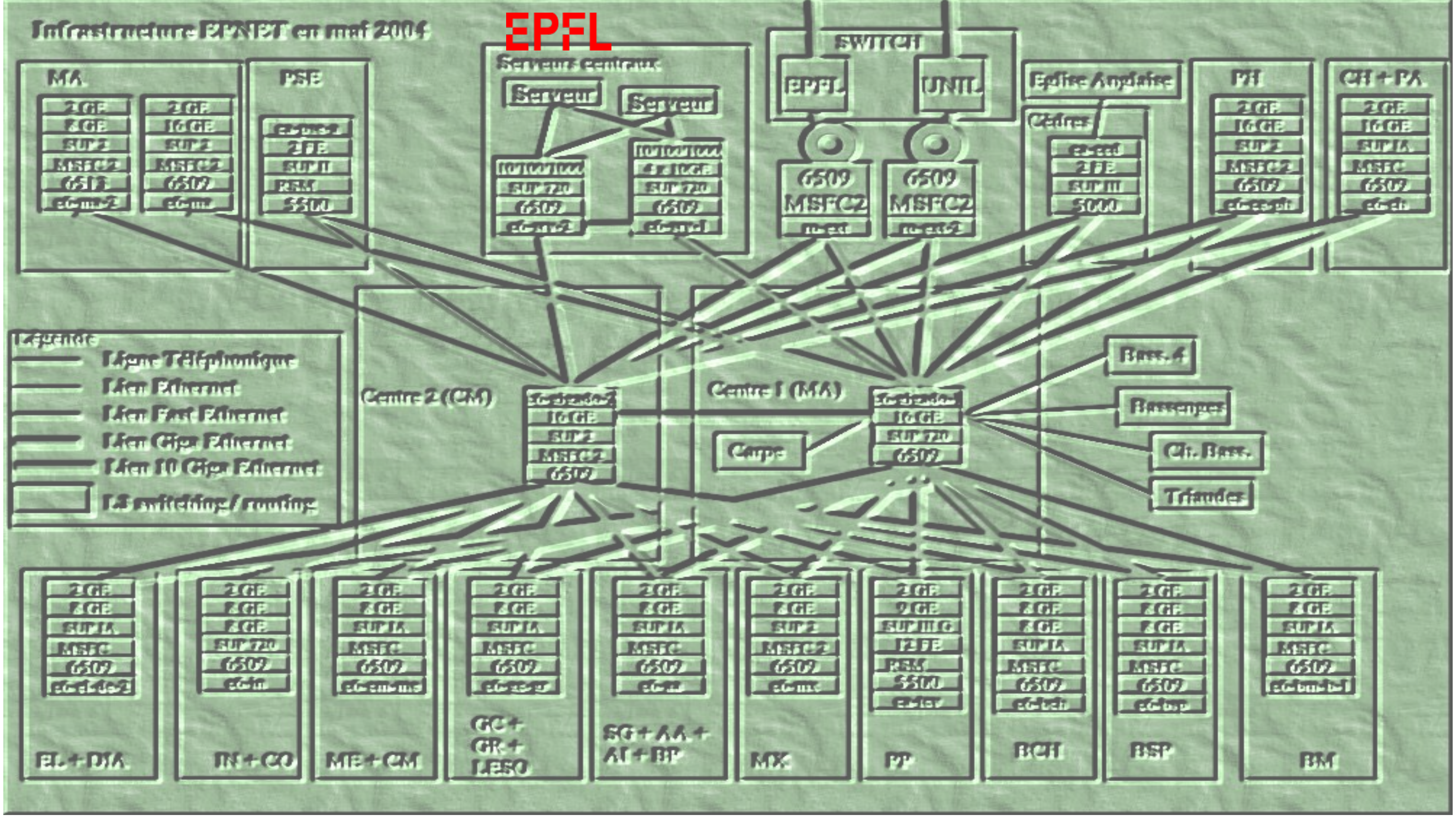


Routing algorithms

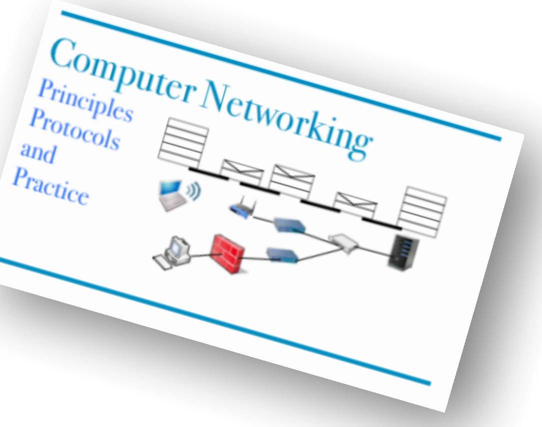
Contents

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?

- Forwarding tables are not easy to set manually (as we do in the lab)
  - in a single domain, it is doable but also time-consuming and error-prone
  - in the WAN, it is very hard to do:
    - multiple domains should coordinate
    - *private* topologies might need to be disclosed to other domains

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

Many routing algorithms, but where do they differ?

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

- to use shortest path
- to use equal-cost multi-path
- to respect policies
- arbitrary

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 (i.e. the routing information) finally stored?

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

Taxonomy of routing protocols

Link State

- Each router maintains a local *topology map* of the entire network
  - obtained by gossiping (= flooding 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
- computes *shortest (min-cost) paths* to each destination prefix based on map
- determines next hop to each prefix and populates its forwarding table

- Typically used for *intra-domain routing* (e.g. OSPF, IS-IS) and advanced bridging methods (e.g. TRILL, SPB Shortest Path Bridging)
- Variants of the optimization objective exist:
  - "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 *neighbors*:
  - i.e., locally-attached networks, neighbor routers,
  - and the *costs* of direct links to these
- then: it ***informs*** its neighbors about the estimated distances to all destinations it knows of (= sends its *distance vector*);

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

Distance Vector example — RIP

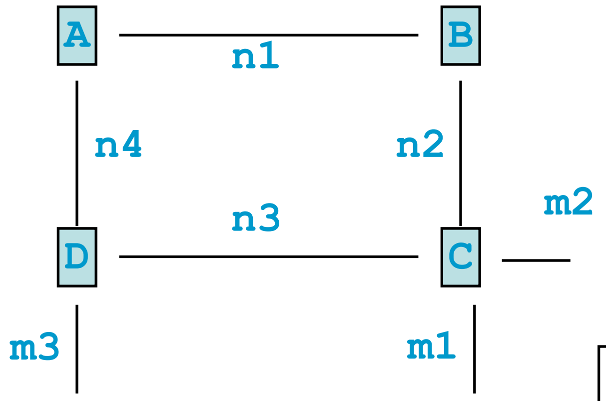
All link costs = 1

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

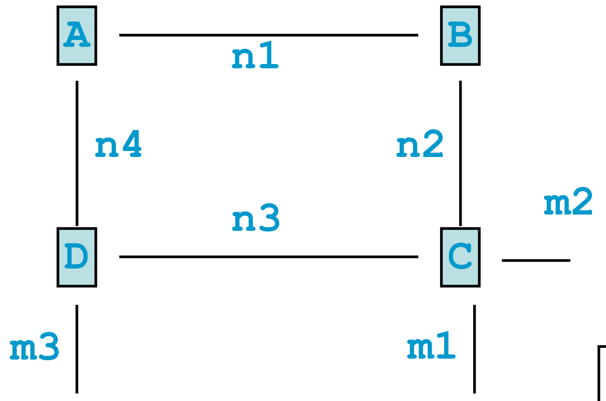
net	dist	nxt
n1	1	n1,A
n4	1	n4,A

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

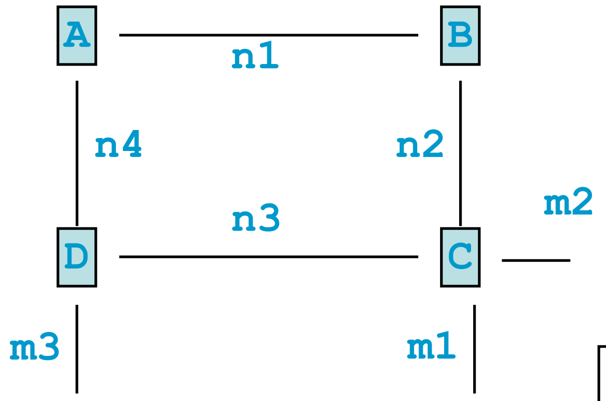
net	dist	nxt
n1	1	n1,A
n4	1	n4,A

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



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

D

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

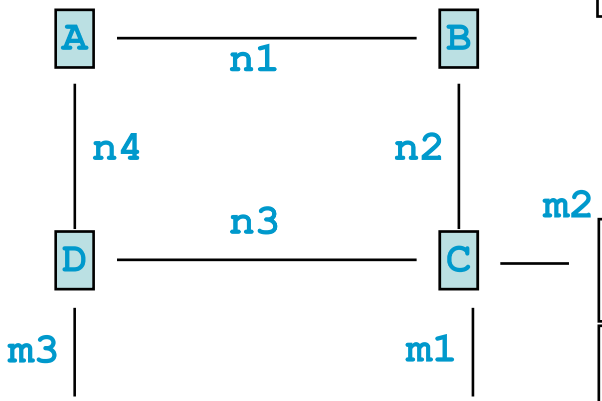
Distance Vector example — RIP: convergence to optimal paths!

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

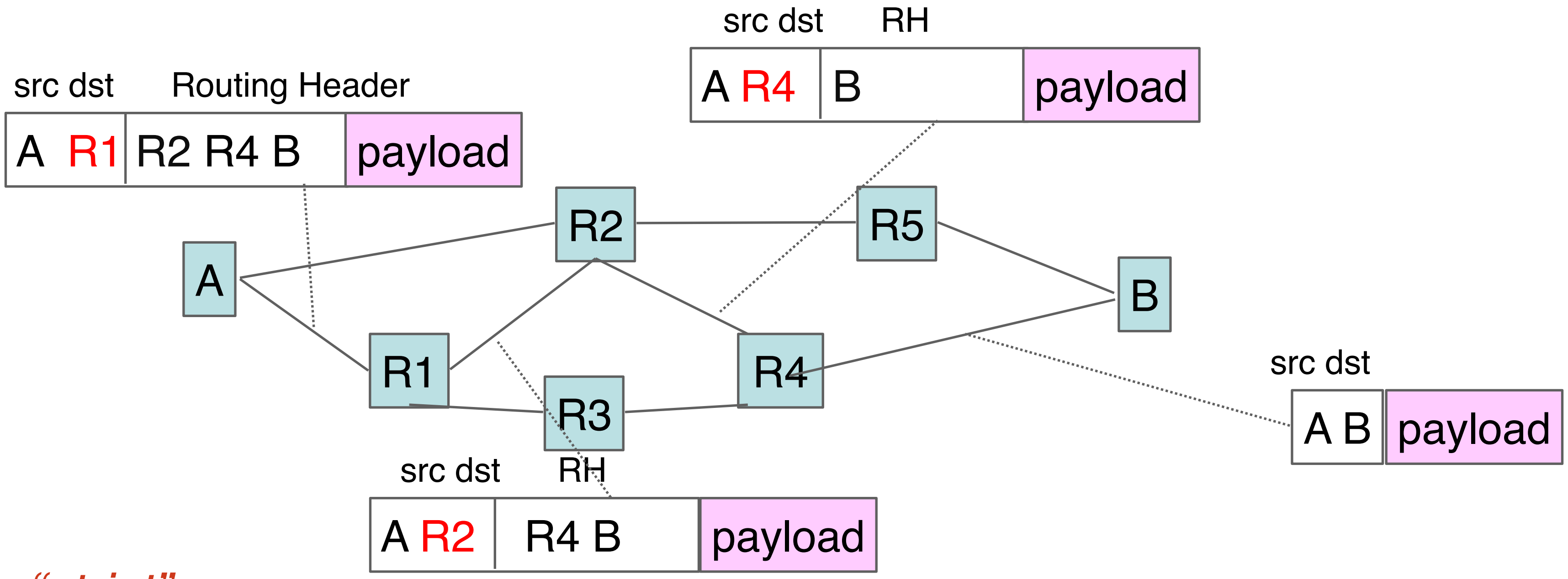
- *Magical*: this message-passing method converges to shortest paths!
- But, it may need *time to converge*
  - hence it is used in *small* networks

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*
- Typically used for *inter-domain routing*, where it is hard to assign costs
  - e.g. BGP computes domain-level paths
  - and each router selects best path according to *multiple criteria*

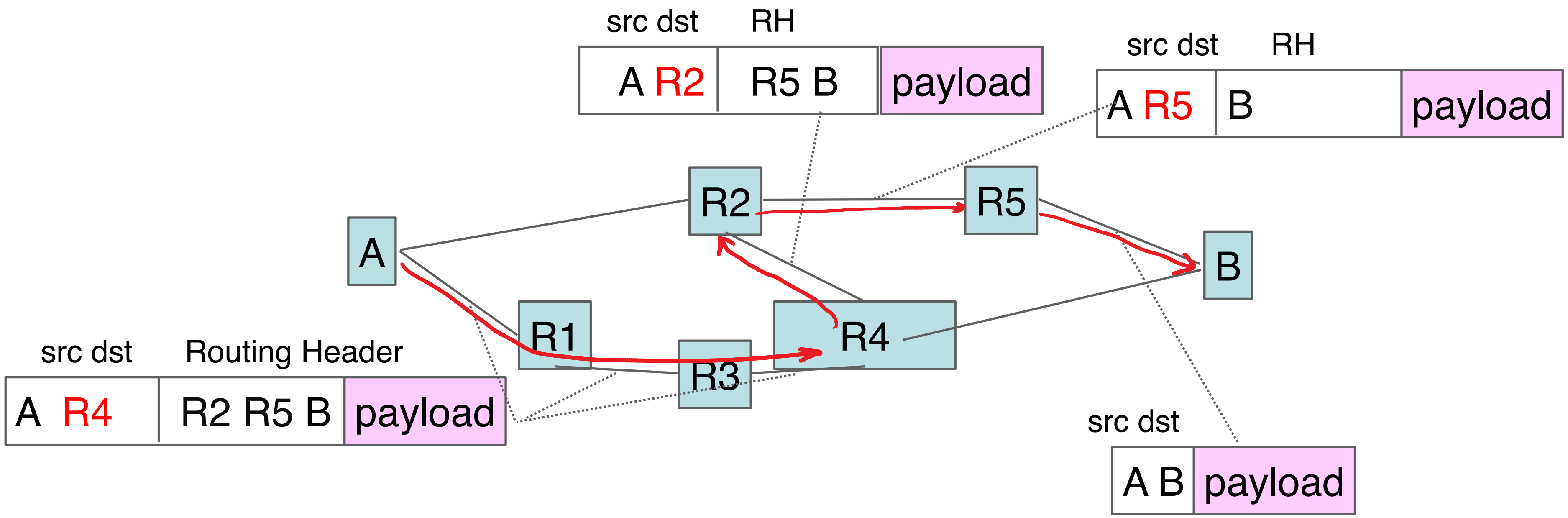
Taxonomy of routing protocols



Source Routing—"strict"

- Source puts explicit paths *into packet headers* (path = sequence of all 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
- Allows fine grained control of traffic (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 notably in data centers

2. OSPF with a Single Area

Link-state routing algorithm

Every router has:

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

Hello protocol is a *ping-style* protocol:

- used to discover neighboring routers
- and to detect failures (e.g. if a neighbor does not respond after 3 times, it is considered "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 an 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

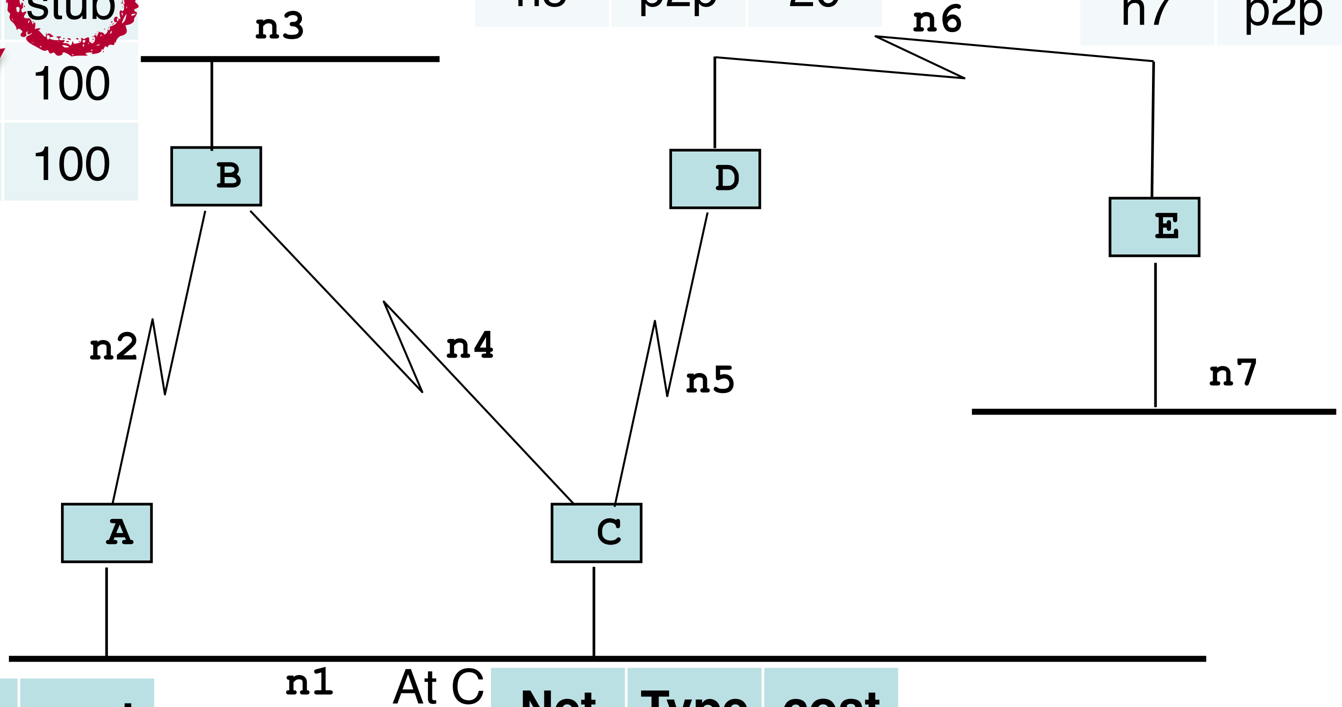
"stub" = non-transit network accessible via a single router

At D

Net	Type	cost
n6	p2p	10
n5	p2p	20

At E

Net	Type	cost
n6	p2p	10
n7	p2p	100



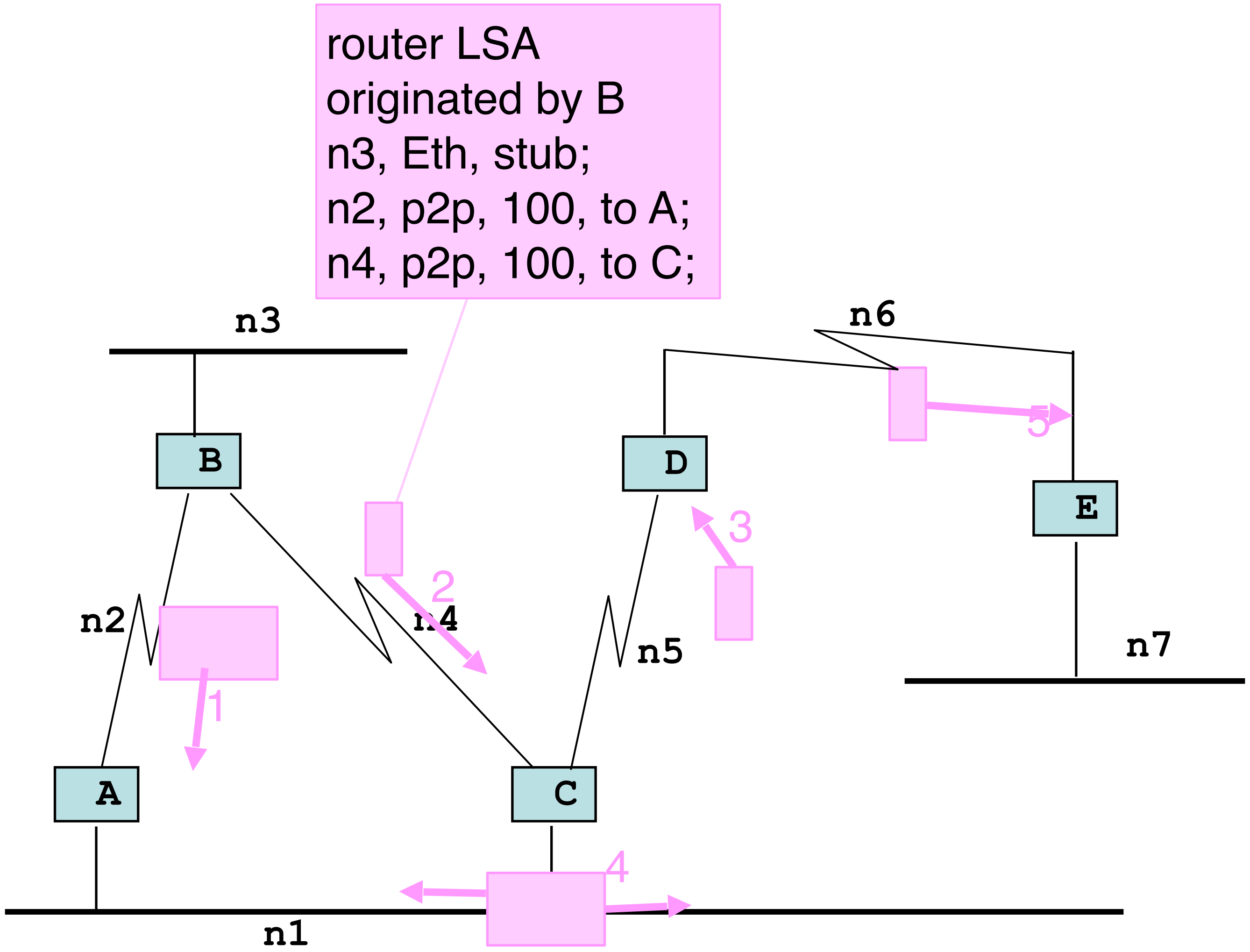
At A

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



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.

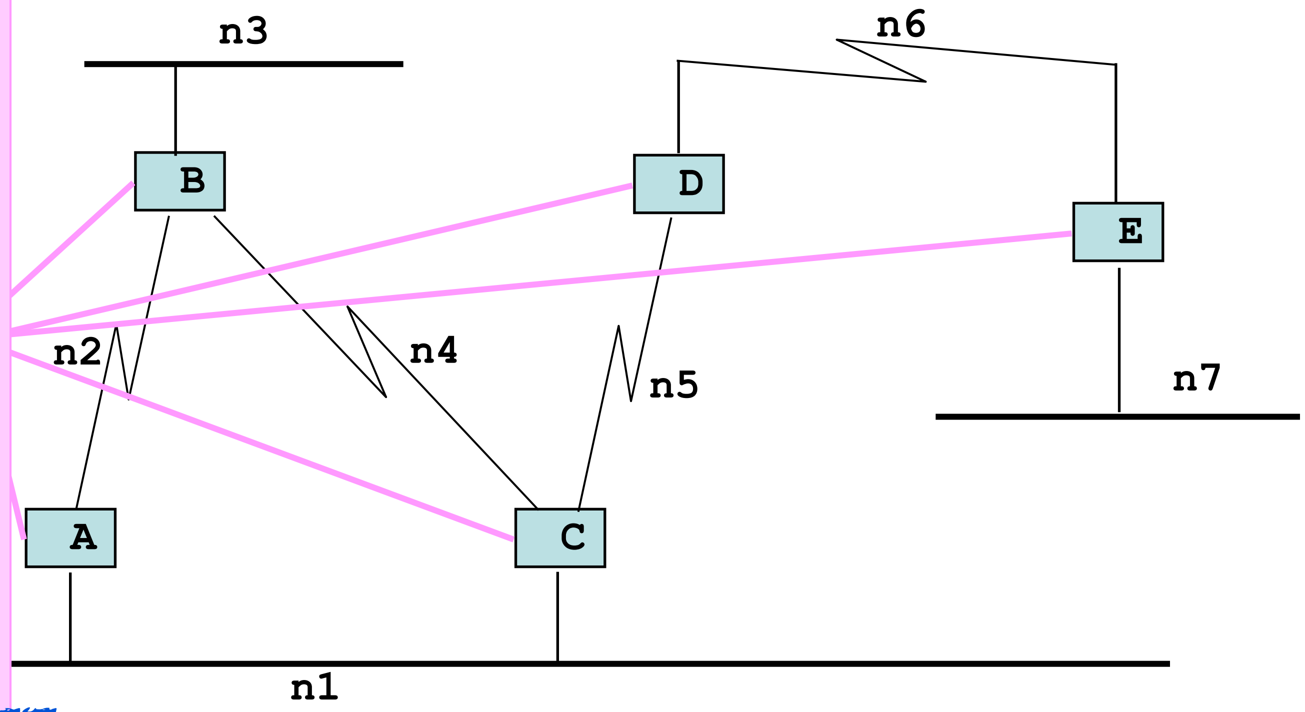
After Flooding

After convergence, all 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).

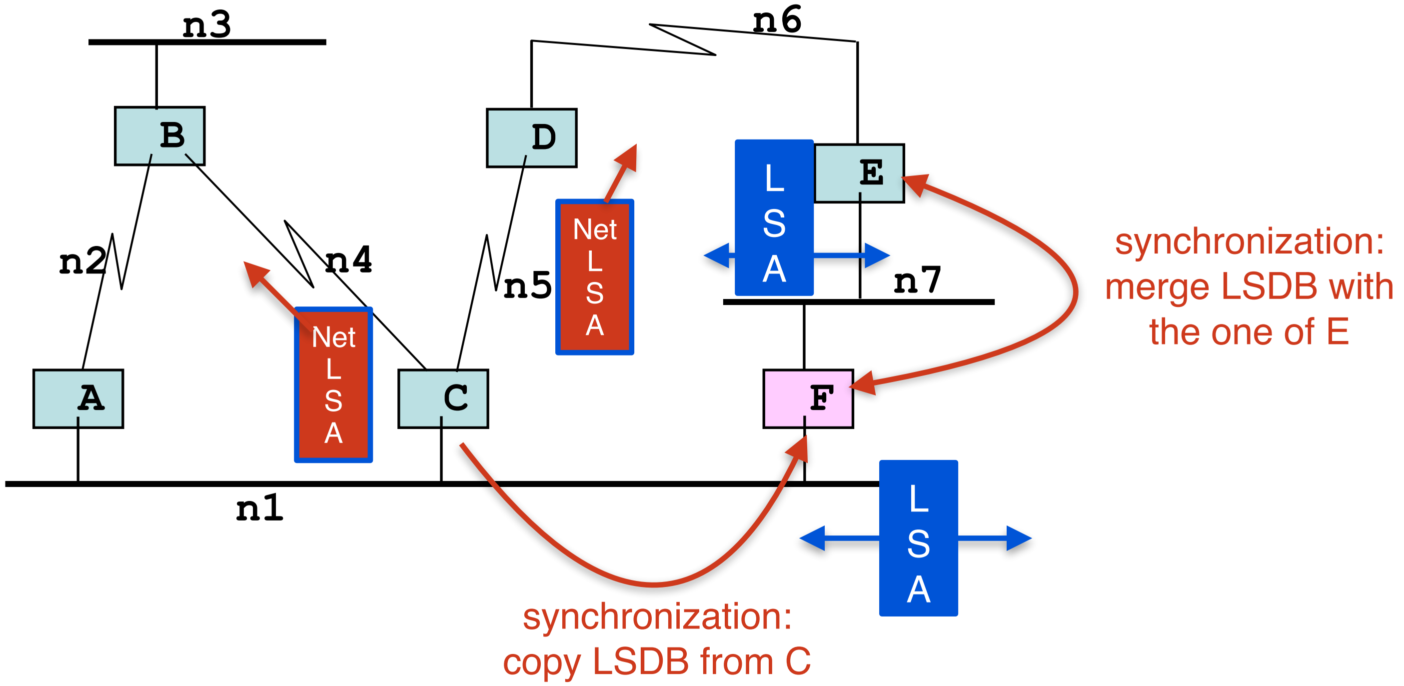
  The designated router "speaks for the switch" and sends a "network LSA" which gives the list of all routers connected to the LAN.

  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 slide 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



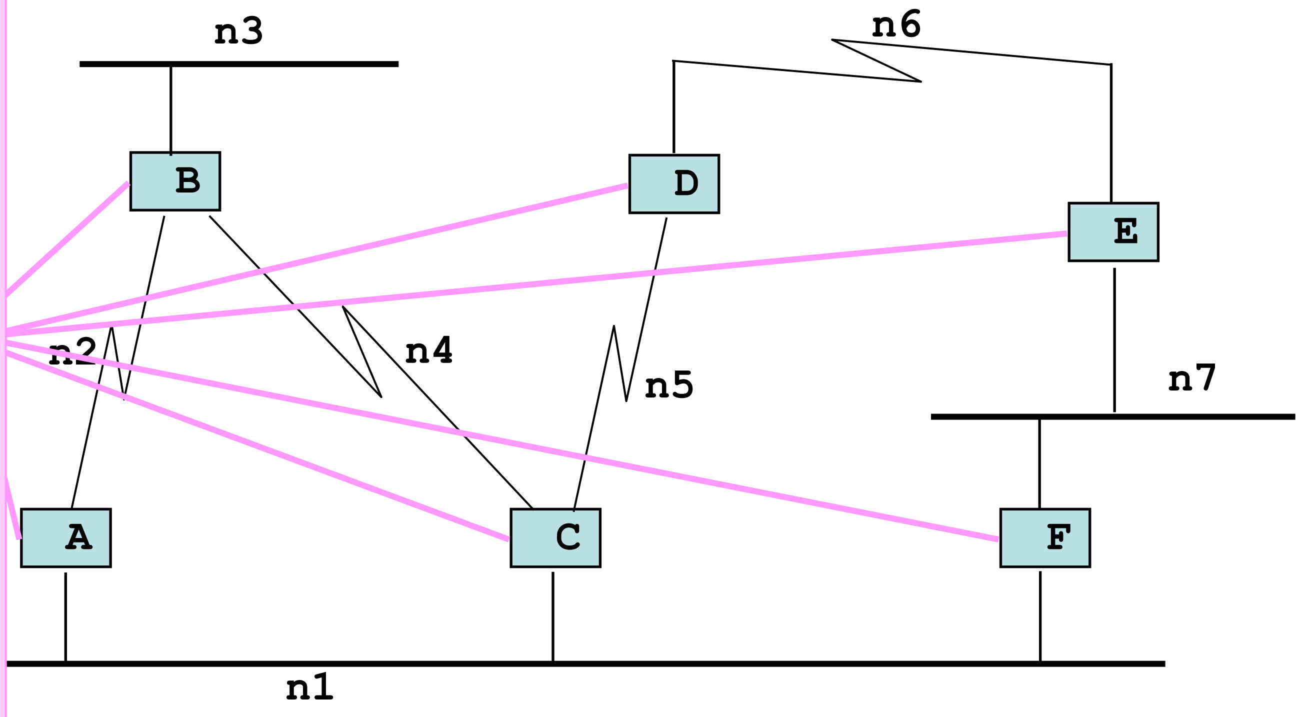
- F discovers neighbors with the hello protocol; assume F discovers C first (C is designated router for n1): F and C establish adjacency (going through a sequence of 8 states, Down to Full). During this process, F and C synchronize their Link State Data Bases (i.e. F copies its LSDB from C).
- When the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

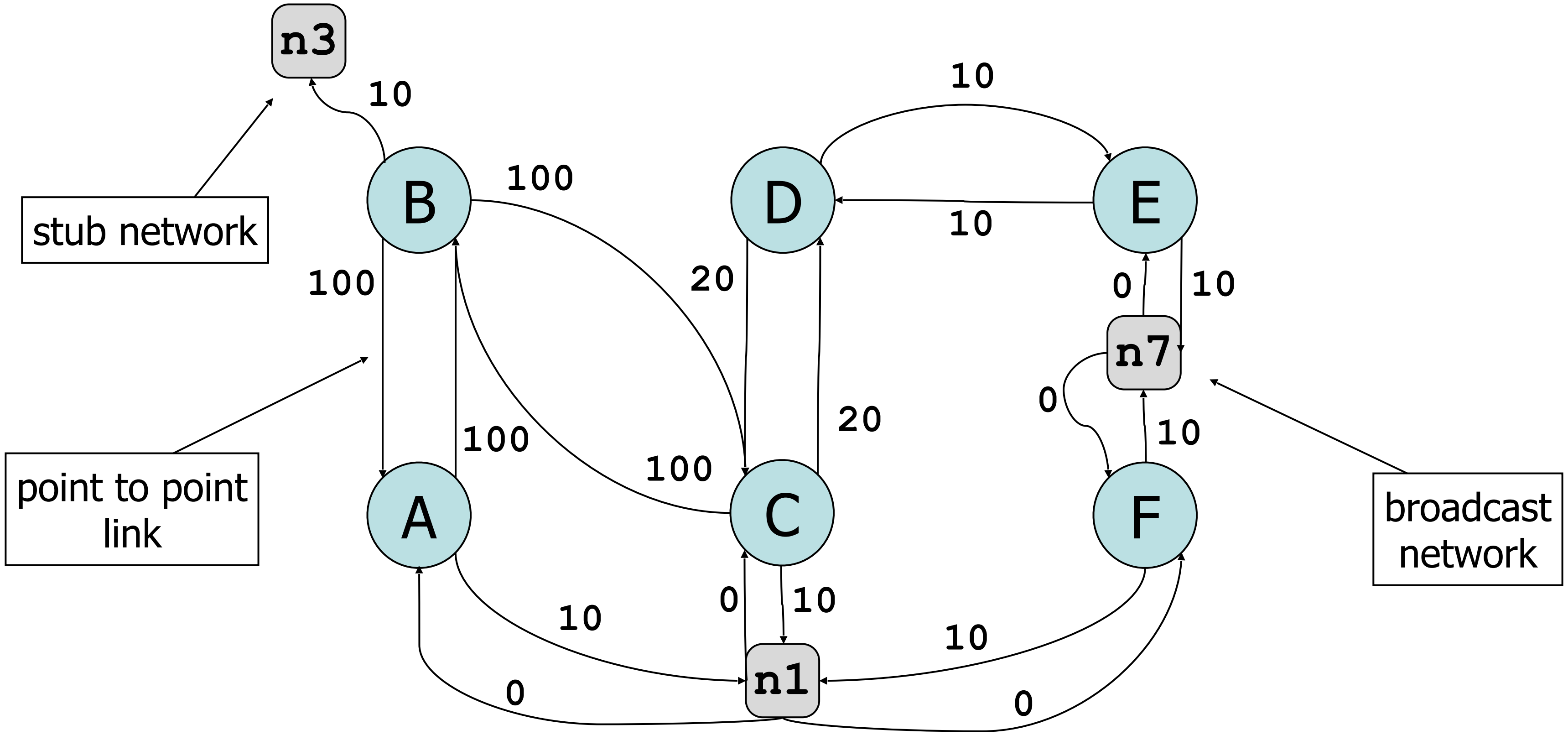
router LSA from B
n3, Eth, stub;
n2, p2p, 100, to A;
n4, p2p, 100, to C
router LSA from A
n2, p2p, 100, to B;
n1, eth, 10, DR=C
router LSA from C
n4, p2p, 100, to B;
n5, p2p, 20, to D;
n1, eth, 10, DR=C
router LSA from F
n7, eth, 10; DR=E
n1, eth, 10, DR=C
router LSA from D
n5, p2p, 20, to C;
n6, p2p,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



Topology graph

- The link state database describes a directed graph, with `outgoing edge cost = cost given in LSA`
- Every *router* or every Ethernet *network* corresponds to a *node* in the graph
- Cost from network node to router node is 0, by default



Practical Aspects

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

Reliable transmission is managed by OSPF with OSPF *acknowledgements* and *timers* (like the *stop and go* protocol).

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 itself
- Paths are computed *independently* at every node
  - link state database (network graph) is same at all routers, but every router performs a different computation, as it computes shortest paths starting from itself
  - synchronization of databases guarantees *no persistent loops* in the graph

Dijkstra's Shortest Path Algorithm

The nodes are 0…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.

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

```
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)
                                pred(j) = {i}
                        else if m(i) + c(i, j) = m(j)
                                m(j) = m(i) + c(i, j)
                                pred(j) = pred(j) ∪ {i}
```

Dijkstra's Shortest Path Algorithm

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

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 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

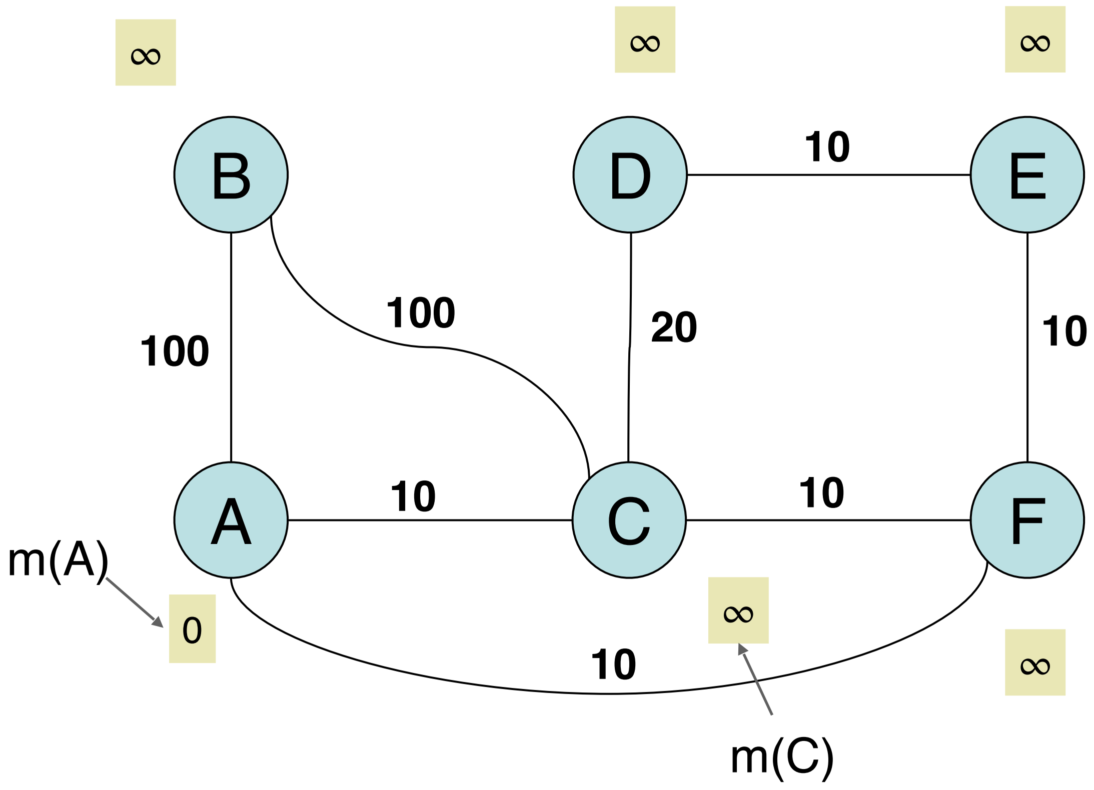
There are multiple versions of Dijkstra's algorithm. The presented version finds all shortest paths, other versions find only one shortest path to every destination. The version presented is very close to what is really implemented in OSPF (with a difference, next-hop versus pred(), see later).

The worst-case complexity of this version is $O\left(N^2\right)$ 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

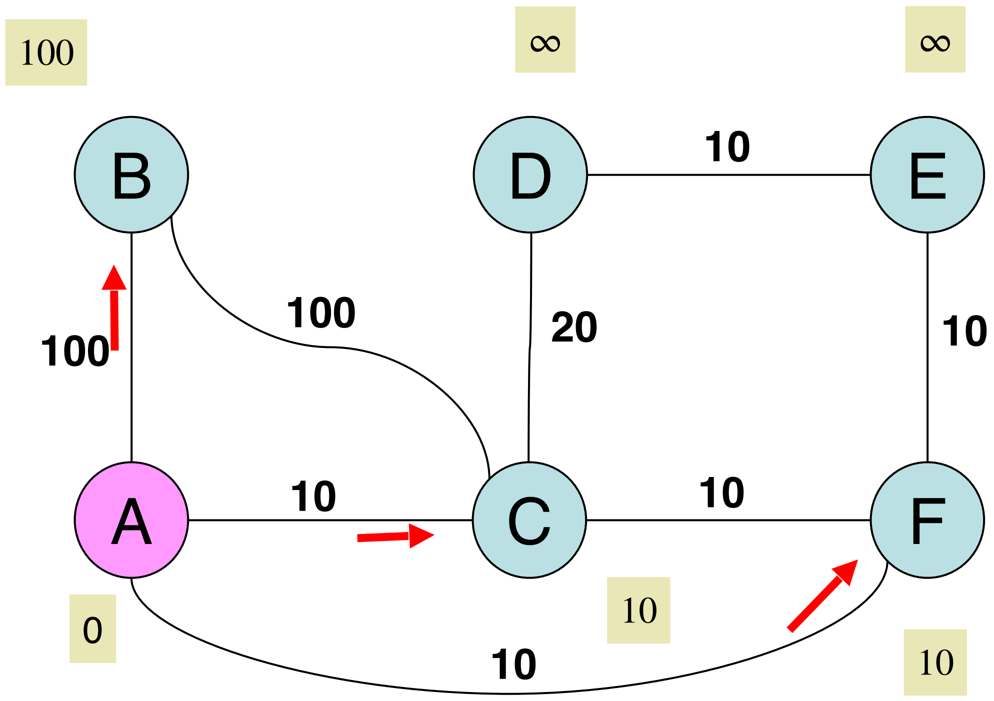


init: $V = \emptyset$

$m(A)=0$
$m(i)=\infty, \; i \neq A$

$pred(i)=\emptyset$

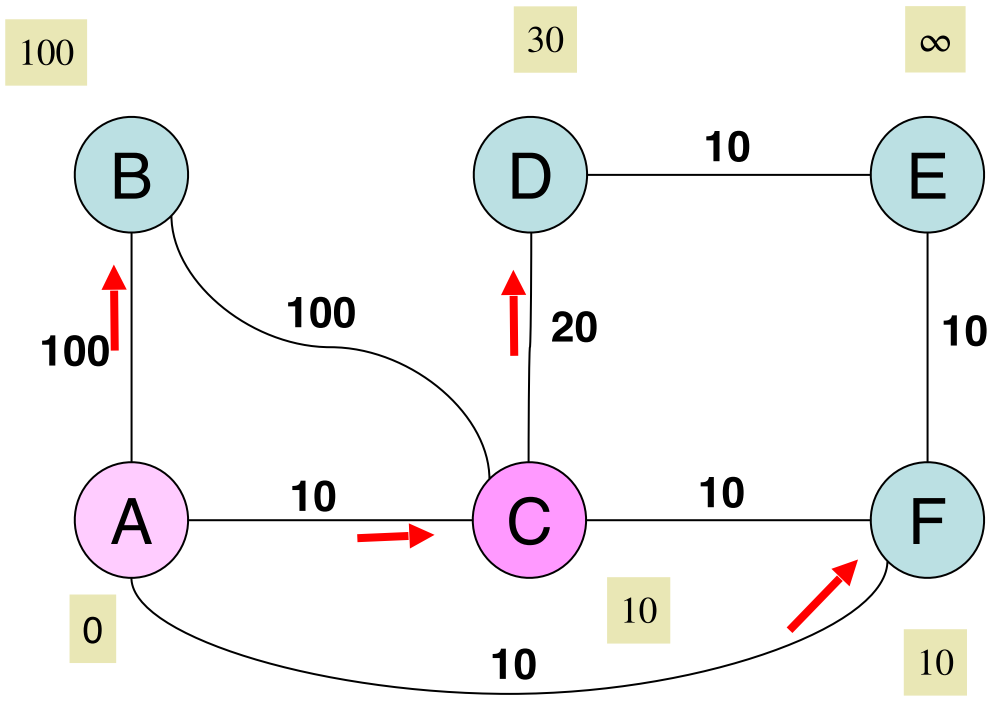
Example: Dijkstra at A
After step 1



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

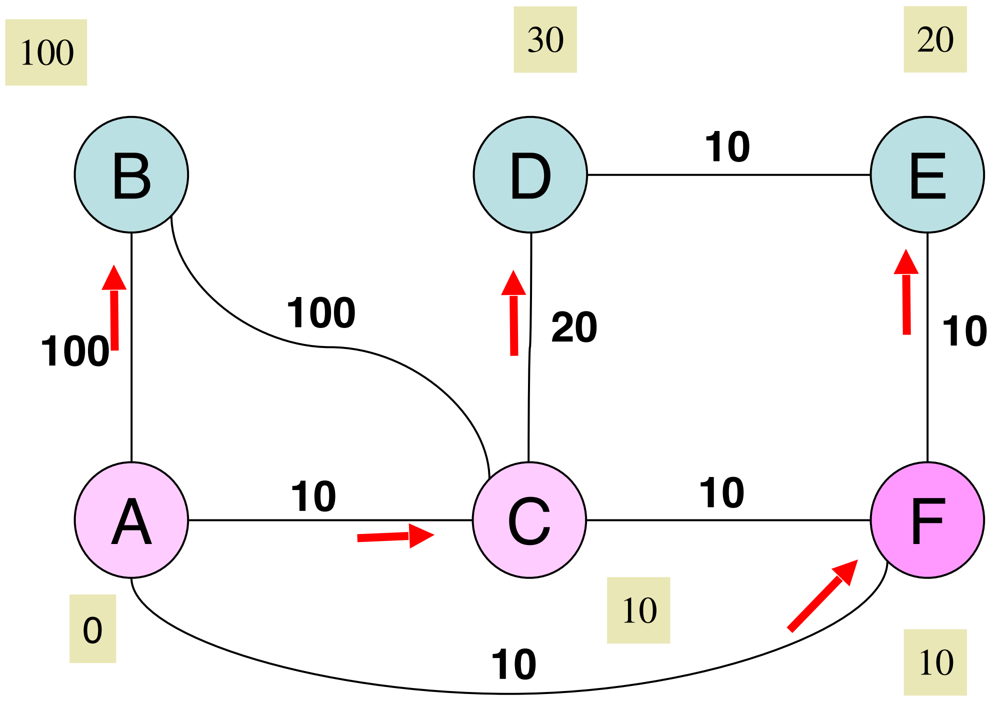
red arrow from A to B means $A \in pred(B)$

Example: Dijkstra at A
After step 2



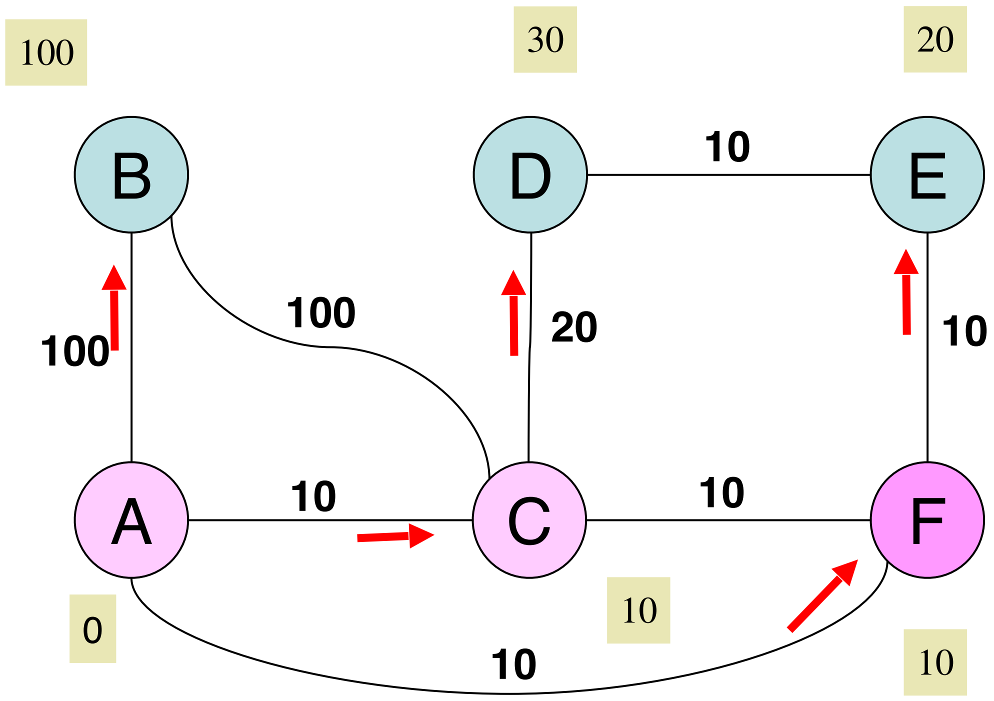
step 2:
i=C
V={A,C}
B, F unchanged
m(D)=30
pred(D)={C}

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?

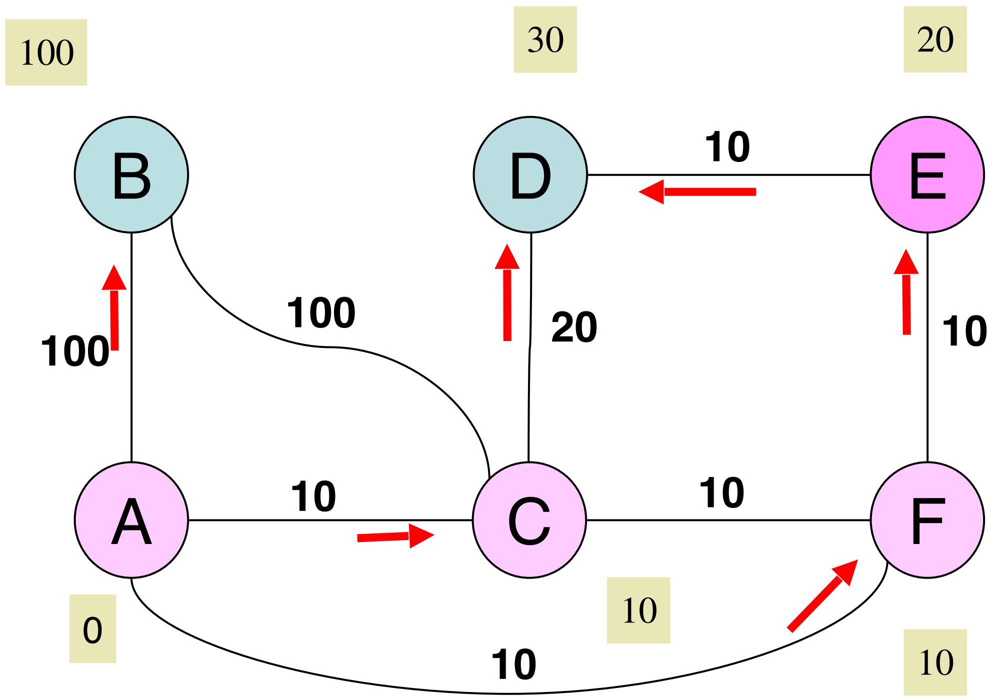


A. B
B. D
C. E
D. I don't know

Go to web.speakup.info or download speakup app

Join room 46045

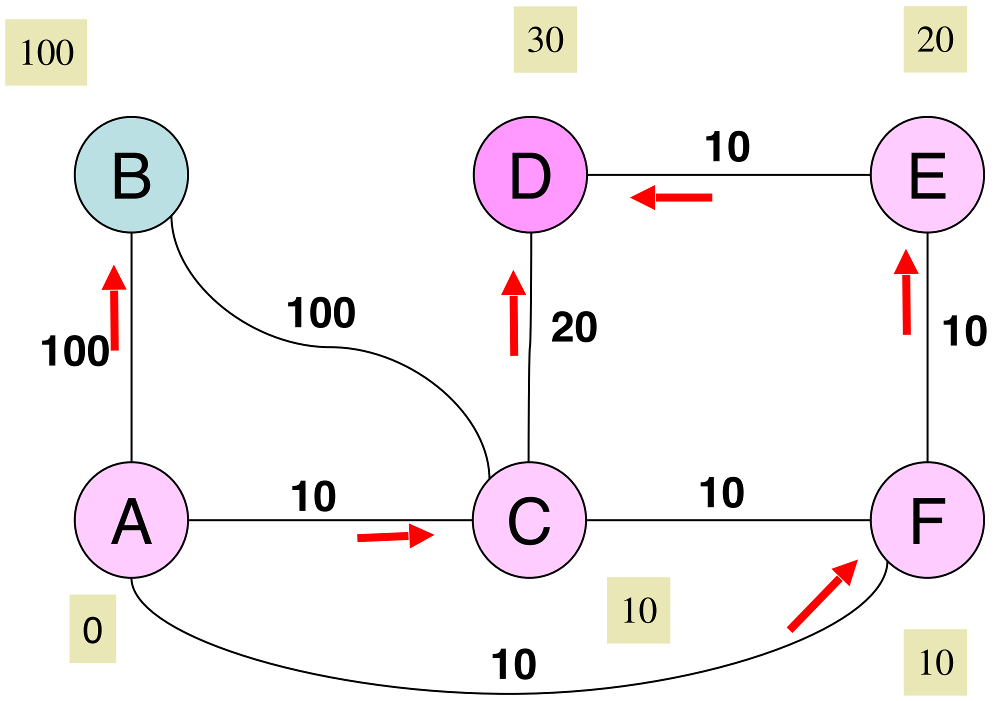
Solution: Dijkstra at A
After step 4



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

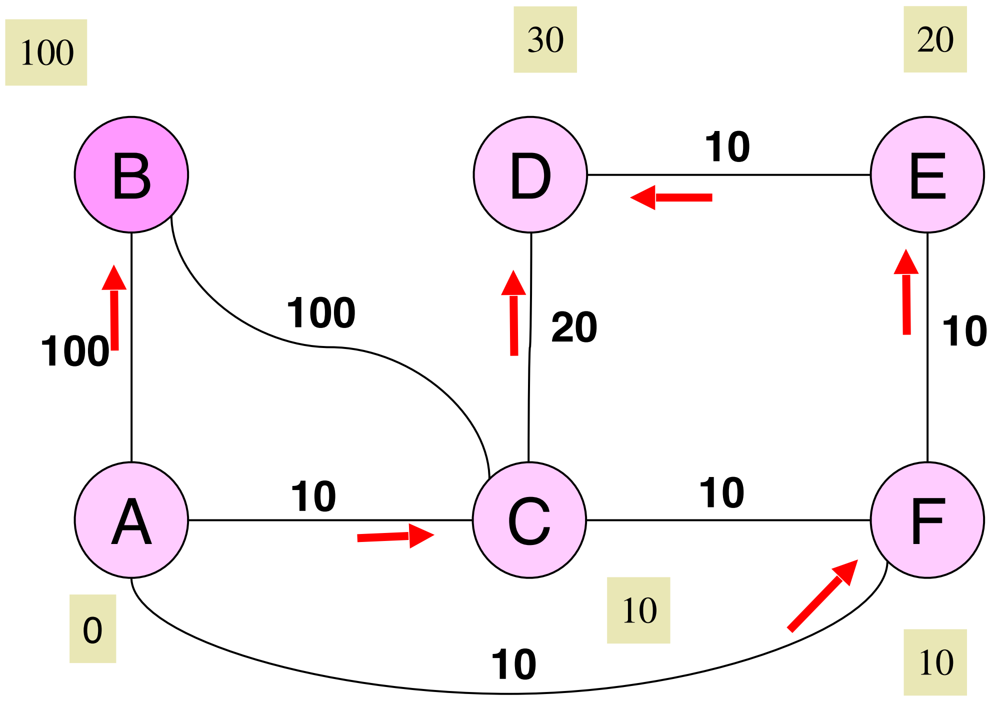
There are two equal-cost paths to D, both are recorded.

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

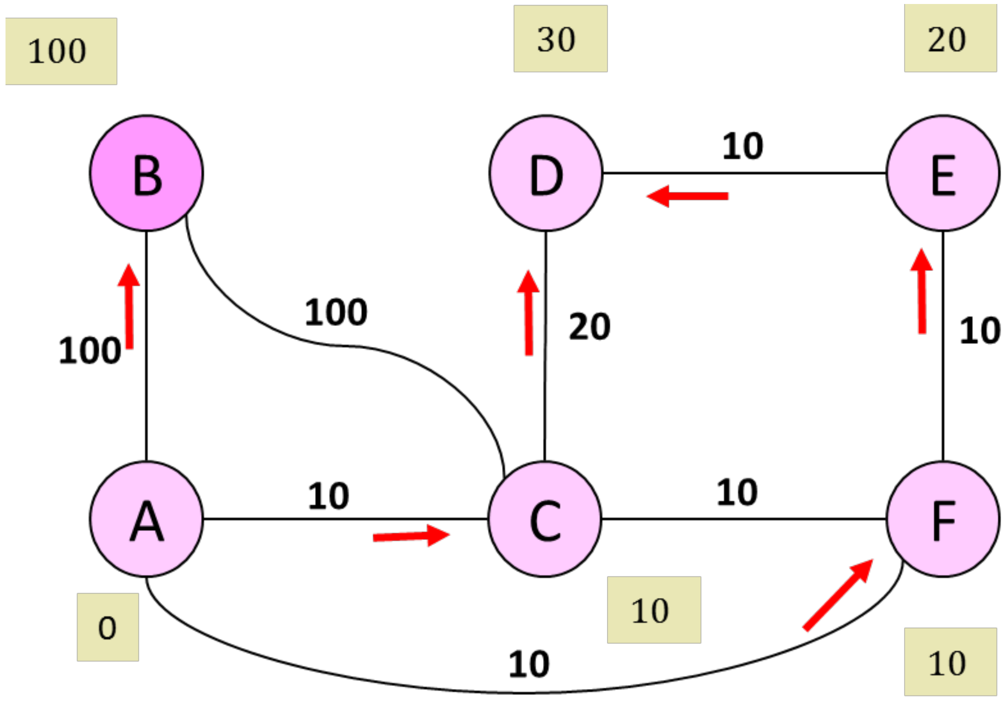
E.g., shortest paths from A to D:

A-C-D

A-F-E-D

to E:

A-F-E



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:

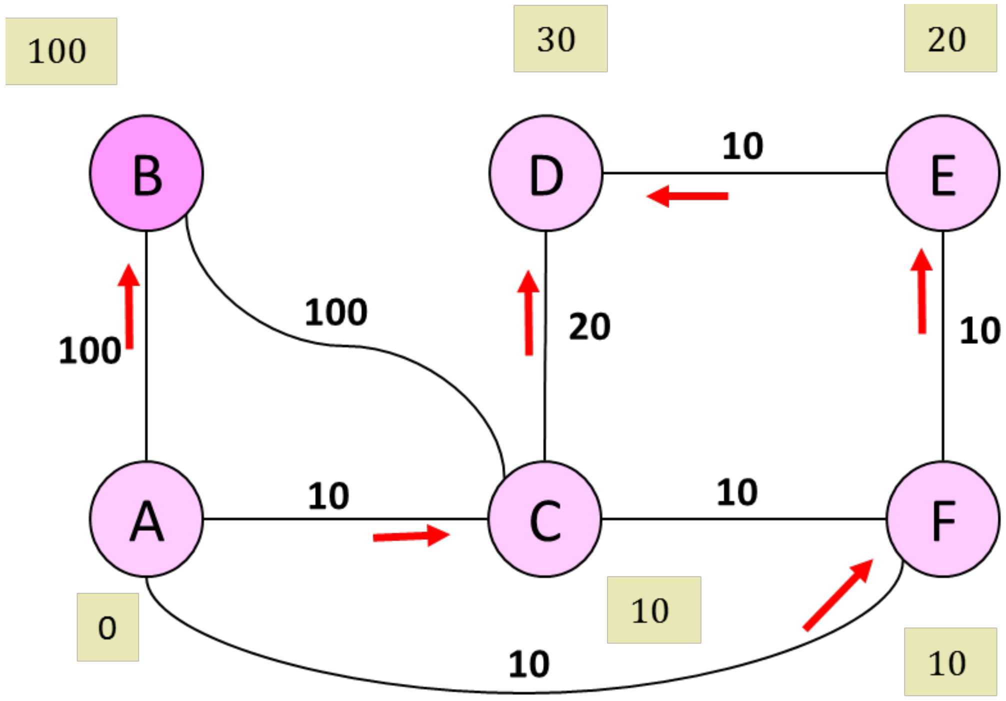
```
m(0) = 0; m(i) = ∞ ∀ i ≠ 0; V = ∅ ; nextHopTo(i) = ∅ ∀i;
for k = 0:N do
        find i ∈ V that minimizes m(i)
        if m(i) is finite
                add i to V
                for all neighbors j ∈ V of i
                        if m(i) + c(i, j) < m(j)
                                m(j) = 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
```

Forwarding Table

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

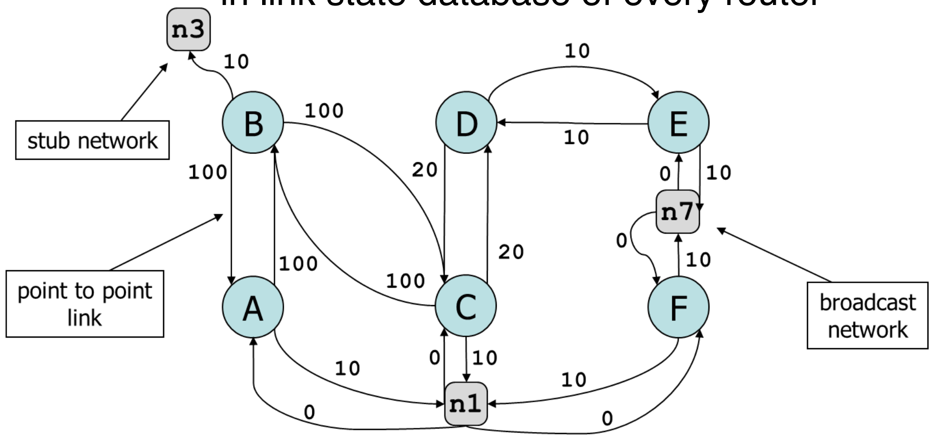
At A

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



Bringing back the stub network nodes

in link state database of every router



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 *larger*.

To optimize the computation:

- *stub networks are removed* before applying Dijkstra;
- then, Dijkstra is run and the routing table contains costs and next hop to edge routers such as B;
- then, stub networks such as n3 are *added to the forwarding 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 forwarding table
- This is good as it allows to exploit the redundancy 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 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

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

It is better to use all available paths network (load balancing) $\Rightarrow$ 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. The definition of a flow depends on the system: a flow is identified by the source and destination addresses and, in some systems, by next header type and (if they exist), source and destination ports.

Per-flow load balancing is implemented by applying a *hash function* $h: m \mapsto h(m) \in [0,1]$ 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* (tuple 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.

Answers A and D !