ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2017 Prof. R. Wattenhofer

Technische Informatik II

Klausur Mittwoch, 23. August 2017, 09:00 - 10:30 Uhr

Nicht öffnen oder umdrehen bevor die Prüfung beginnt!
Lesen Sie die folgenden Anweisungen!

Die Prüfung dauert 90 Minuten und es gibt insgesamt 90 Punkte. Die Anzahl Punkte pro Teilaufgabe steht jeweils in Klammern bei der Aufgabe. Sie dürfen die Prüfung auf Englisch oder Deutsch beantworten. **Begründen Sie** alle Ihre Antworten sofern nichts anderes dabeisteht. Beschriften Sie Skizzen und Zeichnungen verständlich. Was wir nicht lesen können, können wir auch nicht bewerten!

Schreiben Sie zu Beginn Ihren Namen und Ihre Legi-Nummer in das folgende dafür vorgesehene Feld und beschriften Sie **jedes Zusatzblatt** ebenfalls mit Ihrem Namen und Ihrer Legi-Nummer.

Familienname	**Vorname**	**Legi-Nr.**

Aufgabe	**Erreichte Punktzahl**	**Maximale Punktzahl**
1 - Multiple Choice		15
2 - Markovketten		18
3 - Locking		20
4 - Disks und Dateisysteme		18
5 - Routing Loops		19
Summe		**90**

1 Multiple Choice (15 Punkte)

Geben Sie bei jeder Aussage an, ob sie wahr oder falsch ist. Jede korrekte Antwort gibt 1 Punkt, **jede fehlerhafte Antwort -1 Punkt**. Jede unbeantwortete Aussage gibt 0 Punkte. Wenn die Summe der Punkte für alle Aussagen negativ ist, dann wird die Aufgabe insgesamt mit 0 Punkten bewertet. **In dieser Aufgabe können Sie Ihre Antworten nicht begründen.**

Aussage	**wahr**	**falsch**
Es existieren keine Ports im IP Protokoll.	☐	☐
In UDP sorgen Timeouts dafür, dass verloren gegangene Pakete erneut geschickt werden.	☐	☐
Wenn alle Root Name Server gleichzeitig ausfallen, dann kann kein Browser mehr URLs auflösen.	☐	☐
Amplituden-Modulation ist besonders anfällig für Empfangsprobleme durch Noise.	☐	☐
HAVING und WHERE Klauseln können als Synonyme füreinander verwendet werden.	☐	☐
Wenn wir JOINs benutzen benötigen wir sogenannte JOIN Tabellen.	☐	☐
Ein Foreign Key kann nur einen Primary Key referenzieren.	☐	☐
Jede Hashfunktion $h : U \rightarrow M$ erzeugt Kollisionen auf jeder Schlüsselmenge $N \subseteq U$.	☐	☐
Perfect Static Hashing garantiert, dass jede Suche konstant viel Zeit kostet.	☐	☐
Wenn Hashes uniform verteilt für eine Hashtabelle der Grösse $m = 1000$ gewählt werden, dann ist die **Wahrscheinlichkeit einer Kollision** bei **100** Schlüsseln bereits 50%.	☐	☐
Wenn Hashes uniform verteilt für eine Hashtabelle der Grösse $m = 1000$ gewählt werden, dann ist die **Wahrscheinlichkeit einer Kollision** bei **200** Schlüsseln bereits 50%.	☐	☐
Wenn Hashes uniform verteilt für eine Hashtabelle der Grösse $m = 1000$ gewählt werden, dann ist bei **500** Schlüsseln **eine Kollision garantiert**.	☐	☐
Synchronisation kann auch auf Rechnern mit einer einzigen CPU relevant sein.	☐	☐
Wenn mehrere Prozesse auf einem Listen-basierten Set Fine Grained Locking verwenden und ein Prozess sehr langsam ist, dann kann dieser andere Prozesse bremsen.	☐	☐
Die Sicherheit vieler Public-Key-Verschlüsselungsverfahren beruht darauf, dass es schwer ist grosse Primzahlen zu finden.	☐	☐

Lösungen

Aussage	**wahr**	**falsch**
Es existieren keine Ports im IP Protokoll. *Begründung: Ports sind in TCP und UDP definiert.*	✓	
In UDP sorgen Timeouts dafür, dass verloren gegangene Pakete erneut geschickt werden. *Begründung: Per Definition.*		✓
Wenn alle Root Name Server gleichzeitig ausfallen, dann kann kein Browser mehr URLs auflösen. *Begründung: Aufgrund von caching bei Nicht-Root Name Servern (bspw. denen der ISPs) werden im Allgemeinen nur wenige requests direkt von den Root Name Servern beantwortet; die allermeisten Anfragen können ohne deren Mithilfe beantwortet werden.*		✓
Amplituden-Modulation ist besonders anfällig für Empfangsprobleme durch Noise. *Begründung: Die für die Kodierung verwendeten Amplituden werden durch die Addition von Noise besonders gestört.*	✓	
HAVING und WHERE Klauseln können als Synonyme füreinander verwendet werden. *Begründung: Nach GROUP BY müssen wir HAVING benutzen.*		✓
Wenn wir JOINs benutzen benötigen wir sogenannte JOIN Tabellen. *Begründung: Man kann beliebige Tabellen Joinen. Join-Tabellen sind einfach dazu gedacht die Beziehungen zwischen Tabellen explizit darzustellen.*		✓
Ein Foreign Key kann nur einen Primary Key referenzieren. *Begründung: Es muss ja ein 1-1 mapping sein. Ein Key kann nicht mehrere Primary Keys referenzieren, da das mapping gar nicht definiert wäre.*	✓	
Jede Hashfunktion $h : U \to M$ erzeugt Kollisionen auf jeder Schlüsselmenge $N \subseteq U$. *Begründung: Wähle $\|M\| \geq \|N\|$, und lasse h so sein, dass h jeden Schlüssel $k \in N$ auf einen eigenen Bucket abbildet. Oder trivialer Fall: sei $\|N\| \leq 1$, dann gibt es keine zwei Schlüssel, die kollidieren könnten.*		✓
Perfect Static Hashing garantiert, dass jede Suche konstant viel Zeit kostet. *Begründung: In Perfect Static Hashing kostet nur die Konstruktion der Tabelle in Erwartung linear viel Zeit, Suche kostet immer (also auch worst-case) konstant viel.*	✓	
Wenn Hashes uniform verteilt für eine Hashtabelle der Grösse $m = 1000$ gewählt werden, dann ist die **Wahrscheinlichkeit einer Kollision** bei **100** Schlüsseln bereits 50%. *Begründung: Siehe Birthday Problem Rechnung im Skript.*	✓	
Wenn Hashes uniform verteilt für eine Hashtabelle der Grösse $m = 1000$ gewählt werden, dann ist die **Wahrscheinlichkeit einer Kollision** bei **200** Schlüsseln bereits 50%. *Begründung: Siehe Birthday Problem Rechnung im Skript.*	✓	
Wenn Hashes uniform verteilt für eine Hashtabelle der Grösse $m = 1000$ gewählt werden, dann ist bei **500** Schlüsseln **eine Kollision garantiert**.		✓

Begründung: Dafür benötigt es mindestens 1001 *Schlüssel, da sonst per Zufall alle Schlüssel in einen eigenen Bucket kommen könnten.*	
Synchronisation kann auch auf Rechnern mit einer einzigen CPU relevant sein. *Begründung: Solange Prozesses unterbrochen und später fortgesetzt werden können (preemption) können Nebenläufigkeitsprobleme auftreten.*	✓
Wenn mehrere Prozesse auf einem Listen-basierten Set Fine Grained Locking verwenden und ein Prozess sehr langsam ist, dann kann dieser andere Prozesse bremsen. *Begründung: Überholen geht ja nicht.*	✓
Die Sicherheit vieler Public-Key-Verschlüsselungsverfahren beruht darauf, dass es schwer ist grosse Primzahlen zu finden. *Begründung: Grosse Primzahlen finden ist einfach.*	✓

Diese Seite wurde absichtlich leer gelassen.

2 Markovketten (18 Punkte)

In dieser Aufgabe wird das folgende Spiel betrachtet: Ein Schiedsrichter wirft mehrmals eine faire Münze. Zwei Spieler müssen sich vorher jeweils auf ein Tupel von zwei aufeinander folgenden Würfelausgängen festlegen. Spieler A hat dabei die vier Tupel HH, HT, TH und TT zur Auswahl, wobei H für Heads und T für Tails steht. Spieler B sucht sich dann aus den übrigen drei Tupeln eines für sich aus. Danach wird eine Münze solange hintereinander geworfen, bis eines der ausgewählten Tupel in zwei aufeinander folgenden Würfen auftritt. Der Spieler, dessen Tupel zuerst geworfen wurde, gewinnt.

Beispiel: Spieler A entscheidet sich für das Tupel TH und Spieler B für TT. Der Schiedsrichter würfelt die Sequenz $HHTT\ldots$ Spieler B gewinnt, weil das Tupel TT zuerst aufgetreten ist.

Gehen Sie zunächst davon aus, dass noch kein Spieler seine Wahl getroffen hat.

a) [6] Modellieren Sie den Verlauf des Spiels mit einer Markovkette.

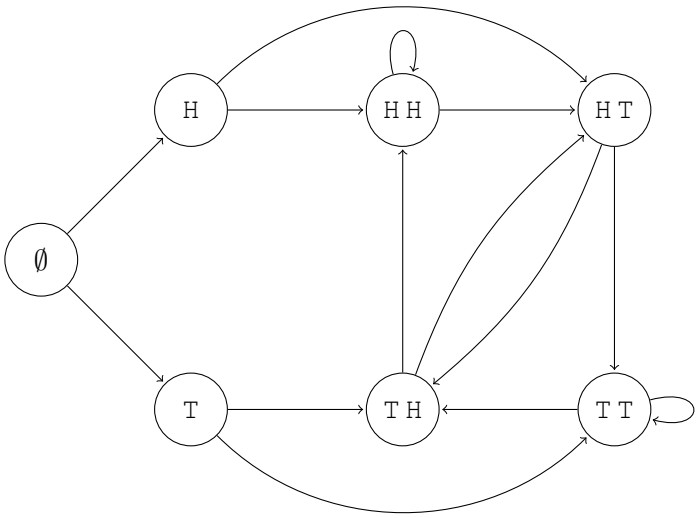
Für den nachfolgenden Teil der Aufgabe legt sich Spieler A auf das Tupel HH fest.

b) [6] Welches Tupel sollte Spieler B wählen, damit er die grössere Chance zu gewinnen hat? Berechnen Sie die entsprechenden Wahrscheinlichkeiten, mit denen die beiden Spieler gewinnen würden.

c) [6] Wie lange dauert es in Erwartung, bis einer der beiden Spieler gewinnt?

Lösungen

a) Alle Kanten haben Gewicht 1/2



b) Der zweite Spieler sollte das Tupel TH wählen. Dann ist die Gewinnwahrscheinlichkeit für den ersten Spieler 1/4 und für den zweiten Spieler 3/4.

Möglichkeit 1: Einzelne Wahrscheinlichkeiten für alle Tupel ausrechnen:

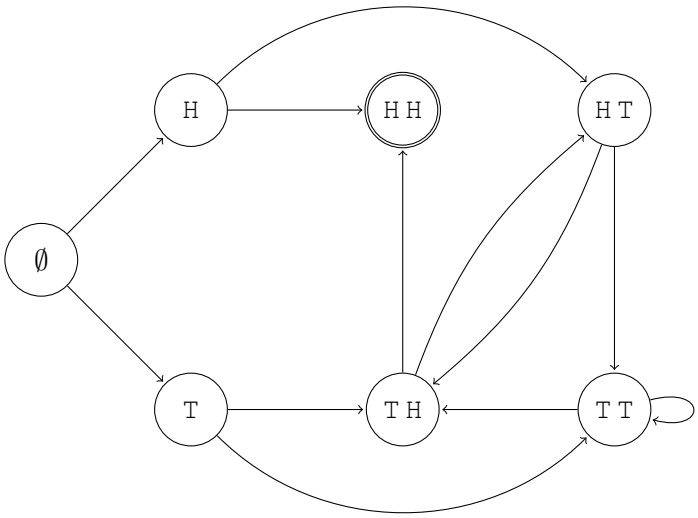
Pr[erster Spieler gewinnt | zweiter Spieler wählt HT] = Pr[nach dem ersten H wird erneut H gewürfelt] = $\frac{1}{2}$

Pr[erster Spieler gewinnt | zweiter Spieler wählt TT] = $\frac{1}{2}$, da HH und TT symmetrische Ereignisse sind

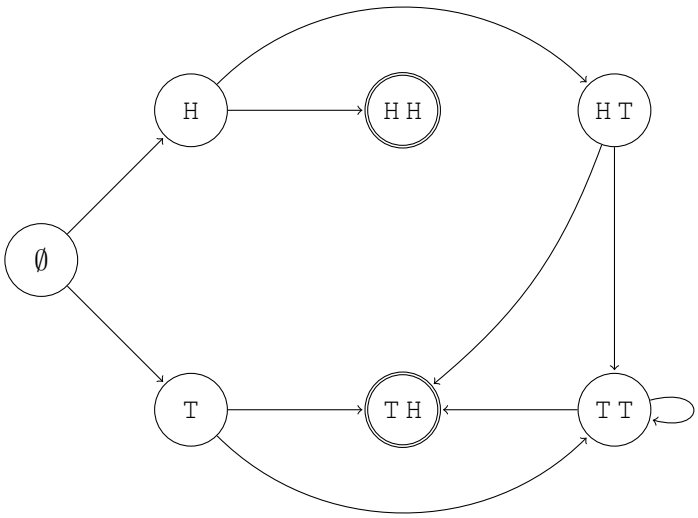
Pr[erster Spieler gewinnt | zweiter Spieler wählt TH] = Pr[H wird zweimal hintereinander gewürfelt] = $\frac{1}{4}$

Damit ist TH die beste Strategie für den zweiten Spieler.

Möglichkeit 2: Alternativ kann man die Markovkette betrachten. Wenn Spieler A sich entschieden hat, entsteht ein Endzustand in der Kette:



Nun kann Spieler B mit einer Wahl von TH den Knoten HH von den restlichen Zuständen abtrennen:



Die Wahrscheinlichkeit, dass der erste Spieler gewinnt beträgt dann nur noch 1/4

c)

$$
\begin{aligned}
\mathbb{E}[\text{Spiellänge}] &= 2 \cdot Pr[\text{Spiel endet direkt in } HH] \\
&\quad + 3 \cdot Pr[\text{Spiel endet nach } HT \text{ direkt in } TH] \\
&\quad + (3 + \mathbb{E}[\text{Spielzüge von } TT \text{ nach } TH]) \cdot Pr[\text{Spiel endet nach } HT \ldots T \text{ in } TH] \\
&\quad + 2 \cdot Pr[\text{Spiel endet direkt in } TH] \\
&\quad + (2 + \mathbb{E}[\text{Spielzüge von } TT \text{ nach } TH]) \cdot Pr[\text{Spiel endet nach } T \ldots T \text{ in } TH] \\
&= 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + (3+2) \cdot \frac{1}{8} + 2 \cdot \frac{1}{4} + (2+2) \cdot \frac{1}{4} \\
&= 3
\end{aligned}
$$

3 Locking (20 Punkte)

In dieser Aufgabe müssen Sie ein System debuggen welches zeitweise nur stark verlangsamt funktioniert (manchmal macht das System keinen Fortschritt). Das Betriebssystem benutzt preemptive Scheduling. Das bedeutet, dass Threads vom Betriebssystem aktiv unterbrochen werden. Insgesamt gibt es drei Threads, von denen das System nur zwei zugleich ausführen kann (dual-core). Der dritte Thread ist pausiert. Die Software verwendet ausserdem ein Lock. Falls ein Thread das Lock benötigt, aber es momentan noch besetzt ist, muss der Thread spinnen. Der einfache Scheduler wird periodisch jede Sekunde einmal ausgeführt. Er setzt den aktuell pausierten Thread fort und pausiert dafür naiv zufällig einen der anderen zwei Threads.

Definition (**Fortschritt**)**.** *Der Fortschritt ist die Summe der Zeit, die einzelne Threads arbeiten. Threads, die pausiert sind oder auf das Lock warten, tragen nichts zum Fortschritt bei. Zwei Threads, die 1s laufen ohne auf das Lock zu warten, machen 2s Fortschritt.*

a) `[4]` Wir betrachten eine Sekunde in welcher der Scheduler nie ausgeführt wird. Erklären Sie wie wenig Fortschritt im schlechtesten Fall gemacht wird.

b) `[4]` Wir nehmen an, das verwendete Lock ist als TTAS Lock implementiert. Erklären Sie wie wenig Fortschritt im schlechtesten Fall gemacht wird in einer Sekunde, in welcher der Scheduler nach der halben Zeit einmal ausgeführt wird.

c) `[4]` Wir nehmen an, es wird ein MCS Lock verwendet. Erklären Sie wie wenig Fortschritt im schlechtesten Fall gemacht wird in einer Sekunde, in welcher der Scheduler nach der halben Zeit einmal ausgeführt wird.

Für das gegebene Szenario wird das MCS Lock nun angepasst, damit dieses effizienter wird. Sie haben eine Funktion `schedule(QNode)`, die den zu einer QNode gehörenden Thread startet, falls dieser momentan pausiert ist (dies pausiert den aufrufenden Thread). In `unlock` des MCS Locks wird nach dem `qnode.next.locked = false` nun noch `schedule(qnode.next)` aufgerufen.

d) `[4]` Wir nehmen an, diese Implementierung des MCS Locks wird verwendet. Erklären Sie wie wenig Fortschritt im schlechtesten Fall gemacht wird in einer Sekunde, in welcher der Scheduler nach der halben Zeit einmal ausgeführt wird.

e) `[4]` Abgesehen vom Fortschritt: Warum ist es bei dieser Implementierung des MCS Locks notwendig, dass der Scheduler noch periodisch ausgeführt wird?

Lösungen

a) Kein Fortschritt falls der pausierte Thread das Lock hält und die anderen zwei Threads das Lock benötigen.

b) 0.5 s. Der Thread der das Lock hält wird mindestens 0.5 s auf einem Core laufen.

c) (Fast) Kein Fortschritt. Das MCS Lock garantiert FIFO. Erste 0.5 s ist der Thread, der das Lock hat, pausiert. Danach kann er laufen und gibt das Lock sofort frei, und verlangt das Lock gleich wieder. Der nächste in der Warteschleife ist jedoch der Thread, der jetzt pausiert ist.

d) 0.5 s. Entweder läuft der Thread mit dem Lock in den ersten 0.5 s und wird dann pausiert. Oder er ist pausiert mit dem Lock in den ersten 0.5 s und er behält das Lock danach oder es wird weitergegeben und die Threads aufgeweckt.

e) Wenn der Scheduler nicht ausgeführt wird, werden nur Prozesse ausgewechselt, die das Lock verwenden. Ein Thread der das Lock nicht verwendet, läuft entweder seit dem Start und wird nie gestoppt oder er wird nie ausgeführt.

4 Disks und Dateisysteme (18 Punkte)

In Tabelle 3 finden Sie die inode region und die data region eines kleinen physischen Dateisystems. Der Inhalt einer inode ist jeweils in zwei Zeilen dargestellt; die erste Zeile enthält `owner` und `group owner` der Datei, die zweite Zeile den permission string für die Datei und den Index des Datenblocks, der den Inhalt der Datei enthält (in dieser Aufgabe hat jede Datei nur einen zugehörigen Datenblock). Das root directory wird durch die inode mit Index 0 repräsentiert.

inode index	inode content
0	`X H` `drwxrwxrwx 1`
1	`Y F` `-rwx-wxrw- 3`
2	`Z G` `----rw-rw- 0`
3	`Y H` `dr-xrwxrwx 2`
4	`Z F` `-rw-r---wx 6`
5	`X G` `d-wx-wxrw- 4`
6	`Y G` `-rw-r-x-wx 5`

data block index	content
0	WINNER
1	`A : 2` `B : 5`
2	`A : 2` `B : 4`
3	winner
4	`A : 3` `B : 1`
5	chicken
6	dinner

Tabelle 3: inode region und data region unseres physischen Dateisystems.

a) `[6]` Zeichnen Sie den korrespondierenden Verzeichnisbaum (directory tree). Zeichnen Sie directories und files als Paare von `(Name, inode Index)`. Als Beispiel, das root directory wird als `(/, 0)` identifiziert.

b) `[3]` Es gibt eine Datei, die nicht in den Verzeichnisbaum eingebunden ist. Was ist der inode-Index und was der Datenblockindex dieser Datei?

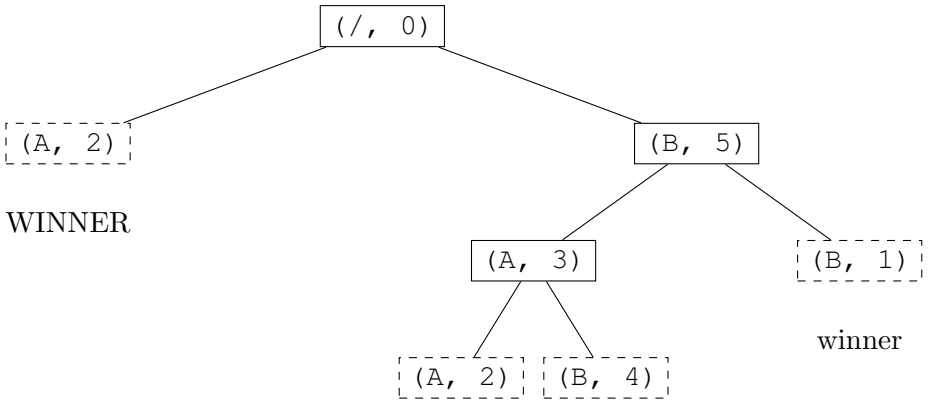
c) `[3]` Wir führen den Befehl `echo "oh no" > /A` aus. Geben Sie den Inhalt aller files (nicht für directories) im Verzeichnisbaum nach Ausführen des Befehls als ein Tupel `Dateipfad : Dateiinhalt` an.

d) Wenn ein user den Befehl `cat /B/A/B` im terminal ausführt, dann gibt er den Inhalt der Datei `/B/A/B` aus, sofern der user die nötigen permissions hat; wenn dem user notwendige permissions fehlen, dann wird eine Fehlermeldung ausgegeben. Geben Sie für jeden der folgenden Fälle an, was beim Ausführen des Befehls `cat /B/A/B` ausgegeben wird. Geben Sie im Falle einer Fehlermeldung zusätzlich an, welche permissions fehlen.

- `[2]` User `X` in Gruppe `F` führt den Befehl aus.
- `[2]` User `Y` in Gruppe `G` führt den Befehl aus.
- `[2]` User `Z` in Gruppe `H` führt den Befehl aus.

Lösungen

a) Im Verzeichnisbaum sind directories mit durchgezogenen Rahmen dargestellt, während files gestrichelte Rahmen haben. (Unter die files haben wir der Übersicht halber deren Inhalt geschrieben; das ist nicht in der Aufgabe verlangt und wurde auch nicht bepunktet.)



b) Die Datei hat inode-Index 6 und Datenblockindex 5.

c) `/A` und `/B/A/A` sind zwei Pfade zur selben Datei (also selbe inode), es ändert sich also der Inhalt an beiden Stellen im Verzeichnisbaum:

```
/A : oh no
/B/A/A : oh no
/B/A/B : dinner
/B/B : winner
```

d) Zur Erinnerung: um einen Pfad aufzulösen brauchen wir execute-Rechte für alle directories auf dem Pfad, Leserechte benötigen wir für die directories nicht. Um eine Datei zu lesen brauchen wir nur Leserechte für die Datei. Das ergibt folgende Lösungen:

- User `X` in Gruppe `F` führt den Befehl aus: X hat die nötigen execute-Rechte für directories (als user für `/` und `/B/`, als other für `/B/A/`) und Leserechte als group für `/B/A/B`. Der Befehl gibt "dinner" aus.
- User `Y` in Gruppe `G` führt den Befehl aus: Y hat die nötigen execute-Rechte für directories (als other für `/`, als group für `/B/`, als user für `/B/A/`), aber Y fehlen als other die Leserechte für `/B/A/B`. Der Befehl gibt eine Fehlermeldung aus.
- User `Z` in Gruppe `H` führt den Befehl aus: Z hat execute-Rechte als group für `/` und `/B/A/` sowie Leserechte als user für `/B/A/B`, aber Z fehlen als other die execute-Rechte für `/B/`. Der Befehl gibt eine Fehlermeldung aus.

5 Routing Loops (19 Punkte)

In einem Netzwerk weiss jeder Knoten anhand der Routing-Tabelle wie er eintreffende Pakete weiterleiten soll. Durch den Ausfall eines Links ist es jedoch (kurzzeitig) möglich, dass die Pakete nicht zugestellt werden können, obwohl noch ein physischer Pfad zwischen den zwei Knoten existiert. Die Pakete werden zwischen zwei oder mehr Knoten hin und her gesendet. Z.B. v_1 leitet das Paket zu v_2 weiter, der es wieder an v_1 weiterleitet, usw. Es hat sich ein "Kurzfristiger Routing Loop" (KRL) gebildet. Den Ausfall eines Links oder Knotens sehen die direkten Nachbar-Knoten ohne Verzögerung.

a) `[4 Punkte]` In einem Netzwerk mit wenigen Knoten wird das Protokoll RIP verwedet. Nach dem Ausfall eines Links kann es bis zu ein paar Minuten dauern, bis ein Knoten von der Änderung erfährt. Weshalb dauert es so lange?

b) `[2 Punkte]` Wie kann ein am KRL beteiligter Knoten den KRL erkennen?

c) `[2 Punkte]` Wann löst sich ein KRL in einem Link-State-Protokoll wieder auf?

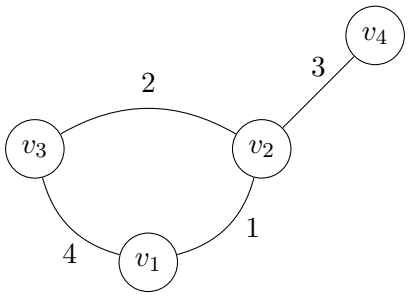


Abbildung 1: Netzwerk mit vier Knoten. Werte bei Links geben Kosten des Links an.

d) `[6 Punkte]` Zeigen Sie am Graph in Abbildung 1, wie ein KRL durch den Ausfall eines Links entstehen kann, wobei der Graph immer noch zusammenhängend bleibt. Geben Sie die Routing Tabellen der am KRL beteiligten Knoten an gleich nach dem Ausfall des Links an. Nehmen Sie an, die Routing-Tabellen wurden mit einem Link-State-Protokoll erstellt.

e) `[5 Punkte]` Kann ein KRL auch auftreten, wenn ein Knoten in einem Netzwerk ausfällt? Wenn nicht, erklären Sie weshalb. Wenn schon, zeigen Sie es mit einem Beispiel.

Lösungen

a) Die Routing Tabelle wird von jedem Knoten alle 30 Sekunden an seine Nachbarn gesendet. Das Netzwerk kann einen maximalen Durchmesser von 15 Hops haben. Somit kann es mehrere Minuten dauern bis jeder Knoten von der Änderung erfährt. Zudem werden die Nachrichten per UDP gesendet und können somit verloren gehen.

b) Wenn der Knoten das selbe Paket zweimal sieht. (Oder das Paket an den Absender zurücksendet.)

c) KRL löst sich auf wenn neuer Link-Zustand zu allen am KRL beteiligten Knoten verteilt und Routing-Tabellen neu berechnet wurden.

d) Link zwischen Knoten v_1 und v_2 fällt aus. v_2 sendet Paket für v_1 über v_3, v_3 sendet es jedoch wieder zurück. Routing-Tabellen von v_2 und v_3:

Routing Tabelle von v_2	
Destination	Next node
v_1	v_3
v_2	deliver
v_3	v_3
v_4	v_4

Routing Tabelle von v_3	
Destination	Next node
v_1	v_2
v_2	v_2
v_3	deliver
v_4	v_2

e) Ja. Bei einem Graphen, bei dem ein Link-Ausfall zu einem KRL führt, kann in diesen Link ein zusätzlicher Knoten eingefügt werden. Wenn dieser Knoten ausfällt, hat es den selben Effekt wie wenn der Link ausfällt. Es kann ein Knoten v_5 zwischen v_1 und v_2 in den Link eingefügt werden mit Links mit Kosten von 0.5 zu v_1 und v_2.