# Basics of Routing and Link-State Routing

Antonio Carzaniga

Faculty of Informatics Università della Svizzera italiana

December 12, 2016

#### **Outline**

- Routing problem
- Graph model
- Classes of routing algorithms
- Broadcast routing
- Link-state routing
- Dijkstra's algorithm

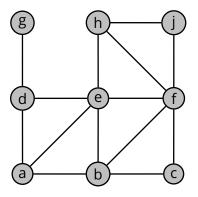


## **Routing Problem**

Finding paths through a network

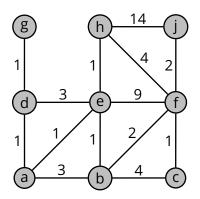
## **Routing Problem**

Finding paths through a network



## **Routing Problem**

Finding paths through a network



**Example:**  $a \rightarrow j$ ?

$$G = (V, E)$$

■ The network is modeled as a graph

$$G = (V, E)$$

► *V* is a set of *vertices* representing the routers

$$G = (V, E)$$

- ▶ *V* is a set of *vertices* representing the routers
- ►  $E \subseteq V \times V$  is a set of **edges** representing communication links
  - e.g.,  $(u, v) \in E$  iff router u is on the same subnet as v

$$G = (V, E)$$

- ▶ *V* is a set of *vertices* representing the routers
- ▶  $E \subseteq V \times V$  is a set of *edges* representing communication links
  - e.g.,  $(u, v) \in E$  iff router u is on the same subnet as v
- G is assumed to be an undirected graph
  - i.e.,  $(u, v) \in E \Leftrightarrow (v, u) \in E$  for all  $u, v \in N$

$$G = (V, E)$$

- ▶ *V* is a set of *vertices* representing the routers
- ►  $E \subseteq V \times V$  is a set of **edges** representing communication links
  - e.g.,  $(u, v) \in E$  iff router u is on the same subnet as v
- ► *G* is assumed to be an *undirected graph* 
  - i.e.,  $(u, v) \in E \Leftrightarrow (v, u) \in E$  for all  $u, v \in N$
- ▶ A **cost** function  $c: F \to \mathbb{R}$ 
  - ▶ costs are always positive: c(e) > 0 for all  $e \in E$
  - ▶ links are symmetric: c(u, v) = c(v, u) for all  $u, v \in N$

■ For every router  $u \in V$ , for every other router  $v \in V$ , compute the path  $P_{u \to v} = u, x_1, x_2, \dots, x_n, v$  such that

- For every router  $u \in V$ , for every other router  $v \in V$ , compute the path  $P_{u \to v} = u, x_1, x_2, \dots, x_n, v$  such that
  - ▶  $P_{u \to v}$  is completely contained in the network graph G. I.e.,  $(u, x_1) \in V, (x_1, x_2) \in V, \dots, (x_n, v) \in V$

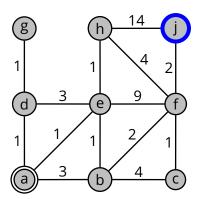
- For every router  $u \in V$ , for every other router  $v \in V$ , compute the path  $P_{u \to v} = u, x_1, x_2, \dots, x_n, v$  such that
  - ▶  $P_{u \to v}$  is completely contained in the network graph G. I.e.,  $(u, x_1) \in V, (x_1, x_2) \in V, \ldots, (x_n, v) \in V$
  - ►  $P_{u \to v}$  is a *least-cost path*, where the cost of the path is  $c(P_{u \to v}) = c(u, x_1) + c(x_1, x_2) + ... + c(x_n, v)$

- For every router  $u \in V$ , for every other router  $v \in V$ , compute the path  $P_{u \to v} = u, x_1, x_2, \dots, x_n, v$  such that
  - ▶  $P_{u \to v}$  is completely contained in the network graph G. I.e.,  $(u, x_1) \in V, (x_1, x_2) \in V, \ldots, (x_n, v) \in V$
  - ▶  $P_{u \to v}$  is a *least-cost path*, where the cost of the path is  $c(P_{u \to v}) = c(u, x_1) + c(x_1, x_2) + ... + c(x_n, v)$
- Compile *u*'s forwarding table by adding the following entry:

$$A(v) \rightarrow I_u(x_1)$$

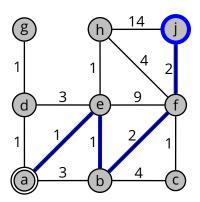
- $\triangleright$  A(v) is the address (or set of addresses) of router v
- ►  $I_u(x_1)$  is the interface that connects u to the first next-hop router  $x_1$  in  $P_{u\to v}=u, x_1, x_2, \ldots, x_n, v$

### **Back To The Example**



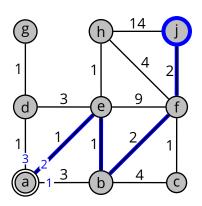
**Example:**  $a \rightarrow j$ 

### **Back To The Example**



- **Example:**  $a \rightarrow j$ 
  - ▶ least-cost path is  $P_{a \rightarrow j} = a, e, b, f, j$

#### **Back To The Example**



- **Example:**  $a \rightarrow j$ 
  - ▶ least-cost path is  $P_{a \rightarrow j} = a, e, b, f, j$
  - a's forwarding table will contain an entry  $|j \rightarrow 2|$  since  $I_a(e) = 2$

■ There are two main strategies to implement a routing algorithm

- There are two main strategies to implement a routing algorithm
- Link-state routing

There are two main strategies to implement a routing algorithm

#### ■ Link-state routing

- global view of the network
- local computation of least-cost paths

There are two main strategies to implement a routing algorithm

#### ■ Link-state routing

- global view of the network
- local computation of least-cost paths

#### Distance-vector routing

■ There are two main strategies to implement a routing algorithm

#### ■ Link-state routing

- global view of the network
- local computation of least-cost paths

#### Distance-vector routing

- local view of the network
- global computation of least-cost paths

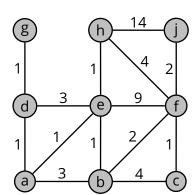
■ Router *u* maintains a complete view of the network graph *G* (including all links and their costs)

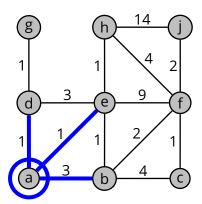
- Router *u* maintains a complete view of the network graph *G* (including all links and their costs)
  - every router v advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
  - ► *link-state advertisements (LSAs)* are broadcast through the entire network

- Router *u* maintains a complete view of the network graph *G* (including all links and their costs)
  - every router v advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
  - ► *link-state advertisements (LSAs)* are broadcast through the entire network
  - routers collect link-state advertisements from other routers, and they use them to compile and maintain a complete view of G

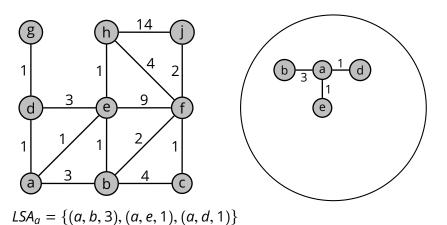
- Router *u* maintains a complete view of the network graph *G* (including all links and their costs)
  - every router v advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
  - ► *link-state advertisements (LSAs)* are broadcast through the entire network
  - routers collect link-state advertisements from other routers, and they use them to compile and maintain a complete view of G
- Using its local representation of *G*, router *u* computes the least-cost paths from *u* to every other router in the network

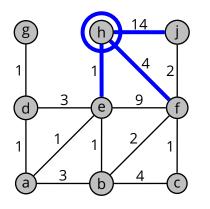
- Router *u* maintains a complete view of the network graph *G* (including all links and their costs)
  - every router v advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
  - link-state advertisements (LSAs) are broadcast through the entire network
  - routers collect link-state advertisements from other routers, and they use them to compile and maintain a complete view of G
- Using its local representation of *G*, router *u* computes the least-cost paths from *u* to every other router in the network
  - the computation is local

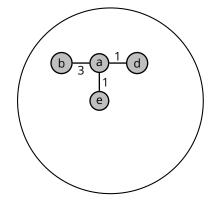




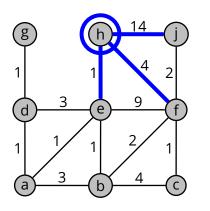
 $LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$ 

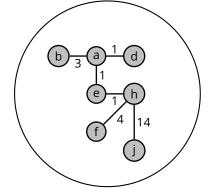




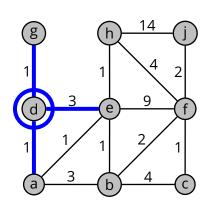


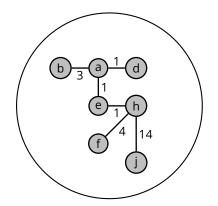
 $LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$  $LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$ 



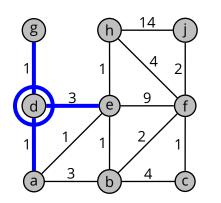


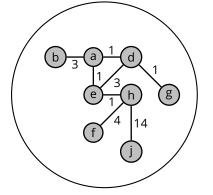
 $LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$  $LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$ 



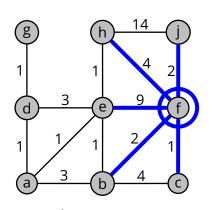


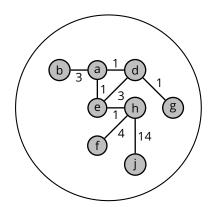
 $LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$   $LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$   $LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$ 





 $LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$   $LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$  $LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$ 





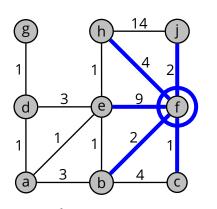
$$LSA_{a} = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$

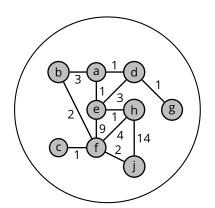
$$LSA_{h} = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$

$$LSA_{d} = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$

$$LSA_{f} = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}$$

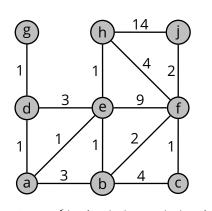
### **Link-State Advertisements**

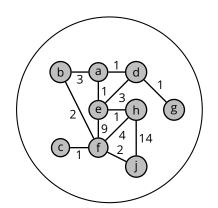




```
LSA_{a} = \{(a, b, 3), (a, e, 1), (a, d, 1)\}
LSA_{h} = \{(h, e, 1), (h, f, 4), (h, j, 14)\}
LSA_{d} = \{(d, a, 1), (d, g, 1), (d, e, 3)\}
LSA_{f} = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}
```

### **Link-State Advertisements**





```
LSA_{a} = \{(a, b, 3), (a, e, 1), (a, d, 1)\}
LSA_{h} = \{(h, e, 1), (h, f, 4), (h, j, 14)\}
LSA_{d} = \{(d, a, 1), (d, g, 1), (d, e, 3)\}
LSA_{f} = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}
```

. .

# **Link-State Routing Ingredients**

What do we need to implement link-state routing?

# **Link-State Routing Ingredients**

What do we need to implement link-state routing?

■ Every router sends its LSA to every other router in the network, so we need a *broadcast routing scheme* 

# **Link-State Routing Ingredients**

What do we need to implement link-state routing?

- Every router sends its LSA to every other router in the network, so we need a *broadcast routing scheme*
- Once we have all the LSAs from every router, and therefore we complete knowledge of G, we need an algorithm to compute least-cost paths in a graph



#### **■** Flooding

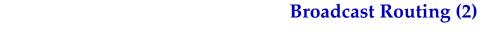
 every router forwards a broadcast packet to every adjacent router, except the one that sent the packet

- every router forwards a broadcast packet to every adjacent router, except the one that sent the packet
- Simple and elegant

- every router forwards a broadcast packet to every adjacent router, except the one that sent the packet
- Simple and elegant
- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router

- every router forwards a broadcast packet to every adjacent router, except the one that sent the packet
- Simple and elegant
- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router
- Any problem with this solution?

- every router forwards a broadcast packet to every adjacent router, except the one that sent the packet
- Simple and elegant
- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router
- Any problem with this solution?
  - cycles in the network create packet storms



- every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
- a router u accepts a broadcast packet p originating at router s only if p arrives on the link that is on the direct (unicast) path from u to s

- every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
- a router u accepts a broadcast packet p originating at router s only if p arrives on the link that is on the direct (unicast) path from u to s
- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router

- every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
- a router u accepts a broadcast packet p originating at router s only if p arrives on the link that is on the direct (unicast) path from u to s
- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router
  - No packet storms even in the presence of cycles in *G*

- every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
- a router u accepts a broadcast packet p originating at router s only if p arrives on the link that is on the direct (unicast) path from u to s
- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router
- No packet storms even in the presence of cycles in G
- Any problem with this solution?

- every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
- a router u accepts a broadcast packet p originating at router s only if p arrives on the link that is on the direct (unicast) path from u to s
- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router
- No packet storms even in the presence of cycles in *G*
- Any problem with this solution?
  - it requires (unicast) routing information
  - so it is obviously useless to implement a routing algorithm

#### ■ Sequence-number controlled flooding

• the originator s of a broadcast packet marks the packet with a sequence number  $n_s$ 

- the originator s of a broadcast packet marks the packet with a sequence number  $n_s$
- every router u stores the most recent sequence number seen from each source router. Let's assume that u has seen sequence numbers from s up to n<sub>s</sub>

- the originator s of a broadcast packet marks the packet with a sequence number  $n_s$
- every router u stores the most recent sequence number seen from each source router. Let's assume that u has seen sequence numbers from s up to n<sub>s</sub>
- ▶ a router accepts a broadcast packet p originating at s only if p carries a sequence number seq(p) that is higher than the most recent one seen from s:  $seq(p) > n_s$

- the originator s of a broadcast packet marks the packet with a sequence number  $n_s$
- every router u stores the most recent sequence number seen from each source router. Let's assume that u has seen sequence numbers from s up to n<sub>s</sub>
- ▶ a router accepts a broadcast packet p originating at s only if p carries a sequence number seq(p) that is higher than the most recent one seen from s:  $seq(p) > n_s$
- accepted packets are forwarded to every adjacent router, except the previous-hop router

- the originator s of a broadcast packet marks the packet with a sequence number  $n_s$
- every router u stores the most recent sequence number seen from each source router. Let's assume that u has seen sequence numbers from s up to n<sub>s</sub>
- ▶ a router accepts a broadcast packet p originating at s only if p carries a sequence number seq(p) that is higher than the most recent one seen from s:  $seq(p) > n_s$
- accepted packets are forwarded to every adjacent router, except the previous-hop router
- ▶ *u* updates its table of sequence numbers  $n_s \leftarrow seq(p)$

■ Executing locally at node *u* 

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

- Executing locally at node u
- Variables storing values known at each iteration
  - ▶ D[v], cost of the least-cost path from u to v

- Executing locally at node u
- Variables storing values known at each iteration
  - ▶ D[v], cost of the least-cost path from u to v
  - ▶ p[v], node preceding v (neighbor of v) on the least-cost path from u to v

- Executing locally at node u
- Variables storing values known at each iteration
  - ▶ D[v], cost of the least-cost path from u to v
  - ▶ p[v], node preceding v (neighbor of v) on the least-cost path from u to v
  - ▶ *N*, nodes of *G* whose least-cost path from *u* is definitely known

```
DIJKSTRA(G = (V, E), u)
    N \leftarrow \{u\}
 2 for all v \in V
            do if v \in neighbors(u)
 4
                   then D[v] \leftarrow c(u, v)
 5
                          p[v] \leftarrow u
 6
                   else D[v] \leftarrow \infty
     while N \neq V
 8
            do find w \notin N such that D[w] is minimum
 9
                N \leftarrow N \cup \{w\}
10
                for all v \in neighbors(w) \setminus N
                      do if D[w] + c(w, v) < D[v]
11
12
                             then D[v] \leftarrow D[w] + c(w, v)
13
                                    p[v] \leftarrow w
```

## Example

14

```
DIJKSTRA(G = (V, E), u)
     N \leftarrow \{u\}
     for all v \in V
 3
            do if v \in neighbors(u)
 4
5
6
                    then D[v] \leftarrow c(u, v)
                           p[v] \leftarrow u
                    else D[v] \leftarrow \infty
 7
      while N \neq V
 8
9
            do find w \notin N such that D[w] is minimum
                 N \leftarrow N \cup \{w\}
10
                 for all v \in neighbors(w) \setminus N
11
                       do if D[w] + c(w, v) < D[v]
12
                              then D[v] \leftarrow D[w] + c(w, v)
13
                                     p[v] \leftarrow w
```