Documentazione per monitor_connection.py

Enrico Cornacchia
Matricola: 0001070824

8 dicembre 2024

1 Introduzione

Questo documento descrive il funzionamento di uno script Python che simula un protocollo di routing semplice, come il Distance Vector Routing. Lo script implementa la logica di aggiornamento delle rotte, gestione delle tabelle di routing e calcolo delle distanze tra nodi.

2 Descrizione del Codice

Il codice è strutturato in una classe `Node` che rappresenta un nodo nella rete. Ogni nodo ha una tabella di routing e una lista di vicini. I metodi principali della classe sono:

- `add_neighbor`: Aggiunge un vicino alla lista dei vicini del nodo e aggiorna la tabella di routing.
- `update_routing_table`: Aggiorna la tabella di routing del nodo in base alle informazioni ricevute dai vicini.
- `print_routing_table`: Stampa la tabella di routing del nodo.

2.1 Dettagli del Codice

- `Node.__init__(self, name)`: Questo metodo è il costruttore della classe `Node`. Inizializza un nodo con un nome, una tabella di routing che contiene la distanza a sé stesso impostata a 0, e una lista di vicini vuota.
- `Node.add_neighbor(self, neighbor, distance)`: Questo metodo aggiunge un vicino alla lista dei vicini del nodo e aggiorna la tabella di routing con la distanza al vicino. Se la distanza è negativa, viene sollevata un'eccezione `ValueError`.
- `Node.update_routing_table(self)`: Questo metodo aggiorna la tabella di routing del nodo in base alle informazioni ricevute dai vicini. Per ogni

vicino, itera sulle destinazioni nella tabella di routing del vicino e calcola una nuova distanza. Se la nuova distanza è minore di quella attualmente registrata, aggiorna la tabella di routing. Restituisce `True` se la tabella di routing è stata aggiornata, altrimenti `False`. Implementa la tecnica dello "split horizon semplice" per prevenire il problema del "count to infinity".

- `Node.print_routing_table(self)`: Questo metodo stampa la tabella di routing del nodo, mostrando le destinazioni e le distanze.

2.2 Esempio di Utilizzo

- `Creazione dei Nodi`: Vengono creati sei nodi: `node_A`, `node_B`, `node_C`, `node_D`, `node_E`, e `node_F`.
- `Aggiunta dei Vicini`: Ogni nodo aggiunge i suoi vicini con le rispettive distanze. Ad esempio, `node_A.add_neighbor(node_B, 1)` aggiunge `node_B` come vicino di `node_A` con una distanza di 1.
- `Aggiornamento delle Tabelle di Routing`: Le tabelle di routing vengono aggiornate iterativamente. In ogni iterazione, ogni nodo aggiorna la sua tabella di routing in base alle informazioni ricevute dai suoi vicini. Il ciclo continua fino a quando le tabelle di routing non convergono (cioè, non ci sono più cambiamenti) o viene raggiunto un limite massimo di iterazioni per evitare loop infiniti.
- `Logging`: Durante ogni iterazione, viene registrato un messaggio di log per monitorare il processo di aggiornamento e verificare la convergenza. Se viene raggiunto il limite massimo di iterazioni, viene registrato un avviso per indicare un possibile ciclo nella rete.
- `Stampa delle Tabelle di Routing`: Alla fine del processo di aggiornamento, le tabelle di routing finali per ogni nodo vengono stampate.

3 Output delle Tabelle di Routing

Di seguito sono riportate due immagini che mostrano l'output delle tabelle di routing.

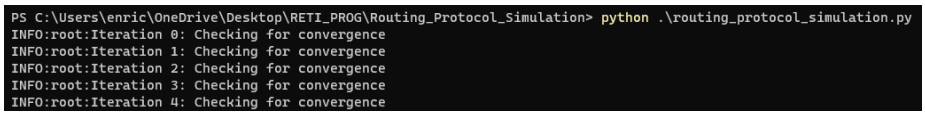

Figura 1: Iterazioni per arrivare a convergere

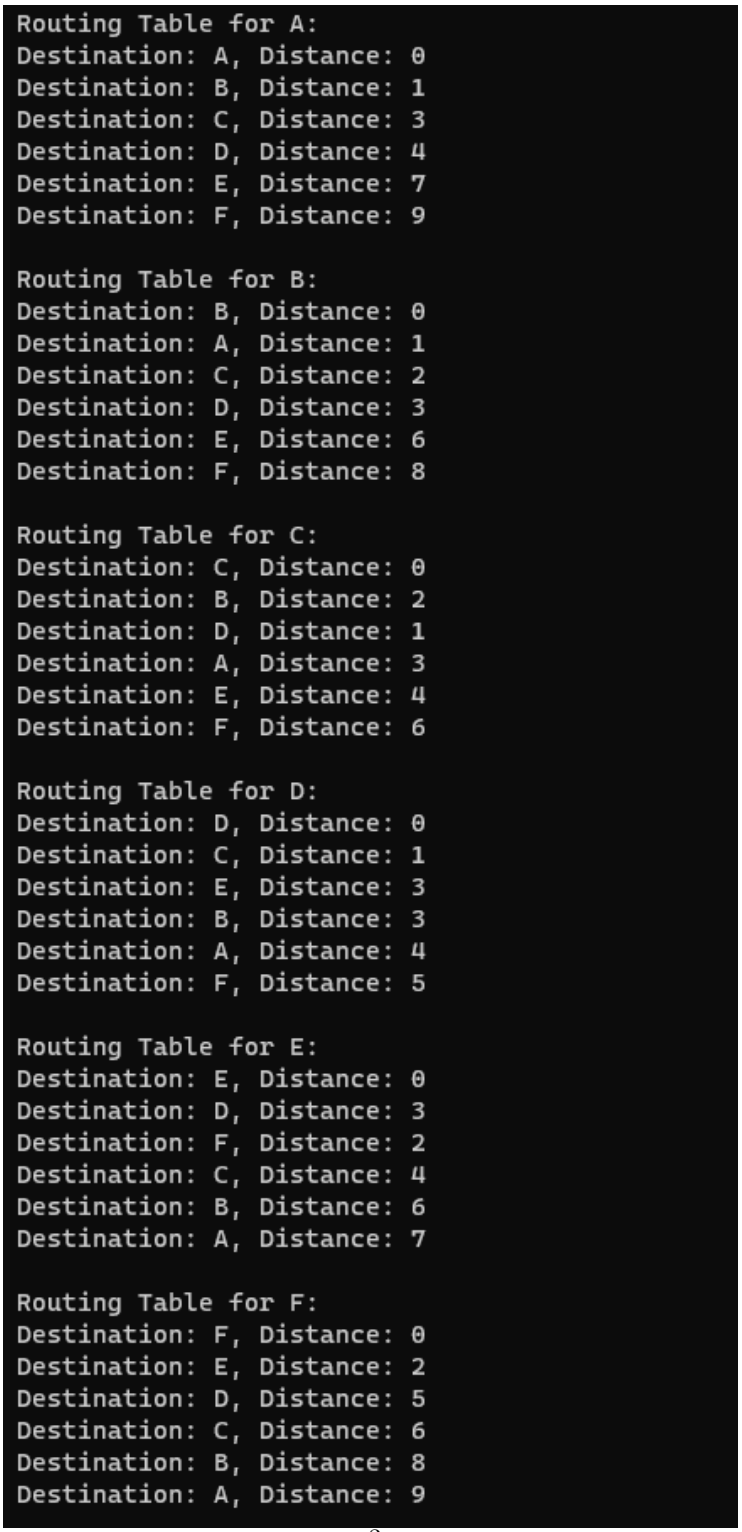

Figura 2: Tabelle di routing

4 Problematiche generali e legate al Distance Vector Routing

Durante lo sviluppo dello script di simulazione del protocollo di routing, sono emersi diversi problemi che sono stati affrontati e risolti. Di seguito è riportata una documentazione dettagliata dei problemi incontrati, i tentativi di risoluzione e il ragionamento dietro le scelte effettuate.

4.1 Problema 1: Convergenza delle Tabelle di Routing

Problematica: Le tabelle di routing non convergevano correttamente, causando un ciclo infinito.

Soluzione:

- *Iterazioni Fisse*: Inizialmente, il codice eseguiva un numero fisso di iterazioni per aggiornare le tabelle di routing. Tuttavia, questo approccio non garantiva la convergenza.
- *Condizione di Convergenza Dinamica*: È stato implementato un controllo dinamico per verificare se le tabelle di routing convergevano. Il ciclo continua fino a quando non ci sono più cambiamenti nelle tabelle di routing.

Ragionamento: Utilizzare una condizione di convergenza dinamica è più efficiente e garantisce che il processo termini solo quando tutte le tabelle di routing sono stabili.

4.2 Problema 2: Logging e Monitoraggio

Problematica: Mancanza di visibilità sul processo di aggiornamento delle tabelle di routing.

Soluzione:

- *Aggiunta di Logging*: È stato aggiunto il modulo `logging` per tracciare le iterazioni e monitorare il processo di aggiornamento delle tabelle di routing.

Ragionamento: Il logging fornisce una visibilità chiara sul flusso di esecuzione e aiuta a identificare eventuali problemi durante il processo di aggiornamento.

4.3 Problema 3: Gestione degli Errori

Problematica: Mancanza di gestione degli errori per input non validi o situazioni anomale.

Soluzione:

- *Validazione degli Input*: È stata aggiunta la validazione degli input per garantire che le distanze e i nodi vicini siano corretti.

- *Gestione delle Eccezioni*: È stata implementata la gestione delle eccezioni per catturare e gestire eventuali errori durante l'esecuzione del programma.

Ragionamento: La gestione degli errori e la validazione degli input migliorano la robustezza del codice e prevengono crash imprevisti.

4.4 Problema 4: Count to Infinity

Problematica: Il problema del "count to infinity" può verificarsi nei protocolli di routing a vettore di distanza, causando un aumento indefinito delle distanze in presenza di un ciclo nella rete.

Soluzione:

- *Split Horizon Semplice*: È stata implementata la tecnica dello "split horizon semplice" nel metodo `update_routing_table`. Questa tecnica impedisce a un nodo di aggiornare la sua tabella di routing con rotte apprese dallo stesso vicino, riducendo così il rischio di problemi come il "count to infinity".

Ragionamento: Utilizzare lo "split horizon semplice" aiuta a prevenire il problema del "count to infinity" in modo efficace, migliorando la stabilità del protocollo di routing.

5 Conclusione

Lo script Python implementa con successo un semplice protocollo di routing utilizzando il Distance Vector Routing. Ogni nodo aggiorna la propria tabella di routing in base alle informazioni ricevute dai vicini, e il processo continua fino a quando le tabelle di routing non convergono. Questo approccio garantisce che ogni nodo abbia la distanza minima a tutti gli altri nodi nella rete.