as B that lead to the stub networks.

# 4. Equal Cost Multipath

OSPF supports multiple shortest paths

IP allows to have multiple next-hops to the same destination in the routing table

This is good as it allows to exploit the multiplicity of paths that exist in many networks.

Routing table at A

| Dest | Next-hop | cost |
|---|---|---|
| B | On-link | 100 |
| C | On-link | 10 |
| n1 | On-link | 10 |
| D | F | 30 |
| D | C | 30 |
| n7 | F | 30 |
| E | F | 20 |
| F | On-link | 10 |
| n3 | B | 110 |

What should router A do when it has several packets to send to destination D ?

A. send them to next-hop F or C with equal probability
B. choose one next-hop and send all packets to this next-hop
C. test the availability of the next-hop before sending
D. something else
E. I don't know

# Solution: Equal Cost Multi-Path often uses Per-Flow Load Balancing

It is better to use all available paths network (load balancing)
⇒ send to all next-hops with equal probability.

However, this may cause packet re-ordering, which is possible but not desirable as it reduces the performance of TCP (TCP might think that a packet is lost when it is out of sequence). Therefore, an alternative approach, called *per-flow load balancing* requires that packets of the same *flow* are sent to the same next-hop. A flow is identified by the source and destination addresses, next header type and (if they exist), source and destination ports.

Per-flow load balancing is implemented using a hash function $h(m) \in [0,1]$ applied to the flow identifier $m$. Assume there are 2 possible next-hops. If $h(m) < 0.5$ the packet is sent to the first, else to the second. The flow identifier (combination of source and destination IP addresses and ports) is the same for all packets of the same TCP connection, so they will be sent to the same next-hop.

# 5. Changes to Topology

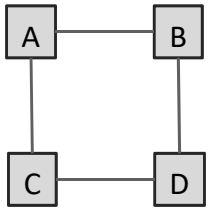
Changes to topology occur e.g. when routers or links crash or are rebooted.

link or router failures are detected by OSPF's hello protocol (after several seconds, in general) or by the BFD protocol at a lower layer (fast: after 10 ms - Bidirectional Forwarding Detection, a hello protocol independent of OSPF).

When a router sees a change in the state of a links or a neighbouring router, it sends a new LSA to all its neighbours. All routers update their link state database and propagate the change to the entire OSPF area.

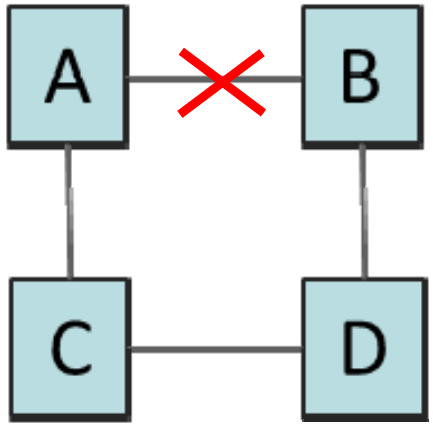
Changes to link state database trigger re-computation of shortest-paths and routing tables.

Link State Database and routing table at A

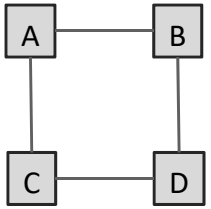



| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| B | east | on-link | 10 |
| C | south | on-link | 10 |
| D | east | B | 20 |
| D | south | C | 20 |

# Example at $t_0$

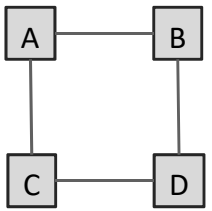



Link State Database and routing table at B



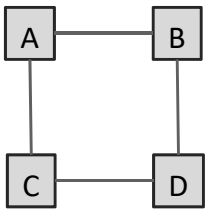

| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| A | west | on-link | 10 |
| D | south | on-link | 10 |
| C | west | A | 20 |
| C | south | D | 20 |

Link State Database and routing table at C




| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| A | north | on-link | 10 |
| D | east | on-link | 10 |
| B | north | A | 20 |
| B | east | D | 20 |

Link State Database and routing table at D

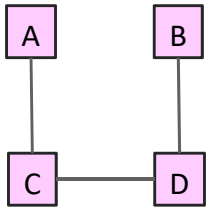



| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| B | north | on-link | 10 |
| C | west | on-link | 10 |
| A | north | B | 20 |
| A | west | C | 20 |

$t_0$: Link A-B crashes

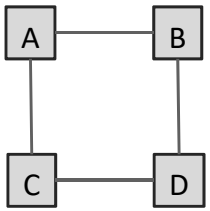
# Example at $t_1$

Link State Database and routing table at A



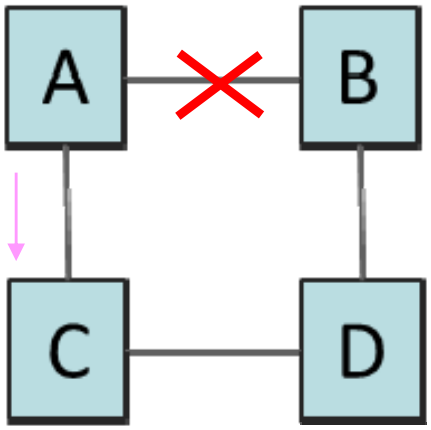

| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| B | south | C | 30 |
| C | south | on-link | 10 |
| ~~D~~ | ~~east~~ | ~~B~~ | ~~20~~ |
| D | south | C | 20 |

Link State Database and routing table at B

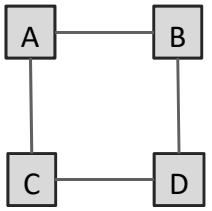



| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| A | west | on-link | 10 |
| D | south | on-link | 10 |
| C | west | A | 20 |
| C | south | D | 20 |

LSA from A
A to C, cost=10

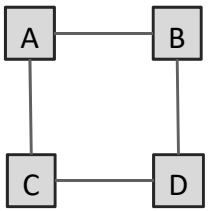



Link State Database and routing table at C




| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| A | north | on-link | 10 |
| D | east | on-link | 10 |
| B | north | A | 20 |
| B | east | D | 20 |

Link State Database and routing table at D

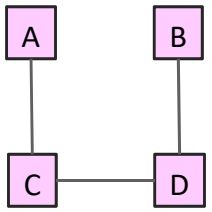



| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| B | north | on-link | 10 |
| C | west | on-link | 10 |
| A | north | B | 20 |
| A | west | C | 20 |

$t_1$: A detects failure first; declares B as invalid neighbour, declares link A-B as invalid, updates its link state database, sends a new LSA to C, with origin A and recomputes routing table. A routing loop exists between A and C. Traffic sent by B to A dies on the link. Half of the traffic from D to A is lost.

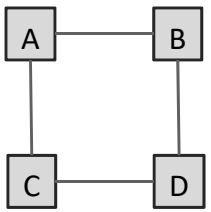
# Example at $t_2$

Link State Database and routing table at A

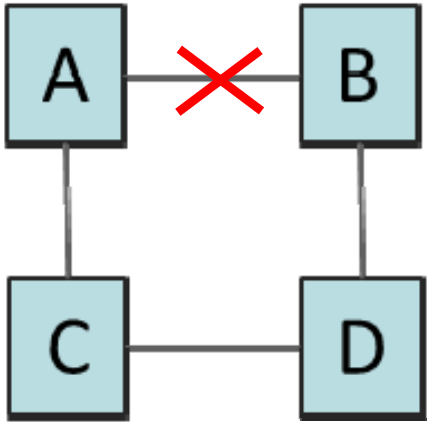



| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| B | south | C | 30 |
| C | south | on-link | 10 |
| ~~D~~ | ~~east~~ | ~~B~~ | ~~20~~ |
| D | south | C | 20 |

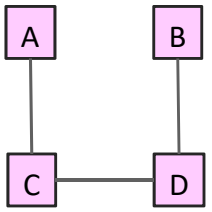
Link State Database and routing table at B




| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| A | west | on-link | 10 |
| D | south | on-link | 10 |
| C | west | A | 20 |
| C | south | D | 20 |



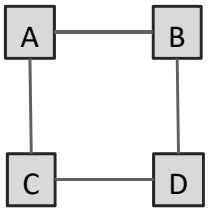

Link State Database and routing table at C




| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| A | north | on-link | 10 |
| D | east | on-link | 10 |
| ~~B~~ | ~~north~~ | ~~A~~ | ~~20~~ |
| B | east | D | 20 |

LSA from A
A to C, cost=10

Link State Database and routing table at D

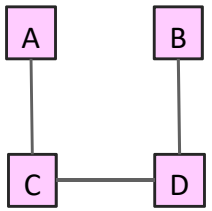



| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| B | north | on-link | 10 |
| C | west | on-link | 10 |
| A | north | B | 20 |
| A | west | C | 20 |

$t_2$: C receives LSA from A, updates its link state database, forwards this LSA to D and recomputes routing table. There is no routing loop but traffic sent by B to A dies on the link and half of the traffic from D to A is lost.

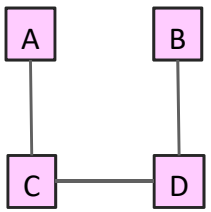
# Example at $t_3$

Link State Database and routing table at A

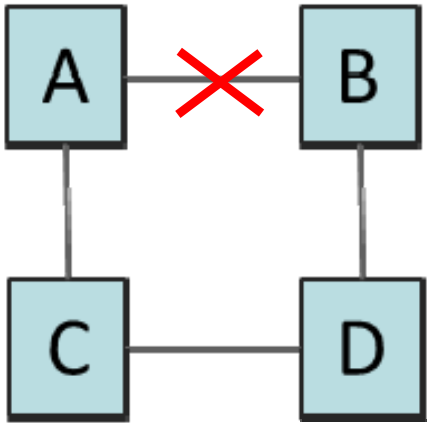



| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| B | south | C | 30 |
| C | south | on-link | 10 |
| ~~D~~ | ~~east~~ | ~~B~~ | ~~20~~ |
| D | south | C | 20 |

Link State Database and routing table at B



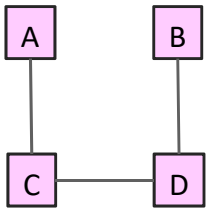

| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| A | south | D | 30 |
| D | south | on-link | 10 |
| ~~C~~ | ~~west~~ | ~~A~~ | ~~20~~ |
| C | south | D | 20 |




LSA from B
B to D, cost=10

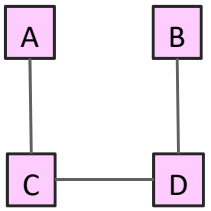
LSA from A
A to C, cost=10

Link State Database and routing table at C




| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| A | north | on-link | 10 |
| D | east | on-link | 10 |
| ~~B~~ | ~~north~~ | ~~A~~ | ~~20~~ |
| B | east | D | 20 |

Link State Database and routing table at D




| Dst | i/f | Nxt hp | cst |
|---|---|---|---|
| B | north | on-link | 10 |
| C | west | on-link | 10 |
| ~~A~~ | ~~north~~ | ~~B~~ | ~~20~~ |
| A | west | C | 20 |

$t_3$: D receives LSA from C, updates its link state database, forwards this LSA to B and recomputes routing table. At about the same time, B now also detects failure; declares A as invalid neighbour, declares link A-B as invalid, updates its link state database, sends a new LSA to D, with origin B and recomputes routing table. All link state databases now have the same contents and new routes are in place.

# When a router crashes, how do all routers in area detect the crash ?

A. The immediate neighbours detect loss of adjacency and flood new LSAs with the updated list of adjacent routers
B. By the hello protocol
C. By timeout of LSAs stored in their link-state database
D. By absence of BFD (Bidirectional Forwarding Detection) messages
E. I don’t know

# Solution

Answer A, in principle.

Answer C is some rare cases possible, but normally neighbours detect the failure well before the LSA ages out (1 hour by default)

With the hello protocol and BFD, only immediate neighbours detect the loss of the crashed router.

# 6. Security of OSPF

Attacks against routing protocols

1. send invalid routing information ⇒ disrupt network operation
2. send forged routing information ⇒ change network paths
3. denial of service attacks

OSPF security protects against 1. and 2. using authentication

OSPFv2 levels of authentication

type 0: none

type 1: password sent in cleartext in all packets

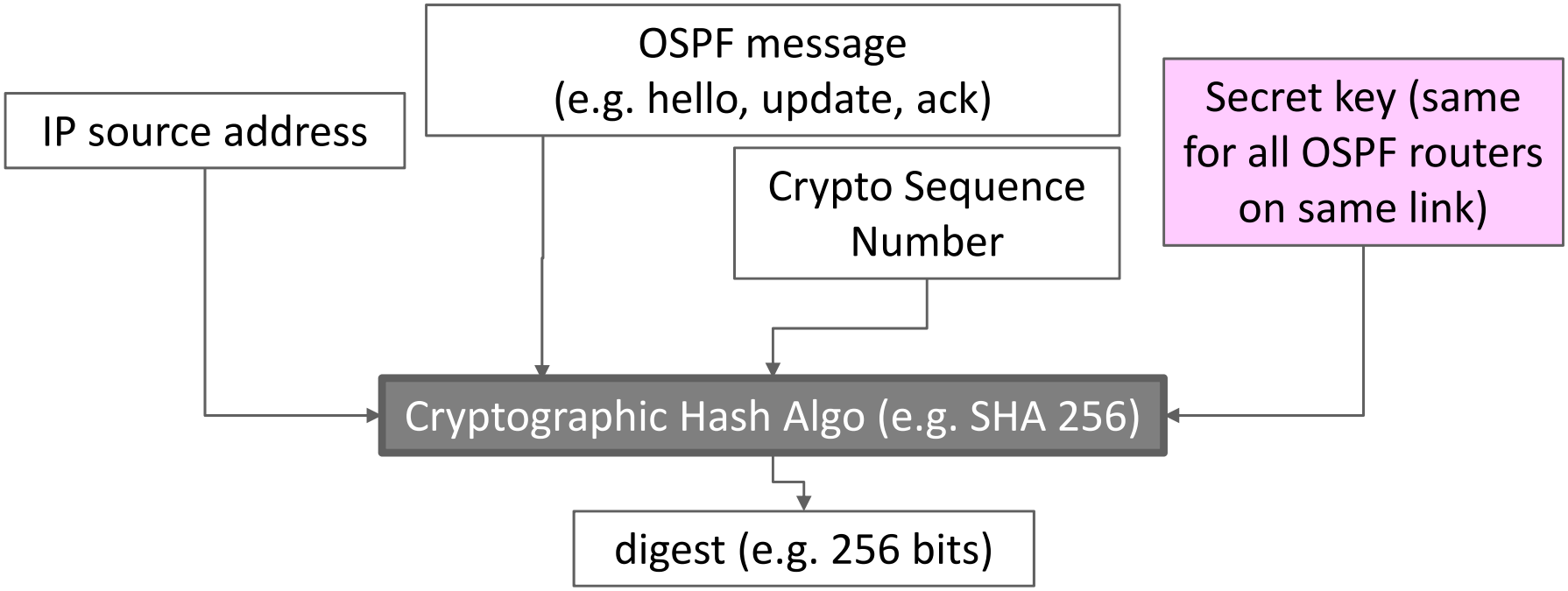
type 2: authentication using MD5 (obsolete) or HMAC-SHA

type 3: similar to type 2 with some improvements (RFC 7474)

OSPFv3 uses IPSEC authentication

similar to type 2 and 3 but at IP layer

# OSPF Type 3 Authentication uses secret, shared keys




Digest and Crypto Sequence Number are appended after OSPF message and sent in IP packet in cleartext.

Keys are shared, all routers on same link must have same pre-installed keys. Keys are expected to have a short lifetime. OSPF does not say how to install the keys, must be done out of band. A key index in authentication header in OSPF message says which key to use.

Crypto Sequence Number contains a permanent "boot count" saved on disk to avoid collision of numbers and is large enough to never wrap around (in $10^{11}$years). Avoids replay attacks.

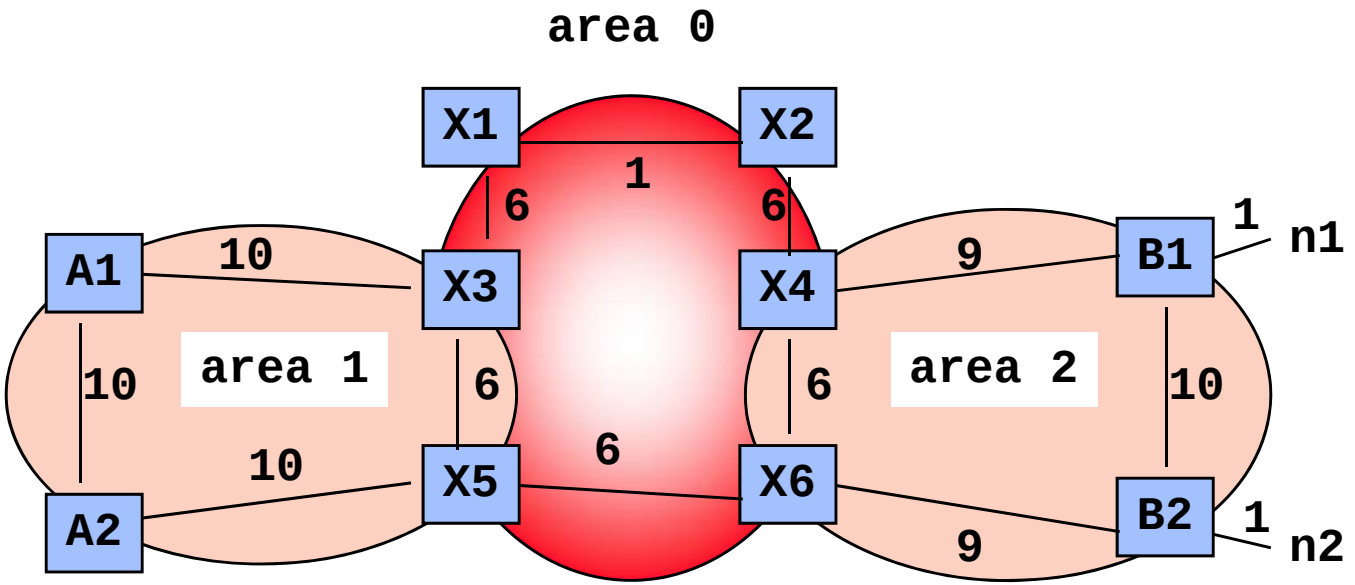
# 7. OSPF with Multiple Areas

Link state floods all information to all routers, therefore does not scale to very large networks.

OSPF uses a system of areas (i.e. a hierarchy of two routing levels) to scale to very large networks.

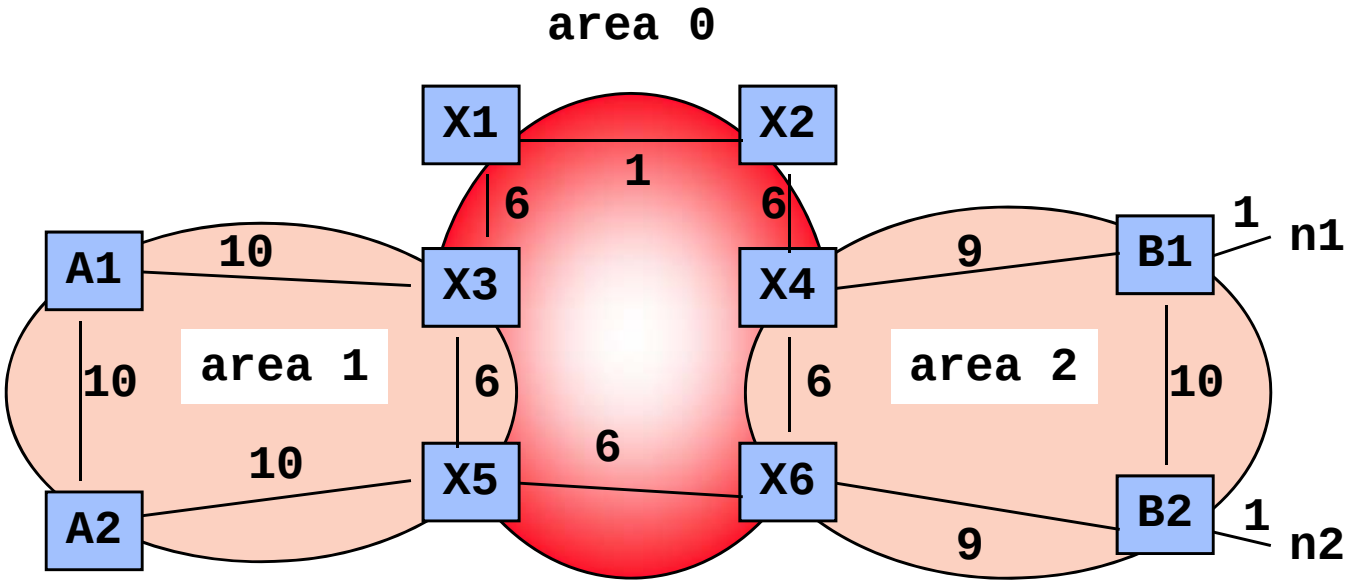
A multi area OSPF domain has one backbone area (area 0) + several non backbone areas.

All inter-area traffic goes through area 0.

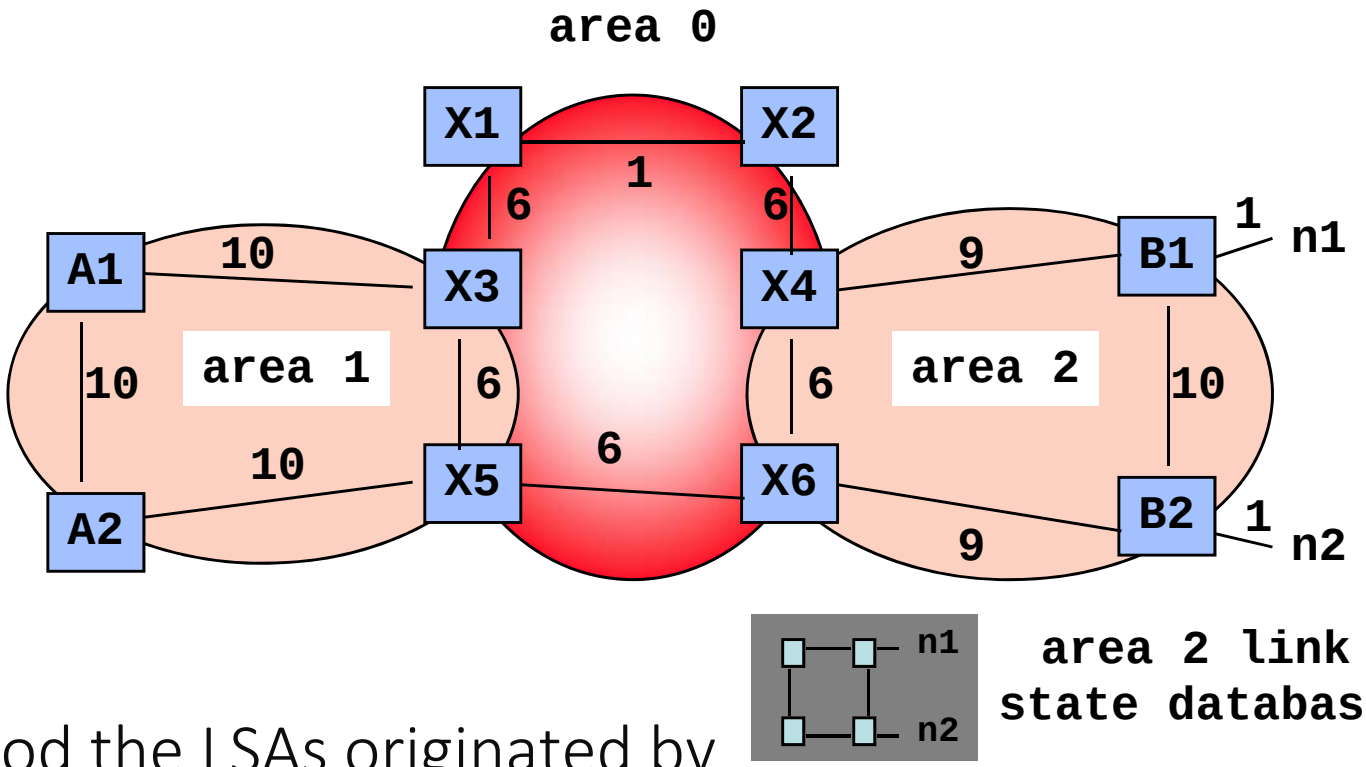


# Principles of OSPF Multi-Area Operation

1. Inside one area, link state is used. One Link State Database per area (replicated in all routers of area)
2. Area Border routers belong to both areas. E.g. X4 belongs to area 1 and to area 0. It has one link state database for area 1 and one for area 0.
3. An area border router injects aggregated distance information learnt from one area into the other area.



# Toy Example Step1







area 2 link state database

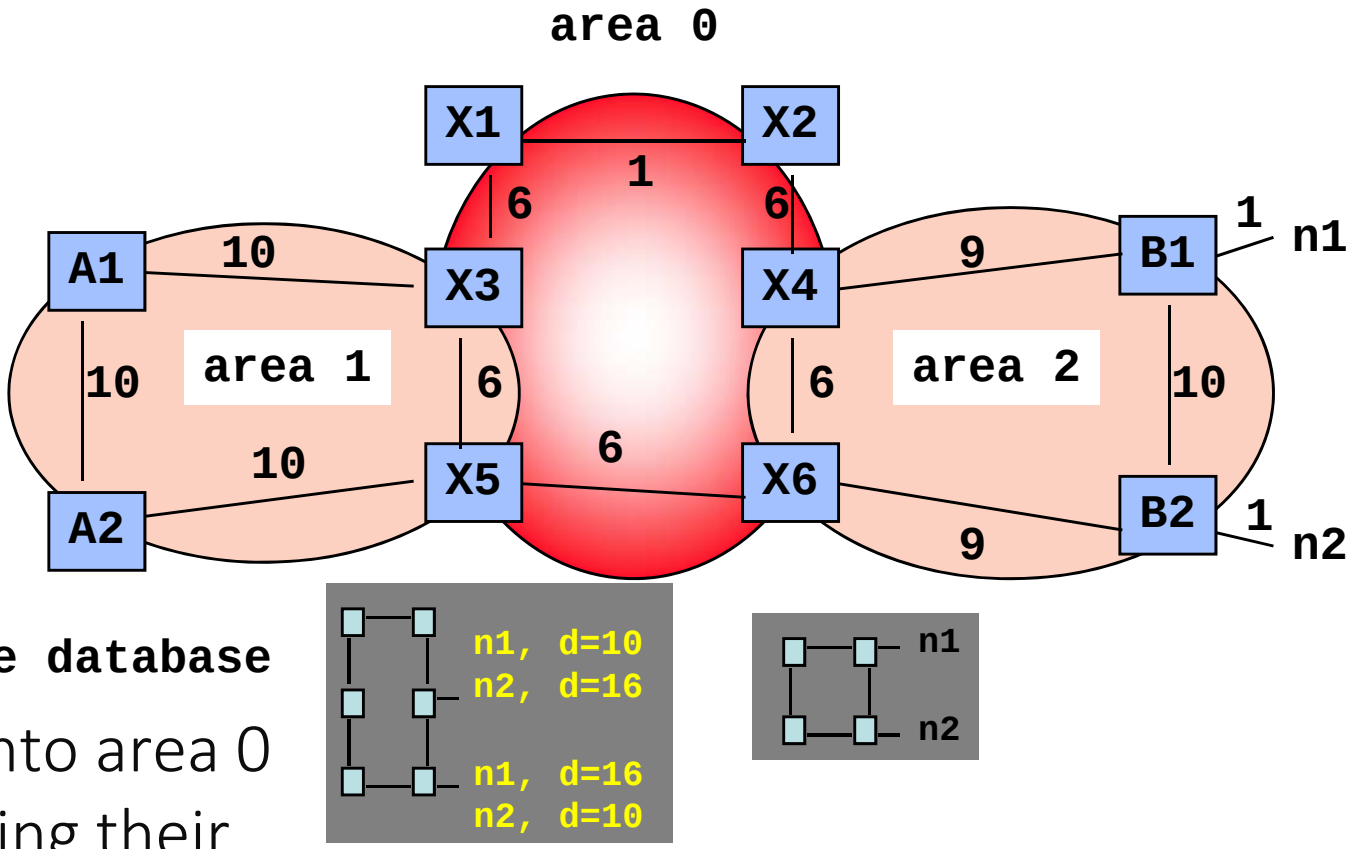
All routers in area 2 flood the LSAs originated by B1 and B2 and know of n1 and n2, directly attached to B1 (resp. B2). This is the normal link state operation. All routers in area 2 have the same link state database, shown above.

All routers in area 2, including X4 and X6 compute their distances to n1 and n2 (using Dijkstra).

X4: distance to n1 =10, to n2 =16

X6: distance to n1 =16, to n2 =10

# Toy Example Step2




**area 0 link state database**

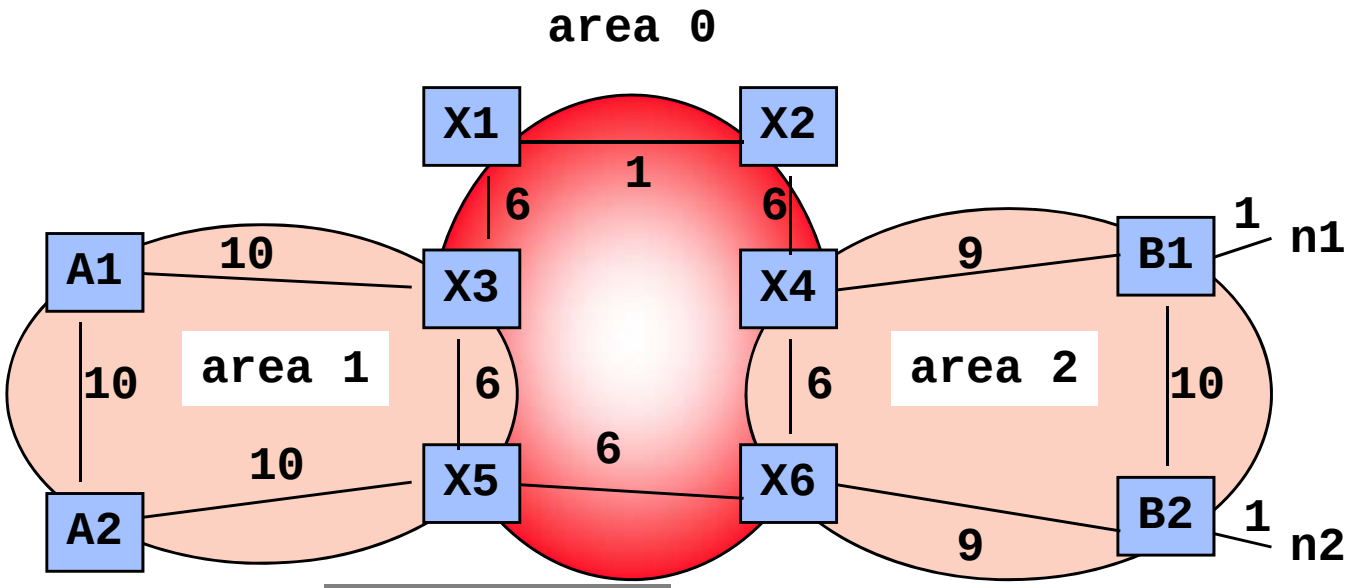
X4 and X6 each flood into area 0 a summary LSA indicating their distances to n1 and n2. All routers in area 0 now have the same link state database, shown above.

All routers in area 0, including X3 and X5 compute their distances to networks outside the area (such as n1) using the Bellman-Ford formula

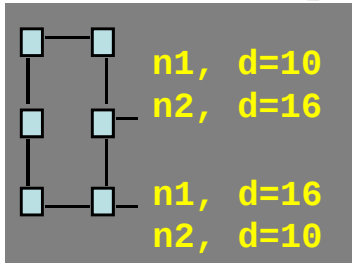
$$d(\text{self}, n1) = \min_{BR \in \text{Area } 0} \{ d(\text{self}, BR) + d(BR, n1) \}$$

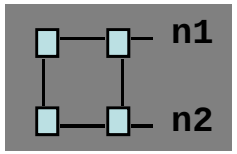
where BR is a border router

# Toy Example Step2 (cont'd)




**area 0 link state database**



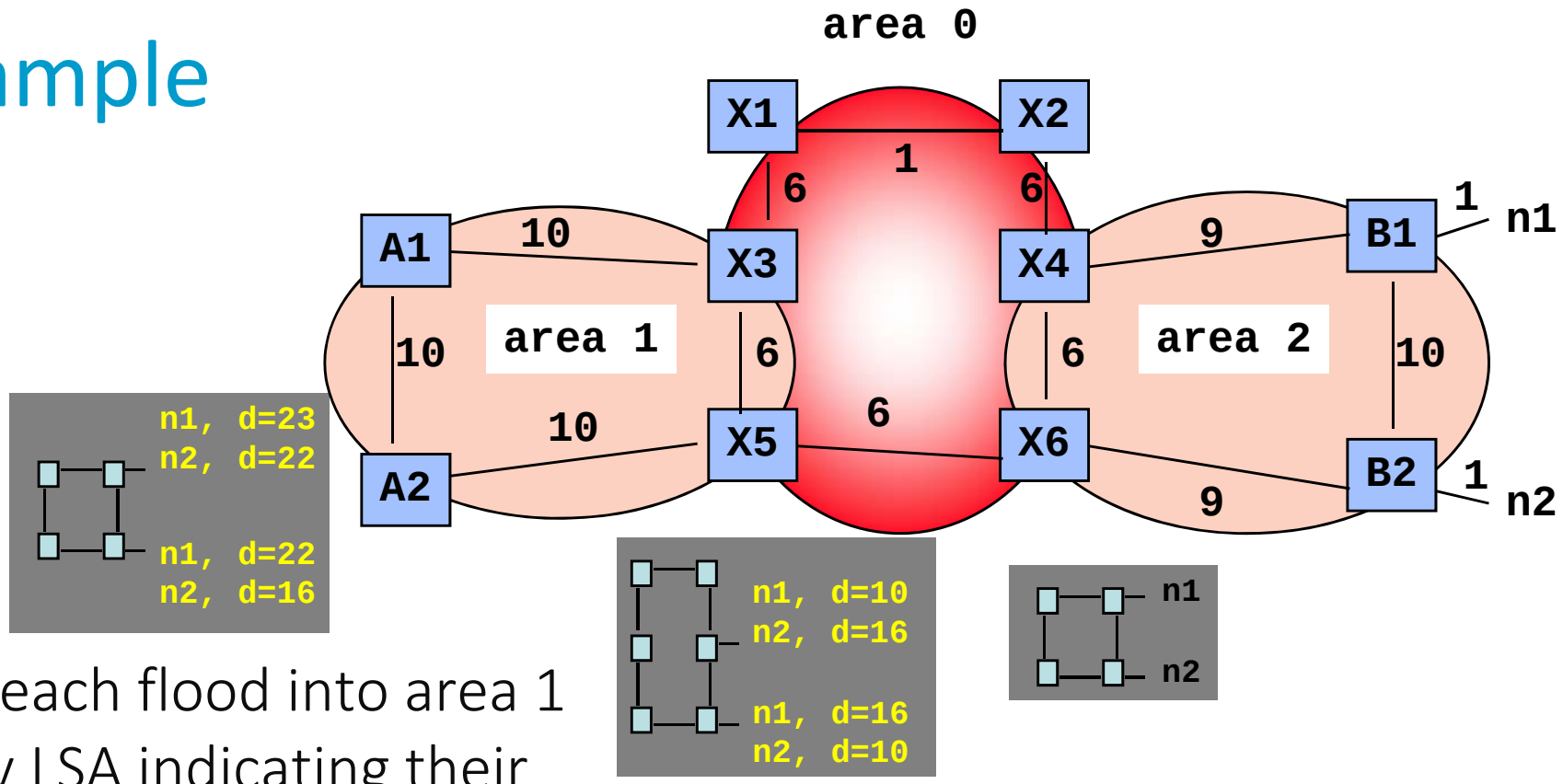




Router X3 computes

$$d(X3,n1) = \min\{d(X3,X4) + d(X4,n1), d(X3,X6) + d(X6,n1)\}$$
$$= \min(\,23, 34) = 23$$
$$d(X3,n2) = \min\{d(X3,X4) + d(X4,n2), d(X3,X6) + d(X6,n2)\}$$
$$= \min(\,29, 22) = 22$$

The process can be used to compute not only the distance, but also the path. To n1, the min is for BR=X4, therefore the shortest path to n1 is via X4 and the next hop to n1 is the next hop to X4. X3 updates its routing table and adds entries to n1 (and n2).

# Toy Example Step 3



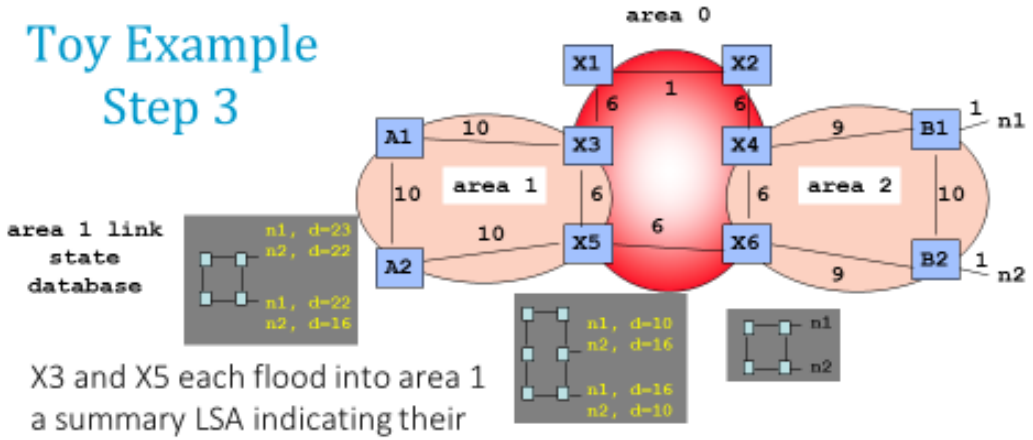

X3 and X5 each flood into area 1 a summary LSA indicating their distances to n1 and n2. All routers in area 1 now have the same link state database, shown above.

All routers in area 1 compute their distances to networks outside the area (such as n1) using the Bellman-Ford formula

$$d(\text{self}, n1) = \min_{BR \in \text{Area 1}} \{ d(\text{self}, BR) + d(BR, n1) \}$$

where BR is a border router. E.g. A1 finds that the distance to n1 is 33 and the shortest path is via X3.

When applying the Bellman-Ford formula to compute $d(\text{self}, n1)$, how does a router such as A1 know the values of $d(\text{self}, BR)$ and $d(BR, n1)$ ?




*A.* $d(\text{self}, BR)$ from its routing table and $d(BR, n1)$ from its link state database

*B.* $d(BR, n1)$ from its routing table and $d(\text{self}, BR)$ from its link state database

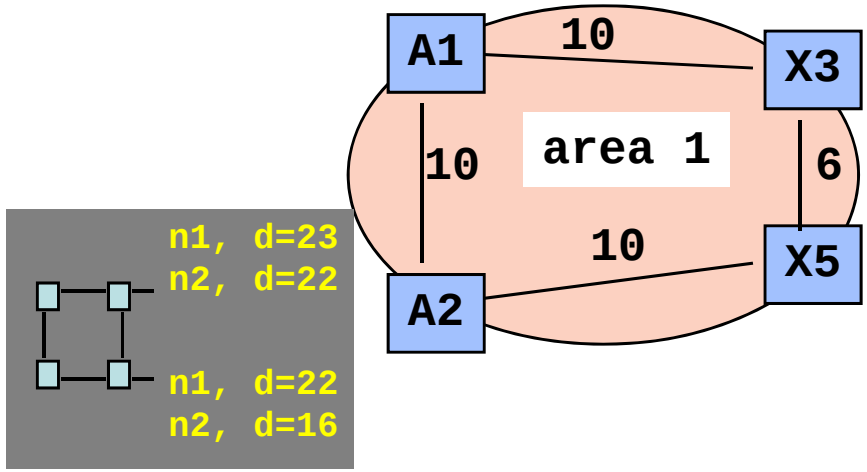
C. both from the routing table

D. both from the link state database

E. I don't know

# Solution

All routers in area 1 have only their routing tables + this information →

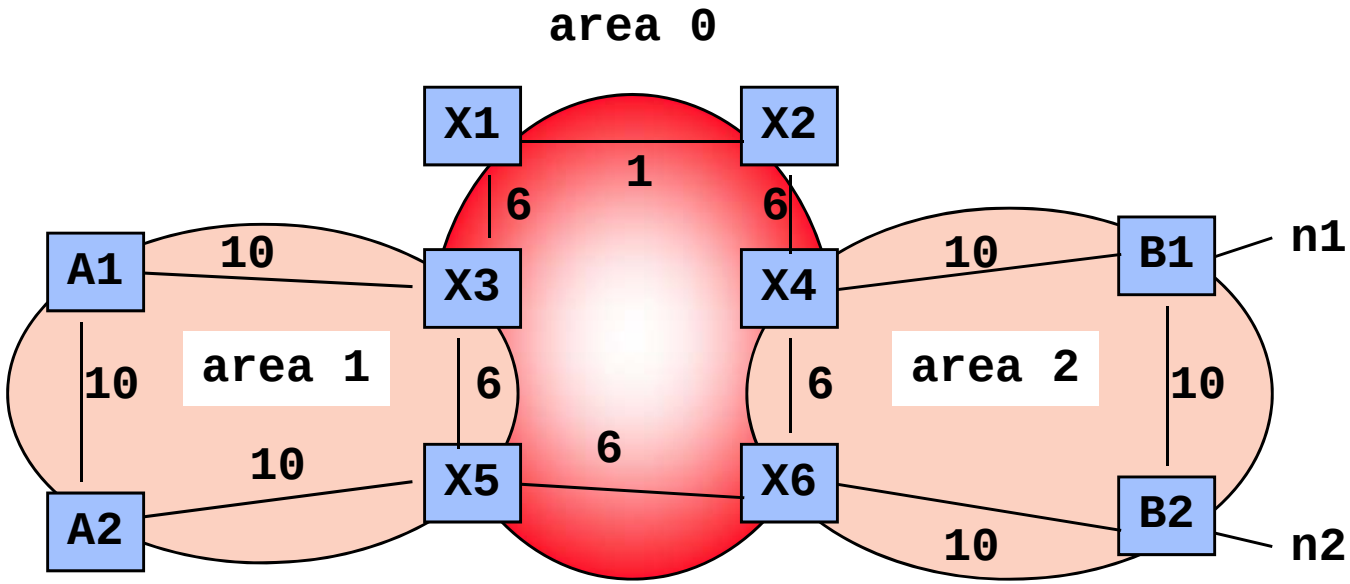



The border routers are in area 1, so A1 knows d(self, BR) from its routing table (after applying Dijkstra).

d(BR, n1) is known from an external-LSA, i.e. is in the link-state database of A1 (and of all routers in area 1).

Answer A.

# How many link state databases does router X3 have ?




A. 1

B. 2

C. 3

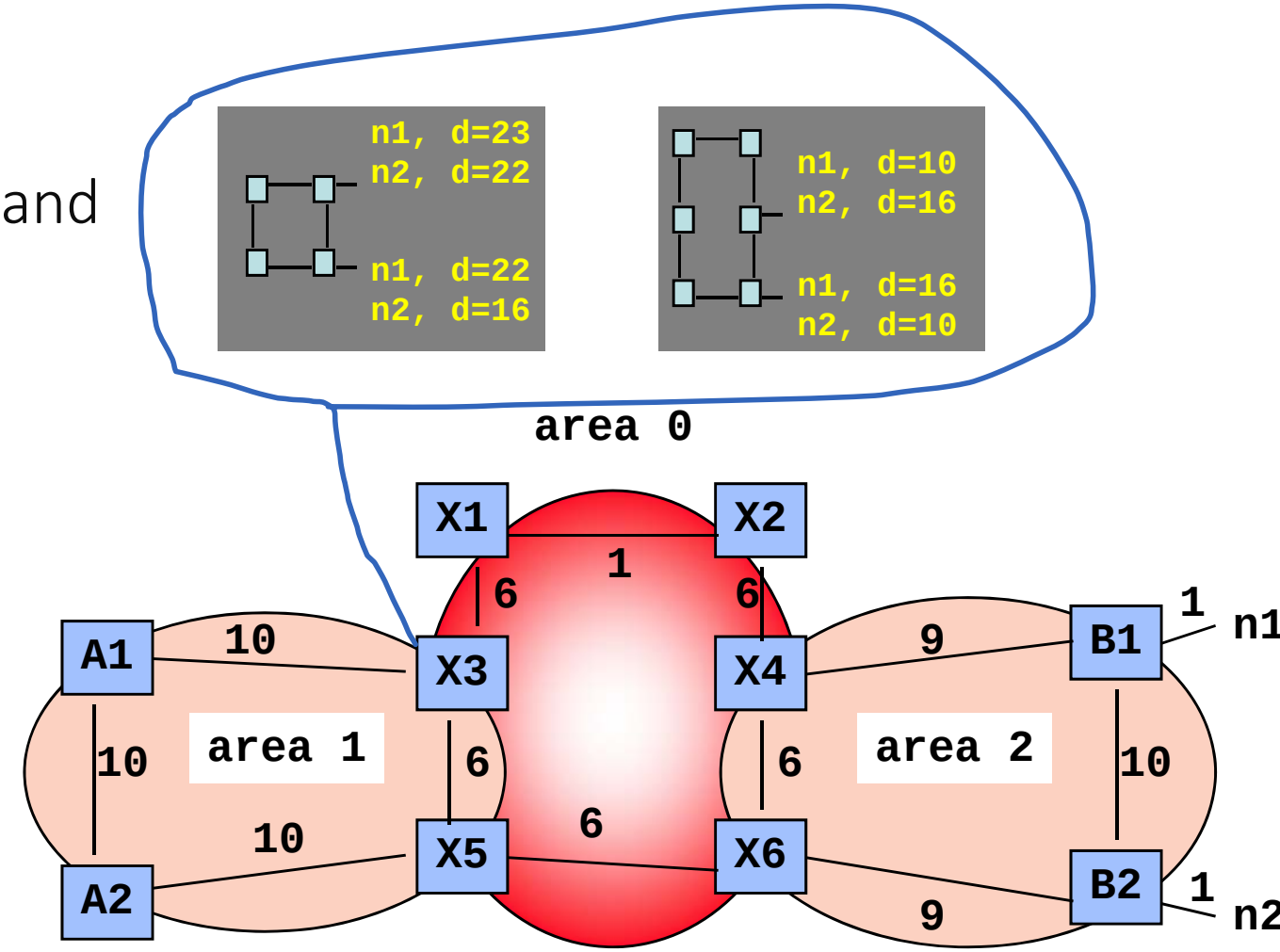
D. 0

E. I don’t know

# Solution

Answer B.

X3 belongs to area 0 and to area 1. It has one link-state database for each.



# Comments

Summary LSAs for all reachable networks are present in all link state databases of all areas

- can be avoided by configuring some areas as “stub areas”: they use a default route to the backbone

Area partitions require specific support

- partition of non-backbone area is handled by having the area 0 link state database keep a map of all area connected components
- partition of backbone cannot be repaired; it must be avoided; can be handled by virtual links through non backbone area

# 8. Other Uses of Link State Routing

Link state routing (OSPF or IS-IS) provides a complete view of area to every node. This can be used to provide advanced functions:

multi-class routing: compute different routes for different types of services (e.g. voice, video)

explicit routes (with source routing): an edge router computes the entire path, not just the next-hop, and writes it in the packet header. Avoids transient loops / supports fast re-route after failure.

# Example: LS bridging

Assume you want to bridge VLANs across a campus. One solution: tunnel MAC packets in IP. **Problem**: automatic creation of tunnels.

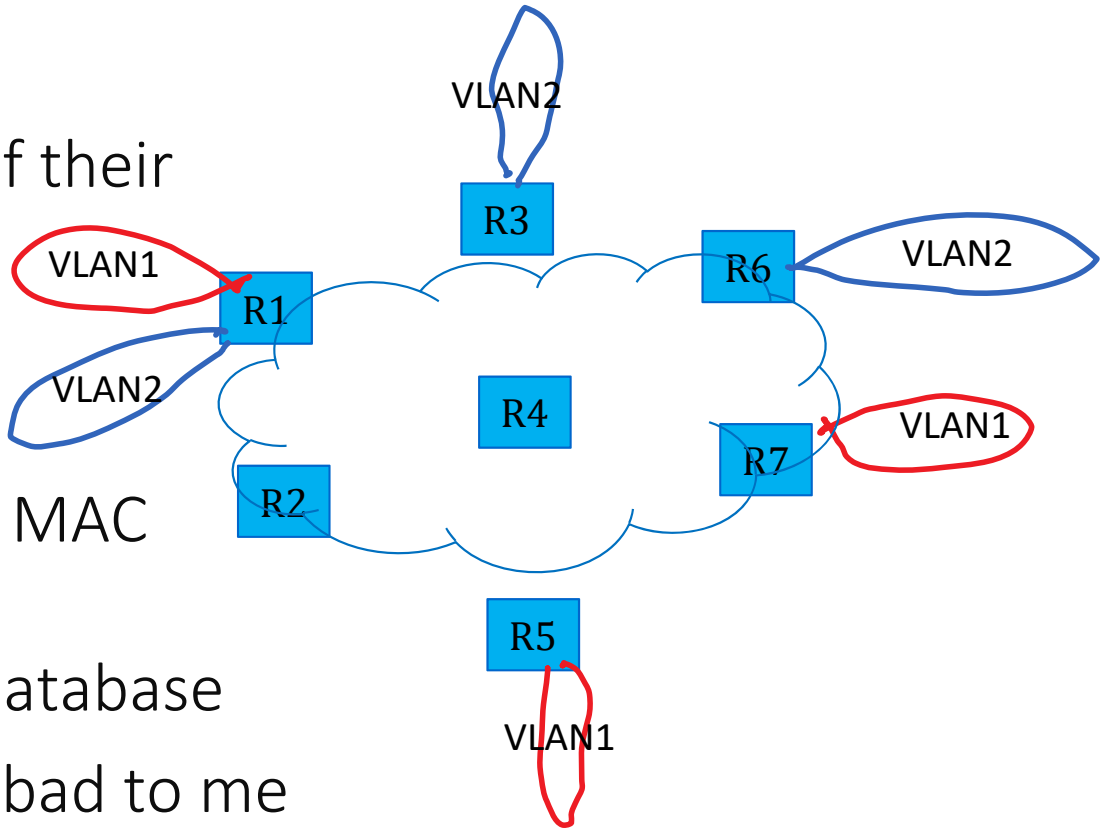
## Can you imagine a solution using Link State Routing in R1, R2, … ?

A. Routers R1, R2 … discover which VLAN is active on any of their ports and put this information in the link state database

B. Routers R1, R2 … overhear all MAC source addresses and put the information in the link state database

C. Both of these solutions seem bad to me

D. I don't know



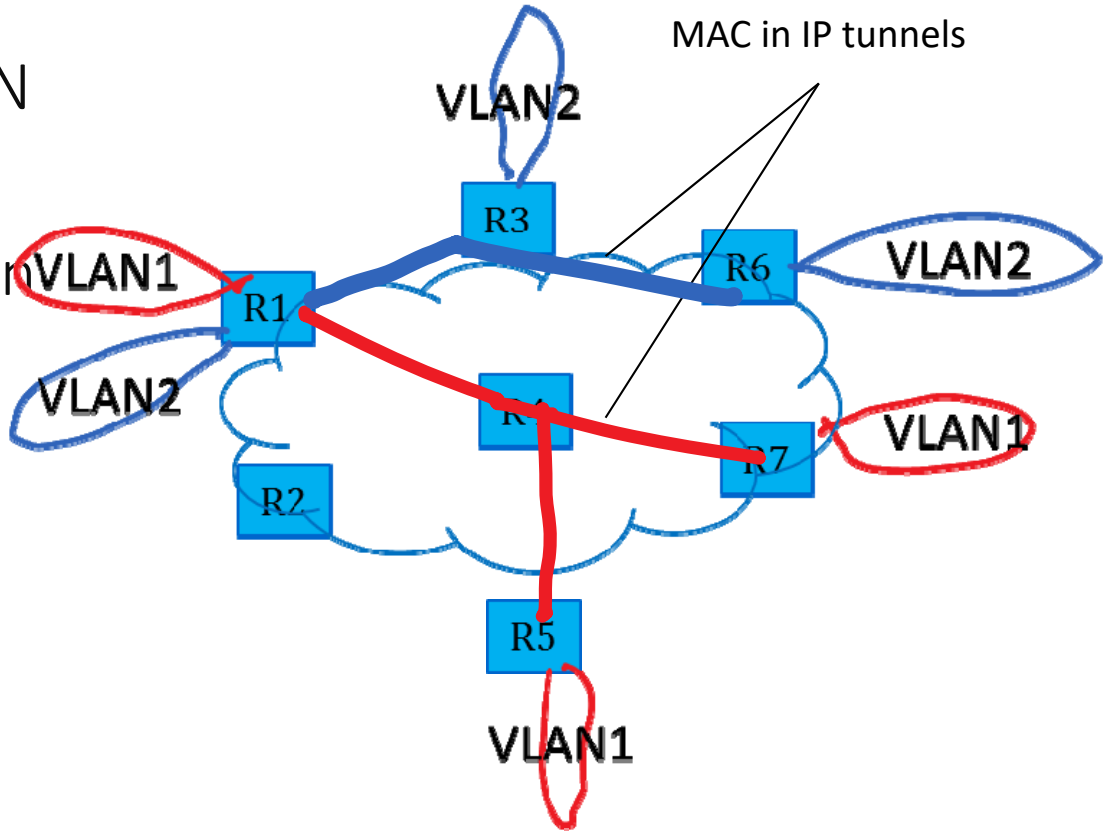
# Solution

B does not help since MAC addresses don't say in which VLAN the machine is

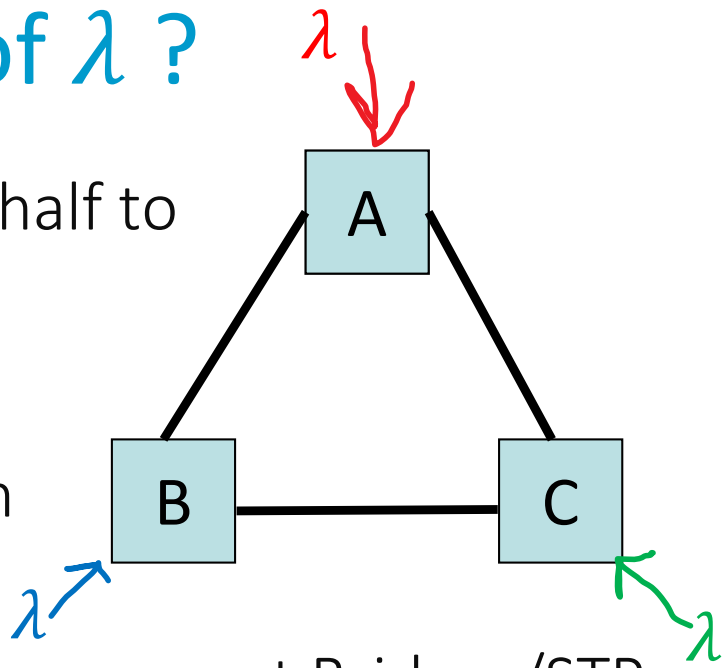
A is a feasible solution: routers can create VLAN tunnels (MAC in IP !) e.g. using IP multicast

This is what Cisco's TRILL does (with IS-IS instead of OSPF)

IEEE's SPB is similar (with MAC in MAC encapsulation); supports explicit routes with 802.1av for video networking in studios.



# What is the maximum value of $\lambda$ ?




Node A sends a total traffic equal to $\lambda$ b/s, half to B, half to C.

Same for B and C.

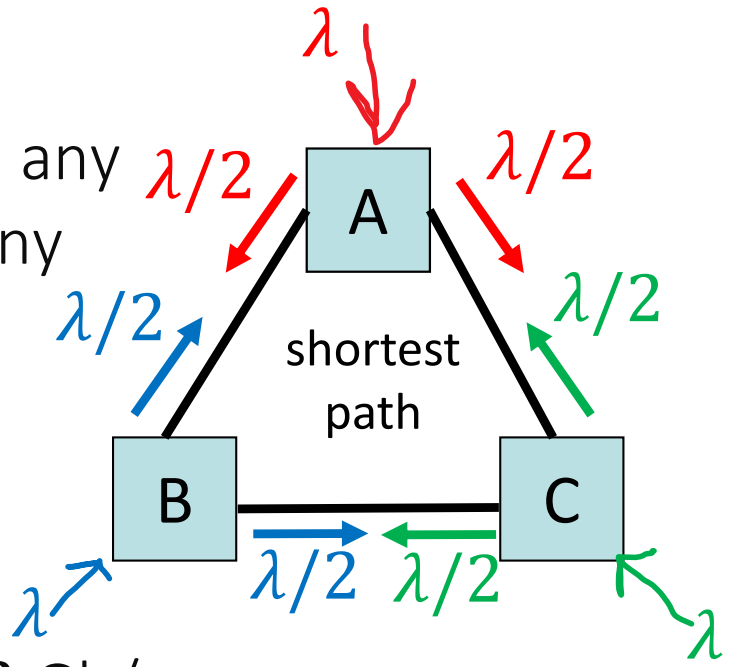
Link capacities are 1 Gb/s in every direction (full duplex). OSPF costs are 10.

Possible configurations: OSPF Routers or Transparent Bridges/STP

A. With OSPF: $\lambda_{\max} = 2$ Gb/s; with TB $\lambda_{\max} = 2$ Gb/s;
B. With OSPF: $\lambda_{\max} = 2$ Gb/s; with TB $\lambda_{\max} = 1$ Gb/s;
C. With OSPF: $\lambda_{\max} = 1$ Gb/s; with TB $\lambda_{\max} = 2$ Gb/s;
D. With OSPF: $\lambda_{\max} = 1$ Gb/s; with TB $\lambda_{\max} = 1$ Gb/s;
E. I don't know.

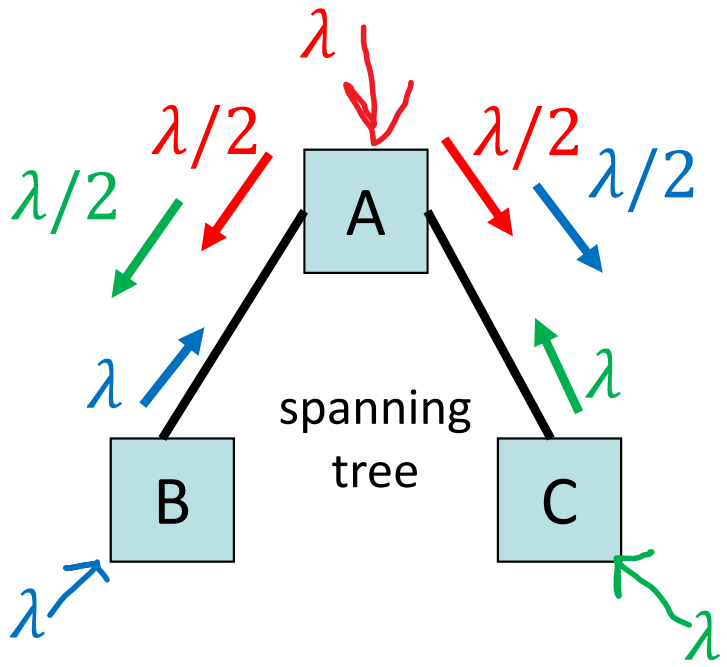
# Solution

OSPF routers use shortest paths. Traffic from any node uses the direct link (same is true with any distance vector, which that also computes shortest paths, and with TRILL bridges).
From A to B, the traffic flow is $\lambda/2$, same on every link and each direction.




The constraints are $\frac{\lambda}{2} \leq 1$Gb/s thus $\lambda_{\max} = 2$ Gb/s,

With transparent bridges, a spanning tree is built, one of the links is disabled. Total traffic on every link and every direction is $\lambda$, thus $\lambda_{\max} = 1$ Gb/s.




Answer B.

# 9. Software Defined Networking

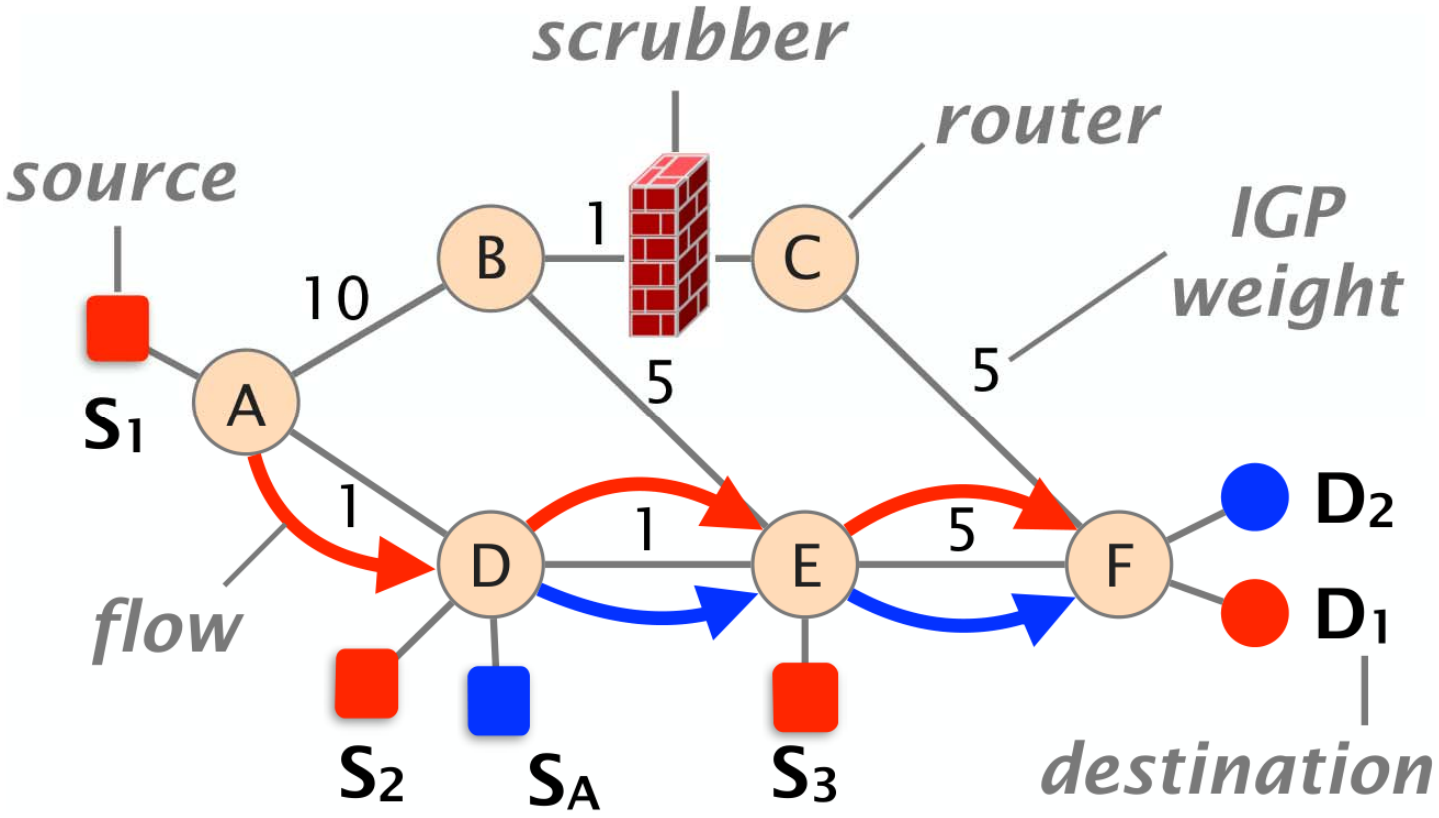
In principle, an IP router uses the destination address and longest prefix match to decide where to send a packet.

Some networks want more control; e.g. handle mission critical traffic with high priority; ban non-HTTP traffic; send suspicious traffic to a machine that does deep packet inspection.

From [S. Vissicchio et al., "Central Control Over Distributed Routing", ACM Sigcomm 2015]

http://fibbing.net

A sudden traffic surge is noticed from A, D and E to F (red). The network operator would like to divert all red traffic to scrubber for inspection. Blue traffic should not be modified.



# Per-Flow Forwarding

This is why some routers can be configured with *per-flow forwarding* rules.

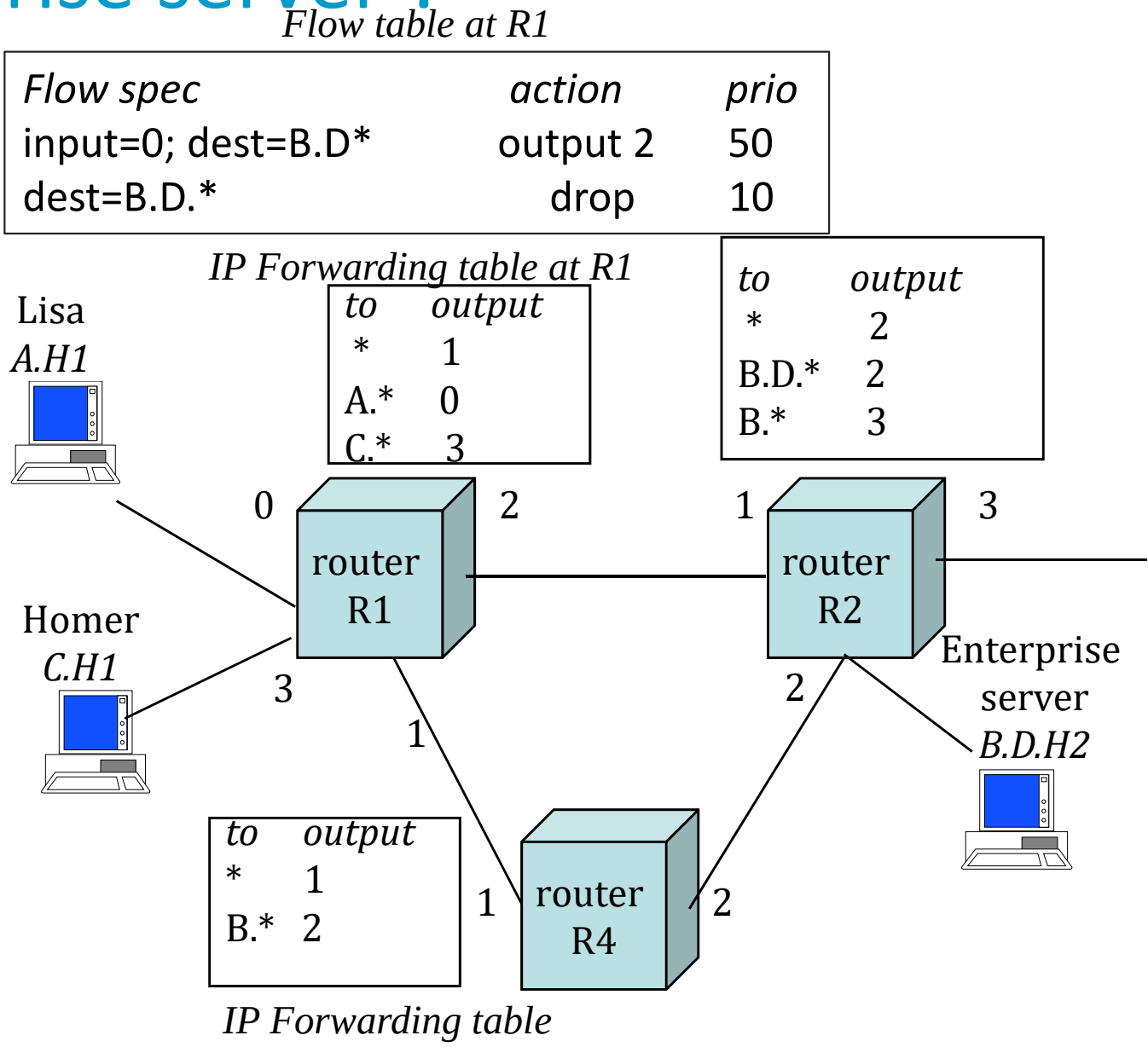
When a packet has to be forwarded, such a router does:

- Look for a rule match in the list of (priority-ordered) per-flow forwarding rules (also called *flow table*)
  if one or several matches exist, follow the rule with highest priority
- If no rule matches, go to the IP forwarding table and do longest prefix match

Same can be used in switches (per flow tables then complement the MAC forwarding table)

# Which way at R1 for packets from Lisa and Homer to Enterprise server ?

A. Lisa:1, Homer: 1
B. Lisa:1, Homer: 2
C. Lisa:2, Homer: 1
D. Lisa:2, Homer: 2
E. None of above
F. I don't know



# Solution

Answer E

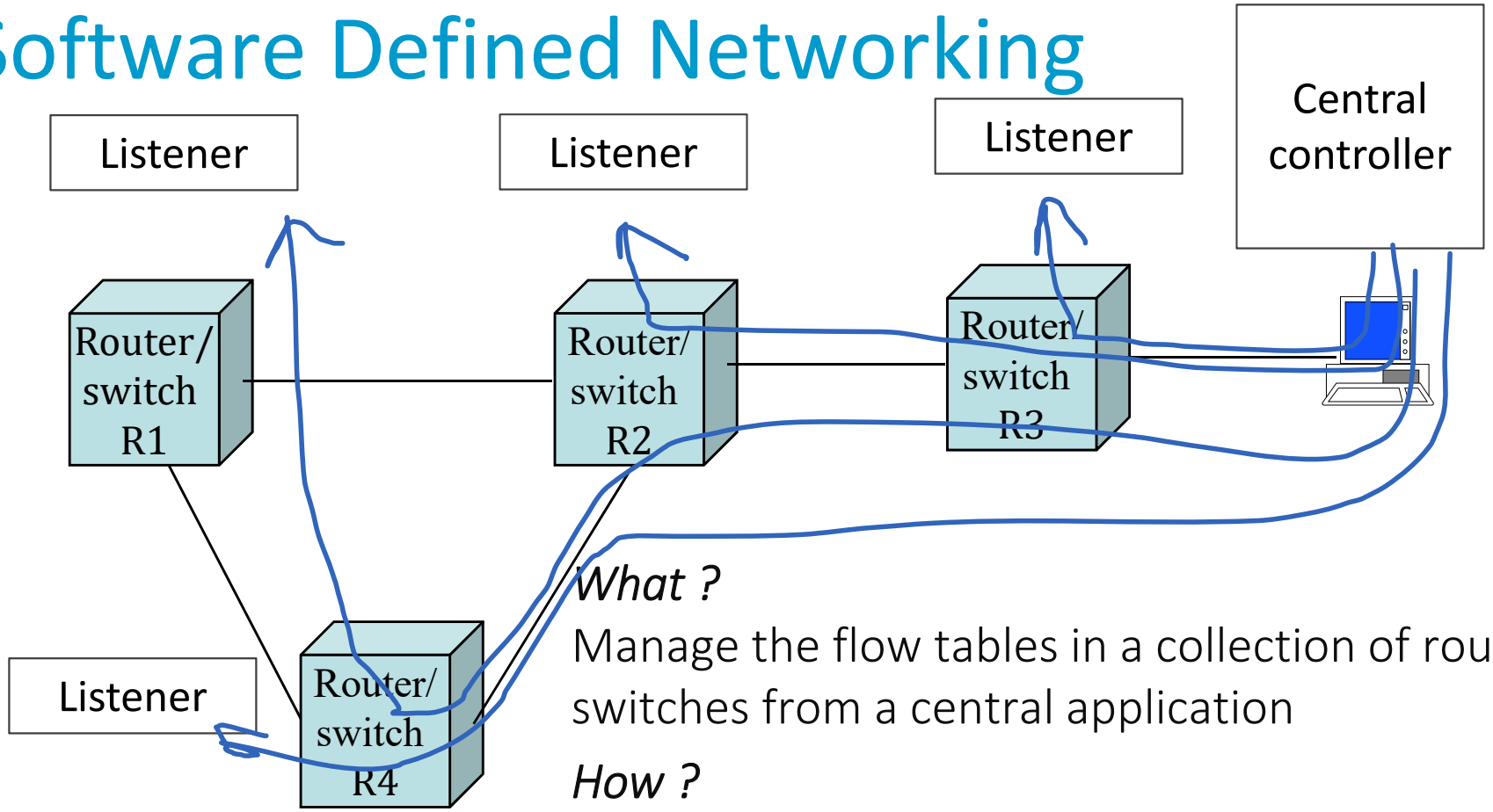
Packets from Lisa to enterprise server match the two flow rules; the first one has higher priority and is applied. Packets are forwarded to port 2. Since there is a match in flow table, the IP forwarding table is not used.

Packets from Homer to enterprise server match the second flow rule and are dropped by R1.

The combined effect of the flow table and the IP forwarding table at R1 is such that

1. all traffic to B.D.* is killed except if arriving on input 0
2. traffic to B.D.* that is not killed is forwarded to output 2
3. traffic to B.* and not B.D.* is forwarded to output 1

# Software Defined Networking




*What ?*

Manage the flow tables in a collection of routers or switches from a central application

*How ?*

A central controller decides rules and communicates them to local controllers on routers

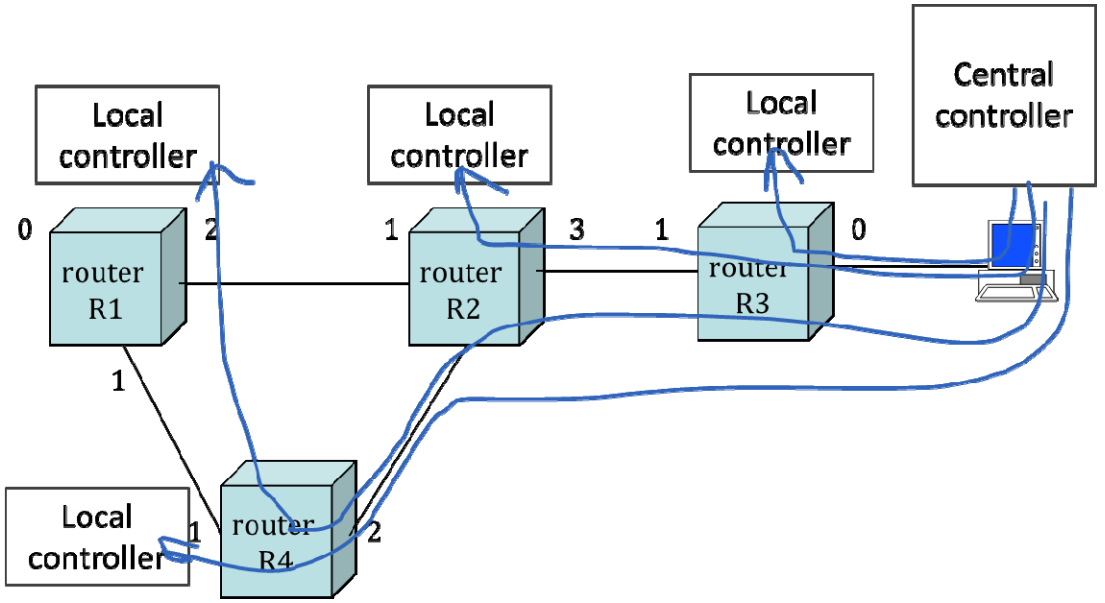
Local controllers, called “listeners” write the per-flow tables on routers or switches

Protocol between local controller and central controller is e.g. OpenFlow, over TCP connections

*Where ?* Mainly in large data centers, also for 5G cellular.

# Do we need OSPF (or another routing protocol) if we have SDN ?

A. No because flow tables can replace IP forwarding tables
B. Yes because flow tables cannot replace IP forwarding tables
C. Yes because the central controller needs a way to communicate with local controllers
D. I don't know



# Solution

Answer C

The central controller communicates with the local controllers in routers over TCP connections. This needs that IP forwarding tables are functional, which in turn requires OSPF or some other routing protocol.

# Conclusion

OSPF (and routing protocols) automatically build connectivity and repair failures.

With link state routing:

- All routers compute their own link state database, replicated in all routers
- All routers compute their routing tables using Dijkstra and the link state database
- Convergence after failure is fast (if detection is fast)
- Supports flexible cost definitions; can be used for routing specific flows in different ways
- Large domains must be split into areas

More control can be obtained by an outside application (SDN). SDN is used today primarily with switches, but also starts to be used with routers.