Byzantinische Fehler

Matthias Reumann

10. April 2020

Inhaltsverzeichnis

1 Byzantinische Fehler

1.1 Definition

Byzantinische Fehler treten in verteilten Systemen auf, wenn fehlerhafte Nachrichten von einem Knoten versendet werden, die von der Struktur und vom Inhalt nicht von fehlerfreien unterscheidet werden können. [2]

Den Namen hat diese Fehlerklasse im Jahr 1982 von Lamport et. al. im Papier *The Byzantine Generals Problem* erhalten.[1] In diesem werden Byzantinische Fehler durch das abstrakte Beispiel der Byzantinischen Generäle näher erklärt.

1.2 Anwendungen

Da Fehlentscheidungen katastrophale Folgen hätten, wird in Boeing Flugzeugsystemen Byzantinischen Fehlern eine hohe Bedeutung zugewiesen. Das Raumfahrtunternehmen SpaceX beachtet byzantinische Fehler in ihren Dragon Raumschiffen, welche Astronauten zur ISS transportieren sollen. Um die Integrität der Blockchain zu gewährleisten, ist die Implementierung von Bitcoin Byzantinische Fehler Tolerant (BFT).

2 Problem der byzantinischen Generäle

Mehrere Generäle und ihre Truppen umzingeln eine feindliche Stadt. Da diese voneinander räumlich getrennt sind können sie nur über Boten miteinander kommunizieren. Jeder General beobachtet den gemeinsamen Feind und schlussendlich muss eine gemeinsame Entscheidung getroffen werden um den Feind zu besiegen. Jedoch könnten unter den Generälen Verräter sein, deren Ziel es ist eine gemeinsame Entscheidung der loyalen Generäle zu verhindern.

Die Generäle müssen einen Algorithmus finden der garantiert, dass

(a) Alle loyalen Generäle die gleiche Entscheidung treffen

(b) Eine geringe Anzahl an Verrätern die loyalen Generäle nicht zu einer Fehlentscheidung führt

Eine Entscheidung unter den Generälen wird gefällt indem jeder General seine Informationen an die anderen Generäle weitergibt. Die Nachricht gesendet vom i.ten General wird als *v(i)* bezeichnet. Jeder General erhält die Nachrichten *v(1)* bis *v(n)*, wobei n die Anzahl der Generäle darstellt.
Die finale Entscheidung wird durch die absolute Mehrheit der Werte dieser Nachrichten bestimmt. In diesem konkreten Beispiel können die Werte entweder *ANGRIFF* oder *RÜCKZUG* sein.
Allerdings könnte es sein, dass ein Verräter unterschiedliche Werte zu unterschiedlichen Generälen sendet, was widerum bedeutet, dass loyale Generäle unterschiedliche *v(i)* Werte besitzen.

Deshalb müssen sogennante *interactive consistency conditions* gelten. Ein führender General, auch Commander genannt, sendet einen Befehl an seine $n-1$ Leutnant Generäle so, dass:

IC1 Alle loyalen Lieutenants den gleichen Befehl ausführen
IC2 Wenn der Commander loyal ist, dann befolgt jeder loyale Leutnant seinen Befehl

2.1 Beispiel mit drei Generälen

In diesem Abschnitt wird eine Situation beschrieben in der drei Generäle miteinander kommunzieren. Wichtig dabei ist, dass niemand der loyalen Generäle weiß wer der Verräter ist.

In Abbildung 1 ist Lieutenant 2 (L2) der Verräter. Der loyale Commander sendet den Befehl *ANGRIFF* an beide Lieutenant L1 und L2. Zur Verifizierung sendet L2 L1 die Nachricht, er habe den Befehl *RÜCKZUG* erhalten. L1 hat nun die Nachrichten *ANGRIFF* und *RÜCKZUG* und kann auf Grund dessen keine eindeutige Entscheidung treffen. Eine mögliche Lösung wäre immer die Entscheidung des Commanders zu übernehmen.

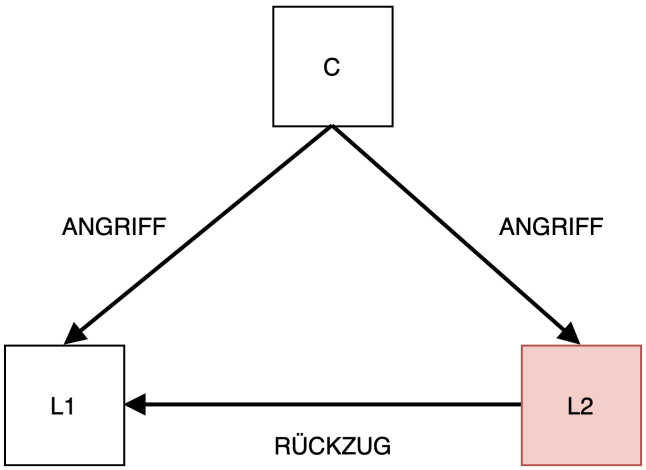

Abbildung 1: L2 ist der Verräter. L1 muss den Befehl des Commanders auführen um die IC2 einzuhalten.

Im nächsten Szenario (Abbildung 2) ist der Commander der Verräter. Dieser sendet unterschiedliche Befehle an L1 und L2. Für L1 ist es die exakt selbe Situation wie in Abbildung 1. L2 steht vor dem selben Problem. Wird hier jedoch die gleiche Argumentation wie bei L1 angewandt, führen L1 und L2 verschiedene Befehle aus.

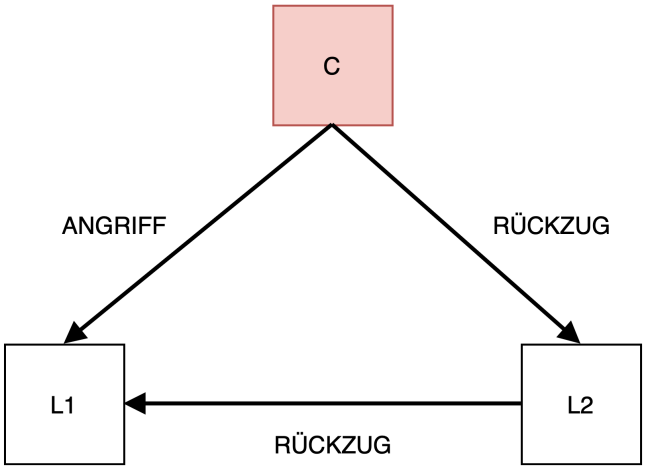

Abbildung 2: Der Commander ist der Verräter. Für L1 ist es die selbe Situtation wie in Abbildung 1.

Wie aus diesem Beispiel hervorgeht, existiert keine Lösung für drei Generäle mit einem Verräter. Um einen byzantinischen Fehler zu erkennen muss daher folgende Eigenschaft gelten: $n \geq 3m + 1$, wobei n die Anzahl der Generäle und m die Anzahl der Verräter in diesem System ist. Mit anderen Worten, mehr als zwei Drittel müssen loyal sein, um einen byzantinischen Fehler zu erkennen.

2.2 Mögliche Lösungen

Eine mögliche Lösung des Problems ist das Hinzufügen eines weiteren Generals. Dadurch kann, wenn jeder Leutnant seine erhaltene Nachricht an die anderen Leutnants sendet, genau ein byzantinischer Fehler erkannt werden.

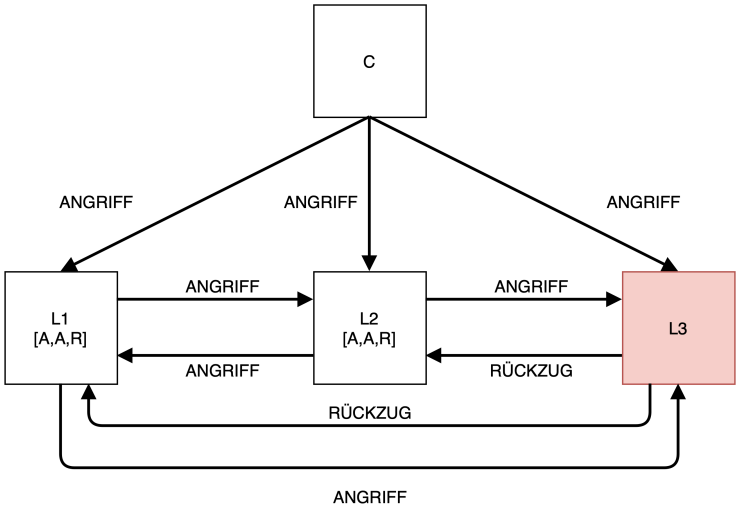

Abbildung 3: Zeigt, dass durch das Hinzufügen eines vierten Generals ein byzantinischer Fehler erkennt werden kann.

Desweiteren, wäre der Einsatz von signierten Nachrichten eine andere valide Lösung. Dadurch könnte im Beispiel der Abbildung 1 festgestellt werden, dass L2 die Nachricht verfälscht hätte.

3 Anleitung zum C++ Beispiel

In diesem Abschnitt wird das in Abbildung 1 und 2 dargestellte Problem mithilfe einer C++ Anwendung demonstriert.

3.1 Ein Lieutenant ist der Verräter

Um das in Abbildung 1 dargestellte Problem mit der Anwendung zu demonstrieren, müssen die Programme *commander* und *lieutenent* wie folgend aufgerufen werden.

Zunächst muss das *lieutenent*-Programm zweimal gestartet werden.

```
$ ./lieutenant -p 10000 -n 10001
[2020-04-10 09:53:01.043] [info] Waiting for order...

...

$ ./lieutenant -t -p 10001 -n 10000
[2020-04-10 09:53:01.043] [info] Waiting for order...
```

Der Parameter *-p* definiert den Port des Lieutenants und *-n* den des benachbarten Lieutenants. Um den Lieutnant als Verräter zu starten wird bei L2 das Flag *-t* übergeben.

In einem echten Netzwerk, z.B. bestehend aus einem Sensor würde, falls ein Threshold unter- oder überschritten wird, der Sensor, der diese Veränderung als erstes aufgenommen hat, die Rolle des Commanders übernehmen. Um dieses Verhalten zu simulieren, wird das *commander*-Programm ausgeführt. Dabei stellen die Parameter *-1* und *-2* die Ports der Lieutenents L1 und L2 dar.

```
$ ./commander -1 10000 -2 10001
[2020-04-10 10:05:08.067] [info] Sending orders to L1 at :10000 and L2 at :10001
[2020-04-10 10:05:08.068] [info] Sending order 'ANGRIFF' to L1
[2020-04-10 10:05:08.068] [info] Sending order 'ANGRIFF' to L2
```

Wie in Abbildung 1 dargestellt, sendet der loyale Commander den Befehl *ANGRIFF* an beide Lieutenants.

Lieutenant L1 verhält sich dabei exact wie in Abbildung 1 dargestellt. Die Problematik dabei ist, ob sich für den Befehl *ANGRIFF* oder *RÜCKZUG* entschieden wird, wie in 2 beschrieben wurde.

```
[2020-04-10 09:53:01.043] [info] Waiting for order...
[2020-04-10 10:05:08.072] [info] Commander sent 'ANGRIFF' order
[2020-04-10 10:05:08.072] [info] Send 'ANGRIFF' order to neighbour
[2020-04-10 10:05:08.072] [info] Waiting for neigbour's order...
```

```
[2020-04-10 10:05:08.075] [info] Neighbours order is 'RÜCKZUG'
[2020-04-10 10:05:08.075] [info] Picking the majority of:
[2020-04-10 10:05:08.075] [info] ANGRIFF
[2020-04-10 10:05:08.075] [info] RÜCKZUG
```

3.2 Der Commander ist der Verräter

Das Starten der Programme ist analog zum vorherigen Abschnitt, jedoch mit dem Unterschied, dass das Flag *-t* nun anstatt bei einem der Lieutenants, sondern beim Starten des Commanders übergeben werden muss.

```
$ ./commander -1 10000 -2 10001 -t
[2020-04-10 10:20:22.792] [info] Sending orders to L1 at :10000 and L2 at :10001
[2020-04-10 10:20:22.793] [info] Commander is a traitor
[2020-04-10 10:20:22.793] [info] Sending order 'ANGRIFF' to L1
[2020-04-10 10:20:22.793] [info] Sending order 'RÜCKZUG' to L2
```

Wie hier beobachtet werden kann, sendet der Commander unterschiedliche Befehle zu L1 und L2.

L1 beobachtet keine Veränderung.

```
$ ./commander -1 10000 -2 10001 -t
[2020-04-10 10:20:17.134] [info] Waiting for order...
[2020-04-10 10:20:22.796] [info] Commander sent 'ANGRIFF' order
[2020-04-10 10:20:22.796] [info] Send 'ANGRIFF' order to neighbour
[2020-04-10 10:20:22.796] [info] Waiting for neigbour's order...
[2020-04-10 10:20:22.799] [info] Neighbours order is 'RÜCKZUG'
[2020-04-10 10:20:22.799] [info] Picking the majority of:
[2020-04-10 10:20:22.799] [info] ANGRIFF
[2020-04-10 10:20:22.799] [info] RÜCKZUG
```

Literatur

[1] Lamport et. al. *The Byzantine Generals Problem*. URL: https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf. (Zugegriffen am: 06.04.2020).

[2] Esra Unal. *Seminararbeit Byzantinische Fehler*. URL: http://www.mi.fu-berlin.de/inf/groups/ag-tech/teaching/2010_SS/S_19510b_Proseminar_Technische_Informatik/esra-uenal-report.pdf. (Zugegriffen am: 06.04.2020).