

Simulatore Distance Vector Routing
Report per il Progetto di
Reti di Telecomunicazioni A.A. 2024/2025

Marco Marrelli

Dicembre 2024

Indice

0.1	Introduzione	2
0.2	Requisiti	2
0.3	Struttura della Codebase	3
0.3.1	Componenti per la Grafica	3
0.3.2	Componenti per la Logica	3
0.4	Implementazione della Grafica	4
0.4.1	Visualizzazione Grafica della Rete	4
0.4.2	Manipolazione della Rete	4
0.5	Parte Logica	5
0.5.1	L'Algoritmo (Distance Vector Routing)	5
0.5.2	Implementazione della Logica	6
0.6	Conclusioni	7

0.1 Introduzione

Questo progetto riguarda, tramite un simulatore, l'algoritmo di Distance Vector Routing (DVR), un algoritmo utilizzato nel routing sulle reti network. L'applicazione creata permette di creare e manipolare (con dei limiti prefissati) una rete di router, visualizzarla graficamente e simulare l'esecuzione dell'algoritmo di routing su di essa. L'output dell'algoritmo (la tabella di routing di ogni router) può essere consultata tramite l'applicazione grafica.

0.2 Requisiti

- Python 3.6 (o superiore)
- Libreria Tkinter (Framework per la parte Grafica)
- Librerie Standard di Python, come:
 - typing (per i Type Hints)
 - math (per Calcoli e Costanti Matematiche)
 - random (per l'aspetto della Randomicità)

0.3 Struttura della Codebase

0.3.1 Componenti per la Grafica

- **gui.py**: Implementa l'interfaccia grafica per il simulatore DVR.
- **constants.py**: Contiene le costanti utilizzate nell'applicazione, inclusi i parametri grafici.
- **utils.py**: Fornisce funzioni di utilità per calcoli nella grafica (percentuale di un numero).

0.3.2 Componenti per la Logica

- **router.py**: Implementa la logica del router e l'algoritmo di Distance Vector Routing.
- **node.py**: Definisce la rappresentazione di un nodo nella rete.
- **distance.py**: Gestisce le distanze tra i nodi e le operazioni su di esse.
- **routing_table.py**: Gestisce le tabelle di routing per i router.

0.4 Implementazione della Grafica

0.4.1 Visualizzazione Grafica della Rete

La visualizzazione grafica della rete è gestita dalla classe **NetworkGUI** (nel file `gui.py`). Questa classe utilizza la libreria Tkinter per creare una finestra, con all'interno un canvas dove vengono disegnati i nodi (router) e le connessioni tra di essi.

- **Creazione della Finestra Principale:** La finestra principale viene creata e configurata con un titolo e dimensioni specifiche.
- **Canvas per il Disegno della Rete:** Un canvas è utilizzato per disegnare i nodi e le connessioni. I nodi sono rappresentati come cerchi dorati e le connessioni come linee.
- **Visualizzazione delle Connessioni:** Le connessioni tra i nodi sono disegnate come linee sul canvas. I costi delle connessioni sono visualizzati come rettangoli bianchi con il testo del costo al centro della linea di connessione.

0.4.2 Manipolazione della Rete

La manipolazione della rete è gestita attraverso vari pulsanti e funzioni che permettono di aggiungere, rimuovere e randomizzare i nodi e le connessioni.

- **Aggiunta di Nodi:** Il pulsante "Add Router" permette di aggiungere un nuovo nodo alla rete. La funzione `add_node()` gestisce l'aggiunta del nodo e la sua visualizzazione sul canvas.
- **Rimozione di Nodi:** Il pulsante "Remove Router" permette di rimuovere un nodo esistente dalla rete. La funzione `remove_node()` gestisce la rimozione del nodo e l'aggiornamento del canvas.
- **Randomizzazione delle Connessioni:** Il pulsante "Randomize" permette di randomizzare le connessioni tra i nodi. La funzione `create_network_connections()` gestisce la creazione e visualizzazione delle connessioni randomiche.
- **Avvio della Simulazione:** Il pulsante "Start Simulation" avvia l'algoritmo di Distance Vector Routing. La funzione `run_algorithm()` gestisce l'esecuzione dell'algoritmo e l'aggiornamento delle tabelle di routing.

0.5 Parte Logica

0.5.1 L'Algoritmo (Distance Vector Routing)

Il Distance Vector Routing (conosciuto anche come routing di Bellman-Ford) è un algoritmo di routing dinamico dove ogni router:

- Mantiene una tabella delle distanze minime note (vettore) verso ogni destinazione;
- Scambia (periodicamente) queste informazioni con i router direttamente collegati ("vicini");
- Aggiorna le proprie distanze quando scopre percorsi più brevi attraverso i vicini.

Viene utilizzata l'equazione di Bellman-Ford per calcolare i percorsi ottimali

$$d_x(y) = \min_v \{c(x, v) + d_v(y)\}$$

- $d_x(y)$: distanza minima dal nodo x al nodo y ;
- $c(x, v)$: costo del collegamento diretto tra x e il vicino v ;
- $d_v(y)$: distanza minima dal vicino v al nodo y ;

Questo approccio permette ai router di selezionare in maniera dinamica la rotta ottimale.

0.5.2 Implementazione della Logica

Questo è il codice (semplificato) dell'algoritmo implementato (visualizzabile nel file `router.py`, nella funzione `updateroutes`)

```
def update_routes(route_tables: List[RouteTable]) -> bool:
    neighbors: List[Node] = []
    node: Node = None
    changed: bool = False

    for neighbor, distance_from_neighbor in neighbors.items():
        current_distance = route_table.get_route(node, neighbor)

        if distance_from_neighbor < current_distance:
            route_table.add_route(node, neighbor, distance_from_neighbor, neighbor)
            changed = True

    for route_table in route_tables:
        if neighbor not in route_table.routes:
            continue

        for dest, dest_distance in route_table.routes[neighbor].items():
            if dest == node:
                continue

            new_distance = distance_from_neighbor + dest_distance
            current_distance = route_table.get_route(node, dest)

            if new_distance < current_distance:
                route_table.add_route(node, dest, new_distance, neighbor)
                changed = True

    return changed
```

Figure 1: `router.py::update_routes()` : Pseudocode dell'algoritmo DVR

La funzione `update_routes` nel file `router.py` è responsabile dell'aggiornamento delle rotte nel router utilizzando l'algoritmo di Distance Vector Routing. Di seguito viene fornita una spiegazione dettagliata del funzionamento della funzione:

- **Inizializzazione del Flag:** Viene inizializzato un flag (booleano) `changed` che viene utilizzato per tracciare se ci sono stati aggiornamenti nelle rotte (`update_routes`, riga 13). Verrà successivamente returnato.
- **Iterazione sui Vicini:** La funzione itera su ogni vicino del router e sulla distanza associata (`update_routes`, riga 16). Per ogni vicino, ottiene la distanza corrente verso di esso dalla tabella di routing (`update_routes`, riga 17).
- **Aggiornamento della Rotta Diretta:** Se la nuova distanza verso il vicino è più corta della distanza corrente, la rotta viene aggior-

nata nella tabella di routing e il flag `changed` viene impostato a `True` (`update_routes`, righe 20 : 23).

- **Iterazione sulle Tabelle di Routing degli Altri Router:** La funzione itera su ogni tabella di routing degli altri router (`update_routes`, riga 26). Se il vicino non è presente nella tabella corrente, viene saltato (`update_routes`, riga 27).
- **Iterazione sulle Destinazioni:** Per ogni destinazione e la sua distanza dal vicino, la funzione verifica se la destinazione è il nodo corrente e, in tal caso, la salta (`update_routes`, righe 30 : 32).
- **Calcolo della Nuova Distanza:** La funzione calcola la nuova distanza verso la destinazione attraverso il vicino e ottiene la distanza corrente verso la destinazione dalla tabella di routing (`update_routes`, righe 35 : 37).
- **Aggiornamento della Rotta Indiretta:** Se la nuova distanza è più corta della distanza corrente, la rotta viene aggiornata nella tabella di routing e il flag `changed` viene impostato a `True` (`update_routes`, righe 40 : 43).
- **Return del Flag changed:** Come già scritto all'inizio, l'ultima riga di codice riguarda il return del flag `changed`, per indicare se ci sono stati aggiornamenti nelle rotte (`update_routes`, riga 45).

0.6 Conclusioni