Routing Protocol

Luca Marchi

December 2024

1 Introduzione

Questo progetto, sviluppato in Python, simula il funzionamento del protocollo di routing **Distance Vector Routing**. A partire da una matrice di adiacenza fornita come input, che descrive i nodi della rete e le relative distanze, il software genera un grafo pesato (la rete) e le tabelle di routing associate. L'algoritmo di Bellman-Ford è utilizzato per aggiornare i distance vector e calcolare i cammini minimi tra ogni coppia di nodi del grafo. Per rendere visibile il grafo e le tabelle di routing, il progetto sfrutta le potenzialità della libreria Tkinter di Python.

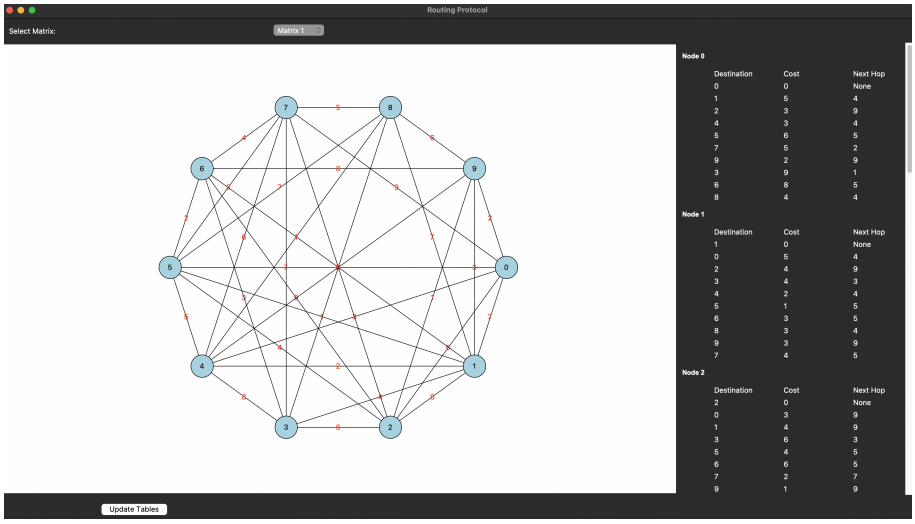

Figure 1: Rappresentazione GUI

2 Organizzazione e design del codice

2.1 Logica

Il file *network.py* contiene la logica e gli algoritmi utilizzati dal software. Al suo interno è presente la classe *Network* che simula una rete di routers e delle loro routing tables.

2.1.1 Creazione del grafo

Il grafo (la rete) viene creato da una matrice di adiacenza (presa come paramentro dal costruttore della classe) attraverso il metodo *graph_build()*, che grazie ad una hashmap associa ad ogni nodo, i vicini e il peso corrispondente alla distanza tra essi.

2.1.2 Aggiornamento routing tables

Infine la funzione *update_tables()* si occupa di aggiornare le tabelle di routing, le quali inizialmente contengono solamente le informazioni relative ai vicini (per tutti i nodi). A ogni chiamata del metodo, a tutti i nodi vengono inviati i distance vector dei router adiacenti che sfruttano le informazioni per aggiornare le routing tables con le distanze minori dai nodi della rete. Il processo viene ripetuto finchè tutti i nodi non hanno scoperto il percorso ottimale per raggingere qualsialsi altro punto del grafo.

```python
def update_tables(self)-> None:
    """Updates the tables of the network.
    """
    # loop through each router
    for router in self.routers:
        # visit the neighbors of the current router
        for neigh in self.routers[router]:
            # use the distance vector of the neighbors to update the table of the current router
            for node, distance_vector in self.tables[neigh].items():
                # if the node is not the current router and the node is not in the table of the current router
                if node != router and node not in self.tables[router]:
                    # update the table of the current router
                    self.tables[router][node] = (self.tables[router][neigh][0] + distance_vector[0], neigh)
                # if the node is not the current router and the node is in the table of the current router
                elif node != router:
                    # if the distance vector of the current node is less than the distance vector of the node in the table of the current router
                    if self.tables[router][neigh][0] + distance_vector[0] < self.tables[router][node][0]:
                        # update the table of the current router
                        self.tables[router][node] = (self.tables[router][neigh][0] + distance_vector[0], neigh)
```

Figure 2: Codice aggiornamento routing tables

2.2 GUI

L'interfaccia grafica permette all'utente di osservare la rete sulla quale si vuole simulare il protocollo di routing e si trova nel file *GUI.py*. La GUI è suddivisa in due parti principali:

- **Grafo**: Tramite un menu a tendina nella parte alta si può scegliere tra 5 tipologie di grafo differenziati dal numero di nodi e dal peso dei relativi

archi. I routers vengono rappresentati dal metodo *draw_graph()* tramite cerchi e i collegamenti tramite linee tra nodi adiacenti.

- **Routing Tables**: Nella parte desta del grafo sono presenti le tabelle di routing disegnate dal metodo *update_tables()*. Le tabelle hanno 3 colonne, la destinazione, il costo per arrivare a quella determinata destinazione, e il prossimo nodo da raggiungere in quel cammino minimo. A ogni pressione del pulsante *Update Tables* queste ultime vengono aggiornate con le informazioni relative ai distance vector dei nodi adiacenti. In questo modo si possono seguire passo passo le iterazioni dell'algoritmo.

Nel costruttore di *GUI.py* viene creata un'istanza della classe *Network*, la quale verrà utilizzata per la creazione del grafo e delle routing tables.

2.3 Tipologie di network

Nel file *graphs.py* sono presenti 5 tipologie di rete rappresentate da una matrice di adiacenza. Viene fornita questa scelta per poter osservare l'andamento del protocollo Distance Vector Routing con grafi di diversa tipologia e apprenderne meglio il funzionamento.

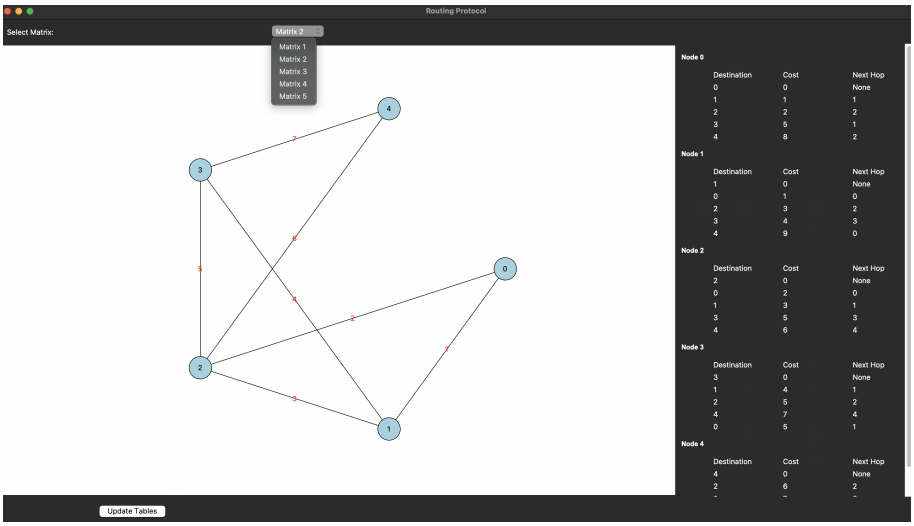

Figure 3: Grafi

3 Come eseguire il programma

Il software viene eseguito attraverso il comando *python*3 *main.py* da riga di comando. Nel file *main.py* viene creata un'istanza della classe *GUI.py* e fatto partire il mainloop.