Distance-Vector Routing

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

November 30, 2018

Outline

- Recap on link-state routing
- Distance-vector routing
- Bellman-Ford equation
- Distance-vector algorithm
- Examples

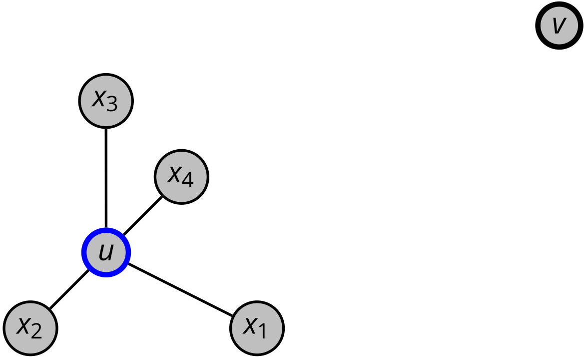
- Goal: each router *u* must compute, for every other router *v*, the next-hop neighbor *x* that is on the least-cost path from *u* to *v*

Recap on Routing

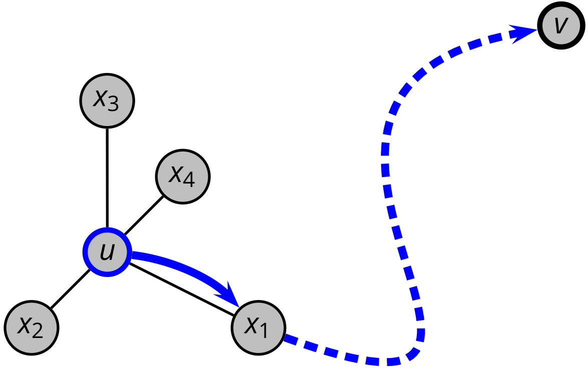
- Goal: each router *u* must compute, for every other router *v*, the next-hop neighbor *x* that is on the least-cost path from *u* to *v*

Recap on Routing

- Goal: each router u must compute, for every other router v, the next-hop neighbor x that is on the least-cost path from u to v



- Goal: each router u must compute, for every other router v, the next-hop neighbor x that is on the least-cost path from u to v



Recap on Link-State Routing

Recap on Link-State Routing

- Every router broadcast a *link-state advertisement (LSA)* containing the costs of its adjacent links

Recap on Link-State Routing

- Every router broadcast a *link-state advertisement (LSA)* containing the costs of its adjacent links
- Routers use LSAs from other routers to compile an image of the entire network

Recap on Link-State Routing

- Every router broadcast a *link-state advertisement (LSA)* containing the costs of its adjacent links
- Routers use LSAs from other routers to compile an image of the entire network
- With a complete knowledge of the network topology, routers perform a local computation (Dijkstra’s algorithm) to find the least-cost paths to every other router

Recap on Link-State Routing

- Every router broadcast a *link-state advertisement (LSA)* containing the costs of its adjacent links
- Routers use LSAs from other routers to compile an image of the entire network
- With a complete knowledge of the network topology, routers perform a local computation (Dijkstra's algorithm) to find the least-cost paths to every other router
- In essence
  - *broadcast transmission of topology information*
  - *global knowledge of the network*
  - *local computation*

Changes in Link Costs

Changes in Link Costs

- Routers monitor the state of their adjacent links
  - e.g., measuring the round-trip time using a local “ping” protocol

Changes in Link Costs

- Routers monitor the state of their adjacent links
  - e.g., measuring the round-trip time using a local “ping” protocol

- The measured costs are used to build LSAs, which are issued also at regular intervals

Changes in Link Costs

- Routers monitor the state of their adjacent links
  - e.g., measuring the round-trip time using a local “ping” protocol
- The measured costs are used to build LSAs, which are issued also at regular intervals
- Changes in link costs are propagated quickly to all routers
- Routers can then react by recomputing paths and by updating their forwarding tables accordingly
  - in fact, this “reaction” is not different from the normal behavior of the protocol

Distance-Vector Routing

- Every router u maintains a *"distance vector"*
  - v is a destination node in the network
  - $D_u[v]$ is the best known distance between u and v
  - $n_u[v]$ is the next-hop router on the best known path to v

Distance-Vector Routing

- Every router u maintains a *"distance vector"*
  - v is a destination node in the network
  - $D_u[v]$ is the best known distance between u and v
  - $n_u[v]$ is the next-hop router on the best known path to v
- Routers exchange their distance vectors with their neighbors

Distance-Vector Routing

- Every router u maintains a *"distance vector"*
  - v is a destination node in the network
  - $D_u[v]$ is the best known distance between u and v
  - $n_u[v]$ is the next-hop router on the best known path to v

- Routers exchange their distance vectors with their neighbors

- If the distance vector of a neighbor leads to a better path to some destinations, the router updates its distance vector and sends it out again to its neighbors

- Every router u maintains a *"distance vector"*
  - v is a destination node in the network
  - $D_u[v]$ is the best known distance between u and v
  - $n_u[v]$ is the next-hop router on the best known path to v

- Routers exchange their distance vectors with their neighbors

- If the distance vector of a neighbor leads to a better path to some destinations, the router updates its distance vector and sends it out again to its neighbors

- After a number of iterations, *the algorithm converges to a point where every router has a minimal distance vector*

- Local transmission of topology information
  - routers exchange their distance vectors only with their neighbors
  - no broadcast protocol needed (a *local broadcast* can be useful)

- Local transmission of topology information
  - routers exchange their distance vectors only with their neighbors
  - no broadcast protocol needed (a *local broadcast* can be useful)
- Local knowledge of the network
  - router u knows its distance $D_u[v]$ and the first step along that path
  - router u does not know about any link cost except its adjacent links

- Local transmission of topology information
  - routers exchange their distance vectors only with their neighbors
  - no broadcast protocol needed (a *local broadcast* can be useful)
- Local knowledge of the network
  - router u knows its distance $D_u[v]$ and the first step along that path
  - router u does not know about any link cost except its adjacent links
- Global computation
  - the computation is actually distributed

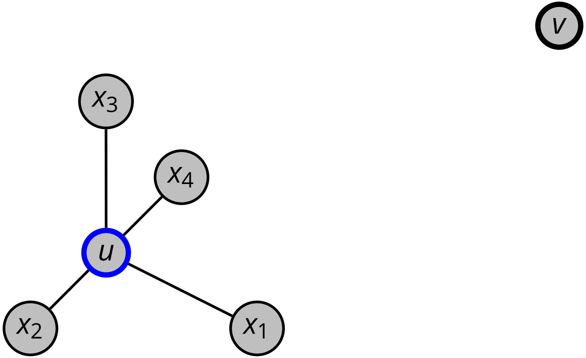
Intuition

- The main idea behind the distance-vector algorithm is expressed well by the *Bellman-Ford equation*

$$D'_u[v] = \min_{x \in neighbors(u)} (c(u,x) + D_x[v])$$

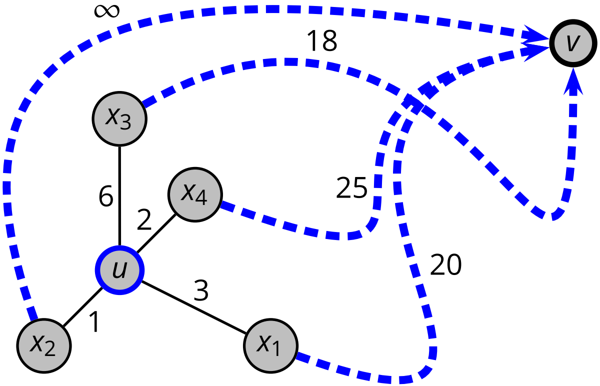
- The main idea behind the distance-vector algorithm is expressed well by the *Bellman-Ford equation*

$$D'_u[v] = \min_{x \in neighbors(u)} (c(u,x) + D_x[v])$$



- The main idea behind the distance-vector algorithm is expressed well by the *Bellman-Ford equation*

$$D'_u[v] = \min_{x \in neighbors(u)} (c(u,x) + D_x[v])$$



- Executing locally at node u

Distance-Vector Algorithm

- Executing locally at node *u*
- Variables storing values known at each iteration

Distance-Vector Algorithm

- Executing locally at node u
- Variables storing values known at each iteration
  - $D_u[v]$, cost of the least-cost path from u to v (distance vector)

Distance-Vector Algorithm

- Executing locally at node u
- Variables storing values known at each iteration
  - $D_u[v]$, cost of the least-cost path from u to v (distance vector)
  - $n_u[v]$, next-hop node (neighbor of u) on the least-cost path from u to v

- Executing locally at node u
- Variables storing values known at each iteration
  - $D_u[v]$, cost of the least-cost path from u to v (distance vector)
  - $n_u[v]$, next-hop node (neighbor of u) on the least-cost path from u to v
  - $D_x[v]$, distance vectors of every neighbor node x

Distance-Vector Algorithm: Initialization

$\triangleright$ Initialization

for $v \in V$
    do if $v \in neighbors(u)$
        then $D_u[v] \leftarrow c(u, v)$
            $n_u[v] \leftarrow v$
        else $D_u[v] \leftarrow \infty$
for $x \in neighbors(u)$
    do for $v \in V$
        do $D_x[v] \leftarrow \infty$
send D_u to all neighbor nodes

Distance-Vector Algorithm: Loop

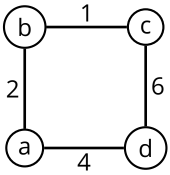
when D'_x is received from neighbor x
  do $D_x \leftarrow D'_x$
    for $v \in N$
      do $D_u[v] \leftarrow \min_{x \in neighbors(u)}(c(u,x) + D_x[v])$
    if D_u was updated
      then send D_u to all neighbor nodes

when link cost $c(u,x)$ changes
  do for $v \in N$
      do $D_u[v] \leftarrow \min_{x \in neighbors(u)}(c(u,x) + D_x[v])$
    if D_u was updated
      then send D_u to all neighbor nodes

Distance-Vector Algorithm: D_u Update

$\triangleright$ updating D_u:
$\triangleright$ $\forall v \in N : D_u[v] \leftarrow \min_{x \in neighbors(u)}(c(u,x) + D_x[v])$
$updated \leftarrow$ FALSE
for $v \in N$
    do for $x \in neighbors(u)$
        do if $D_u[v] > c(u,x) + D_x[v]$
            then $D_u[v] \leftarrow c(u,x) + D_x[v]$
                $n_u[v] \leftarrow x$
                $updated \leftarrow$ TRUE

Example



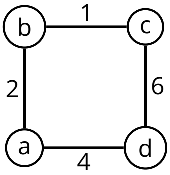
Example

a	a	b	c	d
D_a	0	2	∞	4
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

b	a	b	c	d
D_b	2	0	1	∞
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

c	a	b	c	d
D_c	∞	1	0	6
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

d	a	b	c	d
D_d	4	∞	6	0
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞



Example

a	a	b	c	d
D_a	0	2	∞	4
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

a	a	b	c	d
D_a	0	2	**3**	4
D_b	2	0	1	∞
D_d	4	∞	6	0

b	a	b	c	d
D_b	2	0	1	∞
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

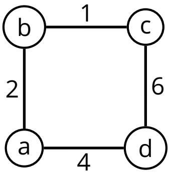
b	a	b	c	d
D_b	2	0	1	**6**
D_a	0	2	∞	4
D_c	∞	1	0	6

c	a	b	c	d
D_c	∞	1	0	6
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

c	a	b	c	d
D_c	**3**	1	0	6
D_b	2	0	1	∞
D_d	4	∞	6	0

d	a	b	c	d
D_d	4	∞	6	0
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

d	a	b	c	d
D_d	4	**6**	6	0
D_a	0	2	∞	4
D_c	∞	1	0	6



Example

a	a	b	c	d
D_a	0	2	∞	4
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

a	a	b	c	d
D_a	0	2	**3**	4
D_b	2	0	1	∞
D_d	4	∞	6	0

a	a	b	c	d
D_a	0	2	3	4
D_b	2	0	1	6
D_d	4	6	6	0

b	a	b	c	d
D_b	2	0	1	∞
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

b	a	b	c	d
D_b	2	0	1	**6**
D_a	0	2	∞	4
D_c	∞	1	0	6

b	a	b	c	d
D_b	2	0	1	6
D_a	0	2	3	4
D_c	3	1	0	6

c	a	b	c	d
D_c	∞	1	0	6
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

c	a	b	c	d
D_c	**3**	1	0	6
D_b	2	0	1	∞
D_d	4	∞	6	0

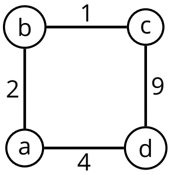
c	a	b	c	d
D_c	3	1	0	6
D_b	2	0	1	6
D_d	4	6	6	0

d	a	b	c	d
D_d	4	∞	6	0
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

d	a	b	c	d
D_d	4	**6**	6	0
D_a	0	2	∞	4
D_c	∞	1	0	6

d	a	b	c	d
D_d	4	6	6	0
D_a	0	2	3	4
D_c	3	1	0	6

Example (2)



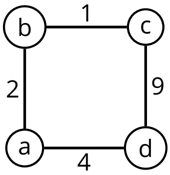
Example (2)

a	a	b	c	d
D_a	0	2	∞	4
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

b	a	b	c	d
D_b	2	0	1	∞
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

c	a	b	c	d
D_c	∞	1	0	9
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

d	a	b	c	d
D_d	4	∞	9	0
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞



Example (2)

a	a	b	c	d
D_a	0	2	∞	4
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

a	a	b	c	d
D_a	0	2	**3**	4
D_b	2	0	1	∞
D_d	4	∞	9	0

b	a	b	c	d
D_b	2	0	1	∞
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

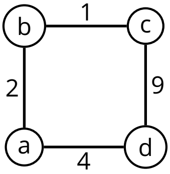
b	a	b	c	d
D_b	2	0	1	**6**
D_a	0	2	∞	4
D_c	∞	1	0	9

c	a	b	c	d
D_c	∞	1	0	9
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

c	a	b	c	d
D_c	**3**	1	0	9
D_b	2	0	1	∞
D_d	4	∞	9	0

d	a	b	c	d
D_d	4	∞	9	0
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

d	a	b	c	d
D_d	4	**6**	9	0
D_a	0	2	∞	4
D_c	∞	1	0	9



Example (2)

a	a	b	c	d
D_a	0	2	∞	4
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

b	a	b	c	d
D_b	2	0	1	∞
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

c	a	b	c	d
D_c	∞	1	0	9
D_b	∞	∞	∞	∞
D_d	∞	∞	∞	∞

d	a	b	c	d
D_d	4	∞	9	0
D_a	∞	∞	∞	∞
D_c	∞	∞	∞	∞

a	a	b	c	d
D_a	0	2	**3**	4
D_b	2	0	1	∞
D_d	4	∞	9	0

b	a	b	c	d
D_b	2	0	1	**6**
D_a	0	2	∞	4
D_c	∞	1	0	9

c	a	b	c	d
D_c	**3**	1	0	9
D_b	2	0	1	∞
D_d	4	∞	9	0

d	a	b	c	d
D_d	4	**6**	9	0
D_a	0	2	∞	4
D_c	∞	1	0	9

a	a	b	c	d
D_a	0	2	3	4
D_b	2	0	1	6
D_d	4	6	9	0

b	a	b	c	d
D_b	2	0	1	6
D_a	0	2	3	4
D_c	3	1	0	9

c	a	b	c	d
D_c	3	1	0	**7**
D_b	2	0	1	6
D_d	4	6	9	0

d	a	b	c	d
D_d	4	6	**7**	0
D_a	0	2	3	4
D_c	3	1	0	9