Simulazione del Protocollo di Routing Distance Vector

Daniele Merighi

11 dicembre 2024

Introduzione

La presente relazione descrive la simulazione del protocollo **Distance Vector Routing** realizzata in Python. Il Distance Vector Routing è un algoritmo distribuito per il calcolo dei percorsi più brevi all'interno di una rete, basato sulla versione dinamica e distribuita dell'algoritmo di **Bellman-Ford**, proposto da Ford-Fulkerson. Questo tipo di protocollo viene impiegato per la gestione delle tabelle di routing nei router, permettendo loro di determinare i percorsi ottimali verso tutte le destinazioni all'interno di una rete.

Obiettivi della Simulazione

La simulazione ha lo scopo di:

- Dimostrare il funzionamento del protocollo Distance Vector Routing in un contesto controllato.
- Analizzare il processo di calcolo dei percorsi più brevi tramite tabelle di routing locali.
- Mostrare la convergenza del protocollo attraverso lo scambio di informazioni tra router.
- Esplorare i limiti del protocollo, come la *convergenza lenta* e il problema del *conteggio all'infinito*.

Descrizione del Protocollo

Il protocollo Distance Vector Routing si basa sui seguenti principi fondamentali:

1. Ogni nodo conosce inizialmente solo i suoi vicini diretti e la distanza verso di essi.

2. Ad ogni iterazione, ogni nodo invia ai propri vicini un vettore che contiene la stima delle distanze verso tutti i nodi conosciuti della rete.

3. Ogni nodo aggiorna la propria tabella di routing basandosi sulle informazioni ricevute dai vicini, calcolando i percorsi più brevi tramite l'equazione di **Bellman-Ford**:

$$d(i, j) = \min_k \{c(i, k) + d(k, j)\}$$

   dove $d(i, j)$ è la distanza stimata tra i nodi i e j, $c(i, k)$ è il costo del collegamento tra i e k, e $d(k, j)$ è la distanza conosciuta tra k e j.

Vantaggi

- Semplicità di implementazione.
- Basso consumo di risorse computazionali e di memoria.
- Adatto a reti di piccole e medie dimensioni.

Problemi

- **Convergenza lenta**: il tempo necessario affinché tutti i nodi conoscano i percorsi ottimali può essere lungo, specialmente in reti grandi o dinamiche.
- **Problema del conteggio all'infinito**: in caso di topologie instabili, i nodi potrebbero calcolare distanze errate crescenti all'infinito. Questo problema può essere mitigato con tecniche come *split horizon* o *hold-down timers*.

Struttura della Simulazione

La simulazione è implementata in Python e utilizza le librerie `networkx` e `matplotlib` per la rappresentazione grafica della rete.

Componenti Principali

- **Classe Router**: rappresenta un nodo della rete, con metodi per aggiornare la tabella di routing e gestire i vicini.
- **Classe Network**: gestisce la topologia della rete e simula lo scambio di informazioni tra i router.
- **Visualizzazione Grafica**: mostra la topologia della rete e i costi dei collegamenti utilizzando grafi.

Esecuzione

Il programma simula una rete composta da sei router (`A, B, C, D, E, F`) con collegamenti e costi prestabiliti. Durante la simulazione:

1. Ogni router scambia informazioni di routing con i propri vicini.
2. Le tabelle di routing vengono aggiornate iterativamente fino alla convergenza.
3. La topologia della rete e le tabelle di routing vengono visualizzate ad ogni iterazione.

Risultati

La simulazione mostra:

- La convergenza del protocollo, con le tabelle di routing che raggiungono uno stato stabile.
- L'efficacia dell'algoritmo di Bellman-Ford nel calcolo dei percorsi ottimali.
- La topologia della rete visualizzata come grafo non orientato.

Conclusioni

Il progetto ha dimostrato il funzionamento e i limiti del protocollo Distance Vector Routing, fornendo una base pratica per comprendere concetti teorici come il routing distribuito e il calcolo dei percorsi minimi. Nonostante i vantaggi di semplicità ed efficienza, il protocollo presenta criticità come la convergenza lenta e il problema del conteggio all'infinito, che ne limitano l'applicabilità a reti di dimensioni maggiori o altamente dinamiche.

Codice Sorgente

Il codice completo è disponibile nel file `distance_vector_routing.py` allegato al progetto. Per ulteriori dettagli sull'implementazione, consultare il file `README.md` incluso nel repository.