Capuano Maurizio **matr. 0001071194** **Reti di Telecomunicazione**

Simulazione del Protocollo di Routing (Distance Vector)

Introduzione

Il protocollo **Distance Vector** (DV) è uno dei principali algoritmi di routing dinamico utilizzato nelle reti di computer. In questo progetto, abbiamo sviluppato una simulazione in Python che riproduce il funzionamento di questo protocollo. Lo scopo principale dell'elaborato è implementare un sistema che simuli il comportamento di una rete composta da nodi che si scambiano informazioni sulle distanze tra di loro per determinare le rotte più brevi.

Obiettivi

Gli obiettivi principali del progetto erano:

1. **Implementazione della logica di aggiornamento delle rotte:** ogni nodo della rete deve aggiornare periodicamente la sua tabella di routing in base alle informazioni ricevute dai nodi vicini.
2. **Gestione delle tabelle di routing:** ogni nodo mantiene una tabella di routing che contiene la distanza minima per raggiungere tutti gli altri nodi.
3. **Calcolo delle distanze tra i nodi:** ogni nodo deve calcolare il percorso più breve per raggiungere gli altri nodi della rete.

Descrizione del Protocollo di Routing Distance Vector

Il protocollo **Distance Vector** è un tipo di protocollo di routing che utilizza un approccio di tipo "distance vector" per determinare la rotta più breve verso una destinazione. Ogni nodo nella rete mantiene una **tabella di routing** che contiene informazioni sulla distanza (o il costo) per raggiungere ogni altro nodo della rete.

Il principio di funzionamento si basa su questi passaggi:

1. **Inizializzazione:** all'inizio, ogni nodo conosce solo la distanza verso i propri nodi vicini. La distanza verso sé stesso è zero, e la distanza verso un nodo non vicini è impostata su infinito.
2. **Aggiornamenti periodici:** ogni nodo invia la sua tabella di routing ai suoi vicini, e aggiorna la propria tabella di routing confrontando le distanze ricevute con le distanze locali.
3. **Convergenza:** il processo continua fino a quando tutte le tabelle di routing non cambiano più, indicando che la rete ha raggiunto uno stato stabile e le rotte più brevi sono state calcolate.

Struttura del Codice

Il codice Python si compone principalmente di due componenti:

1. **La classe** **Node** che rappresenta un nodo della rete e gestisce la sua tabella di routing.

2. **La funzione distance_vector_routing** che simula l'aggiornamento delle tabelle di routing e la convergenza dei nodi.

La Classe Node

Ogni nodo è definito come una classe con i seguenti attributi principali:

- node_id: l'identificativo univoco del nodo.
- neighbors: un dizionario che contiene i vicini diretti e il costo per raggiungerli.
- routing_table: la tabella di routing che memorizza la distanza per ogni nodo della rete.

La classe ha due metodi principali:

- **update_routing_table**: questo metodo aggiorna la tabella di routing di un nodo sulla base delle informazioni ricevute da un altro nodo. Se viene trovata una distanza più breve verso una destinazione, la tabella viene aggiornata. La variabile converged è utilizzata per verificare se la rete ha raggiunto la convergenza, ovvero quando nessuna tabella di routing viene aggiornata. Questo meccanismo permette di continuare gli aggiornamenti fino a che tutti i nodi non hanno trovato la rotta più breve verso ogni destinazione.
- **print_routing_table**: questo metodo stampa la tabella di routing del nodo.

Funzione di Routing distance_vector_routing

Questa funzione gestisce l'aggiornamento delle tabelle di routing in tutta la rete. In ogni ciclo:

- Ogni nodo invia la sua tabella di routing ai suoi vicini.
- Ogni nodo aggiorna la sua tabella in base alle informazioni ricevute.
- La funzione continua fino a quando tutte le tabelle di routing sono stabili (nessuna distanza cambia più).

Funzionamento Dettagliato

La rete di nodi viene simulata creando una serie di oggetti Node, ognuno dei quali ha un dizionario di vicini con i costi per raggiungerli. Ogni nodo invia la sua tabella di routing ai vicini, e i vicini aggiornano le proprie tabelle con i dati ricevuti.

Si prende ad esempio per testing una rete composta da 4 nodi:

- Nodo A è collegato ai nodi B (con costo 1) e C (con costo 4).
- Nodo B è collegato ai nodi A (costo 1), C (costo 2) e D (costo 5).
- Nodo C è collegato ai nodi A (costo 4), B (costo 2) e D (costo 1).
- Nodo D è collegato ai nodi B (costo 5) e C (costo 1).

La simulazione inizia con ciascun nodo che ha una visione parziale della rete e, durante gli aggiornamenti successivi, convergerà verso una tabella di routing ottimale con le rotte più brevi per ogni nodo.

Convergenza e Risultato Finale

Il processo di aggiornamento continua fino a quando la rete non converge, ossia quando nessuna tabella di routing viene aggiornata durante una iterazione. Questo significa che tutti i nodi hanno trovato le rotte più brevi per raggiungere gli altri nodi.

Output delle Tabelle di Routing

Alla fine dell'algoritmo, ciascun nodo avrà la sua tabella di routing, che potrebbe essere simile al seguente:

- **Nodo A**:

  Routing Table for Node A:
  Destination A -> Distance 0
  Destination B -> Distance 1
  Destination C -> Distance 3
  Destination D -> Distance 6

- **Nodo B**:

  Routing Table for Node B:
  Destination B -> Distance 0
  Destination A -> Distance 1
  Destination C -> Distance 2
  Destination D -> Distance 5

- **Nodo C**:

  Routing Table for Node C:
  Destination C -> Distance 0
  Destination A -> Distance 3
  Destination B -> Distance 2
  Destination D -> Distance 1

- **Nodo D**:

  Routing Table for Node D:
  Destination D -> Distance 0
  Destination B -> Distance 5
  Destination C -> Distance 1
  Destination A -> Distance 6

Problemi Riscontrati e Correzioni

Durante i test iniziali, alcuni risultati non corrispondevano alle aspettative. I problemi principali erano:

1. **Aggiornamenti non corretti della tabella**: la distanza tra un nodo e una destinazione non veniva aggiornata correttamente se il percorso attraverso un vicino risultava più breve.
2. **Condizione di convergenza non gestita correttamente**: il processo di aggiornamento non rilevava sempre correttamente quando la rete aveva converso.

Per risolvere questi problemi, è stato modificato il metodo `update_routing_table` in modo che considerasse correttamente le distanze dai vicini e si aggiornasse solo quando veniva trovata una rotta più breve. Inoltre, è stato introdotto un controllo esplicito per determinare la convergenza della rete.

Conclusioni

Il progetto ha permesso di implementare e simulare il protocollo **Distance Vector** in Python. Dopo aver corretto i problemi iniziali, la simulazione è riuscita a produrre risultati coerenti e accurati, dimostrando come i nodi di una rete possano convergere verso le rotte ottimali per raggiungere altre destinazioni.