Nome: Mihai
Cognome: Mazuru
Matricola: 0001080591

Relazione progetto

Introduzione

Il codice proposto simula un protocollo di routing basato sull'algoritmo di Bellman-Ford, noto come Distance Vector Routing Protocol. Questo tipo di protocollo è ampiamente utilizzato nelle reti per determinare i percorsi più brevi tra i nodi, sfruttando tabelle di routing che si aggiornano iterativamente scambiando informazioni con i nodi vicini.

Questo approccio richiede poche risorse computazionali, rendendolo adatto a reti con limitate capacità di elaborazione. Tuttavia, possono verificarsi problemi di convergenza lenta, dovuti a una partenza lenta (cold start), e problemi di stabilità, come il conteggio all'infinito.
I protocolli di routing dinamico, come il distance vector, continuano a essere utilizzati in specifici contesti, con il RIP che ha subito diverse revisioni per migliorarne l'affidabilità e l'efficienza. Tuttavia, permangono alcune limitazioni intrinseche che possono manifestarsi in particolari casi limite, specialmente in reti più grandi o con topologie complesse.

Lo scopo principale del progetto è simulare il comportamento del protocollo, evidenziandone le iterazioni e le modifiche alle tabelle di routing durante il processo di convergenza.

Codice

Il programma è strutturato in diverse funzioni che coprono tutte le fasi principali del protocollo:

- "**initialize_network**" crea una rappresentazione della rete come un dizionario, dove ogni nodo è associato ai suoi vicini diretti insieme ai costi dei collegamenti. Ogni nodo viene mappato su un altro dizionario che contiene i vicini e i rispettivi costi.

```python
def initialize_network():
    network = {
        "A": {"C": 4, "B": 2},
        "B": {"A": 2, "C": 7, "E": 1},
        "C": {"B": 7, "A": 4},
        "D": {"E": 4},
        "E": {"B": 1, "D": 4},
    }
    return network
```

- "**initialize_routing_tables"** inizializza le tabelle di routing per tutti i nodi della rete. Ogni nodo ha una tabella che mappa ogni destinazione alla sua distanza e al prossimo nodo (hop) da raggiungere per arrivarci. Le tabelle vengono inizializzate con la distanza 0 per il nodo stesso e le distanze dai vicini diretti.

```python
def initialize_routing_tables(network):
    routing_tables = {}

    for node in network:
        # ogni nodo conosce sé stesso con distanza 0
        routing_tables[node] = {node: (0, node)}

        for neighbor, cost in network[node].items():
            # inizializza i vicini diretti con i rispettivi costi
            routing_tables[node][neighbor] = (cost, neighbor)

    return routing_tables
```

- "**bellman_ford**" esegue un ciclo di aggiornamento sulla tabella di routing di un nodo utilizzando l'algoritmo di Bellman-Ford. Confronta le distanze correnti con quelle calcolate passando attraverso i vicini e aggiorna la tabella se viene trovato un percorso più breve.

```python
def bellman_ford(node, neighbors, routing_tables):
    updated = False

    for neighbor, cost_to_neighbor in neighbors.items():
        # tabella di routing del vicino
        neighbor_table = routing_tables[neighbor]

        for destination, (neighbor_distance, neighbor_to_cross) in neighbor_table.items():
            # calcola la distanza passando attraverso il vicino
            new_distance = cost_to_neighbor + neighbor_distance

            if (destination not in routing_tables[node] or new_distance < routing_tables[node][destination][0]):
                # mostra il cambiamento di distanza
                old_value = routing_tables[node].get(destination, (float('inf'), None))
                print(f"Updating {node} → {destination}: {old_value[0]} → {new_distance}")
                routing_tables[node][destination] = (new_distance, neighbor)

                # indica che è avvenuto un aggiornamento
                updated = True

    return updated
```

- "**print_routing_tables**" stampa in modo leggibile le tabelle di routing per ogni nodo della rete. Ogni nodo è associato a una lista delle destinazioni, distanze e il prossimo nodo (hop) verso quella destinazione.

```python
def print_routing_tables(routing_tables):
    print("\nRouting Tables:")

    for node, table in routing_tables.items():
        print(f"Node {node}:")

        for destination, (distance, next_hop) in table.items():
            print(f"  {destination}: {distance} via {next_hop}")

        print()
```

- “**main**” avvia il processo di simulazione del protocollo di routing basato sull'algoritmo di Bellman-Ford. Inizializza le tabelle di routing, aggiorna le tabelle iterativamente e stampa le modifiche fino al raggiungimento della convergenza.

```
def main(network):
    routing_tables = initialize_routing_tables(network)

    print("Initial Routing Tables:")
    print_routing_tables(routing_tables)

    # tiene traccia se avviene un cambiamento
    updated = True
    iteration = 0

    while updated:
        print(f"\nIteration {iteration + 1}:")
        updated = False

        for node, neighbors in network.items():

            if bellman_ford(node, neighbors, routing_tables):
                # convergenza non ancora raggiunta se è stato verficato un cambiamento
                updated = True

        print_routing_tables(routing_tables)
        iteration += 1

    # mostra le tabelle finali
    print("\nFinal Routing Tables:")
    print_routing_tables(routing_tables)
    print(f"Convergence after {iteration} iterations")
```

Output

In seguito, una rappresentazione delle tabelle di routing di ogni nodo una volta che l'algoritmo ha raggiunto la convergenza. Le tabelle di seguito fanno riferimento all'esempio di rete proposto nel codice di questo progetto.

```
Final Routing Tables:

Routing Tables:
Node A:
  A: 0 via A
  C: 4 via C
  B: 2 via B
  E: 3 via B
  D: 7 via B

Node B:
  B: 0 via B
  A: 2 via A
  C: 6 via A
  E: 1 via E
  D: 5 via E

Node C:
  C: 0 via C
  B: 6 via A
  A: 4 via A
  E: 7 via A
  D: 11 via A

Node D:
  D: 0 via D
  E: 4 via E
  B: 5 via E
  A: 7 via E
  C: 11 via E

Node E:
  E: 0 via E
  B: 1 via B
  D: 4 via D
  A: 3 via B
  C: 7 via B

Convergence after 3 iterations
```

Diagnosi

Durante l'implementazione, una delle principali difficoltà è stata la gestione delle iterazioni per garantire che il processo terminasse correttamente. È stato necessario introdurre un flag "updated" per monitorare se si verificavano cambiamenti nelle tabelle, evitando cicli infiniti. Inoltre, la

rappresentazione del grafo come un dizionario, pur essendo semplice, potrebbe diventare inefficiente per reti di grandi dimensioni.
Sempre in reti di grandi dimensioni è possibile notare un modesto calo di efficienza nell'algoritmo di Bellman-Ford. Questo è dovuto alla relazione stretta tra il costo computazionale di questo algoritmo e il numero di nodi e collegamenti presenti della rete (O(V*E)).

La rappresentazione tramite dizionario non è stata la prima scelta per la realizzazione di questo progetto. Come primo tentativo è stata utilizzata una matrice di adiacenza. Dopo diverse prove con la suddetta struttura, si è optato per l'utilizzazione una struttura dati diversa, che rendeva il codice più leggibile.

Un'altra sfida è stata garantire che ogni aggiornamento fosse chiaramente tracciabile, migliorando la visualizzazione con messaggi di log che evidenziano i cambiamenti di distanza.

Conclusione

La soluzione proposta con questo progetto è una molto semplice e mirata a reti di piccole dimensioni, però rappresenta in modo esaustivo la logica dietro il protocollo del Distance Vector.
Il programma potrebbe essere esteso per gestire situazioni reali come la rilevazione di loop o l'integrazione di eventi dinamici, come l'aggiunta o la rimozione di collegamenti.