Netztechnik Labor Dokumentation

Tim Porger

Mat.Nr: 5437590

Kurs: Inf23B

15.5.2025

Inhaltsverzeichnis

1. Projektbeschreibung

Dieses Projekt simuliert die Funktionsweise des Spanning Tree Protocols (STP) in einem Netzwerk, um einen minimalen Spannbaum über Netzwerk-Switches (Knoten) zu erstellen. Der minimale Spannbaum dient dazu Broadcast-Storms zu verhindern, die durch Netzwerkschleifen entstehen können. Basierend auf einem verteilten Algorithmus, bei dem jeder simulierte Switch Informationen über die Netzwerktopologie mit seinen Nachbarn austauscht (analog zum Senden und Empfangen von Bridge Protocol Data Units - BPDUs), ermitteln die Knoten schrittweise den optimalen, schleifenfreien Baum (Spanning Tree) vom Root-Knoten zu allen anderen Knoten. Das Projekt liefert als Ergebnis der Simulation den finalen Spanning Tree.

2. Aufbau

Das Projekt verfolgt einen Objekt-Orientierten Ansatz mit Klassen. Der Graph, mit welchem die Simulation durgeführt wird, wird Textuell über die Datei `input.txt` beschrieben. Der beschriebene Graph wird zuvor auf richtigkeit getestet, um sicher sein, dass die Simulation richtig abläuft.

2.1. util

Um den in `input.txt` beschriebenen Graph von in für die Simulation nutzbar zu machen, ist die Datei einzulesen. Der `util`-Ordner enthält dazu die Datei `graph_parser.py`. Durch die Klasse `GraphParser` wird eine gegebene Datei eingelesen und der Graph konfiguriert. Das bedeutet Switches und die Verbindungen dieser werden aufgebaut. Zusätzlich werden die Konstanten `MAX_IDENT = 20, MAX_ITEMS = 100, MAX_COST = 1000, MAX_SWITCH_ID = 10000` definiert um die eingelesene Konfiguration auf maximale Namenslänge de Switches, maximale Zeilen der Datei, maximale Kosten der Verbindungen und größe der Switch-IDs einzuschränken.

2.2. tests

Der Ordner `tests` enthält die Datei `graph_tests.py` in welcher der eingelesene Graph auf folgende Kriterien geprüft wird:

1. Sind alles Switch-IDs > 0?
2. Gibt es nur eine Root-ID?
3. Ist der Graph verbunden?
4. Gibt es Knoten die mit sich selbst verbunden sind?
5. Testweise eine Ausgabe der Switches und deren Verbindungen

2.3. models

Der `models`-Ordner enthält die Klassen, aufgeteilt in Dateien, welche die Objekte der Simulation darstellen.

2.3.1. graph.py

Die Klasse `Graph` stellt den eingelesenen Graphen dar und enthält alle Switches und verbindet die Switches miteinander.

Attribute

- `name:str` Name des Graphen, welcher in der `input.txt` beschrieben wurde
- `switches:Switch[]` Die eingelesenen Switches
- `switch_count:int` Die Anzahl der Switches
- `name_to_index:Map` Für die Übersetzung von Indexen zu Namen

Funktionen

- `get_index(name:string):int` Nimmt einen Switch Namen und gibt den entsprechenden Index zurück
- `append_switch(name:string, switch_id:int)` Erstellt Switch-Objekte und fügt sie dem `switches`-Array hinzu
- `add_link(from_name:string, to_name:string, const:int)` Verlinkt die Switch-Objekte

2.3.2. switch.py

Die Klasse `Switch` repräsentiert einen einzelnen Switch im Graphen. Sie speichert alle relevanten Informationen für das Spanning Tree Protokoll und verwaltet die Verbindungen (Links) zu anderen Switches.

Attribute

- `name:str` Name des Switches
- `switch_id:int` Eindeutige ID des Switches
- `links:int[]` Liste der Kosten zu allen anderen Switches (Index entspricht Switch-ID)
- `next_hop:int` ID des nächsten Switches auf dem Weg zum Root (initial eigene ID)
- `msg_cnt:int` Anzahl der gesendeten BPDUs
- `root_id:int` Aktuell angenommene Root-ID
- `distance_to_root:int` Aktuelle Distanz zum Root

- `best_neighbor_id:int` ID des besten Nachbarn für den Spanning Tree
- `received_bpdus:dict` Empfangene BPDUs von Nachbarn

Funktionen

- `add_link(cost:int)` Fügt einen neuen Link mit gegebenen Kosten zur `links`-Liste hinzu
- `receive_bpdu(neighbor_idx:int, root_id:int, distance_to_root:int)` Speichert empfangene BPDU-Informationen von einem Nachbarn

2.4. Simulation

Die Datei `simulation.py` Klasse `Simulation` steuert die Ausführung des Spanning Tree Protokolls auf dem gegebenen Graphen. Sie initialisiert die Switches, führt die STP-Iterationen durch und gibt am Ende den berechneten Spanning Tree aus. Zudem enthält `simulation.py` die main-Methode zur Ausführung des Programms.

Attribute

- `MAX_MSG_PER_SWITCH:int` Konstante welche definiert wie viele Nachrichten jeder Switch während der Simulation verschickt
- `graph:Graph` Der zu simulierende Graph mit allen Switches und Verbindungen

Funktionen

- `initialize_switches_to_be_root()` Setzt alle Switches so, dass sie sich selbst als Root betrachten und initialisiert die BPDU-Informationen
- `_find_best_path(switch_idx:int):Tuple[int, int, int]` Findet für einen Switch den besten Pfad zum Root anhand der empfangenen BPDUs
- `sptree_iteration(switch_idx:int)` Führt eine Iteration des Spanning Tree Algorithmus für einen bestimmten Switch durch
- `simulate():bool` Führt die Simulation aus, bis jeder Switch mindestens `MAX_MSG_PER_SWITCH` Nachrichten gesendet hat
- `all_switches_sent_enough_messages(min_messages:int):bool` Prüft, ob alle Switches mindestens `MAX_MSG_PER_SWITCH` Nachrichten gesendet haben
- `print_spanning_tree()` Gibt den berechneten Spanning Tree in lesbarer Form aus

3. Ausführung

1. Spanning Tree Konfiguration in die `input.txt` schreiben
2. `MAX_MSG_PER_SWITCH` auf gewünschte Nachrichtenanzahl einstellen
3. `simulation.py` ausführen
4. Ergebnis wird in der Konsole ausgegeben

4. Beispiel

```
Graph mygraph {
  // Switch-IDs
  A = 6;
  B = 1;
  C = 4;
  D = 10;
  E = 5;
  F = 3;

  // Links with costs
  A - B : 9;
  A - C : 10;
  B - D : 12;
  B - E : 10;
  C - D : 3;
  C - E : 11;
  D - E : 2;
  D - F : 4;
  E - F : 2;
}
```

Abbildung 1: Konfiguration des Beispiel Graphen

```
=== Graph Tests ===
✓ All switch IDs > 0
✓ Unique root ID: B (ID: 1)
✓ Graph is connected
✓ No self-loops found
✓ Graph contains 6 switches and 9 edges
==================

Simulating until every switch sent 10 messages...

Simulation complete:

Spanning-Tree of mygraph {
  Root: B;
  Edges:
  A - B;
  C - D;
  D - B;
  E - B;
  F - E;
}
```

Abbildung 2: Ausgabe von `simulation.py`