Relazione sulla Simulazione di Protocollo di Routing

Corso di Reti di Telecomunicazione - A.A. 2024/2025

Elisa Yan - 0001070882
elisa.yan@studio.unibo.it

10 dicembre 2024

1 Introduzione

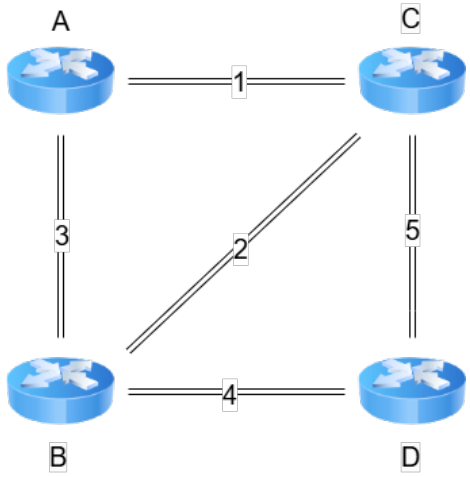

Figura 1: Esempio di distance vector routing

Questo progetto si concentra sulla simulazione di un protocollo di routing semplice, ispirato al Distance Vector Routing, utilizzando il linguaggio Python.

Il sistema è progettato per la modellazione di una rete con nodi e collegamenti bidirezionali, ognuno con un costo associato, e per la simulazione dell'aggiornamento iterativo delle tabelle di routing sulla base delle informazioni condivise dai nodi vicini. Ogni nodo del sistema rappresenta un dispositivo nella rete, come un router, e ognuno ha una tabella di routing utilizzata per mantenere le informazioni sui costi minimi necessari per raggiungere gli altri nodi.

Il Distance Vector Routing è un protocollo di routing progettato per calcolare il percorso ottimale tra i nodi. Si basa su un algoritmo iterativo e distribuito che consente a ciascun nodo di costruire la propria tabella di instradamento mediante l'interscambio di informazioni con i nodi vicini.

2 Funzionamento del sistema

All'inizio, ogni nodo conosce solo il costo per raggiungere i propri vicini diretti, con un costo di zero per sé stesso e considerato infinito per le altre destinazioni.

```
Initial Routing Tables:
Routing table for Node A:
  A --> A (Cost: 0, Next Hop: None)
  A --> B (Cost: inf, Next Hop: None)
  A --> C (Cost: inf, Next Hop: None)
  A --> D (Cost: inf, Next Hop: None)

Routing table for Node B:
  B --> A (Cost: inf, Next Hop: None)
  B --> B (Cost: 0, Next Hop: None)
  B --> C (Cost: inf, Next Hop: None)
  B --> D (Cost: inf, Next Hop: None)

Routing table for Node C:
  C --> A (Cost: inf, Next Hop: None)
  C --> B (Cost: inf, Next Hop: None)
  C --> C (Cost: 0, Next Hop: None)
  C --> D (Cost: inf, Next Hop: None)

Routing table for Node D:
  D --> A (Cost: inf, Next Hop: None)
  D --> B (Cost: inf, Next Hop: None)
  D --> C (Cost: inf, Next Hop: None)
  D --> D (Cost: 0, Next Hop: None)
```

Figura 2: Tabella di routing iniziale

A intervalli regolari, ogni nodo comunica la propria tabella di instradamento aggiornata ai vicini e, quando riceve informazioni dai vicini, le confronta con quelle già presenti nella propria tabella. Se trova un percorso più economico verso una destinazione attraverso un vicino, aggiorna la propria tabella di instradamento.

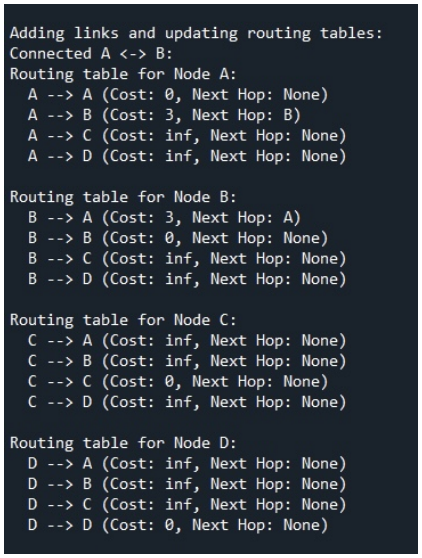

Figura 3: Connessione dei nodi A-B

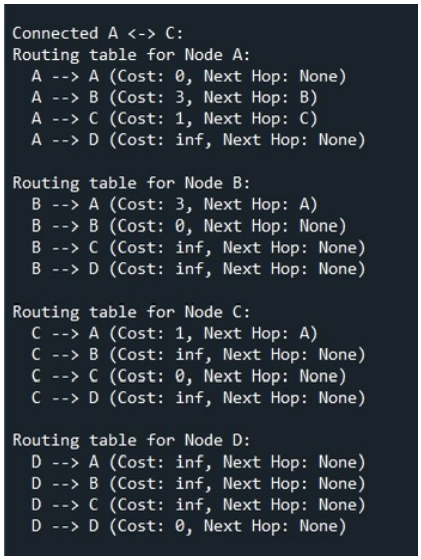

Figura 4: Connessione dei nodi A-C

3 Analisi del codice

La struttura del programma è suddivisa in due classi principali: Node e Network. Tale separazione delle classi facilita la gestione del codice. La classe Network gestisce gli aspetti globali della rete, mentre Node si occupa del comportamento locale di ogni nodo.

3.1 Classe Node

Rappresenta un nodo della rete e gestisce la sua tabella di routing, che contiene i costi e i prossimi hop verso ciascun nodo.

In particolare, i metodi di maggiore importanza sono:

- il metodo `add neighbor()`, aggiunge un nodo vicino alla lista dei vicini del nodo corrente, inserendo anche il corrispettivo costo.
- il metodo `update routing table()`, aggiorna la tabella di routing del nodo basandosi sulle informazioni ricevute dai suoi vicini. Per ogni vicino, il nodo verifica i percorsi verso tutte le destinazioni nella sua tabela di routing del vicino. Se trova un percorso più conveniente aggiorna la propria tabella.

3.2 Classe Network

Rappresenta l'intera rete, collega i nodi tra loro e simula il processo di scambio di informazioni di routing tra di essi. Questa classe fornisce metodi per aggiun-

gere nodi, creare delle connessioni tra i nodi e simulare del comportamento del protocollo Distance Vector Routing.

- `add node()`, questo metodo permette di aggiugere un nuovo nodo alla rete
- `add link()`, crea un collegamento bidirezionale tra due nodi con un costo specificato
- `simulate routing()`, ha lo scopo di modellare il comportamento iterativo e distribuito dell'algoritmo. A ogni iterazione, ogni nodo aggiorna la propria tabella di routing basandosi sulle informazioni ricevute dai vicini. Se durante un'iterazione nessun nodo effettua aggiornamenti in nessun nodo, la simulazione si interrompe anticipatamente.

4 Esecuzione dell'applicazione

Per eseguire l'applicazione, è necessario avere Python installato sul proprio sistema. Una volta verificata l'installazione, posizionarsi nella directory in cui si trova il file dello script e avviarlo utilizzando il seguente comando:

```
python ./dv_routing_simulator.py
```

L'esecuzione del programma mostrerà sul terminale le seguenti informazioni: le tabelle di routing iniziali per ogni nodo, gli aggiornamenti delle tabelle di routing dopo ogni iterazione.

5 Conclusioni

Durante lo sviluppo del progetto, sono stati affrontati problemi come l'inizializzazione corretta delle tabelle di instradamento, la convergenza del protocollo e la gestione dei possibili cicli di instradamento.

Attualmente, il sistema implementa una versione semplificata del protocollo Distance Vector Routing, ma ci sono molte possibilità di miglioramento. Tra queste, si potrebbero includere la gestione dinamica dei collegamenti e una visualizzazione grafica interattiva della rete per facilitarne l'analisi e la comprensione.