**Algorithmen für verteilte Systeme** **Universität Salzburg**
**Sommersemester 2019** **FB Computerwissenschaften**

Vorlesung 2: 13. März 2019

*Vortrag: Sebastian Forster* *Mitschrift: Christian Edelmayer*

## 2.1 Leader Election I

### 2.1.1 Motivation

**Leader Election** bedeutet, dass sich bei einem gegebenen Netzwerk die Knoten dieses Netzwerks auf einen einzigen *Leader* einigen müssen. Alle anderen Knoten des Netzwerks werden dann *Followers* genannt. Wir suchen also einen Algorithmus, der für jeden Knoten entscheidet, ob er der Leader ist oder ein Follower. Der Sinn der Leader Election ist es, dass der Leader in einem (dezentralen) Netzwerk Koordinationsaufgaben übernehmen kann.

### 2.1.2 Symmetry Breaking

Es mag einem zwar trivial erscheinen, einen Leader zu bestimmen, doch in einem Modell mit dezentralen Prozessoren (= Knoten) gestaltet sich das nicht so einfach, denn darin ist jeder Knoten gleichwertig und somit eignet sich auch jeder Knoten als Leader. Es gibt also keinen besseren oder schlechteren Leader. Auch gilt, dass eine Lösung nicht eindeutig ist, es gibt $n$ verschiedene Lösungen. Die Schwierigkeit besteht daher eigentlich darin, eine konsistente Entscheidung zu finden, um eine dieser Lösungen auszugeben.

## 2.2 Kommunikationsmodell

In unserem Kommunikationsmodell gibt es Knoten im Netzwerk. Jeder Knoten kennt die Anzahl seiner Nachbarn und kann über nummerierte Ports beliebige Nachrichten an seine Nachbarn senden und von diesen empfangen. Ein Port ist im Prinzip einfach eine Verbindung von einem Knoten zu einem Nachbarknoten. Es gibt folgende zwei Arten, wie die Kommunikation passieren kann.

### 2.2.1 Synchron

Wir werden uns in der Vorlesung mehr auf das Synchrone fokussieren. *Synchron* heißt, wir haben eine rundenbasierte Kommunikation. Jede Runde besteht aus folgenden Schritten:

1. Nachrichten von Nachbarn empfangen (entfällt in erster Runde)
2. Interne Berechnungen
3. Nachrichten an Nachbarn versenden

Es gibt dabei eine globale Uhr, welche den Beginn jeder Runde vorgibt.

### 2.2.2 Asynchron

*Asynchron* bedeutet, ein event-basiertes Modell zu haben. Dabei werden Knoten erst aktiv, wenn sie Nachrichten empfangen. Danach führen sie auch Berechnungen durch und versenden dann Nachrichten. Im asynchronen Modell kann nur garantiert werden, dass jede Nachrichtenübermittlung endliche Zeit benötigt. Es gibt jedoch keine Garantie, dass jede Nachrichtenübermittlung die gleiche Zeiteinheit dauert.

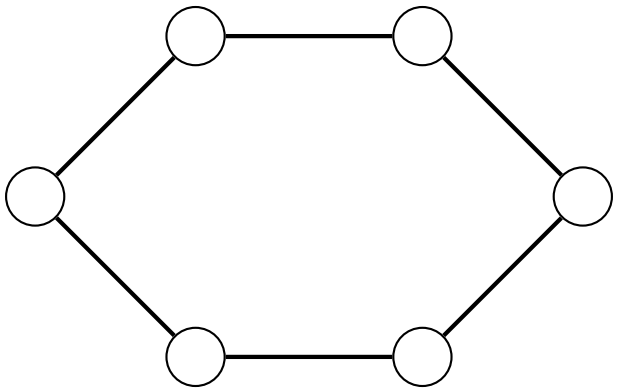
Bezüglich der Komplexitätsmaße interessieren uns die Anzahl der Zeiteinheiten (Runden) und die Anzahl der gesendeten Nachrichten. Dabei ist wichtig, dass wir die interne Berechnungszeit nicht messen, diese wird also nicht analysiert.

### 2.2.3 Varianten

Man kann zwischen einem synchronen und einem asynchronen Modell unterscheiden (siehe oben). Ebenso können Knoten anonym oder benannt sein. Ein Knoten ist benannt, wenn er eine eindeutige ID besitzt (UID). In der Regel sind solche IDs Bit-Strings, oftmals der Länge $O(\log n)$. Die IDs müssen nicht kontinuierlich vergeben sein, es kann auch Lücken geben (es müssen also nicht alle möglichen Zahlenwerte vergeben sein). Weiters kann man noch unterscheiden, ob die Anzahl der Knoten $n$ globales Wissen ist oder nicht. Ist die Anzahl der Knoten $n$ kein globales Wissen, dann nennt man das Modell *uniform*.

### 2.2.4 Leader im Ring

Wir betrachten folgendes Netzwerk (Ring):



Ein *Ring* (-Netzwerk) besteht aus $n$ Knoten und $n$ Kanten, wobei jeder Knoten zwei (Ports zu) Nachbarn hat. Ein Nachbar befindet sich „im Uhrzeigersinn“ (*Clockwise*) und der andere Nachbar befindet sich „gegen den Uhrzeigersinn“ (*Counter-Clockwise*).

## 2.3 Anonyme Netzwerke

In einem *anonymen Netzwerk* haben die Knoten keine IDs. Eine Unterscheidung a priori ist somit nicht möglich. Kurz angemerkt sei, dass es nach einer Leader Election relativ einfach ist, IDs zu vergeben.

Es gibt nun folgendes Theorem:

**Theorem 2.1** ([Angluin '80])**.** *Leader Election in anonymen Netzwerken ist mit deterministischen Algorithmen unmöglich, sogar im synchronen Ring.*

Um dieses Theorem zu beweisen, muss man natürlich erst einmal definieren, was *deterministische Algorithmen* überhaupt sind. Wir definieren einen deterministischen Algorithmus wie folgt:

Ein deterministischer Algorithmus ist beschreibbar durch eine Funktion $f$, welche auf jeden Knoten $v$ angewendet wird. Als **Eingabe** für diese Funktion dient die Anzahl an Knoten $n$, die Anzahl an Ports von $v$ und die gesamte Historie von $v$ (Protokoll aller bisher empfangen Nachrichten eines Knotens mit Zeitpunkt und Eingangsport). Die **Ausgabe** der Funktion sind die zu sendenden Nachrichten des Knotens an Ausgangsports und gegebenenfalls die Entscheidung, ob $v$ Leader oder Follower wird. Anschließend sagen wir, die Funktion löst das Problem, wenn für jedes Netzwerk nach endlich vielen Anwendungen von $f$ ein Knoten als Leader und alle anderen als Follower festgelegt wurden.

Kurz gesagt bedeutet deterministisch hier einfach nur, dass $f$ eine Funktion ist, wobei es zu jeder Eingabe eine eindeutige Ausgabe gibt. Nicht-deterministisch wäre, wenn es zwei oder mehr Ausgaben gäbe.

### 2.3.1 Unmöglichkeits-Beweis

*Beweis.* Wir werden die Unmöglichkeit, sprich das obige Theorem, nun beweisen. Dafür brauchen wir folgende Induktionshypothese, welche im Anschluss an den Unmöglichkeits-Beweis selbst bewiesen wird. Vorerst gehen wir einfach davon aus, dass die Induktionshypothese stimmt (der Beweis dazu folgt später).

**Induktionshypothese:** Für jeden deterministischen Algorithmus sind in jeder Runde jeweils die im und gegen den Uhrzeigersinn empfangenen Nachrichten für alle Knoten gleich.

Aus der Induktionshypothese folgt nun, dass in jeder Runde alle Knoten die gleiche Historie haben. Zudem ist die Anzahl der Ports für jeden Knoten gleich und in jeder Runde ist die Eingabe der Funktion $f$ für alle Knoten gleich. Nachdem die Funktion $f$ deterministisch und die Eingabe für alle Knoten gleich ist, folgt, dass auch die Ausgabe der Funktion $f$ für alle Knoten gleich ist. Wenn $f$ eine Entscheidung trifft, dann muss diese Entscheidung für alle Knoten konsistent sein. Somit bedeutet das also: in jeder Runde macht $f$ entweder alle Knoten zu Leadern, alle Knoten zu Followern oder trifft für keinen Knoten eine Entscheidung. Schlussendlich folgt daraus, dass $f$ also entweder ein falsches Ergebnis liefert oder nicht terminiert, womit das vorher erwähnte Theorem bewiesen wäre (unter der Vorraussetzung, dass die Induktionshypothese gilt). □

### 2.3.2 Induktionsbeweis

Wir werden nun die vorher benutzte Induktionshypothese beweisen.

**Induktionshypothese:** Für jeden deterministischen Algorithmus sind in jeder Runde jeweils die im und gegen den Uhrzeigersinn empfangenen Nachrichten für alle Knoten gleich.

*Beweis. Induktionsbasis:* Gilt trivialerweise, da in der ersten Runde noch keine Nachrichten empfangen werden.

*Induktionshypothese:* Wir nehmen an, die Induktionshypothese gelte für alle Runden $i \leq k$.

*Induktionsschritt:* Wir müssen nun zeigen, dass die Induktionshypothese auch für die Runde $k + 1$ gilt. Nachdem die Induktionshypothese für alle vorherigen Runden gilt, hat jeder Knoten die gleiche Historie. Ebenso sind alle anderen Parameter für alle Knoten gleich (Anzahl der Ports $n$, Anzahl der Knoten $n$), somit ist die Eingabe der Funktion $f$ für alle Knoten gleich. Aus gleicher Eingabe folgt wieder gleiche Ausgabe, also ist die Ausgabe der Funktion $f$ für alle Knoten gleich. Das heißt, alle Knoten senden die jeweils gleiche Nachricht im und gegen den Uhrzeigersinn in der Runde $k + 1$. Aus der Ring-Topologie folgt: in Runde $k + 1$ empfangen alle Knoten jeweils die gleiche Nachricht im und gegen den Uhrzeigersinn. Somit ist Induktionshypothese bewiesen und der vorherige Unmöglichkeits-Beweis vollständig. □

Zusammengefasst bedeutet das, dass sich nie etwas verändern kann, denn ein Knoten kann nie etwas Anderes machen als ein anderer Knoten. Daher ist es auch unmöglich, damit einen Leader zu bestimmen. Es könnten höchstens alle Knoten zu Leadern werden, aber logischerweise wollen wir das nicht. Noch einmal sei erwähnt, dass diese Unmöglichkeit nicht für alle Netzwerke, sondern für anonyme Netzwerke gilt.

## 2.4 Clockwise Algorithmus (Synchron) [LeLann, Chang/Roberts]

Wir nehmen nun an, dass jeder Knoten $v$ einen eindeutigen Identifier $\mathrm{ID}(v)$ hat. Im Umkehrschluss bedeutet das also, dass wir uns nicht mehr in einem anonymen Netzwerk befinden. Noch dazu nehmen wir an, dass wir in einem synchronen Modell sind. Der Algorithmus funktioniert wie folgt:

Jeder Knoten $v$ führt folgenden Algorithmus aus:

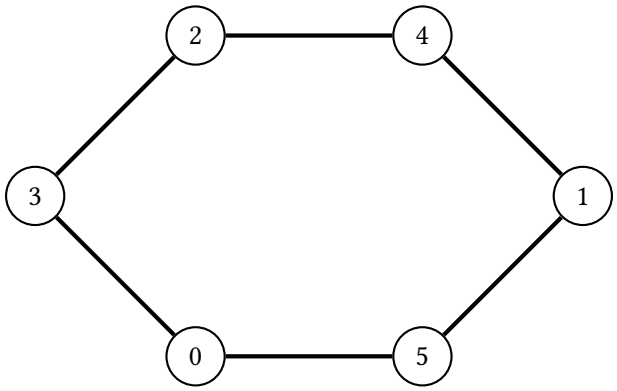
Runde 1:

1 $v$ setzt $T_v := \mathrm{ID}(v)$ (lokale Variable für größte bisher gesehene ID)
2 $v$ sendet $T_v$ an Nachbar im Uhrzeigersinn

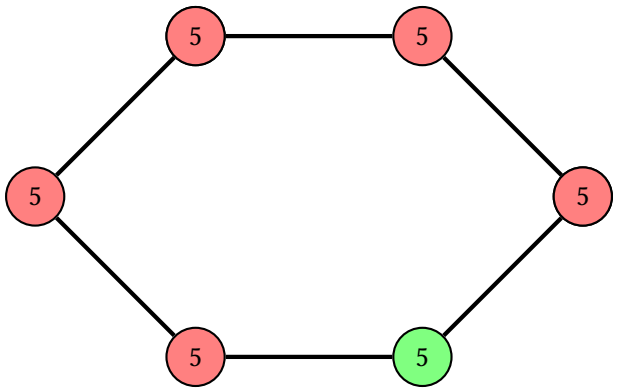
Runde $r \geq 2$:

```
1 if v empfängt Nachricht M then
2     if M > T_v then
3         v setzt T_v := M
4         v entscheidet sich Follower zu sein (sofern nicht bereits vorher geschehen)
5         v sendet T_v an Nachbar im Uhrzeigersinn
6     if M = ID(v) then
7         v entscheidet sich Leader zu sein
```

Wir betrachten folgendes Beispiel:




Die Zahlen sind die IDs der Knoten. Diese sind schon vorgegeben und eindeutig. In der ersten Runde schreiben wir für jeden Knoten die lokale Variable $T_v = \mathrm{ID}(v)$ und senden an den Nachbarn im Uhrzeigersinn eine Nachricht mit dem Inhalt $T_v$. In der nächsten Runde wird empfangen, wobei jeder Knoten, der einen höheren Wert als den eigenen Wert empfangen hat, den höheren (also den empfangenen) Wert übernimmt. Der Knoten links unten, welcher zuvor den Wert 0 hatte, übernimmt den empfangenen Wert 5 beispielsweise. Alle Knoten, die ihren Wert verändert haben, haben sich entschieden, dass sie Follower werden und haben auch ihren Wert verändert. Alle Knoten, die ihren Wert verändert haben, schicken den neuen Wert im Uhrzeigersinn an den Nachbarn. So läuft der Algorithmus weiter und am Ende bekommt der Knoten mit der höchsten ID die Nachricht mit seiner eigenen ID und erkennt, dass er der Leader sein kann. Im Beispiel sieht das am Ende dann so aus:




Rote Knoten sind Follower und der grüne Knoten ist der Leader. Sprich der Knoten, welcher am Anfang die ID 5 hatte, wurde zum Leader (weil 5 die größte ID im Netzwerk war).

### 2.4.1 Laufzeitanalyse Clockwise Algorithmus (Synchron)

**Theorem 2.2.** *Nach höchstens $n + 1$ Runden bestimmt der synchrone Clockwise Algorithmus den Knoten mit der höchsten ID zum Leader und alle anderen zu Followern.*

Wir wollen die Beobachtung, dass die höchste ID in jeder Runde weitergeleitet wird, formalisieren.

*Beweis.* Sei $z$ der Knoten mit der (initial) höchsten ID. Für jedes $i \geq 0$ sei $v_i$ der Knoten, der von $z$ aus nach Traversieren von $i$ Kanten im Uhrzeigersinn erreicht wird. Anders ausgedrückt ist der Knoten $v_i$ der Knoten, welchen man von dem Knoten $z$ aus erreichen kann, wenn man $i$ Kanten im Uhrzeigersinn genommen hat. $v_0$ wäre beispielweise der Knoten selbst, da man 0 Kanten nimmt. $v_1$ wäre der Nachbar von $z$ im Uhrzeigersinn und so weiter. Wir brauchen nun folgende Induktionshypothese.

**Induktionshypothese:** In Runde $i + 1$ empfängt $v_i$ die Nachricht ID($z$) (für jedes $i \leq n$).

Wir werden an dieser Stelle die Induktionshypothese nicht beweisen, der Beweis dazu ist im Prinzip ein Standard-Induktionsbeweis und auch sehr ähnlich zu dem Beweis der vorherigen Induktionshypothese. Aus der Induktionshypothese folgt: in Runde $n + 1$ empfängt $z$ seine eigene ID und wird zum Leader. Falls $v \neq z$, dann ist $v = v_i$ für ein $i \leq n - 1$. Spätestens in Runde $i + 1 \leq n$ empfängt $v_i$ die Nachricht ID($z$) und wird zum Follower.

Somit folgt also eine maximale Laufzeit von $n + 1$ Runden und gleichzeitig folgt auch, dass der Algorithmus korrekt ist. □

### 2.4.2 Nachrichtenkomplexität Clockwise Algorithmus

**Theorem 2.3.** *Der Clockwise Algorithmus versendet insgesamt höchstens $n^2$ Nachrichten.*

Wann sendet der Algorithmus? Er sendet nur dann, wenn etwas mit der Variable $T_v$ passiert. Sprich Knoten $v$ sendet nur, nachdem $T_v$ initialisiert oder erhöht wurde. Man kann leicht beobachten, dass jeder von $T_v$ angenommene Wert einer ID im Netzwerk entspricht (ganz formal gesehen müsste man dazu wieder eine passende Induktionshypothese aufstellen). Daher kann $T_v$ während des Algorithmus höchstens $n$ verschiedene Werte annehmen, da es $n$ verschiedene Knoten gibt. Somit sendet jeder Knoten höchstens $n$ Nachrichten. Wenn jeder Knoten höchstens $n$ Nachrichten versendet und es höchstens $n$ Knoten gibt, dann haben wir eine Nachrichtenkomplexität von $n^2$. Die Frage ist nun, geht es besser als $n^2$ Nachrichten?

## 2.5 Clockwise Algorithmus (Asynchron) [LeLann, Chang/Roberts]

Der *Clockwise Algorithmus* kann auch als asynchroner Algorithmus verstanden werden. Es gilt wieder die Annahme, dass jeder Knoten $v$ einen eindeutigen Identifier ID($v$) hat. Im asynchronen Modell betrachten wir keine Runden, sondern Events. Es gibt zwei mögliche Events: Initialisierung oder Empfang einer Nachricht. Ein (externes) Initialisierungs-Event ist notwendig, damit der Algorithmus gestartet wird, dabei können auch mehrere Knoten gleichzeitig intialisiert werden. Der Algorithmus sieht wie folgt aus:

Jeder Knoten $v$ führt folgenden Algorithmus aus:

```
1  if v wird initialisiert then
2      v setzt T_v := ID(v) (lokale Variable für größte bisher gesehene ID)
3      v sendet T_v an Nachbar im Uhrzeigersinn
4  if v empfängt Nachricht M then
5      if M > T_v oder T_v uninitialisiert then
6          v setzt T_v := M
7          v entscheidet sich Follower zu sein (sofern nicht bereits vorher geschehen)
8          v sendet T_v an Nachbar im Uhrzeigersinn
9      if M = ID(v) then
10         v entscheidet sich Leader zu sein
```

### 2.5.1 Analyse Clockwise Algorithmus (Asynchron)

**Theorem 2.4.** *Nach höchstens $2n - 1$ Zeiteinheiten bestimmt der asynchrone Clockwise Algorithmus den Knoten mit der höchsten ID zum Leader und alle anderen zu Followern (wenn die Übermittlung jeder Nachricht höchstens eine Zeiteinheit dauert).*

**Beweisidee (nicht ganz formal):**

Die Zeitmessung startet mit dem ersten Knoten, der initialisiert wird. Wenn der Knoten initialisiert wurde, dann wird er auf jeden Fall eine Nachricht (seine ID) verschicken. Das heißt, er weckt den Nachbarknoten auf. Somit wird der Nachbarknoten aktiv und das gegenseitige Aktivieren setzt sich fort. Dann vergehen höchstens $n - 1$ Zeiteinheiten bis der Knoten $z$ mit der höchsten ID eine Nachricht empfängt und das erste Mal aktiv wird. Wenn das erste Mal der Knoten mit der höchsten ID aktiv wurde, dann ist es das gleiche Argument wie vorher: nach höchstens $n$ weiteren Zeiteinheiten hat jeder Knoten $\mathrm{ID}(z)$ empfangen und sich entschieden.

## 2.6 Verbesserung der Nachrichtenkomplexität

Mit $n$ Runden sind wir zufrieden, aber $\Theta(n^2)$ versendete Nachrichten ist nicht ganz optimal. Man kann sich also die Frage stellen, ob man einen Leader mit signifikant weniger als $\Theta(n^2)$ Nachrichten bestimmen kann. Ja, das geht. Und die Idee dahinter ist, dass wir es ausnutzen, dass wir in einem synchronen System sind. Es geht darum, dass es als Zustimmung interpretiert werden darf, wenn *keine* Nachricht gesendet wird.

### 2.6.1 Nachrichteneffiziente Leader Election

Der Algorithmus dazu sieht wie folgt aus:

Jeder Knoten $v$ führt folgenden Algorithmus in jeder Runde aus:

**if** *Leader-Nachricht empfangen* **then**
    Werde zum Follower
    Leite Leader-Nachricht im Uhrzeigersinn weiter
**if** *#Runden* $= \mathrm{ID}(v) * n + 1$ *und* $v$ *noch kein Follower* **then**
    Werde zum Leader
    Sende Leader-Nachricht an Nachbar im Uhrzeigersinn

Ein Knoten wird also dann aktiv, wenn die aktuelle Runde gleich seiner $\mathrm{ID}(v) * n + 1$ ist. Das *+ 1* ist eine Korrektur, weil die Runden bei 1 zu zählen beginnen und die IDs bei 0. Falls dem so ist, kann der Knoten zum Leader werden und eine Nachricht an seine Nachbarn im Uhrzeigersinn senden, dass ein Leader bestimmt wurde. Diese Nachricht wird in den folgenden Runden weitergeleitet und jeder Knoten, der die Nachricht empfängt, wird zum Follower.

### 2.6.2 Analyse

Wir unterteilen die Runden in Phasen der Länge $n$ und in jeder Phase (also alle $n$ Runden) entscheidet sich höchstens ein Knoten, Leader zu werden. Der Leader muss also innerhalb von $n$ Runden alle anderen Knoten informieren, damit der Algorithmus korrekt ist (deshalb auch die Formel $\#Runden = \mathrm{ID}(v) * n + 1$). Das wird durch das Weiterleiten der Nachricht im Ring garantiert. Sobald ein Knoten entscheidet, Leader zu sein, wissen wir, dass vor Beginn der nächsten Phase alle anderen Knoten zu Followern geworden sind. Daraus folgt, dass die Anzahl der Nachrichten $O(n)$ ist (es gilt sogar, dass es maximal $n$ Nachrichten sind, denn jeder Knoten sendet nur einmal). Die Anzahl der Runden ist $\Theta(n \cdot \min_v \mathrm{ID}(v))$ und gibt im Prinzip an, wie lange wir hochzählen müssen, um eine ID, die im Netzwerk vergeben wurde, zu erreichen (0 muss nicht die erste ID sein). Es gibt keine Garantie, dass die IDs mit 0 starten. Wir haben in der Regel Bit-Strings als IDs.

**Beispiel:** Für Bit-Strings der Länge $2\lceil \log n \rceil$ als IDs könnte die niedrigste ID Größe $2^{\log n}$ haben. Dann werden $\Theta(n 2^{\log n}) = \Theta(n^2)$ Runden benötigt.

## Literatur

[1] Nancy A. Lynch (1996) *Distributed Algorithms*, Kapitel 3, Morgan Kaufmann.