Progetto Python: Simulazione di Protocollo di Routing

Nome: Sebastiano Lucarelli

Matricola: 0001090126

Descrizione

Creare uno script Python che simuli un protocollo di routing semplice, come il Distance Vector Routing. Gli studenti implementeranno gli aggiornamenti di routing tra i nodi, con il calcolo delle rotte più brevi.

Obiettivi

Implementare la logica di aggiornamento delle rotte, gestione delle tabelle di routing e calcolo delle distanze tra nodi.

Consegne Richieste

Codice Python ben documentato, output delle tabelle di routing per ogni nodo e relazione finale che spieghi il funzionamento dello script.

Introduzione

Il progetto realizzato utilizzando il linguaggio Python, si pone l'obiettivo di simulare un protocollo di Distance Vector Routing, andando a ricreare un ambiente al cui interno i singoli nodi cercano di calcolare in modo autonomo il percorso più breve che li lega ai nodi vicini.

Nei protocolli DVR ogni nodo è in grado di mantenere aggiornate le proprie tabelle di routing sfruttando lo scambio periodico di informazioni con i propri vicini.

Questa simulazione rappresenta un esempio pratico dei principi alla base del routing distribuito.

Il Distance Vector Routing si basa sull'algoritmo di Bellman-Ford, che permette a ciascun nodo di calcolare il percorso più breve che lo collega ad ogni altro nodo della rete, minimizzando i costi di trasmissione. Durante la simulazione, ogni nodo:

- Mantiene una propria tabella di instradamento, inizialmente contenente solo informazioni sui suoi vicini diretti.
- Aggiorna periodicamente la tabella di instradamento in base ai vettori delle distanze ricevuti dai nodi vicini.
- Converge gradualmente verso uno stato stabile in cui tutte le tabelle contengono i percorsi ottimali.

Il processo continua fino momento in cui le tabelle di routing non subiscono più modifiche e ogni nodo conosce il percorso migliore per raggiungere ogni altro nodo della rete.

Descrizione del Funzionamento

Il Distance Vector Routing (DVR) si basa sulla cooperazione tra nodi di una rete che aggiornano le loro tabelle scambiandosi costantemente dati ed informazioni. Ogni nodo (come accennato) mantiene una tabella contenente le distanze verso tutti gli altri nodi che, in un primo momento, sarà popolata solamente con i vicini diretti. Il protocollo da qui poi prosegue a seguire 3 passi principali:

1. Scambio di informazioni: ogni nodo invia periodicamente il proprio "vettore delle distanze" ai vicini. Questo contiene i costi per raggiungere tutti i nodi conosciuti,
2. Aggiornamento delle tabelle: al ricevimento dei vettori dai vicini, il nodo aggiorna la propria tabella confrontando i percorsi esistenti con quelli ricevuti, calcolando nuove distanze sulla base della somma dei costi;
3. Convergenza: il processo continua iterativamente finché le tabelle non smettono di cambiare, indicando che ogni nodo ha calcolato il percorso più efficiente verso tutti gli altri.

Struttura del Codice

Il codice che implementa il **Distance Vector Routing** è organizzato in quattro parti principali, ciascuna responsabile di un aspetto specifico del protocollo:

1. **Inizializzazione**:

   Configura la rete caricando la topologia iniziale e costruisce le tabelle di instradamento per ciascun nodo. In questo momento ogni tabella contiene i nodi vicini con i relativi costi iniziali e un valore "infinito" per i nodi non direttamente raggiungibili.

2. **Scambio di Dati**:

   I nodi trasmettono periodicamente i propri vettori ai vicini tramite funzioni dedicate alla comunicazione. Questi messaggi includono informazioni sui costi aggiornati per raggiungere tutti i nodi conosciuti.

3. **Aggiornamento delle Tabelle**:

   Al ricevimento di un messaggio da un vicino, il codice verifica se il nuovo percorso riduce il costo per raggiungere un nodo specifico. Le modifiche vengono salvate e, se necessario, vengono notificati i vicini per propagare l'aggiornamento.

4. **Gestione degli Errori e Ottimizzazioni**:

   Il codice prevede meccanismi per evitare cicli (come split-horizon) e per gestire variazioni nella rete.

Analisi Delle Tabelle e Dei Risultati

Tabelle di routing iniziali

Nodo A: [A: 0 (via -), B: 2 (via B), C: 4 (via C), E: 1 (via E)]

Nodo B: [A: 2 (via A), B: 0 (via -), C: 1 (via C), D: 4 (via D), E: 7 (via E)]

Nodo C: [A: 4 (via A), B: 1 (via B), C: 0 (via -)]

Nodo D: [B: 4 (via B), D: 0 (via -), E: 2 (via E)]

Nodo E: [A: 1 (via A), B: 7 (via B), D: 2 (via D), E: 0 (via -)]

Queste sono le tabelle iniziali, nonché il primo output restituito dal programma.

Iterazione 1:

Stato dopo l'iterazione 1

Nodo A: [A: 0 (via -), B: 2 (via B), C: 3 (via B), D: 3 (via B), E: 1 (via E)]

Nodo B: [A: 2 (via A), B: 0 (via -), C: 1 (via C), D: 4 (via D), E: 3 (via A)]

Nodo C: [A: 3 (via A), B: 1 (via B), C: 0 (via -), D: 5 (via A), E: 4 (via A)]

Nodo D: [A: 3 (via B), B: 4 (via B), C: 5 (via B), D: 0 (via -), E: 2 (via E)]

Nodo E: [A: 1 (via A), B: 3 (via A), C: 4 (via A), D: 2 (via D), E: 0 (via -)]

Successivamente le tabella effettuano una prima iterazione per andare ad aggiornare le loro tabelle.

Iterazione 2:

Stato dopo l'iterazione 2

Nodo A: [A: 0 (via -), B: 2 (via B), C: 3 (via B), D: 3 (via B), E: 1 (via E)]

Nodo B: [A: 2 (via A), B: 0 (via -), C: 1 (via C), D: 4 (via D), E: 3 (via A)]

Nodo C: [A: 3 (via A), B: 1 (via B), C: 0 (via -), D: 5 (via A), E: 4 (via A)]

Nodo D: [A: 3 (via B), B: 4 (via B), C: 5 (via B), D: 0 (via -), E: 2 (via E)]

Nodo E: [A: 1 (via A), B: 3 (via A), C: 4 (via A), D: 2 (via D), E: 0 (via -)]

Le tabelle effettuano un ulteriore scambio di informazioni così da aggiornare ancora una volta le loro tabelle di routing.

Convergenza raggiunta - Tabelle finali

Nodo A: [A: 0 (via -), B: 2 (via B), C: 3 (via B), D: 3 (via B), E: 1 (via E)]

Nodo B: [A: 2 (via A), B: 0 (via -), C: 1 (via C), D: 4 (via D), E: 3 (via A)]

Nodo C: [A: 3 (via A), B: 1 (via B), C: 0 (via -), D: 5 (via A), E: 4 (via A)]

Nodo D: [A: 3 (via B), B: 4 (via B), C: 5 (via B), D: 0 (via -), E: 2 (via E)]

Nodo E: [A: 1 (via A), B: 3 (via A), C: 4 (via A), D: 2 (via D), E: 0 (via -)]

Alla terza iterazione raggiungiamo la convergenza ed otteniamo le tabelle finali contenenti le vie più brevi.

Problemi Riscontrati

Durante lo sviluppo del progetto sono emersi vari problemi e sfide, ogni anomalia è stata analizzata isolando il componente coinvolto e testandolo in modo indipendente, utilizzando scenari personalizzati per riprodurre il problema. Le soluzioni sono state validate attraverso test realizzati utilizzando grafi differenti, garantendo una risoluzione generale. Di seguito sono descritti i principali problemi affrontati e le soluzioni adottate:

- **Gestione dei collegamenti instabili tra i nodi:**

Durante la simulazione, alcuni link tra i nodi si comportavano in modo instabile, simulando errori di trasmissione o temporanei. Questo causava errori nel calcolo delle distanze minime, portando a valori incoerenti nelle tabelle di routing. Per risolvere il problema è stata introdotta una gestione delle eccezioni per rilevare e ignorare i link instabili durante la fase di aggiornamento delle tabelle. Inoltre, è stato aggiunto un sistema di temporizzazione per riabilitare automaticamente i link instabili dopo un intervallo predefinito.

- **Incoerenza tra le tabelle di routing:**

  Durante le prime iterazioni, alcuni nodi aggiornavano correttamente le loro tabelle, ma non riuscivano a propagare le informazioni corrette ai vicini, portando a disallineamenti tra le tabelle. La funzione di propagazione è stata analizzata e riscritta per includere verifiche di coerenza prima di inviare le informazioni ai vicini. È stata aggiunta una routine di controllo alla fine di ogni iterazione per verificare l'allineamento tra le tabelle dei nodi.

- **Errore nella propagazione delle distanze infinite:**
  In alcune topologie, i nodi isolati non venivano considerati correttamente, portando alla mancata registrazione delle distanze infinite nelle tabelle di routing. La funzione di costruzione dei vettori delle distanze è stata modificata per includere esplicitamente i nodi isolati, con distanze impostate a infinito. È stata aggiunta una verifica al termine di ogni iterazione per garantire che tutti i nodi siano presenti nelle tabelle.

- **Difficoltà nella visualizzazione dell'evoluzione delle tabelle:**

  L'output prodotto inizialmente era poco leggibile, rendendo difficile seguire l’evoluzione delle tabelle di routing nel tempo. È stato aggiunto un metodo __str__ per fornire una rappresentazione chiara delle tabelle di routing e una funzione salva_log per registrare lo stato delle tabelle in un file esterno. Questi strumenti hanno semplificato il monitoraggio delle modifiche e il debug del protocollo.