Blockchain and Distributed Ledgers

Guillem Borrell, NFQ & UC3M
guillemborrell@gmail.com
guillem.borrell@nfq.es

May 11, 2017

Despite the huge hype about distributed ledgers and the Blockchain, these technologies are not going to change *the world* in their current form. This article tries to shed some light on the topic, rooting its conclusions on theorems and fundamental aspects about distributed systems, that allow to understand the inherent limitations of this technology.

1 A non-technical introduction.

1.1 Centralized ledgers.

A centralized ledger is a database, or a physical book, or a clay panel, used to keep a series of transactions in a specific order. It represents the current state of a system, if some transaction is not in the ledger it is not considered at all, and it trivially solves any conflict between peers, since the ledger is the record of all transactions and they are all in order. It is therefore a tool to store relevant data and to guarantee consensus. A centralized ledger, or simply a ledger, is a basic tool for accountancy, bookeeping and notarization. Practically all financial transactions until the introduction of Bitcoin in January 2009 have been stored on a ledger.

Each centralized ledger inherently requires or introduces a centralized authority within a system. The owner of the ledger is the owner of the data, and in most cases it charges a fee to each user to access the data. Authority is a social and economical construct, and it has very little to do with technology. Of course, authority and ownership means liability too. If several peers agree on not owing the data and paying a fee, one can assume that there is a strong underlying motivation for doing so. Maybe the ledger is imposed by a higher authority, like a central bank, or the peers are usually in coflict, and an agreed authority that imposes consensus reduces the amount of time the system is in a transitional state. Either way, the ledger is just a tool with which a trustworthy authority keeps data stored and enforces consensus.

1.2 Present economical context.

Since their introduction, probably 7000 years ago in Mesopotamia, ledgers have been a central part in Economics and Finance. The three keywords mentioned to characterize ledger technology in the previous section: trust, maintenance costs, and consensus, have been present since its beginnings. These three key aspects also help to understand why a ledger is abandoned by its users. When authority is not trustworthy anymore, the fee is higher than the economical benefit, or the enforcement of consensus is not efficient, the peers move from one ledger to another. Until very recent times, this meant that the users went from one central authority to a different central authority that offers significant advantages in terms of trust, cost, and consensus quality. Internet and P2P software have proven that one can efficiently store information and enforce consensus between peers without a central authority. However, the third term in the "distrubuted ledger equation" was still missing: trust.

The actual revolution of the Blockchain, introduced for the first time in [Nakamoto, 2008], was the fact that consensus could be reached with no trust between peers at all. In consequence, a fully distributed ledger without a central authority was possible for the first time. It is important to emphasize that the key aspect here is not that the central authority represents a single point of failure, but that the whole system is based on trust.

This technological achievement was contemporary to many other social and economical changes. The global financial meltdown had eroded the trust in financial institutions, including central banks. At the same time, many applications in P2P business were successful at avoiding human brokers and intermediaries, reducing the operation costs for all peers.

Every new technology that is found to be usfeful comes with a varying amount of hype attached. There are very few pieces of technology with such a level of hype as distributed ledgers. This helps the technology to be more widely known, but at the same time reduces the signal-to-noise ratio of the information that non tech-savvy perceive.

In any case, users needing to store transactions and to reach a consensus have now an alternative to an appointed central authority. However, removing trust from the "distributed ledger equation" comes at a cost, and to understand how mucho one has to pay, we must dive into the technical aspects of Blockchain and distributed ledgers.

1.3 Smart contracts

A smart contracts are computer protocols that facilitate, verify, or enforce the negotiation or performance of a contract, or that make the contractual clause unnecesary [Smart Contract]. Within the context of the Distributed Ledger Technology (DLT from now on), such protocol or application also uses the ledger to store information and to achieve consensus between peers.

This means that one can implement a smart contract without a distributed ledger, since from the functional point of view, the protocol is not more capable depending on how the information is stored, or how the consensus is reached. The key is, again, trust. An application is as trustworthy as it is its infrastructure and its provider. Some

companies decide to use Gmail or Outlook 365 because they trust Google and Microsoft to store their private or sensitive information. The email service provider would be the equivalent of the central authority.

DLT allows the implementation of a smart contract between parts that they do not trust each other nor the broker, in context where a central authority used to be a fundamental requirement as long as all the information stored within the ledger is public. Bitcoin can be defined as a smart contract implemented using the Blockchain as a means of obtaining storage and consensus.

2 The technical aspects.

The fundamental goal of this document is to build an intuition about the good, the bad, and the ugly of the technological aspects of DLT. It is important that this intuition is built upon correct definitions and plausible hypotheses.

2.1 Some important theorems and hypotheses.

To understand the fundamental aspects of DLT we must dive in some theory about distributed systems. We have already mentioned the three key aspects of any ledger: information storage, consensus and trust. The next step is to link these three properties to more quantifiable ones using the CAP theorem [Hyperledger].

2.1.1 The CAP theorem, or Brewer's theorem

This theorem states that a distributed computer system cannot simultaneously provide at the same time more than two out of three guarantees:

1. Consistency: Every peer receives the most recent data stored within the system.
2. Availability: Every request receives a response, without the guarantee that it contains the most recent write.
3. Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped or delayed by the network between peers.

This theorem has three important implications if one wants to use a distributed system, like a DLT, to store the information of a smart contract.

The first two guarantees are the expected properties of any ledger. One expects to always obtain a non-error response from the ledger, and that response to be the most recent information stored. Any relational database offers these two guarantees. But in a distributed system, where the nodes send information to their peers through a network, I may sometimes get an error (the information is not available) or outdated information. This is a theorem, there is nothing we can do about it.

The second implication comes from the fact that these three guarantees cannot be provided simultaneously, meaning that in a distributed system, one may get the most

recent state without errors, but not exactly at the time it is required. In other words, if I want to retrieve data from the DLT and I want that data to be the most recent state of the system, there will be some latency, and there is nothing anyone can do about it. Note that a centralized ledger is not parallel. We could perfectly use a single instance of SQL (or NoSQL) database to store the data with perfect consistency and availability.

Latency is key in systems engineering, because it is one of the determining aspects of the quality of service. We have proved that distributed systems are subject to latency, the question is how much, since some applications require high bandwidth and low latency storage, and DLT may not be able to provide that.

There are several technologies that keep a distributed storage system consistent. If data has to be accessed by all peers, but it can be stored by only a few, we have the Distributed Lock Managers (DLM) like Google's Chubby, or the lock managers present in many distributed file systems and databases. If data has to be accessed by all peers, and at the same time each peer must own a consistent copy of all the data, we need a consensus protocol.

2.1.2 The Byzantine agreement problem and the cost of consistency.

Assume that the Byzantine army is encircling a Persian city. The generals, each leading a portion of the army, have to formulate a plan for ataccking the city. They must only decide whether to attack or retreat. Some generals prefer to attack, others prefer to retreat, others may be traitors, others may be deaf, some general is sick and cannot leave his tent... The important thing is that the strategy will fail if it is not applied consistently by all generals. The army must reach a consensus between the generals, and the whole army must attack or retreat. Lack of consistency implies defeat. This problem was first described by [Lamport et al. 1982], and it has been proven to be unsolvable, which gives a measure on how hard it is to achieve consistency with faulty communications or hostile nodes.

Ledgers must be always consistent, because operations with outdated information are errors from a fundamental point of view. Think about a virtual currency. If two peers see a different balance for a user, that user can exploit the difference for an attack, like spending the very same currency twice. This is the double spending attack, which is an example of a Byzantine fault, and it is solved by the Blockckain network with the Nakamoto consensus. Since the system has some inherent latency, the attacker's goal is to make the peers accept that transaction and transfer the goods before they realize they have been payed with the exact same currency. This attack is impossible with actual cash or with a central ledger. The Nakamoto consensus will be explained later in this document, but the key at this point is to note that consistency comes with a price. The more hostile nodes, or the more sophisticated the attack, the harder is to reach consensus. In consequence, the harder the circumstances, the higher the latency. If the Byzantine fault is too hard, there is a proof that there is no way to find a consensus at all.

2.2 Blockchain as an example

It is time to use Blockchain as an example to clarify the concepts introduced with the CAP theorem and the Byzantine agreement problem, since the Blockchain's Namkamoto consensus is designed precisely to deal with those issues. The protocol will not be described in detail, and the focus will be pointed to the key aspects of the consensus.

Assume Alice wants to send 1BTC to Bob to pay her rent. Bob, who does not trust Alice, wants the Blockchain to certify that the payment is legit. Alice will connect to his Bitcoin node to check it has enough currency to perform the transaction. She will build the transaction and send it to the miners. Each miner will build a block, with or without the transaction. Also, even if a miner puts Alice's transaction in the block, the position within the block may vary between miners. Each miner will then try to compute a *proof of work*, a mathematical problem that is very hard to compute and very easy to verify. The first miner to find the proof of work will submit the new block to the miners and the nodes, putting Alice's transaction at the end of the Blockchain, which allows Bob to verify that Alice has actually submitted the transaction looking at the Blockchain on his node. To keep the miners running, each miner gets some currency once the proof of work is computed, and each user can assign a fee to the transaction to push the transaction into the next block.

This short and incomplete description allows us to analyze this consensus using the CAP theorem. The system is not consistent at all times. The consistency happens in snapshots separated by a given time. During that transient the transactions are being sent to the miners, or the miners are computing the proof of work. The proof of work is such an expensive computation that all the miners are consuming approximately 343 MW worldwide, and following the present trend, they will consume as much electrical power as Denmark by 2020.

The size of each block is fixed at 1MB, and the time between blocks is approximately of 10 minutes. The cost of the proof of work is tuned to keep the time between blocks constant. An average transaction is of 0.5 KiB of size, therefore the average total bandwidth of Bitcoin as a database is between 3 and 4 transactions per second. This average value is far from the worst case scenario. If the winning miner pushes an empty block, the bandwidth has been wasted during 10 minutes. In addition, two miners may find the proof of work at the same time, breaking the consistency within the network temporally. For this reason it is advised to wait for a couple more blocks to appear to verify a transaction. This means that any Bitcoin transaction takes between 10 minutes and half an hour, which is probably too much for most financial transactions[1].

Most these parameters are tunable, but up to an extent. Mining time probably should not be set to 1 second, the same way one shoud not configure a block size of 1 GiB. One

[1] To make things even worse, Bitcoin is so successful that miners receive more transactions that the amount they can fit in a single block. For that reason, they tend to choose the smallest transactions with a higher fee. As a consequence, transactions too large or with too little fee are never submitted. To overcome this situation, the protocol may be modified to make larger blocks, or to reduce the separation between blocks. Neither the former nor the latter are straightforward solutions. Larger blocks mean that miners have to be modified, while shorter block separation may create more branches, delaying the consensus.

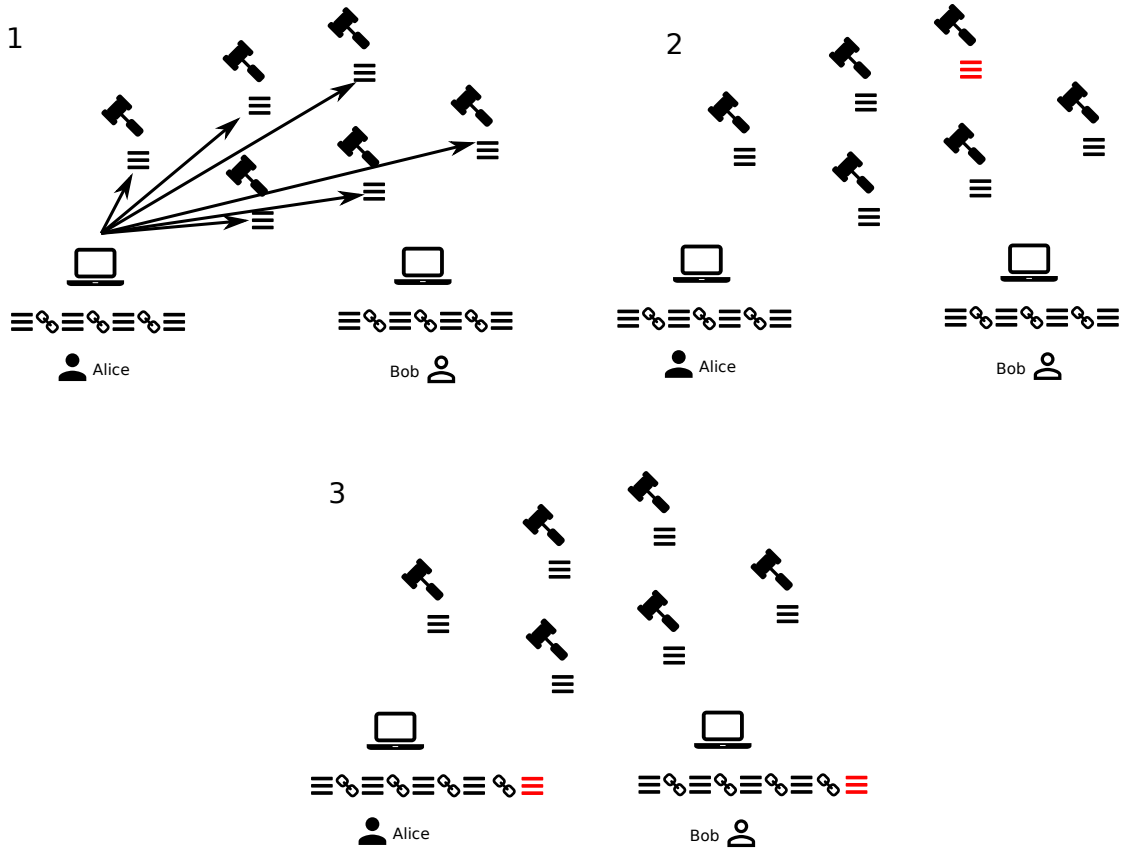

Figure 1: Characteristic steps of the proof of work process characteristic of the Nakamoto consensus. 1) The client sends the transaction to the miners. 2) One of the miners obtains the proof of work. 3) The clients fetch the new block with the proof of work, and chain it to the previous steps of the chain.

of the consequences of the CAP theorem is that those parameters will have an optimal value depending on the latency of the network, the number of nodes, the most common attack...

2.3 A bank as a counterexample

Assume Alice wants to send 1000€ to Bob to pay her rent. Bob does not trust Alice, but he trusts his bank. Alice will ask her bank to send to Bob's account a transfer worth 1000€. If Bob's account is in the same bank as Alice's the whole process takes place in less than a second and it won't require a fee. If Bob's account is in the different bank as Alice's, the two banks will take care of the transaction for a small fee. As Bob trusts his bank, he will consider that the rent is paid as soon as he queries his bank's leger, and sees his acount balance increased by 1000€.

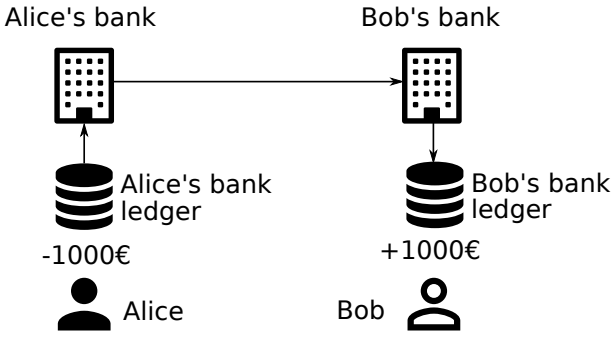

Figure 2: Flow diagram of a common financial transaction through the banking system

Since the banking system is not data-distributed, and each bank maintains its own ledger, the threats to the system are due to security faults or authentication issues, and not due to issues with the consensus protocol. We tend to trust our bank because it seldom fails at managing authentication and safety.

2.4 Trust, complexity and latency.

We have learned that distributed systems have latency, and that reaching Byzantine consensus involves some additional cost due to the distributed nature of the system. At the same time, non-distributed systems based on trust are significantly more efficient and lean. The amount of trust we have on the nodes is inversely proportional on the cost of the consensus: the more we trust, the easier is the consensus. At the same time, the cost of consensus is proportional to the latency. The harder the consensus, the higher the latency. Therefore, the latency, and the quality of the service, is inversely proportional to the trust.

Trust is a gradable quantity. In our examples, we have gone from a trust-based network, banking, to a no-trust-at-all network, Bitcoin. There are other protocols aimed at solving different instances of Byzantine consensus that are faster than the Blockchain and less naive than a centralized ledger. The consensus issue is similar to security and permissions. One can be fully promiscuous and guarantee full permissions to any user, or enforce restrictions at every user's step. How to manage consensus, like permissions, is an engineering decision. This is one important contribution from the second generation of DLT appeared after Bitcoin, like Hyperledger[Hyperledger] or Ethereum[Ethereum]: consensus is a pluggable component.

Each Byzantine consensus algorithm is designed to solve one particular Byzantine failure problem, or a restricted set of them. Plugging in one particular consensus algorithm and implementation is therefore an engineering decision. In any case, the dimensional assumption made previously is still valid. The lower the trust, the harder the consensus, and the higher the latency.

3 Smart contracts.

All the technology described so far is only the storage and peer consensus part. It all lacks any business logic to be useful for any purpose. Any modern DLT exposes an API to create a smart contract, where the inner workings of the storage and the consensus are hidden to the developer and the user. Writing a smart contract feels like writing a program.

Some API may be more feature-extense than others. Some API may allow a node to subscribe to events and react to them, changing the state of the contract accordingly. Or they may provide some sort of authentication and user permissions system. In any case, the application layer is way beyond the details of data persistence and the consensus protocol.

It is key to emphasize that applications are as complex as they need to be, and a DLT may not reduce that complexity at all if it comes from the functional requirements. The main contribution of the DLT is that some algorithms that had to be run in a single computer and database can now be distributed over a network with a limited amount of trust. One should understand that this reduced amount of trust in peers does not mean that the application is any less robust against real-life attacks, like stealing keys or credentials.

3.1 Current platforms for DLT-based smart contracts

There are two major platforms of DLT-based smart contract platforms contemporary to this document: Hyperledger and Ethereum. The underlying design of both plaforms is similar, being Ethereum the first to blueprint such a structure. The main idea is that the ledger must store the status of the contract and the program itself.

This is kind of a revolutionary idea. The database stores the state of the system and the code necessary to mutate it. The main difficulty of this approach comes from one essential characteristic of DLT: any node must be able to verify all the states of the contract. This means that all the nodes should be able to run the program too, and to obtain exactly the same results as any other node. This requires the implementation of a small virtual machine that is fully deterministic on all its results, avoiding some important hardware optimizations[2]. This means that the DLT is a distributed computer, a particularly slow one. On the other hand, this allows full generality of a smart contract: a modern DLT may run any kind of program without affecting the consensus in any way.

[2]Some hardware optimizations that make computers fast return nondeterministic results, meaning that the result of a single operation may not be exactly the same, even if repeated in the same machine. One particularly relevant example of this is floating point division. If two floating point numbers (numbers with some decimals) are divided, the operation has an inherent error that is usually neglectable. But since the results of the execution of the smart contract are hashed, the result has to be exactly the same bit by bit.

4 Pyledger.

One of the difficulties in the DLT ecosystem is that the infrastructure tends to be quite complex for simple experimentation with smart contracts. A distributed ledger is a distributed computer with pluggable consensus, therefore it requires a relatively complex setup that requires some specific technical knowledge. In consequence, creating and running a "Hello, world" smart contract over Hyperledger or Ethereum is far from trivial.

Pyledger[Pyledger] tries to deal with this issue by offering a platform for smart contracts with simplified consensus, or no consensus at all. The goal is to be able to prototype smart contracts with functionalities analogous to the offered by the API of production-quality DLT.

This does not mean that pyledger cannot be used in production, but that it does not intend to compete with the present DLT enterprise architectures. At this point is just a platform for prototyping and experimentation. However, such a platform allows any user to grasp the basic concepts behind smart contracts, and to develop proof-of-concept applications in much shorter cycles. Despite pyledger is a work in progres, it already supports most of the capabilities offered by Ethereum's Solidity programming language and application architecture.

In pyledger a smart contract is coded in Python, which is also an advantage for rapid prototyping. The documentation and instructions for installation and deployment can be found here.

References

[Nakamoto, 2008] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. Link

[Smart Contract] Smart Contract. Wikipedia. Link

[Gilbert & Lynch, 2002] Gilbert, S. and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT Neews. Vol 33, Issue 2, June 2002, pp. 51-59.

[Lamport et al. 1982] Lamport, L. Shostak, R and Pease, M. The Byzantine Generals Problem. ACM Transactions on Programming Languages (TOPLAS), Vol. 4, Issue 3, July 1982, pp. 382-401

[Hyperledger] The Hyperledger Project. Hyperledger white paper. Link

[Ethereum] Ethereum Project. White paper. Link

[Pyledger] Guillem Borrell. The Pyledger project. Link