Distance Vector Routing Simulator

Progetto Laboratorio Reti di Telecomunicazioni

Azael Garcia Rufer 0001081559 - Gaia Pojaghi 0001071280

Introduzione

Il progetto si propone di sviluppare una simulazione del protocollo Distance Vector Routing utilizzando Python, implementando le tabelle di routing, il calcolo delle rotte ottimali e il processo iterativo di aggiornamento delle tabelle fino al raggiungimento della convergenza. Attraverso l'analisi di diverse tipologie di reti (lineare, a stella, circolare, completamente connessa, disconnessa, ecc.), la simulazione ha permesso di valutare il comportamento del protocollo in molteplici scenari differenti, verificando la sua capacità di calcolare percorsi ottimali in modo corretto ed efficiente.

Descrizione del Protocollo Distance Vector

Il protocollo Distance Vector è un metodo distribuito per il calcolo delle rotte più brevi. Ogni nodo nella rete mantiene una tabella di routing che indica la distanza minima da tutti gli altri nodi conosciuti e il next hop. Queste distanze vengono aggiornate scambiando informazioni con i nodi vicini e ricalcolando i percorsi in base ai costi ricevuti.

Passaggi principali:

1. Ogni nodo inizializza la propria tabella di routing, conoscendo solo le distanze verso i vicini diretti. La distanza verso il nodo stesso viene impostata a zero.
2. Periodicamente, ogni nodo scambia la propria tabella di routing con i vicini.
3. Dopo aver ricevuto le tabelle dai vicini, il nodo aggiorna la propria tabella utilizzando la seguente regola:
   Costo(nodo,destinazione)=min(Costo(nodo,vicino)+Costo(vicino,destinazione))
4. Il processo continua fino a quando le tabelle di routing di tutti i nodi convergono (cioè non si verificano più aggiornamenti).

Implementazione

L'implementazione del progetto in Python segue questi passaggi:

a) Definizione della rete: la rete è rappresentata come un dizionario in cui ogni nodo è una chiave e i suoi vicini sono elencati con i relativi costi.

```
network = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 6},
    'C': {'A': 4, 'B': 2, 'D': 3},
    'D': {'B': 6, 'C': 3}
}
```

b) Inizializzazione delle tabelle di routing: ogni nodo conosce inizialmente solo la distanza verso sé stesso (0) e verso i vicini diretti (costo noto). Per tutti gli altri nodi, la distanza iniziale è impostata a infinito (float('inf')). Il next di un nodo verso i vicini diretti viene inizialmente inizializzato a None, mentre il next hop del nodo verso se stesso è il nodo stesso.

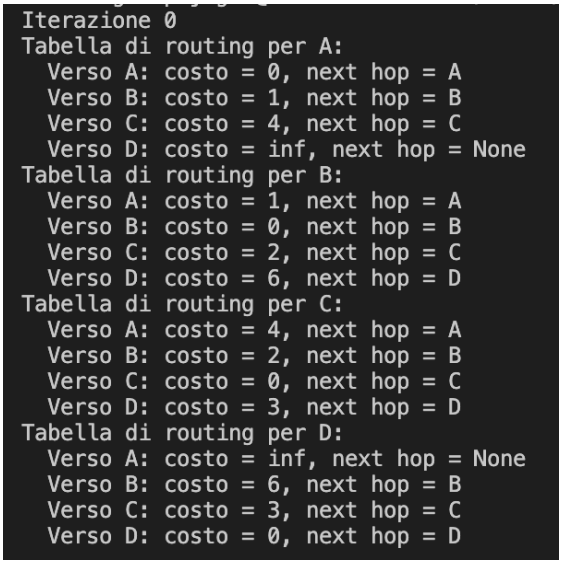
c) Aggiornamento delle tabelle: le tabelle vengono aggiornate iterativamente utilizzando le informazioni ricevute dai vicini, seguendo la logica del protocollo Distance Vector. Ogni nodo verifica se può migliorare a distanza verso una destinazione passando attraverso uno dei suoi vicini e quindi se trova un percorso più corto aggiorna la sua tabella.

d) Simulazione: il programma esegue ciclicamente iterazioni stampando le relative tabelle di routing fino alla convergenza, ovvero quando le tabelle non subiscono più aggiornamenti. A questo punto la tabella di routing finale viene visualizzata a schermo con un opportuno messaggio che indica il numero di iterazioni svolte.

Output e analisi

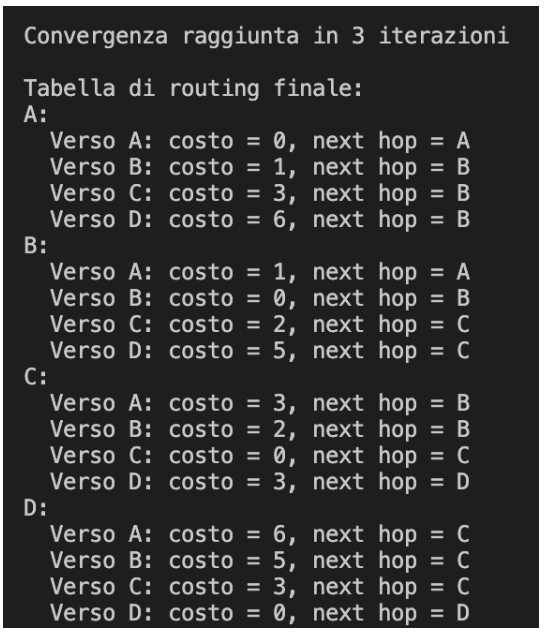
Durante la simulazione il programma stampa le tabelle di routing aggiornate a ogni iterazione. Al termine della simulazione, vengono mostrate le tabelle finali.

Esempio di output visibile a ogni iterazione:



```
Iterazione 0
Tabella di routing per A:
  Verso A: costo = 0, next hop = A
  Verso B: costo = 1, next hop = B
  Verso C: costo = 4, next hop = C
  Verso D: costo = inf, next hop = None
Tabella di routing per B:
  Verso A: costo = 1, next hop = A
  Verso B: costo = 0, next hop = B
  Verso C: costo = 2, next hop = C
  Verso D: costo = 6, next hop = D
Tabella di routing per C:
  Verso A: costo = 4, next hop = A
  Verso B: costo = 2, next hop = B
  Verso C: costo = 0, next hop = C
  Verso D: costo = 3, next hop = D
Tabella di routing per D:
  Verso A: costo = inf, next hop = None
  Verso B: costo = 6, next hop = B
  Verso C: costo = 3, next hop = C
  Verso D: costo = 0, next hop = D
```

Dopo un tot di iterazioni, le tabelle convergono: si visualizza la scritta “Convergenza raggiunta in n iterazioni” e vengono mostrate le tabelle di routing finali.



```
Convergenza raggiunta in 3 iterazioni

Tabella di routing finale:
A:
  Verso A: costo = 0, next hop = A
  Verso B: costo = 1, next hop = B
  Verso C: costo = 3, next hop = B
  Verso D: costo = 6, next hop = B
B:
  Verso A: costo = 1, next hop = A
  Verso B: costo = 0, next hop = B
  Verso C: costo = 2, next hop = C
  Verso D: costo = 5, next hop = C
C:
  Verso A: costo = 3, next hop = B
  Verso B: costo = 2, next hop = B
  Verso C: costo = 0, next hop = C
  Verso D: costo = 3, next hop = D
D:
  Verso A: costo = 6, next hop = C
  Verso B: costo = 5, next hop = C
  Verso C: costo = 3, next hop = C
  Verso D: costo = 0, next hop = D
```

Conclusioni

La simulazione del protocollo Distance Vector Routing ha dimostrato la capacità del modello di calcolare percorsi ottimali in reti di diverse topologie e configurazioni. Attraverso il processo iterativo di aggiornamento delle tabelle di routing, lo script ha raggiunto la convergenza in tutti i test, confermando l'efficacia dell'algoritmo Distance Vector nell'individuare i percorsi più brevi tra i nodi di una rete.

Durante lo sviluppo e l'esecuzione della simulazione, è stato possibile osservare il ruolo critico di parametri come la topologia della rete e i costi dei collegamenti. Ad esempio, in reti ad albero o con cicli e non solo, l'algoritmo ha comunque raggiunto la convergenza correttamente, dimostrando la sua robustezza.

In futuro potrebbe essere interessante estendere questa simulazione includendo scenari più complessi, come l'introduzione di guasti ai nodi.