Simulazione di Protocollo di Routing

Morri Lorenzo

10 dicembre 2024

Indice

1 Obiettivo

In questo progetto ho implementato una simulazione del protocollo Distance Vector Routing. Ogni nodo della rete scambia informazioni con i propri vicini e aggiorna iterativamente la propria tabella di routing per calcolare i costi minimi verso le altre destinazioni.

Nel codice ho utilizzato due classi principali: **Nodo**, che rappresenta un dispositivo della rete, e **Rete**, che gestisce l'insieme dei nodi e delle loro connessioni. Ogni nodo mantiene una tabella di routing con costi e prossimo hop per ogni destinazione, mentre la rete collega i nodi, simulare gli aggiornamenti e verificare che sia stabile.

2 Funzionamento

Ogni nodo inizia con una tabella di routing base, dove conosce solo se stesso. Quando si collega a un altro nodo, aggiorna la tabella. Durante la simulazione, ogni nodo scambia informazioni con i propri vicini e aggiorna la tabella calcolando i percorsi minimi. Questo processo continua fino a quando le tabelle non vengono più modificate.

Ad esempio, se nella rete ci sono i nodi A, B, C e D con collegamenti e costi definiti, al termine della simulazione ogni nodo avrà il percorso migliore per raggiungere qualsiasi destinazione, indicando costo che prossimo hop da seguire.

3 Problemi e soluzioni

Durante lo sviluppo del codice sono emersi alcuni problemi:

- **Esistenza dei nodi**: il metodo per collegare i nodi non verificava se i nodi specificati esistessero già e questo poteva portare a errori durante l'esecuzione. Ho risolto introducendo un controllo che solleva un'eccezione se uno dei nodi non è presente nella rete.
- **Propagazione inefficiente**: durante la simulazione, alcuni nodi continuavano a propagare dati già aggiornati, allungando il processo. Per risolvere, ho aggiunto un modo che interrompe la simulazione non appena le tabelle di routing non vengono modificate ulteriormente.

4 Conclusioni

Al termine della simulazione, il programma stampa le tabelle di routing finali per ciascun nodo. Ogni tabella mostra il costo minimo per raggiungere ogni destinazione e il prossimo hop da seguire. I risultati confermano che l'algoritmo lavora correttamente.

5 Codice

```
# Simulazione del protocollo di Routing

class Nodo:
    def __init__(self, nome):
        # Inizializza un nodo con il proprio nome e una tabella di routing
        # La tabella di routing parte con un unico elemento: se stesso con costo 0
        self.nome = nome
        self.tabella_di_routing = {nome: (0, nome)}  # {destinazione: (costo, prossimo_hop)}

    def aggiorna_tabella_di_routing(self, vicino, tabella_vicino):
        # Aggiorna la tabella di routing in base ai dati ricevuti dal nodo vicino
        aggiornato = False
        for destinazione, (costo_destinazione, prossimo_hop) in tabella_vicino.items():
            # Calcola il nuovo costo passando attraverso il nodo vicino
            nuovo_costo = vicino.tabella_di_routing[self.nome][0] + costo_destinazione
            # Aggiorna solo se la destinazione non e' presente o il nuovo costo e' inferiore
            if destinazione not in self.tabella_di_routing or nuovo_costo < self.tabella_di_routing[destinazione][0]:
                self.tabella_di_routing[destinazione] = (nuovo_costo, vicino.nome)
                aggiornato = True
        return aggiornato

    def stampa_tabella_di_routing(self):
        # Stampa in modo leggibile la tabella di routing del nodo
        print(f"Tabella di routing per {self.nome}:")
```

```
25          print("Destinazione\tCosto\tProssimo Hop")
26          for destinazione, (costo, prossimo_hop) in sorted(
                self.tabella_di_routing.items()):
27              print(f"{destinazione}\t\t{costo}\t{prossimo_hop}
                    ")
28          print()
29
30  class Rete:
31      def __init__(self):
32          # Inizializza una rete vuota
33          self.nodi = {}
34
35      def aggiungi_nodo(self, nome_nodo):
36          # Aggiunge un nuovo nodo alla rete
37          self.nodi[nome_nodo] = Nodo(nome_nodo)
38
39      def collega_nodi(self, nome_nodo1, nome_nodo2, costo):
40          # Collega due nodi con un costo specificato
41          if nome_nodo1 not in self.nodi or nome_nodo2 not in
                self.nodi:
42              raise ValueError("Entrambi i nodi devono esistere
                    nella rete.")
43
44          nodo1, nodo2 = self.nodi[nome_nodo1], self.nodi[
                nome_nodo2]
45          # Aggiorna le tabelle di routing per riflettere la
                connessione diretta
46          nodo1.tabella_di_routing[nome_nodo2] = (costo,
                nome_nodo2)
47          nodo2.tabella_di_routing[nome_nodo1] = (costo,
                nome_nodo1)
48
49      def simula_aggiornamenti_routing(self):
50          # Simula iterativamente gli aggiornamenti delle
                tabelle di routing fino a stabilita'
51          aggiornato = True
52          while aggiornato:
53              aggiornato = False
54              for nodo in self.nodi.values():
55                  for nome_vicino, (costo, _) in list(nodo.
                        tabella_di_routing.items()):  # Itera su
                        una copia
56                      if nome_vicino != nodo.nome:  # Evita di
                            aggiornare verso se stesso
57                          vicino = self.nodi[nome_vicino]
58                          # Propaga le informazioni di routing
                                dai vicini
59                          if nodo.aggiorna_tabella_di_routing(
                                vicino, vicino.tabella_di_routing)
```

```
                            :
                             aggiornato = True

    def stampa_tutte_tabelle_di_routing(self):
        # Stampa le tabelle di routing di tutti i nodi della
            rete
        for nodo in self.nodi.values():
            nodo.stampa_tabella_di_routing()

# Esempio di utilizzo:
if __name__ == "__main__":
    rete = Rete()

    # Aggiungi nodi alla rete
    for nome_nodo in ["A", "B", "C", "D"]:
        rete.aggiungi_nodo(nome_nodo)

    # Collega i nodi con i relativi costi
    rete.collega_nodi("A", "B", 1)
    rete.collega_nodi("A", "C", 4)
    rete.collega_nodi("B", "C", 2)
    rete.collega_nodi("C", "D", 1)

    # Simula gli aggiornamenti delle tabelle di routing
    rete.simula_aggiornamenti_routing()

    # Stampa le tabelle di routing finali
    rete.stampa_tutte_tabelle_di_routing()
```