Eraldo Tabaku, Matricola: 0001089499

Documentazione del progetto di Laboratorio di Reti di Telecomunicazioni

Traccia 2 Progetto Python: Simulazione di Protocollo di Routing

- Descrizione: Creare uno script Python che simuli un protocollo di routing semplice, come il Distance Vector Routing. Gli studenti implementeranno gli aggiornamenti di routing tra i nodi, con il calcolo delle rotte più brevi.
- Obiettivi: Implementare la logica di aggiornamento delle rotte, gestione delle tabelle di routing e calcolo delle distanze tra nodi.
- Consegne richieste: Codice Python ben documentato, output delle tabelle di routing per ogni nodo e relazione finale che spieghi il funzionamento dello script.

Introduzione

Il Distance Vector Routing (DVR) è un protocollo distribuito per il calcolo del percorso più breve tra i nodi di una rete. Ogni nodo invia in maniera periodica le proprie informazioni di routing ai vicini, aggiornando la propria tabella di routing sulla base delle informazioni ricevute. Questo progetto simula il DVR su una rete generata casualmente, includendo una versione semplificata di gestione degli errori, come la rimozione casuale dei collegamenti tra i nodi.

Funzionamento del codice

Di seguito vengono riportate alcune porzioni di codice significative con i relativi commenti riguardo il funzionamento.

Struttura della Classe Node

La classe Node rappresenta un nodo della rete. Ogni nodo mantiene:

- Tabella di routing (routing_table): un dizionario con i costi minimi per raggiungere ciascun nodo.
- Vicini (neighbors): un dizionario che associa a ogni vicino la distanza diretta.

```python
class Node:
    def __init__(self, name):
        self.name = name
        self.routing_table = {name: 0}  # Distanza a se stesso è sempre 0
        self.neighbors = {}

    def add_neighbor(self, neighbor, distance):
        self.neighbors[neighbor] = distance
        self.routing_table[neighbor.name] = distance

    def remove_neighbor(self, neighbor):
        if neighbor in self.neighbors:
            del self.neighbors[neighbor]
        if neighbor.name in self.routing_table:
            del self.routing_table[neighbor.name]

    def update_routing_table(self):
        updated = False
        for neighbor, dist in self.neighbors.items():
            for destination, neighbor_dist in neighbor.routing_table.items():
                alt_distance = dist + neighbor_dist
                if destination not in self.routing_table or self.routing_table[destination] > alt_distance:
                    self.routing_table[destination] = alt_distance
                    updated = True
        return updated
```

Descrizione dei metodi

- add_neighbor: aggiunge un vicino al nodo con una distanza specificata.
- remove_neighbor: rimuove un collegamento con un vicino specifico.
- update_routing_table: aggiorna la tabella di routing in base all'algoritmo di Bellman-Ford: per ogni vicino, valuta se un percorso alternativo offre una distanza minore; restituisce True se la tabella è stata modificata.

Creazione della Rete

```python
def create_random_network(num_nodes, max_distance=10):
    nodes = [Node(chr(65 + i)) for i in range(num_nodes)]

    for i in range(num_nodes):
        for j in range(i + 1, num_nodes):
            if random.random() < 0.5:
                distance = random.randint(1, max_distance)
                nodes[i].add_neighbor(nodes[j], distance)
                nodes[j].add_neighbor(nodes[i], distance)

    return nodes
```

La funzione create_random_network genera una rete casuale con il numero di nodi e una distanza massima tra i nodi specificati come parametri. La creazione randomica della rete sfrutta la libreria “random” importata, che viene utilizzata sia per definire se due nodi sono collegati (con una probabilità del 50%) e sia per stabilire la distanza tra i nodi.

(ovviamente il collegamento tra i nodi è bidirezionale: se A è collegato a B, allora anche B è collegato ad A).

Simulazione del DVR con Gestione degli Errori

```python
def simulate_routing(network, max_iterations=10, error_probability=0.2):
    converged = False
    iteration = 0

    while not converged and iteration < max_iterations:
        converged = True
        print(f"\nIterazione {iteration}:")

        # Simulazione di errori: rimozione casuale di collegamenti
        for node in network:
            if random.random() < error_probability:
                if node.neighbors:
                    neighbor_to_remove = random.choice(list(node.neighbors.keys()))
                    print(f"Errore: Rimosso il collegamento tra {node.name} e {neighbor_to_remove.name}")
                    node.remove_neighbor(neighbor_to_remove)
                    neighbor_to_remove.remove_neighbor(node)

        for node in network:
            if node.update_routing_table():
                converged = False

        for node in network:
            print(f"Tabella di routing per il nodo {node.name}: {node.routing_table}")

        iteration += 1

    if converged:
        print("\nConvergenza raggiunta.")
    else:
        print("\nTermine simulazione: massimo numero di iterazioni raggiunto.")
```

La funzione simulate_routing implementa la simulazione del protocollo:

1. Aggiornamenti delle Tabelle: Ogni nodo aggiorna la propria tabella di routing in base alle informazioni dei vicini.
2. Gestione degli Errori: durante ogni iterazione, con una probabilità “error_probability”, un collegamento casuale tra un nodo e uno dei suoi vicini viene rimosso ed entrambi i nodi aggiornano la loro lista di vicini e tabelle di routing.
3. Convergenza: la simulazione termina quando la rete è stabile oppure quando viene raggiunto il numero massimo di iterazioni (max_iterations).

MAIN

```python
if __name__ == "__main__":
    NUM_NODES = 5
    MAX_DISTANCE = 10
    ERROR_PROBABILITY = 0.2

    network = create_random_network(NUM_NODES, MAX_DISTANCE)

    print("Topologia della rete iniziale (collegamenti):")
    for node in network:
        for neighbor, distance in node.neighbors.items():
            print(f"{node.name} --{distance}--> {neighbor.name}")

    simulate_routing(network, error_probability=ERROR_PROBABILITY)
```

All’interno del main dello script vengono inizialmente definiti numero di nodi, distanza massima tra i nodi e probabilità di errore, in seguito viene creata la rete con l’utilizzo della funzione apposita.

Prima della simulazione viene stampata la topologia della rete.

Infine, viene avviata la simulazione del DVR che andrà a stampare a video gli eventuali collegamenti rimossi e gli aggiornamenti delle varie tabelle di routing per i singoli nodi.

Funzionamento e utilizzo del sistema

Di seguito vengono forniti in paio di catture per mostrare il funzionamento dello script.

```
C:\Users\erald\Desktop\ProgettoRetiDiTelecomunicazione2024---Script-Python>python Script.py
Topologia della rete (collegamenti):
A --2--> B
A --7--> C
A --9--> D
B --2--> A
B --4--> D
B --4--> E
C --7--> A
C --10--> D
D --9--> A
D --4--> B
D --10--> C
D --3--> E
E --4--> B
E --3--> D
```

Dopo aver eseguito lo script da terminale mediante l’apposito comando, viene stampata la topologia della rete mostrando i collegamenti tra i singoli nodi.

Di seguito vengono mostrate le varie iterazione con i relativi aggiornamenti delle tabelle di routing, che hanno portato alla stabilità della rete.

```
Iterazione 0:
Tabella di routing per il nodo A: {'A': 0, 'B': 2, 'C': 7, 'D': 6, 'E': 6}
Tabella di routing per il nodo B: {'B': 0, 'A': 2, 'D': 4, 'E': 4, 'C': 9}
Tabella di routing per il nodo C: {'C': 0, 'A': 7, 'D': 10, 'B': 9, 'E': 13}
Tabella di routing per il nodo D: {'D': 0, 'A': 6, 'B': 4, 'C': 10, 'E': 3}
Tabella di routing per il nodo E: {'E': 0, 'B': 4, 'D': 3, 'A': 6, 'C': 13}

Iterazione 1:
Tabella di routing per il nodo A: {'A': 0, 'B': 2, 'C': 7, 'D': 6, 'E': 6}
Tabella di routing per il nodo B: {'B': 0, 'A': 2, 'D': 4, 'E': 4, 'C': 9}
Tabella di routing per il nodo C: {'C': 0, 'A': 7, 'D': 10, 'B': 9, 'E': 13}
Tabella di routing per il nodo D: {'D': 0, 'A': 6, 'B': 4, 'C': 10, 'E': 3}
Tabella di routing per il nodo E: {'E': 0, 'B': 4, 'D': 3, 'A': 6, 'C': 13}
```

Per completezza, di seguito viene fornito un esempio di output nel caso in cui sia avvenuto un errore, nello specifico la schermata mostra che alla seconda iterazione (iterazione 1) si è verificato un errore dovuto al collegamento rimosso tra i nodi D e C.

```
C:\Users\erald\Desktop\ProgettoRetiDiTelecomunicazione2024---Script-Python>python Script.py
Topologia della rete iniziale (collegamenti):
A --1--> C
B --7--> C
C --1--> A
C --7--> B
C --8--> D
D --8--> C
D --3--> E
E --3--> D

Iterazione 0:
Tabella di routing per il nodo A: {'A': 0, 'C': 1, 'B': 8, 'D': 9}
Tabella di routing per il nodo B: {'B': 0, 'C': 7, 'A': 8, 'D': 15}
Tabella di routing per il nodo C: {'C': 0, 'A': 1, 'B': 7, 'D': 8, 'E': 11}
Tabella di routing per il nodo D: {'D': 0, 'C': 8, 'E': 3, 'A': 9, 'B': 15}
Tabella di routing per il nodo E: {'E': 0, 'D': 3, 'C': 11, 'A': 12, 'B': 18}

Iterazione 1:
Errore: Rimosso il collegamento tra D e C
Tabella di routing per il nodo A: {'A': 0, 'C': 1, 'B': 8, 'D': 9, 'E': 12}
Tabella di routing per il nodo B: {'B': 0, 'C': 7, 'A': 8, 'D': 15, 'E': 18}
Tabella di routing per il nodo C: {'C': 0, 'A': 1, 'B': 7, 'E': 11, 'D': 10}
Tabella di routing per il nodo D: {'D': 0, 'E': 3, 'A': 9, 'B': 15, 'C': 14}
Tabella di routing per il nodo E: {'E': 0, 'D': 3, 'C': 11, 'A': 12, 'B': 18}

Iterazione 2:
Tabella di routing per il nodo A: {'A': 0, 'C': 1, 'B': 8, 'D': 9, 'E': 12}
Tabella di routing per il nodo B: {'B': 0, 'C': 7, 'A': 8, 'D': 15, 'E': 18}
Tabella di routing per il nodo C: {'C': 0, 'A': 1, 'B': 7, 'E': 11, 'D': 10}
Tabella di routing per il nodo D: {'D': 0, 'E': 3, 'A': 9, 'B': 15, 'C': 14}
Tabella di routing per il nodo E: {'E': 0, 'D': 3, 'C': 11, 'A': 12, 'B': 18}

Convergenza raggiunta.
```

(Si rende noto che lo script è stato scritto utilizzando la versione 3.12.7 di Python)