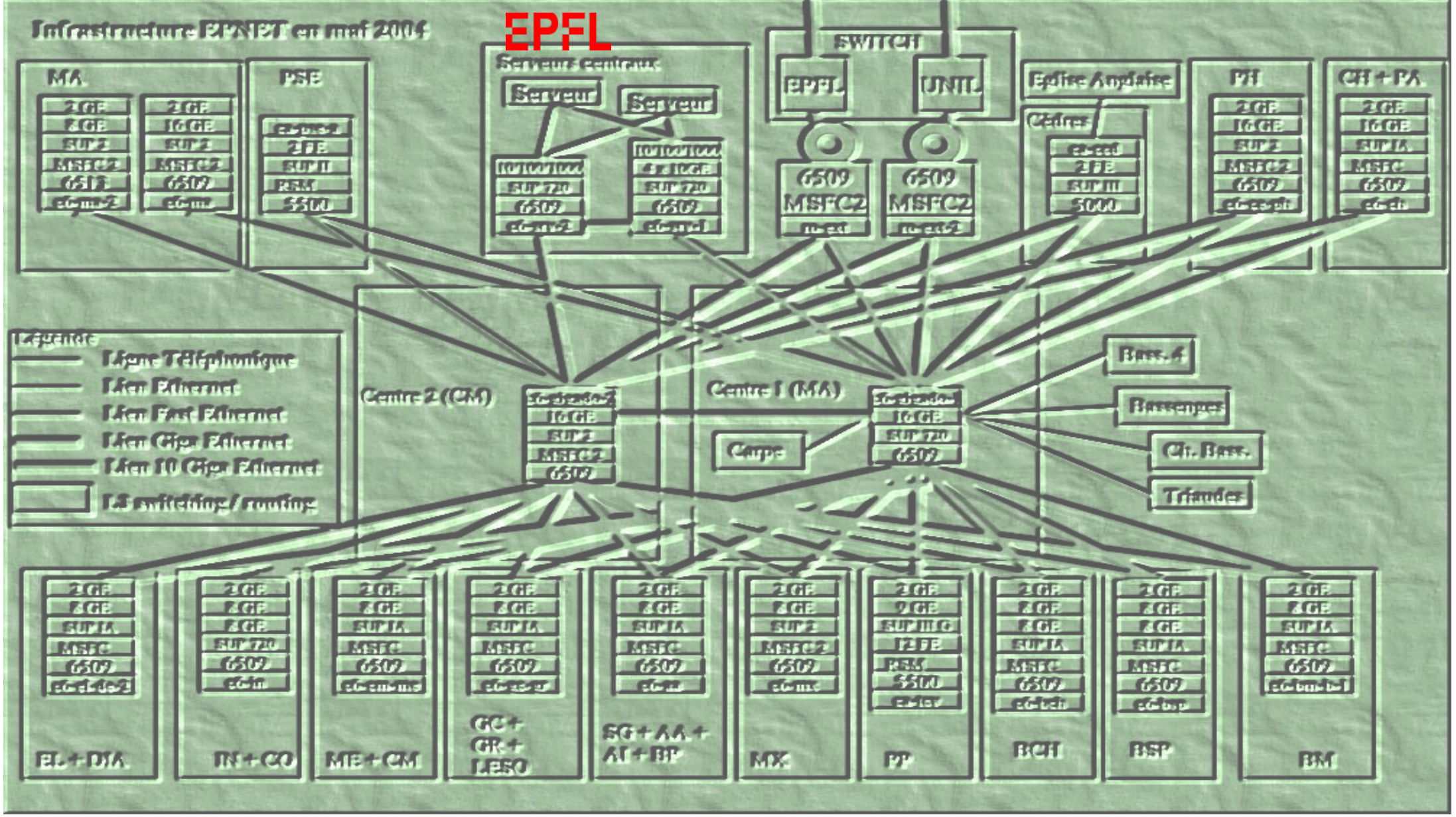


Routing algorithms

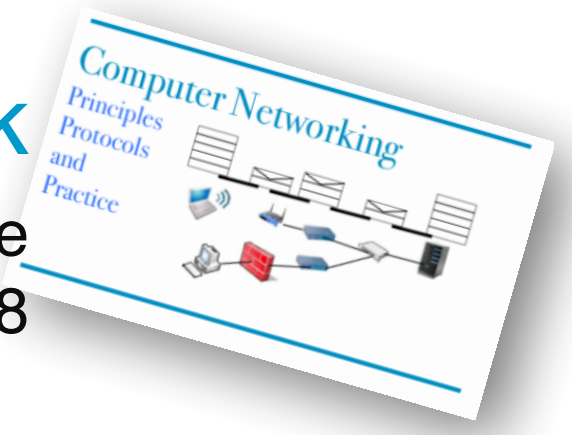
Contents

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)

Textbook

Section 5.1.1, The control plane

RFC 2328



Reminder: the network layer offers

Forwarding (in the Data plane)

- when a packet arrives at a router's input link, the router moves the packet to the appropriate outgoing link
- based on:
  - info in the packet header (e.g. dest IP address, BIER extension header, etc.)
  - the router's *forwarding table* (a.k.a. *routing table*)

Routing (in the Control plane)

- determines the entire *route/path* followed by each packet towards its destination
  hence the entries of the forwarding tables and/or the information in extension header
- typically done by a *routing algorithm*

Routing protocols (or routing algorithms)

Why we need them?

- Manually configuring the forwarding tables is *not* easy
  - in a single domain, it is doable but time-consuming and error-prone
  - in the WAN, it is impossible—about 120000 domains should coordinate

- A routing protocol or algorithm
  - allows routers to *automatically* compute the *best path(s)* to each destination

Let's think: How could we design such a protocol/algorithm?

Many routing algorithms, but where do they differ?

Nature of "best" path — i.e. what is optimization objective of an algorithm?

- to use least-cost path
- to use equal-cost multi-path
- to respect policies
- arbitrary (min delay, low bandwidth usage…)

Scope of network — i.e. what is the underlying network? is topology info available?

- single domain —> *intra-domain* routing (main alg. is OSPF)
- multiple domains —> *inter-domain* routing (main alg. is BGP)

A *domain* is a network under the *same* administrative entity (e.g. a campus network, an enterprise network, or an ISP, etc.)

State location — i.e. where is the output (the routing information) stored?

- inside a local forwarding table
- directly into the packet headers

Taxonomy of routing protocols

Link State

- Each router maintains a local *map* of the entire topology
  - obtained by *gossiping* with other routers (flooding)
  - every link on map has a cost; e.g. cost(1 Gb/s link)=1; Cost(100 Mb/s link)=10
- computes *shortest (min-cost) paths* to each destination prefix based on map
- determines next hop to each prefix and populates its forwarding table

- Used for *intra-domain routing* (e.g. OSPF, IS-IS) and advanced bridging methods (e.g. TRILL, SPB Shortest Path Bridging)
- Many variants:
  - “shortest” may mean “min latency”, or “max available bit rate”, etc.

Taxonomy of routing protocols

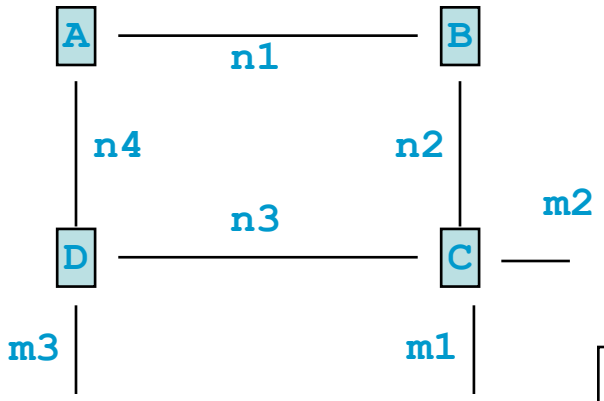
Distance Vector

- *No* global map
- Each router initially knows only about:
  - locally-attached networks and neighbor routers,
  - and the *costs* of direct links to them
- then: it *informs* neighbors about the estimated distances to all destinations it knows (= sends its *distance vector*);

  learns new destinations from vectors received by neighbors and *updates* its distance vector (using the Bellman-Ford algorithm)
- finally: it determines next hops and populates its forwarding table

Distance Vector example — RIP

All link costs = 1



A

net	dist	nxt
n1	1	n1,A
n4	1	n4,A

B

net	dist	nxt
n1	1	n1,B
n2	1	n2,B

D

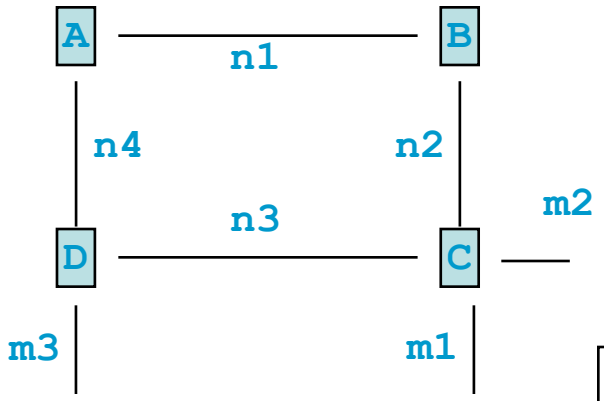
net	dist	nxt
n3	1	n3,D
n4	1	n4,D
m3	1	m3,D

C

net	dist	nxt
n2	1	n2,C
n3	1	n3,C
m1	1	m1,C
m2	1	m2,C

Distance Vector example — RIP

All link costs = 1



A

net	dist	nxt
n1	1	n1,A
n4	1	n4,A

from A

n1	1
n4	1

B

net	dist	nxt
n1	1	n1,B
n2	1	n2,B
n4	2	n1,A

D

net	dist	nxt
n3	1	n3,D
n4	1	n4,D
m3	1	m3,D

from D

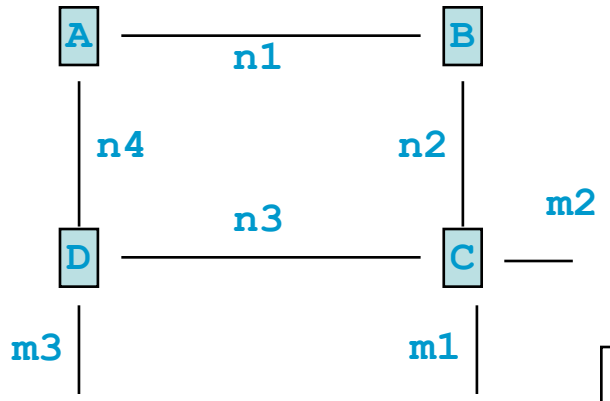
n3	1
n4	1
m3	1

C

net	dist	nxt
n2	1	n2,C
n3	1	n3,C
m1	1	m1,C
m2	1	m2,C
n4	2	n3,D
m3	2	n3,D

Distance Vector example — RIP

All link costs = 1



A

net	dist	nxt
n1	1	n1,A
n4	1	n4,A

B

net	dist	nxt
n1	1	n1,B
n2	1	n2,B
n3	2	n2,C
n4	2	n1,A
m1	2	n2,C
m2	2	n2,C
m3	3	n2,C

from C

n2	1
n3	1
m1	1
m2	1
n4	2
m3	2

D

net	dist	nxt
n3	1	n3,D
n4	1	n4,D
m3	1	m3,D

C

net	dist	nxt
n2	1	n2,C
n3	1	n3,C
m1	1	m1,C
m2	1	m2,C
n4	2	n3,D
m3	2	n3,D

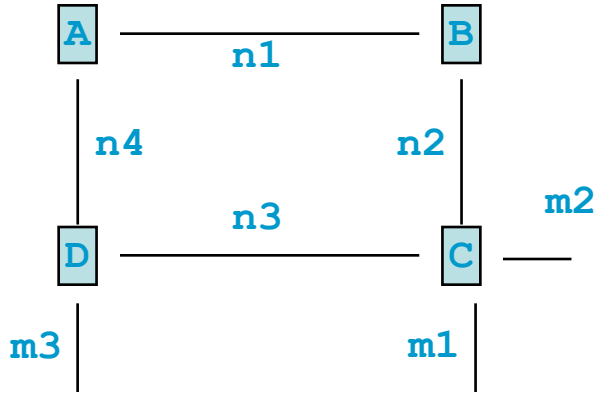
Distance Vector example — RIP: convergence to optimal paths!

A

net	dist	nxt
n1	1	n1,A
n2	2	n1,B
n3	2	n4,D
n4	1	n4,A
m1	3	n4,D
m2	3	n4,D
m3	2	n4,D

B

net	dist	nxt
n1	1	n1,B
n2	1	n2,B
n3	2	n2,C
n4	2	n1,A
m1	2	n2,C
m2	2	n2,C
m3	3	n2,C



D

net	dist	nxt
n1	2	n4,A
n2	2	n3,C
n3	1	n3,D
n4	2	n4,D
m1	2	n3,C
m2	2	n3,C
m3	1	m3,D

C

net	dist	nxt
n1	2	n2,B
n2	1	n2,C
n3	1	n3,C
m1	1	m1,C
m2	1	m2,C
n4	2	n3,D
m3	2	n3,D

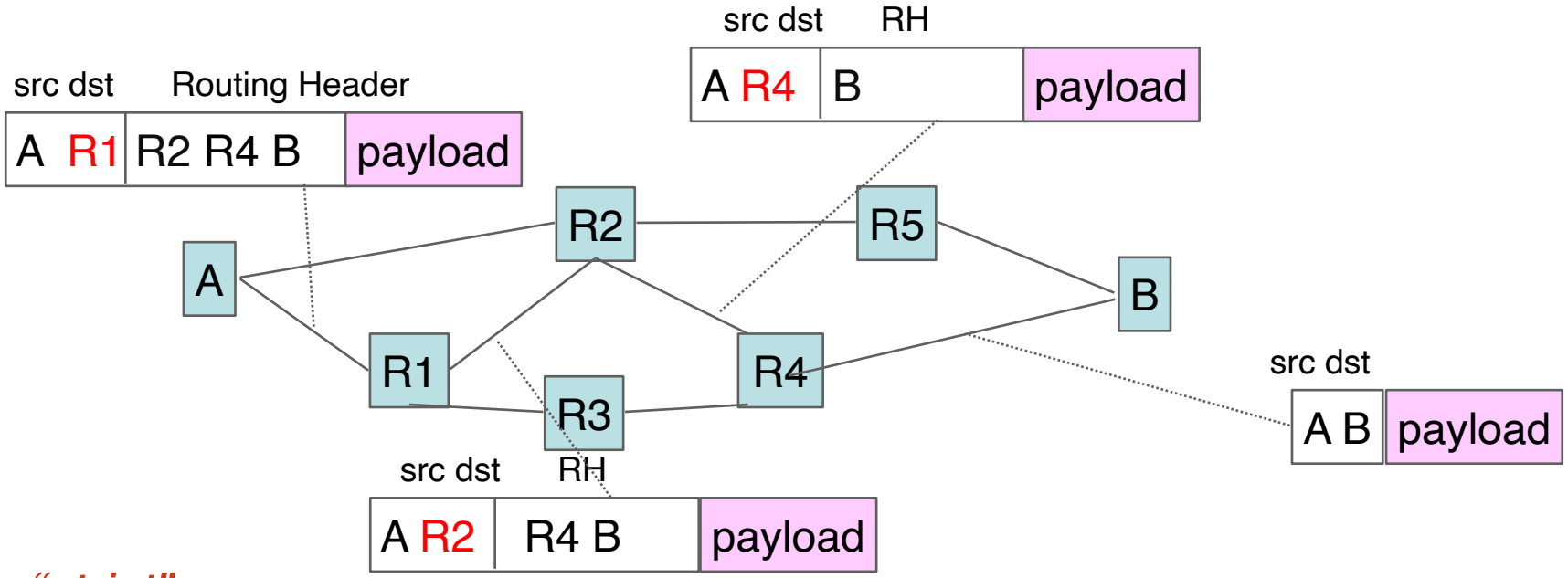
- This message-passing method *converges* to shortest paths!
- But, it may need *time to converge*
- hence it is used in *small* networks

Taxonomy of routing protocols

Path Vector

- Every router knows only:
  - its neighbors +
  - explicit *paths* to all destinations
- computes the “best” path to each destination and populates forwarding table
- Optimization criterion is *not cost*
- Suitable for *inter-domain routing*, because it is hard to assign costs and domains may do not want to disclose their *private* internal topology
  - e.g. BGP computes domain-level paths
  - and each router selects best path according to *multiple criteria*

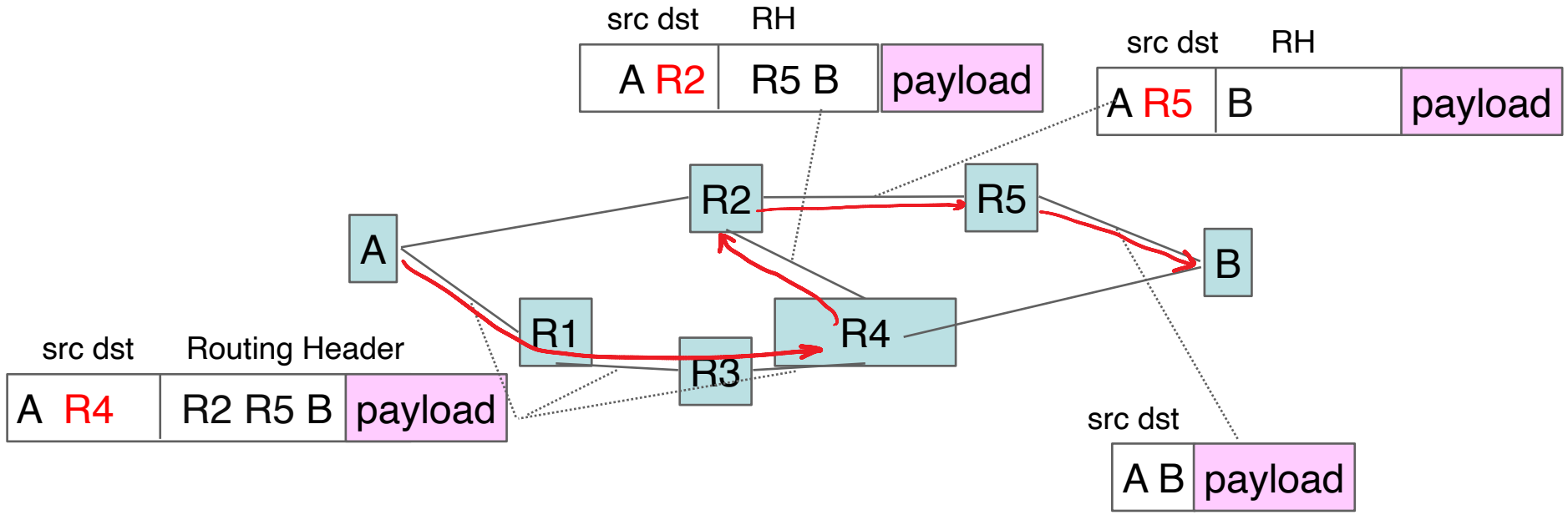
Taxonomy of routing protocols



Source Routing—"strict"

- Source puts explicit paths *into packet headers* (path = sequence of intermediate hops)
- In IPv6, routing header is an extension header—contains intermediate hops and ultimate destination; destination IP address is next intermediate hop
- Intermediate routers are "dumb": just remove themselves from header and forward pkts to next hop
- Used in *ad-hoc networks*, where route computation is done by a control application or by discovery: e.g.: source discovers path by flooding *explorer packets* that accumulate the path followed

Taxonomy of routing protocols



Source Routing—"loose"

- Forces *some* intermediate hops
- Assumes an *underlying* routing algorithm such as link-state routing, e.g., we know how to go from A to R4
- Used for fine grained control (traffic engineering, separation of customers)

Segment routing

- Generalizes loose source routing by allowing the routing header to contain processing instructions for the intermediate hop; e.g., we can ask R4 to apply a security *function* (screening, traffic separation)
- Used in data centers

2. OSPF (for a single area)

Most important link state algorithm

Every router has:

- an *interface database* (= its network interfaces, learnt by configuration)
- an *adjacency database* (= neighbors' states, learnt by a hello protocol)
- a *link state database* (= topology map, learnt by flooding)

Hello is a ***ping-style*** protocol between neighbors:

- discovers neighbor routers
- detects failures (e.g. if neighbor doesn't respond after 4 times, it is "dead")

Link State Database and LSAs

When two routers become neighbors, they first *synchronize* their link state databases:

Case 1: one router is new, hence *copies* what the other already knows

Case 2: existing routers connect, hence they *merge* their databases

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

- Every router sends a LSA with its attached networks and neighbor routers
- LSAs are *flooded* in the entire area and stored in each router's link state database
- LSAs contains a *sequence* number and *age*
  - Sequence number prevents loops;
    only messages with new sequence numbers are accepted and re-flooded
  - Age field is used to periodically resend LSA (e.g. every 30mins or 1h)
    and to flush invalid LSAs

Toy example showing *interface databases*

At B

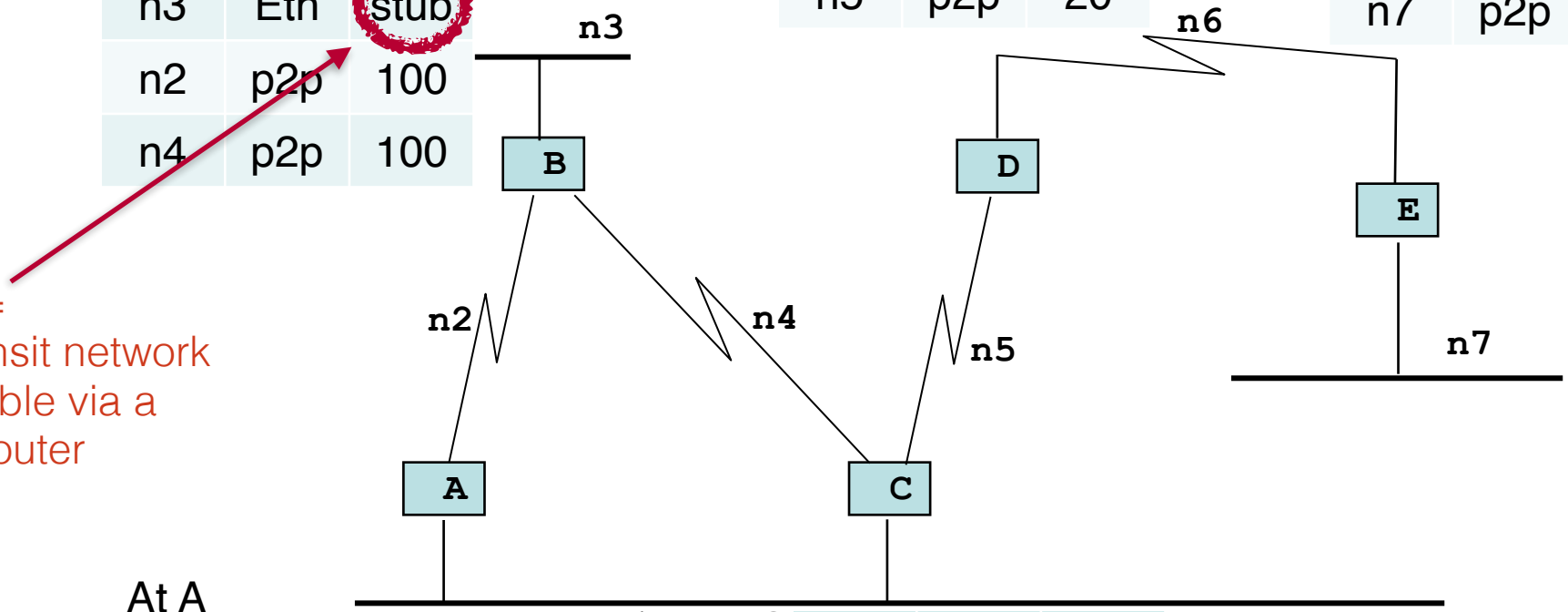
Net	Type	cost
n3	Eth	stub
n2	p2p	100
n4	p2p	100

At D

Net	Type	cost
n6	p2p	10
n5	p2p	20

At E

Net	Type	cost
n6	p2p	10
n7	p2p	stub



"stub" = non-transit network accessible via a single router

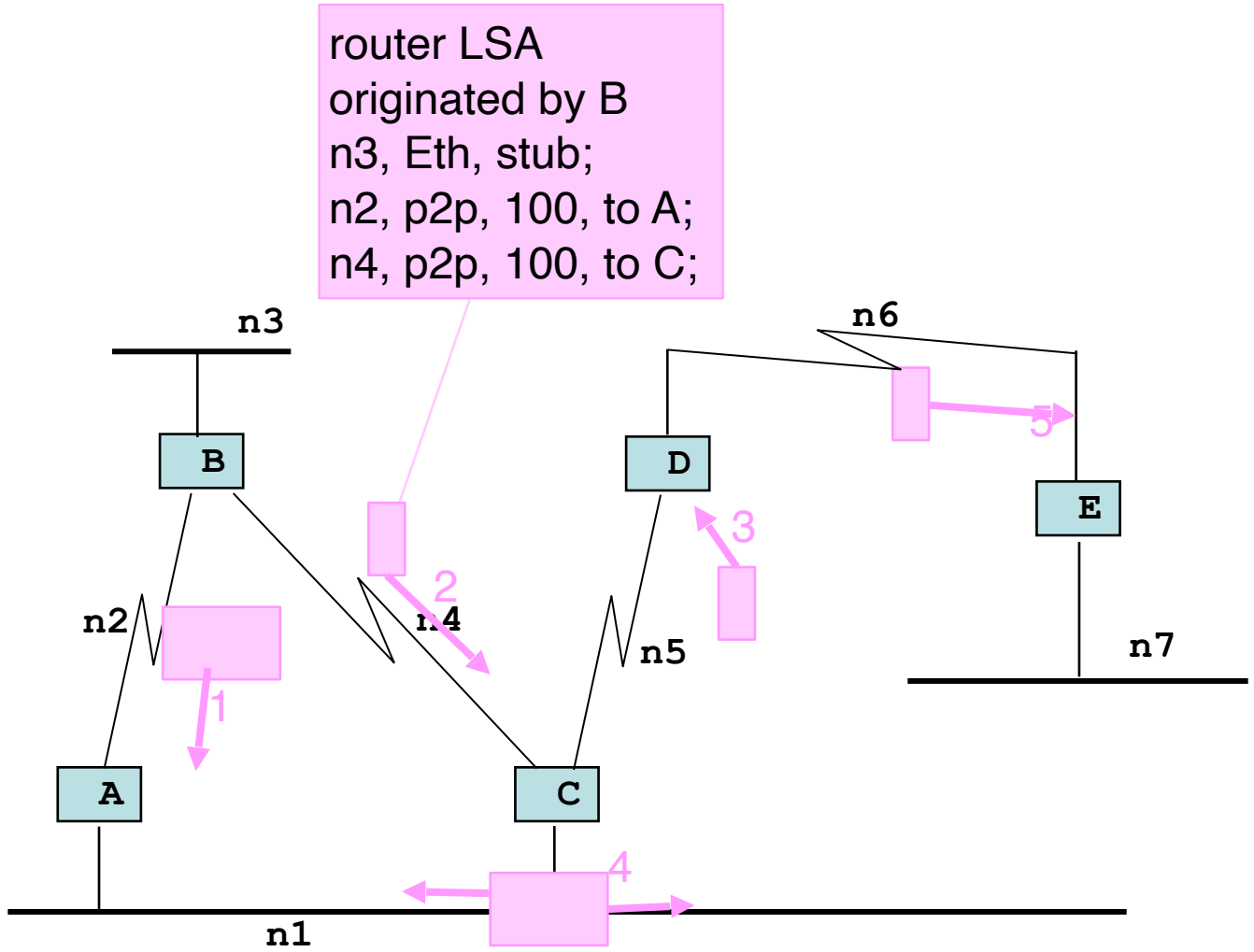
At A

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



Following the path of LSAs and explaining what information they provide:

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.
- A“stub” network means non transit, i.e. there is no other router on this network. A stub network can be reached by only a single router; so, all we need to know is how to reach this router—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.

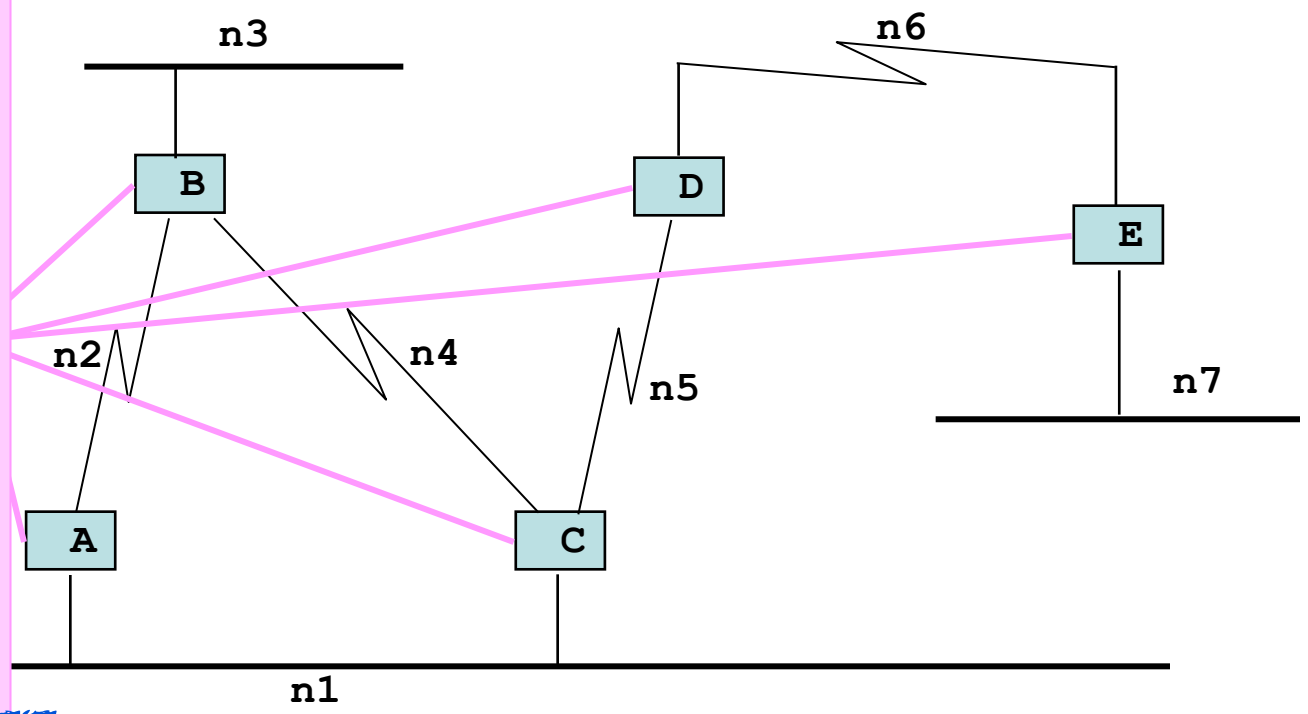
After Flooding

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

All have the *same* database.

Designated router C

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.

  A naive approach to support a LAN with a link-state routing protocol would be to consider that a LAN is equivalent to a full-mesh of point-to-point links as if each router can directly reach any other router on the LAN. However, this approach has two important drawbacks :

  (a) Each router must exchange HELLOs and link state packets with all the other routers on the LAN. This increases the number of OSPF packets that are sent and processed by each router.

  (b) Remote routers, when looking at the topology distributed by OSPF, consider that there is a full-mesh of links between all the LAN routers. Such a full-mesh implies a lot of redundancy in case of failure, while in practice the entire LAN may completely fail (the switch may fail). In case of a failure of the entire LAN, all routers need to detect the failures and flood link state packets before the LAN is completely removed from the OSPF topology by remote routers.

  In order to avoid these issues, the routers elect a designated router per LAN (and a backup designated router).

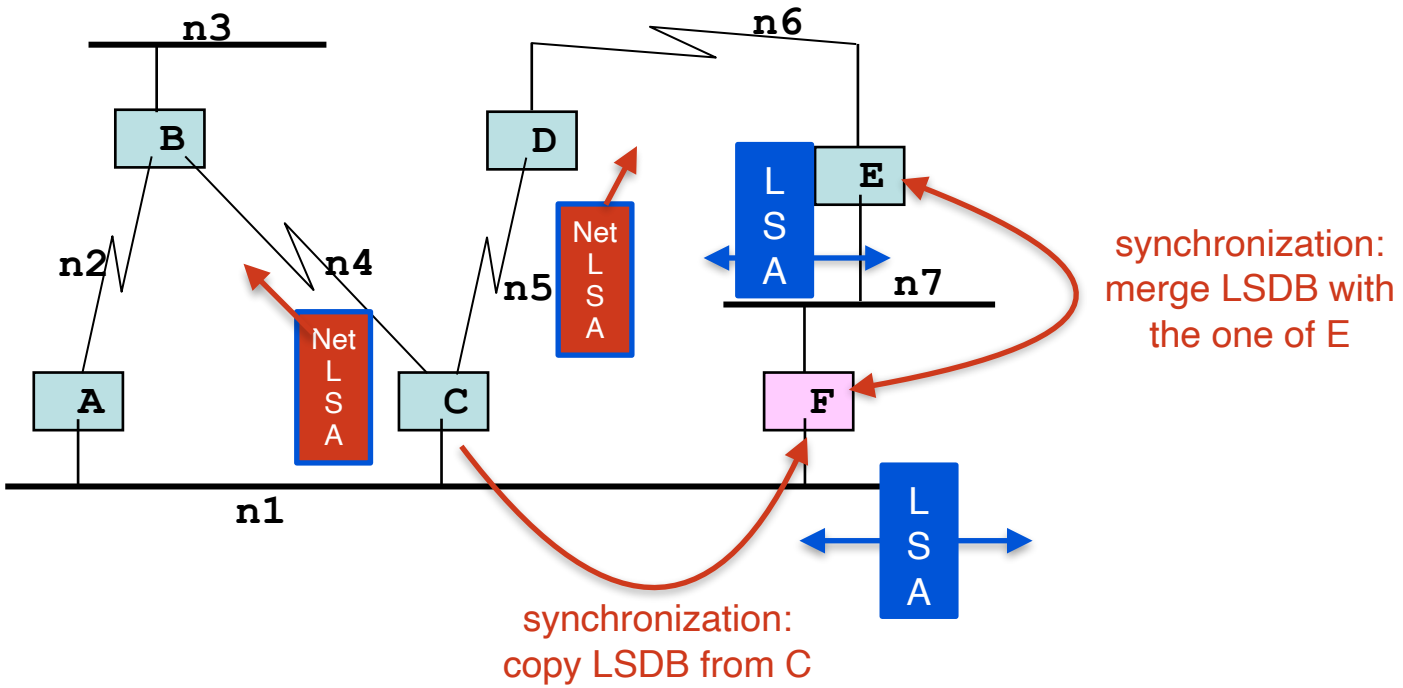
  The designated router "speaks for the switch" and sends a "network LSA" which gives the list of all routers connected to the LAN.

  Typically the designated router of a LAN is the oldest running router in it. E.g. in the previous slide assume that this is C.

- Every router that is connected to an Ethernet LAN floods a "router LSA" indicating its connection to this LAN.

- Router and Network LSAs are just 2 of the available types of LSA. At the time of writing this, there exist 11 types of LSAs. In addition to router and network LSAs, the other types are used in the multi-area case (see later in this lecture) and with external routes (see BGP lecture). There are also other 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

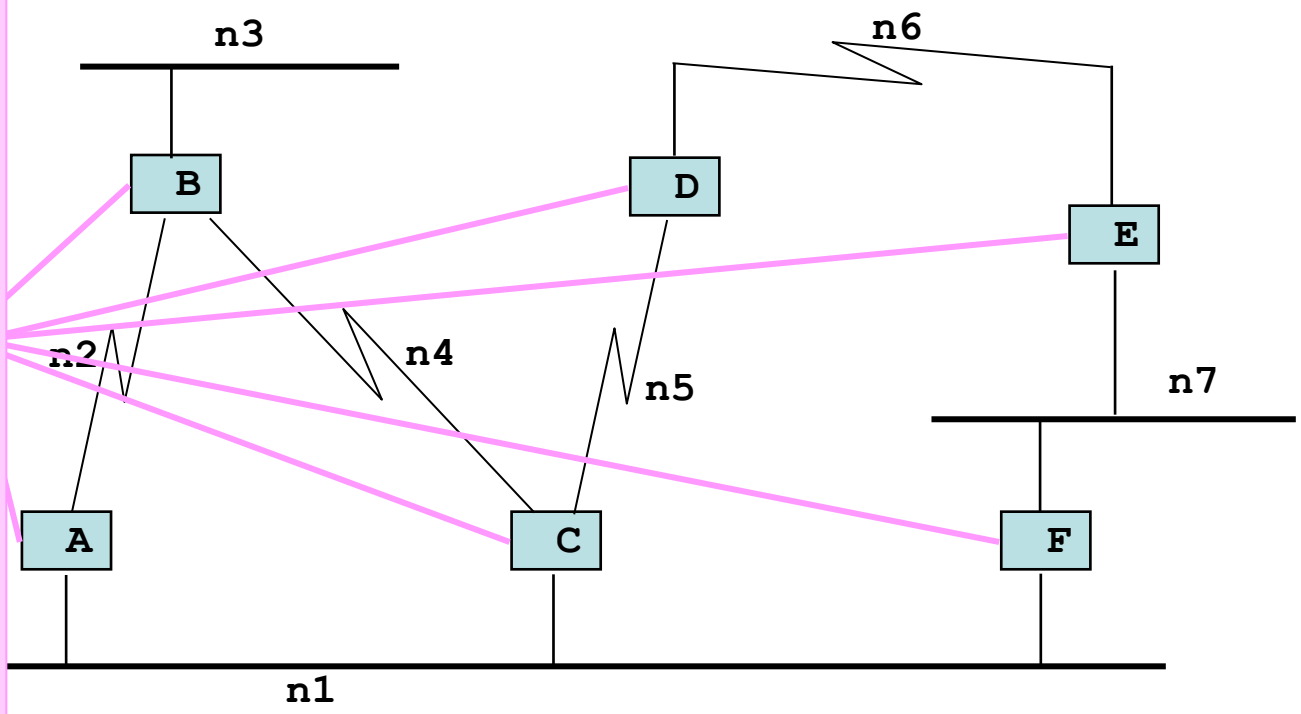


- F discovers neighbors with the hello protocol; assume F discovers C first (C is designated router for n1): F and C establish adjacency (going through a sequence of 8 states, Down to Full). During this process, F and C synchronize their Link State Data Bases (i.e. F copies its LSDB from C).
- When the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

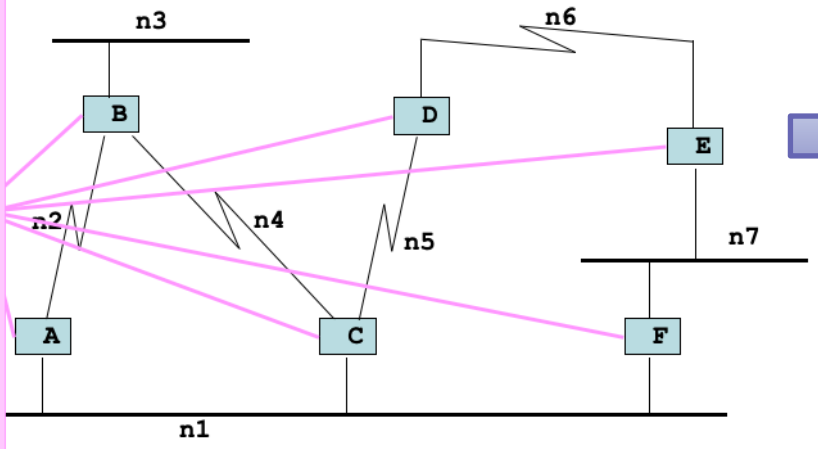
router LSA from B
n3, Eth, stub;
n2, p2p, 100, to A;
n4, p2p, 100, to C
router LSA from A
n2, p2p, 100, to B;
n1, eth, 10, DR=C
router LSA from C
n4, p2p, 100, to B;
n5, p2p, 20, to D;
n1, eth, 10, DR=C
router LSA from F
n7, eth, 10; DR=E
n1, eth, 10, DR=C
router LSA from D
n5, p2p, 20, to C;
n6, p2p,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

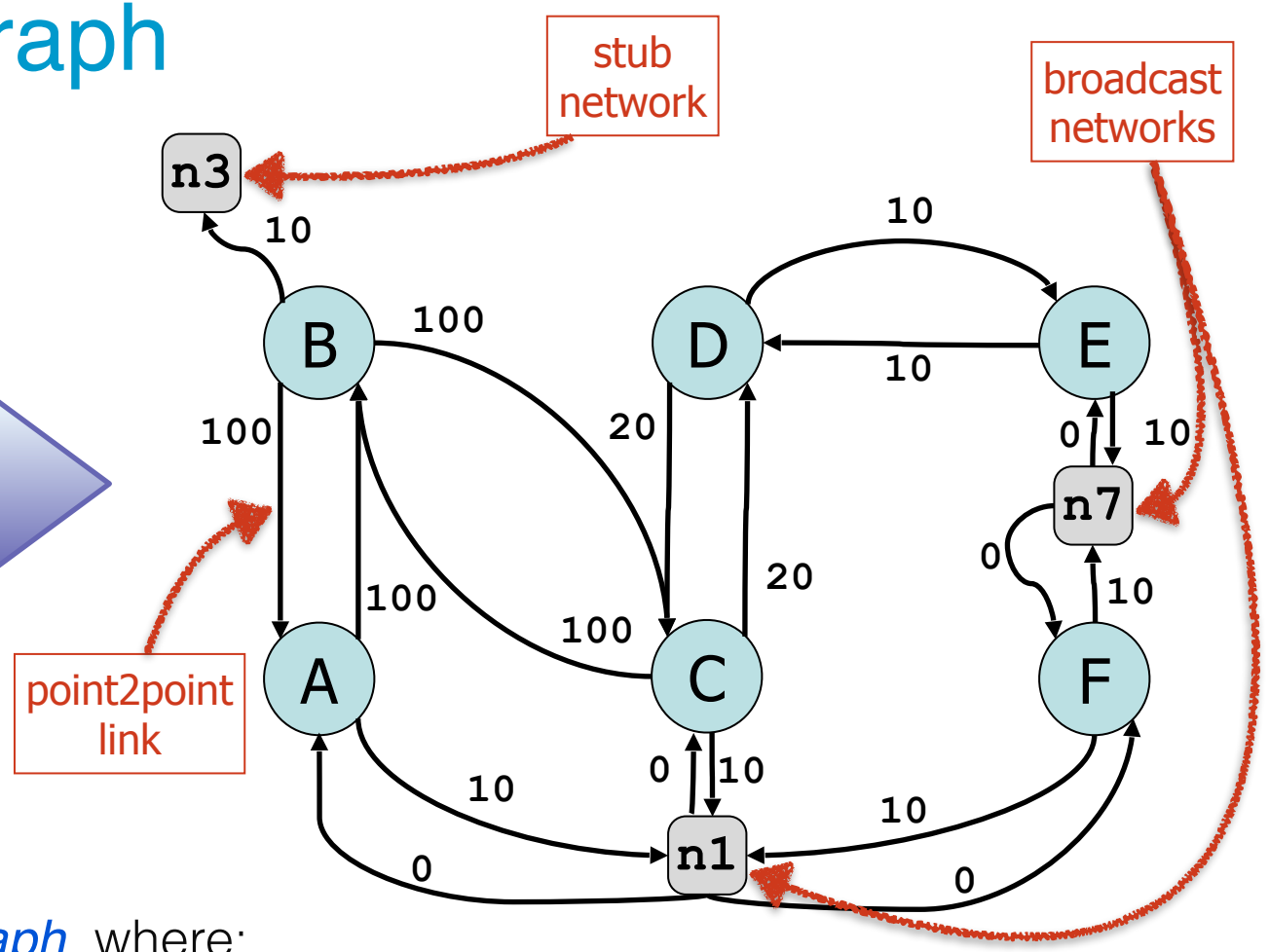


Link State Database at all routers

From LSDB to Topology graph

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; DR=E
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





- The link state database defines a *directed graph*, where:
  - every *router* and every Ethernet *network* corresponds to a *node* in the graph
  - link costs = `costs given in LS database`
  - link cost of "network node —> router" is 0 by default (also 0 inside the network LSA)

Practical Aspects

- OSPF packets are sent *directly over IP* (OSPF=protocol 89 (0x59)).
- Reliable transmission is managed by OSPF using ***acknowledgements*** and ***timers*** (like the *stop and go* protocol).

- OSPFv2 supports IPv4 only
- OSPFv3 supports IPv6 and dual-stack networks
  - 2 LSDBs for each IP variant or common LSDB with all information

- Routers IDs or Area IDs are 32 bit numbers

3. Shortest paths are found with Dijkstra's Algorithm

- Each router runs Dijkstra *independently* on its local LSDB copy. *Why?*
  - every router computes *shortest paths starting from itself*
  - LSAs guarantee *no persistent* inconsistencies (LSDBs are mostly identical)

- Each router computes *one or several* shortest paths to every other node

Dijkstra's Shortest Path Algorithm

$m(0) = 0;\ m(i) = \infty \quad \forall\, i \neq 0; V = \varnothing\ ; pred(i) = \varnothing\ \forall i;$

for $k = 0{:}N$ do

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

  if $m(i)$ is finite

    add i to V

    for all neighbors $j \notin V$ 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

$m(0) = 0;\ m(i) = \infty \quad \forall\, i \neq 0; V = \varnothing\ ; pred(i) = \varnothing\ \forall i;$

for $k = 0 : N$ do

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

  if $m(i)$ is finite

    add i to V

    for all neighbors $j \notin V$ 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 other nodes

Adds *one node at a time* to the set V of visited nodes, by picking the node that is *closest* to the source in terms of least cost path

A few notes:

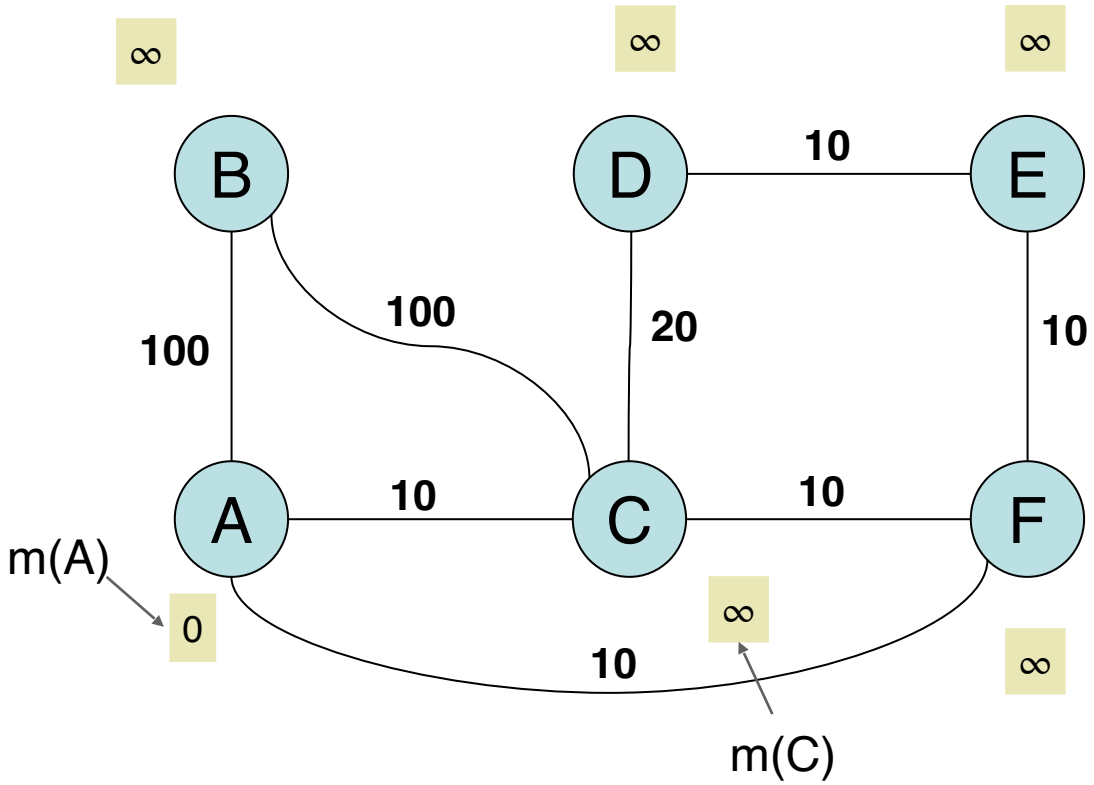
There are multiple versions of Dijkstra's algorithm. The version presented here finds all shortest paths, other versions find only one shortest path to every destination. The version presented is very close to what is really implemented in OSPF (with a difference, next-hop versus pred(), see later).

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 "visited" set; the state of this node (distance from source node 0 and set of predecessors) is the *final value* and will not change in later steps of the algorithm.

The last 3 lines of the pseudo code 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

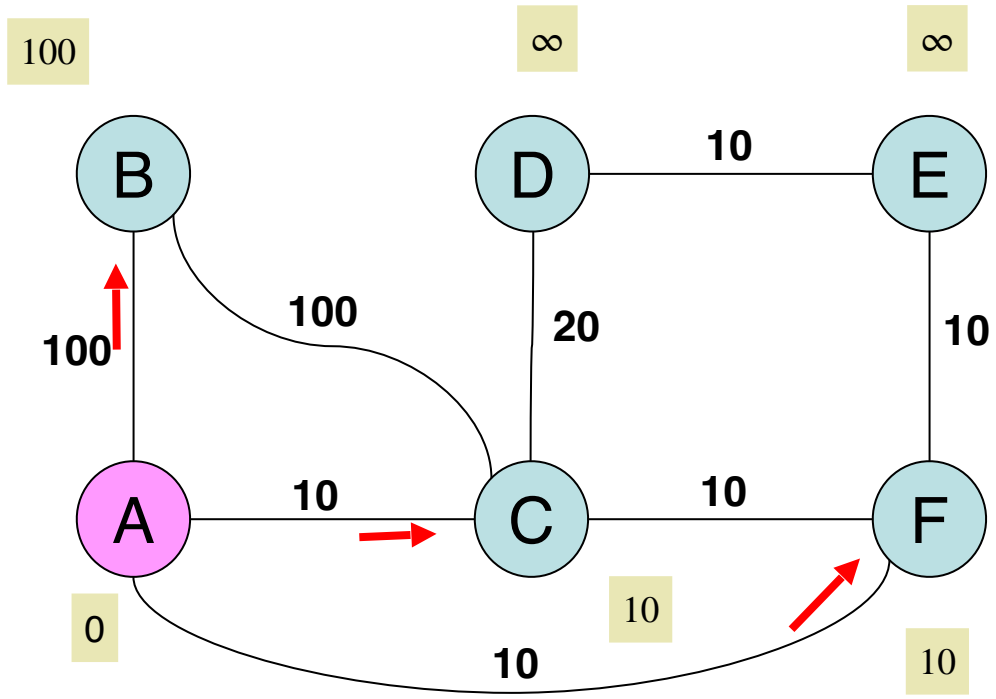


init: V = ∅

m(A)=0
m(i)=∞, $i \neq A$

pred(i)=∅

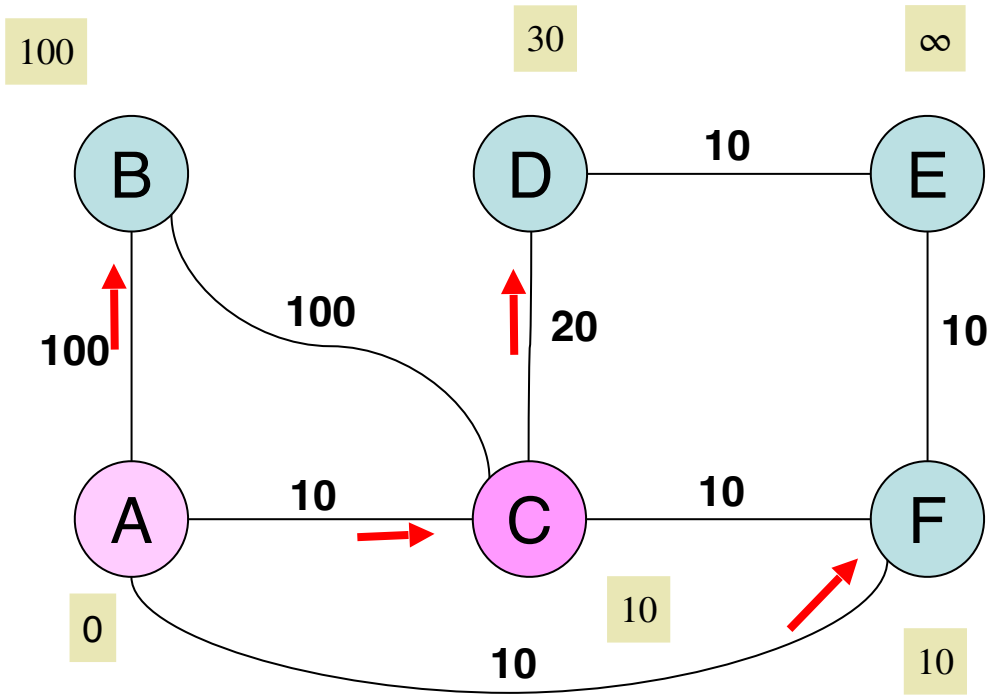
Example: Dijkstra at A
After step 1



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

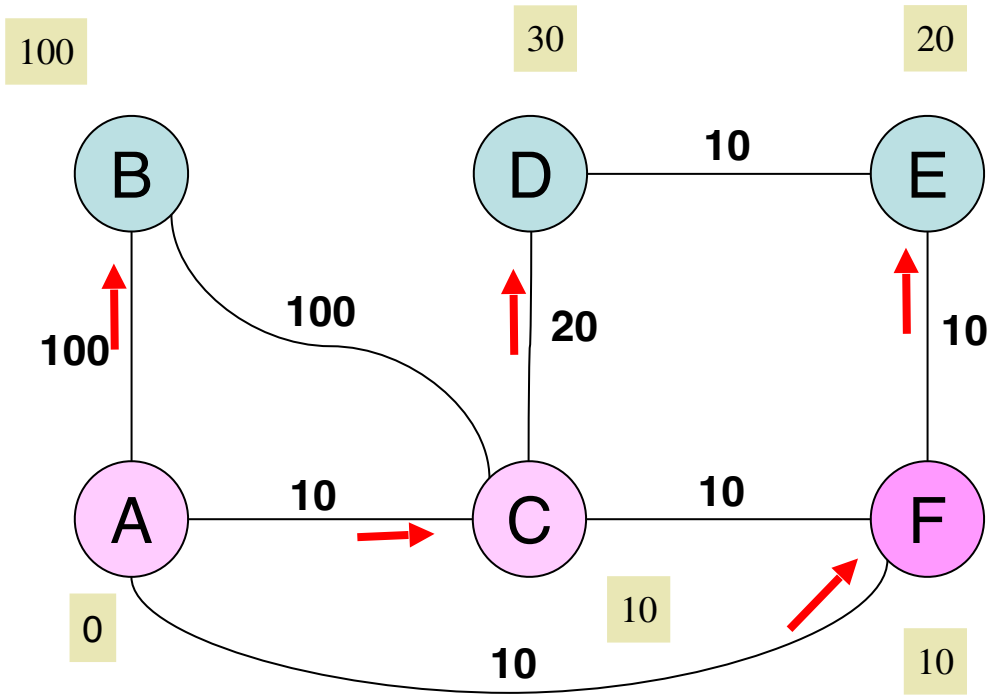
B
↑
A
red arrow from A to B means $A \in pred(B)$

Example: Dijkstra at A
After step 2



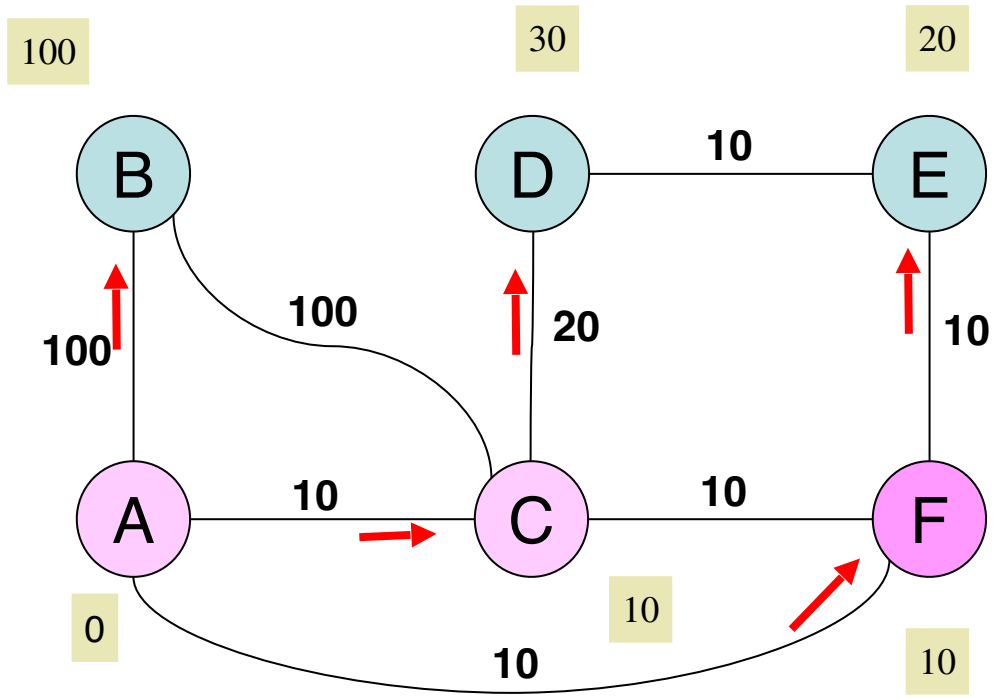
step 2:
i=C
V={A,C}
m(D)=30
pred(D)={C}
B, F unchanged

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?

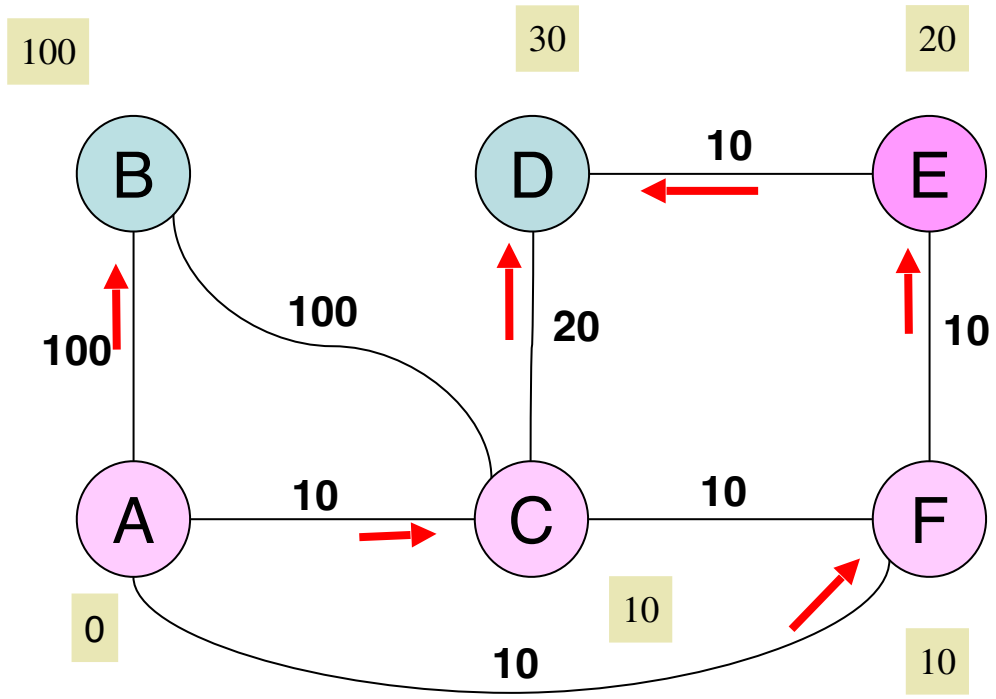


A. B
B. D
C. E
D. I don't know

Go to web.speakup.info or download speakup app

Join room 87072

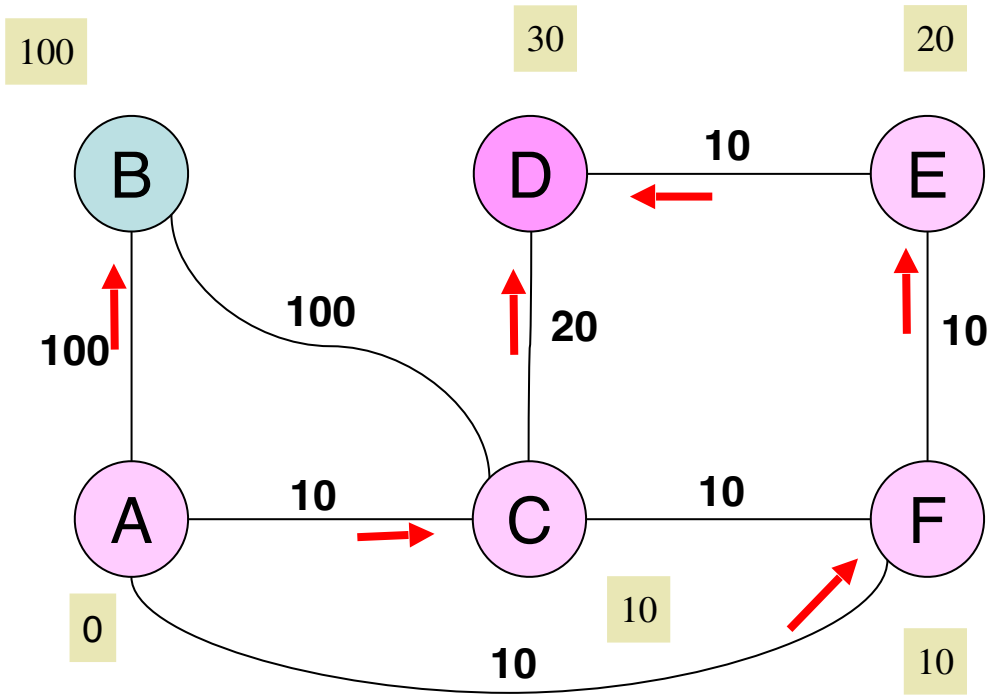
Solution: Dijkstra at A
After step 4



step 4: (Answer C)
i=E
V={A,C,E,F}
m( ) unchanged
pred(D)={C,E}

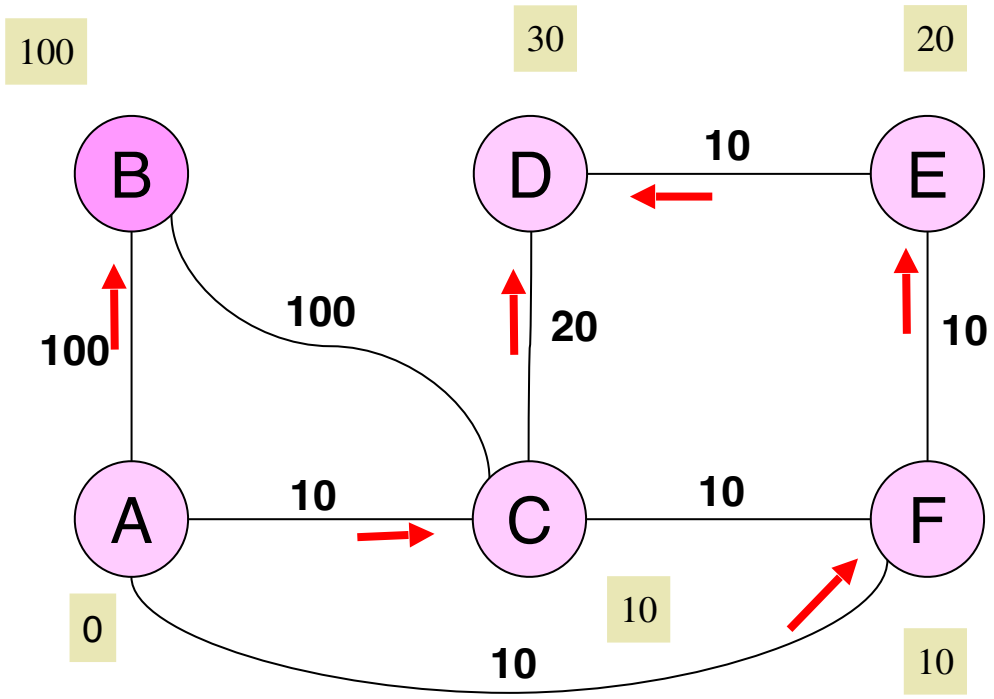
There are two equal-cost paths to D, *both* are recorded.

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

- $pred(i)$ is 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*

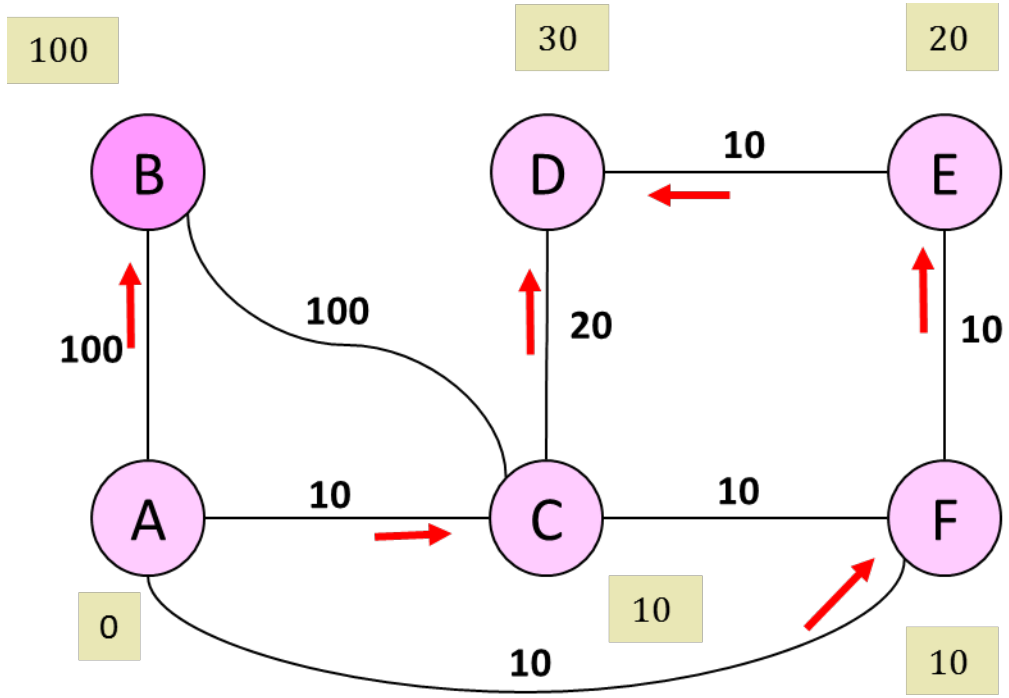
E.g., shortest paths from A to E:

A-F-E

to D:

A-C-D

A-F-E-D



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:

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

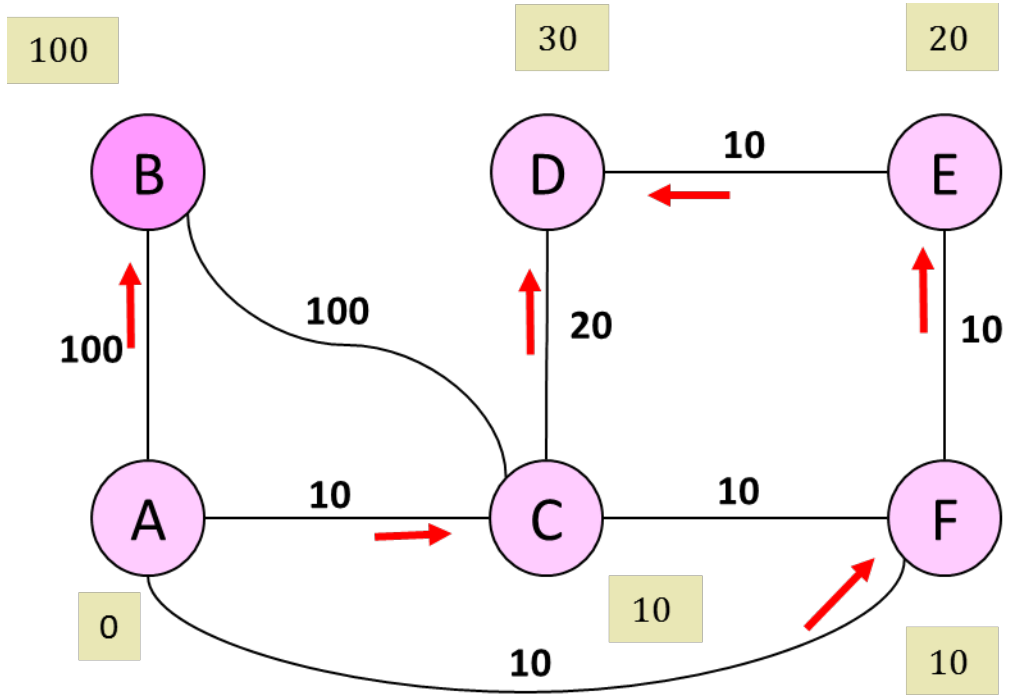
```
for k = 0:N do
        find i ∈ V that minimizes m(i)
        if m(i) is finite
                add i to V
                for all neighbors j ∈ V of i
                        if m(i) + c(i,j) < m(j)
                                m(j) = m(i) + c(i,j)
                                derive nextHopTo(j) 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) ∪ nextHopTo(i)      // add shortest path to j via i
```

From shortest paths to Forwarding Table

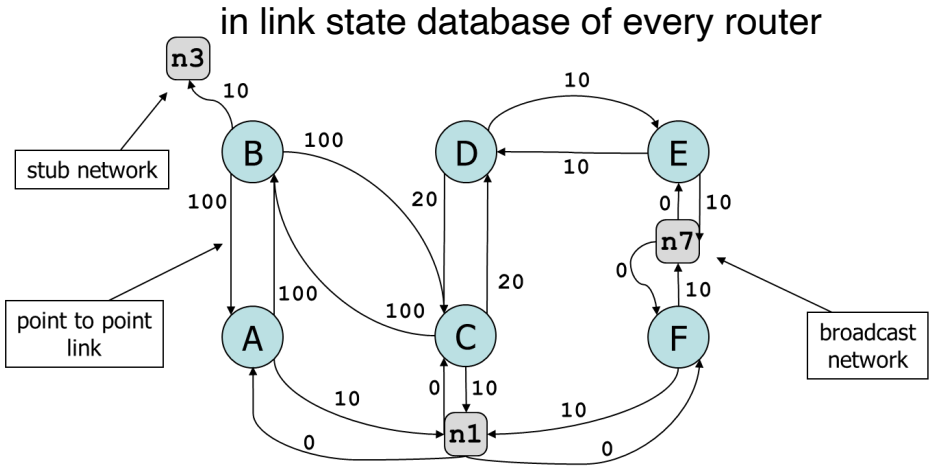
Router A keeps in its forwarding table only the *next-hop* and the *distance* to every destination (not the entire path):

At A

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



Bringing back the network nodes



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

In practice, OSPF adds to the graph the network nodes, which makes the graph *larger*.

To optimize computation:

- first stub networks (such as n3) *are removed*
- then, Dijkstra is run and the forwarding table contains costs and next hop to edge routers (such as B);
- finally, stub networks are *added to the forwarding table* one by one, using information about edge routers

4. Equal Cost Multipath

- OSPF supports multiple shortest paths
- IP allows to have multiple next-hops to the same destination in the forwarding table
  - good: it exploits the *redundancy* of paths
  - bad: the number of multiple paths may be *large*

  so, typically we use a *limit* of multiple paths

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 randomly 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

Go to web.speakup.info or download speakup app

Join room 87072

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 even though it was just received out of order). Therefore, an alternative approach, called *per-flow load balancing* requires that packets of the same flow to be sent to the same next-hop. The definition of a flow depends on the system: a flow is typically identified by the src/dest IP addresses and, in some systems, by next header type and (if they exist), src/dest ports.

Per-flow load balancing is implemented by applying a *hash function* to the *flow identifier* of each packet.

$$hash(flowID) \in [0,1],$$

and comparing against a number of thresholds.

Example: assume there are 2 possible next-hops for a packet. If $h(flowID) < 0.5$ the packet is sent to the first, else to the second. The flow identifier (which, as stated above, is typically a 5-tuple of: src/dest IP addresses, src/dest ports, and transport-layer protocol) is the same for all packets of the same TCP connection; so these packets will be sent to the same next-hop.

So, answers A and D !

5. OSPF's reaction to *changes* in topology

- Changes occur when links or routers fail or reboot

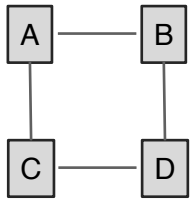
- The routers detect failures via:
  - OSPF's hello protocol (after several seconds, in general)
  - Bidirectional Forwarding Detection (BFD) protocol:
    hello protocol at the Ethernet layer (fast: after 10 ms, independent of OSPF)
  - *directly* — no power on the cable

- Upon a detected failure/change:
  - router floods *new LSA* with new sequence number and *new (alive)* neighbors
  - neighbors propagate new LSA
  - all routers update their LSDB, re-compute shortest paths and forwarding tables

➡ *Takes time,*

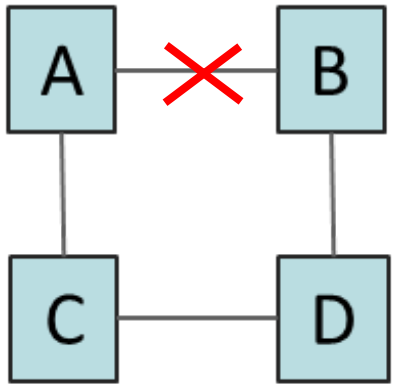
➡ *may result in transient routing loops and packet drops*

Link State Database and routing table at A

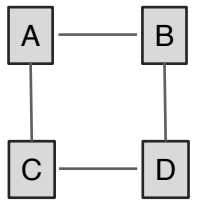


Dst	i/f	Nxt hp	cst
B	east	on-link	10
C	south	on-link	10
D	east	B	20
D	south	C	20

Example at t_0

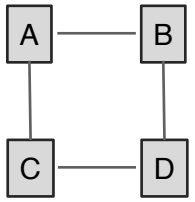


Link State Database and routing table at B



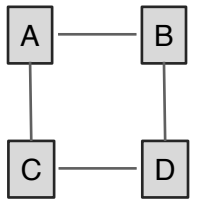
Dst	i/f	Nxt hp	cst
A	west	on-link	10
D	south	on-link	10
C	west	A	20
C	south	D	20

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

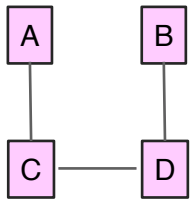
Link State Database and routing table at D



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

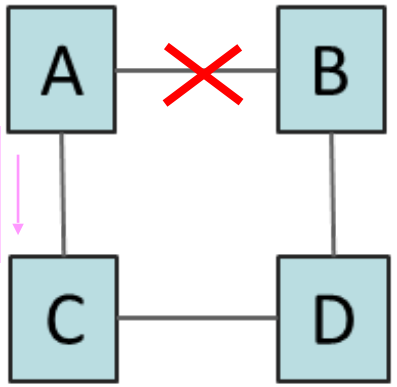
t_0: Link A-B crashes

Link State Database and routing table at A



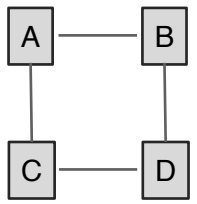
Dst	i/f	Nxt hp	cst
~~B~~	~~south~~	~~C~~	~~30~~
C	south	on-link	10
~~D~~	~~east~~	~~B~~	~~20~~
D	south	C	20

Example at t_1



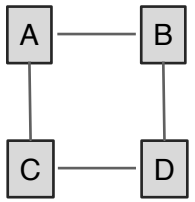
LSA from A
A to C, cost=10

Link State Database and routing table at B



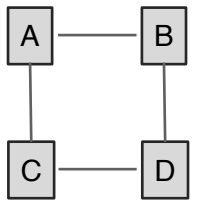
Dst	i/f	Nxt hp	cst
A	west	on-link	10
D	south	on-link	10
C	west	A	20
C	south	D	20

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



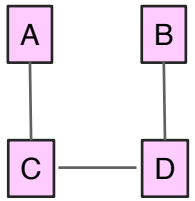
Dst	i/f	Nxt hp	cst
B	north	on-link	10
C	west	on-link	10
A	north	B	20
A	west	C	20

t_1: A detects failure first; declares B as invalid neighbor, declares link A-B as invalid, updates its link state database, sends a new LSA to C, with origin A and recomputes its forwarding table.

A *transient routing loop* is created between A and C for destination B.

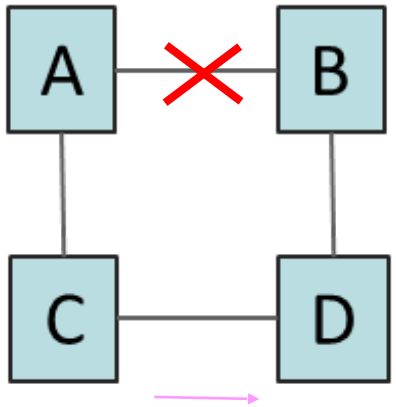
Traffic sent by B to A dies on the link, until B detects the failure. So, half of the traffic from D to A is also lost.

Link State Database and routing table at A



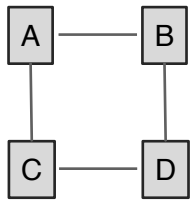
Dst	i/f	Nxt hp	cst
~~B~~	~~south~~	~~C~~	~~30~~
C	south	on-link	10
~~D~~	~~east~~	~~B~~	~~20~~
D	south	C	20

Example at t_2



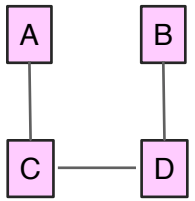
LSA from A
A to C, cost=10

Link State Database and routing table at B



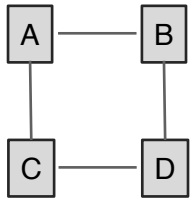
Dst	i/f	Nxt hp	cst
A	west	on-link	10
D	south	on-link	10
C	west	A	20
C	south	D	20

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

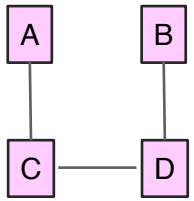
Link State Database and routing table at D



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

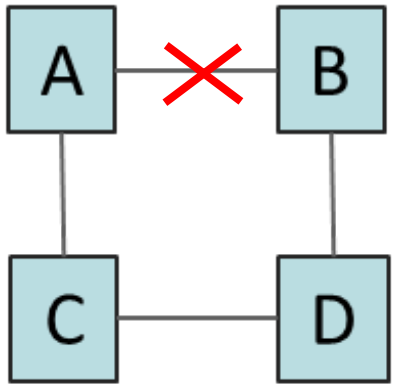
t_2: C receives LSA from A, updates its LSDB, forwards this LSA to D and recomputes forwarding table. B hasn't detected the failure yet. There is no routing loop but traffic sent by B to A dies on the link and half of the traffic from D to A is lost.

Link State Database and routing table at A

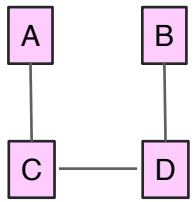


Dst	i/f	Nxt hp	cst
B	south	C	30
C	south	on-link	10
~~D~~	~~east~~	~~B~~	~~20~~
D	south	C	20

Example at t_3



Link State Database and routing table at B

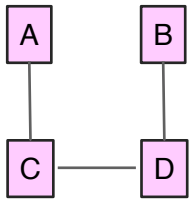


Dst	i/f	Nxt hp	cst
A	south	D	30
D	south	on-link	10
~~C~~	~~west~~	~~A~~	~~20~~
C	south	D	20

LSA from B
B to D, cost=10

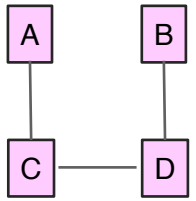
LSA from A
A to C, cost=10

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 LSDB, forwards this LSA to B and recomputes forwarding table.
At about the same time, B also detects failure; declares A as invalid neighbor, declares link A-B as invalid, updates its LSDB, sends a new LSA to D, with origin B and recomputes routing table.
All LSDBs now have the same contents and new routes are in place.

When a router crashes, how do other *remote* routers in area detect the crash?

Go to web.speakup.info or download speakup app

Join room 87072

A. By a newly flooded LSAs: the immediate neighbors detect loss of adjacency and flood new LSAs with the updated list of adjacent/neighbor 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 neighbors detect the failure well before the LSA ages out (1 hour by default)

With the hello protocol and BFD, only immediate neighbors can detect the crashed router.

6. OSPF: Security aspects

Attacks against routing protocols:

(1) send invalid routing information —> disrupt network operation

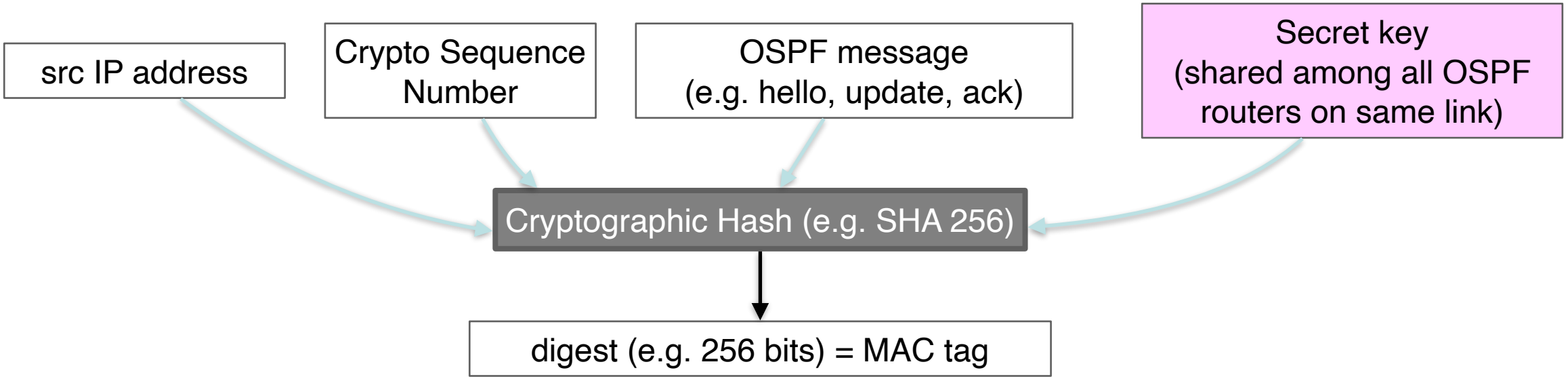
(2) send forged routing information —> change network paths

(3) denial of service attacks

OSPF *authentication* mitigates (1) and (2)

- 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

OSPF Type 3 Authentication using MAC tags and shared keys

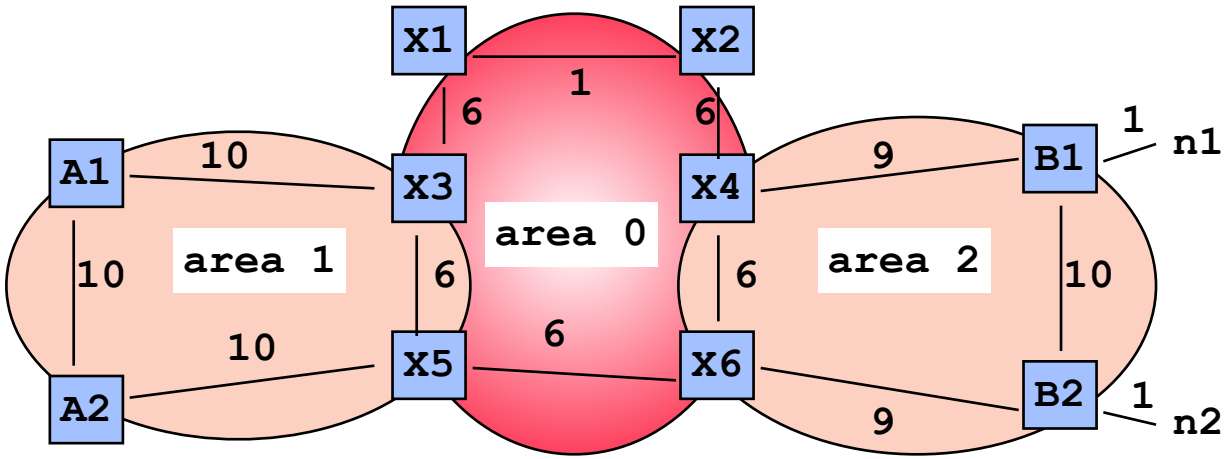


- Digest is appended to OSPF message, and is used as a *message authentication code (MAC tag)*
- Secret key is shared and has a short lifetime:
  - Neighbors have pre-installed the same list of keys (installation via an *out-of-band* mechanism)
  - A key index in the authentication header of an OSPF message points to the used key
- Crypto Sequence Number avoids *replay attacks*
  - 64-bit number incremented for every OSPF packet; numbers never wrap around (in 10^{11} years)
  - First 32 bits are a “boot count” saved on hard disk to avoid collision of numbers after reboot
  - The number is also added to OSPF authentication header in cleartext (for the neighbor to verify the MAC tag)

7. OSPF with Multiple Areas

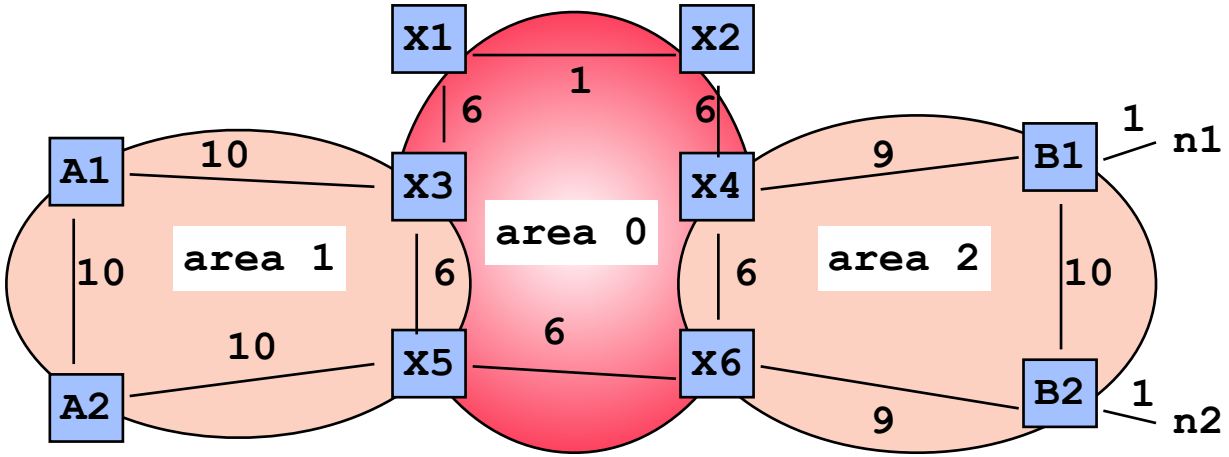
Why? LSA flooding does *not scale* in very large networks

- OSPF uses *multiple* areas and a 2-level *hierarchy*
  - a *backbone area* (area 0)
  - several non-backbone areas
- All inter-area traffic goes through backbone area 0

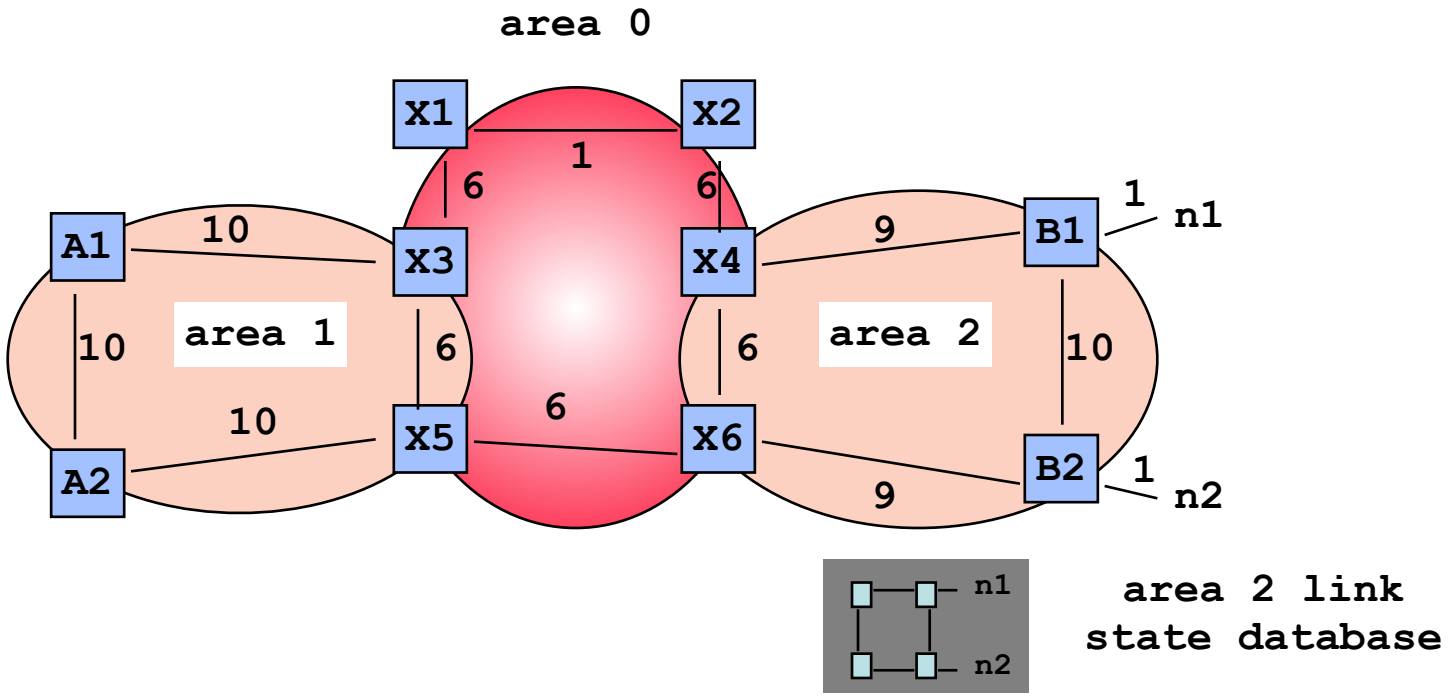


Principles of OSPF Multi-Area Operation

1. Within an area, use classic link state—single LSDB per area (∀routers in area)
2. Between areas, use *border routers:*
   - belong to both areas and have 2 LSDBs
     - e.g., X4 belongs to area 2 and 0 and has one LSDB for each area
   - inject *aggregated distance (cost) information* learnt from one area into the other area, by using *summary LSAs*



Toy Example Step1



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.

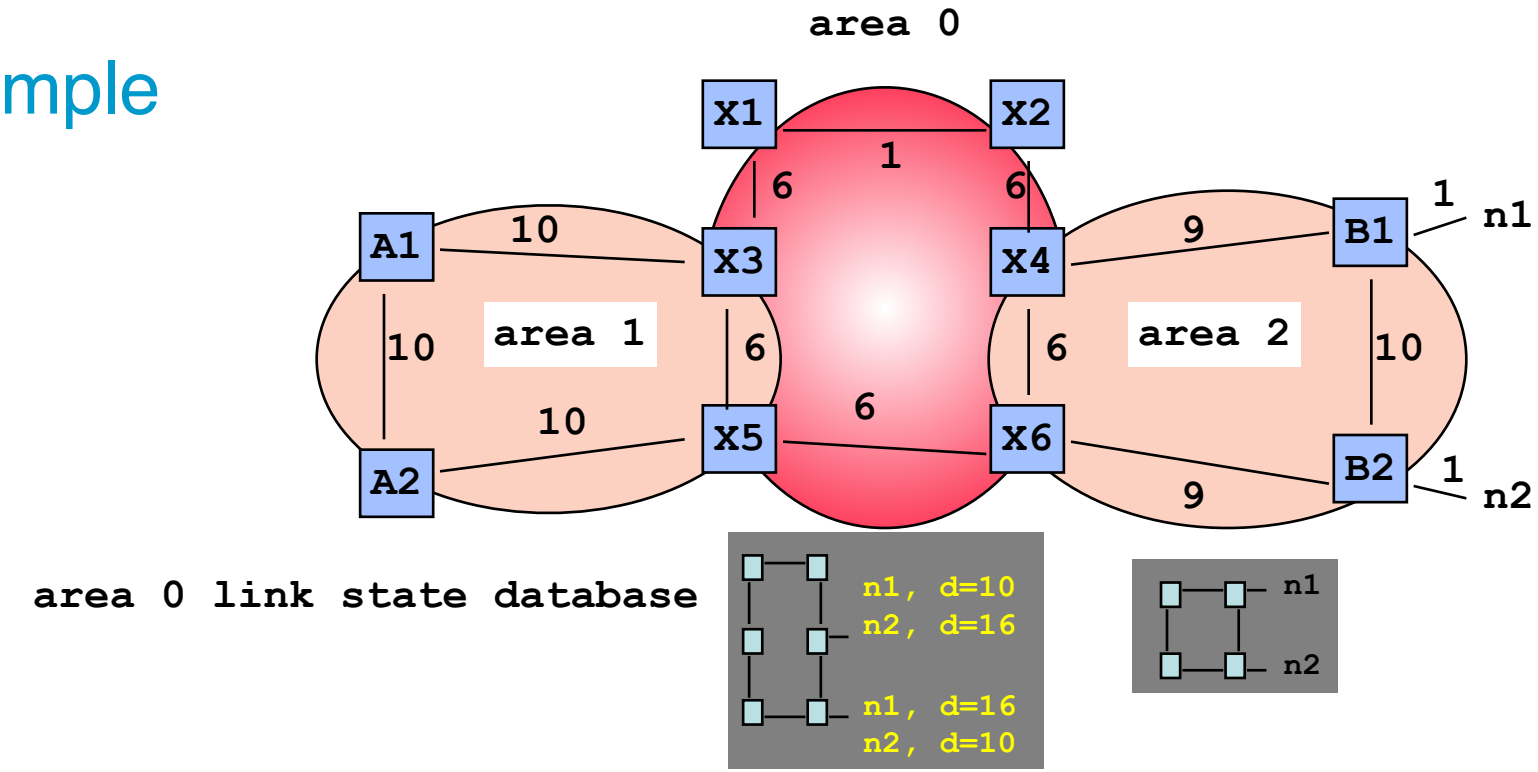
All routers in area 2 have the same link state database, shown above.

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



X4 and X6 each flood into area 0 a *summary LSA* indicating their distances to n1 and n2.

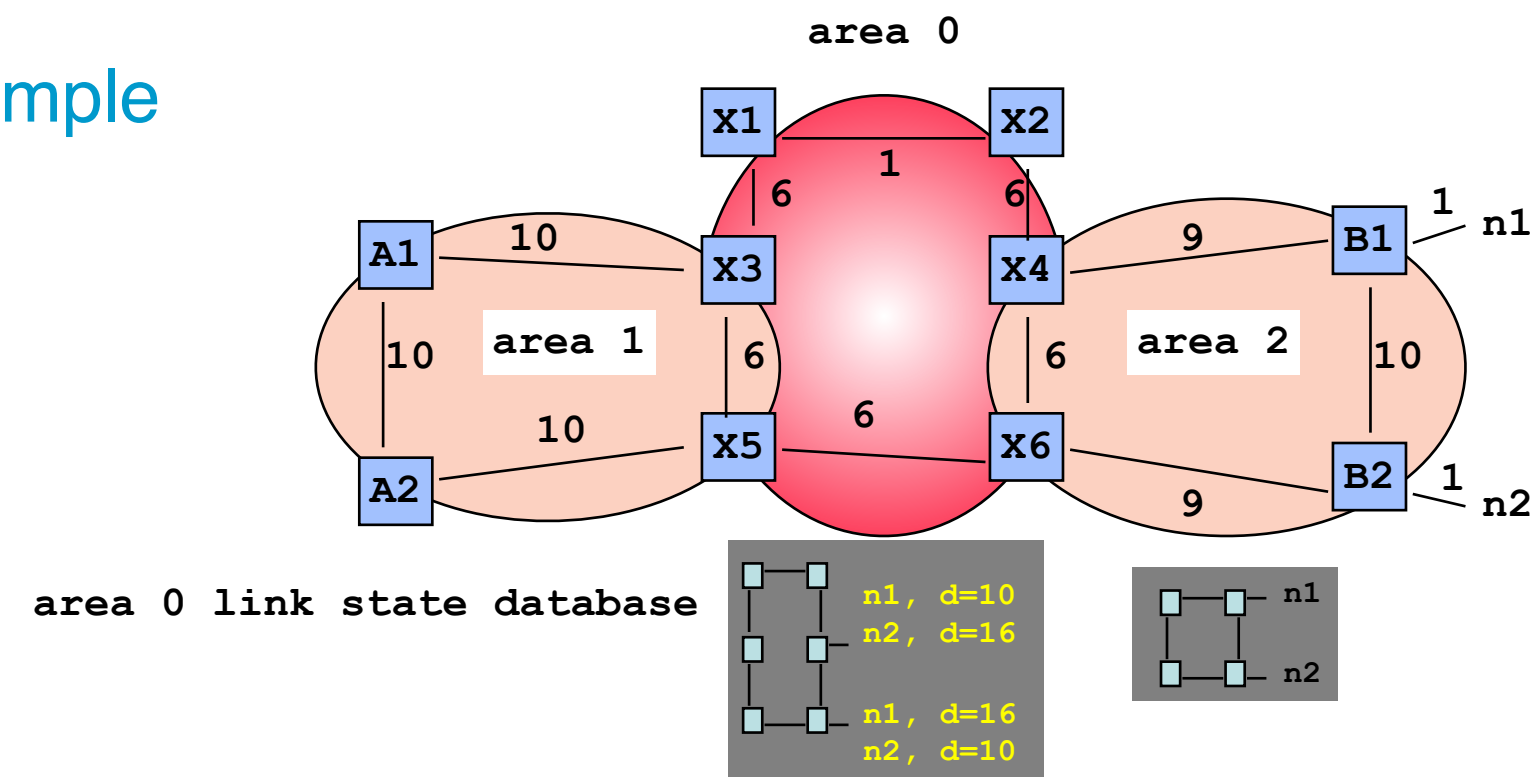
All routers in area 0 now have the same link state database, shown above.

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. E.g.:

$$d(\texttt{self}, n1) = \min_{\texttt{BR} \in \texttt{Area0}} \{d(\texttt{self}, \texttt{BR}) + d(\texttt{BR}, n1)\},$$

where BR is a border router.

Toy Example Step2 (cont'd)



Router X3 computes:

$$d(X3, n1) = \min\left\{ d(X3,X4) + d(X4,n1), d\left(X3,X6\right) + d\left(X6,n1\right) \right\} = \min(\ 23,\ 28) = 23$$

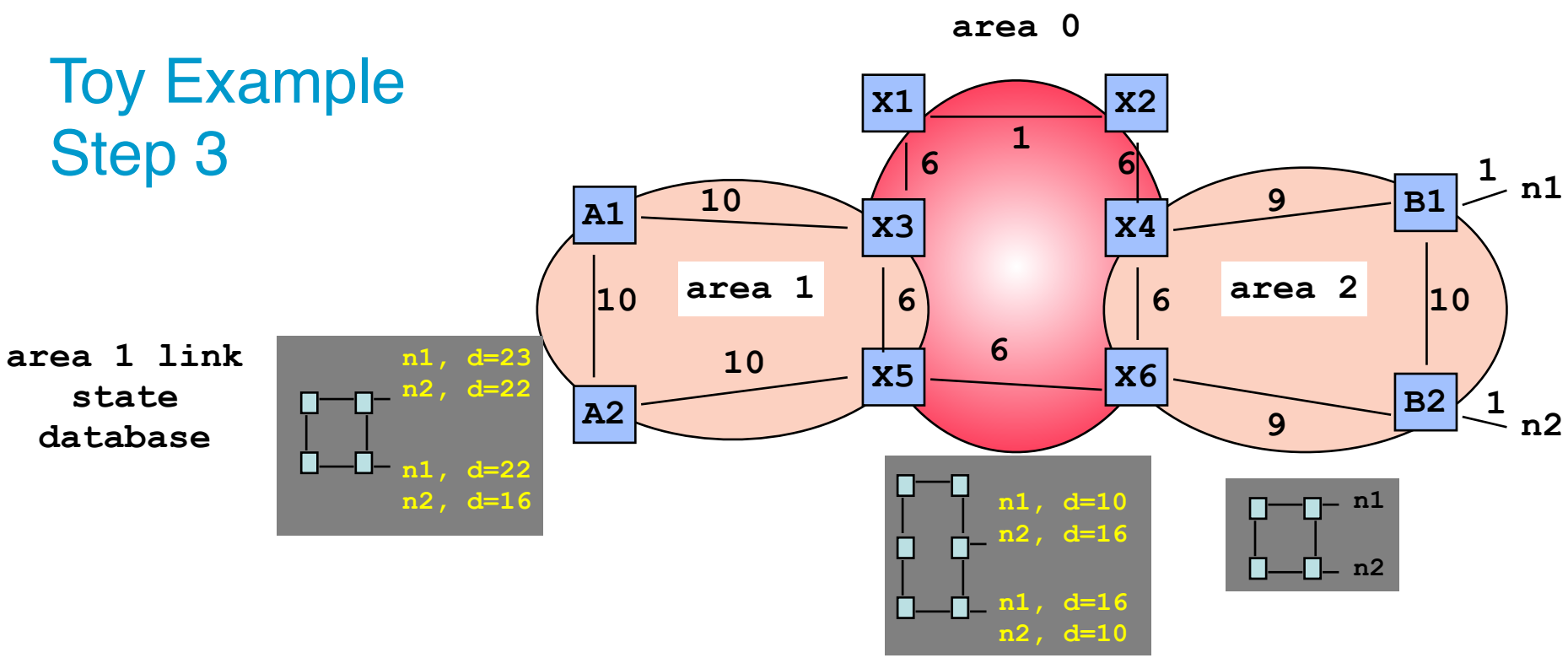
$$d(X3, n2) = \min\left\{ d(X3,X4) + d(X4,n2), d\left(X3,X6\right) + d\left(X6,n2\right) \right\} = \min(\ 29,\ 22) = 22$$

The process can be used to compute not only the distance, but also the *next hop*:

- e.g. 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.

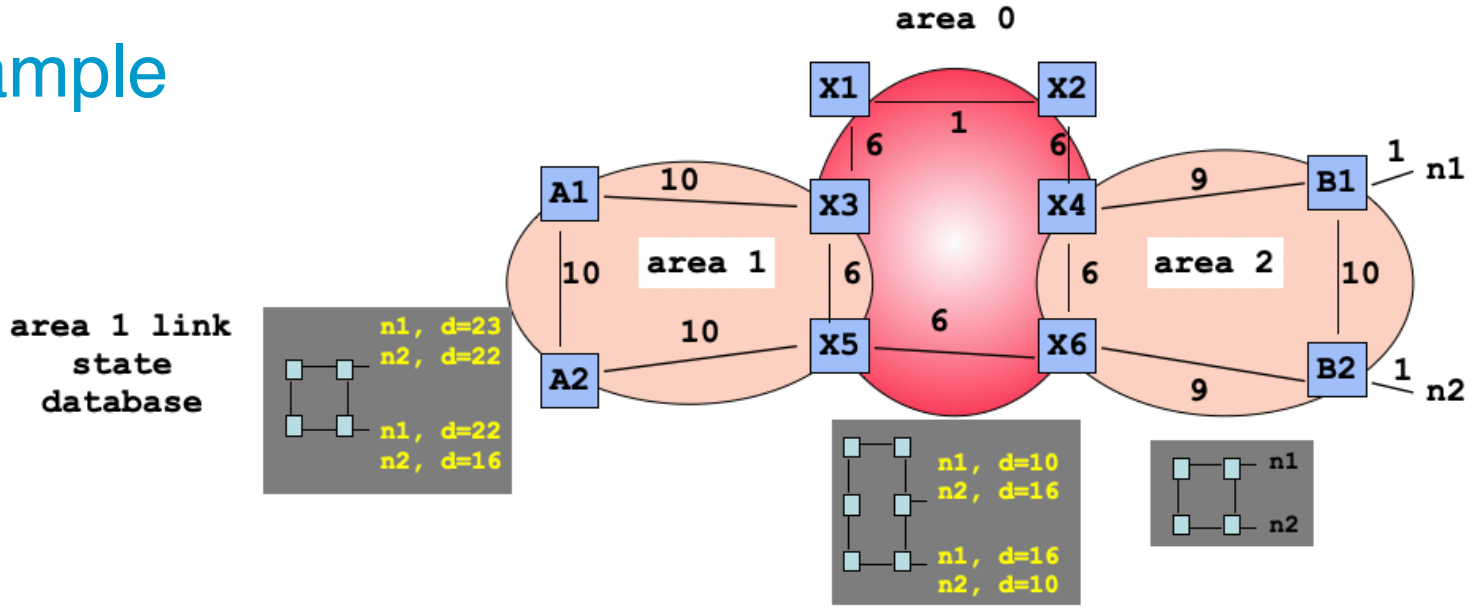
All routers in area 1 now have the same link state database, shown above.

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

$$d(\text{self}, n1) = \min_{\text{BR} \in \text{Area1}} \{d(\text{self}, \text{BR}) + d(\text{BR}, n1)\}$$

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

Toy Example Step 3



Go to web.speakup.info or download speakup app

Join room 87072

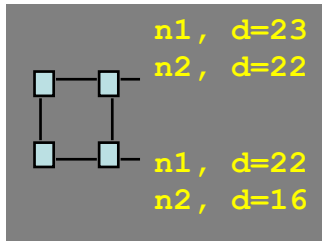
$$d(\text{self}, n1) = \min_{\text{BR} \in \text{Area1}} \{d(\text{self}, \text{BR}) + d(\text{BR}, n1)\}$$

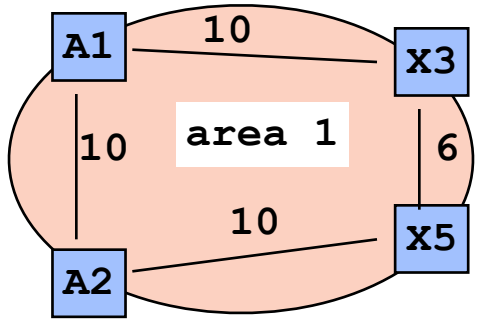
When applying the Bellman-Ford formula to compute $d(\text{self}, n1)$, how does a router such as A1 know the values of $d(\text{self}, BR)$ and $d(BR, n1)$?

A. $d(\text{self}, BR)$ from its routing table after applying Dijkstra and $d(BR, n1)$ from the summary LSA received
B. $d(BR, n1)$ from its routing table after applying Dijkstra and $d(\text{self}, BR)$ from the summary LSA received
C. both from its routing table after applying Dijkstra
D. both from the summary LSA
E. I don't know

Solution

All routers in area 1 have only their routing tables from step 1 + this information —>





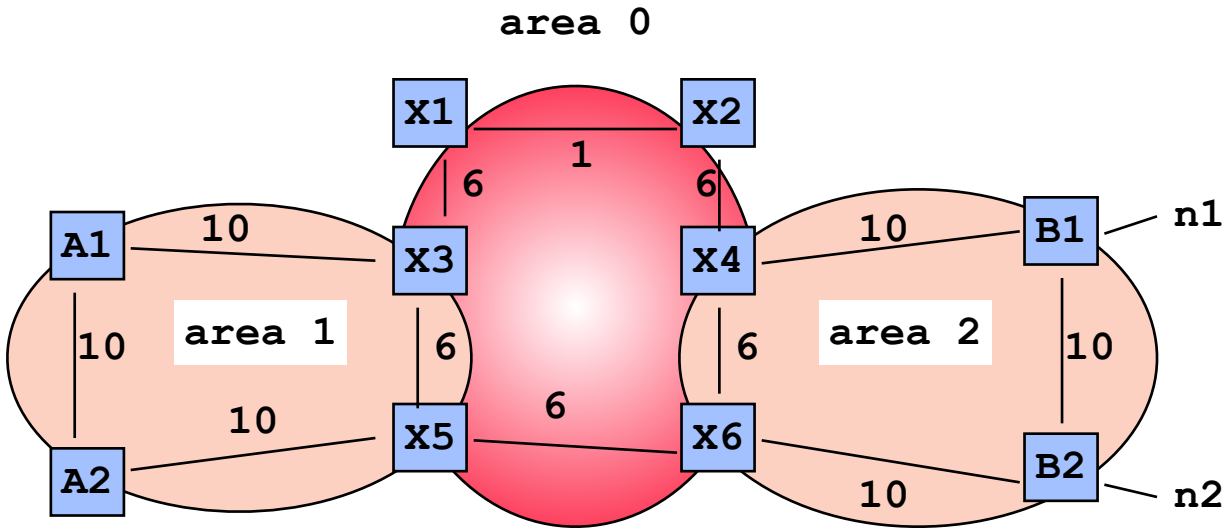
The border routers are in area 1, so A1 knows d(self, BR) from its routing table (after applying Dijkstra).

d(BR, n1) is known from a summary-LSA, which 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



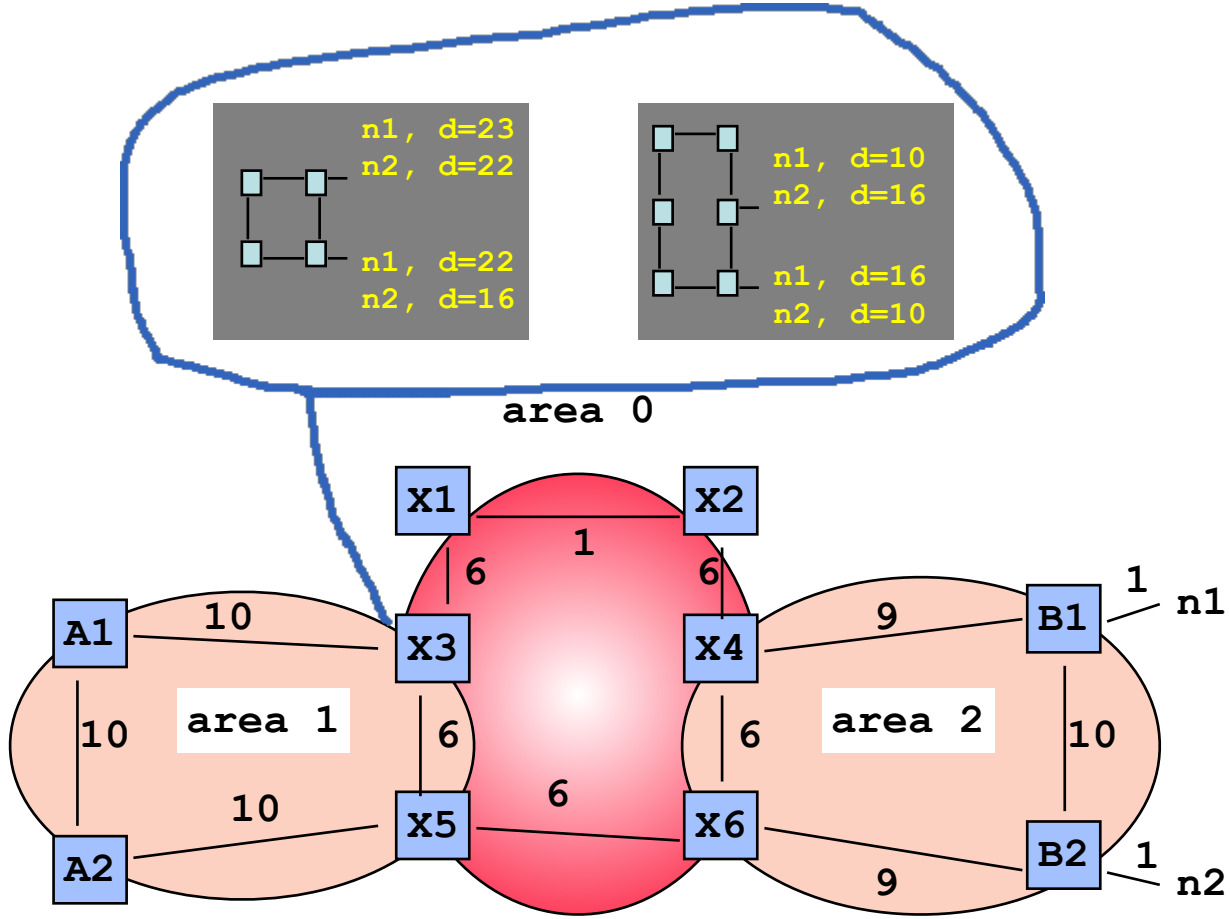
Go to web.speakup.info or download speakup app

Join room 87072

Solution

Answer B.

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



8. Other Uses of LSAs

- With LSAs, routers have a "complete network view" (e.g. topology, costs, latencies, attached subnets, etc.)
  - ➡ This info can be used by other network functions:

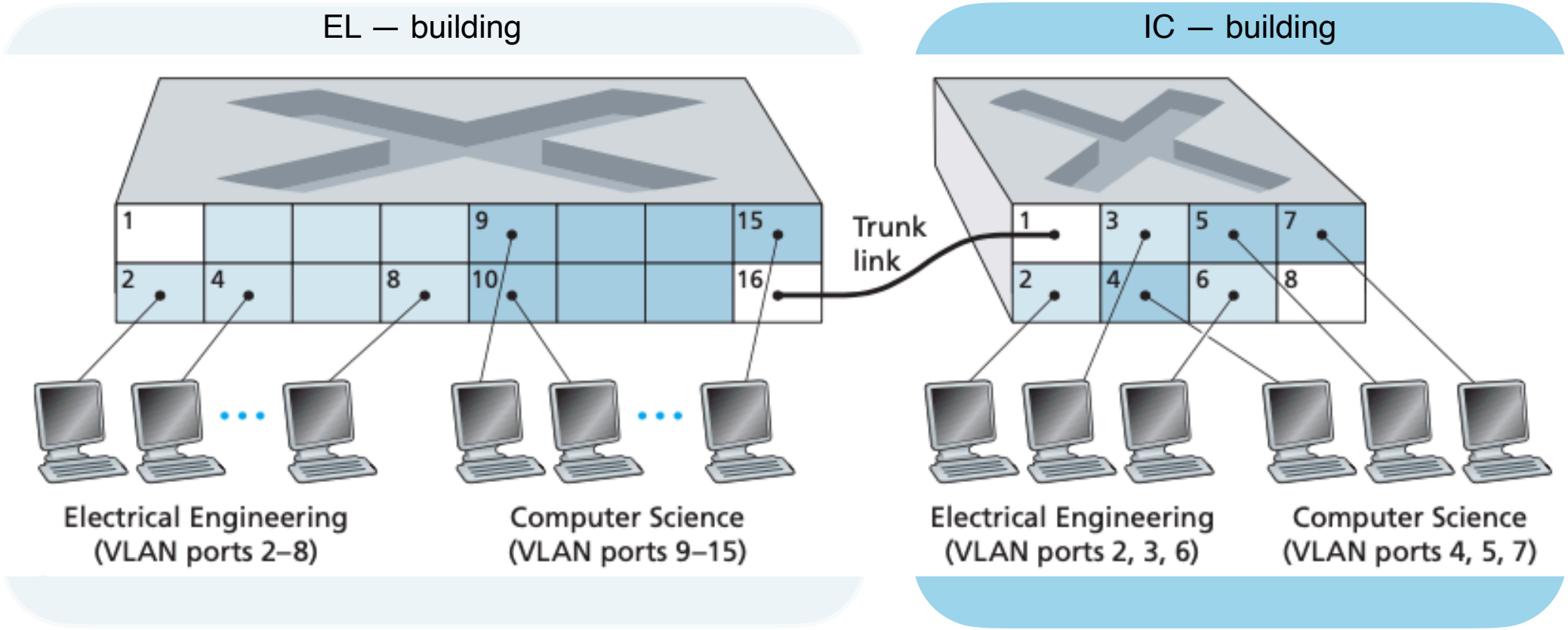
    source routing:

    an edge router computes the entire path (not only the next-hop), then writes it in an extension packet header
    - Avoids transient loops / supports fast re-route after failure
    - Used in deterministic networks (industrial applications)

    multi-class routing:
    - compute different routes for different types of services (e.g. voice, video) based on different objectives (e.g. cost vs latency)
    - use the "differentiated-services" field in IP header to forward traffic

    LS bridging (VLAN-bridging via LS shortest paths):
    - connect different layer-2 VLANs ***over IP*** instead of trunk cables

Reminder: VLANs



- The switches implement 2 virtual LANs
  - the Electrical Engineering LAN in light blue and
  - the Computer Science LAN in blue color
- The switches are interconnected via trunk ports (16 and 1) that belong to both LANs
- The trunk ports use VLAN tags to notify each other about which arriving frame belongs to which VLAN

LS Bridging

You want to bridge VLANs across a campus *without direct cables*.

A solution: use routers that also implement a layer-2 switch,

and *tunnel* (encapsulate) MAC packets in IP!

Problem: automatic creation of tunnels.

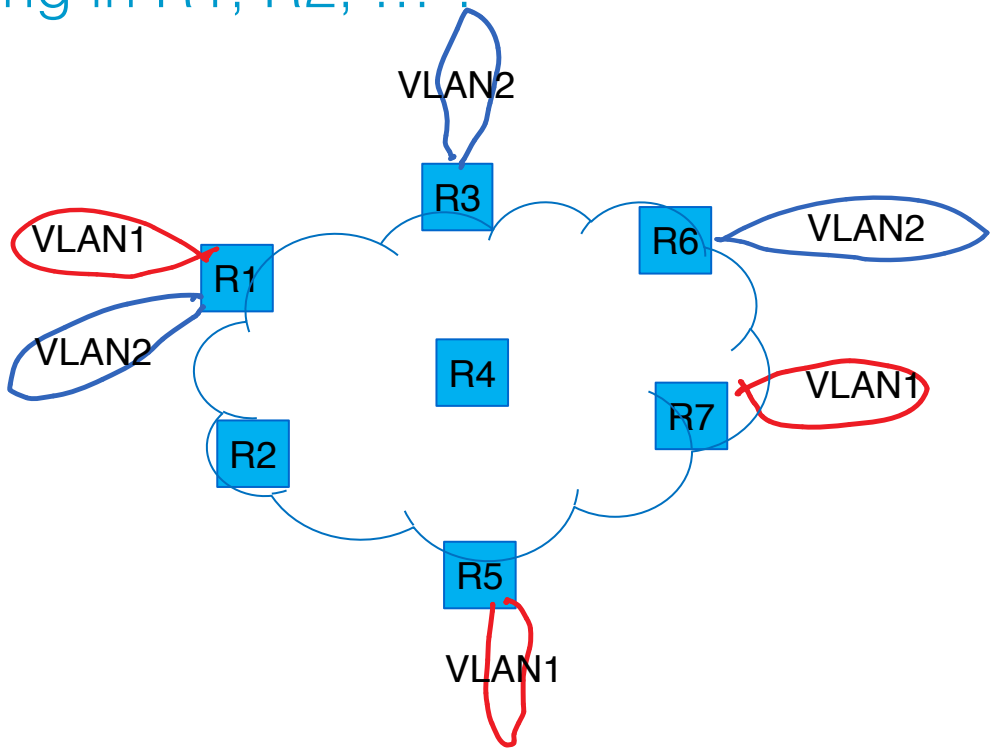
Can you imagine a solution using Link State Routing in R1, R2, … ?

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

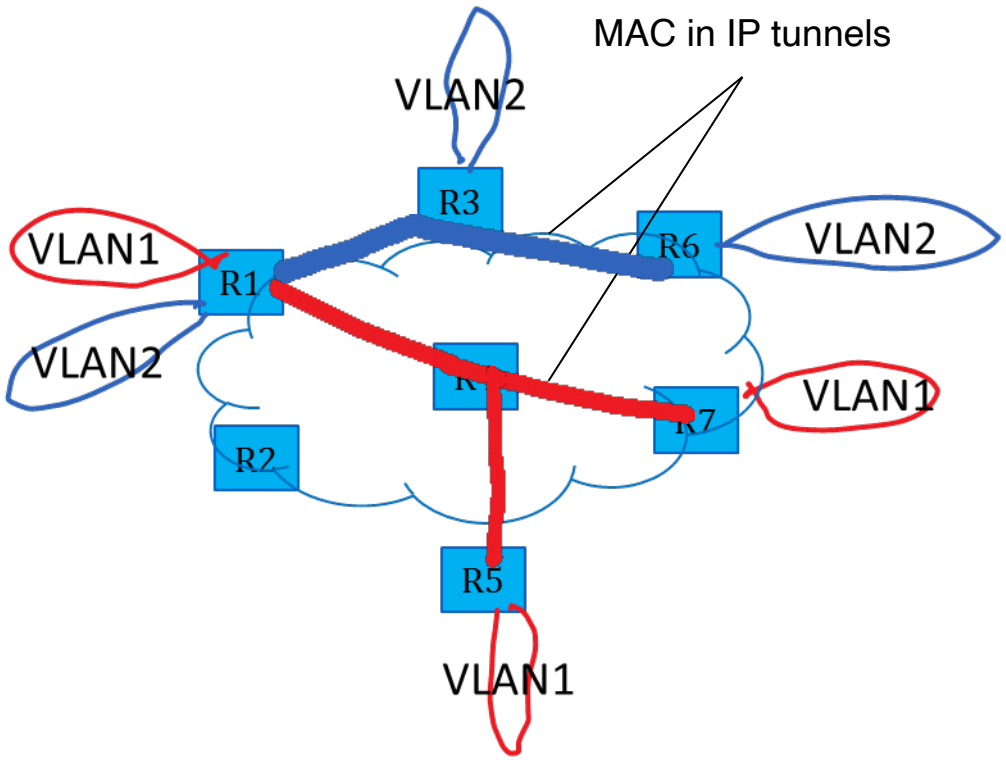


Solution

B does not help since MAC addresses don't say in which VLAN the machine is. Recall that even with VLAN trunk ports we needed the VLAN tag (= info about which VLAN a machine belongs to).

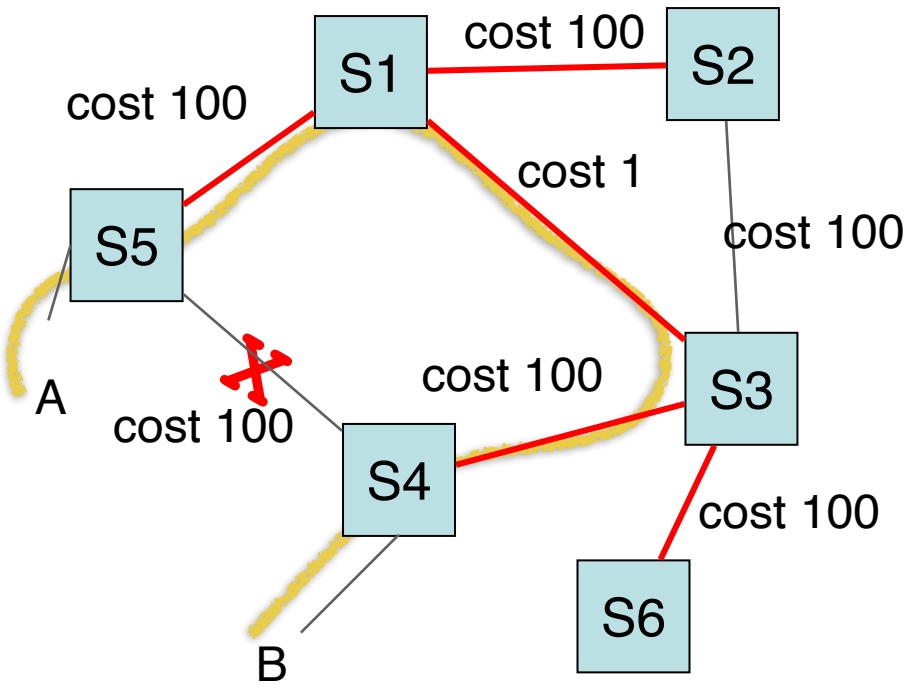
A is a feasible solution: routers can create VLAN tunnels (*MAC in IP !*); and they can use *IP multicast* (and BIER) to avoid excessive packet replication.

- 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.



Do you see any benefit w.r.t. the spanning tree protocol?
See next slides and revisit slide 34 of the MAC lecture

Reminder: a side-effect of the Spanning Tree Protocol (STP)



- All frames go through the spanning tree
- A direct link between two (non-root) switches may not be used even if it is optimal

  e.g., the path from A to B, here, is not optimal

- *Less efficient* than shortest path
- some more sophisticated router-switches implement *LS Bridging instead of STP*

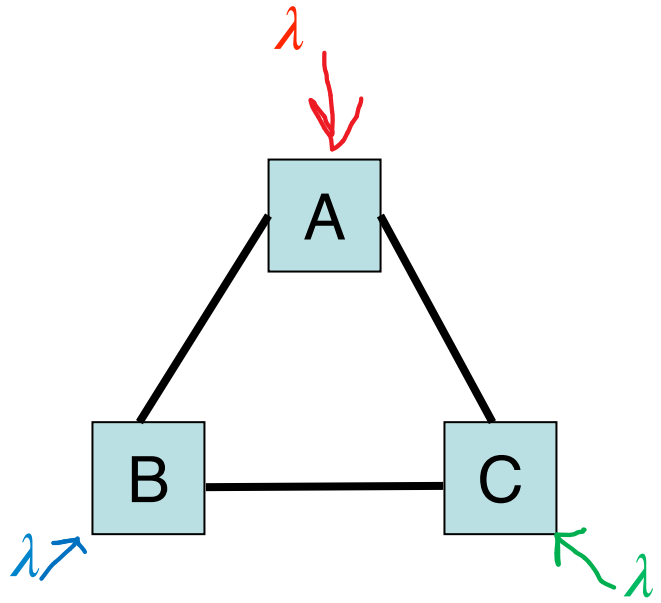
"LS bridging" vs "Classic switches running STP": What is the maximum value of λ we can achieve in each case?

- Node A wants to send a total traffic rate equal to λ 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 the *same* everywhere.
- Possible configurations: OSPF Routers or Switches with STP

Go to web.speakup.info or download speakup app

Join room 87072

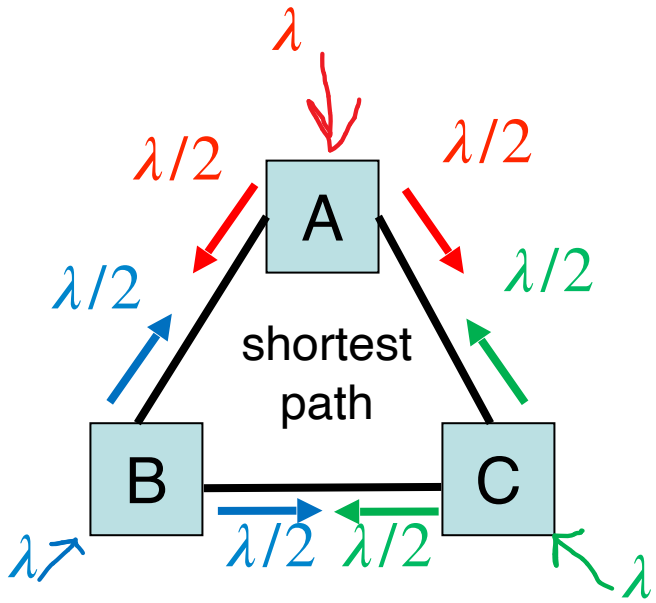
A. With OSPF: $\lambda_{\max} = 2$ Gb/s; with STP $\lambda_{\max} = 2$ Gb/s;
B. With OSPF: $\lambda_{\max} = 2$ Gb/s; with STP $\lambda_{\max} = 1$ Gb/s;
C. With OSPF: $\lambda_{\max} = 1$ Gb/s; with STP $\lambda_{\max} = 2$ Gb/s;
D. With OSPF: $\lambda_{\max} = 1$ Gb/s; with STP $\lambda_{\max} = 1$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



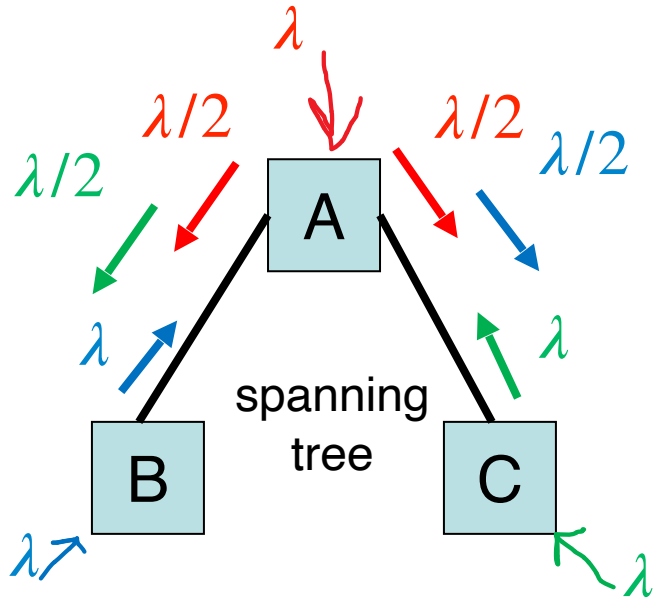
Solution

OSPF routers use shortest paths. Traffic from any node uses the direct link (same is true with any distance vector, which 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 switches, a spanning tree is built, one of the links is disabled. Total traffic on every link and every direction is λ, thus $\lambda_{\max} = 1$ Gb/s.



Answer B.

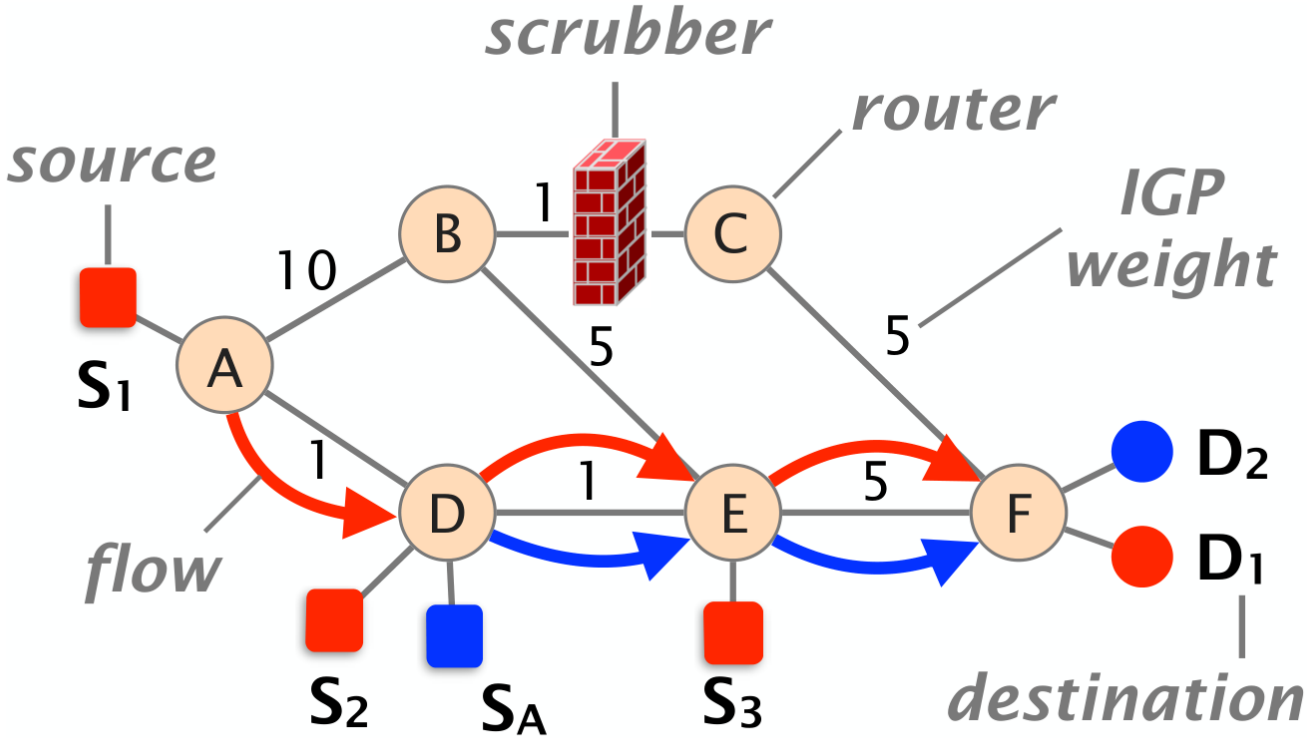
9. Routing in Software Defined Networks (SDNs)

Why it exists? In principle, routers compute best paths using a *distributed* algorithm, populate their forwarding tables, and use dest IP address + longest prefix match to decide where to forward pkts.

Some networks want *more control*; e.g. handle mission-critical traffic with higher priority; ban non-HTTP traffic; send suspicious/DoS traffic to a scrubber for inspection (see figure)

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.



More control… how?

Deep packet inspection (DPI):

routers look not only at IP headers, but also ports and payload
—> classify traffic according to *more* fields

Per-flow forwarding:

when a packet is to be forwarded, the router:

- looks for a match in a flow table = {per-flow forwarding rules with priorities}
- if several rules match, router follows the rule with *highest priority*
- if no rule matches, it goes to the IP forwarding table and does longest prefix match

Same ideas can be used in switches

- per flow tables then complement the MAC forwarding table

At R1: Which way will follow the packets from Lisa and Homer to Enterprise server ?

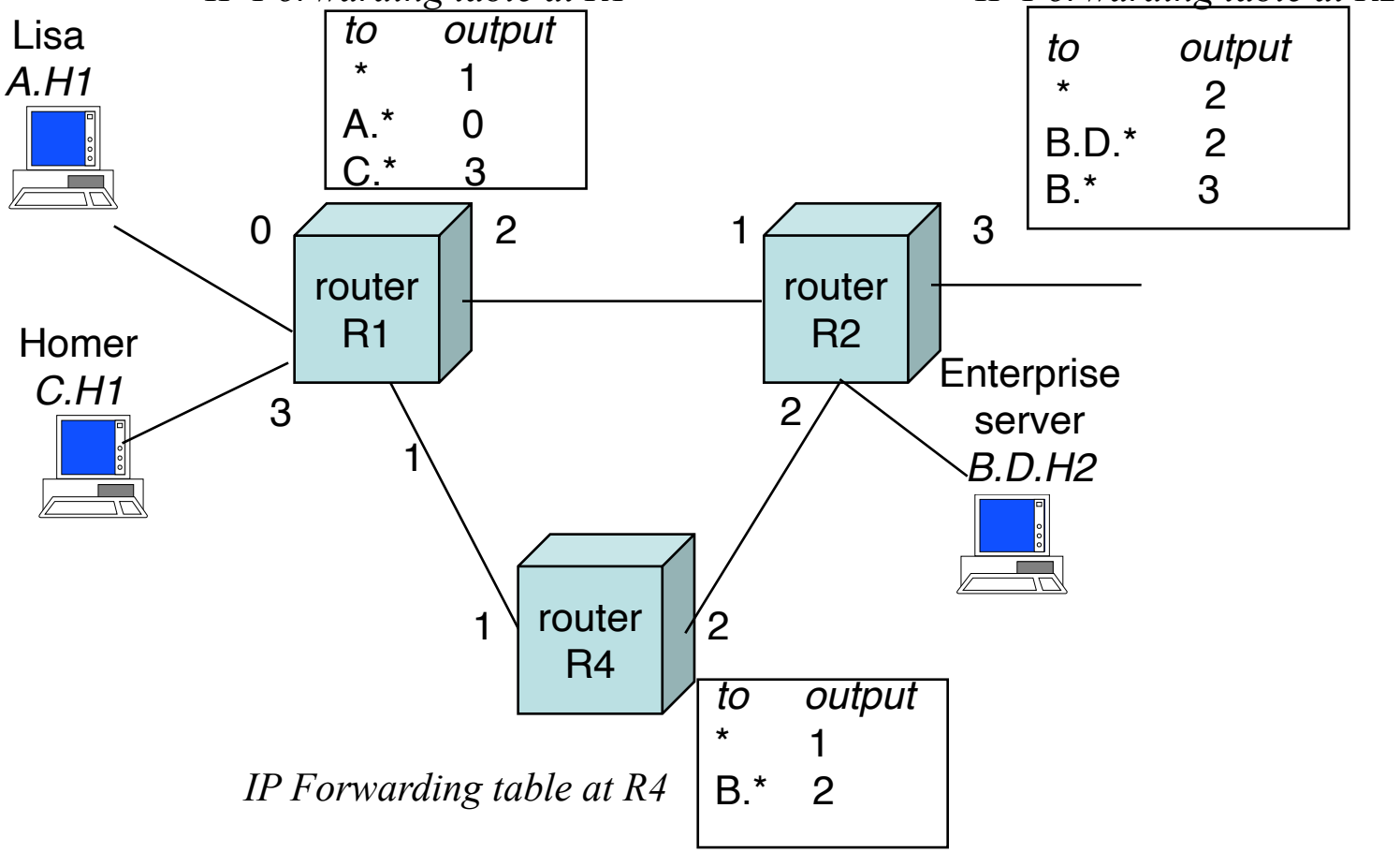
A. Lisa:1, Homer: 1
B. Lisa:1, Homer: 2
C. Lisa:2, Homer: 1
D. Lisa:2, Homer: 2
E. None of above
F. I don't know

Go to web.speakup.info or download speakup app

Join room 87072

Flow table at R1

Flow spec	***action***	***priority***
input=0; dest=B.D*	output 2	50
dest=B.D.*	drop	10



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 in practice

What ?

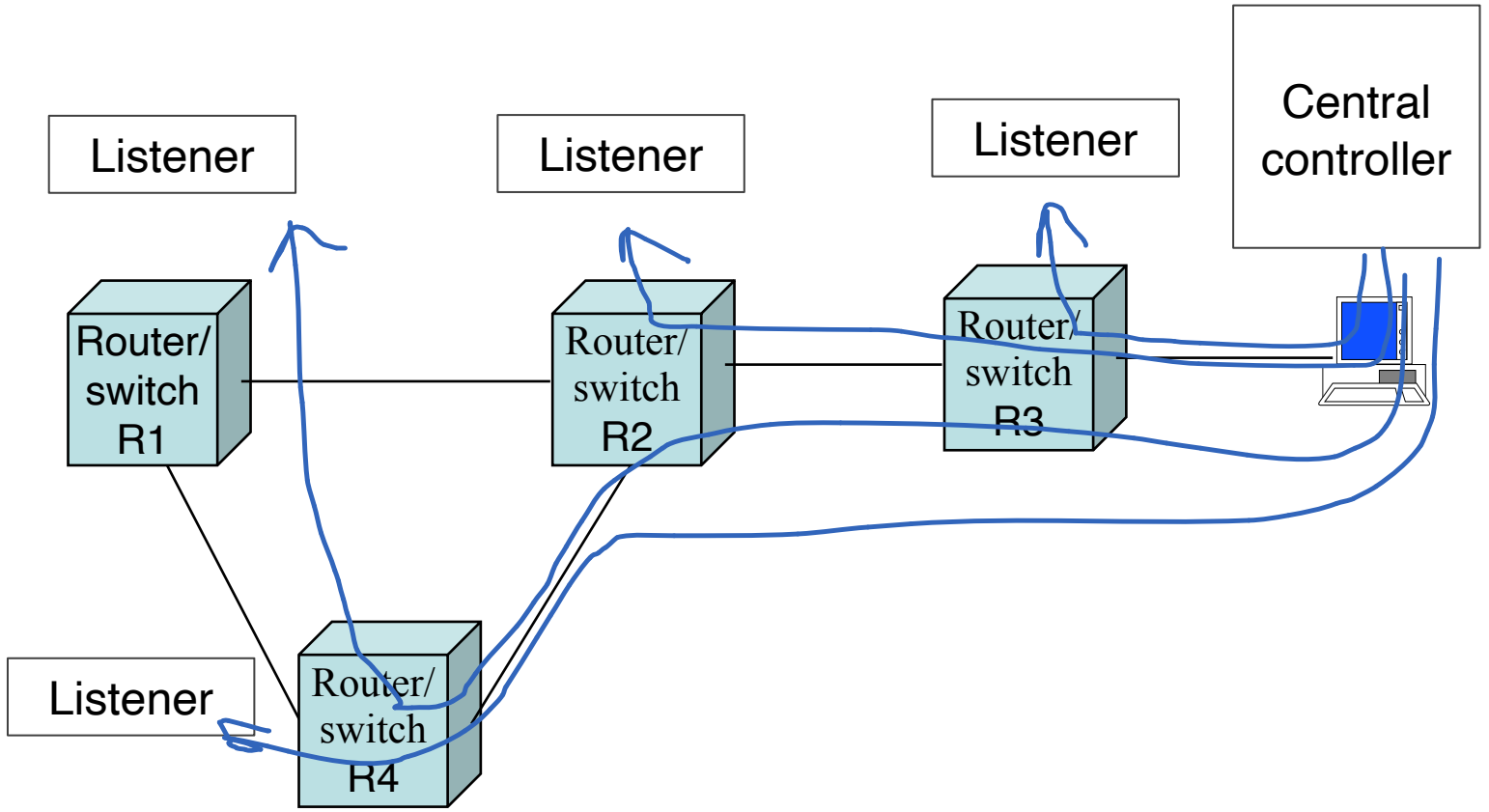
A central *controller* application manages the flow tables in routers and/or switches

How ?

- The central controller decides rules and communicates them to local controllers on routers/switches
- Local controllers, called "*listeners*", write the flow tables on routers or switches
- Protocol between local controller and central controller is e.g. OpenFlow, over TCP connections

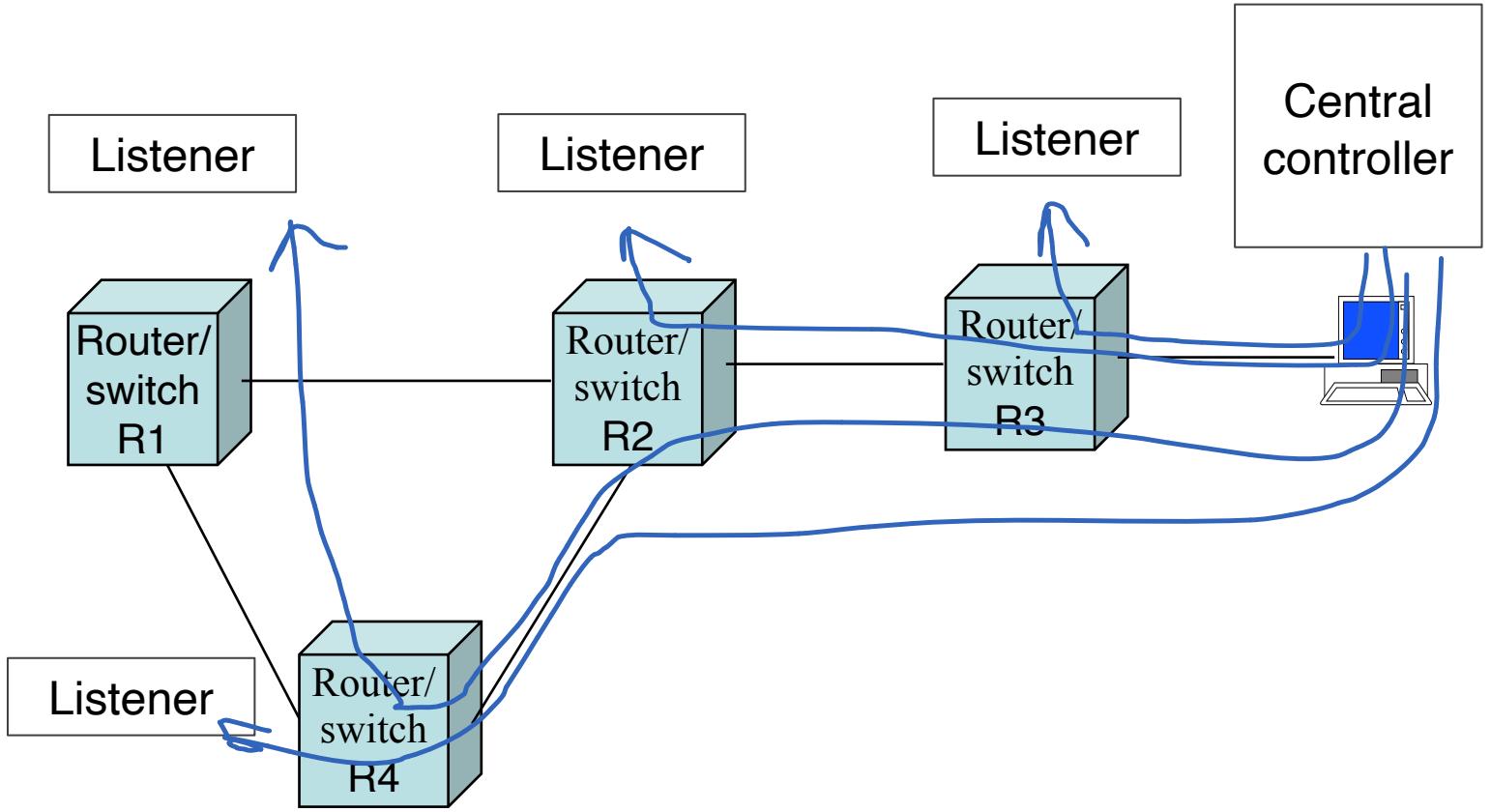
Where ?

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 principle) requires OSPF or some other routing protocol.

But this common-sense observation may not always hold. Remember the *Facebook outage of Oct 4, 2021*, where it seems that routers were disconnected remotely and it was impossible to bring them back in.

Conclusion

Routing protocols automatically build connectivity and repair failures.

With link state routing (like OSPF):

- 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)
- Nonstandard cost definitions are possible; 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) at the expense of losing the robustness of distributed protocols. SDN is used today primarily with switches, but also with routers in some networks.