Progetto di Reti di Telecomunicazione

Sara Panfini, Matricola: 0001082217

12 dicembre 2024

Indice

1 Introduzione

Il progetto ha come obiettivo l'implementazione e la simulazione di un protocollo di routing in Python. L'algoritmo utilizzato è il *Distance Vector Routing*, che permette di determinare il percorso ottimale tra i nodi di una rete, in base al costo associato a ciascun collegamento. Ogni nodo ha una propria tabella di routing, che aggiorna iterativamente fino a raggiungere la convergenza.

2 Struttura del Codice

Il codice è suddiviso in due classi principali.

- `Node` – rappresenta un nodo della rete
- `Network` – rappresenta l'intera rete di nodi

3 Descrizione del Codice

3.1 Classe `Node`

La classe `Node` rappresenta un singolo nodo nella rete. Ogni nodo ha una propria tabella di routing e mantiene informazioni sui nodi che gli sono direttamente vicini. Questa classe si occupa anche di gestire l'aggiornamento della tabella di routing.

```
class Node:
def __init__(self, name):
    self.name = name
    self.neighbors = {}
    self.routing_table = {name: 0}

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

def update_routing_table(self):
    updated = False
    for neighbor, cost in self.neighbors.items():
        for dest, curr_cost in neighbor.routing_table.items():
            new_cost = curr_cost + cost
            if dest not in self.routing_table or new_cost < self.
               routing_table[dest]:
                self.routing_table[dest] = new_cost
                updated = True
    return updated

def __str__(self):
    result = f"\nTabella di routing per il nodo {self.name}\n"
    result += "Destinazione | Costo\n"
    for dest, cost in sorted(self.routing_table.items()):
        result += f"     {dest}       |   {cost}\n"
    return result
```

Classe `Node`

Attributi

- `name`, identificatore del nodo
- `neighbors`, associa ai vicini diretti del nodo i relativi costi di collegamento
- `routing_table`, rappresenta la tabella di routing del nodo

Metodi

- `__init__(self, name)`, inizializza il nodo con un nome, una lista vuota di nodi vicini e la tabella di routing contenente solo il nodo stesso con costo pari a zero

- `add_neighbor(self, neighbor, cost)`, aggiunge un nodo vicino, con il costo di collegamento specificato, alla lista dei vicini e aggiorna la tabella di routing inserendo il costo per quel nodo vicino
- `update_routing_table(self)`, implementa l'algoritmo di DVR, cioè si occupa di aggiornare la tabella di routing del nodo se, esaminando le tabelle di routing dei nodi vicini, riesce a trovare percorsi più convenienti; restituisce `True` se la tabella viene aggiornata, altrimenti `False`
- `__str__`, restituisce una rappresentazione formattata e leggibile della tabella di routing del nodo

3.2 Classe `Network`

La classe `Network` rappresenta l'intera rete di nodi. Si occupa di creare i nodi ed i collegamenti tra di essi e di gestire il processo di aggiornamento delle tabelle di routing di tutti i nodi fino al raggiungimento della convergenza. La convergenza si raggiunge quando nessun nodo può ulteriormente migliorare il costo dei percorsi per raggiungere gli altri nodi.

```
class Network:
def __init__(self):
    self.nodes = {}

def add_node(self, name):
    self.nodes[name] = Node(name);

def add_link(self, node1_name, node2_name, cost):
    node1 = self.nodes[node1_name]
    node2 = self.nodes[node2_name]
    node1.add_neighbor(node2, cost)
    node2.add_neighbor(node1, cost)

def distance_vector_routing(self):
    converged = False
    iterations = 0

    while not converged:
        converged = True
        iterations += 1
        print(f"\n=== Iterazione {iterations} ===")
        for node in self.nodes.values():
            if node.update_routing_table():
                converged = False

        for node in self.nodes.values():
            print(node)

    print(f"Convergenza dopo {iterations} iterazioni")
```

Classe `Network`

Attributi

- `nodes`, memorizza i nodi della rete, con il nome del nodo come chiave e l'istanza di `Node` come valore

Metodi

- `__init__(self)`, inizializza una rete vuota
- `add_node(self, name)`, crea ed aggiunge un nuovo nodo alla rete, con il nome specificato
- `add_link(self, node1_name, node2_name, cost)`, crea un collegamento bidirezionale tra due nodi della rete, specificandone il relativo costo
- `distance_vector_routing(self)`, simula l'esecuzione dell'algoritmo DVR, gestendo l'aggiornamento iterativo delle tabelle di routing dei nodi, e stampa le tabelle dopo ogni iterazione fino alla convergenza, indicando anche il numero totale di iterazioni necessarie a raggiungerla

4 Esecuzione del Codice

L'esecuzione del codice inzia con la creazione di un'istanza della classe `Network`, che gestisce l'intera rete di nodi. Dopo aver configurato la rete, l'algoritmo di routing viene avviato con il metodo `distance_vector_routing(self)`.

4.1 Input

L'input del codice è costituito dalla definizione iniziale dei nodi e dei collegamenti tra di essi, con i relativi costi associati, che rappresentano la rete su cui l'algoritmo DVR viene eseguito.

```
network = Network()

for node_name in ["A", "B", "C", "D", "E"]:
    network.add_node(node_name)

network.add_link("A", "B", 2)
network.add_link("B", "C", 3)
network.add_link("C", "D", 1)
network.add_link("D", "E", 5)
network.add_link("A", "D", 1)
network.add_link("A", "E", 1)
network.add_link("B", "D", 2)
network.add_link("C", "E", 4)
```

Definizione della rete

4.2 Output

Ad ogni iterazione vengono stampate a schermo le tabelle di routing aggiornate di ciascun nodo della rete. Al termine dell'esecuzione, l'algoritmo di routing ha trovato i percorsi ottimali tra i nodi raggiungendo la convergenza. Le tabelle di routing finali indicano i percorsi più efficienti per ciascun nodo, con i costi associati.

```
=== Iterazione 1 ===

Tabella di routing per il nodo A
Destinazione | Costo
     A       |   0
     B       |   2
     C       |   2
     D       |   1
     E       |   1


Tabella di routing per il nodo B
Destinazione | Costo
     A       |   2
```

```
      B        |   0
      C        |   3
      D        |   2
      E        |   3


Tabella di routing per il nodo C
Destinazione | Costo
      A        |   2
      B        |   3
      C        |   0
      D        |   1
      E        |   4


Tabella di routing per il nodo D
Destinazione | Costo
      A        |   1
      B        |   2
      C        |   1
      D        |   0
      E        |   2


Tabella di routing per il nodo E
Destinazione | Costo
      A        |   1
      B        |   3
      C        |   3
      D        |   2
      E        |   0


=== Iterazione 2 ===

Tabella di routing per il nodo A
Destinazione | Costo
      A        |   0
      B        |   2
      C        |   2
      D        |   1
      E        |   1


Tabella di routing per il nodo B
Destinazione | Costo
      A        |   2
      B        |   0
      C        |   3
      D        |   2
      E        |   3
```

```
66
67
68 Tabella di routing per il nodo C
69 Destinazione | Costo
70      A        |   2
71      B        |   3
72      C        |   0
73      D        |   1
74      E        |   3
75
76
77 Tabella di routing per il nodo D
78 Destinazione | Costo
79      A        |   1
80      B        |   2
81      C        |   1
82      D        |   0
83      E        |   2
84
85
86 Tabella di routing per il nodo E
87 Destinazione | Costo
88      A        |   1
89      B        |   3
90      C        |   3
91      D        |   2
92      E        |   0
93
94
95 === Iterazione 3 ===
96
97 Tabella di routing per il nodo A
98 Destinazione | Costo
99      A        |   0
100     B        |   2
101     C        |   2
102     D        |   1
103     E        |   1
104
105
106 Tabella di routing per il nodo B
107 Destinazione | Costo
108     A        |   2
109     B        |   0
110     C        |   3
111     D        |   2
112     E        |   3
113
114
115 Tabella di routing per il nodo C
116 Destinazione | Costo
```

```
      A        |   2
      B        |   3
      C        |   0
      D        |   1
      E        |   3


Tabella di routing per il nodo D
Destinazione | Costo
      A        |   1
      B        |   2
      C        |   1
      D        |   0
      E        |   2


Tabella di routing per il nodo E
Destinazione | Costo
      A        |   1
      B        |   3
      C        |   3
      D        |   2
      E        |   0

Convergenza dopo 3 iterazioni
```

Output atteso

5 Conclusioni

Il progetto mostra come implementare il protocollo *Distance Vector Routing* per simularne l'esecuzione su una rete di nodi, con aggiornamento dinamico delle tabelle di routing. I risultati ottenuti mostrano che l'algoritmo è stato in grado di convergere rapidamente, stabilizzando le tabelle di routing in poche iterazioni. Nonostante la buona performance in reti semplici, l'efficienza del protocollo può, comunque, variare nel caso di gestione di reti più complesse o con topologia differente. Per migliorarne le prestazioni, si potrebbero introdurre varianti dell'algoritmo o tecniche di ottimizzazione.