Simulazione di Protocollo di Routing Distance Vector in Python

Sofia Caberletti 0001071418

1. Introduzione

Il progetto consiste nell'implementazione di una simulazione di un protocollo di routing basato sull'algoritmo **Distance Vector Routing**. Ogni nodo mantiene una tabella di routing, o meglio un distance vector, che contiene le informazioni sui nodi che conosce della rete e le distanze per raggiungerli. L'algoritmo poi, grazie allo scambio dei distance vector tra nodi adiacenti, calcola le rotte più brevi tra i nodi della rete. La simulazione mira a mostrare il processo di aggiornamento iterativo delle tabelle di routing fino al raggiungimento della convergenza.

Gli obiettivi principali del progetto sono:

- Implementare la gestione delle tabelle di routing (distance vector).
- Simulare l'aggiornamento iterativo delle rotte tra nodi.
- Fornire output leggibili che mostrino l'evoluzione delle tabelle di routing fino alla convergenza.

2. Struttura del Progetto

Il progetto è strutturato in tre file principali:

- `main.py`: Contiene il programma principale, che consente all'utente di scegliere tra tre simulazioni di rete eseguibili.
- `node.py`: Definisce la classe `Node`, che rappresenta i singoli nodi della rete e include la logica per aggiornare la tabella di routing del nodo stesso.
- `network.py`: Definisce la classe `Network`, che rappresenta la rete complessiva e gestisce i collegamenti tra nodi e l'algoritmo di scambio e aggiornamento dei distance vector.

3. Funzionamento del Codice

3.1. Classe Node

La classe `Node` implementa la logica del nodo e la gestione delle tabelle di routing:

- Ogni nodo è identificato da un ID unico e mantiene un proprio distance vector `dv`, un dizionario in cui ogni chiave rappresentea la destinazione raggiungibile e il valore è una tupla `(distanza, next_hop)` che memorizza la distanza del nodo dalla destinazione e il prossimo nodo da cui passare per arrivare a destinazione.
- La funzione `update_node_dv` permette al nodo di aggiornare il proprio DV in base ai distance vector ricevuti dai vicini. La funzione riceve il distance vector di un nodo adiacente e per tutte le destinazioni presenti in quest'ultimo vede se è possibile aggiornare la rotta corrispondente nel distance vector del nodo. L'aggiornamento avviene in caso la destinazione non sia ancora presente nella routing table o in caso la distanza dalla destinazione sia minore passando per il nodo vicino. La funzione restituisce un

booleano che indica se il distance vector è stato aggiornato o meno a fronte del ricevimento della routing table del vicino.

```
    def update_node_dv(self, neigbour_dv, neighbour_id):
        updated = False

        for destination, (distance, _) in neigbour_dv.items():
            new_distance = distance + self.dv[neighbour_id][0]
            if destination not in self.dv or new_distance < self.dv[destination]
[0]:
                self.dv[destination] = (new_distance, self.dv[neighbour_id][1])
                updated = True

        return updated
```

- Le funzioni di stampa (`print_distance_vector` e `print_distance_vector_after_neighbour_dv_received`) forniscono una rappresentazione leggibile delle tabelle di routing del nodo.

3.2. Classe Network

La classe `Network` modella l'intera rete e gestisce i lo scambio di informazioni tra nodi:

- Consente di aggiungere alla rete nodi (`add_node`) e collegamenti tra nodi con costi specifici (`add_link`).
- Implementa la funzione `update_dvs`, che esegue l'aggiornamento iterativo delle tabelle di routing per tutti i nodi fino al raggiungimento della convergenza. Durante ogni iterazione, ogni nodo riceve i distance vector dei propri vicini richiamando per ognuno la funzione `update_node_dv` che se necessario aggiornerà la routing table del nodo. Nel caso in cui avvenga l'aggiornamento dopo aver ricevuto il distance vector di un vicino viene stampata la routing table aggiornata. Se dopo un'iterazione nessuna routing table è stata aggiornata significa che la rete è arrivata a convergenza.

```
print("DV INIZIALI: \n")
        self.print_all_dvs()
        net_updated = True
        iterations = 0
        while net_updated:
            net_updated = False
            iterations += 1
            print(f"ITERAZIONE {iterations} ----------------------------------
--------------\n")
            for node_id, node in self.nodes.items():
                for neighbour_id, _ in self.links.get(node_id).items():
                    neighbour_dv = self.nodes.get(neighbour_id).dv
                    node_route_updated = node.update_node_dv(neighbour_dv,
neighbour_id)
                    if node_route_updated:
                        net_updated = True
```

```
node.print_distance_vector_after_neighbour_dv_received(neighbour_id)


        print(f"NESSUN AGGIORNAMENTO, la CONVERGENZA è stata ottenuta in
{iterations-1} iterazioni\n")
        print("DV FINALI: \n")
        self.print_all_dvs()
```

- La funzione `reset_network` permette di resettare lo stato della rete per avviare nuove simulazioni.
- La funzione `print_all_dvs` stampa tutti i distance vectors dei nodi della rete.

3.3. File main.py

Il file `main.py`:

- Configura e gestisce diverse topologie di rete mediante tre funzioni di simulazione predefinite (`simulation1`, `simulation2`, `simulation3`). Forniamo qui il codice della `simulazione1` in quanto ci tornerà utile sapere la sua topologia di rete per mostrare successivamente un esempio di esecuzione e output del progetto.

```
def simulation1(Network):
    print("SIMULATION 1 ----------------------------------------------------\n")
    net.add_node("A")
    net.add_node("B")
    net.add_node("C")
    net.add_node("D")
    net.add_node("E")
    net.add_link("A", "B", 1)
    net.add_link("A", "C", 6)
    net.add_link("B", "C", 2)
    net.add_link("B", "D", 1)
    net.add_link("C", "D", 3)
    net.add_link("C", "E", 7)
    net.add_link("D", "E", 2)
```

- Permette all'utente di scegliere quale simulazione eseguire o uscire dal programma.

4. Risultati e Output

Ogni simulazione produce i seguenti risultati:

1. **Tabelle di routing iniziali**: inizialmente i distance vector contengono solo i costi tra i nodi direttamente adiacenti.
2. **Aggiornamenti iterativi**: Ad ogni iterazione, i nodi aggiornano i propri distance vector basandosi sulle informazioni ricevute dai vicini. Questo processo continua fino a quando nessun nodo necessita ulteriori aggiornamenti. In questa fase iterativa i distance vector vengono stampati solo nel caso in cui subiscano modifiche.
3. **Tabelle di routing finali**: Indicano la convergenza della rete e mostrano le rotte più brevi tra tutti i nodi.

4.1. Esempio di Esecuzione e relativo Output

Eseguiamo adesso la simulazione1 e osserviamone l'output:

Inserisci un numero di simulazione (1, 2 o 3), oppure 0 per uscire: **1**

SIMULATION 1 ----------

DV INIZIALI:

Distance Vector di A: Destinazione A --> Distanza = 0, Next Hop = A

Destinazione B --> Distanza = 1, Next Hop = B

Destinazione C --> Distanza = 6, Next Hop = C

Distance Vector di B: Destinazione B --> Distanza = 0, Next Hop = B

Destinazione A --> Distanza = 1, Next Hop = A

Destinazione C --> Distanza = 2, Next Hop = C

Destinazione D --> Distanza = 1, Next Hop = D

Distance Vector di C: Destinazione C --> Distanza = 0, Next Hop = C

Destinazione A --> Distanza = 6, Next Hop = A

Destinazione B --> Distanza = 2, Next Hop = B

Destinazione D --> Distanza = 3, Next Hop = D

Destinazione E --> Distanza = 7, Next Hop = E

Distance Vector di D: Destinazione D --> Distanza = 0, Next Hop = D

Destinazione B --> Distanza = 1, Next Hop = B

Destinazione C --> Distanza = 3, Next Hop = C

Destinazione E --> Distanza = 2, Next Hop = E

Distance Vector di E: Destinazione E --> Distanza = 0, Next Hop = E

Destinazione C --> Distanza = 7, Next Hop = C

Destinazione D --> Distanza = 2, Next Hop = D

ITERAZIONE 1 ----------

Distance Vector di A: Dopo aver ricevuto DV(B) Destinazione A --> Distanza = 0, Next Hop = A

Destinazione B --> Distanza = 1, Next Hop = B

Destinazione C --> Distanza = 3, Next Hop = B

Destinazione D --> Distanza = 2, Next Hop = B

Distance Vector di A: Dopo aver ricevuto DV(C) Destinazione A --> Distanza = 0, Next Hop = A

Destinazione B --> Distanza = 1, Next Hop = B

Destinazione C --> Distanza = 3, Next Hop = B

Destinazione D --> Distanza = 2, Next Hop = B

Destinazione E --> Distanza = 10, Next Hop = B

Distance Vector di B: Dopo aver ricevuto DV(A) Destinazione B --> Distanza = 0, Next Hop = B

Destinazione A --> Distanza = 1, Next Hop = A

Destinazione C --> Distanza = 2, Next Hop = C

Destinazione D --> Distanza = 1, Next Hop = D

Destinazione E --> Distanza = 11, Next Hop = A

Distance Vector di B: Dopo aver ricevuto DV(C) Destinazione B --> Distanza = 0, Next Hop = B

Destinazione A --> Distanza = 1, Next Hop = A

Destinazione C --> Distanza = 2, Next Hop = C

Destinazione D --> Distanza = 1, Next Hop = D

Destinazione E --> Distanza = 9, Next Hop = C

Distance Vector di B: Dopo aver ricevuto DV(D) Destinazione B --> Distanza = 0, Next Hop = B

Destinazione A --> Distanza = 1, Next Hop = A

Destinazione C --> Distanza = 2, Next Hop = C

Destinazione D --> Distanza = 1, Next Hop = D

Destinazione E --> Distanza = 3, Next Hop = D

Distance Vector di C: Dopo aver ricevuto DV(B) Destinazione C --> Distanza = 0, Next Hop = C

Destinazione A --> Distanza = 3, Next Hop = B

Destinazione B --> Distanza = 2, Next Hop = B

Destinazione D --> Distanza = 3, Next Hop = D

Destinazione E --> Distanza = 5, Next Hop = B

Distance Vector di D: Dopo aver ricevuto DV(B) Destinazione D --> Distanza = 0, Next Hop = D

Destinazione B --> Distanza = 1, Next Hop = B

Destinazione C --> Distanza = 3, Next Hop = C

Destinazione E --> Distanza = 2, Next Hop = E

Destinazione A --> Distanza = 2, Next Hop = B

Distance Vector di E: Dopo aver ricevuto DV(C) Destinazione E --> Distanza = 0, Next Hop = E

Destinazione C --> Distanza = 7, Next Hop = C

Destinazione D --> Distanza = 2, Next Hop = D

Destinazione A --> Distanza = 10, Next Hop = C

Destinazione B --> Distanza = 9, Next Hop = C

Distance Vector di E: Dopo aver ricevuto DV(D) Destinazione E --> Distanza = 0, Next Hop = E

Destinazione C --> Distanza = 5, Next Hop = D

Destinazione D --> Distanza = 2, Next Hop = D

Destinazione A --> Distanza = 4, Next Hop = D

Destinazione B --> Distanza = 3, Next Hop = D

ITERAZIONE 2 ----------

Distance Vector di A: Dopo aver ricevuto DV(B) Destinazione A --> Distanza = 0, Next Hop = A

Destinazione B --> Distanza = 1, Next Hop = B

Destinazione C --> Distanza = 3, Next Hop = B

Destinazione D --> Distanza = 2, Next Hop = B

Destinazione E --> Distanza = 4, Next Hop = B

ITERAZIONE 3 ----------

NESSUN AGGIORNAMENTO, la CONVERGENZA è stata ottenuta in 2 iterazioni

DV FINALI:

Distance Vector di A: Destinazione A --> Distanza = 0, Next Hop = A

Destinazione B --> Distanza = 1, Next Hop = B

Destinazione C --> Distanza = 3, Next Hop = B

Destinazione D --> Distanza = 2, Next Hop = B

Destinazione E --> Distanza = 4, Next Hop = B

Distance Vector di B: Destinazione B --> Distanza = 0, Next Hop = B

Destinazione A --> Distanza = 1, Next Hop = A

Destinazione C --> Distanza = 2, Next Hop = C

Destinazione D --> Distanza = 1, Next Hop = D

Destinazione E --> Distanza = 3, Next Hop = D

Distance Vector di C: Destinazione C --> Distanza = 0, Next Hop = C

Destinazione A --> Distanza = 3, Next Hop = B

Destinazione B --> Distanza = 2, Next Hop = B

Destinazione D --> Distanza = 3, Next Hop = D

Destinazione E --> Distanza = 5, Next Hop = B

Distance Vector di D: Destinazione D --> Distanza = 0, Next Hop = D

Destinazione B --> Distanza = 1, Next Hop = B

Destinazione C --> Distanza = 3, Next Hop = C

Destinazione E --> Distanza = 2, Next Hop = E

Destinazione A --> Distanza = 2, Next Hop = B

Distance Vector di E: Destinazione E --> Distanza = 0, Next Hop = E

Destinazione C --> Distanza = 5, Next Hop = D

Destinazione D --> Distanza = 2, Next Hop = D

Destinazione A --> Distanza = 4, Next Hop = D

Destinazione B --> Distanza = 3, Next Hop = D

Notiamo quindi che la simulazione è andata a buon fine ed è effettivamente arrivata a convergenza dopo due iterazioni, in quanto nella terza iterazione non avvengono più aggiornamenti delle routing tables. I distance vector **iniziali** contengono solo le rotte dei nodi adiacenti con i costi del collegamento diretto mentre i distance vector **finali** presentano le rotte di tutte le destinazioni raggiungibili e con il minor costo.

5. Considerazioni finali e Possibili estensioni

Il progetto dimostra l'efficacia del Distance Vector Routing nell'identificare le rotte più brevi in una rete. Abbiamo verificato la riuscita creazione di una rete, l'aggiunta di nodi e link ad essa e il corretto funzionamento del Distance Vector Routing.

Una possibile estensione del programma potrebbe essere l'introduzione di tecniche migliorative come **Split Horizon** e **Triggered Update** per evitare situazioni come bouncing effect e lenta convergenza nel qual caso si volesse simulare anche il fallimento di nodi e collegamenti con le relative conseguenze sulle routing table.