Moritz Hahn
Sarah Altenkrüger

1	2	Σ

Übungsblatt Nr. 5

(Abgabetermin 14.05.2024)

Aufgabe 5.1

1.

RIP (Routing Information Protocol)
RIP ist ein Beispiel für einen *decentralized routing algorithm*. RIP arbeitet nach dem Distance-Vector-Verfahren, bei dem jeder Router nur Informationen von seinen direkten Nachbarn erhält und diese Informationen weiterverbreitet. Jeder Router kennt also nur die Distanzen zu seinen direkten Nachbarn und die Distanzen, die diese Nachbarn zu anderen Zielen haben. Es gibt keine zentrale oder vollständige Kenntnis des gesamten Netzwerks.

OSPF (Open Shortest Path First)
OSPF ist ein *global routing algorithm*. Es verwendet das Link-State-Verfahren, bei dem jeder Router Informationen über den Zustand aller Links im Netzwerk sammelt und diese Informationen an alle anderen Router im Netzwerk weitergibt. Jeder Router hat dadurch eine vollständige und aktuelle Übersicht über die gesamte Netzwerkstruktur und kann mit dem Dijkstra-Algorithmus die kürzesten Pfade berechnen.

2.

Anzahl der Iterationen
Der Dijkstra-Algorithmus benötigt n Iterationen, wobei n die Anzahl der Knoten im Netzwerk ist. In jeder Iteration wird ein neuer Knoten mit dem aktuell bekannten kürzesten Pfad zu diesem Knoten als "besucht"markiert.

Pseudocode des Dijkstra-Algorithmus

```
function Dijkstra(Graph, source):\\
    dist[source] := 0
    for each vertex v in Graph:
        if v != source:
            dist[v] := infinity
        previous[v] := undefined
    S := empty set
    Q := set of all vertices in Graph

    while Q is not empty:
        u := vertex in Q with smallest dist[u]
        remove u from Q
        add u to S

        for each neighbor v of u:
            if v in Q:
                alt := dist[u] + length(u, v)
```

```
                if alt < dist[v]:
                    dist[v] := alt
                    previous[v] := u

    return dist, previous
```

3.

Der Dijkstra-Algorithmus funktioniert **nicht** korrekt mit negativen Kantengewichten. Der Grund dafür ist, dass der Algorithmus darauf basiert, dass die kürzeste bekannte Distanz zu einem Knoten nie verbessert wird, nachdem dieser Knoten als "besucht"markiert wurde. Bei negativen Kantengewichten könnte es jedoch passieren, dass eine kürzere Verbindung zu einem Knoten gefunden wird, nachdem der Knoten bereits als besucht markiert wurde, was zu falschen Ergebnissen führen würde. In solchen Fällen eignet sich der Bellman-Ford-Algorithmus besser, da er auch negative Kantengewichte korrekt verarbeiten kann.

4.

Schritt	Confirmed	Tentative
1	(A, 0, -)	{(B, 2, B), (C, 1, C)}
2	(C, 1, C)	{(B, 2, B), (E, 5, C)}
3	(B, 2, B)	{(D, 6.5, E), (E, 5, C), (F, 3, C)}
4	(F, 3, C)	{(D, 6.5, E), (E, 5, C), (G, 8.5, D)}
5	(D, 6.5, E)	{(E, 5, C), (G, 8.5, D), (H, 11, E)}
6	(E, 5, C)	{(G, 8.5, D), (H, 11, E), (I, 10.5, E)}
7	(G, 8.5, D)	{(H, 11, E), (I, 10.5, E)}
8	(I, 10.5, E)	{(H, 11, E)}
9	(H, 11, E)	-

Aufgabe 5.2

1.

Knoten	A	B	C	D	E	F	G	H
B	3	0	3	8	1	7	3	2
C	2	1	0	3	4	2	2	1
D	1	4	2	0	6	4	3	6

Link-Kosten von A zu seinen Nachbarn

- A zu B: 2
- A zu C: 5
- A zu D: 1

Berechnung der Pfadkosten von A zu den Zielknoten B bis H

Ziel	Über B (Kosten)	Über C (Kosten)	Über D (Kosten)	Günstigster Pfad
B	$2 + 0 = 2$	$5 + 1 = 6$	$1 + 4 = 5$	2 (B)
C	$2 + 3 = 5$	$5 + 0 = 5$	$1 + 2 = 3$	3 (D)
D	$2 + 8 = 10$	$5 + 3 = 8$	$1 + 0 = 1$	1 (D)
E	$2 + 1 = 3$	$5 + 4 = 9$	$1 + 6 = 7$	3 (B)
F	$2 + 7 = 9$	$5 + 2 = 7$	$1 + 4 = 5$	5 (D)
G	$2 + 3 = 5$	$5 + 2 = 7$	$1 + 3 = 4$	4 (D)
H	$2 + 2 = 4$	$5 + 1 = 6$	$1 + 6 = 7$	4 (B)

Routing-Tabelle für Knoten A

Ziel	Kosten	Next-Hop
B	2	B
C	3	D
D	1	D
E	3	B
F	5	D
G	4	D
H	4	B

2.

Nach einem Linkausfall zwischen A und D aktualisiert Knoten A seine Distanzvektoren und sendet diese an seine Nachbarn B und C. Angenommen, das Update von A erreicht B früher als C. Knoten B berechnet daraufhin seine neue Routing-Tabelle und sendet einen aktualisierten Distanzvektor an C. Wenn das Update von B den Knoten C früher erreicht als das Update von A, entsteht ein Problem.

Ablauf

1. **Linkausfall und Update von A**:
   - Der Link zwischen A und D fällt aus.
   - A sendet den neuen Distanzvektor an B und C: [0, 1, 1, ∞].
2. **B empfängt das Update von A zuerst**:
   - B aktualisiert seine Routing-Tabelle:
     - Zu D: der aktuelle kürzeste Pfad war über A (Kosten 2). Jetzt wird der neue Pfad von B über C zu D in Betracht gezogen.
     - Die Kosten von B zu D könnten jetzt $1(B \to C) + 1(C \to D) = 2$ sein (obwohl C das Update noch nicht erhalten hat).
   - B sendet seinen neuen Distanzvektor an C.
3. **C empfängt das Update von B vor dem Update von A**:
   - C sieht, dass B noch denkt, dass der Pfad zu D existiert (Kosten 2).
   - C nimmt an, dass es über B immer noch einen Pfad zu D gibt.
   - C aktualisiert seine Routing-Tabelle, wobei der Pfad zu D über B geführt wird und setzt die Kosten zu D auf 3 (eigentlich unendliche Kosten).

- C sendet seinen neuen Distanzvektor an seine Nachbarn (einschließlich A und B).

4. **C empfängt das Update von A**:
   - C sieht nun, dass der direkte Pfad zu D von A auf unendlich gesetzt wurde, aber dies hat keine sofortige Wirkung, da C bereits die falsche Information von B erhalten hat.
   - Dieser Prozess wiederholt sich, wobei jeder Knoten immer wieder den Pfadkostenwert zu D erhöht, während sie auf die neue korrekte Information warten.

Count-To-Infinity-Problem

Das oben beschriebene Szenario führt zum sogenannten **Count-To-Infinity-Problem**. Dies tritt auf, weil Routing-Updates nicht synchron und gleichzeitig an alle Knoten gesendet werden und Knoten fälschlicherweise annehmen, dass alternative Pfade existieren, obwohl diese in Wirklichkeit nicht mehr vorhanden sind. Jeder Knoten zählt den Abstand zu einem nicht erreichbaren Zielknoten immer weiter hoch (bis zur Unendlichkeit).

Das Problem wird „Count-To-Infinity“ genannt, weil die Knoten die Distanz zu einem nicht erreichbaren Knoten immer weiter erhöhen. Da es keine klare Definition von „Unendlichkeit“ in Routing-Tabellen gibt, ist eine Obergrenze für solche Zählungen oft ein großer Wert (z.B. 16 in RIP), ab welchem der Knoten erkennt, dass der Zielknoten unerreichbar ist.

Lösungsmöglichkeiten

Um das Count-To-Infinity-Problem zu vermeiden, gibt es verschiedene Ansätze:

- **Split Horizon**: Ein Router sendet keine Routing-Informationen über einen bestimmten Pfad zurück zu dem Router, von dem er diese Information erhalten hat.
- **Poison Reverse**: Erweiterung des Split Horizon, bei dem eine Routing-Information explizit als unendlich markiert wird.
- **Triggered Updates**: Sofortige Updates bei Veränderungen.
- **Holddown Timer**: Verzögerung von Updates, um zu verhindern, dass falsche Informationen verbreitet werden.

Diese Mechanismen helfen, das Count-To-Infinity-Problem zu vermeiden, indem sie verhindern, dass falsche oder veraltete Informationen das Netzwerk destabilisieren.