Simulazione del Protocollo Distance Vector Routing

Martin Tomassi
0001077737

20 novembre 2024

Sommario

In questa relazione viene presentata la progettazione e l'implementazione di uno script Python per la simulazione del protocollo Distance Vector Routing (DVR). L'obiettivo principale è stato quello di creare un ambiente di simulazione che permettesse di analizzare il comportamento del protocollo, con particolare attenzione all'implementazione dello Split Horizon per prevenire loop di routing. Vengono discussi in dettaglio l'architettura del sistema, le funzionalità implementate e i risultati ottenuti attraverso test approfonditi.

Indice

1 Introduzione

L'obiettivo di questo progetto è stato lo sviluppo di uno script Python che simuli il funzionamento del protocollo *Distance Vector Routing* (DVR), consentendo l'analisi dettagliata del comportamento del protocollo in diverse condizioni di rete e l'osservazione degli effetti di meccanismi come lo Split Horizon.

2 Obiettivi del Progetto

Il progetto si è prefissato i seguenti obiettivi specifici:

- **Implementazione del Protocollo DVR**: Realizzare una simulazione accurata del protocollo Distance Vector Routing utilizzando il linguaggio di programmazione Python.
- **Gestione Dinamica delle Tabelle di Routing**: Consentire l'aggiornamento dinamico delle tabelle di routing per ogni nodo.
- **Calcolo dei Percorsi Ottimali**: Utilizzare l'algoritmo di Bellman-Ford per determinare le rotte più brevi tra i nodi della rete.
- **Interfaccia Utente Interattiva**: Fornire un'interfaccia a linea di comando che permetta all'utente di interagire con la simulazione, monitorare lo stato della rete ed eseguire operazioni specifiche.
- **Implementazione dello Split Horizon**: Integrare meccanismi avanzati come lo Split Horizon per prevenire problemi di routing quali i loop infiniti.
- **Robustezza e Gestione degli Errori**: Assicurare la stabilità dello script attraverso una gestione efficace degli errori e una validazione accurata degli input.

3 Architettura del Sistema

3.1 Struttura Generale

Lo script è strutturato in moduli distinti che gestiscono le diverse componenti del sistema:

- **Parsing degli Argomenti**: Utilizza il modulo `argparse` per gestire gli argomenti passati da linea di comando, consentendo all'utente di specificare il file CSV contenente la topologia della rete.
- **Gestione del Grafo**: Si avvale della libreria `NetworkX` per rappresentare la rete come un grafo non orientato ponderato, dove i nodi corrispondono ai router e gli archi rappresentano i collegamenti con i relativi costi.
- **Tabelle di Routing**: Ogni nodo possiede una istanza della classe `RoutingTable`, che mantiene le informazioni sulle rotte verso gli altri nodi, includendo il costo e il prossimo salto (*next hop*).
- **Algoritmo di Bellman-Ford**: L'algoritmo è implementato nella funzione `bellman_ford_iteration`, che aggiorna le tabelle di routing basandosi sulle informazioni ricevute dai nodi adiacenti.
- **Split Horizon**: La funzionalità dello Split Horizon è integrata nel sistema e può essere attivata o disattivata dall'utente durante l'esecuzione, influenzando il comportamento dell'algoritmo di aggiornamento.
- **Interfaccia Utente**: Un'interfaccia a linea di comando permette all'utente di interagire con la simulazione, eseguendo comandi per visualizzare le tabelle di routing, calcolare percorsi, eseguire iterazioni dell'algoritmo e modificare le impostazioni.
- **Gestione degli Errori**: Lo script include meccanismi per la validazione degli input e gestisce le eccezioni comuni, garantendo una maggiore robustezza.

3.2 Implementazione delle Tabelle di Routing

La classe `RoutingTable` è responsabile della gestione delle tabelle di routing per ogni nodo. Ogni istanza della classe contiene:

- **Nodo Destinazione**: Il nodo verso cui la rotta è diretta.
- **Costo**: Il costo stimato per raggiungere il nodo di destinazione.
- **Prossimo Nodo (*Next Hop*)**: Il nodo adiacente attraverso cui inoltrare i pacchetti per raggiungere la destinazione.

L'inizializzazione delle tabelle di routing assegna un costo zero verso sé stessi, i costi diretti verso i nodi adiacenti e un costo infinito verso gli altri nodi.

3.3 Algoritmo di Bellman-Ford

L'algoritmo di Bellman-Ford è implementato per aggiornare iterativamente le tabelle di routing:

1. Ogni nodo condivide la propria tabella di routing con i nodi adiacenti.
2. Per ogni destinazione, i nodi calcolano il costo totale attraverso ogni vicino.
3. Se viene trovato un percorso con costo inferiore, la tabella di routing viene aggiornata con il nuovo costo e il prossimo salto.

La funzione `bellman_ford_iteration` esegue un'iterazione completa dell'algoritmo per tutti i nodi.

3.4 Implementazione dello Split Horizon

Lo Split Horizon è integrato nella funzione `bellman_ford_iteration` e può essere attivato tramite una variabile globale `SPLIT_HORIZON`. Quando attivo, il meccanismo evita di considerare percorsi che ritornerebbero al nodo mittente, prevenendo così possibili loop di routing.

3.5 Gestione degli Errori e Validazione

Lo script effettua controlli su:

- **Archi con Peso Negativo**: Interrompe l'esecuzione se vengono rilevati, poiché non supportati dall'algoritmo.
- **Input dell'Utente**: Valida i comandi e i parametri inseriti, assicurandosi che i nodi esistano e che gli argomenti siano corretti.
- **Eccezioni**: Gestisce eccezioni durante la lettura del file CSV e l'elaborazione dei dati.

4 Funzionalità Implementate

4.1 Gestione della Topologia della Rete

Il file CSV utilizzato come input definisce i nodi e gli archi della rete. Ad esempio:

Lo script legge il file e costruisce il grafo corrispondente utilizzando `NetworkX`.

```
# Nodi
R1,R2,R3,R4,R5,R6

# Archi con Peso
R1,R2,1
R1,R3,6
R2,R3,2
R3,R4,3
R4,R5,2
R2,R4,6

```

Listing 1: Esempio di file CSV per la rete

4.2 Interfaccia Utente Interattiva

L'interfaccia a linea di comando offre i seguenti comandi:

- **help**: Visualizza il menu dei comandi disponibili.
- **show-table <nodo>/all**: Mostra la tabella di routing per il nodo specificato o per tutti i nodi.
- **shortest-path <nodo1> <nodo2>**: Calcola e visualizza il percorso più breve tra due nodi.
- **update-tables <n>/stable**: Esegue un numero specificato di iterazioni dell'algoritmo o fino alla stabilità.
- **toggle-split-horizon**: Attiva o disattiva lo Split Horizon.
- **exit**: Termina la simulazione.

4.3 Esecuzione dello Script

Per eseguire lo script:

```
python3 routing_simulation.py -f network.csv

```

Listing 2: Esecuzione dello script

Una volta avviato, l'utente può interagire utilizzando i comandi descritti.

5 Risultati e Test

5.1 Test di Funzionalità

Sono stati condotti test per verificare la corretta inizializzazione delle tabelle di routing e l'aggiornamento attraverso l'algoritmo di Bellman-Ford.

5.2 Verifica della Convergenza

Eseguendo l'algoritmo fino alla stabilità, si è osservato che le tabelle di routing convergono correttamente, riflettendo i percorsi ottimali tra i nodi.

5.3 Effetto dello Split Horizon

Attivando lo Split Horizon, i test hanno mostrato una riduzione dei tempi di convergenza e l'eliminazione di possibili loop di routing, confermando l'efficacia del meccanismo.

5.4 Gestione degli Errori

Inserendo archi con peso negativo e comandi non validi, lo script ha risposto adeguatamente, segnalando gli errori e prevenendo comportamenti indesiderati.

6 Conclusioni

Il progetto ha permesso di implementare con successo una simulazione del protocollo Distance Vector Routing, fornendo uno strumento utile per l'analisi dei meccanismi di routing. L'integrazione dello Split Horizon e una gestione robusta degli errori hanno migliorato l'affidabilità e l'efficacia della simulazione.