Byzantine Fault Tolerance And Blockchain

Distributed Systems
Nicola Santolini

29/08/2018

1 Introduction

In the last few years topics like Blockchain, cryptocurrency, Bitcoin and Ethereum have become very popular and have catalyzed global interest. Even if the publication of the Bitcoin protocol by Satoshi Nakamoto dates back to 2008 it is closely related to an older and classic problem in information technology described by Leslie Lamport, Robert Shostak and Marshall Pease in their 1982 paper, "The Byzantine Generals Problem" [1]. In the following pages a description and analysis of the relationship between topics like the Byzantine Generals Problem, Byzantine Fault Tolerance, consensus and Blockchain is presented.

2 Consensus

Before talking about the classical Byzantine General Problem it's important to specify what consensus is. In information technology reaching an agreement among many distributed processes (about the value of a property, an action to do, etc.) it's a well-known and studied problem. On an asynchronous network in presence of even a single process that can fail/die without forewarning, the consensus problem is impossible to solve. Working under the assumption of synchrony is possible. Different fault model can be defined:

- *crash model*, where a fault corresponds to a processor halting
- *crash plus link model*, where either a processor may crash or a link can go down
- *omission mode*l, where a process fails either by sending only a subset of the required messages or receiving only a proper subset of the messages that have been sent to it
- *Byzantine failure model*, where a process can fail by adopting an arbitrary behaviour

The problem of reaching consensus under Byzantine faults is known as the Byzantine General Agreement (BGA) problem.

3 The Byzantine Generals Problem

3.1 The Two Generals Problem

This problem (first published in 1975 [2] and given its name in 1978) could be considered the precursor of the generic Byzantine Generals Problem. It 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, so they need to cooperate and attack at the same time. The main problem is that the communication has constraints. General 1 has to send a messenger across the enemy camp that will deliver the time of the attack to General 2 in order to decide on a time. However, there is a possibility that the messenger will get captured by the enemies and then the message won't be delivered. That will result in General 1 attacking while General 2 and his army don't. Even if the first message goes through, General 2 has to acknowledge (with an ACK that is quite similar 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 then 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 without knowing whether their last messenger get captured or not. Since the possibility of the message not getting through is always grater than 0, the generals can never reach an agreement with 100% confidence. The Two Generals Problem has been proven to be unsolvable.

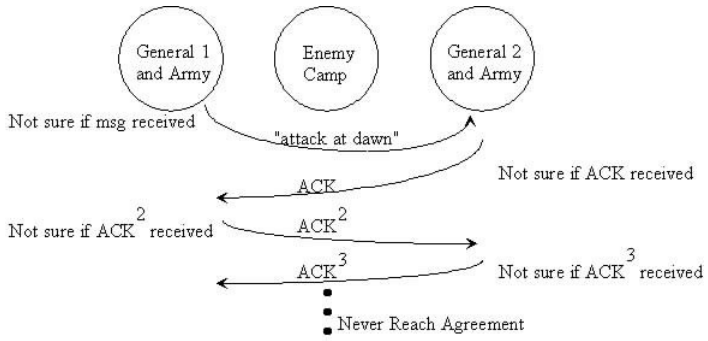

Figure 1: Two Generals Problem representation

3.2 Byzantine Generals Problem

It was described in 1982 in a famous paper by Lamport, Shostak and Pease and it is a generalized version of the Two Generals Problem. The military battle is an expedient used to abstract a concrete and common situation involving a reliable computer system that must be able to deal with the failure of one or more of its components. In this situation a component in fail can adopt a particular behaviour and start sending conflicting informations to other parts of the system. As explained before, the abstracted situation is similar to that of Two Generals Problem. In this case several (more than two) divisions of the Byzantine army are camped outside an enemy city with a commander, called general, for each division. The generals can communicate only via messages delivered by messengers. As in the previous problem, a single division can't capture the enemy city on his own, so cooperation between generals is necessary. Each general has to observe the enemy to decide upon a common plan of action. An important difference with the previous situation is the possible presence of one or more traitors. In particular a general can be a traitor, acting maliciously to prevent the loyal generals from reaching agreement on a common battle plan. As a first step Lamport formalised the requirements for an algorithm applicable by the generals in this way:

- `All loyal generals decide upon the same plan of action.` That means that while traitors may do anything they want, the loyal generals must do what the algorithm says they should. What the traitors will do is irrelevant for the algorithm.

- `A small number of traitors cannot cause the loyal generals to adopt a bad plan.` This assumption means that the loyal generals in addition to reach agreement should agree on a reasonable plan. But it can be quite hard to define what a bad or a reasonable plan is, and it probably is less important than the way the generals reach agreement.

Assuming n as the number of generals and *v(i)* as the information communicated by the ith general we can define a method that combines all values $v(1), \ldots, v(n)$ into a battle plan. First condition that requires the loyal generals to agree on a common plan of action is guaranteed if they adopt the same method for combining the informations they collect. In a context where the only two possibilities to choose from are "attack" or "retreat", $v(i)$ can be considered the opinion of the ith general of which option is best and the combining method can be a criterion like a majority vote rule. In this situation a small number of traitors can influence the decision only if the loyal generals were almost equally divided between the two possible options and so neither decision could be considered bad. But this approach has a flaw: it requires that every loyal general obtain the same values $v(1), \ldots, v(n)$ but a traitor may send different values to different generals. To satisfy the first assumption that requires the loyal generals to agree on the same plan of action it's necessary that each one obtains the same informations. That leads to another requirement that has to be verified for each i:

- if the ith general is loyal, then the value he sends must be used as the value of $v(i)$ by every loyal general.

These considerations leads to a further refinement step in the problem's formalization, in particular about how a single general send his value to the others. The approach historically chosen in the paper is to model the situation with a commanding general sending an order to his lieutenants, obtaining the final definition of the problem:

- `Byzantine Generals Problem.` A commanding general must send an order to his n - 1 lieutenants 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.`

These conditions are defined the *interactive consistency* conditions. The model that defines a commanding general (not necessary loyal) that sends an order to many lieutenants has been chosen to solve the original problem.

3.3 Impossible Results

Many assumptions can be made on the features of messages and the communication. In the base case that involves only oral messages, which means that the content of a message is completely at the discretion of the sender, it's proved that no solution will work unless more than two-thirds of the generals are loyal. For example in case of three generals and the presence of one traitor no solution can work. In this situation there are two options:

- the traitor is a lieutenant
- the traitor is the commander

The first case, assuming that the only two possible decisions are "attack" and "retreat", is shown in Figure 2. The commander is loyal and he sends an "attack" order, but `Lieutenant 2` is a traitor and he reports to the other one that he receives a "retreat" message. In this situation `Lieutenant 1` doesn't know who the traitor is and can't be sure of what kind of message the commander sends to the other lieutenant. There is no way for `Lieutenant 1` to distinguish between the situations represented in pictures and to understand who is the traitor, and whenever he receives an "attack" message from the commander, he must obey it.

In the situation shown in Figure 3 `Lieutenant 2` must obey to the received "retreat" order while `Lieutenant 1` obeys the "attack" order, violating condition IC1. So it's proved that no solution exists for three generals in presence of a single traitor.

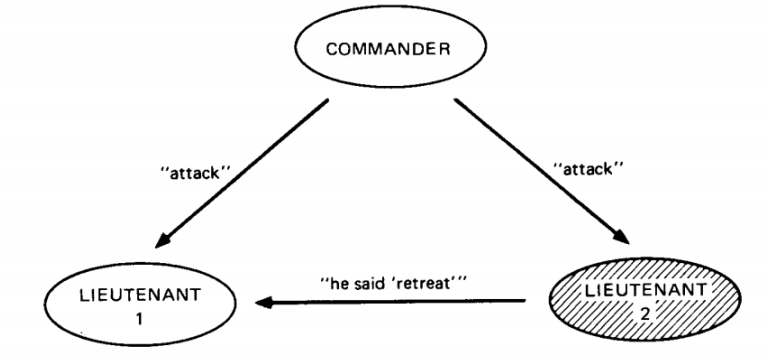

Figure 2: Case with 3 generals and traitorous lieutenant

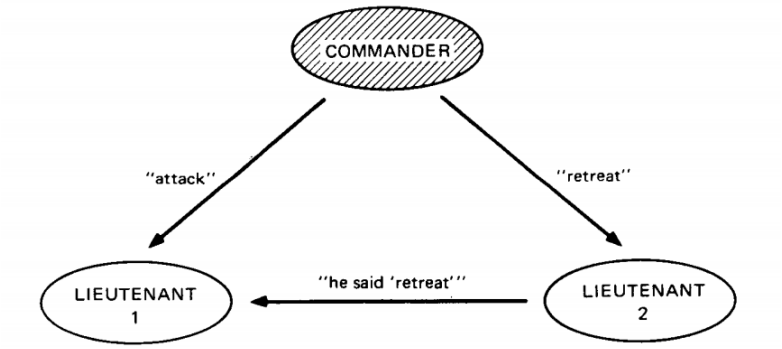

Figure 3: Case with 3 generals and traitorous commander

This result can be extended to demonstrate that no solution with fewer than $3m + 1$ generals can cope with m traitors. The proof is given by contradiction assuming the existence of a solution that works for a group of $3m$ or less generals and trying to use it to solve an instance of the Byzantine Generals Problem with three generals and one traitor, which is known to be impossible. During the demonstration the actors of that hypothetical solution are called the Albanian Generals. Each Byzantine general simulates approximately one-third of the Albanian generals (one Byzantine general is equal to at most m Albanian generals) and the Byzantine commander simulates the Albanian commander plus at most m - 1 Albanian lieutenants. Making these assumptions it can be proved that the situation of the Albanian generals requires a solution that imply the same conditions of the previous case with three Byzantine generals, that has been shown to have no possible solutions.

4 Oral Messages Algorithm

In the original paper Lamport and his colleagues after the problem's discussion also provided a first solution, applicable in presence of at least $3m + 1$ generals and m traitors (using only oral messages). The first assumption is that there is an algorithm that involves sending messages and that every loyal general executes the algorithm correctly. The features of a messaging system based on oral messages are:

- every message that is sent is delivered correctly
- the receiver of a message knows who sent it
- the absence of a message can be detected

First two assumption inhibits the traitor from interfering with the communication of other two generals. The third assumption avoids that a general tries to prevent a decision by not sending messages. In this

first version of the solution, every general is able to communicate directly with every other general. It's also necessary to define a default order to obey in case a traitorous commander decides not to send any order, in this case "retreat" is the default order.
The proposed solution presents the Oral Messages algorithm OM(m). The algorithm solves the Byzantine Generals Problem for $3m + 1$ or more general in presence of at most m (nonnegative) traitors. The algorithm needs a *majority* function, with property that if a majority of the values v_i equal v, then *majority*(v_1, ..., v_{n-1}) equals v. There are different options applicable for the *majority* function, the chosen one was to take the majority value among the v_i if it exists, otherwise the value "retreat" as default. In Figure 4 the algorithm is presented. In the following pictures two possible situations with

***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 $j \neq i$, 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}).

Figure 4: Byzantine Generals Problem solution algorithm

four generals and one traitor are figured to prove the validity of the algorithm. In the first case the traitor is `Lieutenant 3`. The commander at the first step sends a v order to the three lieutenants. In the second step, `Lieutenant 2` receives the value v from `Lieutenant 1` and a different value x from the traitorous `Lieutenant 3`. In step 3 `Lieutenant 2` uses the informations he collected to obtain the correct value, in particular *majority*(v, v, x) = v. Figure 6 shows what happens if the traitor is the commander and

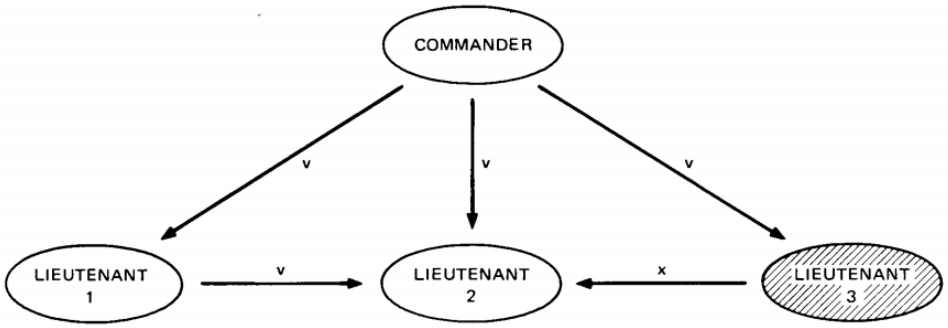

Figure 5: Case with 4 generals and traitorous lieutenant

he decides to send three different values x, y and z to the lieutenants. The three lieutenants obtains the same values $v_1 = x$, $v_2 = y$ and $v_3 = z$ and they all obtain the same value from *majority*(x, y, z) in the third step independently from the values of x, y and z. It can be demonstrated by induction that the OM(m) algorithm satisfies condition IC1 and IC2 for any m if there are more than $3m$ generals and at most m traitors.

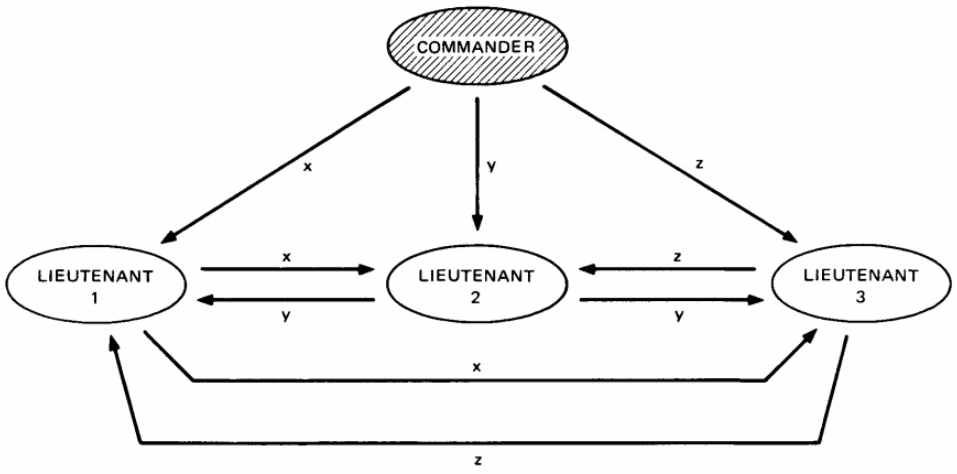

Figure 6: Case with 4 generals and traitorous commander

5 Reliable Systems

As explained before, the Byzantine Generals Problem is an abstraction of a more concrete and common situation when want to operate with a so-called "reliable computer system". A possible solution for implementing this kind of systems is to use several processors to compute the same result and than performing a majority vote of their output to obtain a single value. That could be useful to face problems like hardware components' fails, but the majority voting assumes that every non-faulty component will produce the same output and this can be true only if they all use the same input. This could be very difficult to guarantee due to problems like malfunctioning and synchronization issues. In order to lead to a reliable system the majority voting should satisfy two conditions:

- all non-faulty processors must use the same input value (so they can produce the same output)
- if the input unit is non-faulty, then all non-faulty processes use the value it provides as input (so they produce the correct output)

These two conditions can be considered equivalent to the *interactive consistency* condition IC1 and IC2. In this case the "commander" is the unit that generates the input and the "lieutenants" are the processors, while "loyal" means non-faulty. To conclude, the Byzantine Generals solution can be used to guarantee that all processors use the same input value when getting input value from a possibly faulty input device. That's important because even redundant inputs cannot provide reliability on their own; it is still necessary to ensure that the non-faulty processors use that redundant data to produce the same output.

6 BGA and Blockchain

As we have seen the consensus problem has been studied for many years and it has kept his relevance over time. Even if the Byzantine Generals Problem was formalised in the 80s, new studies and solutions have been presented in subsequent years. In recent years these topics have returned back to the top of the trends especially thanks to the advent of blockchain technologies. In particular the implications of blockchain systems in cryptocurrencies brought a big change in the world of information technology. Bitcoin and Ethereum are the most popular examples of a really huge constellation of new currencies, technologies and blockchain systems that have been born in less than ten years.

6.1 The Blockchain

Even if studies on "chains-of-blocks" have been done since the 90s, the real revolution happened in 2008 with the publication by the anonymous Satoshi Nakamoto of the celebrated Bitcoin original paper [3], and the subsequent implementation of it's blockchain.
Blockchains are diverse and can be approached from a variety of perspectives. They can be defined as a public, decentralized database, that keeps public records in an append-only fashion. Moreover, once added into the database (the blockchain) a record cannot be modified and it is very difficult to falsify entries. This last feature is called persistence. When an entry in the database needs to be updated, a new record must be appended to the existing information. A crucial feature of blockchains is that each record can be viewed by any member of the public, allowing for any person to verify the authenticity of each transaction recorded for any single entry in the blockchain. This transparency means that blockchains are auditable.
Being distributed is a killer feature of blockchains that make them immediately interesting considering that nowadays many databases need to be decentralized. The opportunity of avoid creating potential single points of failure and creating a generally more robust system made blockchain databases more appealing than traditional ones. A distributed database cannot be hacked, manipulated, or otherwise disrupted in the same way a database built on one single operator can be. In addition, a traditional centralized database requires a user-controlled access system. A blockchain, on the other hand, is operated by unknown and untrusted parties, potentially it could be an individual person, an organization, a computer operating automatically, or whatever else. The lack of trust inherent in the blockchain system leads us to the topic of consensus problem. Because any entity, individual, or party can submit information to the blockchain (in practice, try to add information to the database), it is necessary for the distributed operators of the blockchain to evaluate and agree on all new records before they are permanently incorporated into the blockchain. Because the author can't be trusted a priori, it is important that new informations are reviewed and confirmed before being accepted. This review results in the consensus.

6.2 Consensus in the Blockchain

Making a parallel between the BGA problem and the blockchain context, the "generals" are the parties participating in the distributed network running the blockchain in question. The messengers are the communications across the network on which the blockchain is running. The collective goal of the "loyal generals" is to decide whether or not to accept a new piece of information submitted to the blockchain as valid or not; in the military siege context a valid piece of information would be a correct opportunity to decide for an "attack". Loyal generals are the honest blockchain participants, who are interested in ensuring the integrity of the blockchain and guaranteeing that only correct informations are accepted. The traitorous generals, on the other hand, would be any party seeking to falsify information on the blockchain.
There are several potentially meaningful attacks that could be done on a blockchain. An individual may want to double spend some currency or spend a Bitcoin he does not actually own, or someone could want to alter a specific transaction. It's quite obvious how much consensus and blockchain are bound together. There are four main methods used to reach consensus in a blockchain (and all distributed systems):

- the Practical Byzantine Fault Tolerance algorithm (PBFT),
- the Proof-of-Work algorithm (PoW),

- the Proof-of-Stake algorithm (PoS),
- the Delegated Proof-of-Stake algorithm (DPoS)

6.3 Practical Byzantine Fault Tolerance

Historically the Practical Byzantine Fault Tolerance algorithm was the first of these solutions to be presented. It was published in 1999 [4] as a strongly optimized solution for the consensus problem working even on an asynchronous environment like the Internet. It can be used to establish consensus in blockchain systems, but is only one of those potential solutions. Three examples of blockchains that rely on the PBFT for consensus are Hyperledger, Stellar, and Ripple. The algorithm was designed for a state machine replication tolerating Byzantine faults. It offers both safety and liveness while at most one third of total n replicas are simultaneously faulty, without relying on synchrony assumption. The system model the algorithm is designed for is a distributed asynchronous one where nodes are connected by a network. The service is modelled as a state machine that is replicated across the nodes of the distributed system. Each replica maintains the service state and implements the service operations. The replicas move through a sequence of numbered configurations called *views*. In a view one replica is the *primary* and the others are *backups*.
Denoting the set of replicas as $\mathtt{R}$, the primary of a view is replica $\mathtt{p}$ such that $\mathtt{p} = \mathtt{v}\ mod\ |\mathtt{R}|$, where $\mathtt{v}$ is the view number. View changes are triggered when a fail of the primary is detected. The main steps of the algorithm are:

- a client sends a request to invoke an operation directly to the primary
- the primary multicasts the request to the backups
- replicas execute the request and send a reply to the client
- the client waits for $\mathtt{f} + 1$ (where $\mathtt{f}$ is the maximum number of faulty replicas) replies with the same result from different replicas; this result is the final one of the operation

The algorithm impose two requirement for the replicas:

- they must be *deterministic*, which means that the execution of an operation in a given state and with a given set of arguments must always produce the same result
- they must start in the same state

When the primary receives a request from a client, it starts a three-phase protocol to multicast the request to the replicas. The tree phases are *pre-prepare*, *prepare*, and *commit*. The first two phases are used to totally order requests sent in the same view. The prepare and commit phases ensure that requests that commit are totally ordered across views. Figure 7 shows the operations of the algorithm in the normal case with a non-faulty primary.

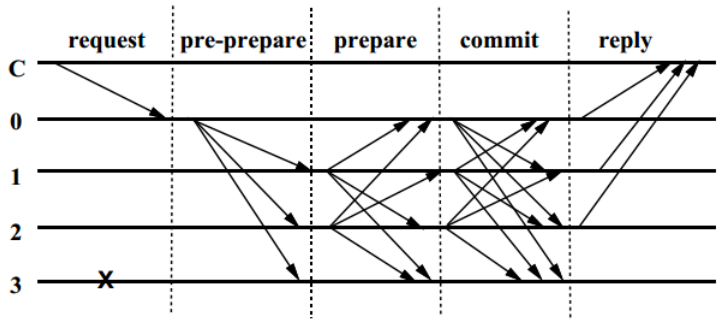

Figure 7: PBFT algorithm, replica 0 is the primary, replica 3 is faulty, C is the client

Among other considerations, this method of establishing consensus requires less effort than other methods. However, it comes at the cost of anonymity on the system.

6.4 Proof-of-Work

The best-known method of reaching consensus on a blockchain is the Proof-of-Work (PoW) scheme. The first traces of PoW date back to the Hashcash protocol, proposed to contrast spam emails, but this technique has reached its popularity for it's application in Bitcoin. A very important component of PoW are hash functions, sometimes called "cryptopuzzles". The system needs a leader, that is responsible for broadcasting a new block to the network, so that the other peers can verify the validity of its contents. In Proof of Work, in order for a node to be elected as a leader and choose the next block to be added to the blockchain they have to find a solution to a particular mathematical problem, the cryptopuzzle mentioned before. Assuming that the hash function used is cryptographically secure, the only way to find a solution to that problem is by trying all possible combinations (bruteforce). In probabilistic terms, the node who will solve the problem first the majority of the time is the one who has access to the most computing power. These actors are also called miners.
The success of the system is related to two properties:

- it is hard to find a solution for that given problem
- when given a solution to that problem it is easy to verify that it is correct

Miners are incentivized to keep mining: when a new block is mined, that miner gets rewarded with some currency (block reward, transaction fees). Other nodes verify the validity of the block by checking that the hash of the data of the block is less than a preset number. Due to the limited supply of computational power, miners are also incentivized not to cheat. Attacking the network would cost a lot because of the high cost of hardware, energy, and potential mining profits missed.
Figure 8 illustrates very well how Bitcoin and any other coin that uses Proof of Work discourages malicious behaviour.

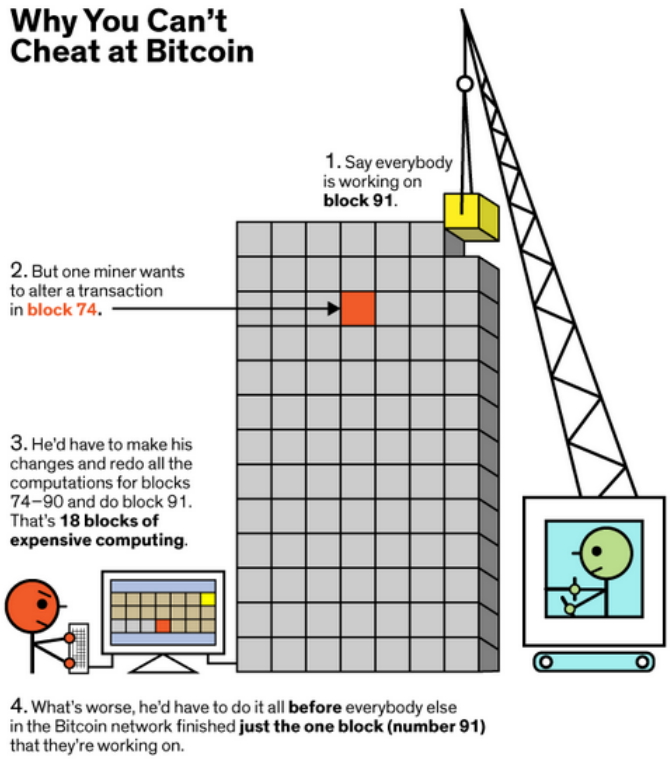

Figure 8: Bitcoin security concept

Talking about the Bitcoin PoW-based network, these are the main functional steps:

- new transactions are broadcasted to all nodes

- each node collects new transactions into a block
- each node works on finding a difficult Proof-of-Work for its block
- when a node finds a Proof-of-Work, it broadcasts the block to all nodes
- nodes accept a block only if all transactions in it are valid and not already spent
- nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as previous hash

Nodes always consider the longest chain as the correct one and will keep working on it. If two nodes broadcast different versions of the next block at the same time, some nodes may receive one or the other first. Every single node works on the first it received, but saves the other branch in case it becomes longer. When the next Proof-of-Work is found and one branch becomes longer, the nodes that were working on the other branch will then switch to the longer one. This Bitcoin-style PoW scheme allows

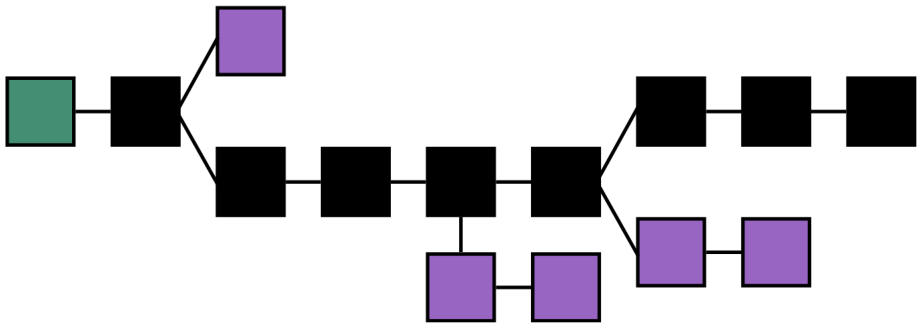

Figure 9: The main chain (black) consists of the longest series of blocks from the first (genesis) block (green) to the current block. Orphan blocks (purple) exist outside of the main chain.

for easy, broad participation, which in turn ensures greater network stability with minimal requirements on each participant, allowing participants to remain for example anonymous. Proof-of-Work now is the most popular algorithm being used by currencies such as Bitcoin and Ethereum (that is actually working on a Proof-of-Stake implementation), each one with its own differences.

6.5 Proof-of-Stake

Proof-of-Stake was proposed for the first time by a user of the BitcoinTalk forum in 2011. This technique is based on the concept of "stake", that is used in the leader election process. In a cyptocurrency context, stake can be defined as an amount of currency that actors want to lock up for a certain amount of time. In return, they get a chance proportional to their stake to be the next leader and select the next block. In a very simplified way we can say that in PoS "if A has more stake than B, then A is more likely to be elected", in this case, validate the next block.
Compared to the PoW context, in PoS the actors are *validators* rather than miners and new blocks are *forged*, more than mined. To became a validator, a node has to deposit a certain amount of coin into the network as stake, in a way similar to a security deposit. The size of the stake determinates the chance of the validator to be chosen to forge the next block. In this system the trust is entrusted to the stake: if a node validates a block with fraudulent transactions, it will lose some of his stake. As long as the stake is higher than the fees that the validator gets doing his job, we can trust him.
Proof of Stake replaces the energy and computational power required by PoW with stake, so it permits to save resources. It's also more decentralized, because in PoW miners can aggregate their computational power in "mining pools" to mine more blocks and divide more rewards, and that's quite strange for a system that was designed to be distributed. Moreover, a too large aggregation of miners could be very dangerous for the integrity of the network.
A relevant flaw PoS is that if an individual takes the control of more than the half of the stake (the majority), he can control the network and start approving fake transactions. But this kind of "51%

attacks" is a weak point common to the PoW algorithm, where if a miner or a group of miners obtain the majority of the hashing power, they can effectively control the blockchain. However it's important to note that if Bitcoin would be converted to Proof-of-Stake, acquiring the 51% of all the coins would require to get an amount of currency equal to more than 79 billions dollars, that it's almost impossible. Another risk is related to the selection of next validators, that can't be totally randomized but at the same time has to be based on the size of the stake deposited by each node without advantaging rich ones too much.

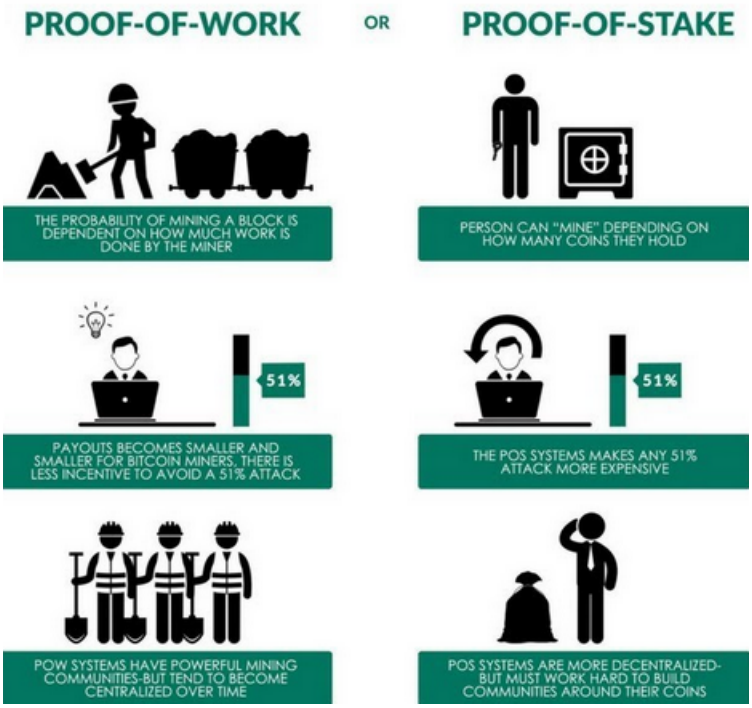

Figure 10: PoW vs PoS

PoS brings more risks than PoW, but researchers are very active on this topic. Hybrids consensus algorithms have been proposed, such as the PoW-PoS combination used by Decred. Active research towards a secure and decentralized Proof of Stake protocol is being done by the Ethereum Foundation with Casper The Friendly Ghost and Casper The Friendly Finality Gadget. There are also various existing coins which use pure PoS, such as Nxt and Blackcoin.

6.6 Delegated Proof-of-Stake

Delegated Proof of Stake (DPoS) is an optimized variant of PoS that provides a high level of scalability at the cost of limiting the number of validators on the network. DPoS is a system in which a fixed number of elected entities (called block producers or witnesses) are selected to create blocks in a round-robin order. Block producers are voted into power by the users of the network, who each get a number of votes proportional to the number of tokens they own on the network (their stake). Alternatively, voters can choose to delegate their stake to another voter, who will vote in the block producer election in their place. The idea is that the community members vote for super representatives to secure their network and super representatives will be rewarded by validating transactions for the next block. In DPoS:

- block producers are those responsible for creating new blocks; they are elected by voters and their number is limited
- block validators are nodes that verify that blocks created by block producers follow the consensus rules; any user is able to run a block validator and verify the network

A round in a DPoS blockchain with N block producers is as follows:

- N block producers get elected from the pool of block producer candidates
- the ith block producer signs the ith block, until i = N

A block is finalized when it is voted on by (2/3 + 1) of the block producers. Otherwise, the longest chain rule is followed. Each DPoS implementation has his specific reward model, that defines block reward and inflation mechanisms. DPoS is a protocol that sacrifices decentralization for throughput due to the low number of block producers. As a result of the limited number of block producers, DPoS is able to handle transaction throughput that is multiple orders of magnitude greater than today's PoW.
While DPoS makes sense for many applications that require a high level of scalability, many blockchain developers believe that DPoS is not decentralized enough to be a base layer that acts as a store of value and ledger of ownership for applications that handles, among other things, financial transactions. However this technology has interesting potential applications. Researchers are exploring the possibility of using a PoW network like Ethereum as a secure base layer and build a DPoS chain on an upper layer, running the majority of an application on the highly-scalable DPoS chain, while still using the secure base layer for parts of the application that require high security, like in-game currency or ownership of assets.

7 Bibliography

1 : Leslie Lamport, Robert Shostak, and Marshall Pease. *"The byzantine generals problem"*. ACM Trans.Program.Lang.Syst. 4(3):382–401, July 1982.

2 : E. A. Akkoyunlu, K. Ekanadham, R. V. Huber. *"Some contraints and tradeoffs in the design of network communications"*. 1975. Department of Computer Science, State University of New York

3 : Satoshi Nakamoto. *"Bitcoin: a peer-to-peer electronic cash system"*. 2008. http://www.bitcoin.org.

4 : Miguel Castro, Barbara Liskov. *"Practical Byzantine Fault Tolerance"*. 1999. Laboratory for Computer Science, Massachusetts Institute of Technology

5 : https://medium.com/@chrshmmmr/consensus-in-blockchain-systems-in-short-691fc7d1fefe

6 : https://blockonomi.com/practical-byzantine-fault-tolerance/

7 : https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419

8 : https://medium.com/loom-network/understanding-blockchain-fundamentals-part-2-proof-of-work-proof-of-stake-b6ae907c7edb

9 : https://medium.com/loom-network/understanding-blockchain-fundamentals-part-3-delegated-proof-of-stake-b385a6b92ef