Programmazione di Reti
Distance vector routing

Palazzini Luca

November 28, 2024

1 Funzionamento del protocollo

Lo script python simula il protocollo Distance Vector Routing, overo un tipo di protocollo di routing basato sull'algoritmo di Bellman-Ford. I nodi all'inizio dell'esecuzione conoscono solo i propri vicini, ovvero i nodi connessi a loro stessi. Tramite l'ulitizzo dell'algoritmo tutti i nodi dovranno sapere il prossimo passo per arrivare a qualsiasi altro nodo nel miglior modo possibile.

1.1 Specifiche dello script

Lo script inizialmente legge da un file la configurazione di rete per creare dei router, contenenti solo i collegamenti ad i vicini e la propria routing table, questo per avvicinarsi a come un nodo potrebbe funzionare nel mondo reale.

```python
class Router:
    id: str
    # The links between this router and another
    links: dict[frozenset[str], int]
    # A routing table that contains for each destination
    # the next hop and the cost to reach that node
    routing_table: OrderedDict[str, tuple[str, int]]
    dirty_table: bool

    def __init__(self, identificator: str) -> None:
        self.id = identificator
        self.routing_table = OrderedDict()
        self.links = dict()
        self.dirty_table = True
        # Adds itself to the table at cost 0
        self.routing_table[self.id] = (self.id, 0)
        self.links[frozenset([self.id, self.id])] = 0

    def add_link(self, link: tuple[frozenset[str], int]) -> None:
        self.links[link[0]] = link[1]
        for r in link[0]:
            if (r == self.id):
                continue
            self.routing_table[r] = (r, link[1])
```

Listing 1: Classe del router

Inizializzata la rete, il programma inizierá a scambiare le routing table di tutti i nodi con tutti i vicini, salvando ad ogni step le nuove routing table per poter vedere come la rete si evolve nel tempo. Come si puó vedere dal codice, l'algoritmo che si svolge ha tre fasi:

- Tutte le tabelle di routing finiscono in "rete"
- Tutti i router ottengono le tabelle di routing mandate dai vicini dalla "rete"
- Vengono salvate tutte le tabelle di routing del tempo attuale t

```python
def run_distance_vector(routers: dict[str, Router], t_max: int) -> dict[int, OrderedDict[str, OrderedDict[str, tuple[str, int]]]]:
    tables: dict[int, OrderedDict[str, OrderedDict[str, tuple[str, int]]]] = dict()
    # Initialize T=0
    tables[0] = OrderedDict()
    for id, router in routers.items():
        tables[0][id] = router.get_frozen_table()
    # Set exit variables
    done: bool = False
    t: int = 1
    # Run the algorithm
    while not done and t < t_max:
        # Create the new container for the tables at t
        tables[t] = OrderedDict()
        # Send all messages (routing tables) to the network
        network: dict[tuple[str, str], OrderedDict[str, OrderedDict[str, tuple[str, int]]]] = dict()
        for id, router in routers.items():
            # Skip routers that have not been updated last t
            if not router.dirty_table:
                continue
            # Check all the other connected routers
            neighbors: set[str] = router.get_neighbors()
            # Send the current routing table to all other neighbors
            for other_id in neighbors:
                network[(id, other_id)] = router.get_frozen_table()
            # Set the router's dirty flag to false
            router.dirty_table = False
        # Recieve all routing tables from the network
        for (sender_id, receiver_id), table in network.items():
            routers[receiver_id].update_table(sender_id=sender_id, sender_table=table)
        # Save current routing tables to dict
        done = True
        for id, router in routers.items():
            tables[t][id] = router.get_frozen_table()
            if router.dirty_table:
                done = False
        t += 1
    # If early exit the last table will be same as previous
    if done == True:
        del tables[t - 1]
    return tables
```

Listing 2: Simulazione della rete

Una volta un router ottiene le routing table dei suoi vicini, potrá aggiornare la propria controllando il proprio costo per il nodo e quello del vicino (piú il costo del link su cui é il nodo), nel caso sia minore esso puó rimpiazzare il valore della tabella di routing per quel nodo.

```
1     def update_table(self, sender_id: str, sender_table: OrderedDict[
      str, tuple[str, int]]) -> None:
2         current_link_weight: int = self.links[frozenset([sender_id,
      self.id])]
3         for dest, connection in sender_table.items():
4             if dest not in self.routing_table or self.routing_table[
      dest][1] > connection[1] + current_link_weight:
5                 self.routing_table[dest] = (sender_id, connection[1] +
      current_link_weight)
6                 self.dirty_table = True
```

Listing 3: Funzione di aggiornamento delle tabelle

2 Utilizzo del codice

Il codice non richiede alcuna libreria o applicazione aggiuntiva oltre a quelle già presenti nella repository, eccetto se lo si prova ad eseguire in ambiente windows (bisognerá scaricare il modulo window-curses).
Nonostante ciò queste sono le librerie utilizzate (native di python3) nel progetto:

- Python 3.x
- Curses (per la `TUI` (terminal UI))

Per modificare la configurazione della rete si puó creare un file di testo contenente i router e le connessioni fra essi come sotto riportato:

- Nella prima sezione del file, scrivere tutti i nodi che compongono la rete, come:
  `r Router1`
  `r Router2`
  etc...
- Nella seconda sezione del file, scrivere tutte le connessioni fra i nodi, con il costo delle collezioni, come:
  `l Router1 Router2 5`
  `l Router2 Router3 7`
  etc...

Per eseguire il codice correttamente basterá eseguire i seguenti passaggi:

1. Eseguire lo script con il comando seguente

```
1 ./main.py <configurazione_rete.txt>
2 # OR
3 python main.py <configurazione_rete.txt>
```

Listing 4: Esecuzione dello script

2. Se il programma ritorna con un errore di una libreria non trovata, l'output sará trovato in "output.json", altrimenti si puó utilizzare l'interfaccia `TUI`

3. Utilizzare il tasto `h` per controllare i comandi sotto riportati
    - `A`/`D` o `Freccia Destra`/`Sinistra` per cambiare la pagina, ovvero il valore del tempo
    - `W`/`S` o `Freccia Su`/`Giú` per muovere la lista delle tabelle di routing su o giú
    - `Q`/`ESC` per uscire dall'applicazione

3 Considerazioni aggiuntive

- Il programma non é necessariamente uguale a ció che succede con nodi reali, chiaramente questo script in python é una simulazione non realistica.