Basics of Routing and Link-State Routing

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

December 6, 2016

Outline

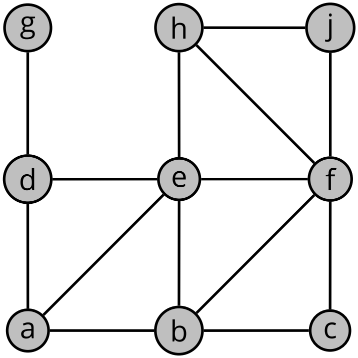
- Routing problem
- Graph model
- Classes of routing algorithms
- Broadcast routing
- Link-state routing
- Dijkstra’s algorithm

Routing Problem

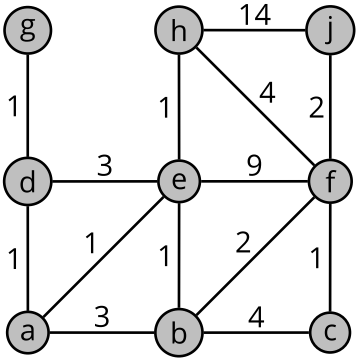
Routing Problem

- Finding paths through a network

- Finding paths through a network



- Finding paths through a network



- Example: $a \longrightarrow j$?

- The network is modeled as a graph

$$G = (V, E)$$

- The network is modeled as a graph

$$G = (V, E)$$

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

- The network is modeled as a graph

$$G = (V, E)$$

  - 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

- The network is modeled as a graph

$$G = (V, E)$$

  - 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***, meaning that ***links are bidirectional***
    - i.e., $(u, v) \in E \Leftrightarrow (v, u) \in E$ for all $u, v \in N$

- The network is modeled as a graph

$$G = (V, E)$$

  - 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***, meaning that ***links are bidirectional***
    - i.e., $(u, v) \in E \Leftrightarrow (v, u) \in E$ for all $u, v \in N$
  - A ***cost*** function $c : E \rightarrow \mathbb{R}$
    - 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 $P_{u \to v} = u, x_1, x_2, \ldots, x_n, v$ such that

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

- For every router $u \in V$, for every other router $v \in V$, compute the path $P_{u \to v} = u, x_1, x_2, \ldots, x_n, v$ 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 $c(P_{u \to v}) = c(u, x_1) + c(x_1, x_2) + \ldots + c(x_n, v)$

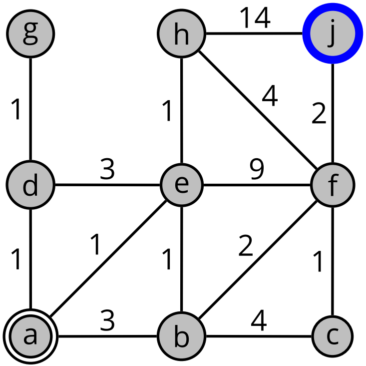
- For every router $u \in V$, for every other router $v \in V$, compute the path $P_{u \to v} = u, x_1, x_2, \ldots, x_n, v$ such that
  - $P_{u \to v}$ is completely contained in the network graph G. I.e., $(u, x_1) \in V, (x_1, x_2) \in V, \ldots, (x_n, v) \in V$
  - $P_{u \to v}$ is a *least-cost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

- Compile u's forwarding table by adding the following entry:

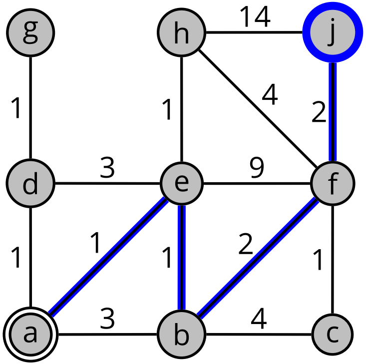
$$A(v) \longrightarrow I_u(x_1)$$

  - $A(v)$ 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 $P_{u \to v} = u, x_1, x_2, \ldots, x_n, v$

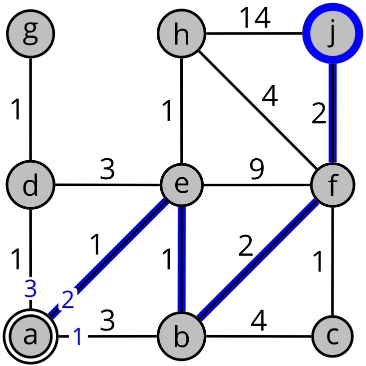
Back To The Example



- Example: $a \longrightarrow j$



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



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

Two General Strategies

- There are two main strategies to implement a routing algorithm

Two General Strategies

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

Two General Strategies

- There are two main strategies to implement a routing algorithm
- ***Link-state routing***
  - global view of the network
  - local computation of least-cost paths

Two General Strategies

- 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***

Two General Strategies

- 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

Link-State Routing

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

Link-State Routing

- 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

Link-State Routing

- 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*

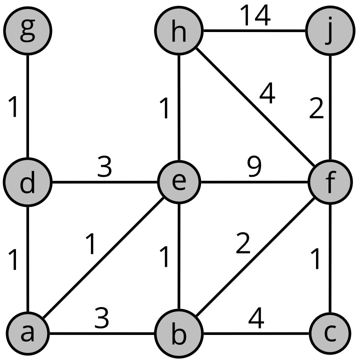
Link-State Routing

- 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

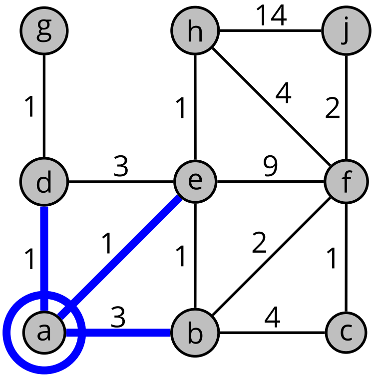
Link-State Routing

- 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

Link-State Advertisements

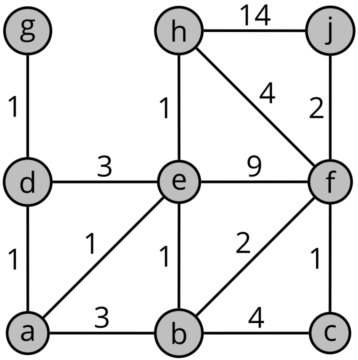


Link-State Advertisements

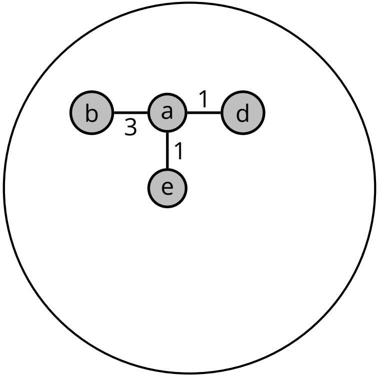


$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$

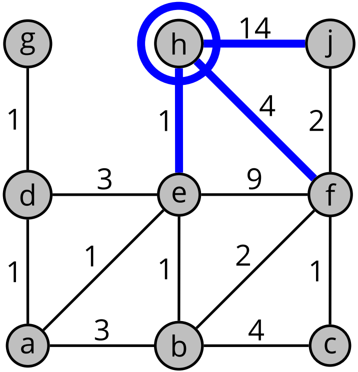
Link-State Advertisements

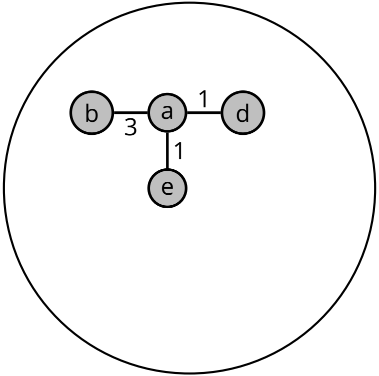


$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$



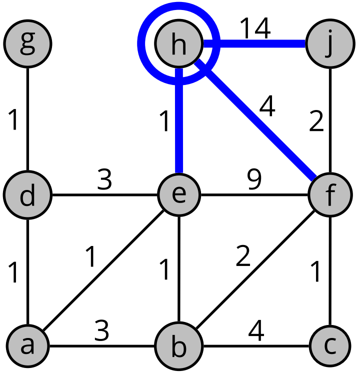
Link-State Advertisements

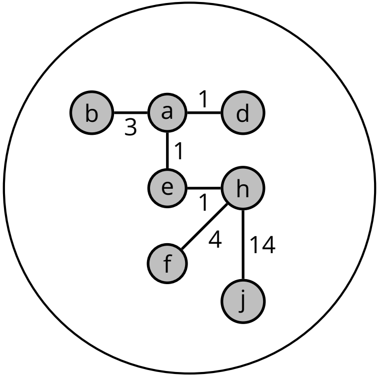




$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$

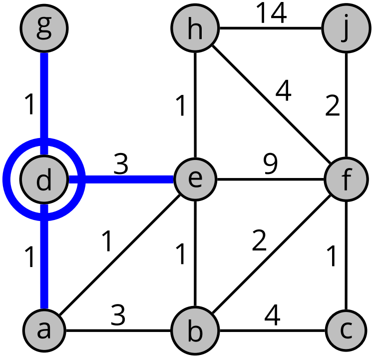
Link-State Advertisements

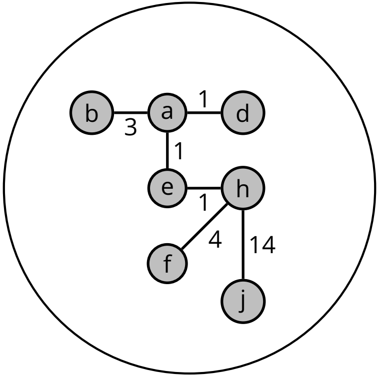




$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$

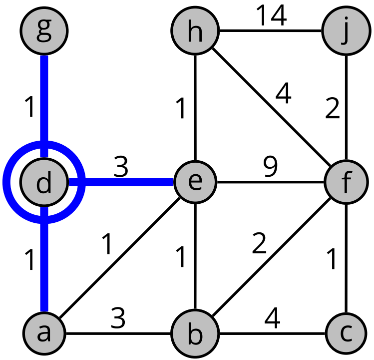
Link-State Advertisements

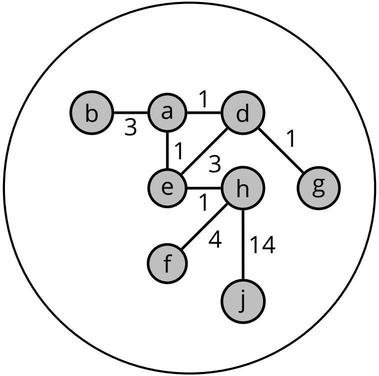




$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$
$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$

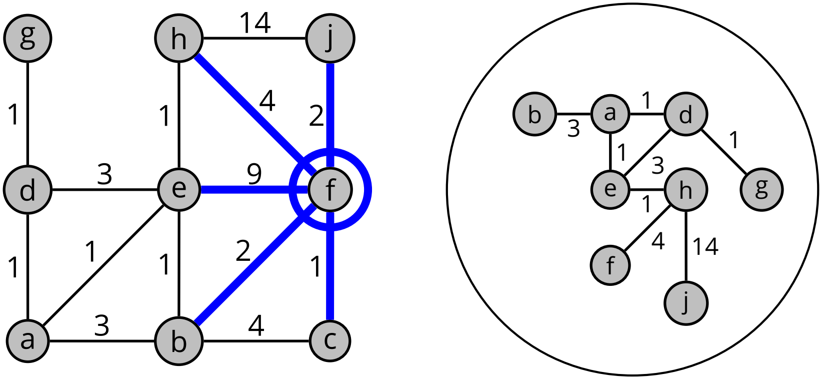
Link-State Advertisements





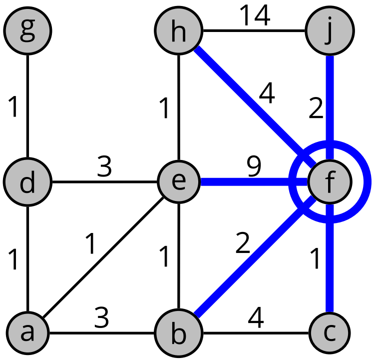
$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$
$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$

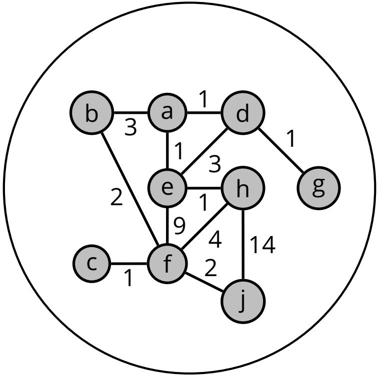
Link-State Advertisements



$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$
$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$
$$LSA_f = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}$$

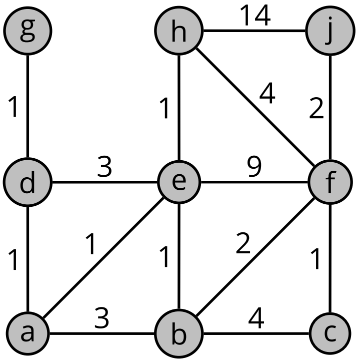
Link-State Advertisements

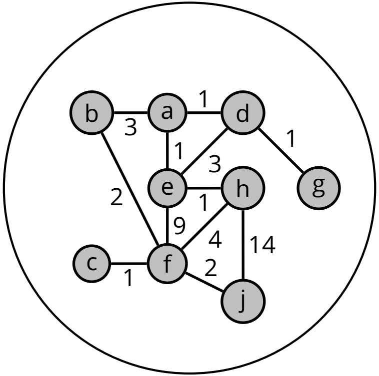




$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$
$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$
$$LSA_f = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}$$

Link-State Advertisements





$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$
$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$
$$LSA_f = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}$$
$$\ldots$$

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***

Broadcast Routing

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

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

- Simple and elegant

- ***Flooding***
  - 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

- ***Flooding***
  - 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?

- ***Flooding***
  - 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*

Broadcast Routing (2)

- ***Reverse-path broadcast***
  - 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*

Broadcast Routing (2)

- ***Reverse-path broadcast***
  - 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

Broadcast Routing (2)

- ***Reverse-path broadcast***
  - 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*

Broadcast Routing (2)

- ***Reverse-path broadcast***
  - 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?

Broadcast Routing (2)

- ***Reverse-path broadcast***
  - 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

Broadcast Routing (3)

- ***Sequence-number controlled flooding***

- ***Sequence-number controlled flooding***
  - the originator *s* of a broadcast packet marks the packet with a sequence number n_s

- ***Sequence-number controlled flooding***
  - 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_s

- ***Sequence-number controlled flooding***
  - 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_s
  - 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: $seq(p) > n_s$

- ***Sequence-number controlled flooding***
  - 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_s
  - 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

- ***Sequence-number controlled flooding***
  - 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_s
  - 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*

Dijkstra's Algorithm

- 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's Algorithm

```
DIJKSTRA(G = (V, E), u)
N ← {u}
for all v ∈ V
      do if v ∈ neighbors(u)
            then D[v] ← c(u, v)
                 p[v] ← u
            else  D[v] ← ∞
while N ≠ V
      do find w ∉ N such that D[w] is minimum
         N ← N ∪ {w}
         for all v ∈ neighbors(w) \ N
              do if D[w] + c(w, v) < D[v]
                    then D[v] ← D[w] + c(w, v)
                         p[v] ← w
```

Example

```
DIJKSTRA(G = (V, E), u)
N ← {u}
for all v ∈ V
    do if v ∈ neighbors(u)
        then D[v] ← c(u, v)
             p[v] ← u
        else  D[v] ← ∞
while N ≠ V
    do find w ∉ N such that D[w] is minimum
       N ← N ∪ {w}
       for all v ∈ neighbors(w) \ N
           do if D[w] + c(w, v) < D[v]
               then D[v] ← D[w] + c(w, v)
                    p[v] ← w
```