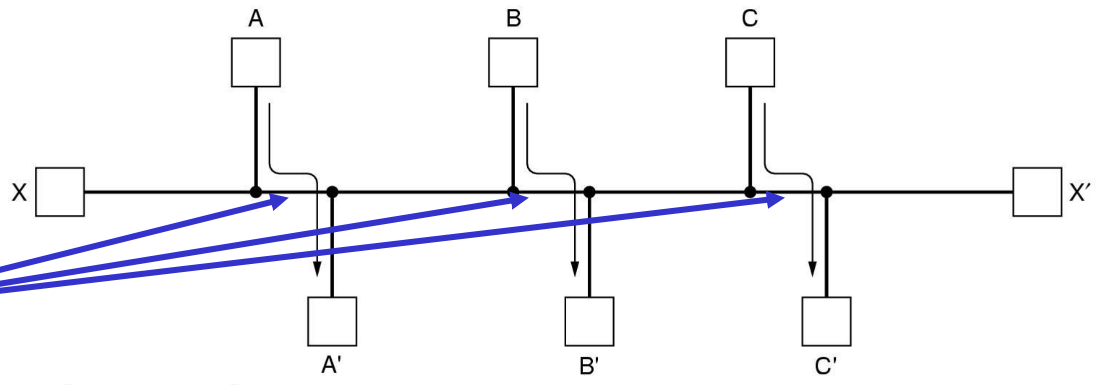
Network Layer: Routing Algorithms

- routing algorithm = logic a router uses to decide, for each incoming Packet, which output link the Packet should be transmitted on
  - ***datagram packet-switching:*** this decision must be made for every Packet
  - ***virtual circuit packet-switching:*** routing decisions made only at v.c. set-up
- desirable properties of a routing algorithm:
  - **correctness**, **simplicity**, **efficiency** – obviously
  - **robustness** – since usually the entire network can't be "re-booted" !!!
  - **stability** – routing algorithm reaches equilibrium in a reasonable time
  - **fairness**, **optimality**

  – often in conflict, e.g.

link capacities fully utilised by A-A', B-B', C-C' traffic respectively



- optimality – with respect to what ? What are we trying to optimise ?!
  - average Packet delay ? total Packet throughput ?
    - but these goals are also in conflict: operating any network near capacity implies long queueing delays in node buffers
  - compromise – minimise number of relays (or ***hops***) a Packet needs

Network Layer: Routing Algorithms (cont.)

- ***least-cost routing:***
  - a value is assigned to each link in the network: this is the *cost* of using this link
  - the cost of a route is the combination of the values of its links
  - the best route is the one with the lowest cost ⇒ know how to relay incoming Packets
- cost assigned to a link could be:
  - 1 for each link – finds route with the *fewest hops*
  - (financial) cost of using the link – finds *cheapest* route
  - packet delay on the link – finds *minimum-delay* route
  - packet transmission time on the link – finds *maximum-bandwidth* route
  - …or some combination of these, or other factors !
- routing algorithms can be divided into 2 classes:
  - ***nonadaptive***, or ***static*** – routing decisions are pre-determined and not based on measurements (or estimates) of the current network topology and traffic load
  - ***adaptive*** – routing decisions may be changed when network topology and/or traffic load changes
    - extreme case: select a new route for each Packet
    - may get information just from neighbouring routers, or from all routers
    - may re-determine routes periodically, or when topology changes, or when traffic load changes more than a threshold percentage, or…

Network Layer: Routing Algorithms (cont.)

- once the "cost" of each link is known, the routers can run a routing algorithm to determine the best routes for each possible sender-receiver transmission

- in practice: routing algorithm should be ***adaptive*** and ***de-centralised***

- the 2 most common routing algorithms are <u>**distance-vector**</u> and <u>**link-state**</u>

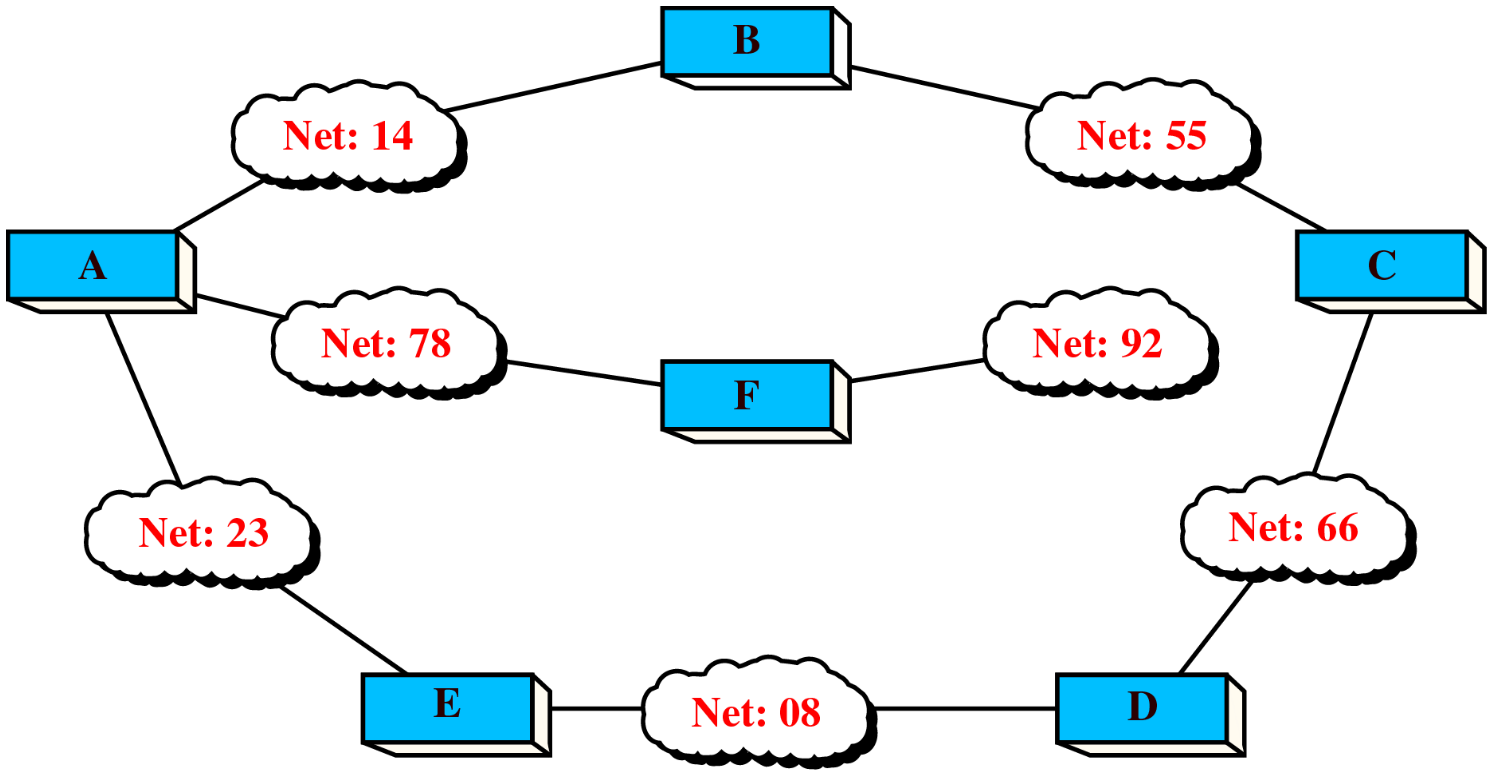
    - distance-vector: *each router exchanges information about the entire network with neighbouring routers at regular intervals*
        - neighbouring routers = connected by a direct link (e.g. a LAN)
        - regular intervals: e.g. every 30 seconds

    - link-state: *each router exchanges information about its neighbourhood with all routers in the network when there is a change*
        - neighbourhood of a router = set of neighbour routers for this router
        - each router's neighbourhood information is ***flooded*** through the network
        - change: e.g. if a neighbouring router does not reply to a status message

- link-state ***converges*** faster in practice, so more widely used
    - converges = determines optimal routes, given a particular network topology

Network Layer: Distance-Vector routing

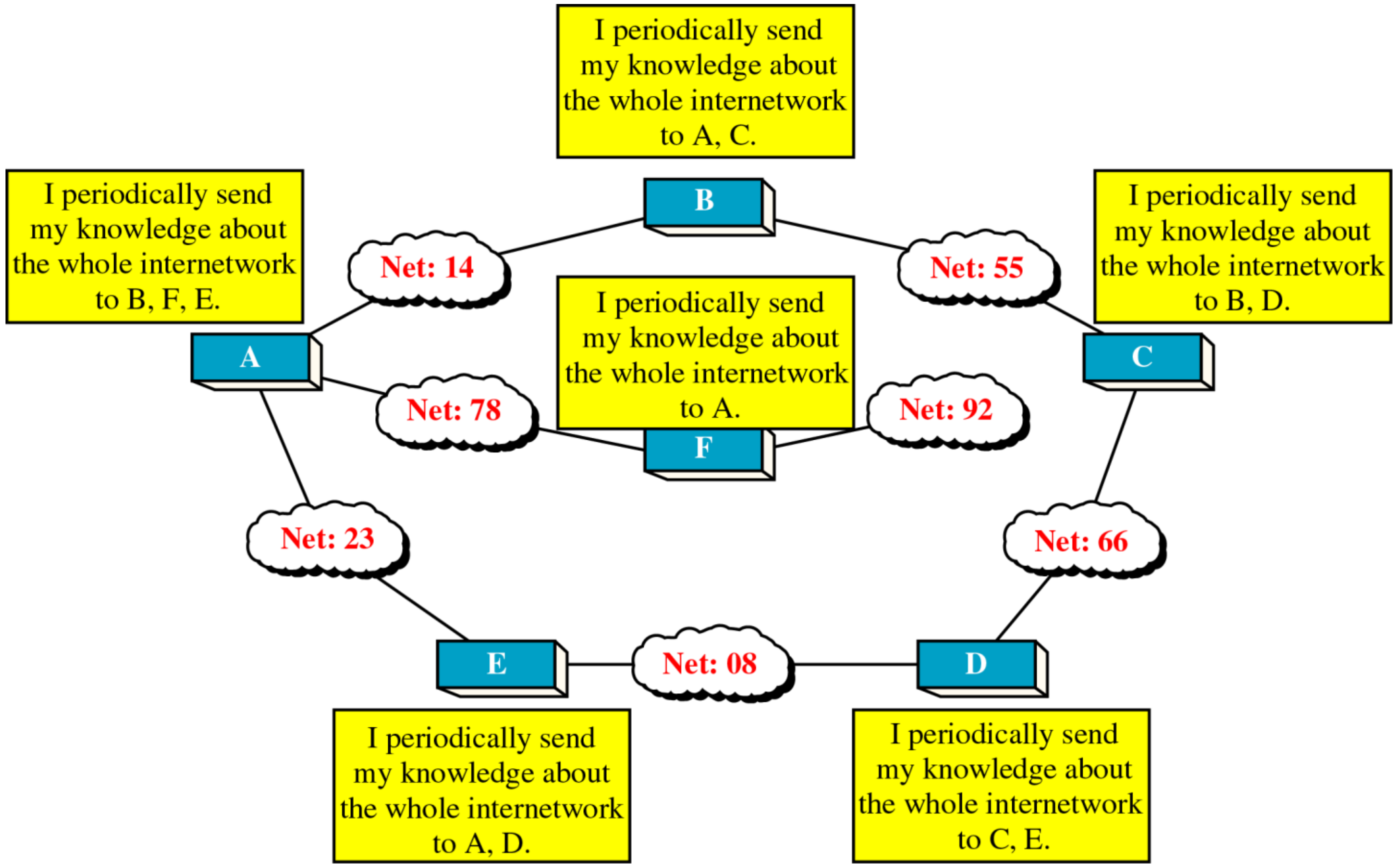
- cost = 1 for every link ⇒ finds ***minimum-hop*** routes



- "clouds" represent LANs; number in cloud represents network ID
- A, B, C, D, E, F are routers (or gateways)

Network Layer: Distance-Vector routing (cont.)

• each router sends its information about the entire network only to its neighbours:


I periodically send my knowledge about the whole internetwork to A, C.
I periodically send my knowledge about the whole internetwork to B, F, E.
I periodically send my knowledge about the whole internetwork to B, D.
B
Net: 14
Net: 55
I periodically send my knowledge about the whole internetwork to A.
A
C
Net: 78
Net: 92
F
Net: 23
Net: 66
E
Net: 08
D
I periodically send my knowledge about the whole internetwork to A, D.
I periodically send my knowledge about the whole internetwork to C, E.

• how do non-neighbouring routers learn about each other and share information ?

  • a router sends its information to its neighbours; each neighbour router adds this information to its own, and sends the updated information to ***its*** neighbours; so first router learns about its neighbours' neighbours, ...

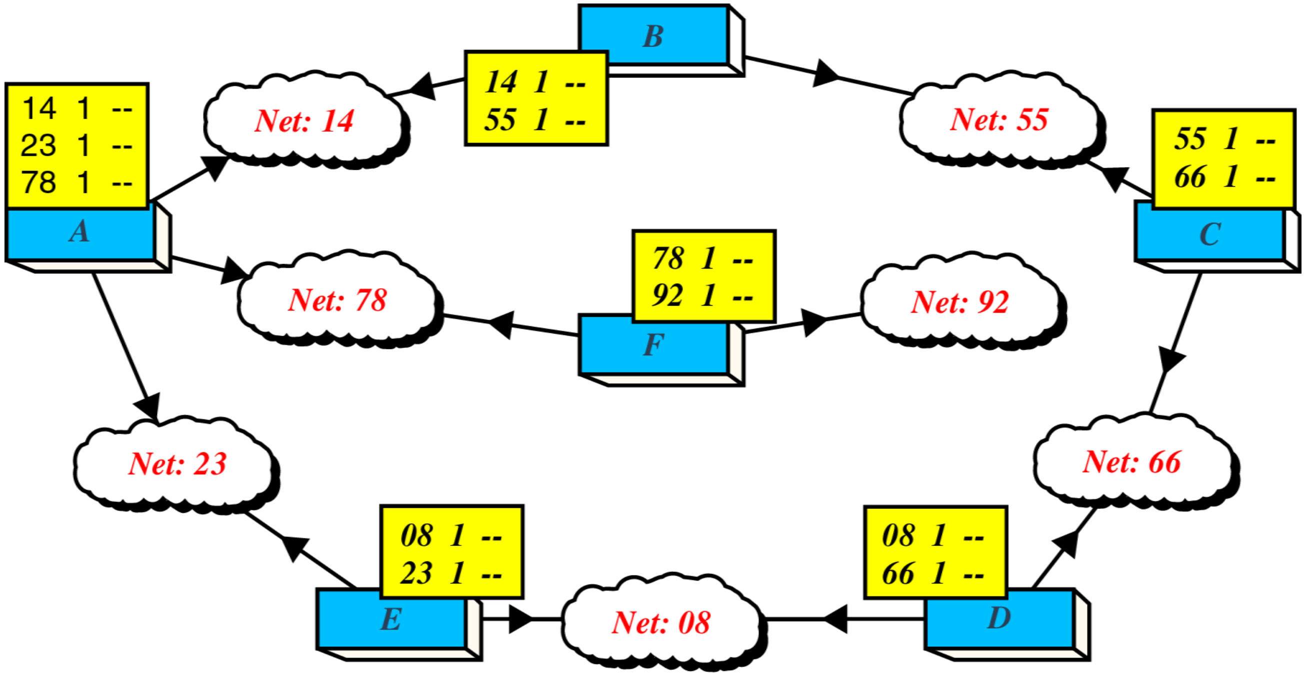
Network Layer: Distance-Vector routing (cont.)

• each router stores information about the network in its ***routing table***

Network ID	Cost	Next Hop
. . . .	. . . .	. . . .
. . . .	. . . .	. . . .
. . . .	. . . .	. . . .
. . . .	. . . .	. . . .
. . . .	. . . .	. . . .

Network ID = final destination of Packet
Cost = number of hops from this router to final destination
Next Hop = neighbouring router to which Packet should be sent

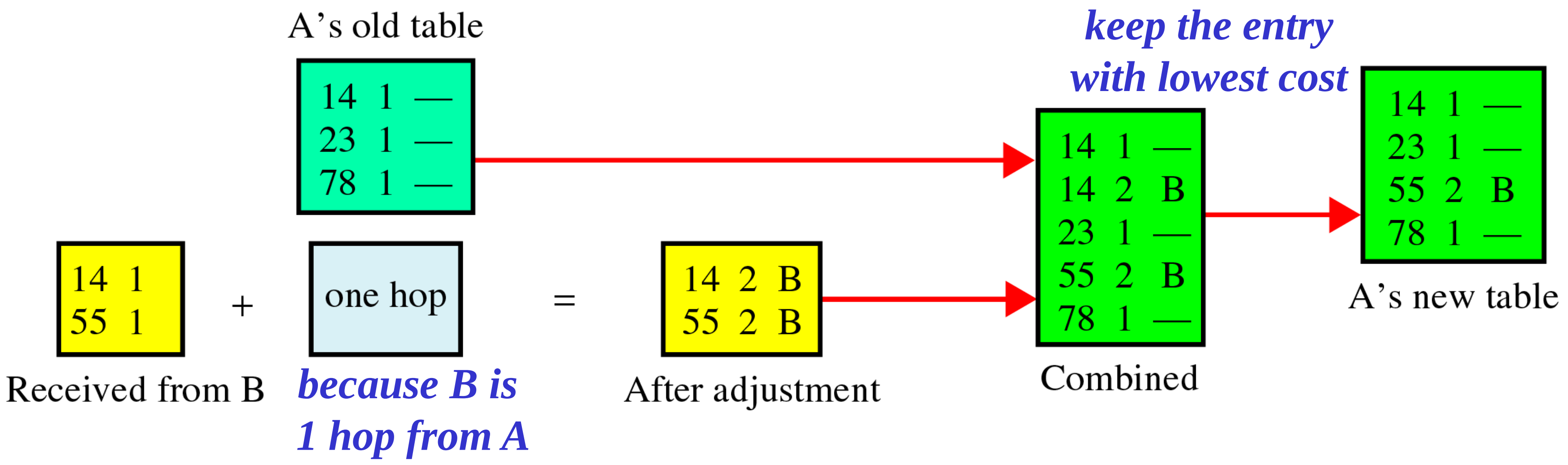
• initially, all a router knows is the network IDs of the networks to which it is directly connected



initial routing table exchanges (no multi-hop routes yet)

Network Layer: Distance-Vector routing (cont.)

• how is a router's routing table updated when new information is received ?

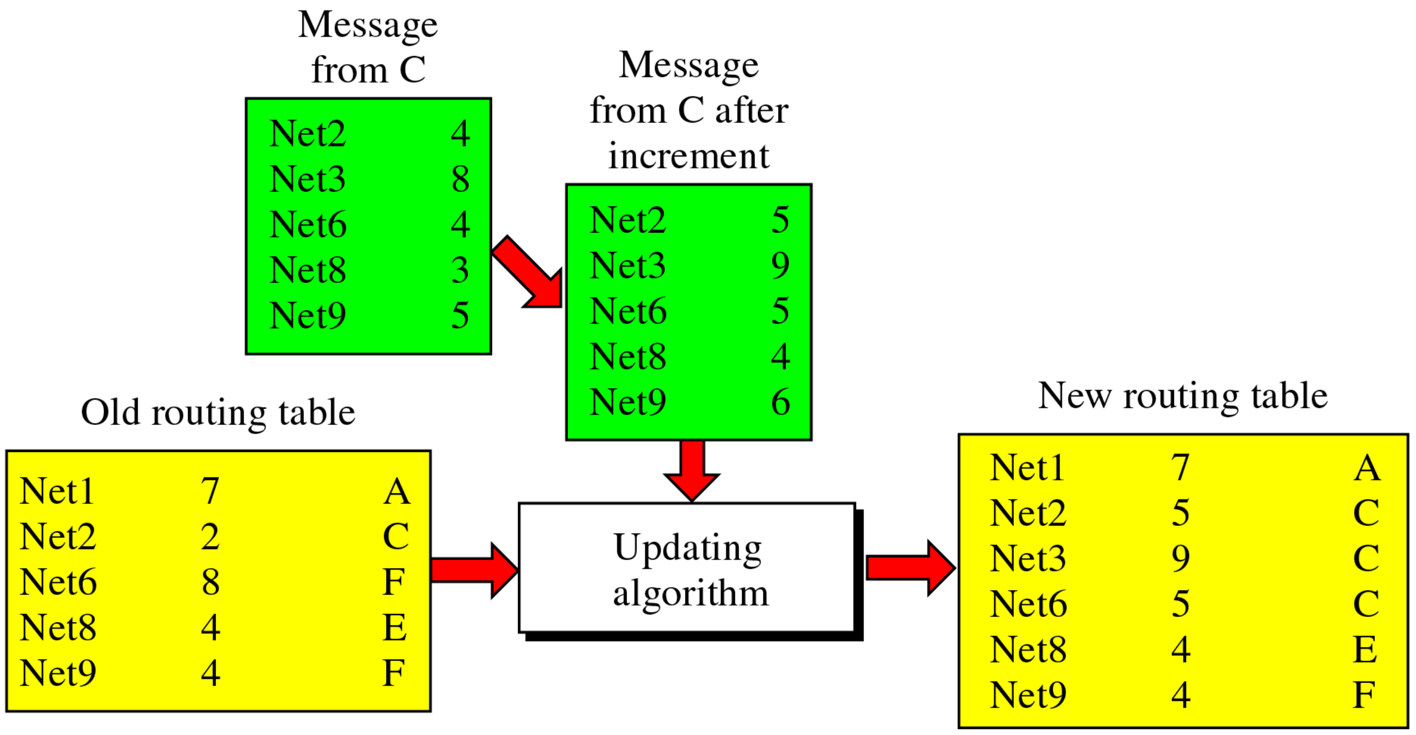


• routing table update algorithm ***(distributed Bellman-Ford algorithm)***:

- • add 1 to cost of each incoming route (since each neighbour is 1 hop away)
- • if a new destination learned, add its information to the routing table
- • if new information received on an existing destination:
  - • if Next Hop field is the same, replace existing entry with the new information ***even if the cost is greater*** ("new information invalidates old")
  - • if Next Hop field is not the same, ***only*** replace existing entry with the new information ***if the cost is lower***

Network Layer: Distance-Vector routing (cont.)

- example of routing table update algorithm (unrelated to earlier Example network):



Message from C

Net2	4
Net3	8
Net6	4
Net8	3
Net9	5

Message from C after increment

Net2	5
Net3	9
Net6	5
Net8	4
Net9	6

Old routing table

Net1	7	A
Net2	2	C
Net6	8	F
Net8	4	E
Net9	4	F

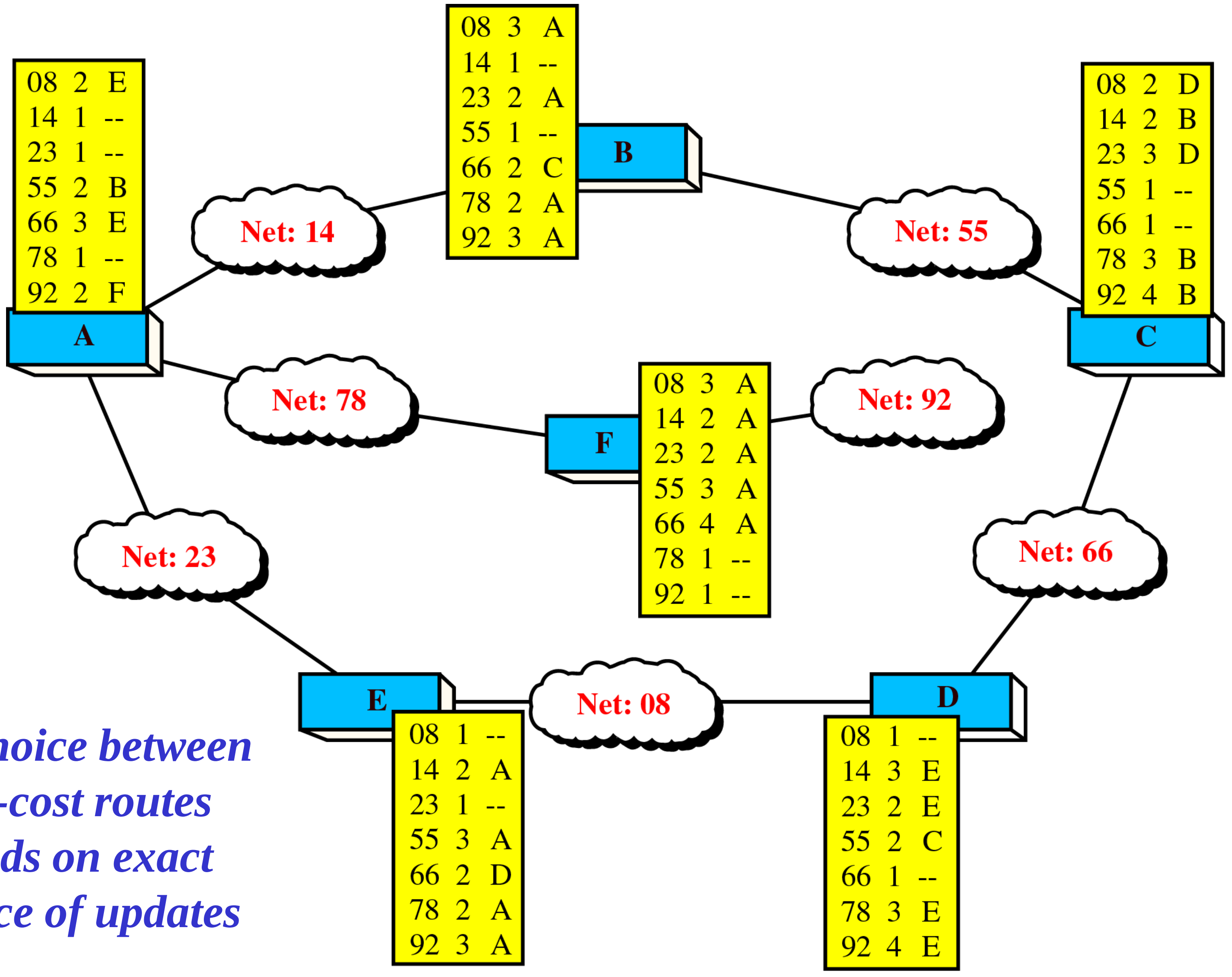
New routing table

Net1	7	A
Net2	5	C
Net3	9	C
Net6	5	C
Net8	4	E
Net9	4	F

Note: no new information about Net1 received, so its entry in the routing table is not updated

Network Layer: Distance-Vector routing (cont.)

• final (converged) routing tables for earlier Example network:



A:

08	2	E
14	1	--
23	1	--
55	2	B
66	3	E
78	1	--
92	2	F

B:

08	3	A
14	1	--
23	2	A
55	1	--
66	2	C
78	2	A
92	3	A

C:

08	2	D
14	2	B
23	3	D
55	1	--
66	1	--
78	3	B
92	4	B

F:

08	3	A
14	2	A
23	2	A
55	3	A
66	4	A
78	1	--
92	1	--

E:

08	1	--
14	2	A
23	1	--
55	3	A
66	2	D
78	2	A
92	3	A

D:

08	1	--
14	3	E
23	2	E
55	2	C
66	1	--
78	3	E
92	4	E

Note: choice between equal-cost routes depends on exact sequence of updates

Network Layer: Distance-Vector routing in practice

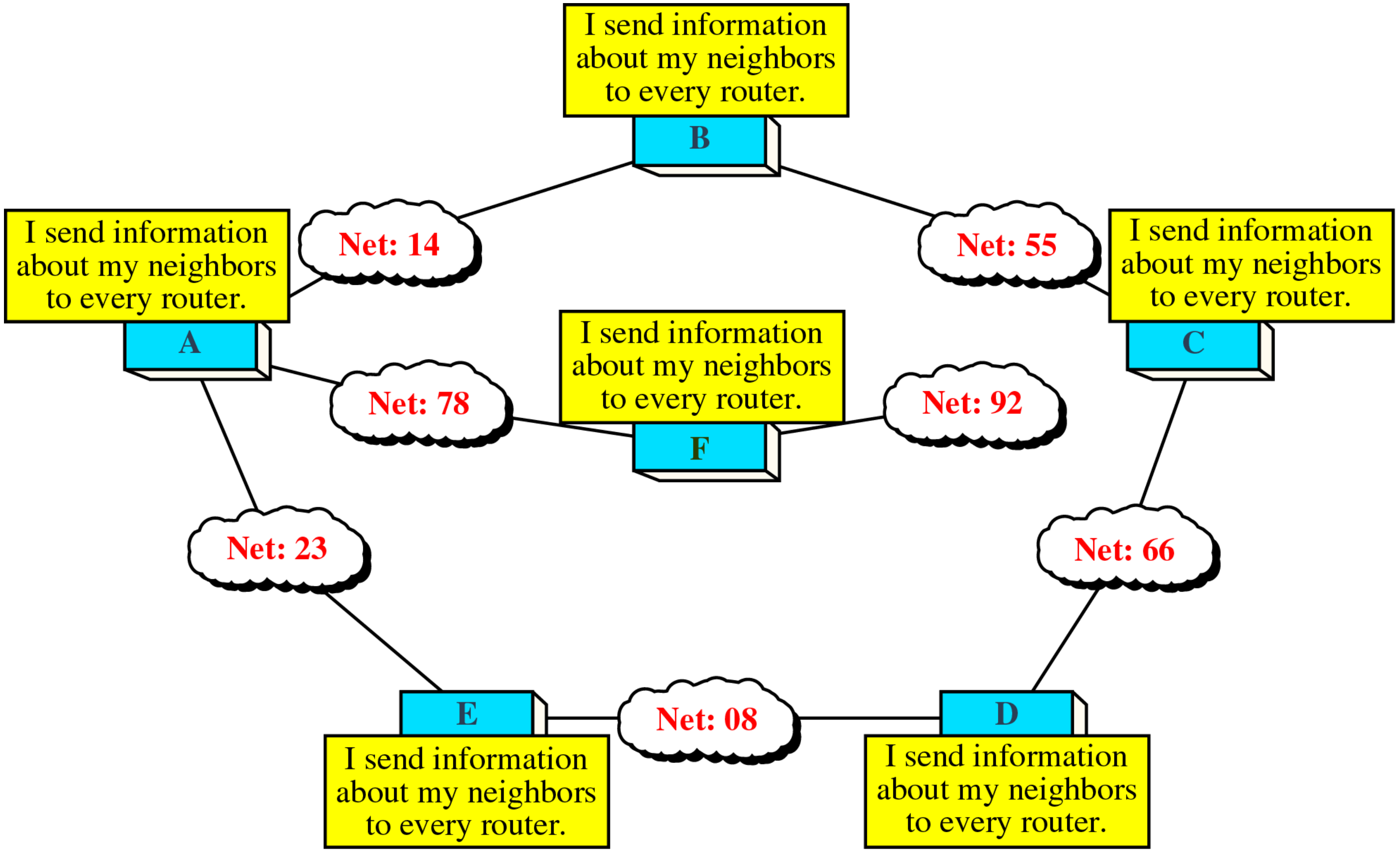
- original ARPANET (forerunner of the Internet) used distance-vector routing
    - subsequently used in the Internet as ***RIP*** (Routing Information Protocol)
    - a variation of distance-vector routing is used in ***BGP*** (Border Gateway Protocol), which finds routes from one *autonomous system* (AS) to another AS
        - AS = a part of the Internet (e.g. a network) managed by one entity

- link cost can be something other than 1 for each link…
    - e.g. packet delay, number of packets queued, …

- problem with distance-vector routing: ***count-to-infinity problem***
    - this refers to the slow convergence of distance-vector routing algorithms under some conditions
    - basic flaw – slow reaction to link/router failure <u>because</u> information only comes from neighbouring routers and it may be out-of-date (e.g. it may not properly reflect the impact of the failure on route costs)
    - many ad-hoc solutions have been tried (e.g. “split horizon”), but either they also fail to solve the count-to-infinity problem, or they are hard to implement
    - this slow convergence was one of the main reasons why other types of routing algorithm were explored, leading to link-state routing

Network Layer: Link-State routing

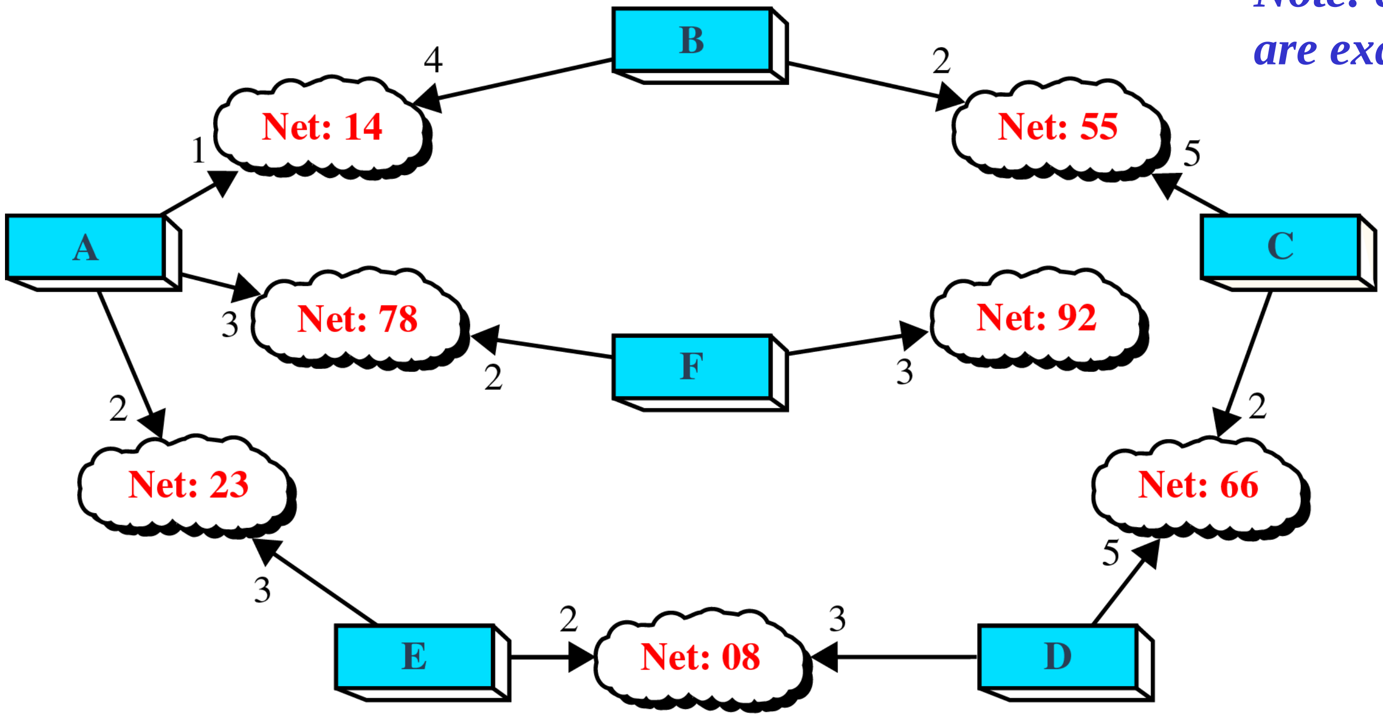
• each router sends information about its neighbourhood to every other router:


I send information about my neighbors to every router.
B
I send information about my neighbors to every router.
A
Net: 14
Net: 55
I send information about my neighbors to every router.
C
I send information about my neighbors to every router.
F
Net: 78
Net: 92
Net: 23
Net: 66
E
Net: 08
D
I send information about my neighbors to every router.
I send information about my neighbors to every router.

Network Layer: Link-State routing (cont.)

- link cost is usually a weighted sum of various factors
  - e.g. traffic level, security level, packet delay, …
- link cost is ***from*** a router ***to*** the network connecting it to another router
  - when a packet is in a LAN (which is typically a broadcast network), every node – including the router – can receive it ⇒ no cost assigned when going from a network to a router

Note: costs shown are examples only



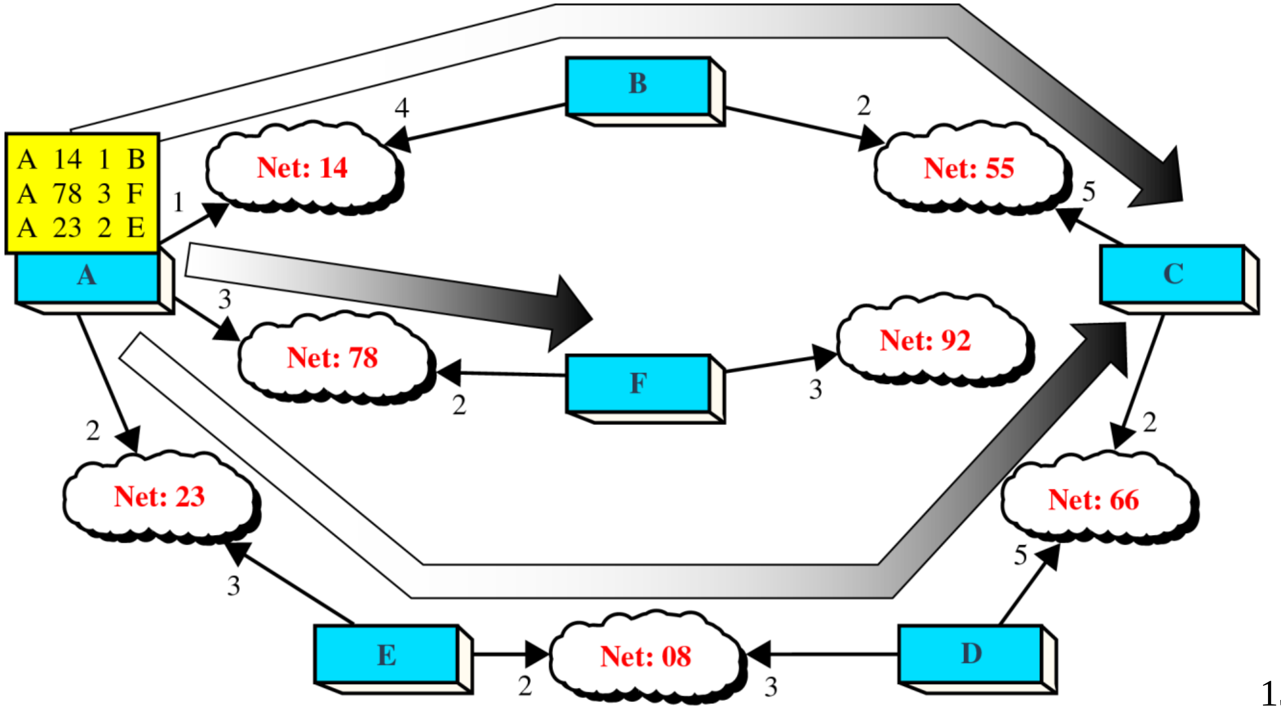
Network Layer: Link-State routing (cont.)

• routers share information by ***advertising***, which means sending ***link-state packets:***

Advertiser	Network	Cost	Neighbor
....	......	....	....
....	......	....	....
....	......	....	....

Advertiser: ID of sending router
Network: ID of destination network
Cost: link cost to neighbour
Neighbour: ID of neighbour router

• a router gets its information about its neighbourhood by sending short ECHO packets to its neighbours and monitoring the responses:



the Figure shows how router A's link-state packet is flooded to all other routers

Network Layer: Link-State routing (cont.)

• every router builds a link-state packet and floods it through the network, so when all such packets have been received at a router, it can build its link-state database:

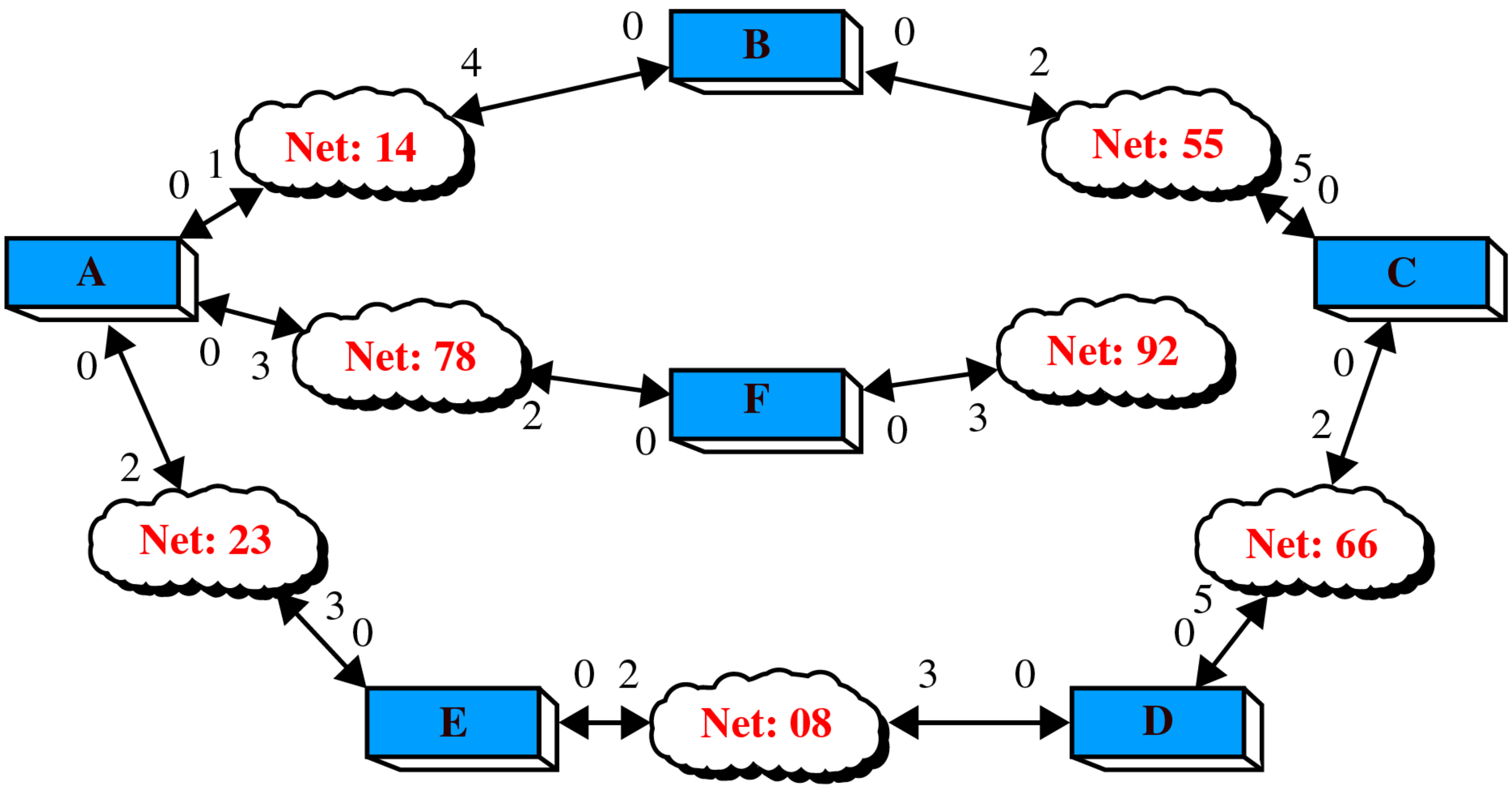
Advertiser	Network	Cost	Neighbor
A	14	1	B
A	78	3	F
A	23	2	E
B	**14**	**4**	**A**
B	**55**	**2**	**C**
C	55	5	B
C	66	2	D
D	**66**	**5**	**C**
D	**08**	**3**	**E**
E	23	3	A
E	08	2	D
F	**78**	**2**	**A**
F	**92**	**3**	**—**

Assuming that every router receives the same set of link-state packets (as if the routers were synchronised), every router builds the <u>same</u> link-state database.

Using this database, each router can then <u>calculate</u> its routing table.

Network Layer: Link-State routing (cont.)

- to calculate its routing table, a router uses ***Dijkstra's Shortest-Path algorithm***
  - first, identify all link costs in the network: either from the link-state database, or using the fact that the cost of any link from a network to a router is 0



Network Layer: Link-State routing – Dijkstra’s algorithm

- this algorithm builds a ***shortest-path spanning tree*** for the router: such a tree has a route to all possible destinations, and no loops
    - the router running the algorithm is the **root** of its shortest-path spanning tree
    - even if all routers’ link-state databases are identical, the trees determined by the routers are different (since the root of each tree is different)
- a *node* is either a network or a router; nodes are connected by *arcs*
- the algorithm keeps track of 2 sets of nodes and arcs – ***Temporary*** and ***Permanent***
    - initially, the Temporary set contains all neighbour nodes of the router itself, and the arcs connecting them to the router; only the router is initially Permanent
    - when **all** nodes and arcs are in the Permanent set, the algorithm has terminated

identify the Temporary node whose arc has the **lowest cumulative cost** from the root: this node and arc are moved into the Permanent set;

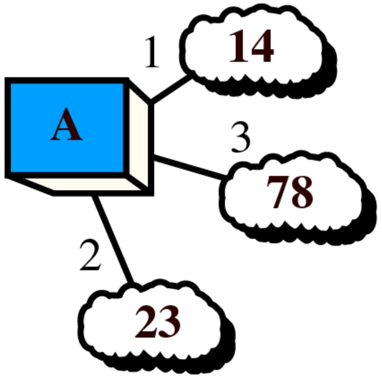
any nodes which are connected to the new Permanent node and are not already in the Temporary set, along with the connecting arcs, are made Temporary. Also, if any node already in the Temporary set has a ***lower*** cumulative cost from the root by using a route passing through the new Permanent node, then this new route replaces the existing one;

repeat until all nodes and arcs are Permanent.

Network Layer: Link-State routing – Dijkstra’s algorithm (cont.)

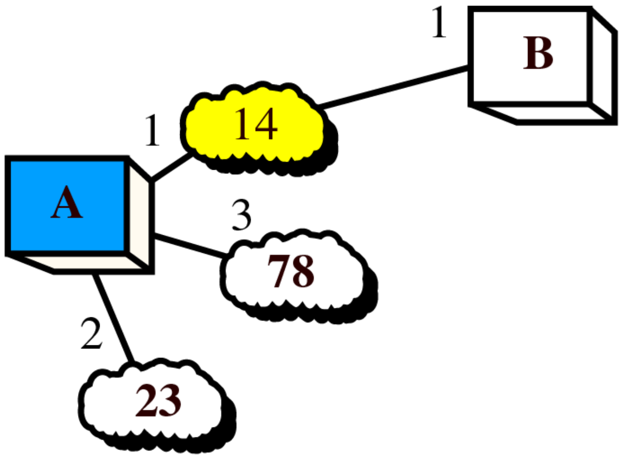
• as an Example, let’s follow the steps of the algorithm run by router A

1.



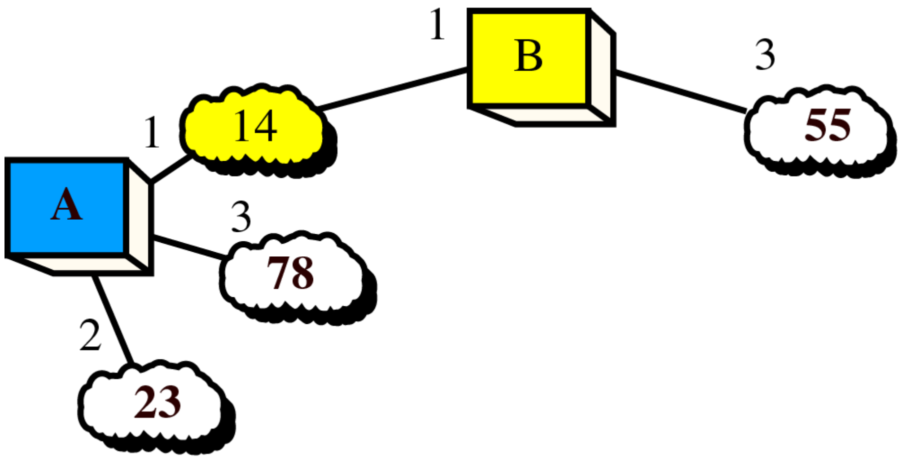
Root is A, networks
14, 78, 23 added

2.



14 permanent, B added

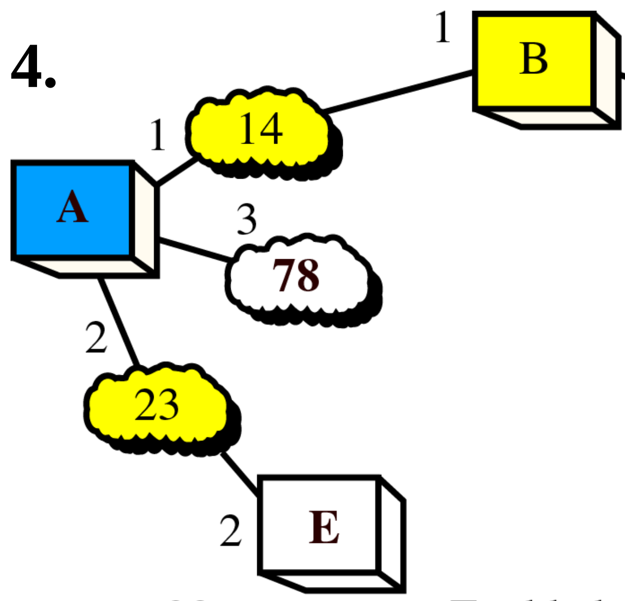
3.



B Permanent, 55 added

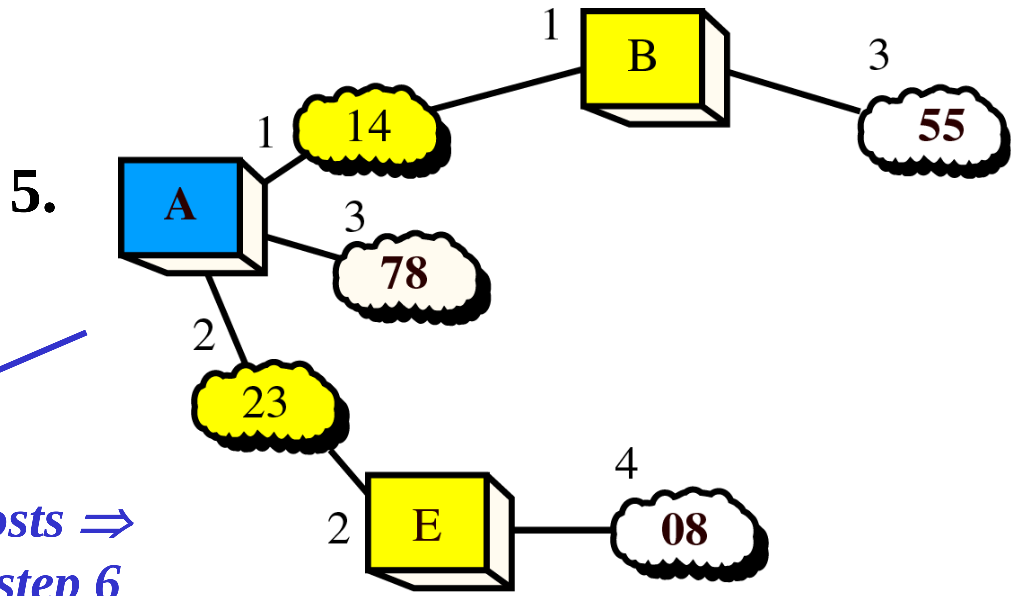
Note: arcs are marked with their <u>cumulative</u> cost from the root (not individual costs)

Network Layer: Link-State routing – Dijkstra's algorithm (cont.)



23 permanent, E added

Note: arcs are marked with their ***cumulative*** *cost from the root (not individual costs)*

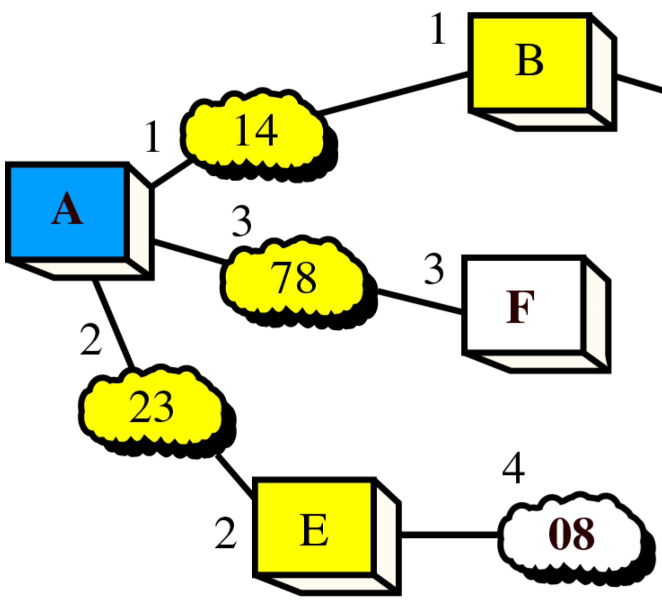


E permanent, 08 added

Note: equal cumulative costs ⇒ choose one arbitrarily in step 6

Network Layer: Link-State routing – Dijkstra's algorithm (cont.)

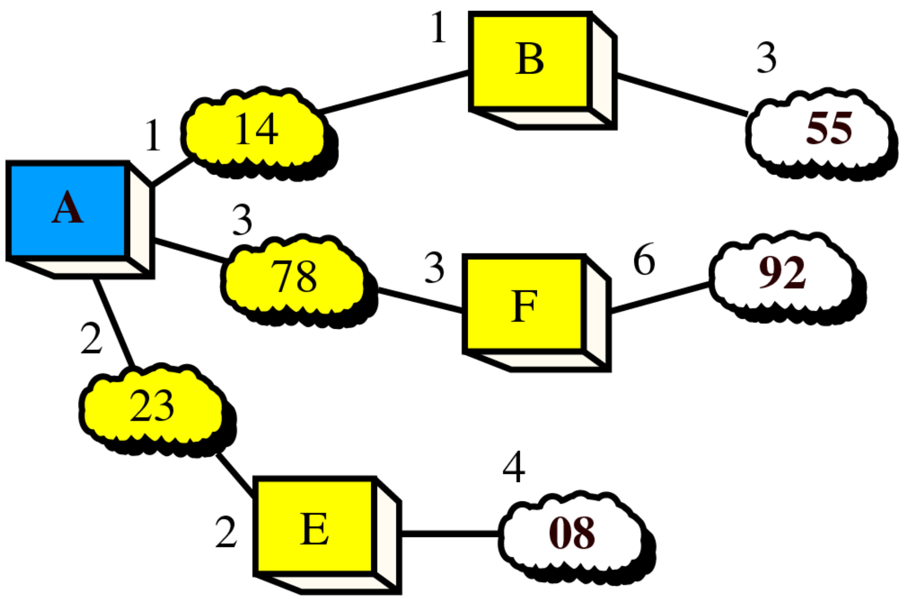
6.



78 permanent, F added

Note: arcs are marked with their ***cumulative*** *cost from the root (not individual costs)*

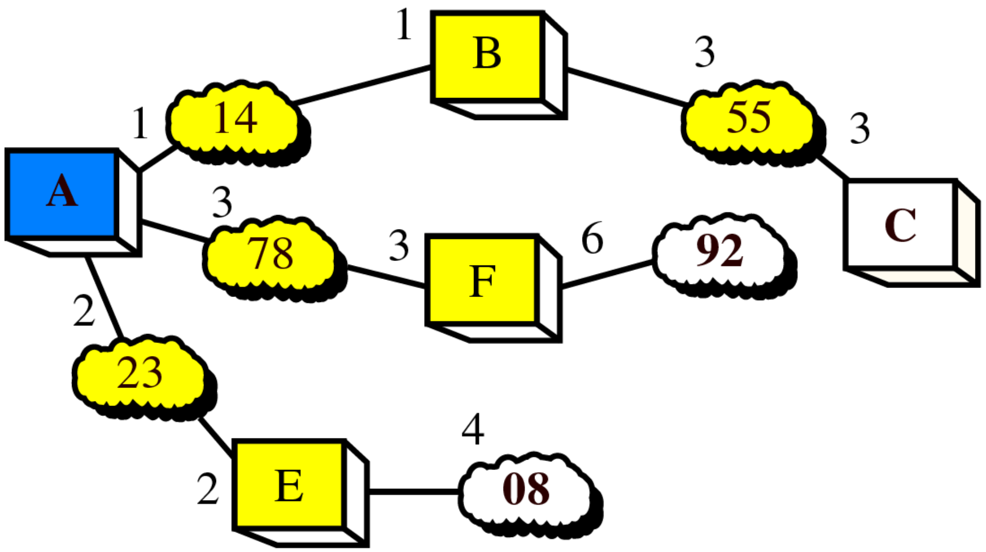
7.



F permanent, 92 added

Network Layer: Link-State routing – Dijkstra's algorithm (cont.)

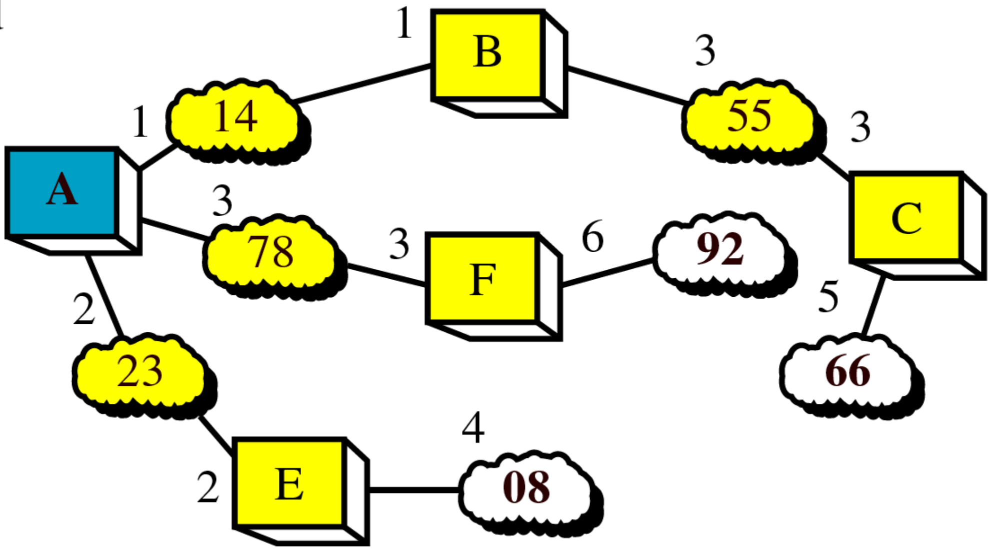
8.



55 permanent, C added

Note: arcs are marked with their <u>cumulative</u> cost from the root (not individual costs)

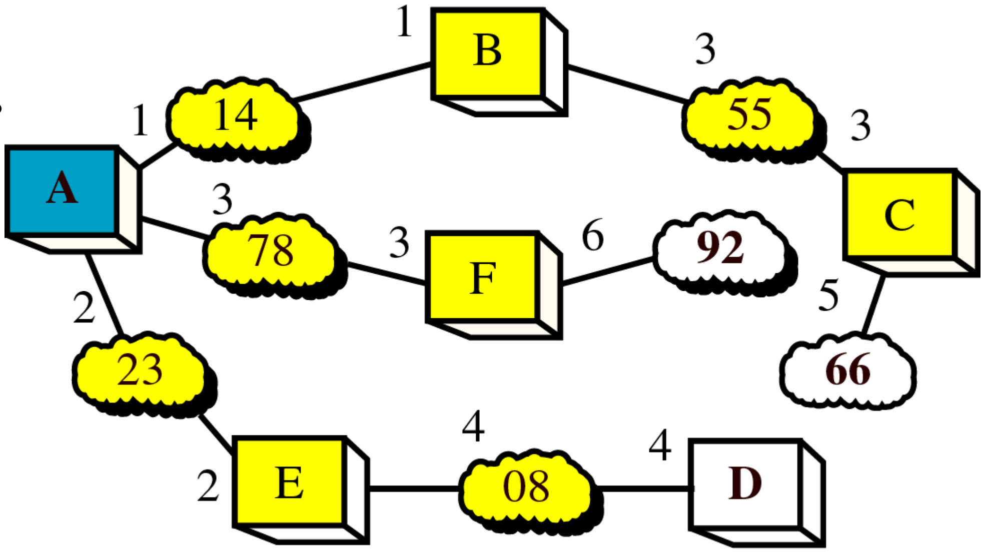
9.



C permanent, 66 added

Network Layer: Link-State routing – Dijkstra’s algorithm (cont.)

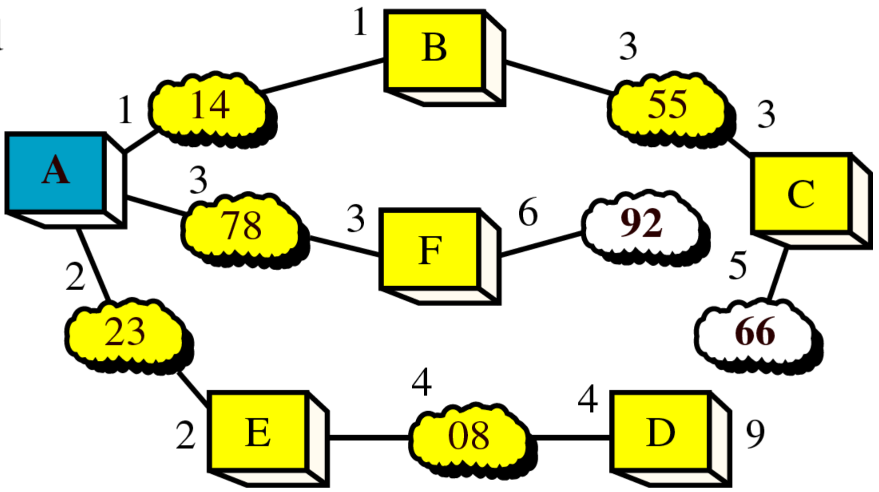
10.



08 permanent, D added

Note: arcs are marked with their ***cumulative*** *cost from the root (not individual costs)*

11.

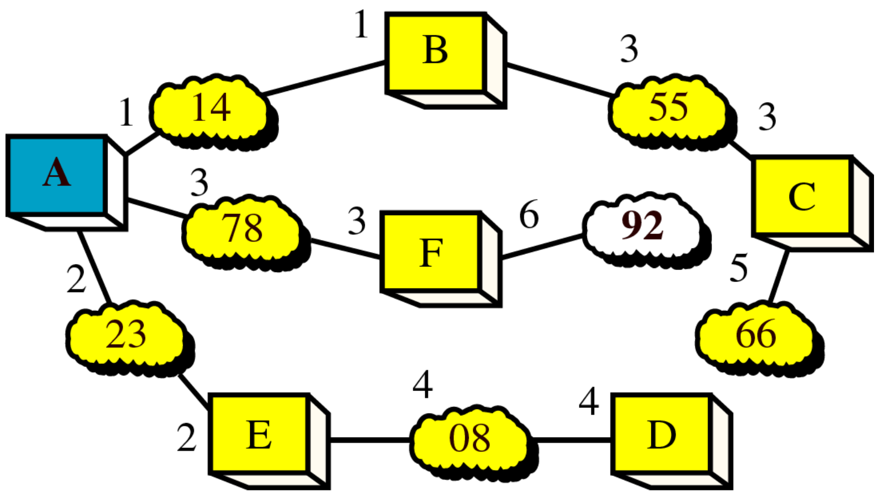


D permanent, 66 added.
But 9 > 5, so that link deleted

if the new arc to network 66 from router D had a lower cumulative cost than the one from router C, then the new link would replace the old one

Network Layer: Link-State routing – Dijkstra’s algorithm (cont.)

12.

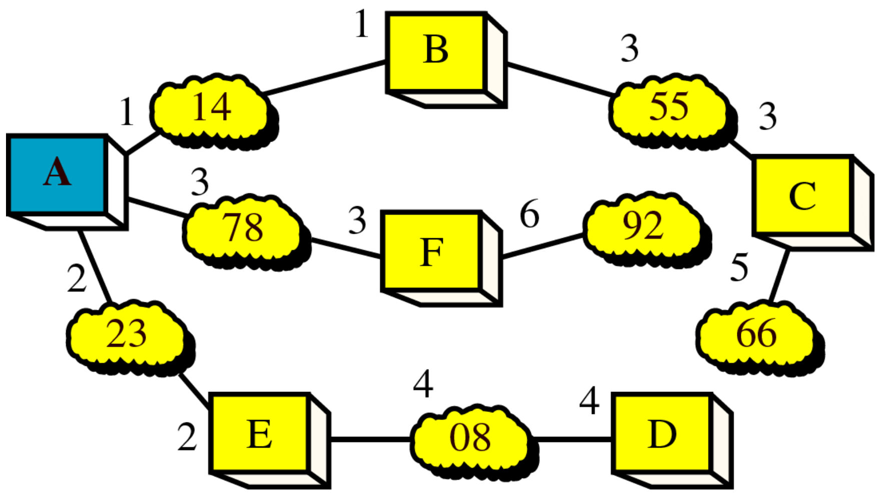


66 permanent

Note: arcs are marked with their <u>cumulative</u> cost from the root (not individual costs)

13.

all nodes and arcs are Permanent ⇒ STOP: this router’s shortest-path spanning tree has been found



92 permanent

Network Layer: Link-State routing – routing table

• once a router has found its shortest-path spanning tree, it can build its routing table

    • to complete the Example, here is router A's link-state routing table:

Net	Cost	Next router
08	4	E
14	1	--
23	2	--
55	3	B
66	5	B
78	3	--
92	6	F

Note: each router's routing table will (in general) be different

Networks 14, 23, and 78 don't have a "Next router" entry because they are directly connected to this router

• in large networks, the *memory required* to store the link-state database and the *computation time* to calculate the link-state routing table can be significant

• in practice, since the link-state packet receptions are not synchronised, routers may be using different link-state databases to build their routing tables: how inaccurate the results are depends on how different the routers' "views" of the network are

Network Layer: Link-State routing in practice

• link-state routing algorithms have several desirable properties, e.g. rapid convergence; small amount of traffic generated; rapid response to topology changes

• examples from the Internet are the ***Open Shortest Path First*** (OSPF) and ***Intermediate System to Intermediate System*** (IS-IS) routing protocols

- link costs can be configured in OSPF. Possible link costs include:
    - ➢ 1 for each link
    - ➢ reliability: assigned by administrator, indicates how often the link fails
    - ➢ packet delay
    - ➢ link bandwidth
    - ➢ financial cost of the link
- OSPF requires a lot of memory: each router holds its routing table & link-state database
- Dijkstra's algorithm computations are processor-intensive
    - ➢ legacy routers may be unable to relay packets when these calculations are taking place – which ***could*** be every time a link-state packet is received
- OSPF ***can*** consume a lot of bandwidth if the network topology changes often
- link-state packets sent to all routers using **<u>reliable flooding</u>**
    - ➢ need sequence number and time-to-live (TTL) field in each packet…