ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2018 Prof. R. Wattenhofer

Technische Informatik II

Klausur

Mittwoch, 8. August 2018, 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 gibt keine Punkte!

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		6
2 - Extended Slotted ALOHA		22
3 - Flow Control mit LPs		20
4 - Evil Scheduling		22
5 - Hashing		20
Summe		**90**

1 Multiple Choice (6 Punkte)

Geben Sie bei jeder Aussage an, ob sie wahr oder falsch ist. Jede korrekte Antwort gibt 1 Punkt. Jede fehlerhafte Antwort und unbeantwortete Aussage gibt 0 Punkte. **In dieser Aufgabe können Sie Ihre Antworten nicht begründen oder erklären.**

a) `[3 Punkte]` Wir betrachten das Dining-Philosophers-Problem.

	wahr	**falsch**
Es gibt mehr als eine Möglichkeit um das Dining-Philosophers-Problem zu lösen.	☐	☐
Bei nur drei Prozessen kann das Dining-Philosophers-Problem nicht auftreten.	☐	☐
Peterson's Algorithmus kann auf moderner Hardware zum Dining-Philosophers-Problem führen.	☐	☐

b) `[3 Punkte]` Wir nehmen an, ein Angreifer erfährt den Klartext und den dazugehörigen Ciphertext einer Nachricht. Der Ciphertext besteht aus Cipher Block Chaining (CBC) Blöcken.

	wahr	**falsch**
Wenn die Nachricht durch das Shiften jedes Zeichens um k Positionen im Alphabet verschlüsselt wurde kann der Angreifer den Schlüssel k herausfinden.	☐	☐
Wenn jeder Block durch ein anderes One-Time-Pad verschlüsselt ist, kann der Angreifer jedes der One-Time-Pads bestimmen.	☐	☐
Durch die Verwendung des CBC-Modus kann es für den Angreifer schwieriger sein, den Schlüssel zu finden als beim ECB-Modus.	☐	☐

Lösungen

a) [3 Punkte] Wir betrachten das Dining-Philosophers-Problem.

	wahr	falsch
Es gibt mehr als eine Möglichkeit um das Dining-Philosophers-Problem zu lösen. *Begründung: Das Problem kann auch gelöst werden indem die Philosophen erst die Essstäbchen mit den ungeraden Nummern nehmen. Es gibt also mehrere Lösungen.*	✓	
Bei nur drei Prozessen kann das Dining-Philosophers-Problem nicht auftreten. *Begründung: Kann immer noch auftreten.*		✓
Peterson's Algorithmus kann auf moderner Hardware zum Dining-Philosophers-Problem führen. *Begründung: Das Dining-Philosophers-Problem ist über Deadlocks, was nicht durch Umordnen von Instruktionen passieren kann.*		✓

b) [3 Punkte] Wir nehmen an, ein Angreifer erfährt den Klartext und den dazugehörigen Ciphertext einer Nachricht. Der Ciphertext besteht aus Cipher Block Chaining (CBC) Blöcken.

	wahr	falsch
Wenn die Nachricht durch das Shiften jedes Zeichens um k Positionen im Alphabet verschlüsselt wurde kann der Angreifer den Schlüssel k herausfinden. *Begründung: k kann bestimmt werden durch das Subtrahieren des ersten Zeichens des Klartextes vom ersten Zeichen des Ciphertextes.*	✓	
Wenn jeder Block durch ein anderes One-Time-Pad verschlüsselt ist, kann der Angreifer jedes der One-Time-Pads bestimmen. *Begründung: Jedes One-Time-Pad kann durch XOR des verschlüsselten Blocks mit dem Klartext-Block und dem vorherigen Ciphertext-Block bestimmt werden.*	✓	
Durch die Verwendung des CBC-Modus kann es für den Angreifer schwieriger sein, den Schlüssel zu finden als beim ECB-Modus. *Begründung: Der Angreifer kennt den Klartext und den Ciphertext der Nachricht, der CBC-Modus hat dadurch keinen Einfluss auf die Schwierigkeit des Bestimmens des Schlüssels.*		✓

2 Extended Slotted ALOHA (22 Punkte)

In dieser Aufgabe schauen wir uns Slotted ALOHA an. Wie in Abbildung 1 dargestellt gibt es vier Sender S_i ($i \in 1, 2, 3, 4$) und einen Empfänger E. Die vier Sender senden Pakete zum Empfänger. Der Empfänger kann nur Pakete empfangen und die Sender können nur Pakete senden. In jedem Zeitslot sendet Sender S_i mit Wahrscheinlichkeit q_i und Sendeleistung P_i. Die Leistung am Empfänger ist $P_i^E = 1/2 * P_i$. Der Empfänger kann eine Nachricht nur dann erfolgreich empfangen wenn die signal-to-interference-plus-noise ratio (SINR) grösser als 1 ist, das heisst, $\beta = 1$. Für alle Aufgaben nehmen wir ambient noise $N = 0$ an. Wir definieren den Mindestdurchsatz $D_{\min}$ des Netzwerks als $D_{\min} = \min(D_i)$, wobei D_i die durchschnittliche Anzahl erfolgreich von S_i übertragener Pakete pro Zeitslot darstellt.

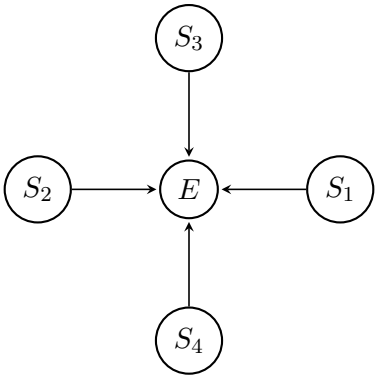


Abbildung 1: Wireless Netzwerk mit 5 Knoten.

a) `[6 Punkte]` Gemäss Algorithmus 5.46 wählen wir $q_i = 1/5$. Alle Sender haben die gleiche Sendeleistung $P_i = 1$. Was ist $D_{\min}$?

b) `[6 Punkte]` Finden Sie die Sendewahrscheinlichkeit q ($q_i = q$ für alle i) welche $D_{\min}$ maximiert. Beweisen Sie, dass Ihre Lösung optimal ist!

c) `[10 Punkte]` P_i und q_i können jetzt frei und für jeden Sender individuell gewählt werden. Finden Sie Werte für P_i und q_i, so dass $D_{\min}$ maximiert wird. Was ist $D_{\min}$?

Lösungen

a) Da alle Sender mit $P_i = 1$ senden, darf maximal 1 Sender gleichzeitig senden, weil $SINR = \frac{1/2}{1/2} = 1 = \beta$ wenn zwei Sender gleichzeitig senden. Alle Sender senden mit Wahrscheinlichkeit q. Das heisst, für einen Sender ist die Wahrscheinlichkeit ein Paket in einem Zeitslot erfolgreich zu übertragen $Q_s = q * (1-q)^3$. Da alle Sender mit der gleichen Wahrscheinlichkeit übertragen gilt $D_{\min} = Q_s$. Wenn wir $q = 1/5$ einsetzen erhalten wir $D_{\min}(q = 1/5) = 64/625 = 0.1024$.

b) Um das Maximum zu finden lösen wir einfach die Gleichung $\frac{d}{dq} q * (1-q)^3 = 0$ nach q auf. Mit der Produkt und Kettenregel für Ableitungen und anschliessendem ausklammern von $(1-q)^2$ erhalten wir

$$\frac{d}{dq} q * (1-q)^3 = (1-q)^2 * (1-4q) \tag{1}$$

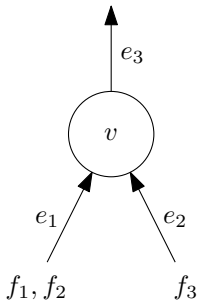
Wir sehen jetzt, dass dieser Ausdruck nur null sein kann wenn entweder $(1-q) = 0$ oder $(1-4q) = 0$. Daraus folgt, dass die beiden Extremalstellen $q_1 = 1/4$ und $q_2 = 1$ sind. Durch einsetzen in $D_{\min}$ sieht man sofort, dass $q_2 = 1$ ein minimum ist, und somit $q_1 = 1/4$ die optimale Sendewahrscheinlichkeit ist. Letztlich bekommt man $D_{\min}(q = 1/4) = 27/256 = 0.1055$.

c) Wir können die P_i so wählen, dass gewisse Sender andere übertönen können. Wenn wir dazu die Sendewahrscheinlichkeiten noch geschickt wählen, können wir dafür sorgen, dass in jedem Zeitslot genau ein Paket ankommt. Da wir $D_{\min}$ maximieren wollen, müssen wir dafür zusätzlich sicherstellen, dass alle Sender den gleichen durchschnittlichen Durchsatz haben, i.e., $D_i = D_{min} = 1/4 \forall i$.

Dafür wählen wir zuerst die Sendeleistungen, zum Beispiel, als $P_1 = 1, P_2 = 2, P_3 = 4, P_4 = 8$. Merke, dass P_4 jetzt immer Erfolg hat wenn er sendet, und P_1 nur dann wenn kein anderer sendet, etc. Jetzt können wir die q_i einfach so wählen, dass für jeden Sender $D_i = 1/4$ gilt. Dies ist der Fall für $q_1 = 1, q_2 = 1/2, q_3 = 1/3, q_4 = 1/4$.

3 Flow Control mit LPs (20 Punkte)

Flow-Probleme können nicht immer global gelöst werden. In manchen Fällen (z.B. in einem Router) muss lokal entschieden werden, welche Flows mit welcher Geschwindigkeit weitergeleitet werden. Im Folgenden betrachten wir ein *lokales* Flow-Problem im Knoten v, das heisst, gegeben die eingehenden Flows f_i (mit $1 \leq i \leq 3$) auf den Kanten e_1 und e_2 müssen wir entscheiden, welche Flows mit den Raten r_i auf der Kante e_3 weitergeleitet werden. Die Kanten haben Kapazitäten $c(e_1) = c(e_2) = 3$ und $c(e_3) = 5$. Jeder Flow f_i ist splittable und hat Demand $d_i = i$, also $d_1 = 1$, $d_2 = 2$ und $d_3 = 3$. Falls mehrere Flows eine Kante passieren, darf die Summe ihrer Raten die Kapazität der Kante nicht übersteigen.



Es gelten ausserdem folgende Prioritäten: Flow f_1 ist **sehr** wichtig, Flow f_2 ist wichtig und Flow f_3 ist weniger wichtig.

a) `[4 Punkte]` Stellen Sie ein LP auf, das eine Allokation der gegebenen Flows unter Berücksichtigung der Prioritäten berechnet.

b) `[6 Punkte]` Eine Studentin behauptet, sie könne einfach die Flows f_1 und f_2 zusammenfassen zu f_x mit der Rate $r_x = r_1 + r_2$. Stellen Sie das resultierende LP für f_3 und f_x auf, und lösen Sie dieses mit dem Simplex-Algorithmus beginnend bei $(0, 0)$. Wählen Sie dazu eine geeignete, möglichst einfache Funktion, die vom Simplex-Algorithmus maximiert werden soll und die Prioritäten abbildet.

c) `[6 Punkte]` Angenommen der Flow f_3 kann nur vollständig ($r_3 = d_3$) oder gar nicht ($r_3 = 0$) zugewiesen werden. Weiterhin muss die Kapazität der ausgehenden Kante e_3 vollständig ausgenutzt werden. Könnte dieses Problem dennoch mit einem LP gelöst werden?

d) `[4 Punkte]` Geben Sie für beide Fälle (Flow f_3 ist teilweise zuweisbar/nur vollständig zuweisbar) eine Max-Min-Fair Allokation der Flows an, wenn die ausgehende Kante e_3 jeweils vollständig ausgelastet sein muss.

Lösungen

a)

$$\begin{array}{lll} r_1 \geq 0 & r_1 \leq 1 = d_1 & r_1 + r_2 \leq 3 = c(e_1) \\ r_2 \geq 0 & r_2 \leq 2 = d_2 & r_1 + r_2 + r_3 \leq 5 = c(e_3) \\ r_3 \geq 0 & r_3 \leq 3 = d_3 & \end{array}$$

Nun muss eine Funktion maximiert werden, die die Prioritäten adäquat umsetzt. Zum Beispiel:

$$f_1(r) = \underbrace{3r_1}_{\textbf{sehr } wichtig} + \underbrace{2r_2}_{wichtig} + \underbrace{r_3}_{weniger\ wichtig} .$$

b)

$$r_x = r_1 + r_2 \qquad \Longrightarrow\ r_x \leq 3 = d_x = d_1 + d_2 = c(e_1)$$

Das neue LP ist also:

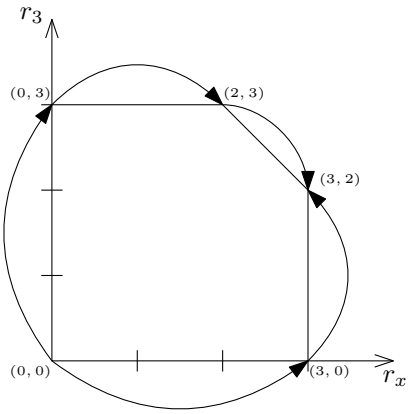
$$\begin{array}{lll} r_x \geq 0 & r_x \leq 3 = d_x & r_x + r_3 \leq 5 = c(e_3) \\ r_3 \geq 0 & r_3 \leq 3 = d_3 & \end{array}$$

Wir maximieren nun die Funktion:

$$f_2(r) = \underbrace{2r_x}_{\textbf{sehr } wichtig\,/\,wichtig} + \underbrace{r_3}_{weniger\ wichtig} .$$

Daraus ergeben sich zwei korrekte Lösungen mit dem Simplex-Algorithmus:

- $(0,0) \to (3,0) \to (3,2)$
- $(0,0) \to (0,3) \to (2,3) \to (3,2)$



c) Man kann hier beobachten, dass $d_1 + d_2 = 3$ noch zu wenig sind, um e_3 ganz auszulasten. Deshalb muss $r_3 = d_3 = 3$ gelten und r_1 bzw. r_2 können durch folgendes LP bestimmt werden:

$$\begin{array}{lll} r_1 \geq 0 & r_1 \leq 1 = d_1 & r_1 + r_2 \leq 2 = c(e_3) - r_3 \\ r_2 \geq 0 & r_2 \leq 2 = d_2 & \end{array}$$

Wir maximieren nun die Funktion:

$$f_3(r) = \underbrace{2r_1}_{\textbf{sehr } wichtig} + \underbrace{r_2}_{wichtig} .$$

d)

- f_3 **teilweise zuweisbar:** $r_1 = 1,\ r_2 = 2,\ r_3 = 2$
- f_3 **nur vollständig zuweisbar:** $r_1 = 1,\ r_2 = 1,\ r_3 = 3$

4 Evil Scheduling (22 Punkte)

Im Folgenden finden Sie jeweils zwei Threads eines parallelisierten Programms, das nicht thread-safe ist. Treten Sie an die Stelle des Schedulers, und geben Sie eine Ausführungsreihenfolge der Programmzeilen für jeden Thread an, die fehlerhaftes Verhalten hervorruft.

a) `[8 Punkte]` Bringen Sie Thread 0 dazu, den Fehler (error) auszugeben!

Expandieren Sie zuerst nicht-atomare Instruktionen in mehrere entsprechende atomare Instruktionen. Geben Sie dann die Ausführungsreihenfolge der einzelnen Programmzeilen mit der Notation *Thread.Zeile* an. Zum Beispiel für "Thread 0 führt Zeile 5 aus" schreiben Sie "0.5".

Globale Objekte:

```
int first = 0
int second = 0
```

```
procedure Thread 0
    first++;
    second++;
    if second == 2 and first ≠ 2 then
        error("This should not happen.")
    end if
end procedure
```

```
procedure Thread 1
    first++
    second++;
end procedure
```

Expandierter Code:

```
1: procedure Thread 0
2:
3:
4:
5:
6:
7:
8:     if second == 2 and first ≠ 2 then
9:         error("This should not happen.")
10:    end if
11: end procedure
```

```
1: procedure Thread 1
2:
3:
4:
5:
6:
7:
8: end procedure
```

b) `[14 Punkte]` Bringen Sie beide Threads in den kritischen Abschnitt (critical section)!

Beschreiben Sie dazu den Programmablauf in **Textform**.

Hinweis: Wenn Sie die Aufgabe nicht lösen können, dürfen Sie einzelne Zeilen aus dem Programm streichen. Je mehr Zeilen Sie streichen, desto weniger Punkte können Sie jedoch maximal erreichen. Notieren Sie dazu die gestrichenen Zeilen, wiederum in der Notation *Thread.Zeile*, wie oben.

Globale Objekte:

int `first` = 0
int `second` = 0
semaphore `sem` = 0
semaphore `monitorMutex` = 1
condition variable `cond`, initially empty

```
1:  procedure Thread 0
2:      while true do
3:          first++;
4:          second++;
5:          if first ≠ 2 and second ≠ 2 then
6:              sem.wait(Thread 0)
7:              monitorMutex.wait(Thread 0)
8:              cond.conditionWait(monitorMutex,
    Thread 0)
9:              monitorMutex.signal()
10:             critical_section()
11:         end if
12:     end while
13: end procedure
```

```
1:  procedure Thread 1
2:      while true do
3:          first++;
4:          second++;
5:          if first ≠ 2 and second == 2 then
6:              monitorMutex.wait(Thread 1)
7:              cond.conditionSignal()
8:              monitorMutex.signal()
9:              critical_section()
10:         end if
11:         sem.signal()
12:         first = 0
13:         second = 0
14:     end while
15: end procedure
```

Lösungen

a)

```
1: procedure Thread 0
2:     t = first
3:     t = t + 1
4:     first = t
5:     t = second
6:     t = t + 1
7:     second = t
8:     if second == 2 and first ≠ 2 then
9:         error("This should not happen.")
10:    end if
11: end procedure
```

```
1: procedure Thread 1
2:     t = first
3:     t = t + 1
4:     first = t
5:     t = second
6:     t = t + 1
7:     second = t
8: end procedure
```

Zum Beispiel: 0.2, 0.3, 1.2, 1.3, 0.4, 1.4, 0.5 - 0.8, 1.5 - 1.8, 0.9

b)

```
procedure Thread 0
    while true do
        t = first
        t = t + 1
        first = t
        t = second
        t = t + 1
        second = t
        if first ≠ 2 and second ≠ 2 then
            sem.wait(Thread 0)
            monitorMutex.wait(Thread 0)
            cond.conditionWait(monitorMutex,
Thread 0)
            monitorMutex.signal()
            critical_section()
        end if
    end while
end procedure
```

```
procedure Thread 1
    while true do
        t = first
        t = t + 1
        first = t
        t = second
        t = t + 1
        second = t
        if first ≠ 2 and second == 2 then
            monitorMutex.wait(Thread 1)
            cond.conditionSignal()
            monitorMutex.signal()
            critical_section()
        end if
        sem.signal()
        first = 0
        second = 0
    end while
end procedure
```

Führe Thread 1 aus, bis ans Ende der Schleife, um Semaphor sem auf 1 zu bringen. Führe first++ gleichzeitig aus, so dass es nur um total 1 erhöht wird. Führe second++ in Thread 0 aus und gehe durch den if-Test. Führe second++ in Thread 1 aus und gehe durch den if-Test. Führe Thread 0 aus, bis er auf Bedingung cond wartet. Führe Thread 1 aus, bis zum kritischen Abschnitt, was Thread 0 unlockt. Führe Thread 0 aus, bis zum kritischen Abschnitt.

5 Hashing (20 Punkte)

Zunächst betrachten wir das sogenannte Zufallshashing: Gegeben ist eine zufällige Permutation der Zahlen von 1 bis m. Die Probing-Sequence ist

$$h_i(k) = (h(k) + r_i) \ mod \ m,$$

wobei r_i das i-te Element der Permutation ist.

a) `[4 Punkte]` Leidet Zufallshashing an primärem Clustering?

b) `[4 Punkte]` Leidet Zufallshashing an sekundärem Clustering?

Ein Ingenieur schlägt vor, ein dynamisches Dictionary mit einer rekursiven Probing-Sequence zu implementieren. Konkret schlägt er vor, ein Objekt k in die Zelle $h(k)$ einzufügen, falls diese leer ist, sonst in $h(h(k))$, falls diese leer ist, sonst in $h(h(h(k)))$, falls diese leer ist, etc. Somit wäre die rekursive Probing-Sequence:

$$h_1(k) = h(k)$$
$$h_{i+1}(k) = h(h_i(k)).$$

c) `[6 Punkte]` Wie kann das Löschen in so einem rekursiven Dictionary funktionieren?

d) `[6 Punkte]` Welches gravierende Problem kann beim Einfügen auftreten?

Lösungen

a) Nein. Wenn zwei Probing-Sequences in der gleichen Zelle landen, werden diese nicht erneut in der darauffolgenden Zelle kollidieren da der jeweilige additive Offset unterschiedlich ist bei unterschiedlichem i. Es kommt also nicht zu primärem Clustering.

b) Ja. Wenn zwei Objekte den gleichen Hash-Wert besitzen, werden diese beiden fortwährend der gleichen Probing-Sequence unterliegen da bei beiden die selben r_i addiert werden. Es kommt also zu sekundärem Clustering.

c) Trifft man bei einer Löschoperation auf eine leere Zelle in der rekursiven Probing-Sequence, so ist unklar ob dort ein Wert gelöscht wurde, oder aber niemals ein Wert in dieser Zelle vorhanden war. Dieses Problem tritt auch bei anderen Probing-Sequences auf. Eine Möglichkeit dies dennoch umzusetzen wäre, Felder mit einem „gelöscht“- bzw. „continue“-Flag zu markieren.

d) Beim Einfügen ergibt sich das Problem, dass die rekursive Probing-Sequence möglicherweise nach kurzer Zeit zu einem bereits besuchten Feld zurückkehrt und so ein neuer Wert trotz Kapazität in der Tabelle nicht eingefügt werden kann. Insbesondere sei hier auf die gängige Hashfunktion $h(k) = k \ mod \ p$ hingewiesen, es müsste also bei der Auswahl der Hashfunktion auf diese Eigenschaft geachtet werden.