Simulazione di Protocollo di Routing: Distance Vector Routing

Alessandro Ambrogiani
alessandr.ambrogian4@studio.unibo.it
0001080493

November 21, 2024

1 Introduzione

Questa relazione descrive il progetto di simulazione del protocollo di routing Distance Vector Routing in Python. L'obiettivo del progetto è simulare il funzionamento del protocollo attraverso una rete di nodi che condividono informazioni di routing per determinare i percorsi più brevi verso tutti gli altri nodi della rete, includendo il nodo successivo (*next hop*) per ciascun percorso.

2 Obiettivi

L'obiettivo principale è implementare un programma Python che:

- Gestisca la costruzione di una rete di nodi.
- Implementi la logica di aggiornamento delle tabelle di routing per il protocollo Distance Vector.
- Mostri le tabelle di routing finali per ogni nodo, incluse le distanze e i *next hop*.

3 Struttura del Codice

Il programma è strutturato attorno alla classe `Nodo`, che rappresenta ogni nodo della rete. Ogni istanza della classe ha una tabella di routing, una lista di vicini e dei metodi per gestire gli aggiornamenti.

3.1 Classe Nodo

La classe `Nodo` contiene i seguenti attributi e metodi:

- `nome`: identificativo del nodo.

- `tavola_routing`: un dizionario che memorizza le rotte, le loro distanze e i *next hop*.
- `vicini`: un dizionario che memorizza le distanze verso i nodi vicini.

3.1.1 Metodo `__init__`

Il metodo `__init__` inizializza un nodo con un nome specifico, la propria tabella di routing e un dizionario dei vicini. Il nodo conosce inizialmente solo la distanza verso se stesso e imposta il *next hop* a se stesso.

```
class Nodo:
    def __init__(self, nome):
        self.nome = nome
        self.tavola_routing = {nome: (0, nome)}  #
            Distanza e next hop verso se stesso
        self.vicini = {}
```

3.1.2 Metodo `aggiungi_vicino`

Il metodo `aggiungi_vicino` consente di aggiungere un nodo vicino con una distanza specificata. Aggiorna la lista dei vicini e inizializza la tavola di routing del nodo corrente per includere il vicino, impostando il *next hop* al vicino stesso.

```
def aggiungi_vicino(self, vicino, distanza):
    self.vicini[vicino] = distanza
    self.tavola_routing[vicino.nome] = (distanza, vicino.nome)
```

3.1.3 Metodo `invia_aggiornamenti`

Il metodo `invia_aggiornamenti` permette al nodo di inviare la propria tabella di routing a tutti i vicini. Questo metodo simula la fase di invio degli aggiornamenti nel protocollo Distance Vector.

```
def invia_aggiornamenti(self):
    aggiornato = False
    for vicino in self.vicini:
        if vicino.ricevi_aggiornamento(self.nome, self.tavola_routing):
            aggiornato = True
    return aggiornato
```

3.1.4 Metodo `ricevi_aggiornamento`

Il metodo `ricevi_aggiornamento` riceve la tabella di routing di un nodo vicino e aggiorna la propria tabella di routing se trova un percorso più breve. Questo metodo aggiorna sia la distanza che il *next hop* per ciascuna destinazione.

```
def ricevi_aggiornamento(self, nodo_vicino, tavola_vicino
    ):
    aggiornato = False
    distanza_vicino = self.tavola_routing[nodo_vicino][0]

    for destinazione, (distanza, _) in tavola_vicino.
        items():
        nuova_distanza = distanza_vicino + distanza
        if (destinazione not in self.tavola_routing or
            nuova_distanza < self.tavola_routing[
                destinazione][0]):
            self.tavola_routing[destinazione] = (
                nuova_distanza, nodo_vicino)
            aggiornato = True

    return aggiornato
```

3.1.5 Metodo `stampa_tavola_routing`

Il metodo `stampa_tavola_routing` stampa la tabella di routing per ciascun nodo, includendo la distanza e il *next hop* per ogni destinazione.

```
def stampa_tavola_routing(self):
    print(f"Tavola di routing per {self.nome}:")
    for destinazione, (distanza, next_hop) in self.tavola_routing.items():
        print(f"  {destinazione}: {distanza}, Next hop: {next_hop}")
    print("\n")
```

4 Output del Programma

Alla fine della simulazione, viene stampata la tavola di routing per ciascun nodo, mostrando la distanza e il *next hop* per ogni destinazione. Di seguito un esempio di output per una rete con quattro nodi (`A`, `B`, `C`, `D`).

```
Tavola di routing per A:
  A: 0, Next hop: A
  B: 1, Next hop: B
  C: 3, Next hop: B
  D: 4, Next hop: B

Tavola di routing per B:
  B: 0, Next hop: B
  A: 1, Next hop: A
  C: 2, Next hop: C
  D: 3, Next hop: C
```

```
Tavola di routing per C:
  C: 0, Next hop: C
  A: 3, Next hop: B
  B: 2, Next hop: B
  D: 1, Next hop: D

Tavola di routing per D:
  D: 0, Next hop: D
  B: 3, Next hop: C
  C: 1, Next hop: C
  A: 4, Next hop: C
```

5 Conclusioni

Il programma dimostra con successo il funzionamento del protocollo Distance Vector Routing, esteso per includere il *next hop* nella tabella di routing. Questo permette a ciascun nodo di apprendere non solo le distanze minime, ma anche il percorso effettivo per raggiungere ogni destinazione.