Computer Network I

Reti di Calcolatori I

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Informatica

Riccardo Caccavale

(riccardo.caccavale@unina.it)

Network Layer

Routing: Introduction

- We recall that, on packet receival, a **router can perform 3 actions**:
  - **Forwarding the packet** as is to a suitable neighbor.
  - **Forwarding a modified/replaced packet** (e.g., fragmentation, masquerading, etc.).
  - **Dropping the packet** (e.g., if expired).
- As we saw**, routers are endowed with forwarding tables** that specify the local forward strategy (in the neighborhood of the router). The way these tables are created can be centralized or decentralized.
- To decide where packets should be forwarded routers make use of **routing algorithms**, which determine good paths (i.e., sequence of routes) from senders to receivers.
- Typically, **a "good" path is one that has the minimal cost** (shortest or fastest path, etc.), in real-world **there are also policy issues** to be considered (e.g., rules specifying if an organization can talk with another etc.).

Network Layer

Routing: Problem Formulation

- We can **formalize** our network as a graph $G = (N, E)$, where:
  - N is a set of **nodes** (the routers of our network).
  - $E \subseteq N \times N$ is a set of **edges** (physical links) represented as a pair of nodes.

- An **edge also has a value representing its cost**, which may reflect the physical length of the link, the link speed, or the monetary cost associated with a link.

- We can formalize a generic **cost** as a function $c: N \times N \to \mathbb{R}$ such that, given a generic couple of nodes $(x, y) \in N \times N$:
  - $\forall x \in N$, then $c(x, x) = 0$
  - If $(x, y) \notin E$, then $c(x, y) = \infty$
  - If $(x, y) \in E$, then also $(y, x) \in E$ (undirected graph) and $c(x, y) = c(y, x)$

Network Layer

Routing: Problem Formulation

- Following this formulation, a **path** π in G is a **sequence of nodes** $\pi = (x_1, x_2, \dots, x_n)$ such that **all adjacent nodes are linked**, formally:

$$\forall x_i, x_{i+1} \in \pi, (x_i, x_{i+1}) \in E$$

- The **overall cost of the path** is the sum of the links' costs:

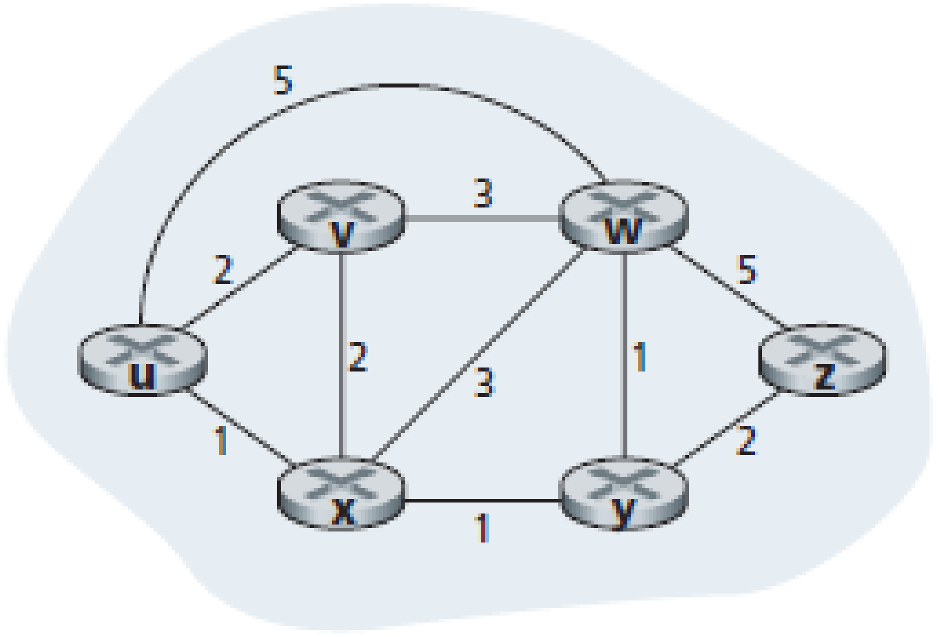
$$c(\pi) = \sum_{i=1}^{n-1} c(x_i, x_{i+1})$$

- Let's call $P(x, y)$ the **set of all possible paths** connecting the nodes x and y, we can define **the best path** as the path $\pi^* \in P(x, y)$ having minimum cost:

$$\forall \pi \in P(x, y), c(\pi^*) \leq c(\pi)$$

Network Layer
Routing: Problem Formulation

- In this example we have a graph composed of 5 nodes:
  - $N = \{u, v, w, x, y, z\}$
  - $E = \{(u,w), (u,v), (w,x), \dots\}$
  - $c(u,w) = 5,\ c(u,v) = 2,\ c(u,x) = 1, \dots$
- For instance, the best path π^* between nodes x and z is:
  - $\pi^* = (x, y, z)$
  - $c(x,y) = 1,\ c(y,z) = 2$
  - $c(\pi^*) = 3$



Network Layer

Routing: Problem Formulation

- Algorithms able to find such optimal path π^* between two nodes are called **shortest path algorithms**.

- In the case of computer networks, we are interested in **finding such optimal path** not for just two nodes but **for all possible couples of nodes**.

- A routing algorithm **converges** when the shortest path for all couples of nodes in the network is found.

- On networks **there are 2 families of algorithms** that are typically used:
  - Distance-Vector.
  - Link-State.

Network Layer

Routing: Distance-Vector Algorithm

- The **Distance-Vector** (DV) algorithm is a **decentralized** approach that exploit **local knowledge** about the network combined with nodes communication to find the shortest paths.
    - It is based on the **Bellman-Ford algorithm** that is used to compute paths.
- Here **each node receives some information** from one or more of its directly attached neighbors, performs a calculation, and then **distributes the results** of its calculation back to its neighbors.
- This algorithm is **asynchronous** in that it does not require all of the nodes to be coordinated with each other.

Network Layer

Routing: Distance-Vector Algorithm

- Let's call $d^*(x, y)$ be the cost (distance) of the best path from node x to node y. We can apply the **Bellman equation**:

$$d^*(x, y) = \min_v\{c(x, v) + d^*(v, y)\}$$

- where v is a neighbor of x. This is quite intuitive, **if v is the first node of the best path**, then the rest of the nodes must be the ones into the best path from v to y.

- If we are able to find such $\boldsymbol{v}$ **we can just add it into the forwarding table** and forward to it all packets directed to y.

- To do that, **we should have an estimation of $\boldsymbol{d^*(x, y)}$** for all neighbors (the distance vector).

Network Layer

Routing: Distance-Vector Algorithm

- The pseudocode:

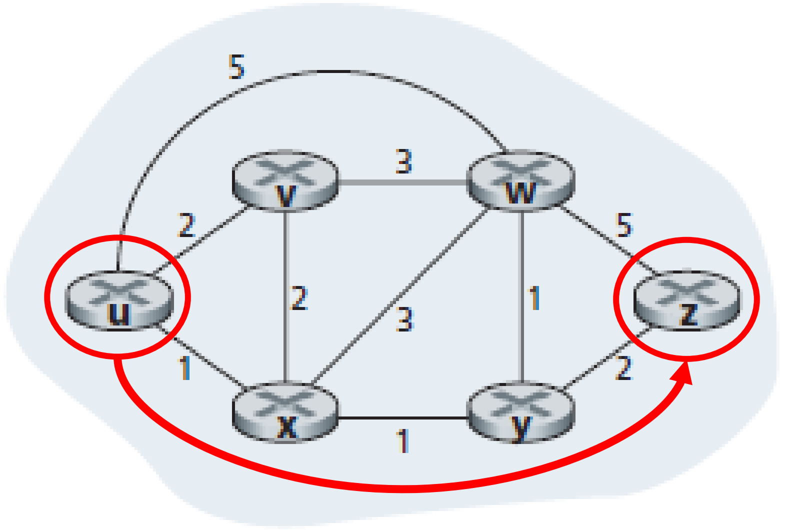
```
DistanceVector(x)
 for all nodes v
  if v is a neighbor of x then
   $D_x(v) = c(x,v)$
  else
   $D_x(v) = \infty$
 for all neighbors w and destinations y
  $D_w(y) = ?$
 send initial $D_x(\cdot)$ to all neighbors

 repeat
  if cost of a neighbor updated then
   for all nodes y
    $D_x(y) = \min_v\{c(x,v) + D_v(y)\}$
   if $D_x(y)$ changed for any destination y then
    send updated $D_x(\cdot)$ to all neighbors
 until false //forever
```

- We use the **data-structure $D_w(v)$** to store the distance vectors from all nodes w to target nodes v.
- During the **initialization** phase:
  - Only **distances to neighbors are set**.
- During the **online** phase:
  - The **news** from the neighborhood are received.
  - Distance vectors are **updated** (Bellman equation).
  - Local changes are **communicated** to adjacent nodes.

Network Layer

Routing: Distance-Vector Algorithm

- To clarify this process **let's focus on just one specific path**, between u and z:
  1. On start, **u only knows its neighbors**. The cost to z is infinity ($D_u(z) = \infty$).
  2. Node **y wakes up** and discovers a better path to z, which is the direct link y-z with cost 2 ($D_y(z) = c(y,z) = 2$). It communicates the good news to x.
  3. Node **x discovers that the best path to z now passes through y** with cost 3 ($D_x(z) = c(x,y) + D_y(z) = 1 + 2$), and forwards the good news to u.
  4. Node **u receives the news**. Now there is a path to z that costs less than infinity passing through x ($D_u(z) = c(u,x) + D_x(z) = 1 + 3$).



- The **same process** takes place for all nodes/paths.

Network Layer
Routing: Distance-Vector Algorithm

- As for the computational complexity, every time the cost of a node is updated, all nodes have to re-evaluate their edges. This process leads to a typical **worst-case complexity of $\boldsymbol{O}(|\boldsymbol{N}||\boldsymbol{E}|)$**.

- Pros:
  - The **algorithm is asynchronous**, this facilitates the computation as routers can on-line and in any stage adapt to updated costs.
- Cons:
  - **Slow convergence**: updates spread slowly as all nodes have to detect the change and send an updated to the adjacent ones.
  - **Count-to-infinity**: since only improved paths are communicated, if a link or a node fails, the propagation of this “bad news” will be slow: the adjacent node of a broken link will immediately switch to an alternative route, but they cannot be sure that the new best path does to not pass through the failure.

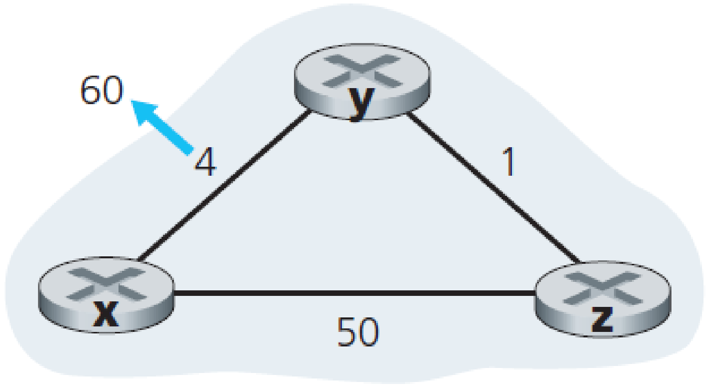
Network Layer

Routing: Count-to-infinity

- Let's consider this **simplified example** in which the link x-y gets a sudden increase of cost.

- $D_y(x) = 4$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

This is the **initial situation** before the increment (c(x,y) is still 4).



Network Layer

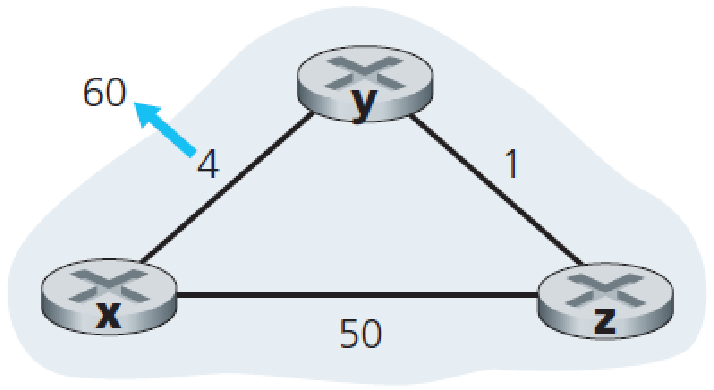
Routing: Count-to-infinity

- Let’s consider this simplified example in which the link x-y gets a sudden increase of cost.

- $D_y(x) = 4$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $\mathbf{D_z(x)} = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

Node y detects the increment and updates distance-vector to x:
$D_y(x) = \min\{ 60, 1 + D_z(x) \} = 1 + D_z(x) = 6$,

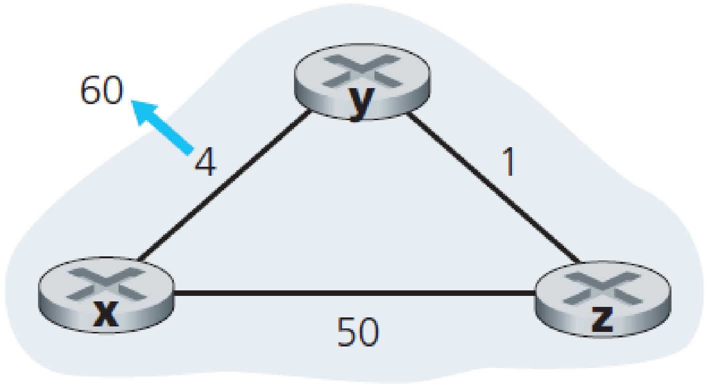
- $D_y(x) = 1 + \mathbf{D_z(x)} = 6$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.



Network Layer

Routing: Count-to-infinity

- Let’s consider this simplified example in which the link x-y gets a sudden increase of cost.



- $D_y(x) = 4$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

- **$D_y(x)$** $= 1 + D_z(x) = 6$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

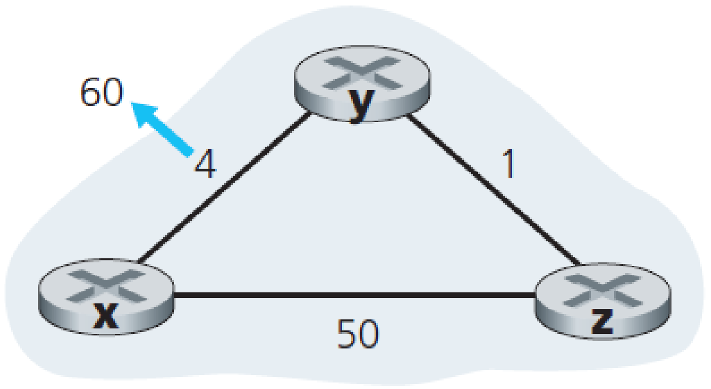
But vector $D_y(x)$ is used by z to compute the path to x, so also $D_z(x)$ must be updated!

- $D_y(x) = 1 + D_z(x) = 6$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 +$ **$D_y(x)$** $= 7$,
- $D_z(y) = 1$.

Network Layer

Routing: Count-to-infinity

- Let's consider this simplified example in which the link x-y gets a sudden increase of cost.



- $D_y(x) = 4$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

- $D_y(x) = 1 + D_z(x) = 6$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

- $D_y(x) = 1 + D_z(x) = 6$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $\mathbf{D_z(x)} = 1 + D_y(x) = 7$,
- $D_z(y) = 1$.

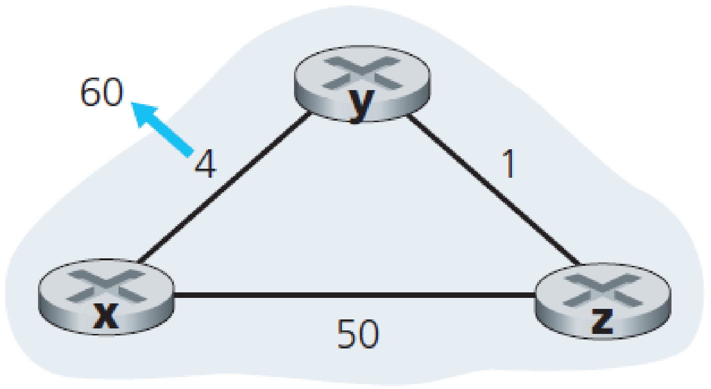
But vector $D_z(x)$ is itself used by y to compute the path to x, so $D_y(x)$ must be updated again!

- $D_y(x) = 1 + \mathbf{D_z(x)} = 8$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 7$,
- $D_z(y) = 1$.

Network Layer

Routing: Count-to-infinity

- Let’s consider this simplified example in which the link x-y gets a sudden increase of cost.



- $D_y(x) = 4$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

- $D_y(x) = 1 + D_z(x) = 6$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 5$,
- $D_z(y) = 1$.

- $D_y(x) = 1 + D_z(x) = 6$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- **$D_z(x)$** $= 1 +$ **$D_y(x)$** $= 9$,
- $D_z(y) = 1$.

- **$D_y(x)$** $= 1 +$ **$D_z(x)$** $= 8$,
- $D_y(z) = 1$,
- $D_x(y) = 4$,
- $D_x(z) = 4 + D_y(z) = 5$,
- $D_z(x) = 1 + D_y(x) = 7$,
- $D_z(y) = 1$.

This is a loop: the 2 vectors will be continuously updated until a stable configuration is reached.

Network Layer
Routing: Link-State Algorithm

- The **Link-State** (LS) algorithm is a **centralized** approach that exploit full knowledge about the network in order to find the best path.
  - The LS algorithms are based on variation of the **Dijkstra algorithm**.

- Notice that such level of knowledge can be achieved in practice by having **each node to broadcast link-state packets**, containing IDs and costs of its attached links, to all other nodes in the network.

- The basic version of the algorithm computes the **shortest paths from a starting node to all possible destination nodes** (so it must be executed **on all nodes**).

Network Layer

Routing: Link-State Algorithm

- The pseudocode:

```
LinkState(x):
 N' = {x}
 for all nodes v
  if v is a neighbor of x then
   D(v) = c(x,v)
  else
   D(v) = ∞

 repeat
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for each neighbor v of w and not in N':
        D(v) = min( D(v), D(w) + c(w,v) )
 until N'= N
```

- We use **the data-structure D(v)** to store the distance from the current node to all possible destination nodes.

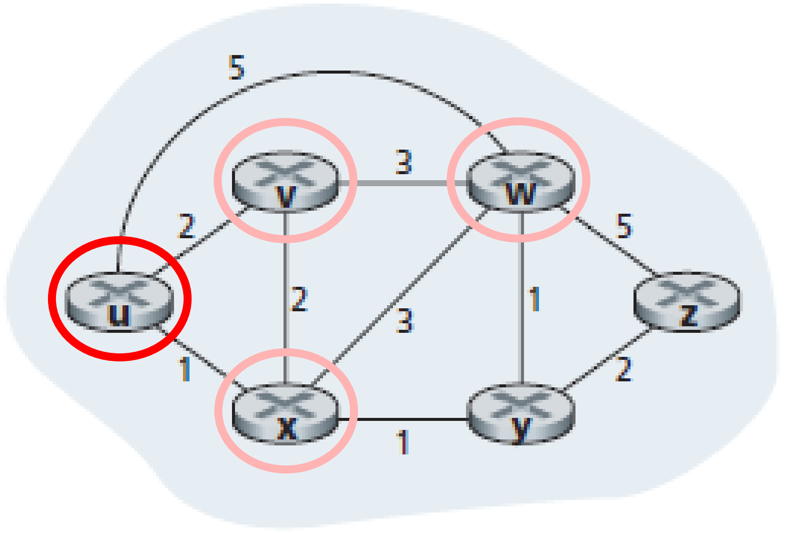
- During the **initialization** phase:
  - We assume that **all costs are known**!
  - Only **distances to neighbors are set**.

- During the **construction** phase:
  - Select the **closest** node w.
  - **Explore the neighborhood** of w, checking if the path through w is better than the current one.
  - Exit when **all nodes have been explored**.

Network Layer

Routing: Link-State Algorithm

- Let's consider the previous 6-nodes example. Here we show how the LS algorithm works when invoked on the u node:
  - In this case, we will use an additional function p(x) to store the node preceding the selected one.
  - If we want to retrieve the path, we can just backtrack all predecessors until the current node is reached.

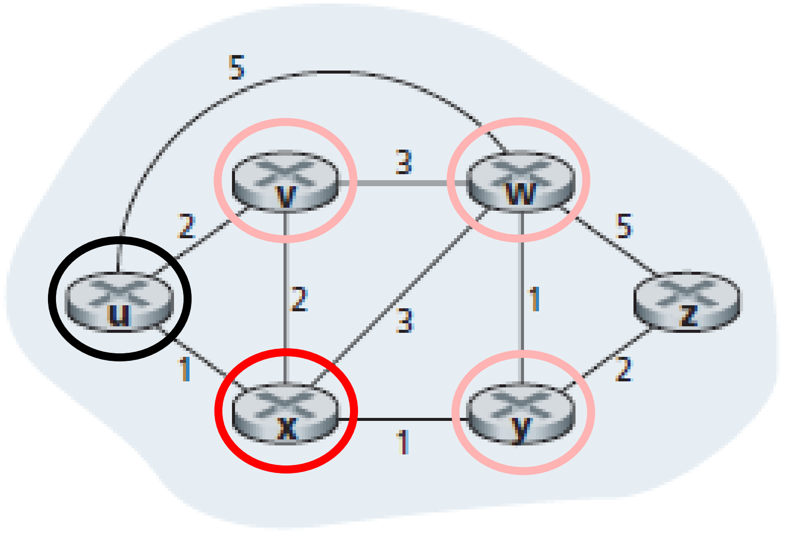


step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Network Layer

Routing: Link-State Algorithm

- Let’s consider the previous 6-nodes example. Here we show how the LS algorithm works when invoked on the u node:
  - In this case, we will use an additional function p(x) to store the node preceding the selected one.
  - If we want to retrieve the path, we can just backtrack all predecessors until the current node is reached.

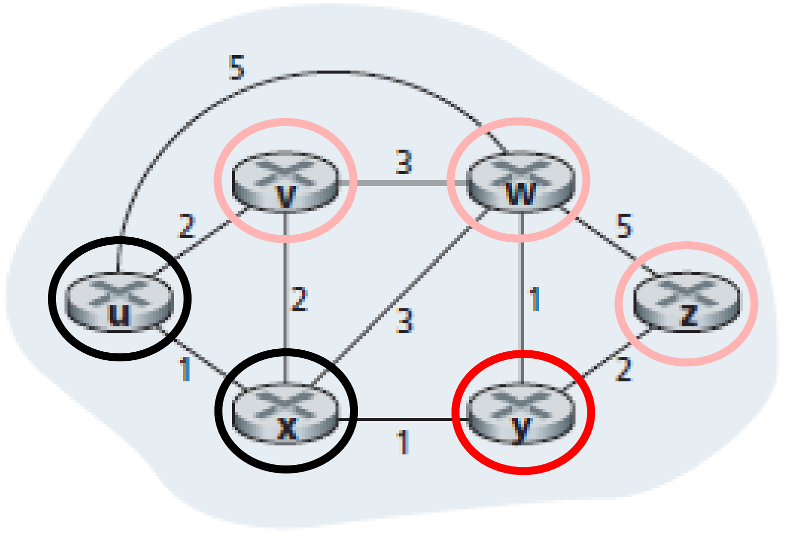


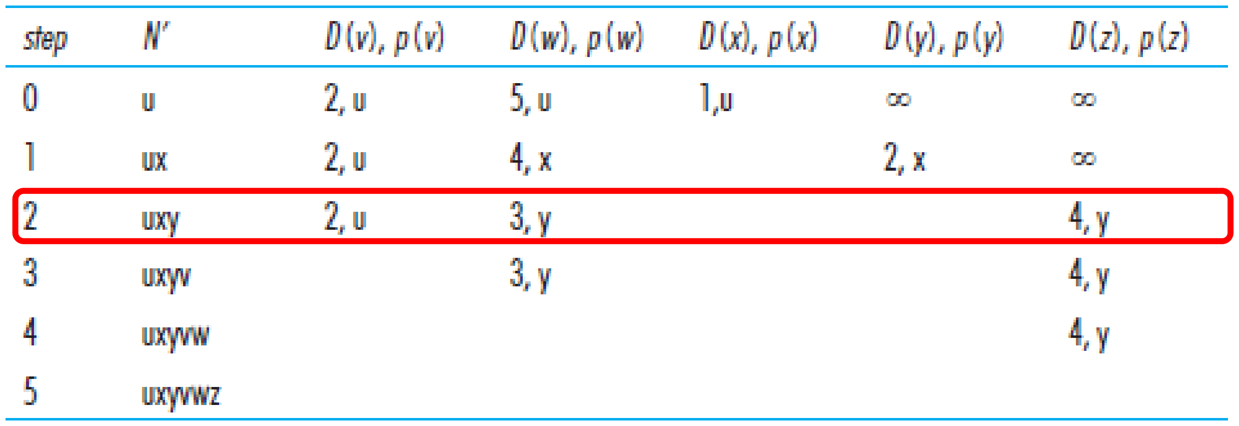
step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Network Layer

Routing: Link-State Algorithm

- Let’s consider the previous 6-nodes example. Here we show how the LS algorithm works when invoked on the u node:
  - In this case, we will use an additional function p(x) to store the node preceding the selected one.
  - If we want to retrieve the path, we can just backtrack all predecessors until the current node is reached.



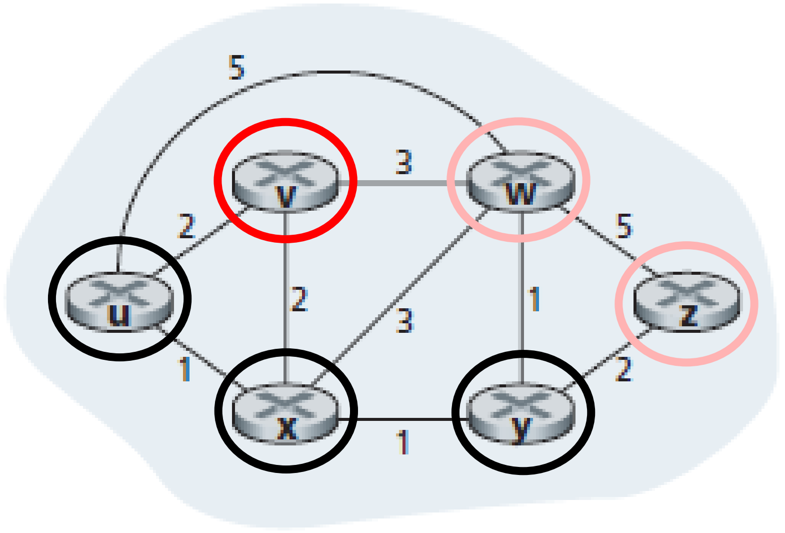


step	N′	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Network Layer

Routing: Link-State Algorithm

- Let's consider the previous 6-nodes example. Here we show how the LS algorithm works when invoked on the u node:
  - In this case, we will use an additional function p(x) to store the node preceding the selected one.
  - If we want to retrieve the path, we can just backtrack all predecessors until the current node is reached.

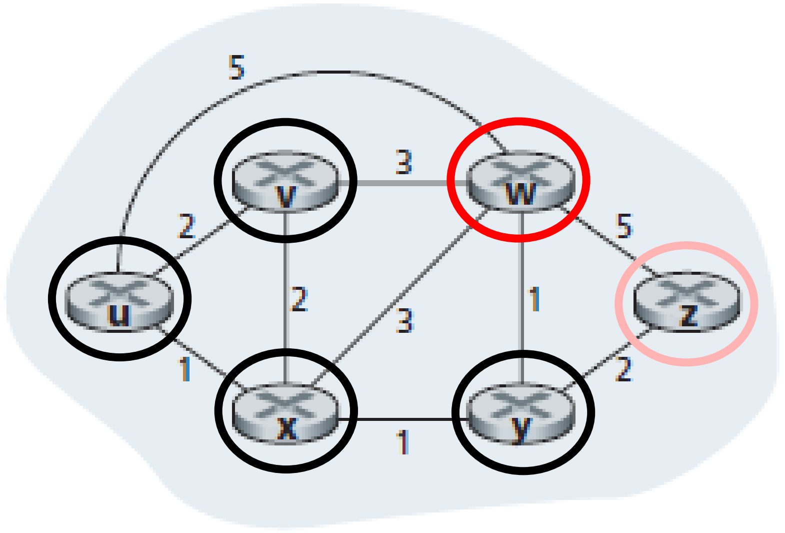


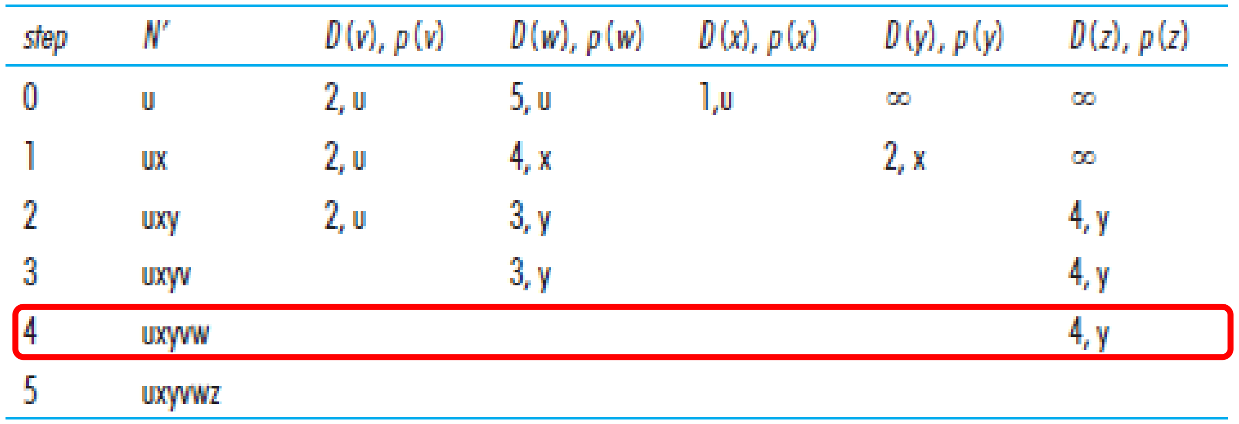
step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Network Layer

Routing: Link-State Algorithm

- Let's consider the previous 6-nodes example. Here we show how the LS algorithm works when invoked on the u node:
  - In this case, we will use an additional function p(x) to store the node preceding the selected one.
  - If we want to retrieve the path, we can just backtrack all predecessors until the current node is reached.



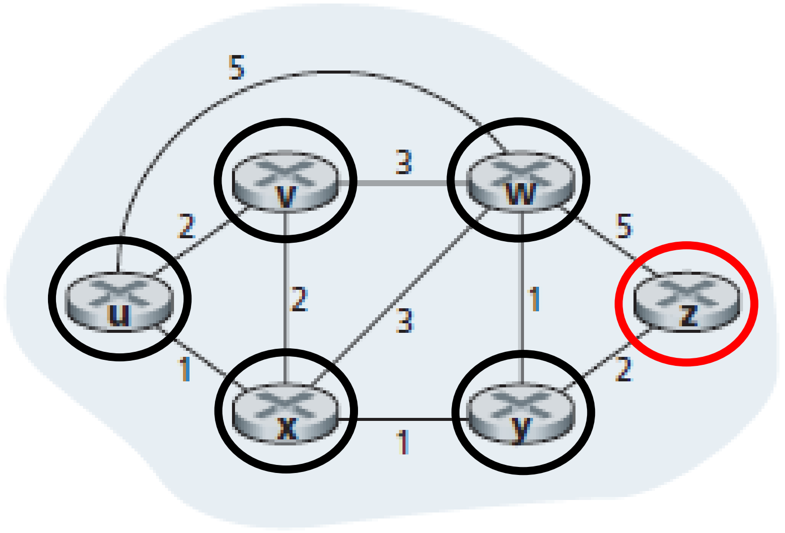


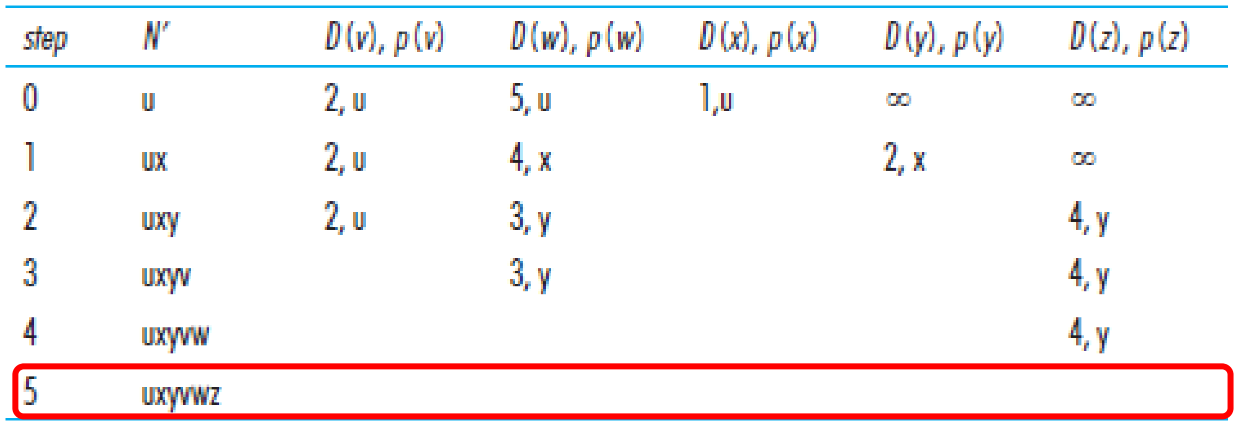
step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Network Layer

Routing: Link-State Algorithm

- Let’s consider the previous 6-nodes example. Here we show how the LS algorithm works when invoked on the u node:
  - In this case, we will use an additional function p(x) to store the node preceding the selected one.
  - If we want to retrieve the path, we can just backtrack all predecessors until the current node is reached.





step	N′	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1,u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Network Layer
Routing: Link-State Algorithm

- As for the computational complexity, **at each iteration we check all nodes** of the network, except the source node, then a new source is selected. This means that **we are removing one node form each iteration**, so the total number of iterations is $|N|(|N| + 1)/2$ which **is $\boldsymbol{O}(|\boldsymbol{N}|^2)$ in the worst case**.
- This version of the algorithm is quite basic, complexity can be reduced by introducing more sophisticated data-structures (e.g., heaps), **we can reach better performance**: $O(|N| + |E| \log(|E|))$.

- Pros:
  - **Faster convergence**: all communications are performed simultaneously.
  - **No count-to-infinity**: the state of all links is propagated.
- Cons:
  - **Synchronous**: nodes must receive information from the whole network before to start.

Network Layer
Traceroute

- In Linux we can use **traceroute** to trace the route of our packets toward a specific destination:
  - Install traceroute:
    - $ sudo apt-get install traceroute
  - Trace a route:
    - $ traceroute ADDRESS

- The traceroute command will return **IPs of all devices encountered** from the source host to the destination host.

- If **we try it several times** targeting a distant host (e.g., google.com), **we should see different devices** in the list (due to routes updating).

Network Layer
Routing: DV vs. LS

- The **DV and LS algorithms take complementary approaches** toward computing routing:
  - In the DV algorithm exploits **local information** about neighbors.
  - The LS algorithm requires **global information** about all nodes.

- **Message complexity**: **LS is more complex** as synchronous communication between all nodes is needed. In DV, communication is needed only if a best path changes.
- **Speed of convergence**: **DV requires time to converge**, in the meantime we can have suboptimal paths or loops. DV algorithms also suffer of count-to-infinity problem.
- **Robustness**: **LS is considered more robust** as **forwarding tables are calculated separately**. Each node receives information from all other nodes and creates its own table. In DV all computations are chained, each node depends on the table of others. If one table is incorrect, all tables get the error.
  - In 1997 a **malfunctioning router from a small ISP caused a chain reaction**, which flooded backbone routers, and disconnected part of Internet for several hours.

- In the end, there is no winner, **both solutions are used in Internet**.