Computer Networks (2025-2026)

Part 4: Network Layer

Jeroen Famaey

Andrei Belogaev

{jeroen.famaey, andrei.belogaev}@uantwerpen.be

University of Antwerp
Faculty of Science

Routing algorithms

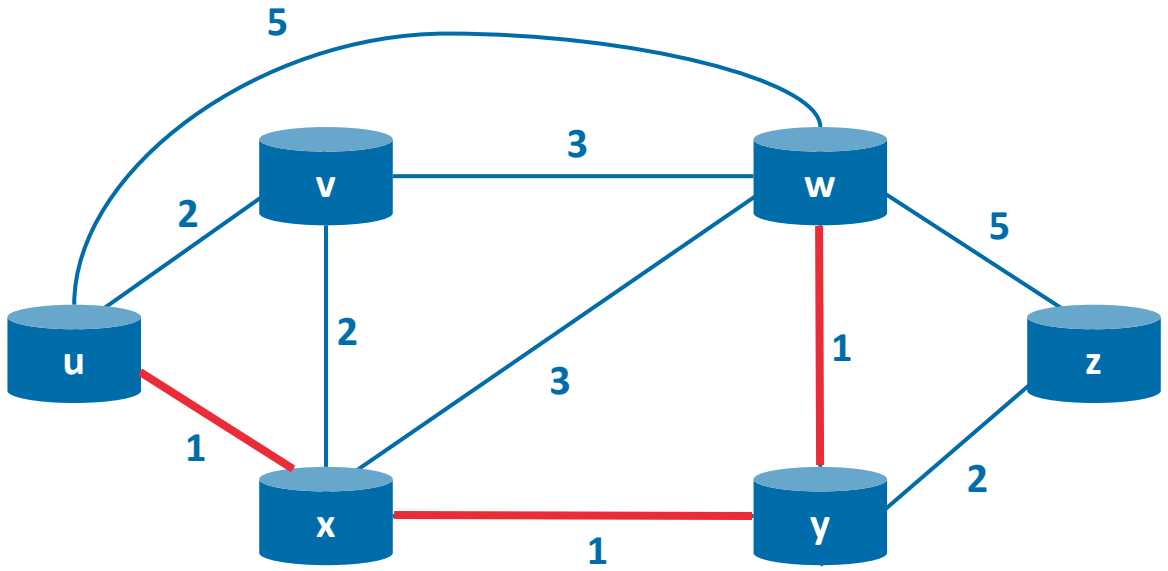
Routing algorithms

- **Goal:** Determine good paths (or routes), from senders to receivers, through the network of routers
- Routing problems are formulated using graph theory
  - A graph $G = (N, E)$ consists of a set of nodes (also called vertices) ***N*** and a set of edges ***E***
  - In network-layer routing, the nodes are routers, and the edges are physical links between these routers
  - Each edge between any pair of nodes (x, y) has a cost $c(x, y)$ associated with it (e.g., representing its length, monetary cost)
  - A routing algorithm tries to find the **least-cost path** of edges (x_1, x_2, ..., x_p) between the source x_1 and destination x_p

Example:

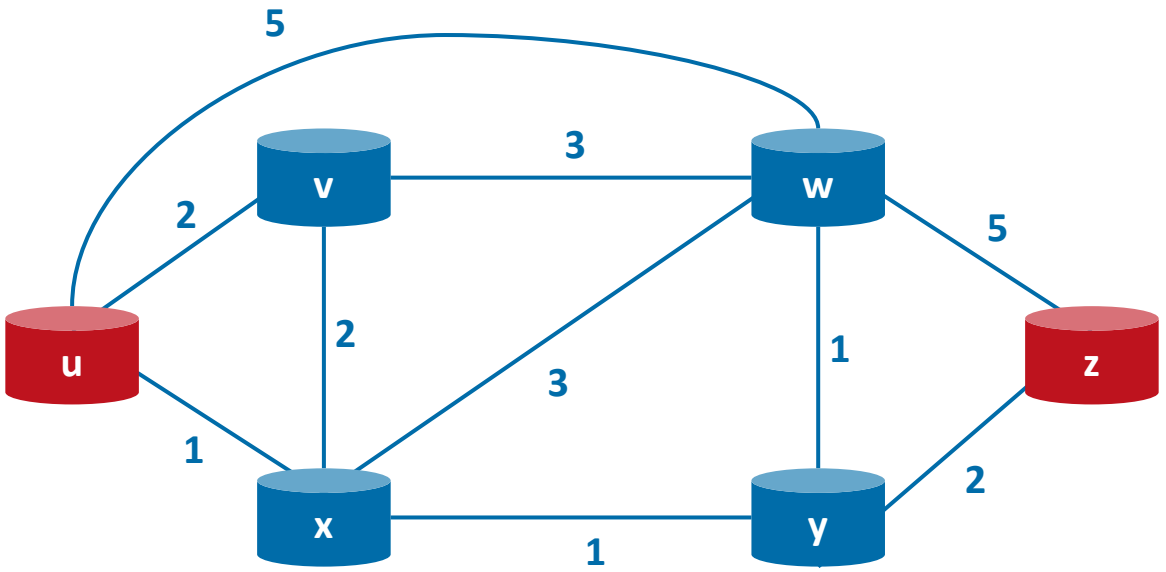
Least-cost path from u to w is (u, x, y, w)

Cost: $c(u, x) + c(x, y) + c(y, w) = 3$



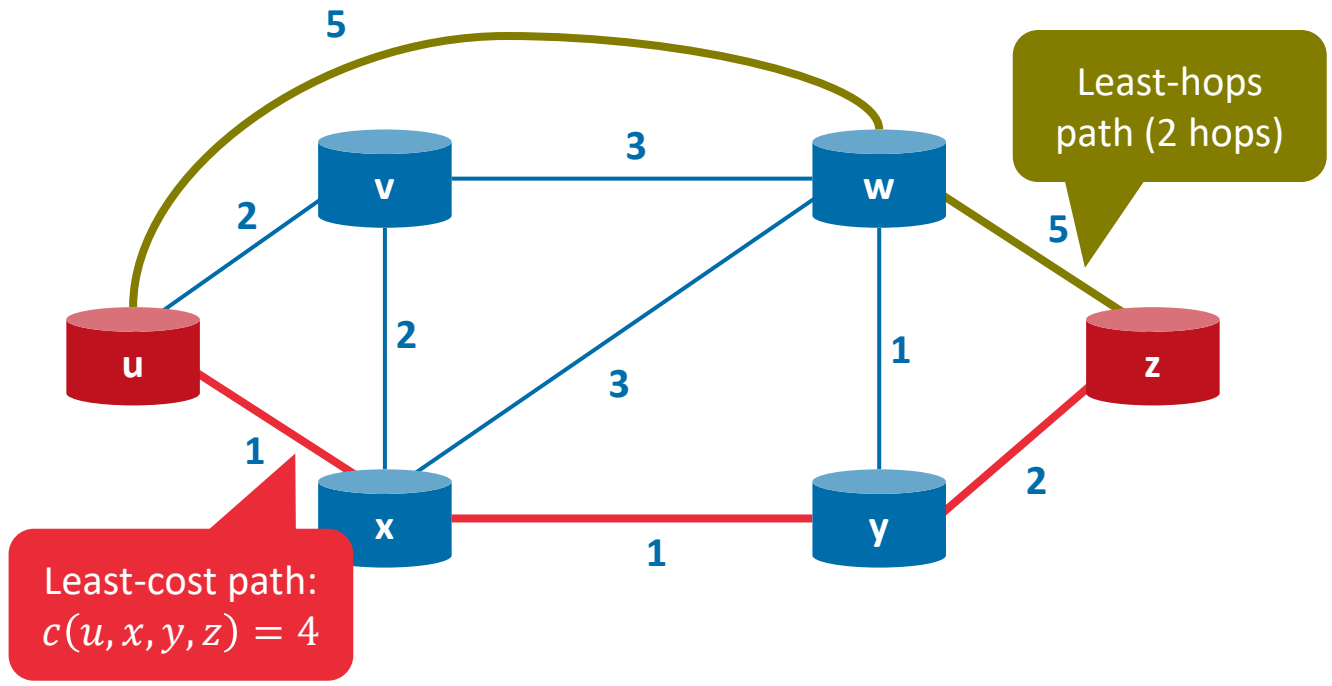
Exercise: Find the least-cost path

Exercise: What is the **least-cost path** from node u to node z? What is the **least-hops path**?



Solution: Find the least-cost path

Exercise: What is the **least-cost path** from node u to node z? What is the **shortest path**?



Classification of routing algorithms

Centralized (use complete global knowledge)

Static (routes change very slowly over time)

Load-sensitive (adapt to changing traffic load)

Decentralized (iterative, distributed)

Dynamic (routes adapt to real-time data)

Load-insensitive (link cost unaffected by load)

The (centralized) link-state (LS) routing algorithm

- Centralized routing algorithm that assumes the entire network topology and edge costs are known
- Can apply known least-cost path algorithms (e.g., Dijkstra's algorithm)
- Notations
  - N: set of all nodes
  - s: source node
  - $D(v)$: cost of the currently known least-cost path from the source node s to destination v
  - $p(v)$: previous node along the current least-cost path from the source s to v
  - N': subset of nodes for which the least-cost path from the source s is known
- **Dijkstra's algorithm:**

Initialization: $N' = \{s\}$

$\forall v \in N: D(v) = c(s,v)$

$c(s,v) = \infty$ if s and v are not direct neighbours

Repeat: $w = \underset{v \notin N'}{\operatorname{argmin}} D(v)$

$N' = N' \cup \{w\}$

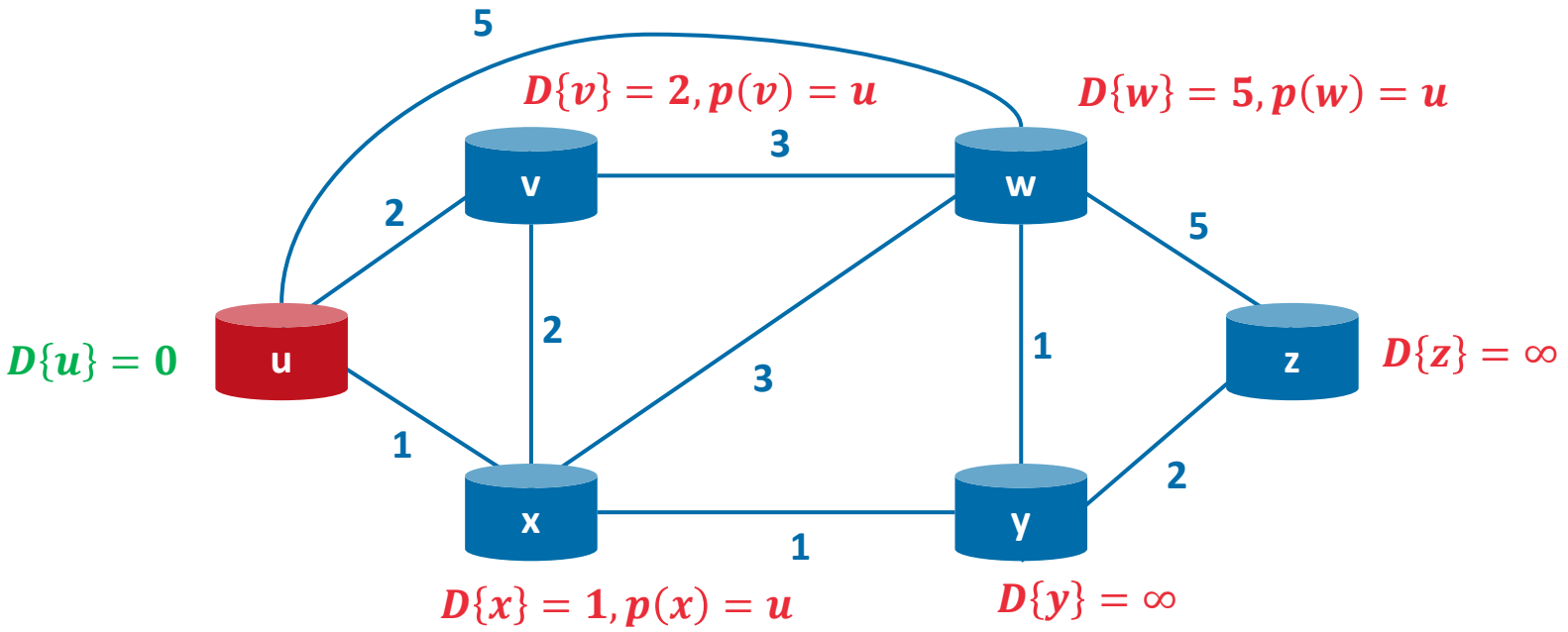
$\forall$ neighbours v *of* $w \wedge v \notin N': D(v) = \min\big(D(v), D(w) + c(w,v)\big)$

Until: $N' = N$

Example: Dijkstra's algorithm (calculate least-cost paths from source u)

Initialization

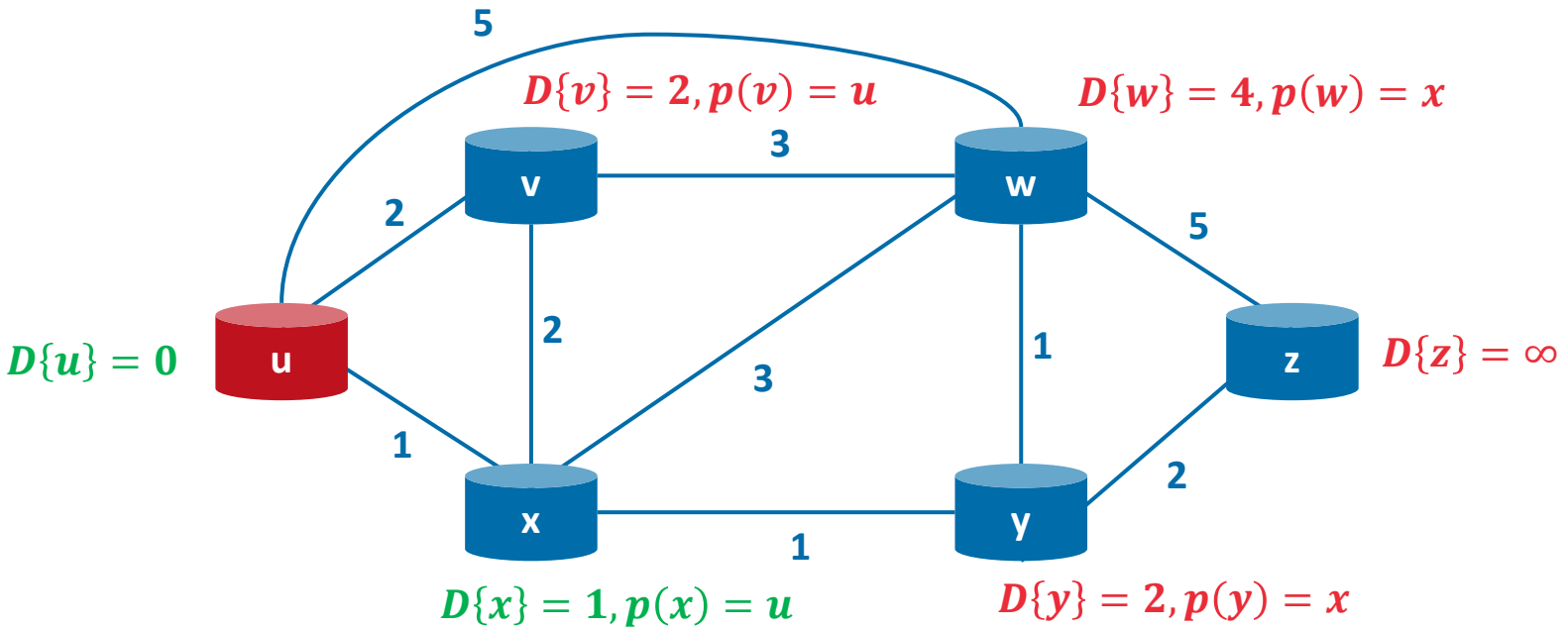
$N' = \{u\}$



Example: Dijkstra's algorithm (calculate least-cost paths from source u)

Iteration 1

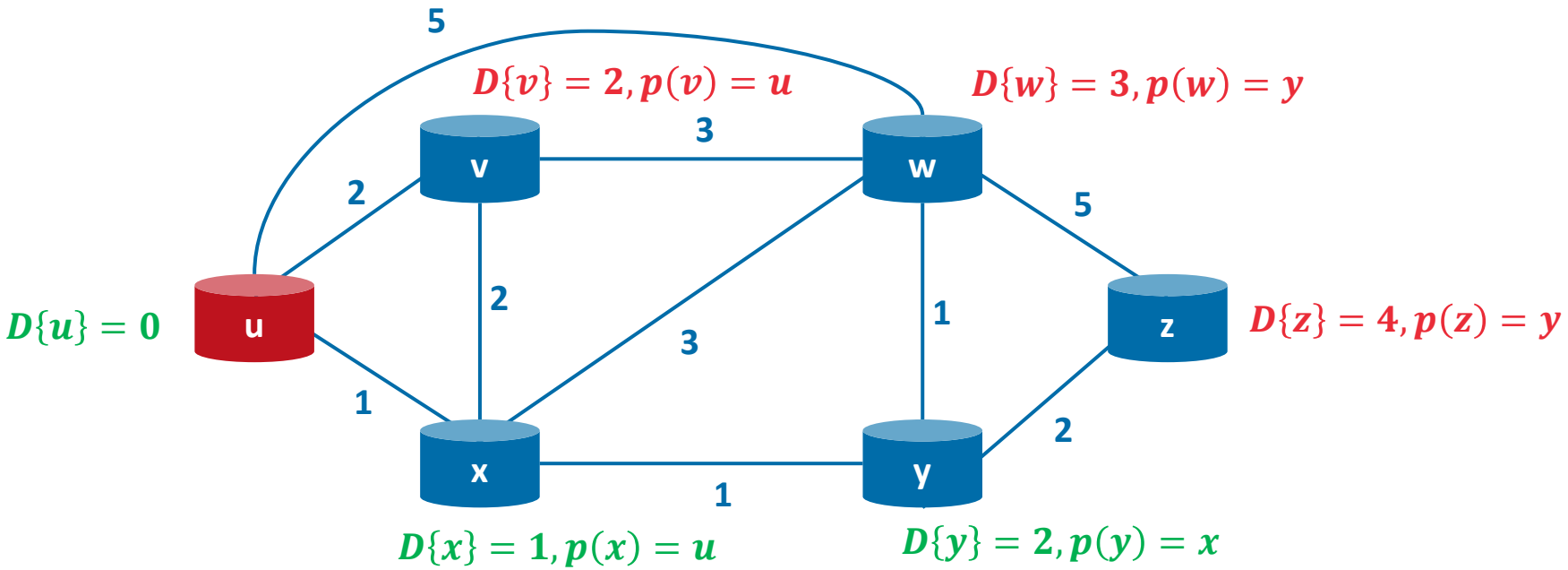
$N' = \{u, x\}$



Example: Dijkstra's algorithm (calculate least-cost paths from source u)

Iteration 2

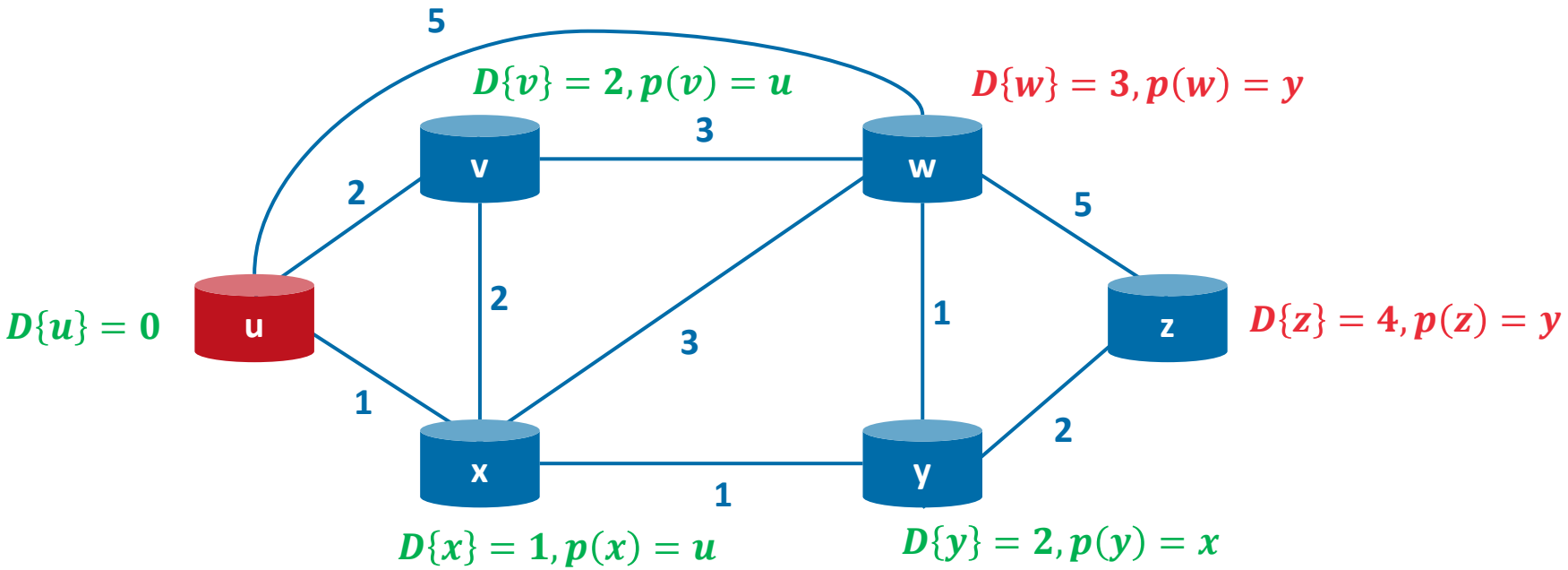
$N' = \{u, x, y\}$



Example: Dijkstra's algorithm (calculate least-cost paths from source u)

Iteration 3

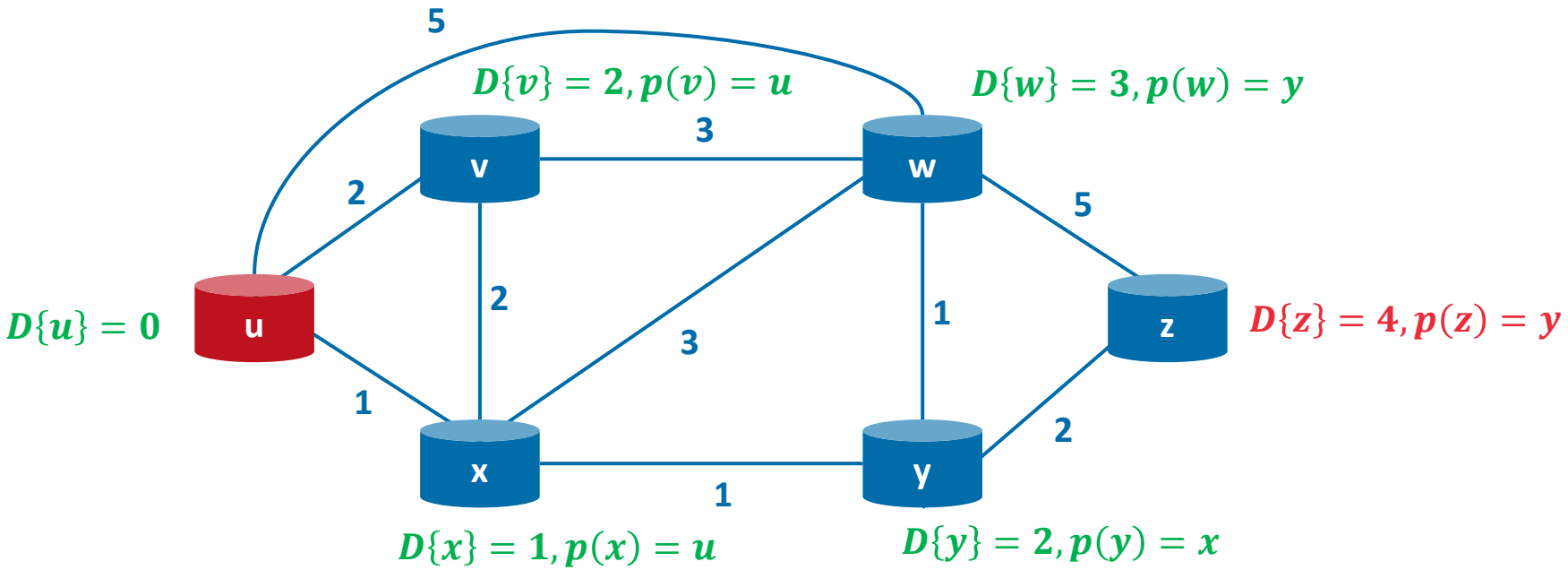
$N' = \{u, x, y, v\}$



Example: Dijkstra's algorithm (calculate least-cost paths from source u)

Iteration 4

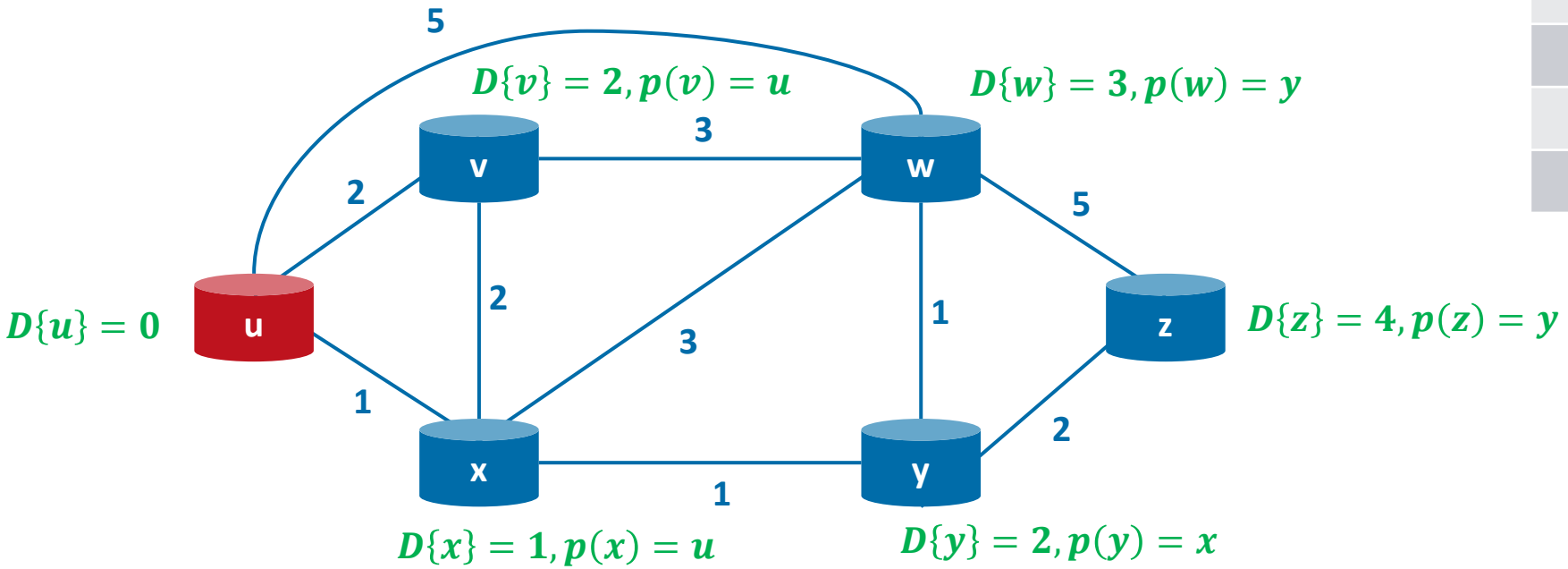
$N' = \{u, x, y, v, w\}$



Example: Dijkstra's algorithm (calculate least-cost paths from source u)

Iteration 5

$N' = \{u, x, y, v, w, z\} = N$



Forwarding table for node u:

Destination	Next Hop
v	v
w	x
x	x
y	x
z	x

Computational complexity of naive Dijkstra's algorithm

Initialization: $N' = \{s\}$

$\forall v \in N: D(v) = c(s, v)$

Repeat: $w = \underset{v \notin N'}{\operatorname{argmin}} D(v)$

In iteration i, we need to search n-i nodes

$N' = N' \cup \{w\}$

$\forall$ neighbours v *of* $w \wedge v \notin N': D(v) = \min\big(D(v), D(w) + c(w, v)\big)$

Until: $N' = N$

Total number of nodes visited is

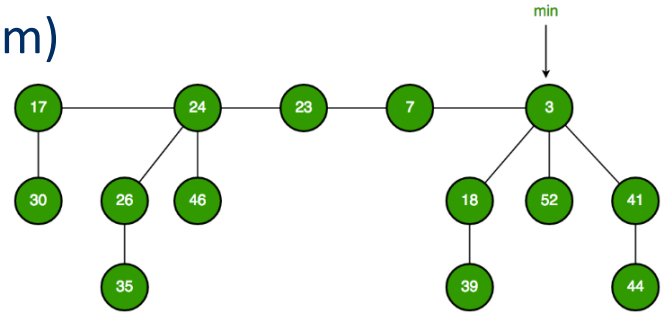
$$(n-1) + (n-2) + \cdots + 1 = \frac{n \times (n-1)}{2} = O(n^2)$$

Total number of edges considered is *m*

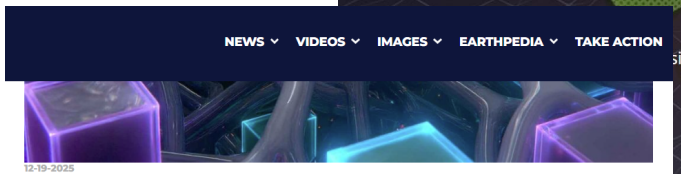
In worst case, $\mathrm{m} = \frac{n \times (n-1)}{2} = O(n^2)$

Computational complexity of Dijkstra's algorithm

- Implementation using Fibonacci heap has lower complexity: O(n * log n + m)
  - O(log n) to extract minimum value
  - O(1) to update element
  - Considered as "speed limit" for > 40 years
  - Usage of binary heap instead of Fibonacci heap is widely used due to lower constants



- In 2025, scientists from Tsinghua University in Beijing developed a new algorithm: $O(m * \log^{2/3} n)$ complexity
  - Becomes better than classical implementations for graphs with more than 10^{19} nodes (according to L. Castro et al. "Implementation and brief experimental analysis of the Duan et al. (2025) algorithm for single-source shortest paths")



12-19-2025

New algorithm finds the shortest path to any point in record time

By Jordan Joseph
Earth.com staff writer

Follow Earth on Google

Shortest path algorithms sit at the heart of modern graph theory and many of the systems that move people, data, and goods around the world.

After nearly seventy years of relying on the same classic approach, researchers have now created a faster way to solve this task.

New Method Is the Fastest Way To Find the Best Routes

18

A canonical problem in computer science is to find the shortest route to every point in a network. A new approach beats the classic algorithm taught in textbooks.

The (decentralized) distance vector (DV) routing algorithm

- Distributed routing algorithm where each node only relies on information obtained from neighbours
- Relies on the Bellman-Ford equation:

$$d_x(y) = \min_v \{c(x,v) + D_v(y)\}$$

where $D_x(y)$ is the least-cost distance from x to y

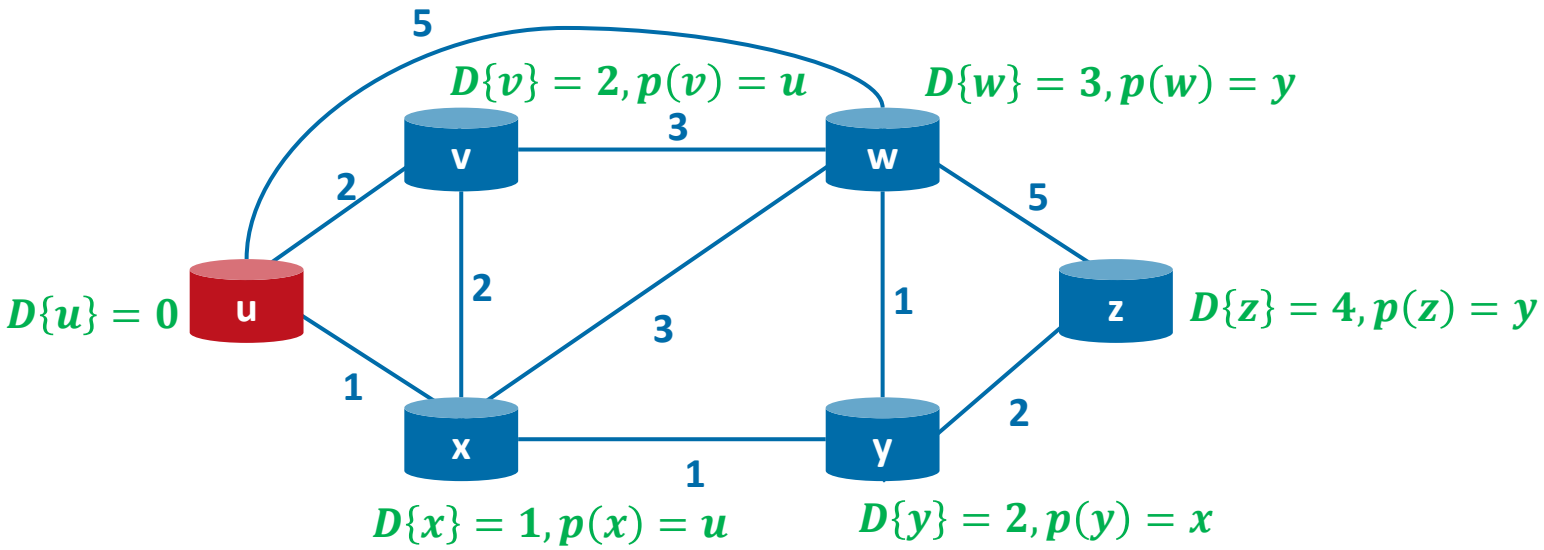
- The solution to the Bellman-Ford equation provides node x's forwarding table.
- Each node x keeps track of
  - An estimate distance vector $\boldsymbol{D}_x = [D_x(y): \forall\, y \in N]$
  - The cost $c(x,v)$ to each of its direct neighbours v
  - The distance vector of each of its neighbours v, $\boldsymbol{D}_v = [D_v(y): \forall\, y \in N]$

The (decentralized) distance vector (DV) routing algorithm

Relies on the Bellman-Ford equation:

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

where $d_x(y)$ is the least-cost distance from x to y



Homework: Check if the Bellman-Ford equation holds for source node u and destination node z in the previous example graph.

DV algorithm pseudocode (at each node x)

Initialization:

$\forall\, y \in N\colon\ D_x(y) = c(x, y)$

$\forall$ neighbours of w: send $D_x = [D_x(y)\colon\ \forall\, y \in N]$ to w

Repeat:

(whenever link cost changes, or distance vector received from neighbour)

$\forall\, y \in N\colon\ D_x(y) = \min_v\{c(x, v) + D_v(y)\}$

set $v^* = \text{argmin}\{c(x, v) + D_v(y)\}$ as next hop for dest y

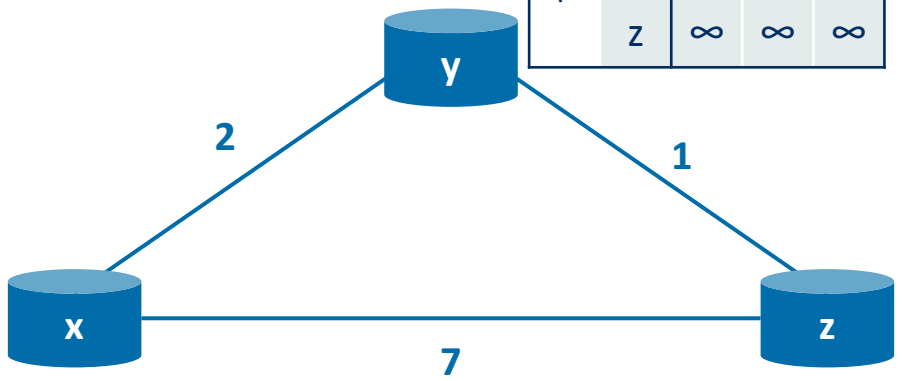
if $D_x(y)$ changed for any destination y:

$\forall$ neighbours of w: send $D_x = [D_x(y)\colon\ \forall\, y \in N]$ to w

forever

Example: DV algorithm

Initialization



Node y table:

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

Node z table:

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

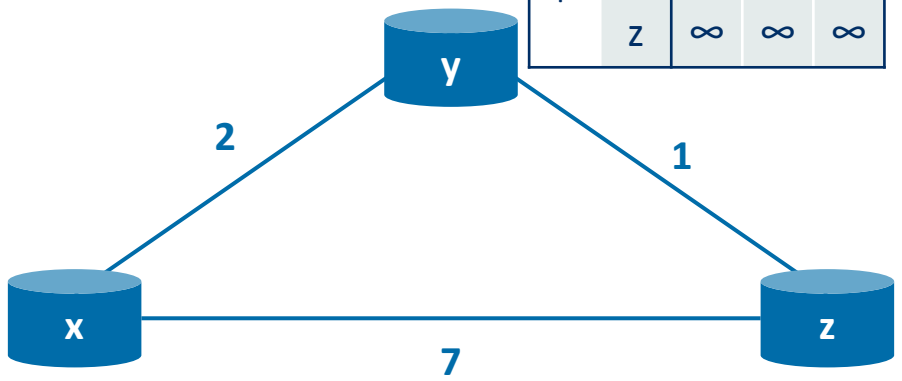
Node x table:

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

Example: DV algorithm

Step 1

Node x received D_y and D_z



Node y table:

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

Node z table:

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

Node x table:

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

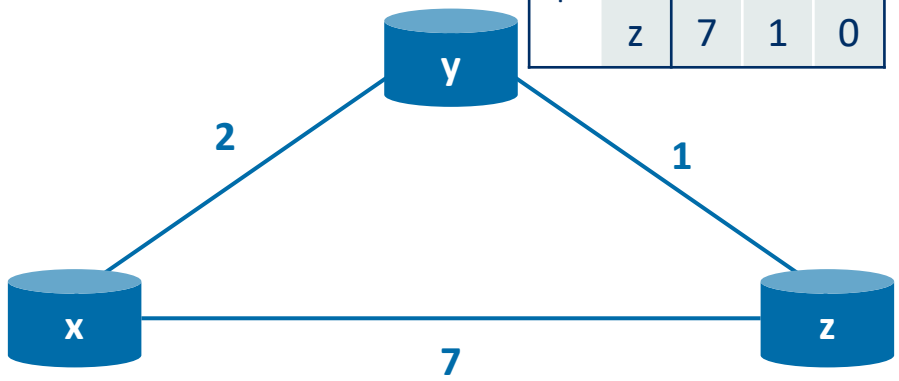
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

$c(x, y) + D_y(z)$

Example: DV algorithm

Step 2

Node y receives D_x and D_z



Node y table:

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

Node z table:

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

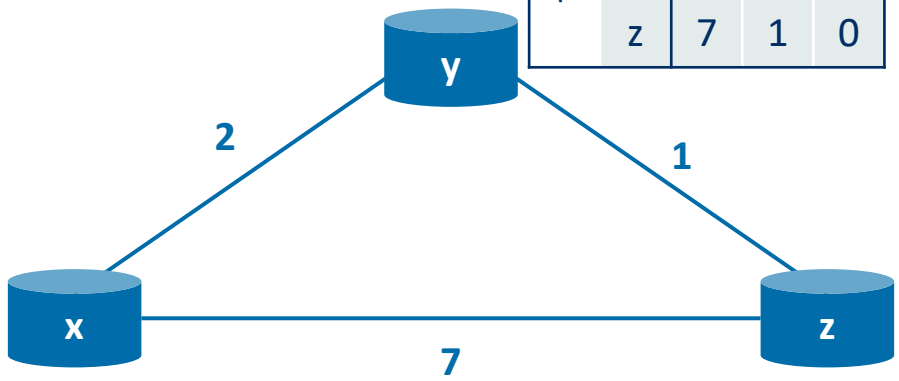
Node x table:

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

Example: DV algorithm

Step 3

Node z receives D_x and D_y



Node y table:

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

Node z table:

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

$c(z, y) + D_y(x)$

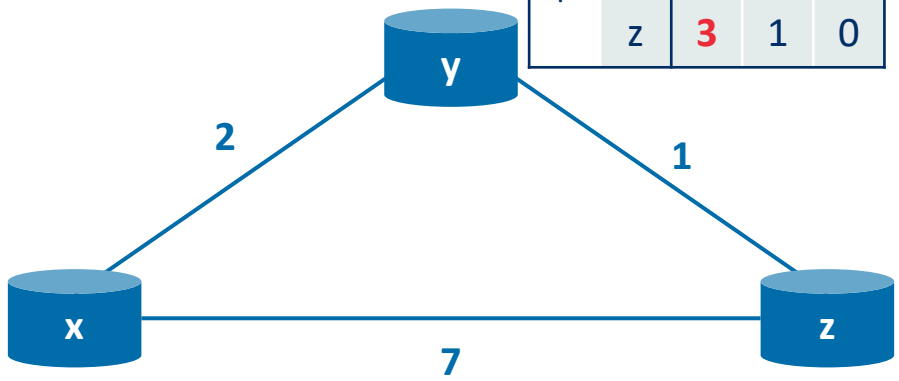
Node x table:

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

Example: DV algorithm

Step 4

Node x and y receive D_z



Node y table:

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

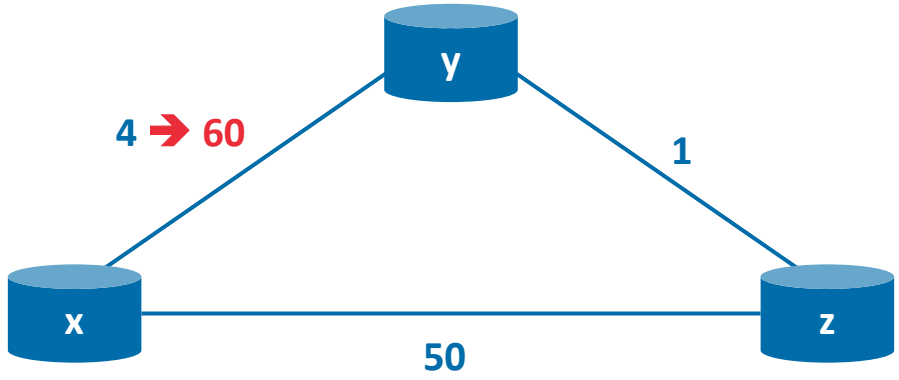
Node z table:

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Node x table:

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Routing loops in DV due to link cost increase (count-to-infinity problem)



Node y table:

Before change:

from \ cost to	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

Step 1: cost change, advertise D_y to x and z

from \ cost to	x	y	z
x	0	4	5
y	6	0	1
z	5	1	0

Routing loop!

Step 2: receive D_x and D_z

from \ cost to	x	y	z
x	0	51	50
y	6	0	1
z	7	1	0

Step 3: recalculate DVs, advertise D_y to x and z

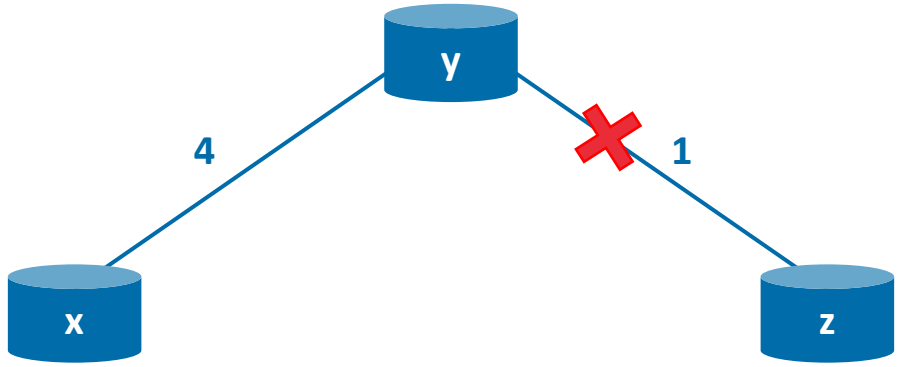
from \ cost to	x	y	z
x	0	51	50
y	8	0	1
z	7	1	0

...

Step 46: self-correction

from \ cost to	x	y	z
x	0	51	50
y	51	0	1
z	50	1	0

Routing loops in DV due to link loss (count-to-infinity problem)



Node y table:

Before loss:

from \ cost to	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

Step 1: link loss, advertise D_y to x

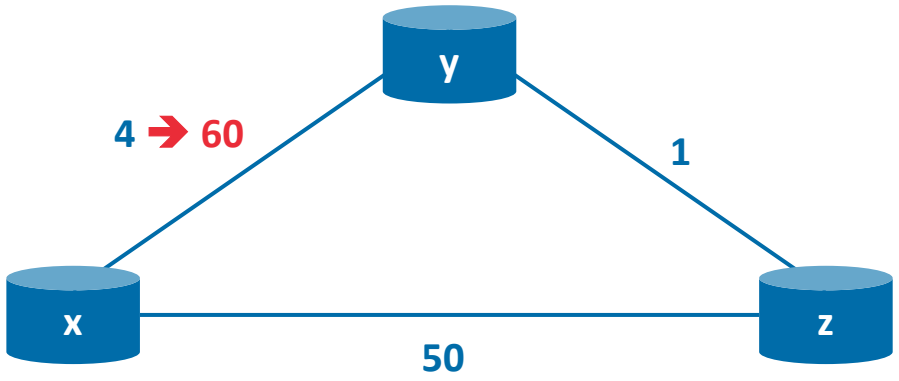
from \ cost to	x	y	z
x	0	4	5
y	4	0	**9**
z	5	1	0

Step 2: receive D_x, recalculate DVs

from \ cost to	x	y	z
x	0	4	**13**
y	4	0	9
z	5	1	0

...

Poisoned reverse



Solution – Poisoned Reverse:

If a node x routes through y to get to z, then x will always tell y that $D_x(z)=\infty$

Node y table

Before change:

from \ cost to	x	y	z
x	0	4	∞
y	4	0	1
z	∞	1	0

Poisoned reverse!

Node y table

Step 1: cost change

from \ cost to	x	y	z
x	0	4	∞
y	60	0	1
z	5	1	0

Advertisement D_y

Step 2: Advertise D_y to z

from \ cost to	x	y	z
y	60	0	1

Node z table

Step 3: receive D_y, recalculate DVs

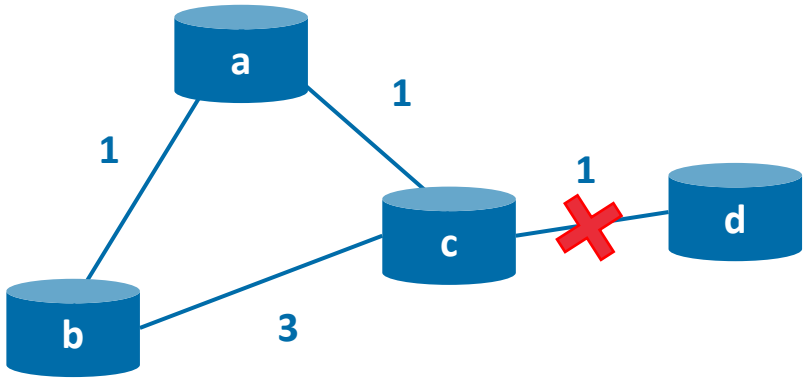
from \ cost to	x	y	z
x	0	∞	50
y	60	0	1
z	50	1	0

Node y table

Step 4: receive D_z, self-correction

from \ cost to	x	y	z
x	0	51	50
y	51	0	1
z	50	1	0

Poisoned reverse



Poisoned Reverse does not always work

If a node y routes through z to get to x, then y will always tell z that $D_y(x)=\infty$

Node c table

Before change:

from \ cost to	a	b	c	d
a	0	1	1	∞
b	1	0	2	3
c	1	2	0	1
d	∞	∞	1	0

Node c table

Step 1: link loss, update DVs

from \ cost to	a	b	c	d
a	0	1	1	∞
b	1	0	2	3
c	1	2	0	**6**
d	∞	∞	1	0

Node a table

Step 2: update DVs at node a after D_c

from \ cost to	a	b	c	d
a	0	1	1	**7**
b	1	0	∞	∞
c	1	∞	0	6

Node b table

Step 3: update DVs at node b after D_c and D_a

from \ cost to	a	b	c	d
a	0	1	1	7
b	1	0	2	**8**
c	1	2	0	6

Node c table

Step 4: update DVs at node b after D_a and D_b

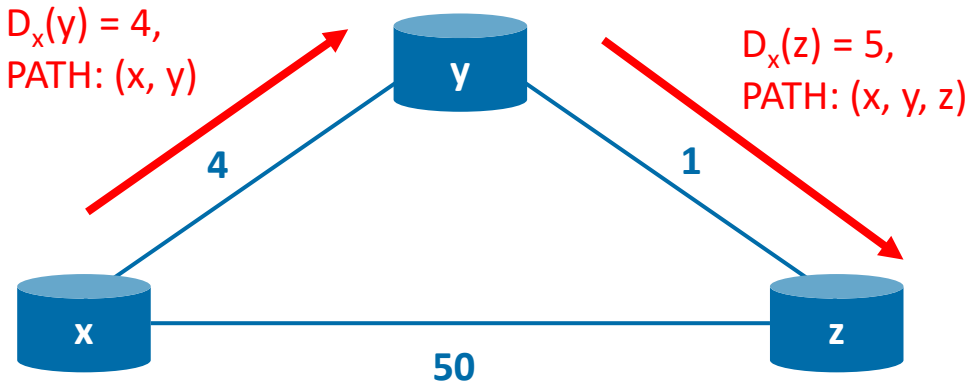
from \ cost to	a	b	c	d
a	0	1	1	∞
b	1	0	2	8
c	1	2	0	**11**
d	∞	∞	1	0

...

Path-vector routing

- **Key idea:** advertise the entire path to the destination node along with distance metric

- **Advantages:**
  - avoid count-to-infinity problem due to fast loop detection (node cannot be twice in the path)
  - support of flexible routing policies based on local preference of one path over another



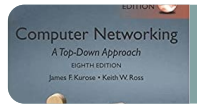
Routing protocols summary

- In **Link-State protocols**, all routers have complete information about connectivity and link costs. Routers calculate shortest paths using Dijkstra's algorithm. The algorithm ensures that the chosen paths are loop-free.

- In **Distance-Vector protocols**, routers rely on the knowledge provided by their neighbors. Each router advertises the vector of current shortest distances to all reachable nodes in the network. Special mechanisms, such as Poisoned Reverse, are used to avoid loops. To overcome count-to-infinity, they need a threshold value.

- In **Path-Vector protocols**, a list of all visited nodes is advertised together with the distance vector. Routers reject the paths that already contain their own ID.

Routing on the Internet: OSPF & BGP



Sections 5.3 and 5.4

Routing on the Internet happens on two levels

- The Internet is split into **autonomous systems** (ASs)
  - Each AS consists of a group of routers that are under the same administrative control (e.g., owned by one ISP)
  - Each AS has a globally unique autonomous system number (ASN)

- Two different types of routing algorithms
  - Within an AS (**intra-AS**): Interior Gateway Protocol (IGP), such as Open Shortest Path First (OSPF)
  - Across ASs (**inter-AS**): Exterior Gateway Protocol (EGP), such as Border Gateway Protocol (BGP)

- Advantages of a two-stage approach
  - **Scalability**: The number of routers on the Internet is too big for a single routing algorithm to handle
  - **Administrative autonomy**: Each ISP wants to enforce its own policies on the routing algorithms it uses

Interior Gateway Protocols (IGPs)

- Open Shortest Path First (OSPF)
  - Based on link-state routing and Dijkstra's shortest path algorithm
  - Standardized as RFC 2328 (IPv4) and RFC 5340 (IPv6)
  - Widely used in large enterprise networks

- Intermediate System to Intermediate System (IS-IS)
  - Conceptually very similar to OSPF (also based on link-state routing and Dijkstra's shortest path algorithm)
  - Standardized as ISO/IEC 10589:2002
  - Widely used in large service provider networks

- Enhanced Interior Gateway Routing Protocol (EIGRP)
  - Based on distance-vector routing
  - Developed as a proprietary protocol by Cisco, but eventually standardized as RFC 7868 in 2016

Open Shortest Path First (OSPF)

- Based on link-state routing and Dijkstra's shortest path algorithm
  - Routers **flood** link-state information [i.e., current knowledge of the entire AS] **to all other routers in the autonomous system**, not only direct neighbours
  - Link-state is advertised when a link changes (cost change, comes up, or goes down)
  - Link-state is also advertised periodically (at least once very 30 minutes)
  - Link availability is checked by sending periodic HELLO messages to all direct neighbours
- Works directly on top of IP, i.e., none of the transport layer protocols are used. OSPF should take care of reliable delivery of its messages on its own.
- Link weights are manually configured by the administrator
- Other features
  - Security: Exchanges between OSPF routers can be authenticated (when IPv6 is used, can rely on IP Authentication Header)
  - Multi-path: If multiple same-cost paths exist, they can be used simultaneously to each carry part of the traffic
  - Multicast support
  - Hierarchical division of AS into multiple independent routing areas

OSPF initialization and link-state database synchronization

RouterID = 0.0.0.1

RouterID = 0.0.0.2

Not the same as IP address

a

b

Hello (srcID = 0.0.0.1,
Active Neighbor = None)

Hello (srcID = 0.0.0.2,
Active Neighbor = **0.0.0.1**)

dst IP address: 224.0.0.5 (IPv4) or ff02::5 (IPv6)
multicast to all OSPF routers in the same broadcast area (to routers b and c)

DB Description (srcID = 0.0.0.1,
seq = x, Init, Master, More)

DB Description is unicast

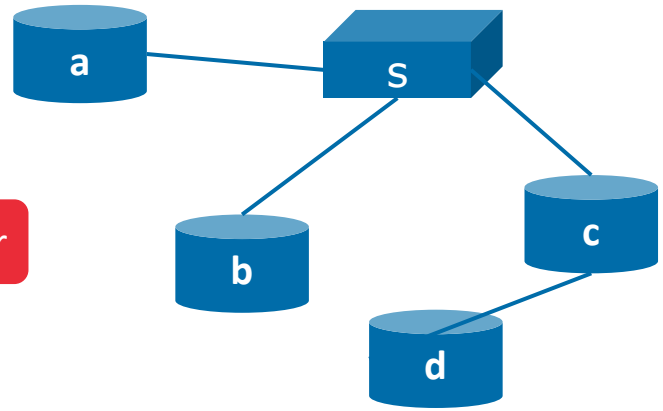
DB Description (srcID = 0.0.0.2,
seq = y, Init, **Master**, More)

My RouterID is higher => I will be Master

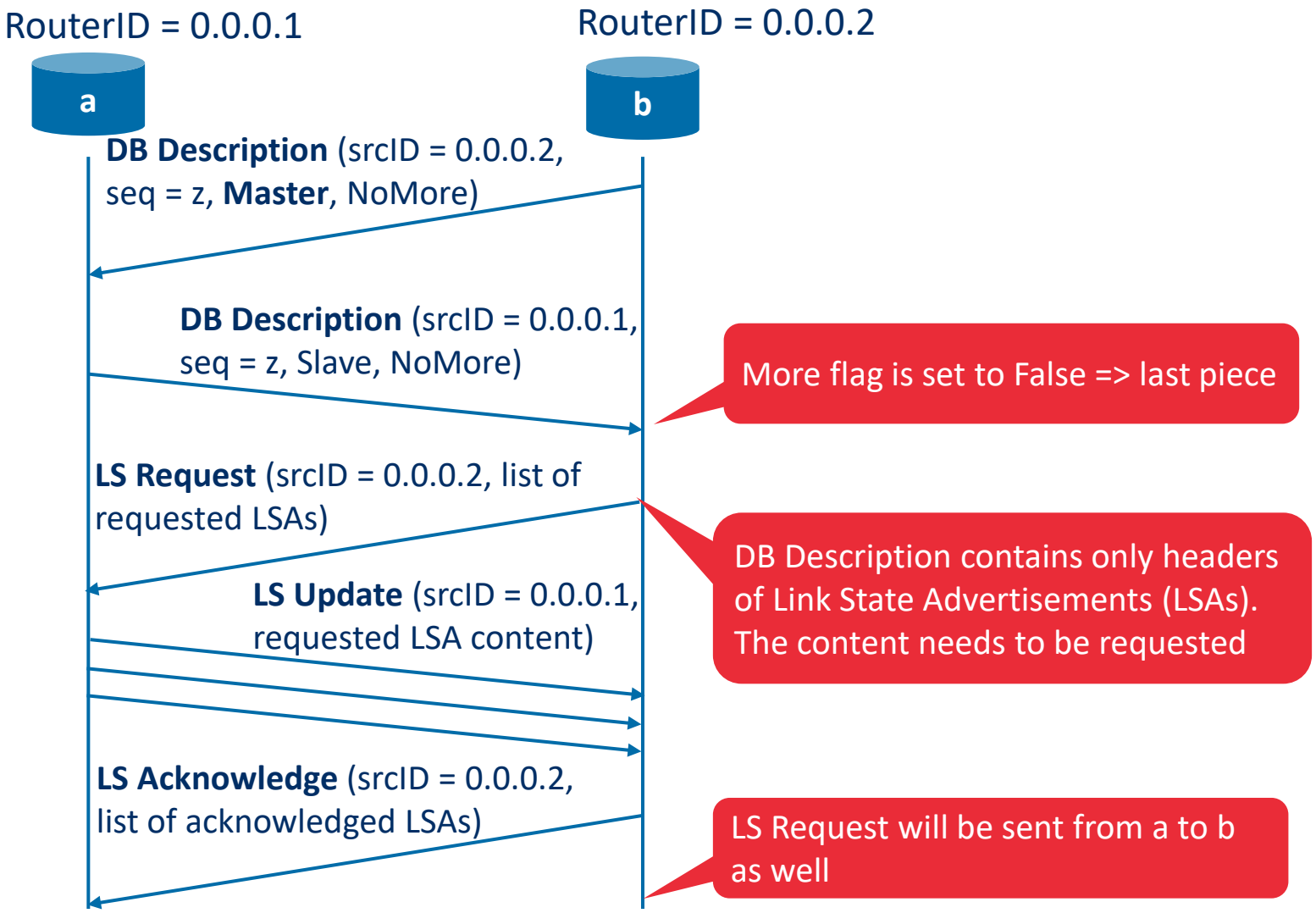
DB Description (srcID = 0.0.0.1,
seq = y, **Slave**, More)

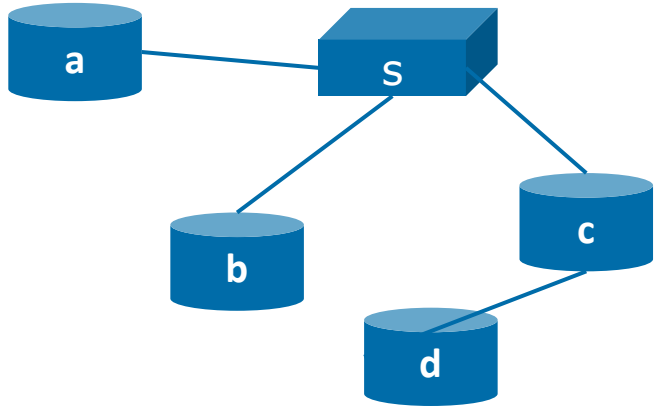
Same seq => previous packet is acknowledged.
If unexpected seq: repeat previous packet

...



OSPF initialization and link-state database synchronization

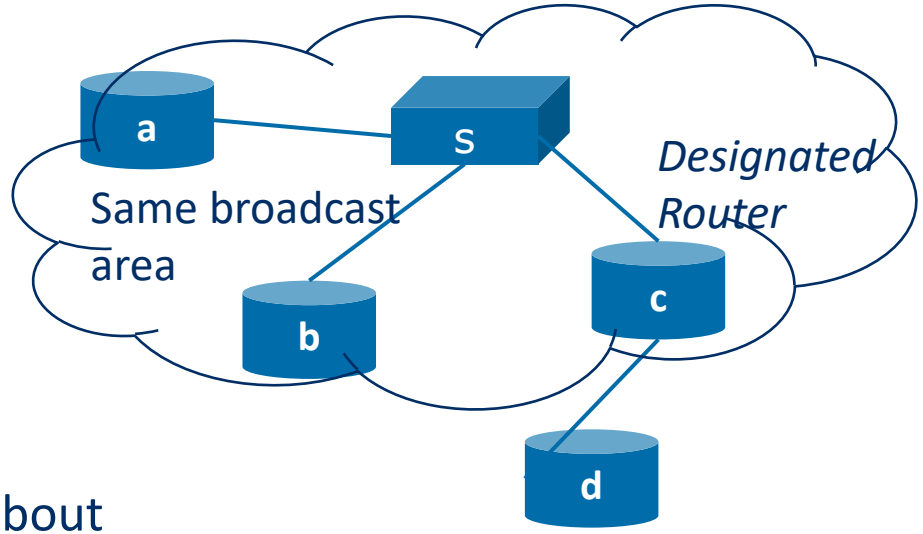




OSPF Link-State Advertisements (LSAs)

- **Router LSA** contains information about interfaces of the router. For each interface, it provides a RouterID of the neighboring router connected through this interface and link cost (metric).
  In *OSPFv2*, also provides the IP address of the Designated Router in the broadcast area.

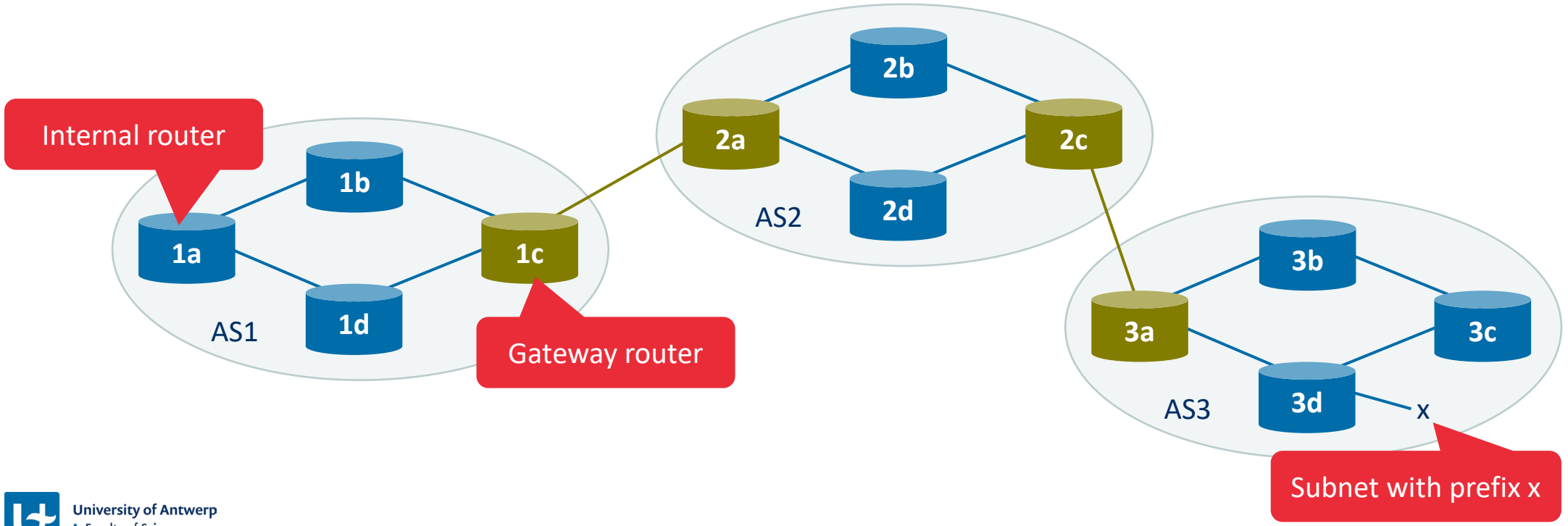
- **Network LSA** is generated by designated router, and contains information about all attached routers in this broadcast area. Designated router is responsible for other routers in this area.
  In *OSPFv2,* also provides the network mask.



- **Intra-area prefix LSA** (*OSPFv3* only) contains information about network IP addresses (prefixes) reachable through a router.

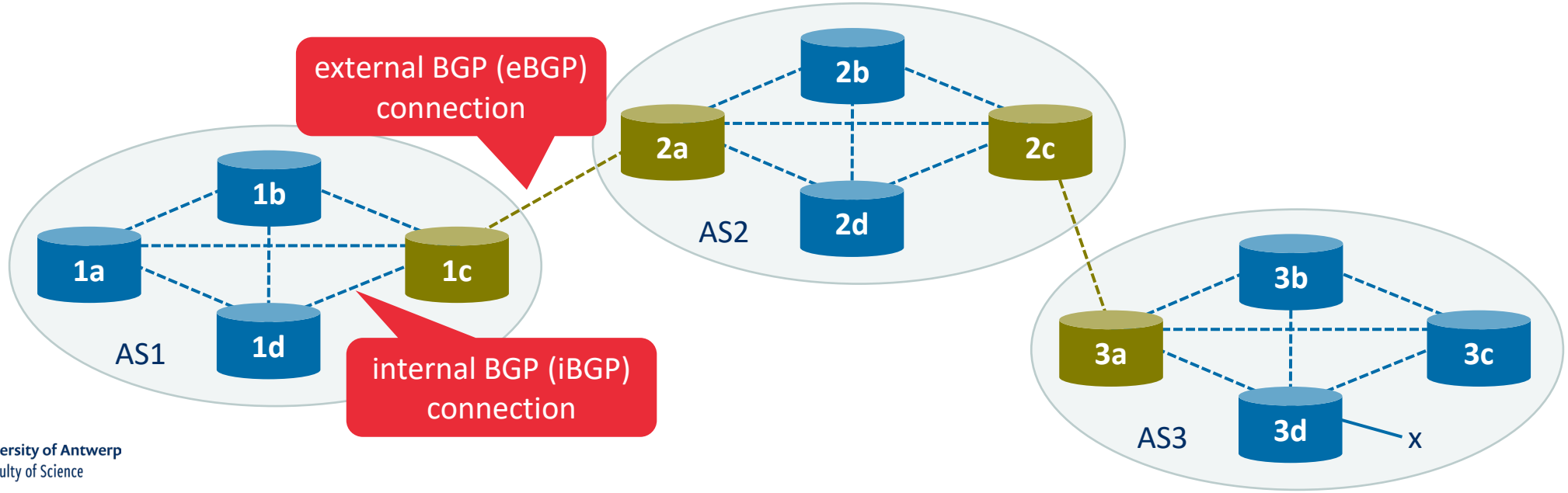
Border Gateway Protocol (BGP)

- BGP is used for routing between all ASs on the Internet
- Decentralized and asynchronous, based on **path-vector** routing
- Relies on TCP for delivery of its messages
- Routing is done based on CIDR prefixes (e.g., 138.16.68/22). Summary prefixes can be advertised instead of many specific prefixes from the AS (e.g., one /22 prefix instead of multiple /24 prefixes)

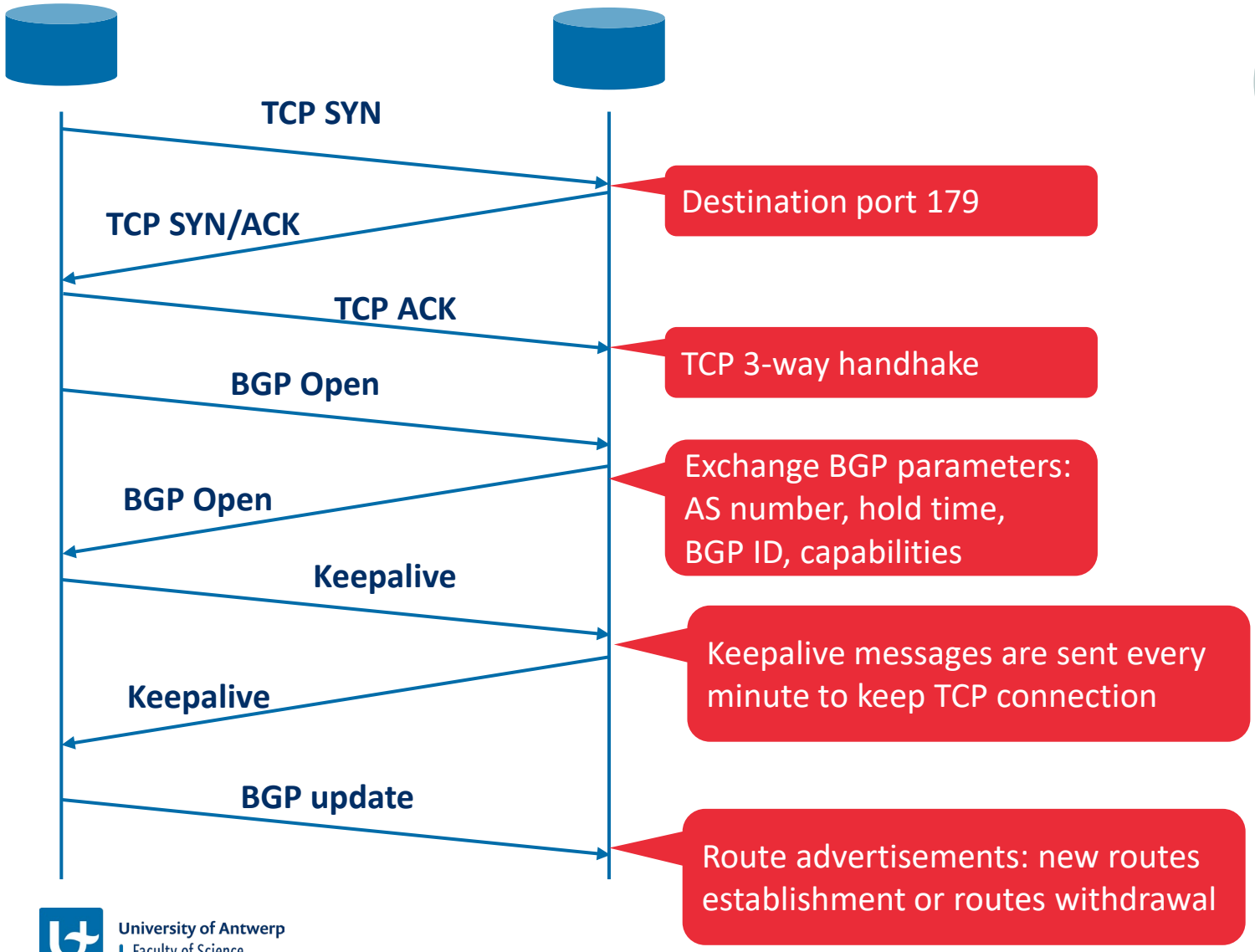


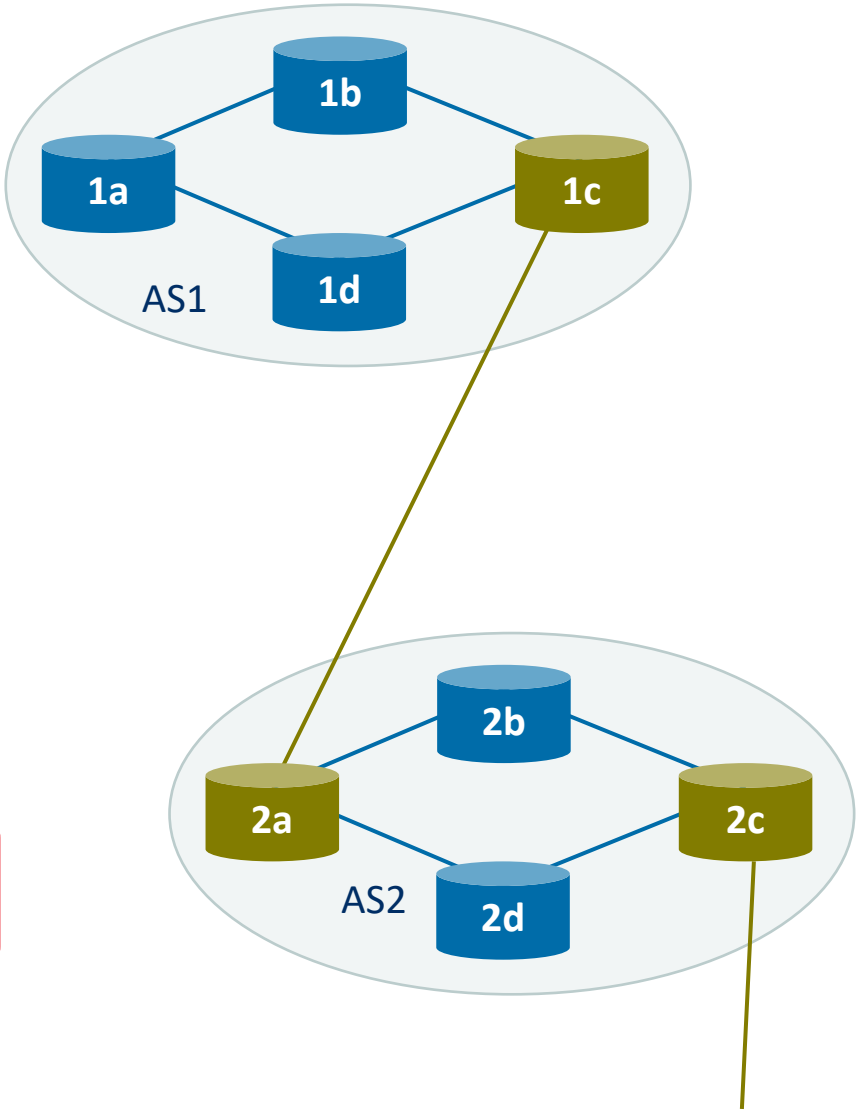
BGP connections

- How is information about subnet x in AS3 advertised to the Internet?
- Routers maintain a semi-permanent TCP connection on port 179 to exchange BGP information
  - **iBGP connection**: TCP connection between two routers within the same AS
  - **eBGP connection**: TCP connection between two routers in different ASs
- Within an AS, all routers maintain an iBGP connection to each other (in a full mesh or using Route Reflectors), while eBGP connections are only maintained between direct neighbours in different ASs.

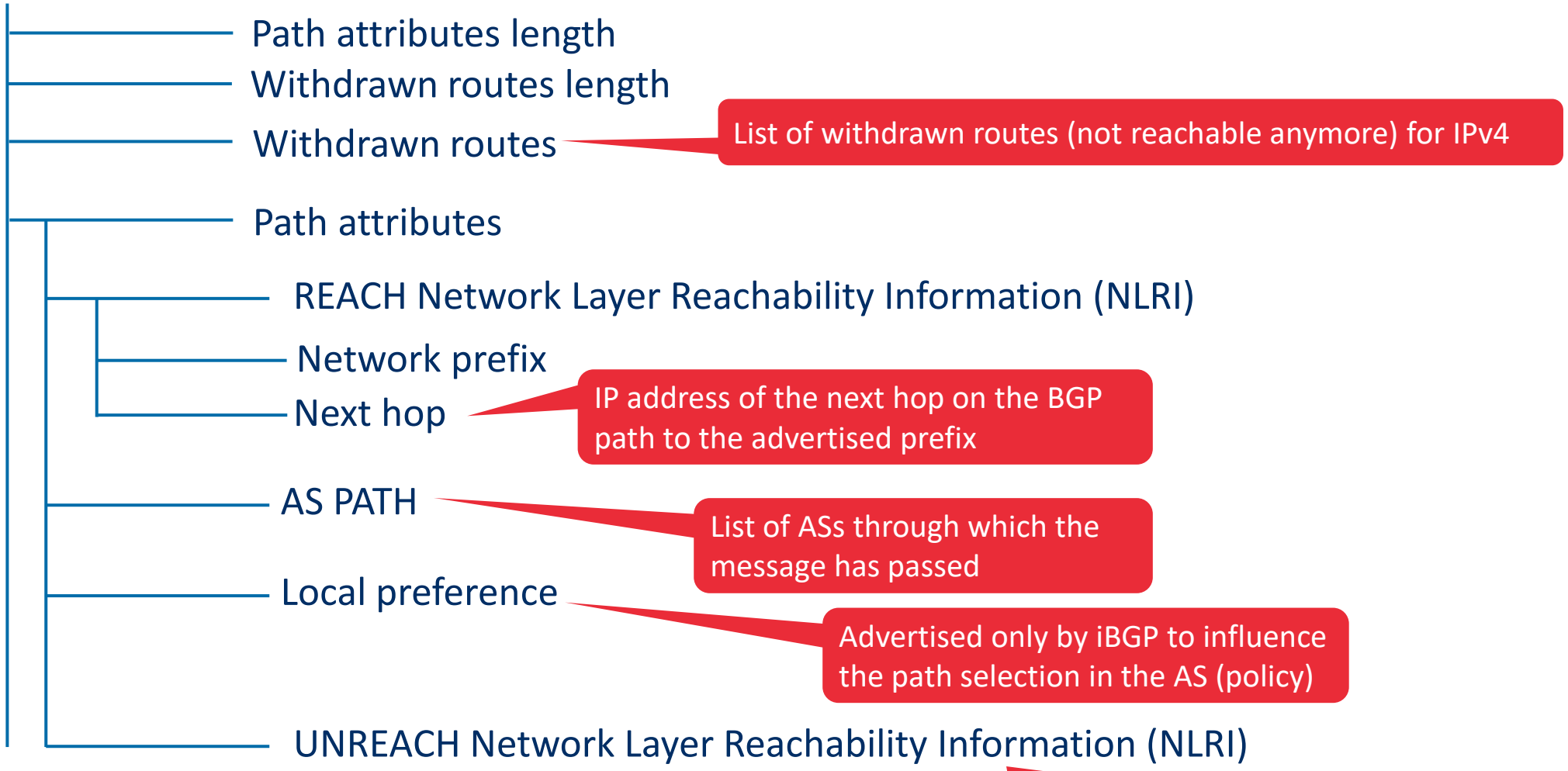


BGP initialization





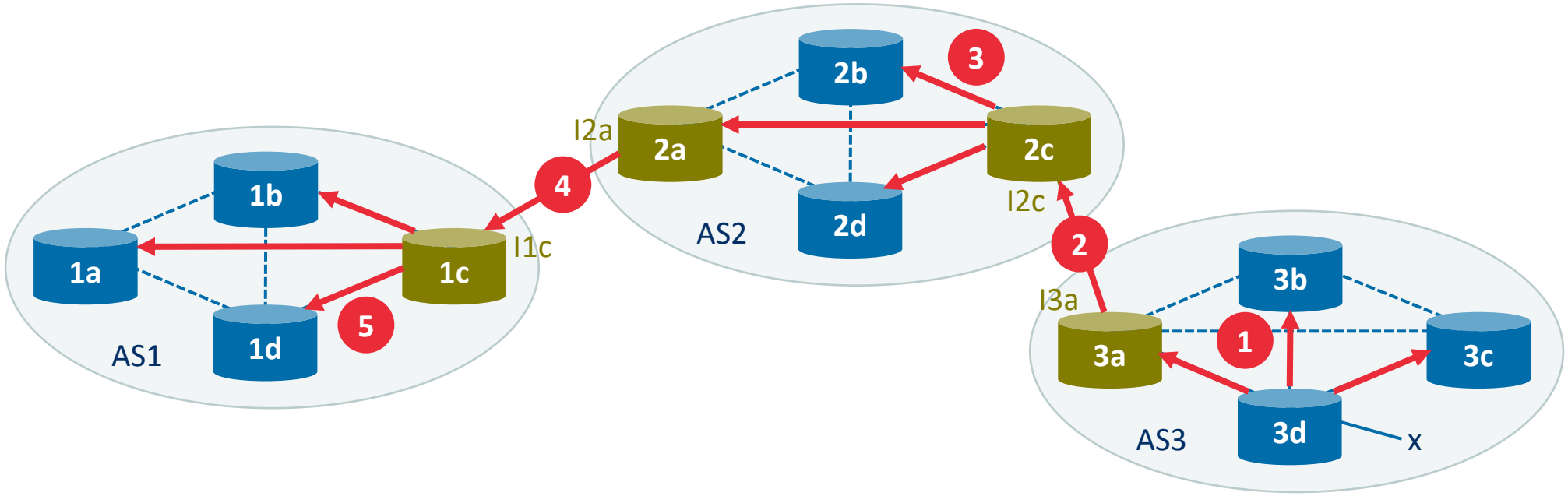
BGP Update messages



BGP advertisement messages

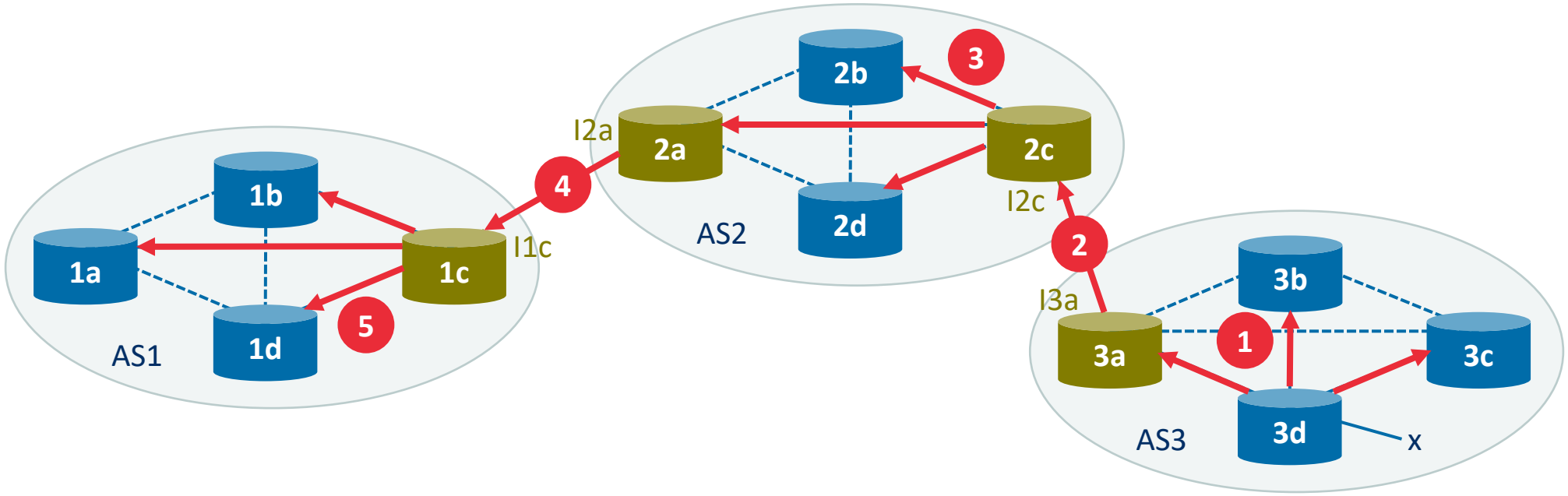
- BGP advertisements (also called routes) consist of an address prefix and several attributes
  - AS-PATH: List of ASs through which the message has passed
  - NEXT-HOP: IP address of the router interface that begins the AS-PATH

- **Example (advertising subnet x):**



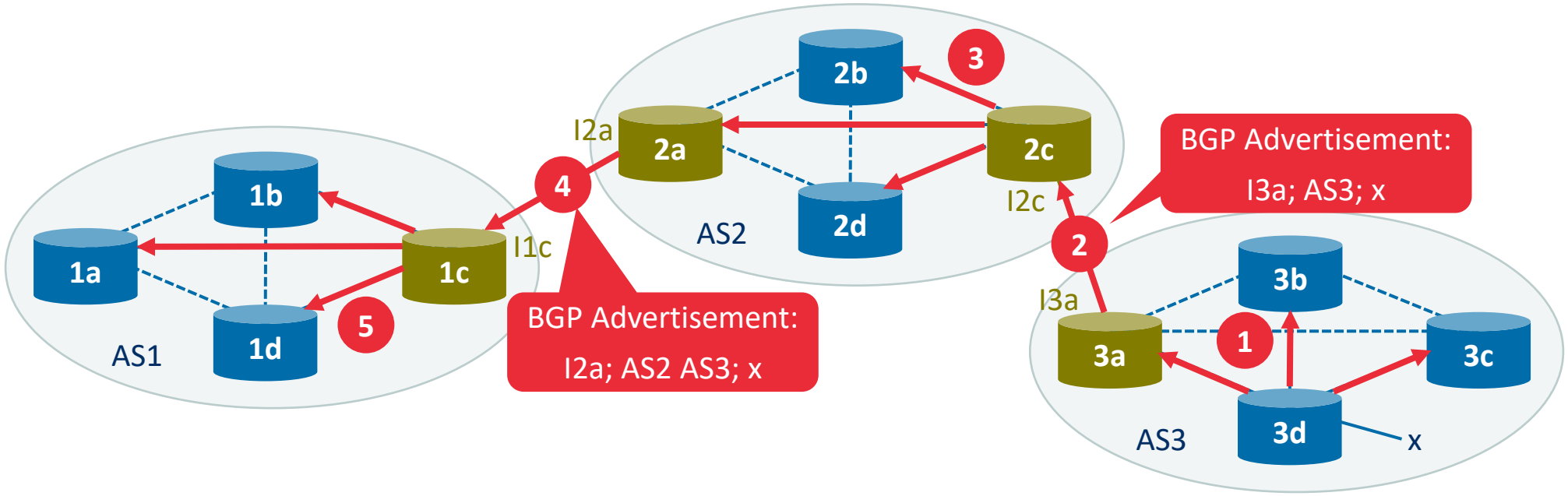
Exercise: BGP advertisement messages

Exercise: What is the content of the BGP advertisements sent at steps 2 and 4, assuming a structure “NEXT-HOP; AS-PATH; prefix”?



Solution: BGP advertisement messages

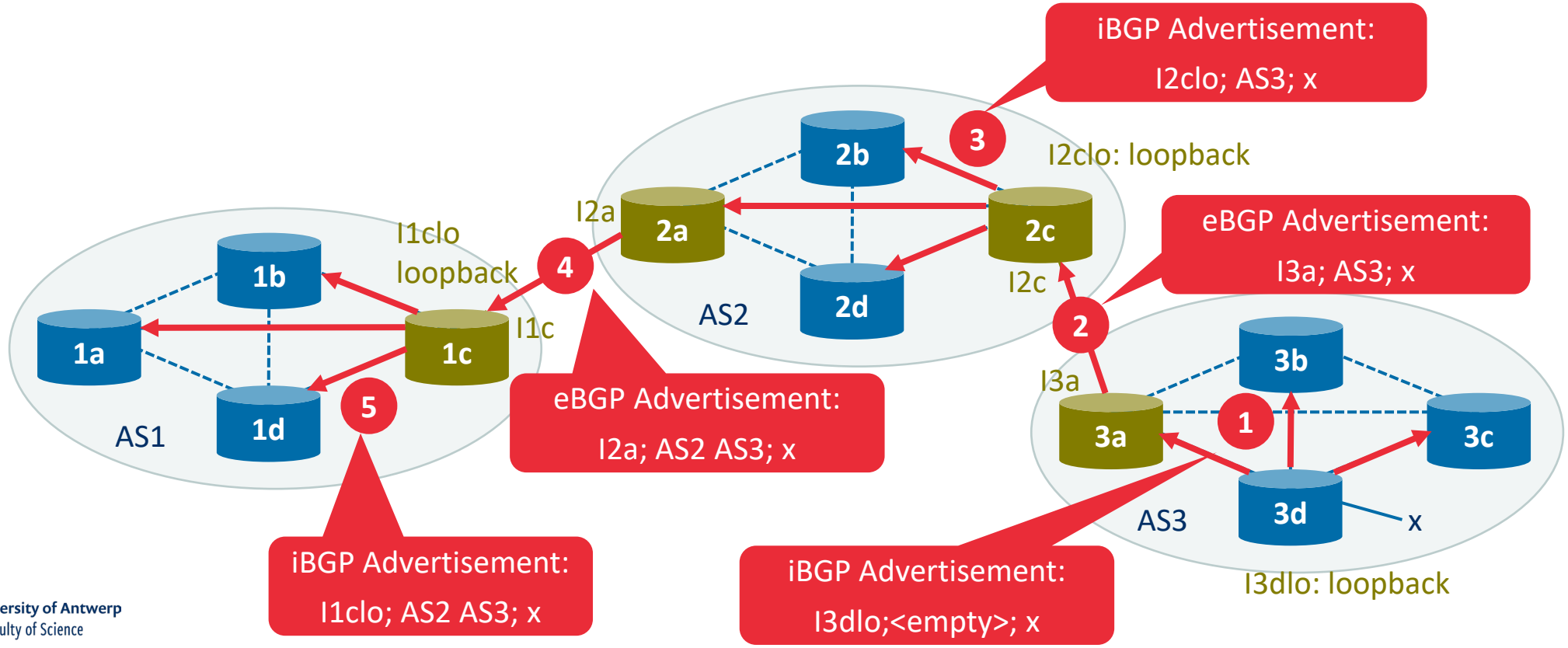
Exercise: What is the content of the BGP advertisements sent at steps 2 and 4, assuming a structure “NEXT-HOP; AS-PATH; prefix”?



BGP advertisement messages: iBGP next hop self

IP address of interface I3a is unknown inside AS2. To be able to route to next hop inside AS2, iBGP substitutes IP address of the next hop with internal IP address.

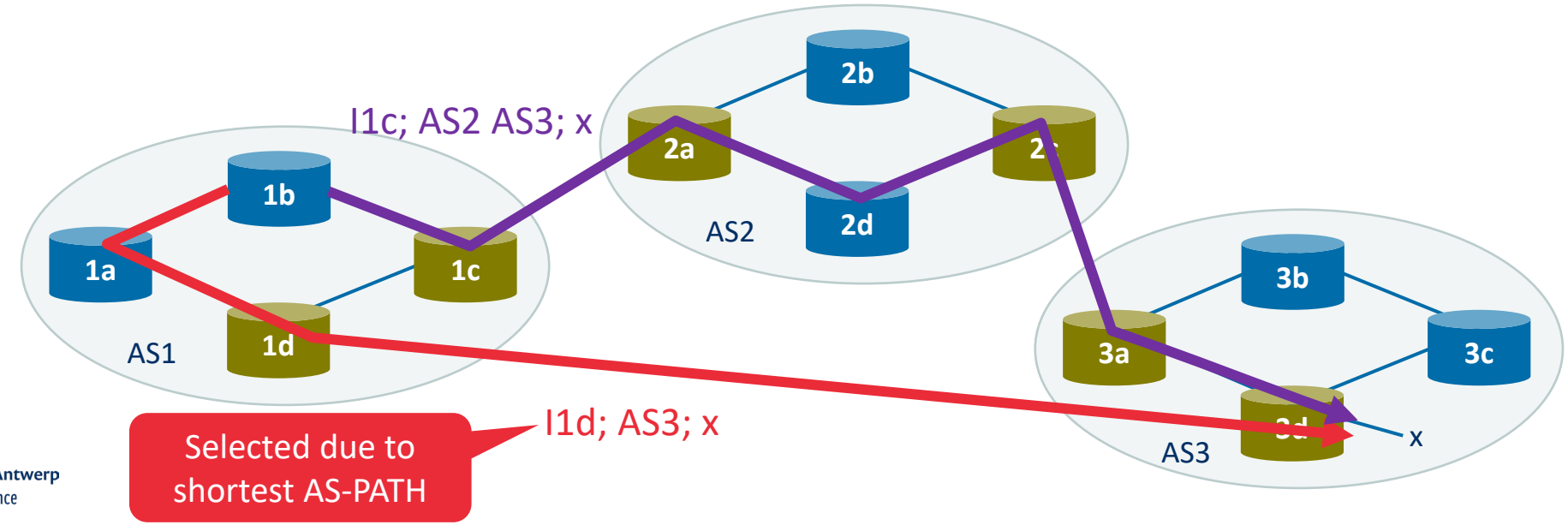
A good practice is to use loopback interface.



BGP route-selection algorithm

- BGP sequentially invokes elimination rules until only 1 route remains
  1. If any routes have the **local preference value** set, the route with the highest local preference is selected
  2. From the remaining routes, the one with the **shortest AS-PATH** is selected
  3. From the remaining routes, the one with the **closest NEXT-HOP** router is selected (determined using intra-AS routing, e.g., OSPF)
  4. If more than 1 route remains, a route is selected based on the BGP identifier

- **Example**: Which of the 2 routes would be selected?



IP-Anycast

- BGP can be used to advertise the same IP address for multiple servers, allowing routers to automatically select the nearest one (e.g., DNS server replication)