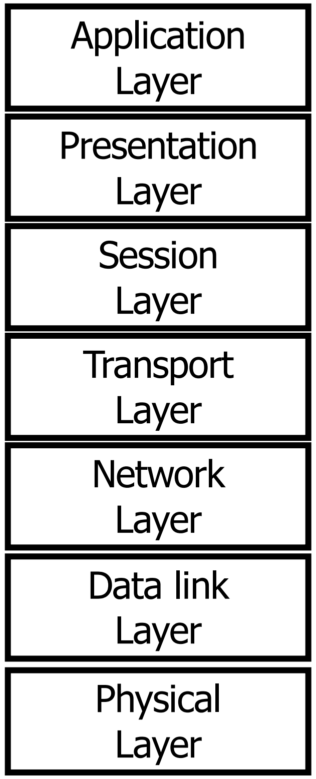
Technische Universität Berlin

Computer Networks

Network Layer

Chapter

Top-Down-Approach



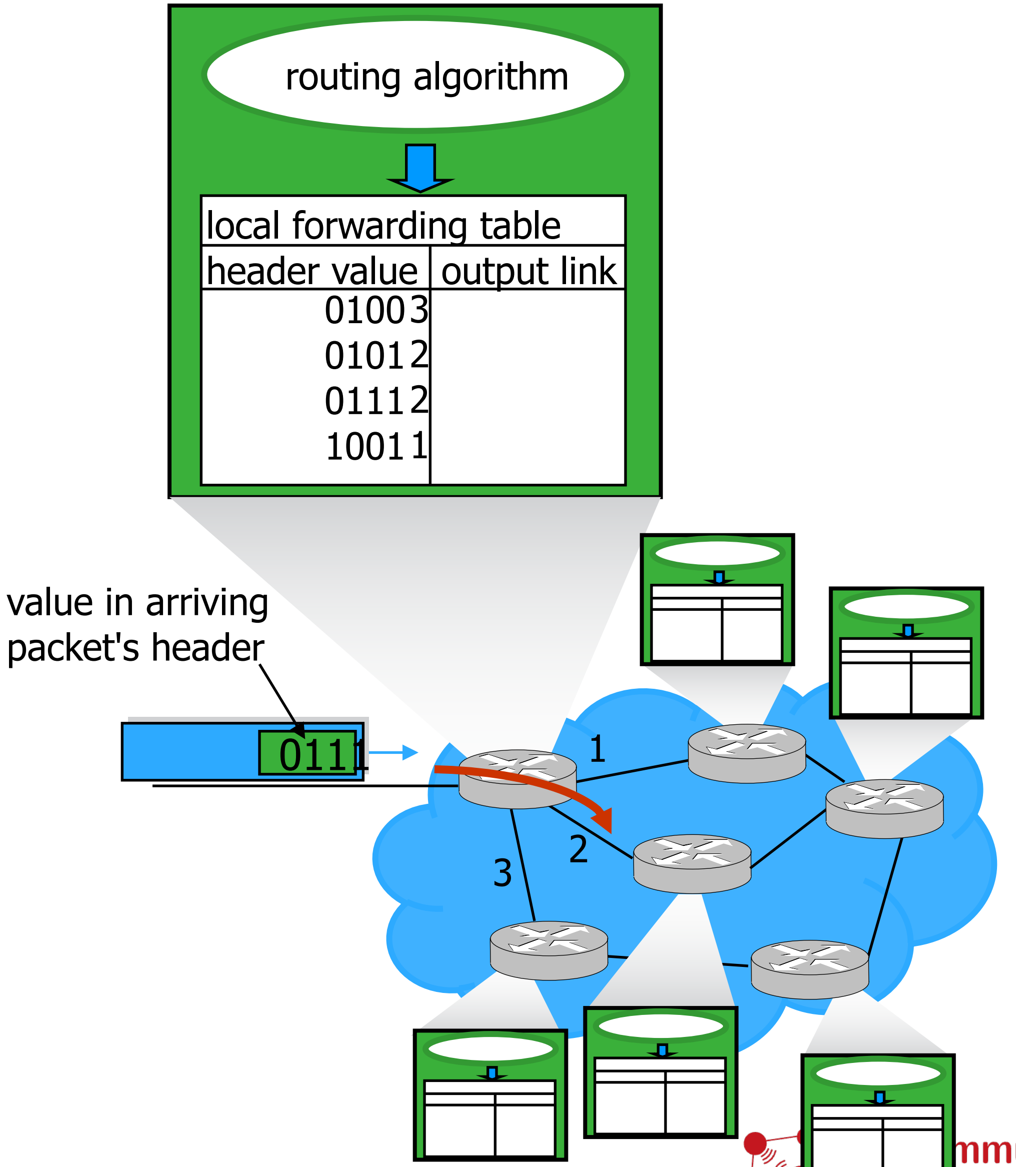
Network Layer

Introduction

- Network layer: responsible for the communication between hosts, which are connected via routers (and, thus, networks)
  - Hosts send and receive, routers forward messages
  - **Forwarding**: router received messages on one interface and forwards it on another interface (the right one!) toward the destination
  - **Routing**: procedure to decide which path to use for which messages

Introduction

- Forwarding and routing

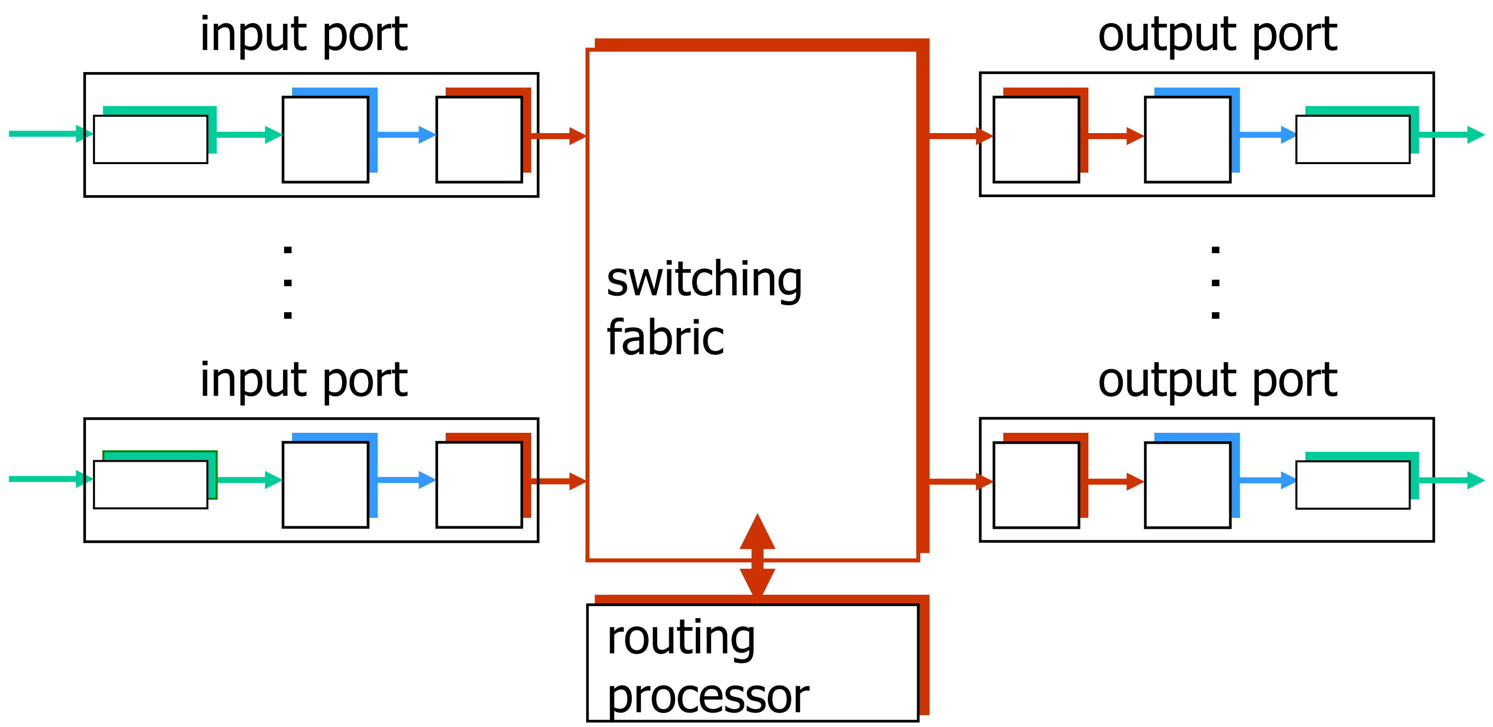


header value	output link
0100	3
0101	2
0111	2
1001	1

Forwarding

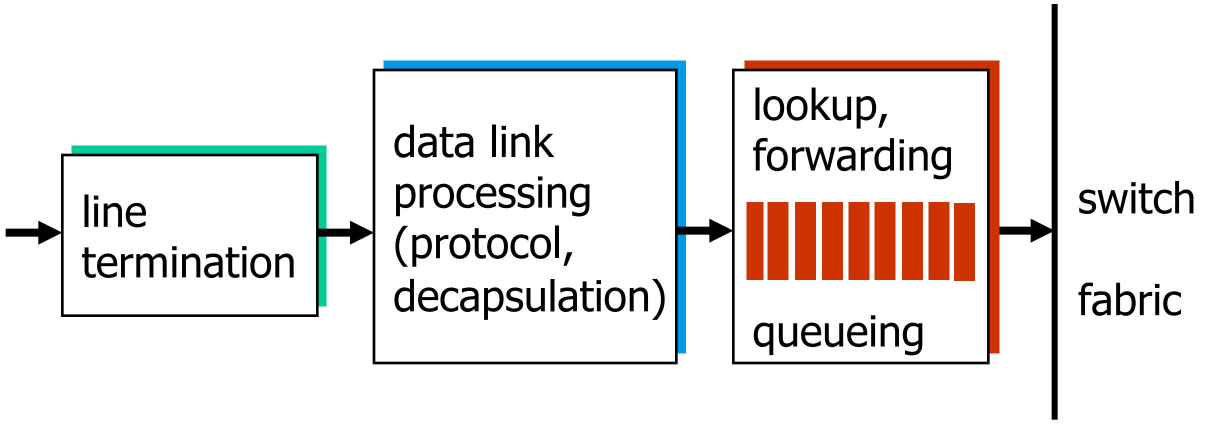
Router

- Tasks: forwarding, execution of routing protocols
- Conceptual architecture:



Router: Input

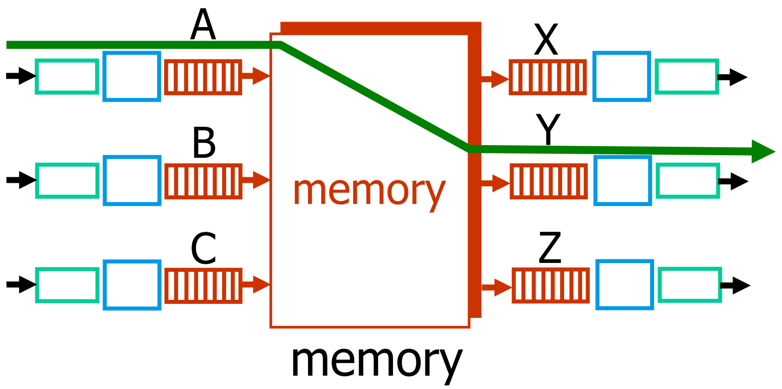
- Input interface



- **Buffering**, if packets are received faster than processed internally
- **Packet loss**, if buffer overflows
- **Efficient forwarding**: each interface often has a copy of the routing table
- Most important: efficient data structures for fast search

Router: Switching Fabric

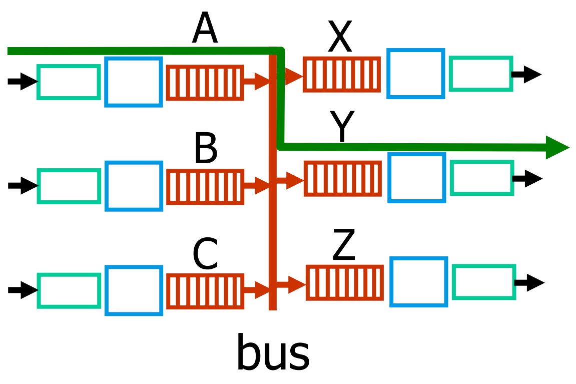
- Memory coupled switching



  - CPU copied packets from input interface to main memory, after deciding where to forward the packet, it copies the packet to the output interface
  - Internal memory bus is used twice (limiting the performance)
  - Architecture of old routers but sometimes used today as well

Router: Switching Fabric

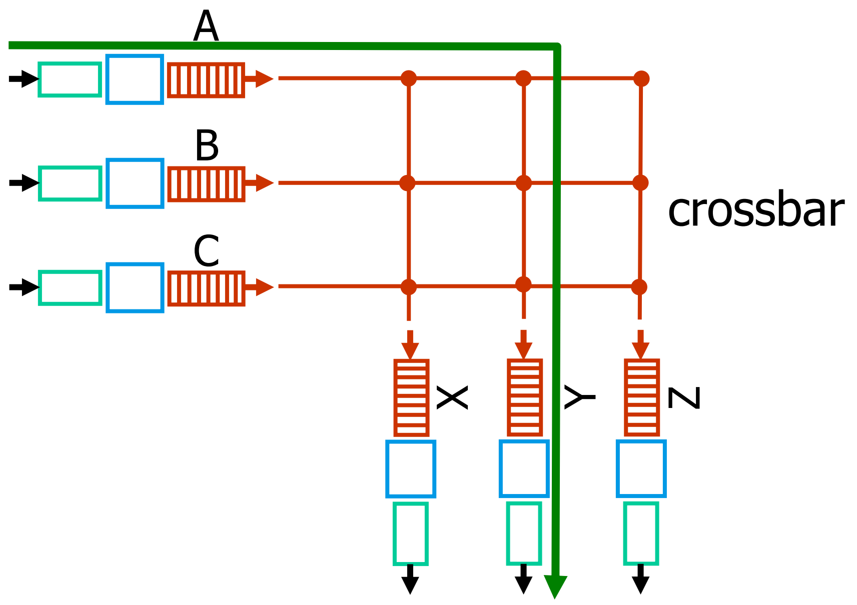
- Bus coupled switching



  - Single bus connects all ports, can be used for a single transfer per time → competition
  - Often used for small routers

Router: Switching Fabric

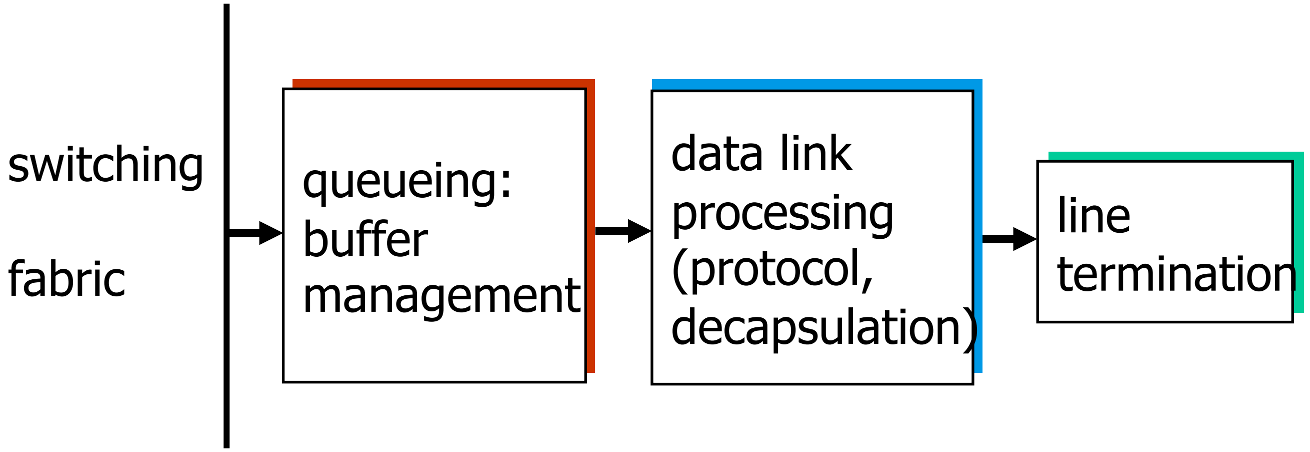
- Switching matrix / switching network



- Known from interconnects of CPUs in parallel computers
- Example: Crossbar, every port can be connected to every other (quadratic number of connections)
- Often also multi-stage

Router: Output

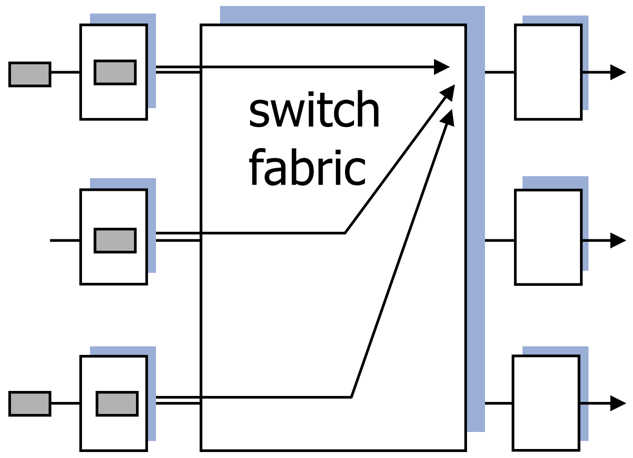
- Output interface


switching
fabric
queueing: buffer management
data link processing (protocol, decapsulation)
line termination

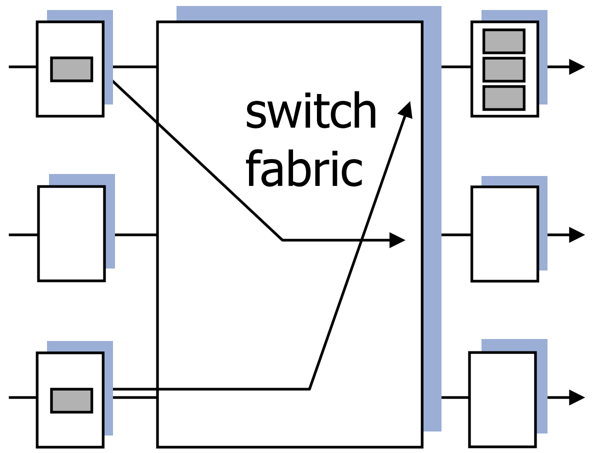
- **Buffering**, if switching fabric can deliver packets faster than transmission over output interface
- **Packet loss**, if buffer overflows
- **Active queue management**: decision, which packet should be discarded
- **Scheduling**: if multiple packets are buffered, which to be processed next

Impact of Buffering and Packet Loss

- If switching fabric is faster than # ports x data rate:
  - Normally no input buffering needed
  - If multiple inputs want to forward to the same output port, buffering is needed



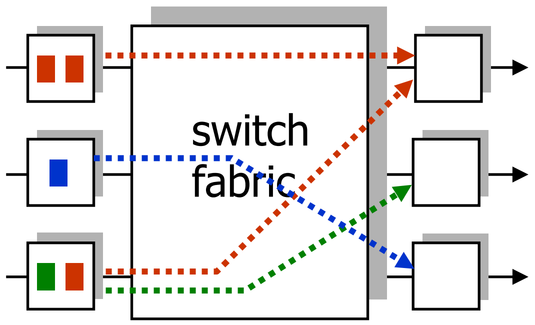
output port contention
at time *t*



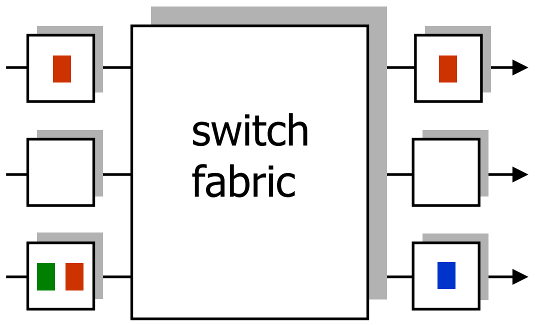
one packet
time later

Impact of Buffering and Packet Loss

- Head-of-the-Line (HOL) Blocking
  - If multiple inputs want to forward to the same output, some need to wait until switching fabric becomes available again
  - All packets waiting at one such input port need to wait, despite their output would be available



output port contention at time t – only one red packet can be transferred



green packet experiences HOL blocking

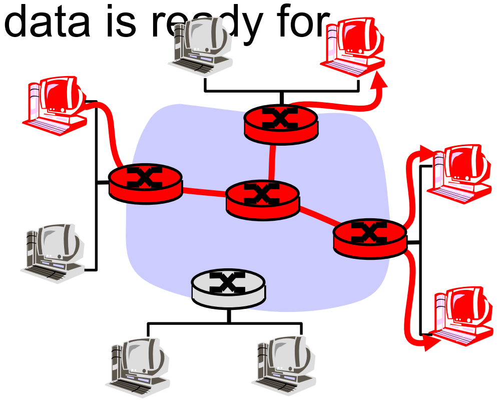
Routing

Routing

- **Routing**: Mechanism to decide which path to use to transmit packets through a network
  - **Intradomain**: within a **routing domain** (= within one administrative domain), algorithms usually find shortest paths, but scalability is limited to smaller networks, two popular variants:
    - **Link-State**: every router has full topology information of the entire routing domain, every router independently calculates shortest paths to all other routers (actually to all networks) using, e.g., the Dijkstra algorithm, example: OSPF
    - **Distance vector**: every router only knows its direct neighbors and the destinations that can be reached via this neighbor, shortest paths are distributed using, e.g., the Bellman-Ford algorithm, example: RIP
  - **Interdomain**: between multiple routing domains
    - Exchanges routing information may include entire paths, rules and filters decide which paths to use, example: BGP

Routing

- **Unicast routing (point to point)**
  - Proactive: routing information is exchanged and updated proactively, routing tables can immediately be used for data exchange → we concentrate on this
  - Reactive: routing information is established only when data is ready for transmission
- **Multicast routing (point to multipoint)**
  - Efficient forwarding to multiple receivers
  - Extension to unicast routing
- **Ad hoc routing**
  - Dynamic network topology
  - Routing information can get outdated quickly
- **Data centric routing**
  - Routing based on content rather than destination address
  - Example: named data networking (NDN)

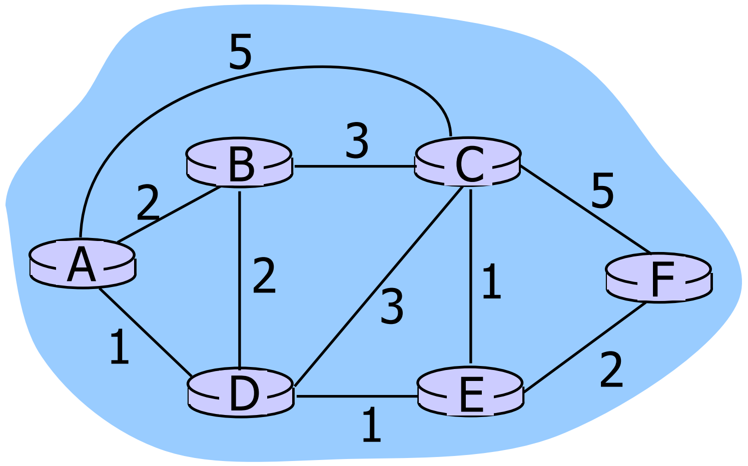


Routing: Graphs

- Graphs are often used as an abstraction for networks
- Allows the use of search algorithms from graph theory
- Basic terminology:
  - **Graph** G = (N,E)
  - **Nodes** N
  - **Edges** E $\subseteq$ N $\times$ N,
    every edge is a pair (v,w) $\in$ E, v and w are called **neighbors**
  - A graph is **undirected** if all edges are symmetric:
    (v,w) $\in$ E $\Rightarrow$ (w,v) $\in$ E
    we only focus on undirected graphs in the following
  - **Cost** is a function c: E $\rightarrow$ C
    for brevity we use the notation c(v,w)

Routing: Graphs

- Example of an undirected graph



- N = {A, B, C, D, E, F}
- E = {(A,B), (A,C), (A,D), (B,A), (B,C), (B,D), (C,A), (C,B), (C,D), (C,E), (C,F), (D,A), (D,B), (D,C), (D,E), (E,C), (E,D), (E,F), (F,C), (F,E)}
- c(A,B) = c(B,A) = 2, c(A,C) = c(C,A) = 5, ...

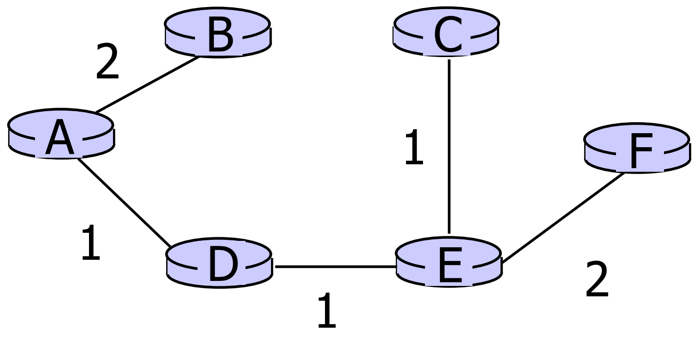
Routing: Graphs

- Some more terminology
  - A **path** is a sequence $(v_1,v_2,\ldots, v_n)$, so that all pairs $(v_1,v_2), (v_2,v_3),\ldots, (v_{n-1},v_n) \in E$
  - The cost of a path is the sum of all edge costs $c(v_1,v_2)+c(v_2,v_3)+..+c(v_{n-1},v_n)$, it is called **distance** $D(v_1,v_n)$
  - A path between two nodes v_1 and v_2 is called a **shortest path**, if there is no other path between both nodes with smaller cost (please note, there might be multiple shortest paths with the same cost)
  - A **cycle** is a path where start node = end node
  - A graph is **connected**, if there is a path between every pair of nodes

Routing: Graphs

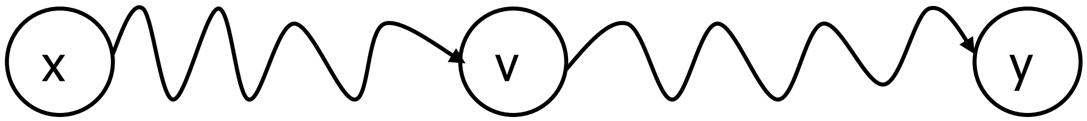
- And even more…
  - A **tree** is connected graph that has no cycles
  - A **spanning tree** of a graph G = (N,E) is a tree B = (N,E´) mit E´$\subseteq$ E
  - A **minimal spanning tree** of a graph G for a node v is a spanning tree of G, that contains the shortest path between v and every other node in G

Example: minimal spanning tree for node A:



Routing: Graphs

- Properties of shortest paths
  - If v is part of a shortest path between x and y, then the shortest path from x to y can be constructed from the shortest paths between x and v and v and y



  - Recursion scheme: $D(x,y) = \min_v\{D(x,v) + D(v,y)\}$
  - This is the fundamental basis for shortest path algorithms

Link State Routing

Link State Routing

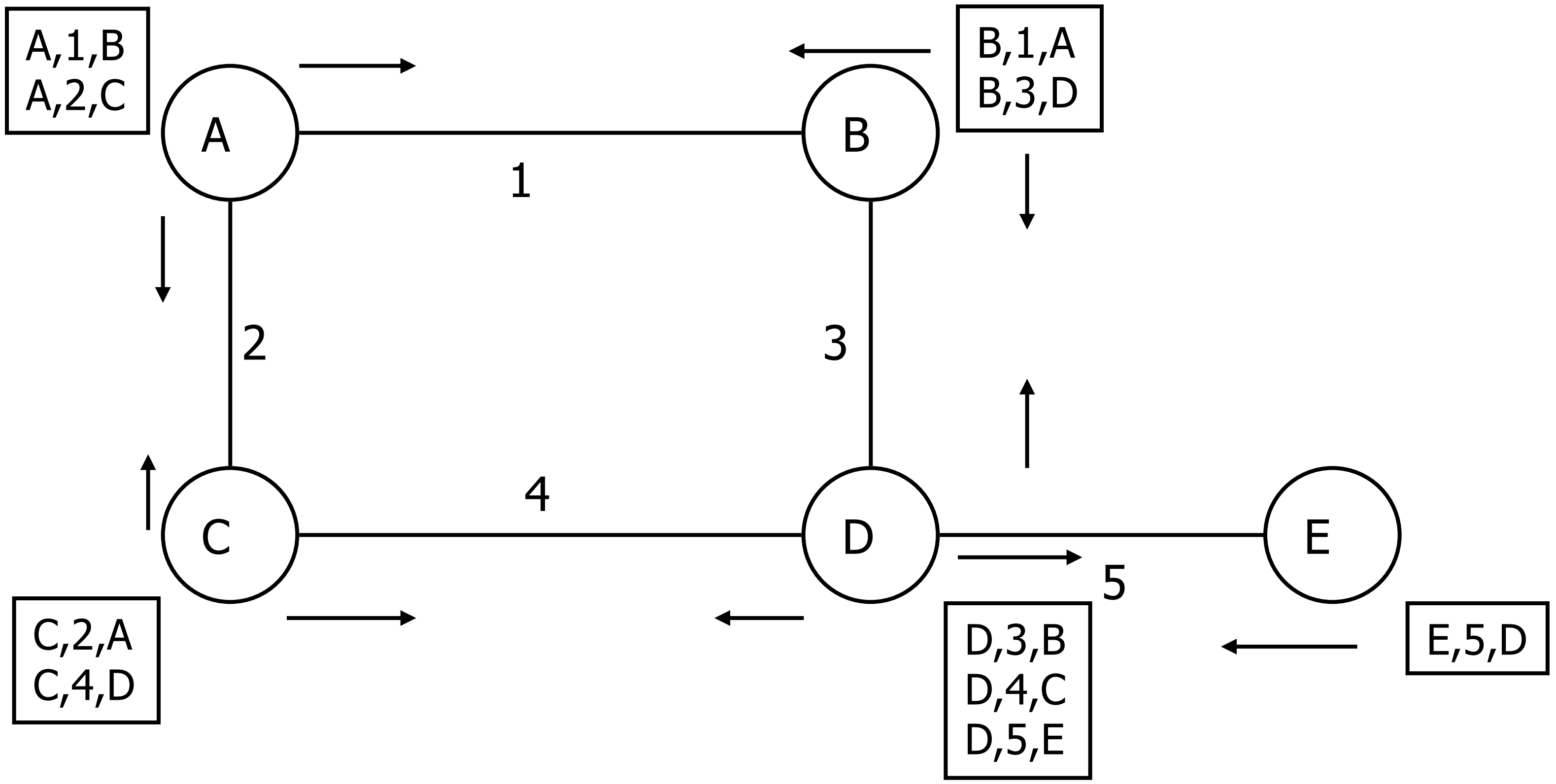
- All nodes have full knowledge of network topology
- This is achieved by **flooding** link state advertisements
- Every node calculates the shortest path to all other nodes using the **Dijkstra algorithm**
- If the topology changes (this can be identified by the underlying data link layer), changes are flooded again followed by a re-calculation of shortest paths

LSA Flooding

- Flooding of **Link-State-Advertisements (LSA) containing**
  - ID of the node that created the LSA
  - All direct neighbors (IDs and path cost to the neighbor)
  - Sequence number
  - Time to live
- Every node creates LSA containing all direct neighbors and sends it to all neighbors
- Received LSAs are first processed to update local topology information and then forwarded to all neighbors except the one from which it was received
- To improve reliability, acknowledgements and retransmissions are used in combination with sequence numbers and time to live field

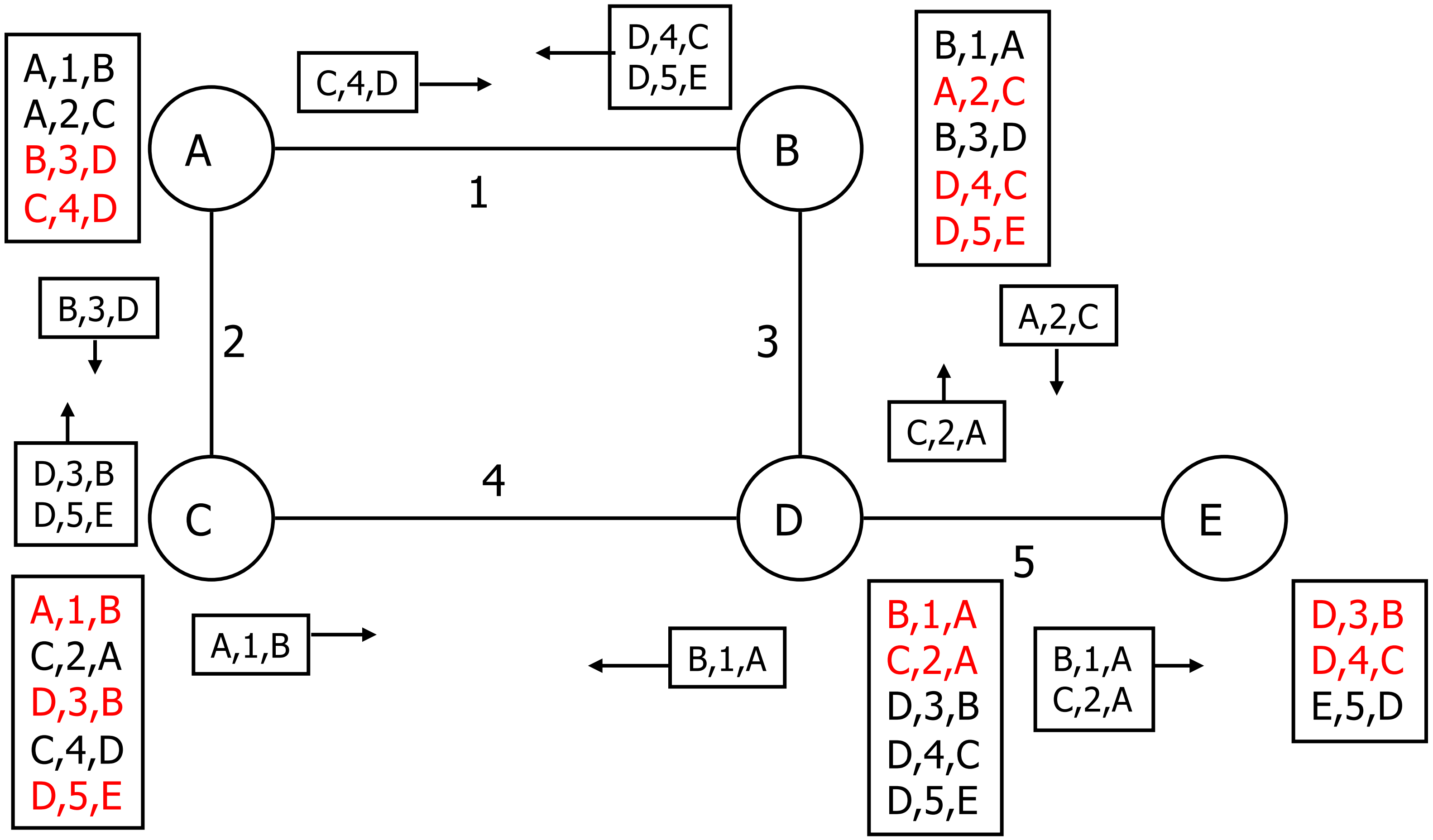
LSA Flooding: Example

- Example: initial LSA creation and distribution to neighbors



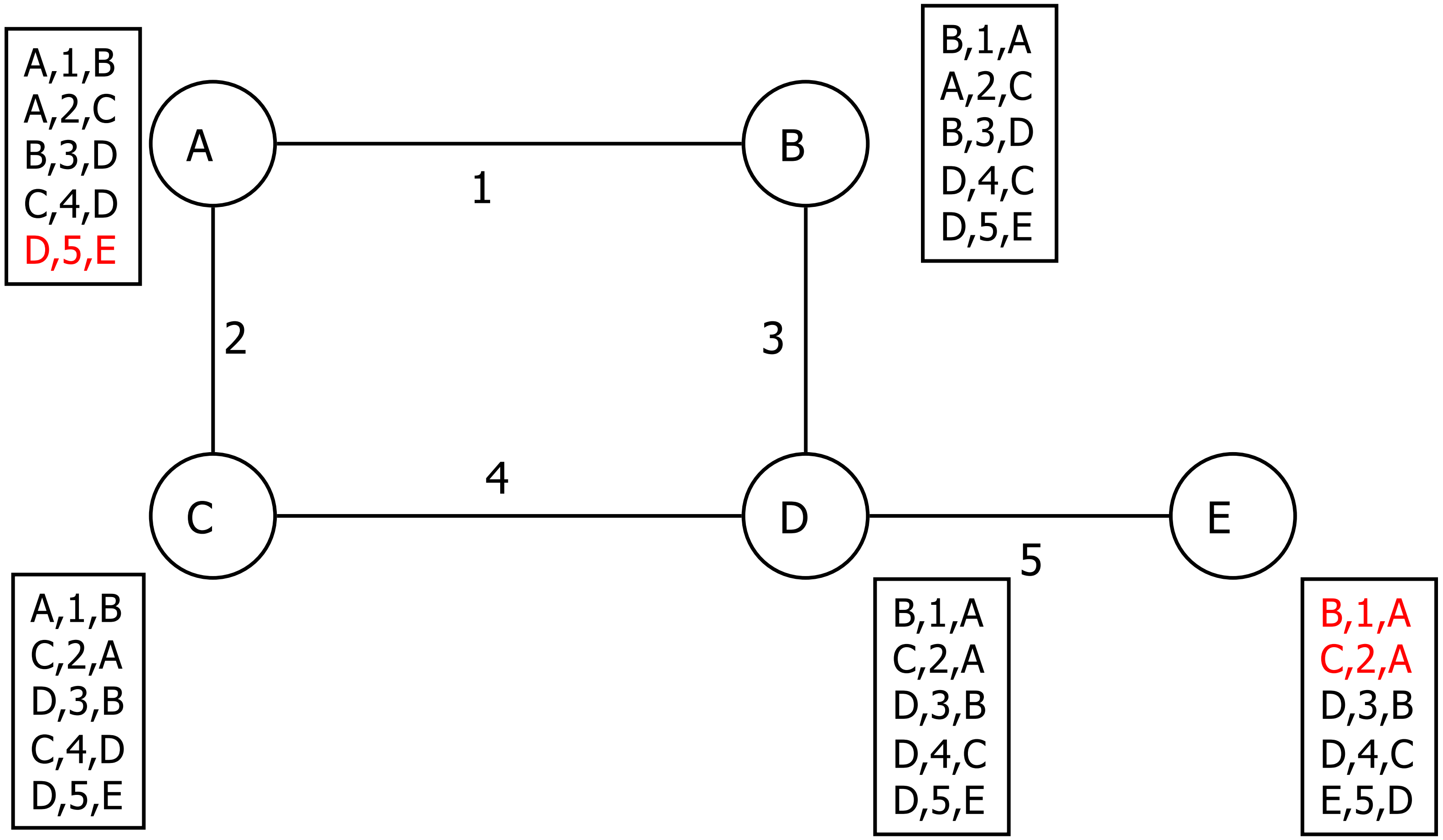
LSA Flooding: Example

- Example: received LSAs are used to update link states; LSAs are forwarded to all neighbors



LSA Flooding: Example

- Example: final update of link state information



Dijkstra Algorithm

- Main idea
  - Iteratively calculate the minimal spanning tree
  - The set of nodes N´ contains all nodes, to which shortest paths are already known
  - N´ is initialized with the **start node u**
  - In every iteration, the neighbor node v is added, that has the lowest path cost w from nodes in N´ and the edge to v has the shortest path:
    $D(u,v) = \min_{w \text{ in } N'}\{D(u,w) + c(w,v)\}$
  - [this is a specialization of the general recursion scheme $D(u,v) = \min_w\{D(u,w) + D(w,v)\}$, where the second part is an edge rather than a path
  - The algorithm terminates if N = N´

Dijkstra Algorithm

- Used data structures
  - Set of nodes N, start node u
  - N´ is the set of nodes to which shortest paths from u are known already
  - Costs c(v,w) between nodes v and w
    - Positive costs, if v and w are neighbors
    - 0 if v = w
    - ∞ else
  - Distance D(v) summarizes costs of the currently known shortest path from u to v
  - p(v) denotes the predecessor of v on the currently known shortest path from u to v

Dijkstra Algorithm

- Initialization

```
N´= {u};
For all $v \in N$: $D(v)=c(u,v)$;
```

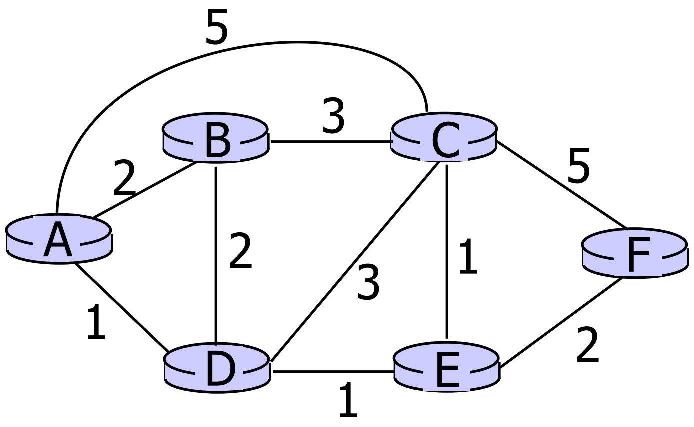
- Repeat until N´= N

```
Find $w \in N \setminus N´$, such that $\forall v \in N \setminus N´: D(w) \leq D(v)$;
/* w is a neighbor of a node in N´ with minimal costs */
$N´= N´ \cup \{w\}$;
For all $v \in N \setminus N´$
    If $D(w)+c(w,v) < D(v)$, then
     $D(v) = D(w)+c(w,v)$
     $p(v) = w$
     /* if a node can be reached with lower costs via w,
         the costs and its predecessor are updated */
```

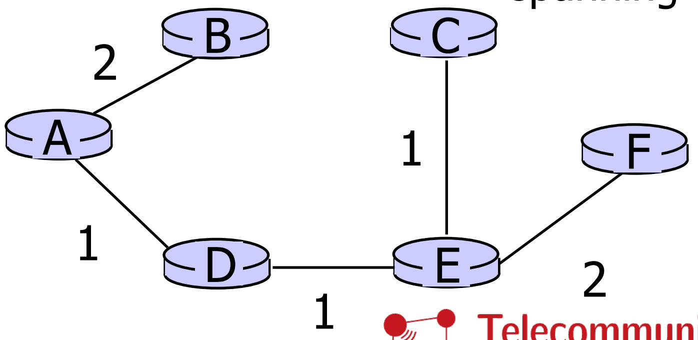
Dijkstra Algorithm: Example

		in each cell D(·), p(·)				
Step	N´	B	C	D	E	F
0	A	2,A	5,A	1,A	∞,-	∞,-
1	A,D	2,A	4,D	″	2,D	∞,-
2	A,D,B	″	4,D	″	2,D	∞,-
3	A,D,B,E	″	3,E	″	″	4,E
4	A,D,B,E,C	″	″	″	″	4,E
5	A,D,B,E,C,F	″	″	″	″	″

(Entries denotes as ″ cannot be changed anymore, because node is in N´)



resulting minimal spanning tree



Dijkstra Algorithm

- Calculation of the routing table
  - Routing table contains an entry for every destination node v including the next hop on the shortest path
  - How to derive the next hop node from vector p(v) with predecessors?
  - Recursive function
    nexthop(v) = if p(v)=u then v else nexthop(p(v))

Forward Search Algorithm

- More practical approach to the Dijkstra algorithm
  - Every node collects LSAs and directly calculates routing table
  - Most popular version: **Forward-Search-Algorithmus**
  - All entries are stores in the form (destination, costs, next hop) within two separate lists
    - Acknowledged list (similar to N´)
    - Tentative list (similar to all neighbors from nodes in N´)
  - nexthop(v) is the next node to reach a node v from the start node u
  - Values for c(w,v) can directly be read from the LSAs
  - Values for D(w) and nexthop(w) are stored in the two lists

Forward Search Algorithm

- Initialization

```
acknowledgedList = <(u,0,-)>, tentativeList = <>;
```

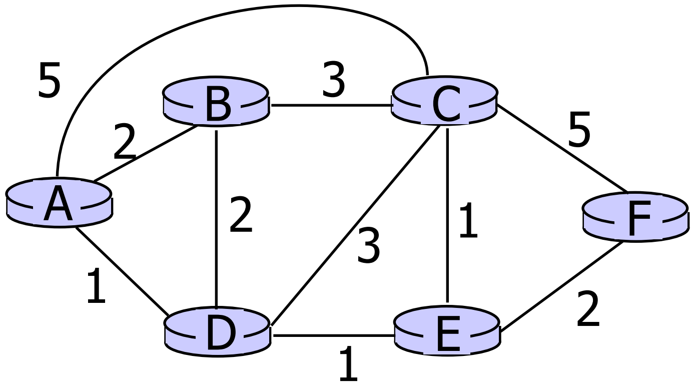
- Repeat

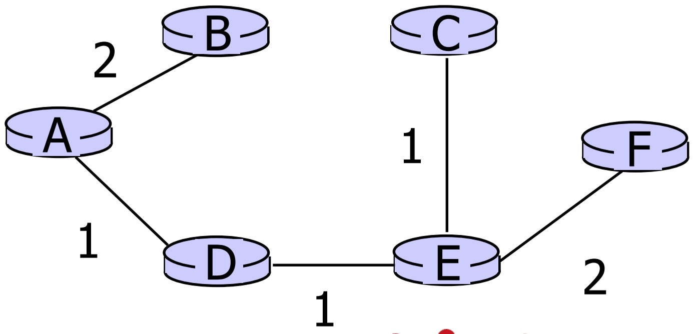
```
w = last node that was added to acknowledgedList
For all neighbors v of w
    If v is neither in achnowledgedList nor in tentativeList
        Add (v, D(w)+c(w,v), nexthop(w)) to tentativeList
    If v is in tentativeList and D(w)+c(w,v) < D(v)
        replace entry v in tentativeList by
        (v, D(w)+c(w,v), nexthop(w))
Move entry with lowest costs from tentativeList to acknowledgedList
```

- Until tentativeList is empty

Forward Search Algorithm: Example

Step	acknowledgedList	tentativeList
0	(A,0,-)	
1	(A,0,-)	(B,2,B), (C,5,C), (D,1,D)
2	(A,0,-), (D,1,D)	(B,2,B), (C,4,D), (E,2,D)
3	(A,0,-), (D,1,D), (B,2,B)	(C,4,D), (E,2,D)
4	(A,0,-), (D,1,D), (B,2,B), (E,2,D)	(C,3,D), (F,4,D)
5	(A,0,-), (D,1,D), (B,2,B), (E,2,D), (C,3,D)	(F,4,D)
6	(A,0,-), (D,1,D), (B,2,B), (E,2,D), (C,3,D), (F,4,D)	





Link State Routing: OSPF

- OSPF (Open Shortest Path First)
  - Widely used routing protocol, RFC 2328
  - Periodic flooding of link state advertisements
  - Exchange of LSAs directly via IP
    - Typ-1-LSA: costs of connections between routers, routers are identified by their IP address
    - Typ-2-LSA: networks connected to routers (i.e., aggregates of end systems)
  - OSPF supports five different cost metrics: normal, financial, throughput, reliability, delay
  - Metrics are set manually, minimal spanning tree is calculated for for each metric
  - In addition: authentication, support of hierarchies for large routing domains

Distance Vector Routing

Bellman Ford Algorithm

- Distributed shortest path search:
  - Every node tells its direct neighbors which other nodes can be reached at which cost (distance)
  - Initialization: only information about direct neighbors, with every exchange, paths to more nodes are learned and updated to converge to shortest paths
  - Makes use of property of shortest paths:
    $D(u,v) = \min_w\{D(u,w) + D(w,v)\}$
  - Here, for the first part just a single edge is used:
    $D(u,v) = \min_w\{c(u,w) + D(w,v)\}$
  - Whenever a new shortest path to a destination has been found, this is distributed again to all neighbors
  - Algorithm terminates, if no changes are made after receiving an update (the routing algorithm converges)

Bellman Ford Algorithm

- Data structures
  - Cost c(v,w) between nodes v and w
    - Positive cost if v and w are neighbors
    - 0 if v = w
    - ∞ else
  - Distance $D_u(v)$: shortest distance from u to v as known so far
  - Distance vector $\mathbf{D}_u = (D_u(v), v \in N)$
  - Every node u has
    - Distances $D_u(v)$ for all nodes $v \in N$
    - Local copies $\underline{D}_w(v)$ of the distances $D_w(v)$ from all neighbors w to all $v \in N$
    - $\text{nexthop}_u(v)$, to reach v from u for all nodes $v \in N$

Bellman Ford Algorithm

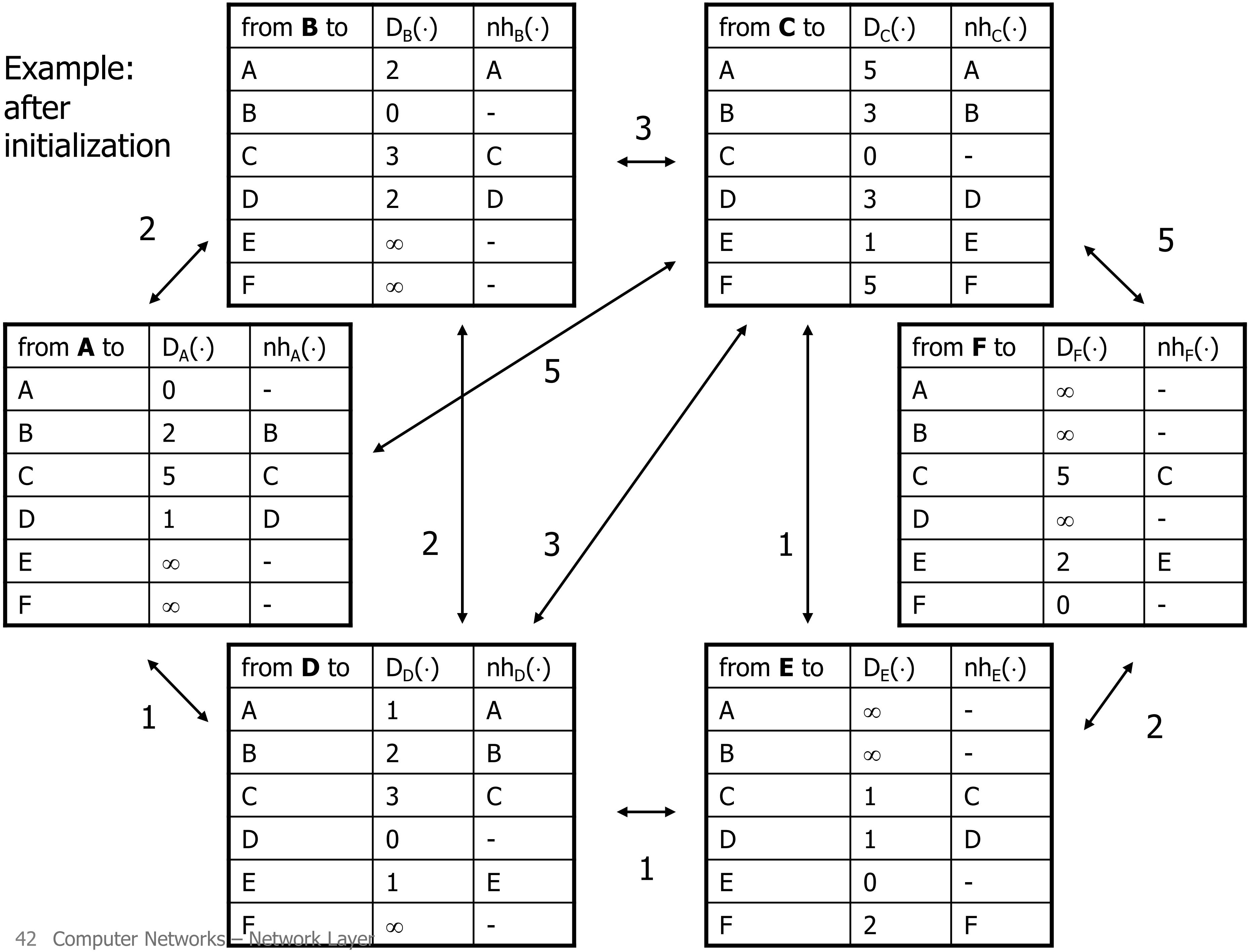
- Initialization (at node u)

```
For all $v \in N$:
    If v is neighbor of u: $D_u(v)=c(u,v)$; $nexthop_u(v)=v$;
    Else $D_u(v)=\infty$; $nexthop_u(v)=$ -;
For all neighbors w:
    For all $v \in N$: $\underline{D}_w(v)=\infty$;
    Send $\mathbf{D}_u$ to w;
```

- Repeat

```
Wait for
    Change of costs to neighbor w ($\mathbf{D}_u$ changes) or
    Reception of distance vector $\mathbf{D}_w$ from neighbor w ($\underline{\mathbf{D}}_w$ changes)
For all $v \in N$:
    $D_u(v) = \min_w\{c(u,w)+\underline{D}_w(v)\}$;
    $nexthop_u(v)$ = node w from this minimum
If $\mathbf{D}_u$ changes: send $\mathbf{D}_u$ to all direct neighbors
```

Example: after initialization



from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	2	A
B	0	-
C	3	C
D	2	D
E	∞	-
F	∞	-

from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	5	A
B	3	B
C	0	-
D	3	D
E	1	E
F	5	F

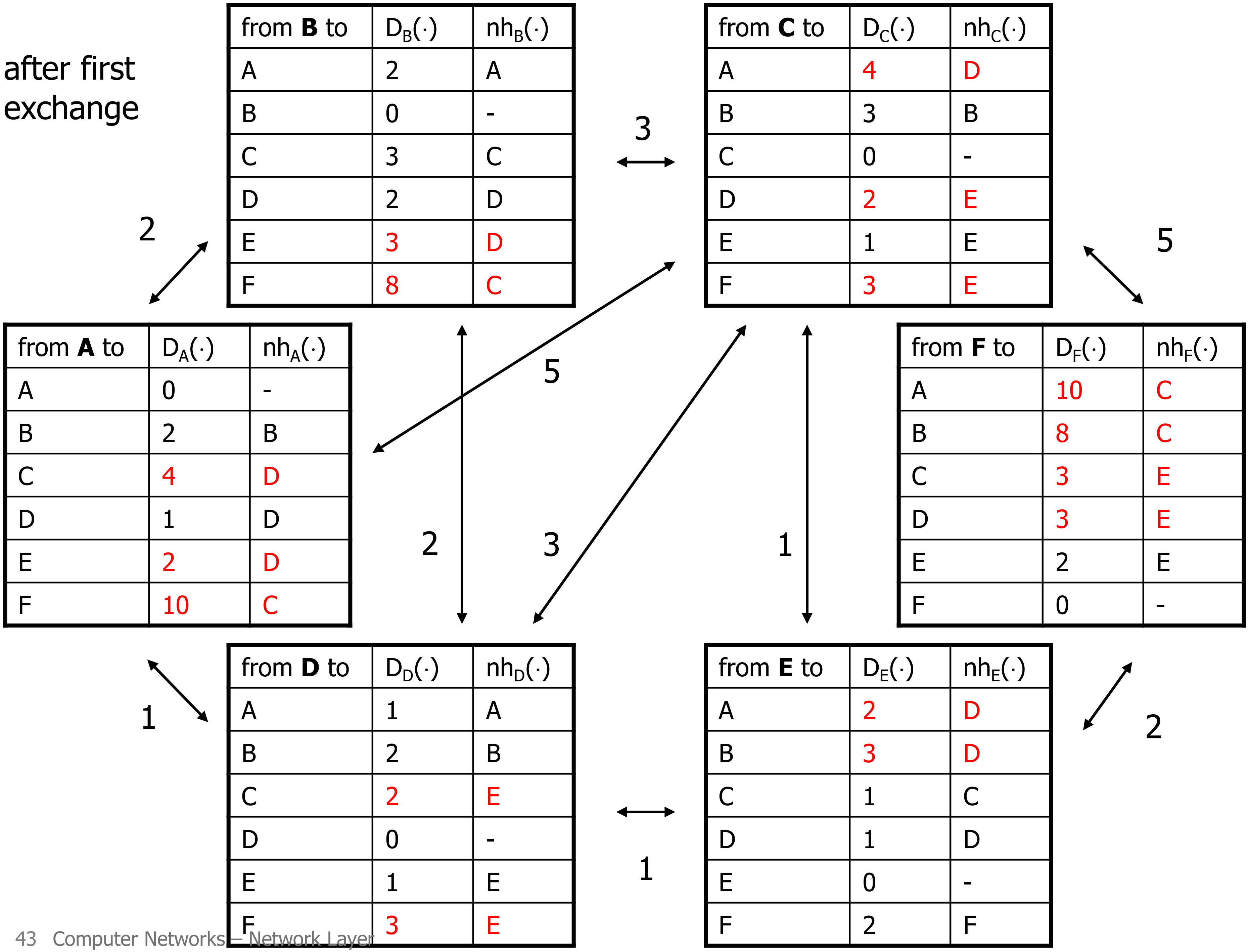
from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	2	B
C	5	C
D	1	D
E	∞	-
F	∞	-

from **F** to	$D_F(\cdot)$	$nh_F(\cdot)$
A	∞	-
B	∞	-
C	5	C
D	∞	-
E	2	E
F	0	-

from **D** to	$D_D(\cdot)$	$nh_D(\cdot)$
A	1	A
B	2	B
C	3	C
D	0	-
E	1	E
F	∞	-

from **E** to	$D_E(\cdot)$	$nh_E(\cdot)$
A	∞	-
B	∞	-
C	1	C
D	1	D
E	0	-
F	2	F

after first exchange



from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	2	A
B	0	-
C	3	C
D	2	D
E	3	D
F	8	C

from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	4	D
B	3	B
C	0	-
D	2	E
E	1	E
F	3	E

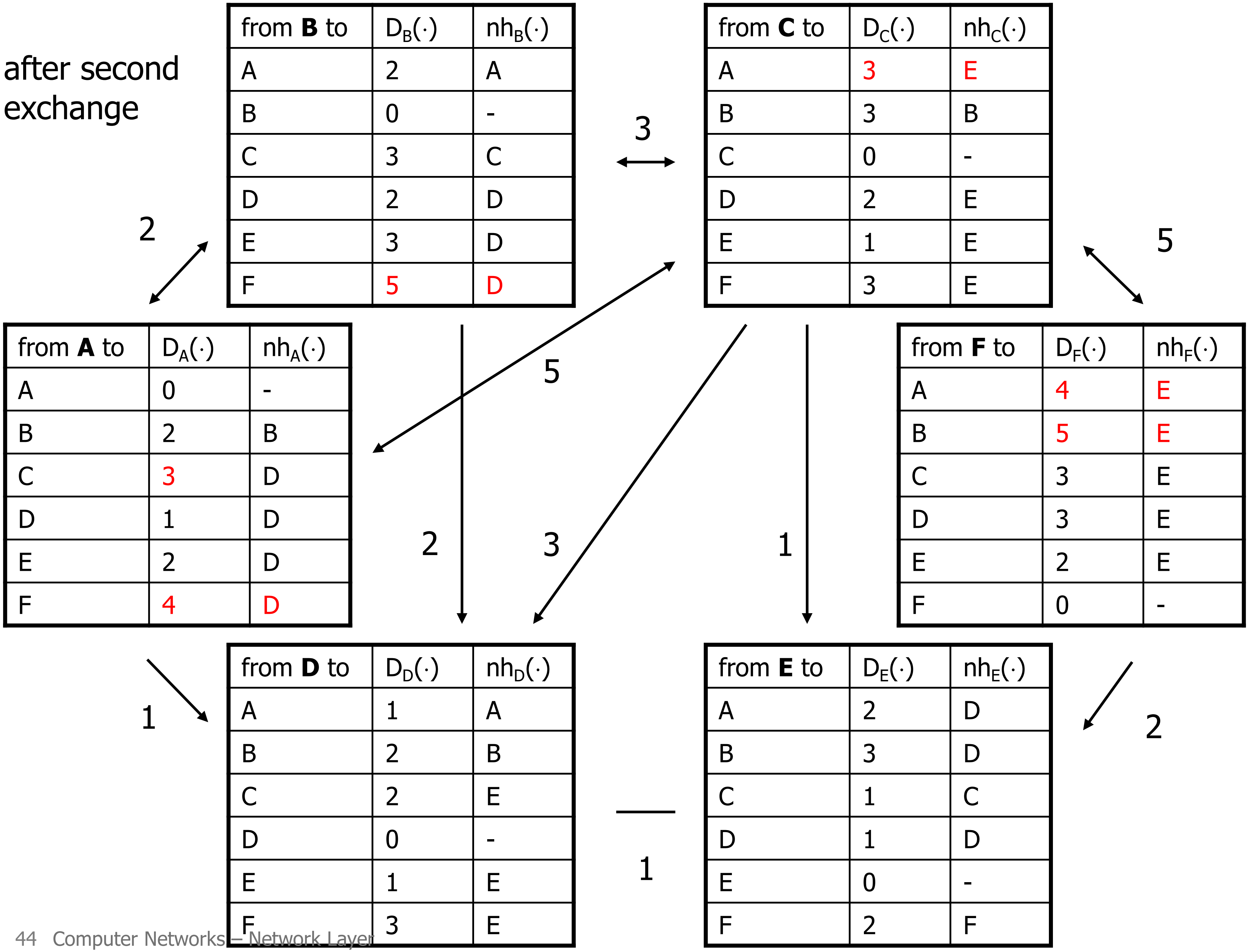
from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	2	B
C	4	D
D	1	D
E	2	D
F	10	C

from **F** to	$D_F(\cdot)$	$nh_F(\cdot)$
A	10	C
B	8	C
C	3	E
D	3	E
E	2	E
F	0	-

from **D** to	$D_D(\cdot)$	$nh_D(\cdot)$
A	1	A
B	2	B
C	2	E
D	0	-
E	1	E
F	3	E

from **E** to	$D_E(\cdot)$	$nh_E(\cdot)$
A	2	D
B	3	D
C	1	C
D	1	D
E	0	-
F	2	F

after second exchange



from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	2	A
B	0	-
C	3	C
D	2	D
E	3	D
F	5	D

from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	3	E
B	3	B
C	0	-
D	2	E
E	1	E
F	3	E

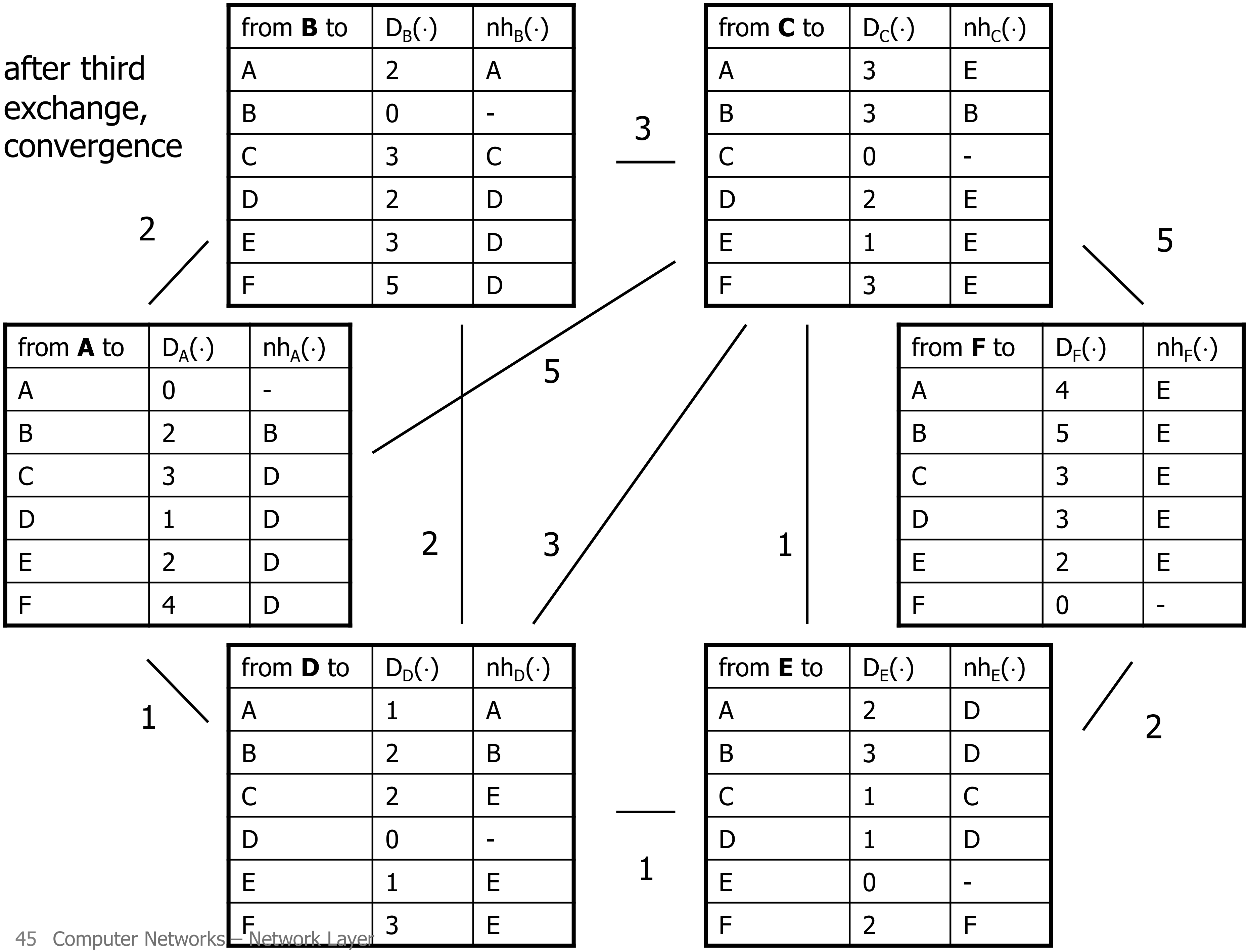
from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	2	B
C	3	D
D	1	D
E	2	D
F	4	D

from **F** to	$D_F(\cdot)$	$nh_F(\cdot)$
A	4	E
B	5	E
C	3	E
D	3	E
E	2	E
F	0	-

from **D** to	$D_D(\cdot)$	$nh_D(\cdot)$
A	1	A
B	2	B
C	2	E
D	0	-
E	1	E
F	3	E

from **E** to	$D_E(\cdot)$	$nh_E(\cdot)$
A	2	D
B	3	D
C	1	C
D	1	D
E	0	-
F	2	F

after third exchange, convergence



from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	2	A
B	0	-
C	3	C
D	2	D
E	3	D
F	5	D

from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	3	E
B	3	B
C	0	-
D	2	E
E	1	E
F	3	E

from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	2	B
C	3	D
D	1	D
E	2	D
F	4	D

from **F** to	$D_F(\cdot)$	$nh_F(\cdot)$
A	4	E
B	5	E
C	3	E
D	3	E
E	2	E
F	0	-

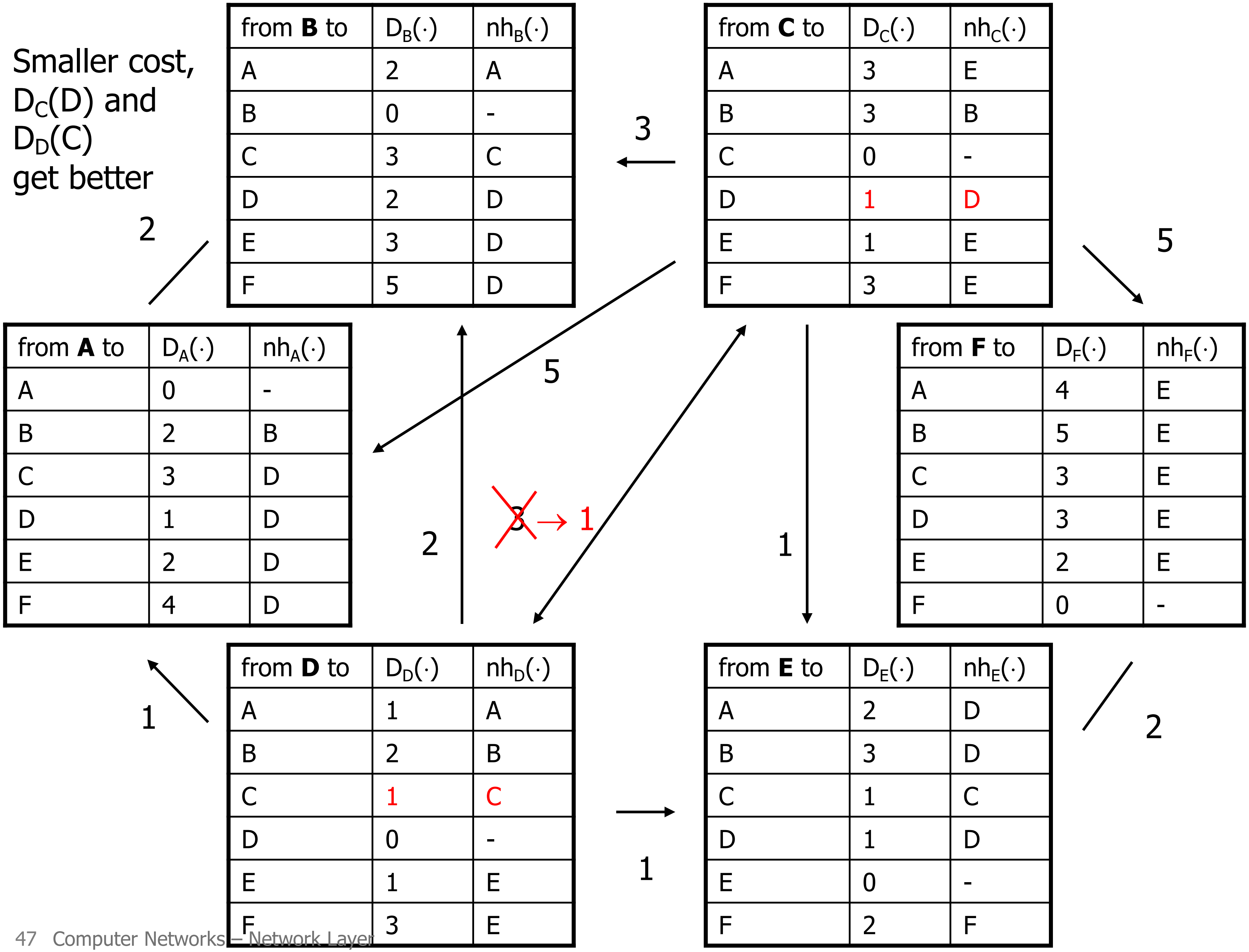
from **D** to	$D_D(\cdot)$	$nh_D(\cdot)$
A	1	A
B	2	B
C	2	E
D	0	-
E	1	E
F	3	E

from **E** to	$D_E(\cdot)$	$nh_E(\cdot)$
A	2	D
B	3	D
C	1	C
D	1	D
E	0	-
F	2	F

Distance Vector Routing

- Behavior in case of topology changes
  - The Bellman Ford algorithm also works in case of topology changes
  - If connection costs are getting smaller, the algorithm converges quickly: **good news travel fast**
  - If connection costs are increasing, cycles may appear temporarily, and it may take some time to convergence: **bad news travel slowly**

Smaller cost, $D_C(D)$ and $D_D(C)$ get better



from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	2	A
B	0	-
C	3	C
D	2	D
E	3	D
F	5	D

from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	3	E
B	3	B
C	0	-
D	1	D
E	1	E
F	3	E

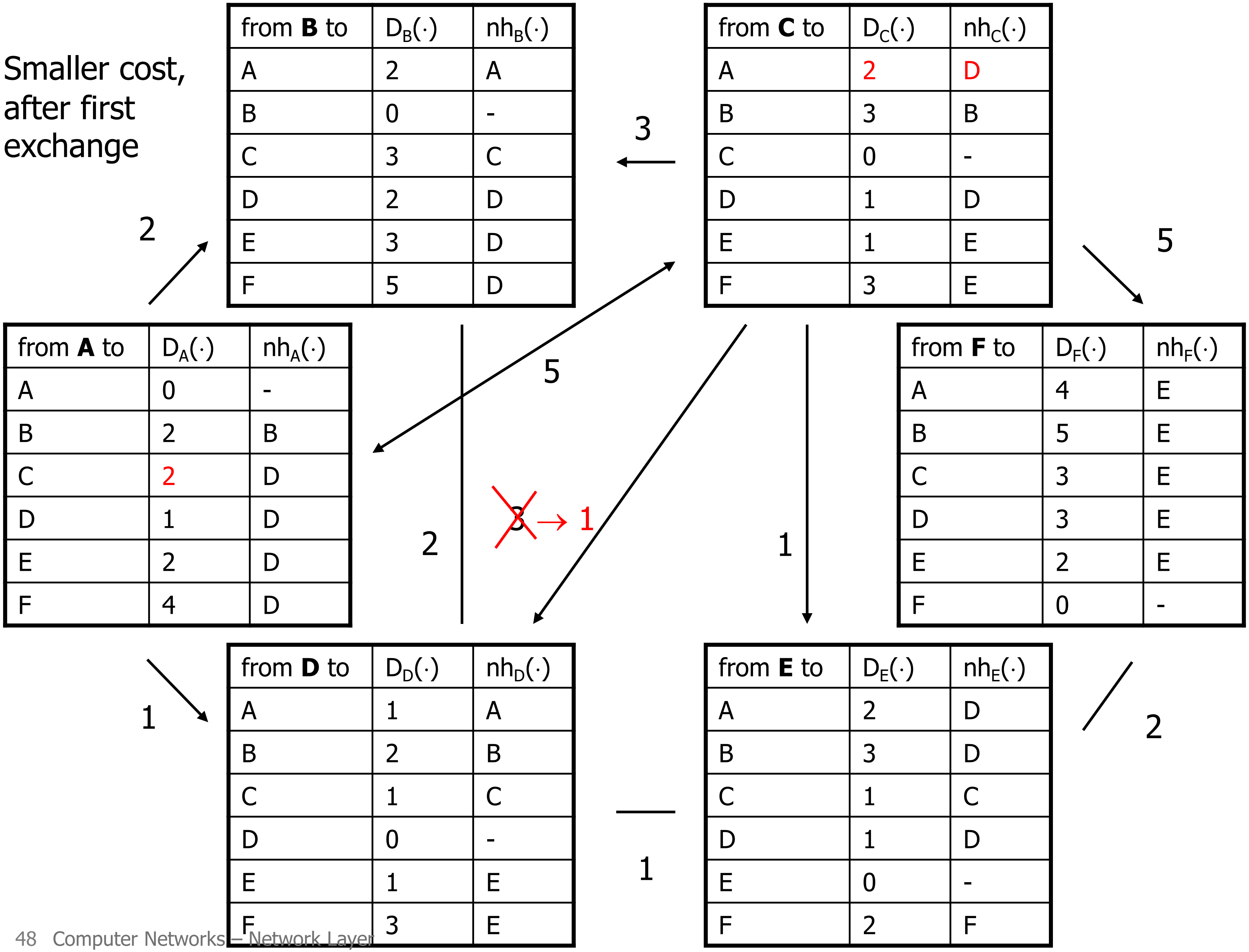
from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	2	B
C	3	D
D	1	D
E	2	D
F	4	D

from **F** to	$D_F(\cdot)$	$nh_F(\cdot)$
A	4	E
B	5	E
C	3	E
D	3	E
E	2	E
F	0	-

from **D** to	$D_D(\cdot)$	$nh_D(\cdot)$
A	1	A
B	2	B
C	1	C
D	0	-
E	1	E
F	3	E

from **E** to	$D_E(\cdot)$	$nh_E(\cdot)$
A	2	D
B	3	D
C	1	C
D	1	D
E	0	-
F	2	F

Smaller cost, after first exchange



from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	2	A
B	0	-
C	3	C
D	2	D
E	3	D
F	5	D

from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	2	D
B	3	B
C	0	-
D	1	D
E	1	E
F	3	E

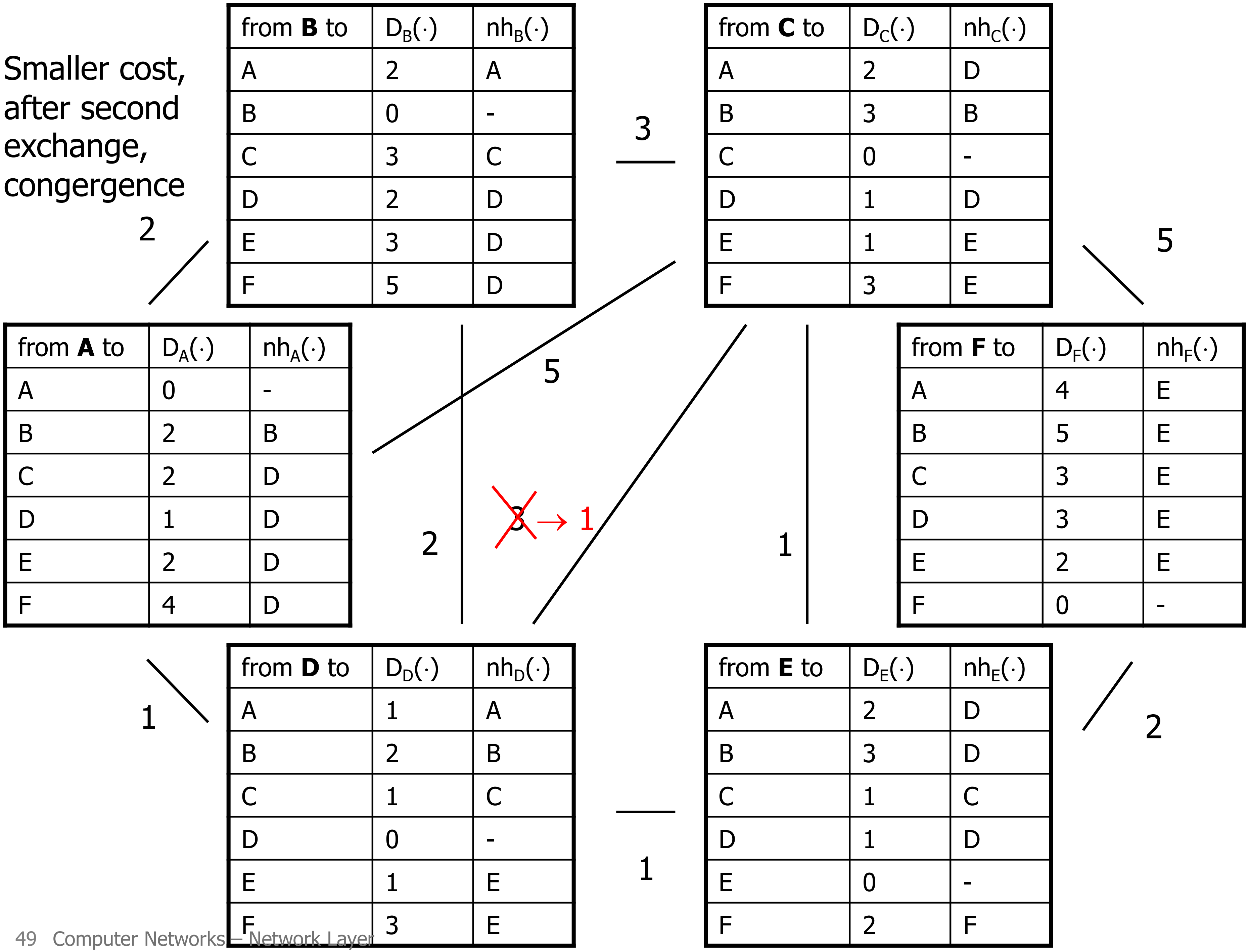
from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	2	B
C	2	D
D	1	D
E	2	D
F	4	D

from **F** to	$D_F(\cdot)$	$nh_F(\cdot)$
A	4	E
B	5	E
C	3	E
D	3	E
E	2	E
F	0	-

from **D** to	$D_D(\cdot)$	$nh_D(\cdot)$
A	1	A
B	2	B
C	1	C
D	0	-
E	1	E
F	3	E

from **E** to	$D_E(\cdot)$	$nh_E(\cdot)$
A	2	D
B	3	D
C	1	C
D	1	D
E	0	-
F	2	F

Smaller cost, after second exchange, congergence



from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	2	A
B	0	-
C	3	C
D	2	D
E	3	D
F	5	D

from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	2	D
B	3	B
C	0	-
D	1	D
E	1	E
F	3	E

from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	2	B
C	2	D
D	1	D
E	2	D
F	4	D

from **F** to	$D_F(\cdot)$	$nh_F(\cdot)$
A	4	E
B	5	E
C	3	E
D	3	E
E	2	E
F	0	-

from **D** to	$D_D(\cdot)$	$nh_D(\cdot)$
A	1	A
B	2	B
C	1	C
D	0	-
E	1	E
F	3	E

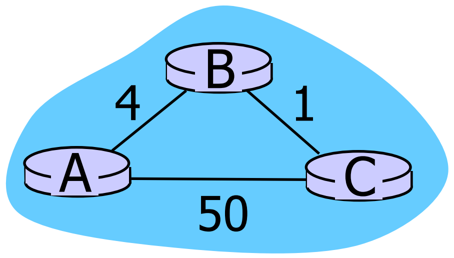
from **E** to	$D_E(\cdot)$	$nh_E(\cdot)$
A	2	D
B	3	D
C	1	C
D	1	D
E	0	-
F	2	F

Distance Vector Routing

- Larger cost
  - Initial tables

from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	4	A
B	0	-
C	1	C

from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	4	B
C	5	B



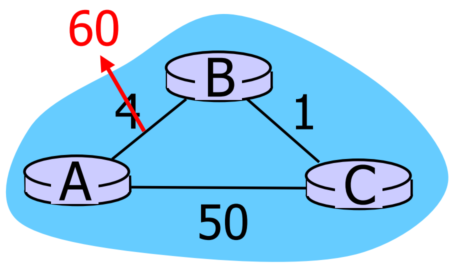
from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	5	B
B	1	B
C	0	-

Distance Vector Routing

- Larger cost
  - Increased cost

from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	6	C
B	0	-
C	1	C

from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	51	C
C	50	C



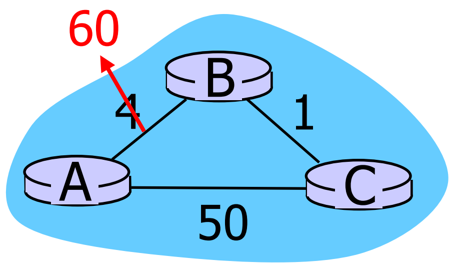
from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	5	B
B	1	B
C	0	-

Distance Vector Routing

- Larger cost
  - After first exchange

from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	6	C
B	0	-
C	1	C

from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	51	C
C	50	C



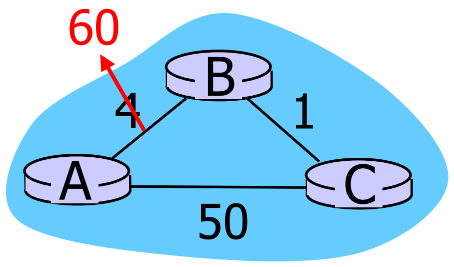
from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	7	B
B	1	B
C	0	-

Distance Vector Routing

- Larger cost
  - After second exchange

from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	8	C
B	0	-
C	1	C

from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	51	C
C	50	C



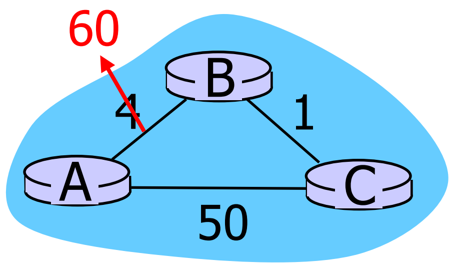
from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	7	B
B	1	B
C	0	-

Distance Vector Routing

- Larger cost
  - After third exchange

from **B** to	$D_B(\cdot)$	$nh_B(\cdot)$
A	8	C
B	0	-
C	1	C

from **A** to	$D_A(\cdot)$	$nh_A(\cdot)$
A	0	-
B	51	C
C	50	C



from **C** to	$D_C(\cdot)$	$nh_C(\cdot)$
A	9	B
B	1	B
C	0	-

  - until we reach 50 ...

Count to Infinity Problem

- Outdated information in distributed routing tables may contain a cyclic path
- The slow iteration only ends, when the costs of the alternate path have been reached
- Solutions
  - Limit maximum costs (e.g., 16)
  - **Poisoned reverse**: if the shortest path from u to v goes via next hop w, u sends to w the infinite cost for the distance from u to v
  - Poisoned reverse may avoid cycles of length 3 but not larger cycles

Routing: Summary

- Link State Routing
  - Centralized
  - Complexity of Dijkstra algorithm is O(n^2) for n nodes
  - Efficient implementations achieve O(n log n)
  - Scalability limited
  - Message overhead: O(ne) for n nodes and e edges
  - Robustness: does not tolerate faulty / malicious information
  - Dynamic metrics may lead to instability (thus not used)
- Distance Vector Routing
  - Distributed
  - Convergence problems in case of cycles
  - Scalability limited
  - Robustness: error propagation if faulty / malicious information is provided
  - Dynamic metrics not supported

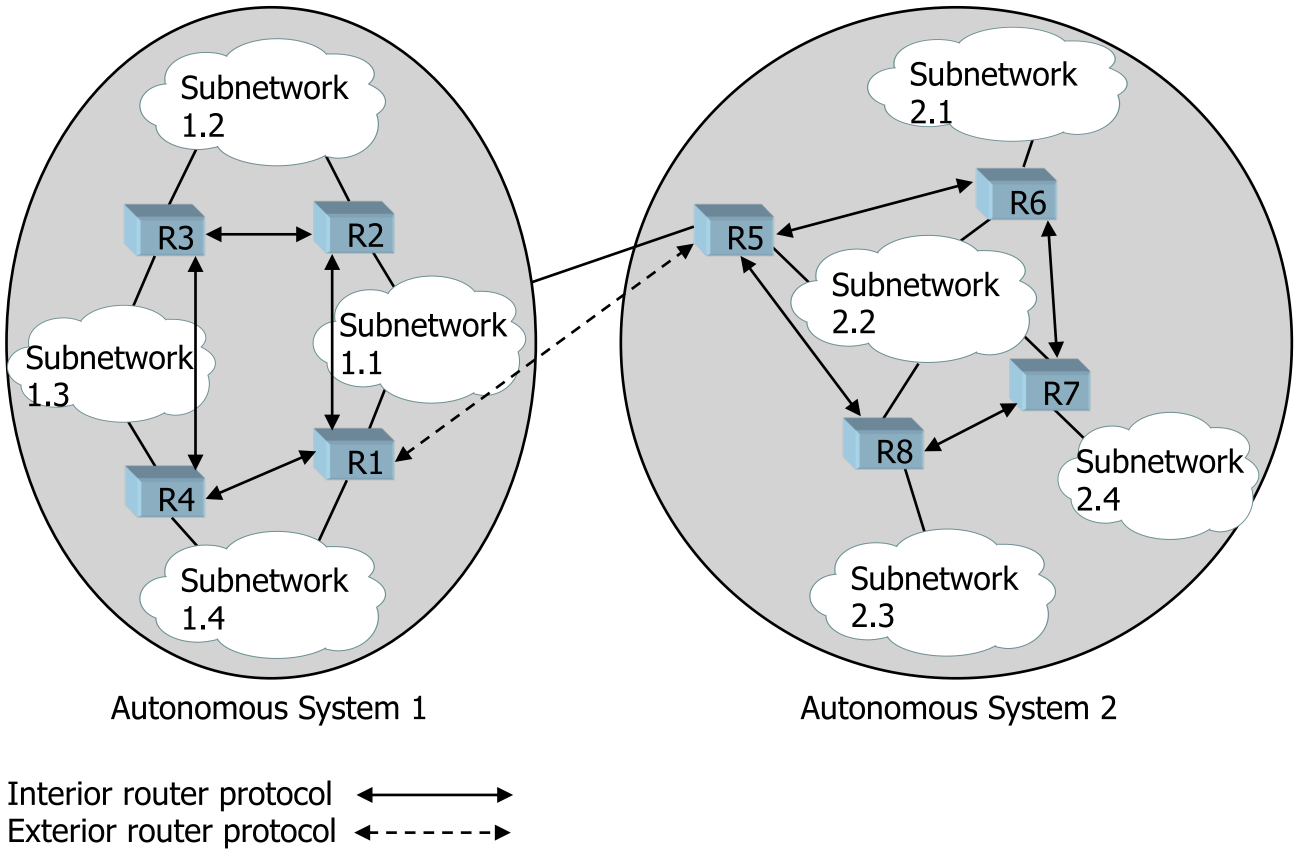
Interdomain Routing

Interdomain Routing

- Routing hierarchy
  - Within a routing domain
    - Intradomain routing, **Interior Gateway Protocol (IGP)**, e.g., RIP, OSPF
  - Between routing domains
    - Interdomain routing, **Exterior Gateway Protocol (EGP)**, e.g., BGP: **Border Gateway Protocol**, RFC 1771
    - Routing domains: autonomous systems (AS)
      - **Stub AS**: has only one connection to another AS
      - **Multihomed AS**: has multiple connections to other AS, albeit no transit supported
      - **Transit AS**: has multiple connections to other AS and allows transit of packets
    - Transit AS identified by an AS number (centralized, 16 bit in BGP)
    - At least one router in an AS is a gateway and performs the EGP functions

Interdomain Routing

- Example for 2 AS
  - R1 and R5 are gateways



Interdomain Routing

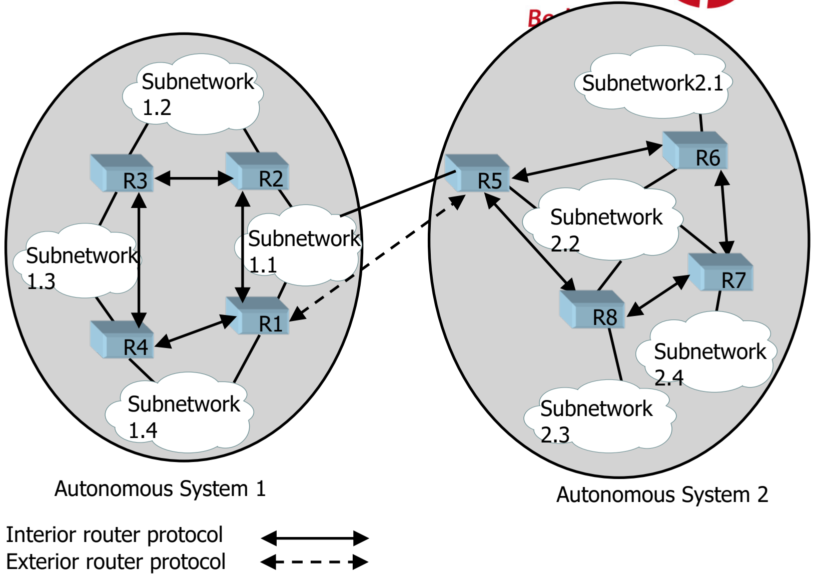
- Path-based routing
  - Exchange of entire routing paths
  - Internal costs are ignored (impossible due to size of networks and nonuniformly used cost metrics)
  - Router announces paths, which other routers may use
  - Router selects a path based on individual rules (e.g., path length, AS in path)
  - Cycles can be identified (router appears twice in path)
  - Paths can be cancelled

Border Gateway Protocol (BGP)

- Functions of BGP
  - Gateways exchange advertisements in BPG sessions via TCP
  - Advertisements include paths that include gateways to the individual networks
  - A path consists of
    - List of networks (address and prefix, CIDR) in an AS
    - Sequence of AS numbers to this AS
    - IP address of the gateway
  - When forwarding a path, the router adds its own AS to the beginning of the path and itself as a next hop
  - This way, it allows to route traffic via itself
  - A path that contains this router will be dropped

Border Gateway Protocol (BGP)

- Example
  - R1 creates routing table for AS1 with IGP
  - R1 sends advertisement to R5 in AS2
    - Networks: all subnets in AS1
    - Sequence of AS numbers: {AS1}
    - Next hop: IP address of R1
  - Assumption: R5 is also neighbor of R9 in AS3
  - R5 forwards the information from R1 to R9 in an advertisement
    - Networks: all subnets in AS1
    - Sequence of AS numbers: {AS2,AS1}
    - Next hop: IP address of R5
  - R9 decides whether the received path is its preferred path to AS1, and further forwards the path {AS3, AS2,AS1}



Border Gateway Protocol (BGP)

- Example configuration
  - A, B, C: provider network, transit AS
  - X: customer, multihomed AS
  - W, Y: customers, stub AS
  - X does not want transit traffic, thus, it does not send advertisements (in most cases, it actually cannot because it has no AS number)
  - A sends path AW to B
  - B sends path BAW to X
  - B sends path BAW not to C, because it may not want to route traffic via C