SIMULAZIONE DI PROTOCOLLO DI ROUTING

IRENE SOFIA LOTTI

Numero di matricola: 0001071300

INTRODUZIONE INIZIALE

Il progetto simula il funzionamento di un semplice protocollo di Routing, basato sull'algoritmo Distance Vector. Questo è un protocollo che viene usato nei sistemi di rete per determinare le rotte più efficienti tra i nodi, minimizzando i costi complessivi.

L'obiettivo del progetto è implementare una rete simulata, composta da 6 nodi connessi tramite collegamenti definiti da costi specifici.

```
#Creazione nodi
for node_name in ["A", "B", "C", "D", "E", "F"]:
    network.add_node(node_name)

#Creazione connessioni (link) e relativi costi
network.add_link("A", "B", 1)
network.add_link("A", "C", 4)
network.add_link("B", "C", 2)
network.add_link("B", "D", 6)
network.add_link("C", "D", 3)
network.add_link("A", "F", 4)
network.add_link("E", "F", 2)
network.add_link("D", "E", 8)
```

Ogni nodo ha una tabella di instradamento che rappresenta la conoscenza attuale delle rotte verso gli altri nodi. Vengono effettuati dei cicli iterativi di aggiornamento, e grazie a questo ogni nodo converge mantenendo costo ottimale.

I risultati vengono sia stampati a video, sia salvati in un file ("risultati.txt"). Le stampe riguardano le tabelle di routing iniziali e finali di tutti i nodi

DISTANCE VECTOR

Tale algoritmo permette a ogni nodo di calcolare i percorsi più brevi verso gli altri nodi della rete, basandosi sulle informazioni ricevute dai nodi vicini. L'algoritmo permette uno scambio iterativo delle tabelle di instradamento tra i nodi. Le rotte vengono aggiornate fino a che non viene raggiunto lo stato di convergenza.

1. Ogni nodo ha una propria tabella di instradamento. Le varie righe della tabella mostrano qual è il minor costo (e quindi il percorso più breve) verso gli altri nodi della rete
2. Ogni nodo invia periodicamente la propria tabella di Routing ai vicini
3. I nodi che ricevono le tabelle aggiornano i loro percorsi. Il percorso che scelgono è sempre quello con minor costo
4. L'algoritmo si ripete sino a che nessuna tabella subisce delle modifiche. Questo vuol dire che il sistema ha raggiunto la convergenza

Il codice può essere considerato un'applicazione del principio di Bellman-Ford, applicato tramite una variante semplificata.

CODICE

La struttura del codice è a oggetti. Abbiamo, in particolare, due classi:

- Network: rappresenta la rete
- Node: rappresenta ogni nodo

NODE

```
class Node:
    def __init__(self, name):
        self.name = name          # Nome del nodo
        self.routing_table = {name: 0}  # Tabella instradamento per ogni nodo. {nome-nodo
        self.neighbors = {}       # Distanze dirette verso i vicini


    """Confronta le informazioni nella tabella di routing attuale con quelle
    ricevute dai nodi vicini. Se trova un percorso più beve, aggiorna la tabella.
    Return: true se c'è stato un aggiornamento, false altrimenti """

    def update_routing_table(self):
        updated = False
        for neighbor, cost in self.neighbors.items():
            for dest, actual_cost in neighbor.routing_table.items():
                if dest == self.name:  #il costo verso sé stessi rimane 0
                    continue
                new_cost = cost + actual_cost
                if dest not in self.routing_table or new_cost < self.routing_table[dest]:
                    self.routing_table[dest] = new_cost
                    updated = True
        return updated
```

La tabella di instradamento è implementata come un dizionario. Le chiavi sono i nomi dei nodi, mentre i valori rappresentano il costo per raggiungerli. I vicini diretti sono memorizzati in un altro dizionario, che associa i nodi vicini ai costi dei collegamenti

update_routing_table -> metodo che implementa la logica di aggiornamento. Nella pratica, per ogni vicino si confrontano i costi riportati nella sua tabella con quelli giù noti, aggiornando la tabella locale se si trova in un percorso più breve

NETWORK

```
class Network:
    def __init__(self):
        self.nodes = {}
        self.output_lines = []

    """crea un nodo con il nome name e lo aggiunge alla rete """
    def add_node(self, name):
        self.nodes[name] = Node(name)
```

add_node -> aggiunge nuovi nodi alla rete

add_link -> collega due nodi. In particolare, viene specificato il costo del collegamento e vengono inizializzate le tabelle di routing con i costi diretti

simulate -> esegue l'algoritmo Distance Vector. Il ciclo viene ripetuto fino al raggiungimento della convergenza.