Loom Network
Distributed Enterprise Platform for Healthcare Providers
Follow
7.8K

Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance

Georgios Konstantopoulos Follow
Dec 1, 2017 · 7 min read

Listen to this article
Powered by Play.ht
01:54 / 08:21
Speed

Articles in this Understanding Blockchain Fundamentals series:

1. Part 1: Byzantine Fault Tolerance 👈
2. Part 2: Proof of Work & Proof of Stake
3. Part 3: Delegated Proof of Stake

* * *

Blockchains are inherently decentralized systems which consist of different actors who act depending on their incentives and on the information that is available to them.

Whenever a new transaction gets broadcasted to the network, nodes have the option to include that transaction to their copy of their ledger or to ignore it. When the majority of the actors which comprise the network decide on a single state, **consensus** is achieved.

> *A fundamental problem in distributed computing and multi-agent systems is to achieve overall system reliability in the presence of a number of faulty processes. This often requires processes to agree on some data value that is needed during computation.*[1]

These processes are described as consensus.

- What happens when an actor decides to not follow the rules and to tamper with the state of his ledger?
- What happens when these actors are a large part of the network, but not the majority?

In order to create a secure consensus protocol, it must be *fault tolerant.*

Firstly, we will talk briefly about the unsolvable Two Generals Problem. Then we will extend that to the *Byzantine Generals' Problem and discuss Byzantine Fault Tolerance* in distributed and decentralized systems. Finally, we will discuss how all this relates to the blockchain space.

* * *

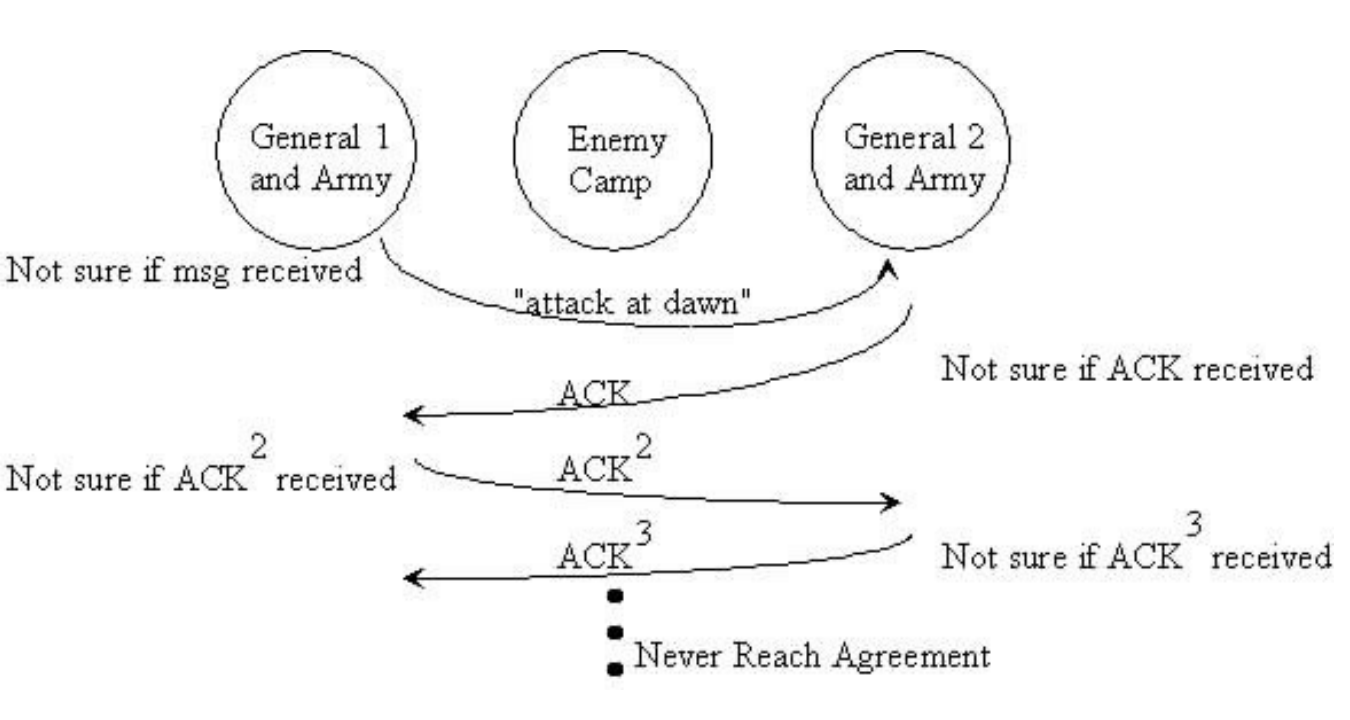
The Two generals Problem

This problem (first published in 1975 and given its name in 1978) describes a scenario where two generals are attacking a common enemy. General 1 is considered the leader and the other is considered the follower. Each general's army on its own is not enough to defeat the enemy army successfully, thus they need to cooperate and attack at the same time. This seems like a simple scenario, but there is one caveat:

In order for them to communicate and decide on a time, General 1 has to send a messenger across the enemy's camp that will deliver the time of the attack to General 2. However, there is a possibility that the messenger will get captured by the enemies and thus the message won't be delivered. That will result in General 1 attacking while General 2 and his army hold their grounds.

Even if the first message goes through, General 2 has to *acknowledge (ACK,* notice the similarity to the 3-way handshake of TCP *)* that he received the message, so he sends a messenger back, thus repeating the previous scenario where the messenger can get caught. This extends to infinite ACK's and thus the generals are unable to reach an agreement.

> *There is no way to guarantee the second requirement that each general be sure the other has agreed to the attack plan. Both generals will always be left wondering whether their last messenger got through.*



Since the possibility of the message not getting through is always > 0, the generals can never reach an aggrement with 100% confidence.

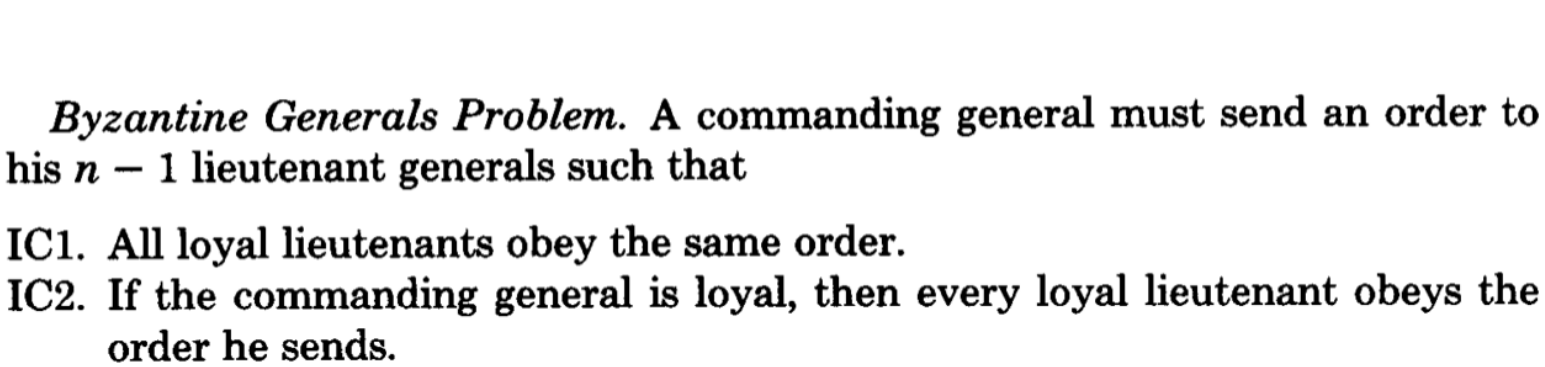
The Two Generals Problem has been proven to be **unsolvable.**

* * *

The Byzantine Generals Problem

Famously described in 1982 by Lamport, Shostak and Pease, it is a generalized version of the Two Generals Problem with a twist. It describes the same scenario, where instead more than two generals need to agree on a time to attack their common enemy. The added complication here is that one or more of the generals can be a traitor, meaning that they can lie about their choice (e.g. they say that they agree to attack at 0900 but instead they do not).

The leader-follower paradigm described in the Two Generals Problem is transformed to a commander-lieutenant setup. In order to achieve consensus here, the commander and every lieutenant must agree on the same decision (for simplicity *attack* or *retreat).*



Byzantine Generals Problem. **A commanding general must send an order to his $n - 1$ lieutenant generals such that**

IC1. All loyal lieutenants obey the same order.
IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

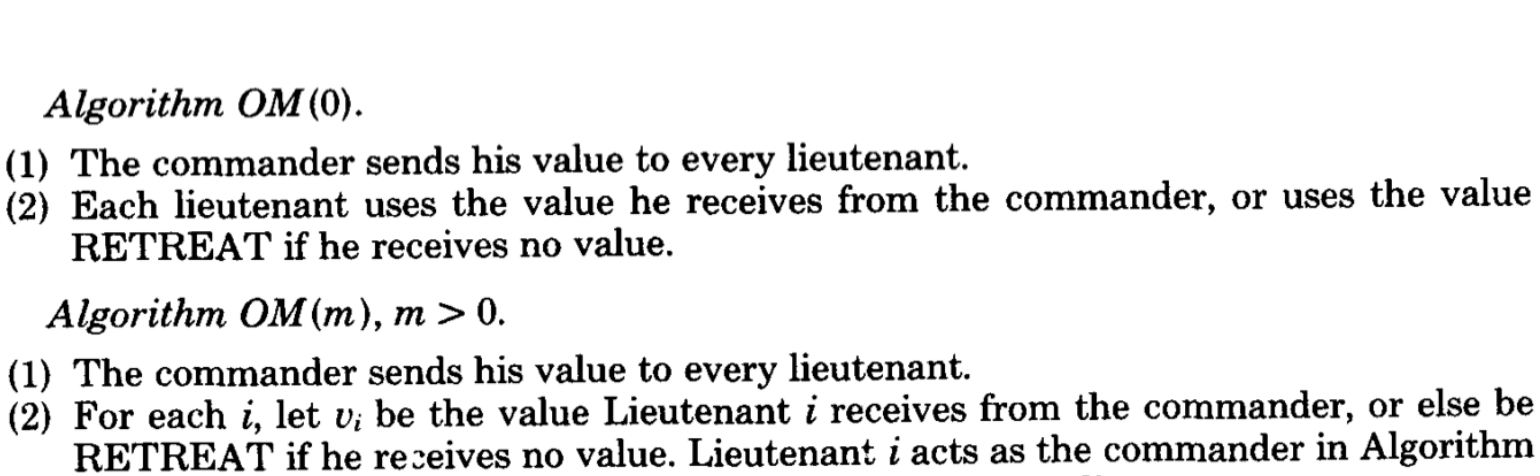
page 3, The Byzantine Generals Problem

Adding to IC2., it gets interesting that if the commander is a traitor, consensus must still be achieved. As a result, all lieutenants take the majority vote.

The algorithm to reach consensus in this case is based on the value of majority of the decisions a lieutenant observes.

> **Theorem:** For any *m*, Algorithm *OM(m)* reaches consensus if there are more than *3m* generals and at most *m* traitors.

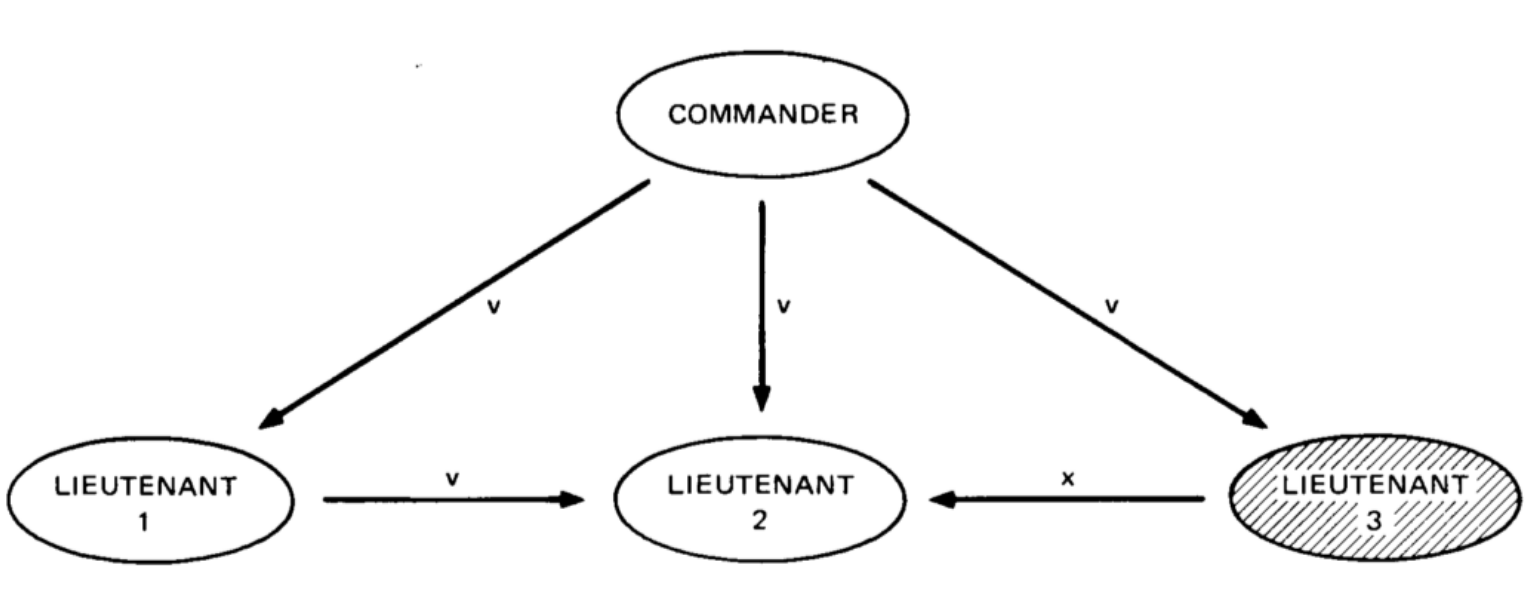
This implies that the algorithm can reach consensus as long as 2/3 of the actors are honest. If the traitors are more than 1/3, consensus is not reached, the armies do not coordinate their attack and the enemy wins.



```
Algorithm OM(0).
(1) The commander sends his value to every lieutenant.
(2) Each lieutenant uses the value he receives from the commander, or uses the value
    RETREAT if he receives no value.
Algorithm OM(m), m > 0.
(1) The commander sends his value to every lieutenant.
(2) For each i, let v_i be the value Lieutenant i receives from the commander, or else be
    RETREAT if he receives no value. Lieutenant i acts as the commander in Algorithm
    OM(m - 1) to send the value v_i to each of the n - 2 other lieutenants.
(3) For each i, and each j ≠ i, let v_j be the value Lieutenant i received from Lieutenant j
    in step (2) (using Algorithm OM(m - 1)), or else RETREAT if he received no such
    value. Lieutenant i uses the value majority(v_1, . . . , v_{n-1}).
```

m = 0 → no traitors, each lieutenant obeys | m > 0 → each lieutenant's final choice comes from the majority of all lieutenant's choices

This should be more clear with a visual example from Lieutentant 2's point of view— Let C be Commander and L{i} be Lieutenant i:



OM(1): Lieutenant 3 is a traitor — L2 point of view

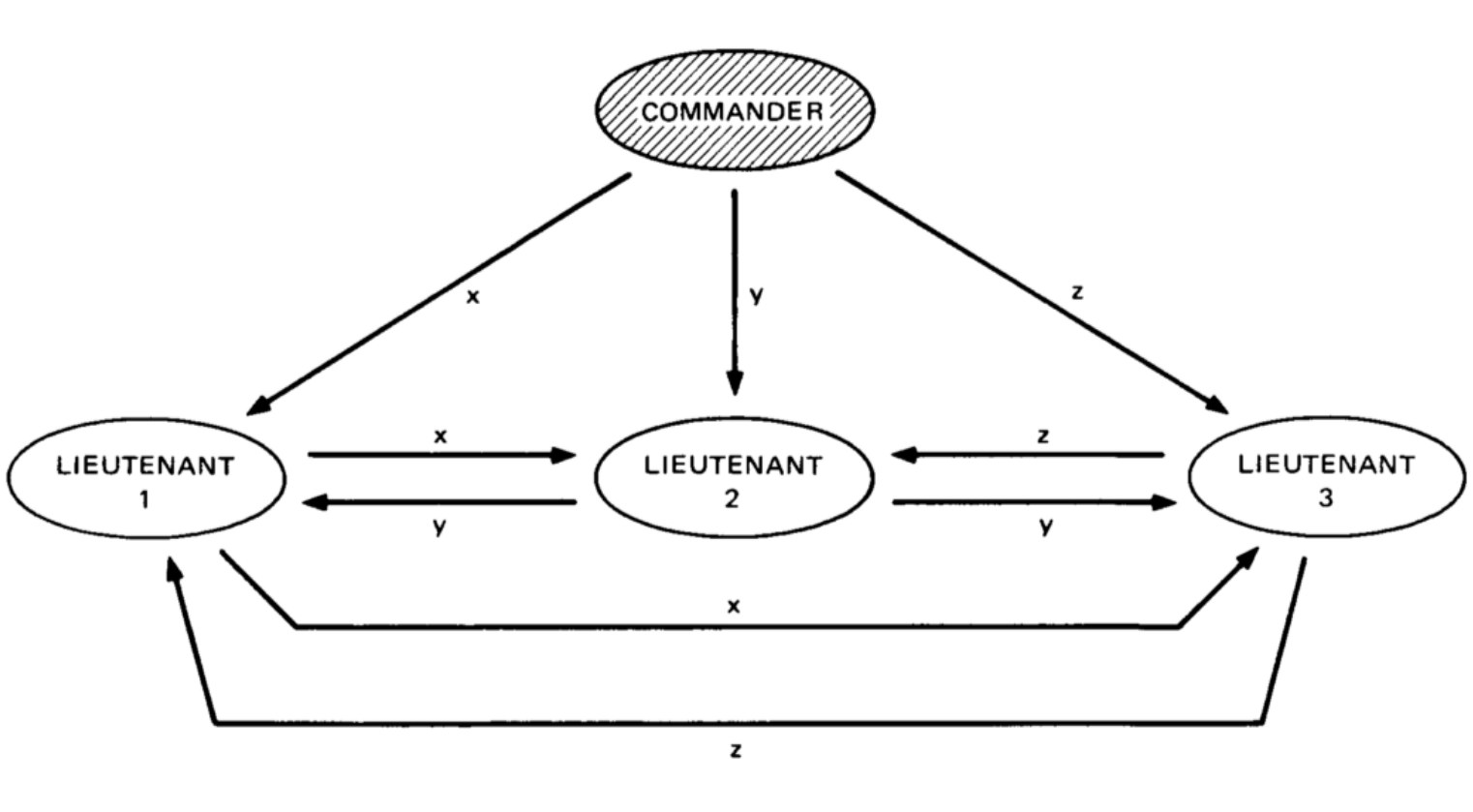
Steps:

1. Commander sends *v* to all Lieutenants
2. L1 sends *v* to L2 | L3 sends *x* to L2
3. L2 ← *majority(v,v,x)* == *v*

The final decision is the majority vote from L1, L2, L3 and as a result consensus has been achieved

> *The important thing to remember is that the goal is for the majority of the lieutenants to choose the* ***same*** *decision, not a specific one.*

Let's examine the case of the commander being a traitor:



OM(1): Commander is a traitor

Steps:

1. Commander sends *x, y, z* to L1, L2, L3 respectively
2. L1 sends *x* to L2, L3 | L2 sends *y* to L1, L3 | L3 sends *z* to L1, L2
3. L1 ← majority(x,y,z) | L2 ← majority(x,y,z) | L3 ← majority(x,y,z)

They all have the same value and thus consensus is reached. Take a moment here to reflect that even if *x, y, z* are all different the value of *majority(x, y, z)* is the same for all 3 Lieutenants. In the case x,y,z are totally different commands, we can assume that they act on the default option *retreat*.

For a more hands-on approach and a more complex example with 7 generals and 2 traitors, I suggest you read this article.

* * *

Byzantine Fault Tolerance

Byzantine Fault Tolerance is the characteristic which defines a system that tolerates the class of failures that belong to the Byzantine Generals' Problem. Byzantine Failure is the most difficult class of failure modes. It implies no restrictions, and makes no assumptions about the kind of behavior a node can have (e.g. a node can generate any kind of arbitrary data while posing as an honest actor).

Top highlight

Byzantine Faults are the most severe and difficult to deal with. Byzantine Fault Tolerance has been needed in airplane engine systems, nuclear power plants and pretty much any system whose actions depend on the results of a large amount of sensors. Even SpaceX was considering it as a potential requirement for their systems.

The algorithm mentioned in the previous section is Byzantine Fault Tolerant as long as the number of traitors do not exceed one third of the generals. Other variations exist which make solving the problem easier, including the use of digital signatures or by imposing communication restrictions between the peers in the network.

* * *

How does this all relate to blockchain?

Blockchains are decentralized ledgers which, by definition, are not controlled by a central authority. Due to the value stored in these ledgers, bad actors have huge economic incentives to try and cause faults. That said, Byzantine Fault Tolerance, and thus a solution to the Byzantine Generals' Problem for blockchains is much needed.

In the absence of BFT, a peer is able to transmit and post false transactions effectively nullifying the blockchain's reliability. To make things worse, there is no central authority to take over and repair the damage.

The big breakthrough when Bitcoin was invented, was the use of Proof-of-Work as a probabilistic solution to the Byzantine Generals Problem as described in depth by Satoshi Nakamoto in this e-mail.

* * *

Conclusion

In this article, we discussed some basic concepts of consensus in distributed systems.

In the next article, we will discuss and compare some the algorithms that are used in blockchains in order to achieve Byzantine Fault Tolerance.

Subscribe below to get notified when it goes live!

* * *

Loom Network *is the multichain interop platform for scaling high-performance dapps — already live in production, audited, and battle-tested.*

Deploy your dapp to Loom's Basechain once and reach the widest possible user base across all major blockchains today.

New to Loom? ***Start here***.

Want to stake your LOOM tokens and help secure Basechain? ***Find out how***.

Like what we're doing here? Stay in the loop by ***signing up for our private mailing list***.

Thanks to James Martin Duffy.

Cryptocurrency Blockchain Bitcoin Ethereum Algorithms

7.8K claps

WRITTEN BY
Georgios Konstantopoulos
Follow
Independent consultant & engineer focused on blockchain scalability, infosec and mechanism design. That Layer 2 guy. Aspiring Rustacean. me@gakonst.com

Loom Network
Follow
Building a distributed enterprise platform for managing data— focused on healthcare providers and government departments.

See responses (14)