Simulazione protocollo di Routing

Relazione per il progetto
del corso di
Reti di Telecomunicazione
A.A. 2024/25

Lorenzo Tordi 0001042969
Alex Frisoni 0001089191,
Lukasz Wojnicz 0001071295

25 novembre 2024

Indice

Capitolo 1

Introduzione

Il programma in Python permette di eseguire una simulazione di Protocollo di Routing.

Caratteristiche del codice:

- `Protocollo simulato`: Distance Vector Routing (simile al protocollo RIP).
- `Metodo di convergenza`: Iterazioni finché le tabelle non cambiano.
- `Efficienza`: Efficiente per reti piccole, ma potrebbe diventare lento per reti grandi.

Capitolo 2

Classe Node

Questa classe rappresenta un nodo nella rete, contenente informazioni sul proprio nome, sui vicini diretti e sulla tabella di routing.

Proprietà:

- `self.name`: il nome del nodo (es., "A", "B", ecc.).
- `self.routing_table`: la tabella di routing, un dizionario che associa ogni destinazione a una coppia *(Costo, Next Hop)*, dove:
  - `Costo`: il costo totale per raggiungere la destinazione.
  - `Next Hop`: il nodo successivo verso la destinazione.
- `self.neighbors`: un dizionario che memorizza i vicini diretti e i costi dei collegamenti.

Metodi:

- `__init__(self, name)`: inizializza un nodo con il nome dato, una tabella di routing vuota e una lista di vicini vuota.
- `update_table(self, neighbor_name, neighbor_table)`: aggiorna la tabella di routing del nodo in base alla tabella di routing di un nodo vicino.
  - **Parametri**:
    - `neighbor_name`: nome del vicino che invia la tabella.
    - `neighbor_table`: la tabella di routing del vicino.
  - **Logica**:
    Per ogni destinazione nella tabella del vicino:

    - Calcola il nuovo costo come somma del costo verso il vicino e del costo dal vicino alla destinazione.
    - Se la destinazione non è presente nella tabella locale o il nuovo costo è più basso del costo attuale:
        - Aggiorna la tabella con il nuovo costo e imposta il vicino come *next hop*.
        - Imposta `updated` a `True`.
    - Restituisce `True` se ci sono stati aggiornamenti; altrimenti `False`.

- `__str__(self)`: restituisce una rappresentazione leggibile della tabella di routing del nodo.

```python
class Node:
    def __init__(self, name):
        self.name = name
        self.routing_table = {}  # Destinazione: (Costo, Next Hop)
        self.neighbors = {}      # Vicini: Costo

    def update_table(self, neighbor_name, neighbor_table):
        updated = False
        for dest, (cost_to_dest, next_hop) in neighbor_table.items():
            new_cost = self.neighbors[neighbor_name] + cost_to_dest
            if dest not in self.routing_table or new_cost < self.routing_table[dest][0]:
                self.routing_table[dest] = (new_cost, neighbor_name)
                updated = True
        return updated

    def __str__(self):
        return f"Node {self.name} Routing Table: {self.routing_table}"
```

Figura 2.1: Classe Node

Capitolo 3

Classe Network

Questa classe gestisce la rete nel suo insieme, contenendo tutti i nodi e i collegamenti tra di essi.

Proprietà:

- `self.nodes`: un dizionario che associa i nomi dei nodi agli oggetti `Node`.

Metodi:

- `__init__(self)`: inizializza una rete vuota.
- `add_node(self, name)`: aggiunge un nuovo nodo con il nome specificato alla rete.
- `add_link(self, node1, node2, cost)`: crea un collegamento tra due nodi con un dato costo.
  - Aggiorna i vicini di entrambi i nodi.
- `simulate_routing(self)`: simula il protocollo di routing fino al raggiungimento della convergenza.
  - **Logica**:
    - Usa un ciclo per continuare finché non ci sono più aggiornamenti alle tabelle di routing.
    - A ogni iterazione:
      - Ogni nodo aggiorna la propria tabella in base alle tabelle dei vicini.
      - Se una tabella viene aggiornata, il ciclo continua; altrimenti, si interrompe.

* Alla fine di ogni iterazione, stampa le tabelle di routing di tutti i nodi.

```python
class Network:
    def __init__(self):
        self.nodes = {}

    def add_node(self, name):
        self.nodes[name] = Node(name)

    def add_link(self, node1, node2, cost):
        self.nodes[node1].neighbors[node2] = cost
        self.nodes[node2].neighbors[node1] = cost

    def simulate_routing(self):
        converged = False
        iteration = 0
        while not converged:
            print(f"Iteration {iteration}")
            converged = True
            for node_name, node in self.nodes.items():
                for neighbor_name in node.neighbors:
                    if node.update_table(neighbor_name, self.nodes[neighbor_name].routing_table):
                        converged = False
            for node in self.nodes.values():
                print(node)
            iteration += 1
            print("-" * 50)
```

Figura 3.1: Classe Network

Capitolo 4

Configurazione della rete

- Viene creata una rete con 4 nodi: ”A”, ”B”, ”C”, ”D”.
- Sono aggiunti i collegamenti con i rispettivi costi:
  - A <-> B: costo 1
  - A <-> C: costo 5
  - B <-> C: costo 2
  - B <-> D: costo 4
  - C <-> D: costo 1

Inizializzazione delle tabelle di routing:

- Ogni nodo inizializza la propria tabella di routing conoscendo i vicini diretti:
  - Per ogni vicino, imposta il costo e il *next hop* come il vicino stesso.
  - Aggiunge sé stesso con costo 0 e *next hop* sé stesso.

```
# Definizione della rete
network = Network()
network.add_node("A")
network.add_node("B")
network.add_node("C")
network.add_node("D")

network.add_link("A", "B", 1)
network.add_link("A", "C", 5)
network.add_link("B", "C", 2)
network.add_link("B", "D", 4)
network.add_link("C", "D", 1)

# Inizializza le tabelle di routing
for node in network.nodes.values():
    for neighbor, cost in node.neighbors.items():
        node.routing_table[neighbor] = (cost, neighbor)
    node.routing_table[node.name] = (0, node.name)
```

Figura 4.1: Configurazione della rete

Capitolo 5

Simulazione del protocollo

La simulazione segue il ciclo iterativo:

1. Ogni nodo aggiorna la propria tabella in base alle informazioni ricevute dai vicini.

2. Se un nodo trova un percorso più economico, aggiorna la propria tabella.

3. Quando non ci sono più aggiornamenti, il sistema converge e le tabelle diventano stabili.

Risultato
Alla fine della simulazione, ogni nodo ha una tabella di routing che rappresenta i percorsi più economici per raggiungere qualsiasi altro nodo nella rete. Le informazioni includono:

- Il costo totale del percorso.

- Il prossimo nodo da attraversare (*next hop*) lungo il percorso.

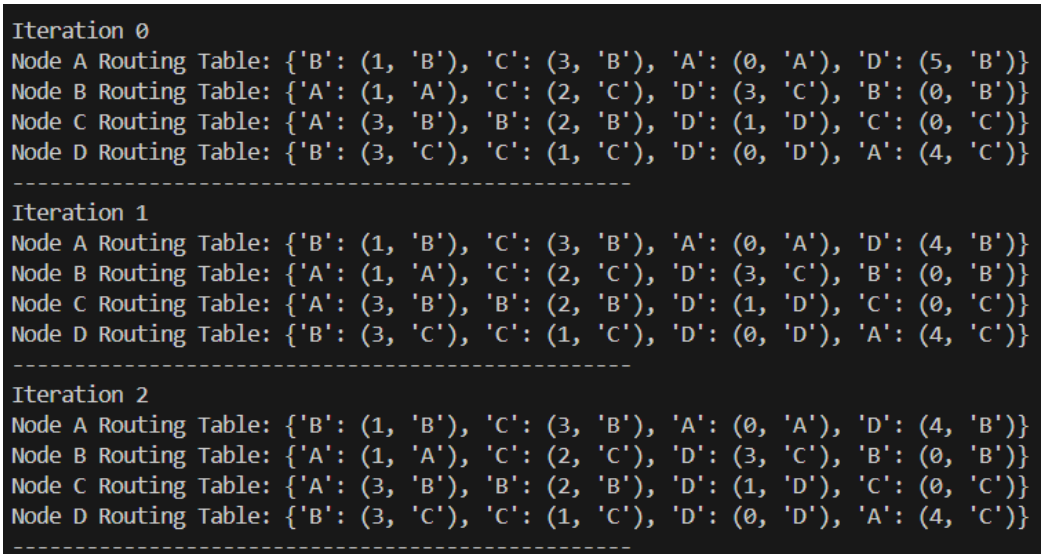

Figura 5.1: Risultato