Lecture Notes for CSCI 351: Distributed Computing

Set 10-Blockchains

Professor Talmage

April 28, 2023

---

1 What is a Blockchain? One better, why is a Blockchain?

As we have have discussed, many distributed computing problems can be simplified by using a centralized algorithm, in which a central process determines order, priority, or other global metrics. This works well, since it basically moves a problem from a distributed model to a sequential, so we can use sequential algorithms to solve it, and there can be no disagreement since there is a single authoritative voice. There are two problems I have consistently cited as reasons to avoid this type of solution: efficiency and fault-tolerance.

Centralized solutions may not be the most efficient, as all communication needs an extra hop through the central coordinator, instead of going directly to the recipient. It is less fault-tolerant, as if the central process fails, the entire system is compromised. Another real-world concern, which we have not discussed in our algorithmic conversations, is that the central server has complete knowledge of every process' actions. For applications where privacy could be important, this is a fatal flaw. In a sense, this is a type of Byzantine failure, though it is not really captured algorithmically, as the central server may perform the algorithm exactly, except keep a copy of all actions to use for nefarious purposes outside the algorithm.

Blockchains are fundamentally an approach to solving Byzantine Agreement (Consensus), with a lot of publicity towards the privacy argument. Instead of a centralized server which is assumed to be not Byzantine, blockchains are a fully-distributed approach to agreement. Recall the proof that the Consensus object is universal, as we can implement any ADT by agreeing on each element of the sequence of operation instances which should take effect. Blockchains are a total order of events agreed upon by participating processes. With some cryptography sprinkled generously on top, people claim that Byzantine processes cannot change already agreed-upon parts of the order.

Of course, this is ideally done in an asynchronous system with Byzantine failures, so we know this is impossible in a strict sense by FLP. Thus, all blockchains either have implicit synchrony assumptions or only probabilistic guarantees of success. Once we have a solution to Byzantine Agreement, of some such form, we can then use it to implement any ADT. The most talked-about application is payment systems, though there are many other applications, such as elections without a central authority. There are many hyperbolic claims that blockchain will solve almost any problem presented, but that is just an application of Consensus, and is dependent on the correctness of the Consensus protocol.

2 Bitcoin Overview

We will start with a high-level overview of Blockchain, following the original whitepaper which introduced it[1]. Once we have this basic idea of how blockchains operate, we can explore more technical details of the guarantees they do and do not provide, and discuss alternate ideas to remedy some of their worst problems.

We assume that

- Each process can cryptographically hash any value in a way that cannot be broken.
- A majority of participants are correct—specifically that a majority of the computational power in the system is honest.

2.1 Hashing

We should talk a little about cryptographic hashing. It's a deep and interesting mathematical topic which I cannot speak about authoritatively, but we can discuss some of the high points.

The idea of a hash function is that it is a one-way, or non-invertible, one-to-one function. That is, any two inputs should give different outputs, but no one should be able to take your output and re-construct your input. So far, so good. We also generally desire an output that is much smaller than the input, though that is less for cryptographic reasons and more for practicality or hash tables. This is mathematically impossible in general (you may recall an exercise to prove this from CSCI 341!), but as long as not every bitstring is actually a possible input, one can build hash functions with very low probability of collision. (Note: This already means that any solution is at best probabilistically correct.)

A common variation of this idea is public/private key hashing. Such a system has a pair of values $(pub, priv)$ and provides two functions $sign$ and $verify$, such that $verify(pub, x, sig)$ returns True if and only if sig is the output of $sign(priv, x)$. Public keys are distributed publicly and we use them to check that a given hashed value was signed (hashed) by the party that owns the private key. As long as the private key remains private (only known to the signer) and the hashing algorithm works as assumed, then we can verify that a value is the same as the value the signer saw. Thus, if p_s says "I want you to see value x, so I signed it to yield hash value sig.", we can run $verify(pub, x, sig)$ to ensure that x has not been corrupted. If it has, then the mismatch between sig and x will cause verification to fail, allowing us to detect tampering.

2.2 Custody Chains

The first component of a blockchain is to provide a chain of custody–an uneditable list of owners of a particular item (commonly called a coin). The idea is that this will enable someone to prove that they own a particular coin. To do this, note that a coin is now a history, not a single value. For payer P to transfer a coin to recipient R, P concatenates

- the coin (previous transaction list plus the pair (P, R)),
- the public key of R, and
- the cryptographic signature from the transaction in which P obtained the coin,

[1] `https://bitcoin.org/en/bitcoin-paper`

to form the *transaction* object, then signs the result. R (or anyone else) can now check that this transfer is correct by using P's public key to verify that signature, confirming the transaction and verifying that P had the coin and was able to spend it (by checking back through the entire chain of transactions stored in the coin with the public keys stored in the coin). This prevents an attack where someone claims that P paid them when P did not actually give them that coin.

The problem is that there is another type of attack: P could validly obtain a coin, then spend it many times by re-signing the history and different recipients, creating different chains. Checking a particular chain of signatures would indicate a valid chain of custody, because any one chain does not know about others. The entire point of a blockchain is to use distributed consensus to agree on the canonical chain to prevent this type of branching, or double-spending.

2.3 Blockchains

Of course, running an entire asynchronous, fault-tolerant consensus algorithm to agree on each transaction's hash individually would be wildly inefficient. Instead, transactions are collected in bundles (the eponymous *blocks*), and the hash of an entire block is published at once. Similar to the original custody chains, each block contains the hash of the previous block, as well as of several transactions. This combines the transaction chains for each coin into a total order of all transactions, which is what we want to have to have reliable knowledge of who owns which coin. Double-spending should now be impossible, since the second transaction will be invalid and thus never added to a block of confirmed transactions. (By an honest actor!)

So, how do we decide what transactions go in a block, since that is an agreement problem of its own? This is where things get very handwavy. When P completes a transaction, it broadcasts it to all. All processes[2] collect transactions which are valid by the existing prefix of the chain until they have enough to fill a block, which has a pre-defined size. Once any node has enough transactions, they race to complete the block, performing a time-expensive computation. When they complete that computation, they broadcast their new block.

When a node receives a new block, it checks to make sure it is valid, both by verifying its signature and checking that each transaction is transferring a coin from its previous owner to a new owner. If it is valid, then the node abandons any effort it may have expended to build a new block and adds this block to its chain. If it knows of enough new transactions that are not in the new chain, it will start trying to build the next block in that chain.

Processes always use the longest chain any participant broadcasts, so if they see a conflicting block (a branch), they keep both versions. Each node works on extending the branch it saw first, unless it receives a new block that makes the other branch longer, in which case it switches and starts again. Since computing a new block is supposed to be computationally hard, it is hard to discard a significant portion of the blockchain, as that would require re-computing more blocks from a branch point to override the existing chain.

The Bitcoin whitepaper does not really address very long delays. As far as I can tell, it is possible that, even without dishonest nodes, if messages between two halves of the system are very slow, then each half could have a different branch of equal length. First, resolving which branch dominates is not trivial if they're of similar length, and the system may switch back and forth repeatedly as one side or the other gets slightly longer. (Is it possible for a blockchain to thrash?)

Second, the external cost of rolling back a large number of blocks is huge, as transactions can suddenly no longer exist, so the money you thought you had reverts to the previous owner. This is the type of attack the entire system is build to address, where an attacker gives you money, then

[2]This is a simplification, since there can be two types of nodes: those simply owning and exchanging coins and those which build the chain.

deliberately severs communication with the network and privately constructs an alternate history in which you never got that money. The claim is that, as long as a majority of the computing power in the system is honest, no dishonest actor can add malicious blocks fast enough to override the longest chain and undo something in a different branch. Except they maybe can, since finding a next block is probabilistic, so there is a chance a bad actor can add bad blocks to the longest chain. Just most of the time, good actors will add valid blocks faster. Thus, the guarantee is probabilistic, based on the assumption of majority correct computational power.

The paper also does not address the problem of starvation–one participant may never have its transactions appear in the blockchain because its messages are not reaching nodes with enough computational power to get the next block.

Other work, which provides more formal analysis of the algorithm and what it can guarantee, addresses some of these concerns. For example, [1] proves that Bitcoin's algorithm should work, resilient to less than half of the system's computational power being faulty, if the communication delay is much faster than the time needed to compute a new block. If communication delay slows relative to the time to build a new block, then the system's degree of fault tolerance decreases. (Reference 15 in that paper addresses the same question experimentally.) Note that this work is limited because it makes simplifying assumptions, such as no processes entering or leaving the system, to reduce the problem to a tractable form.

Specifically, [1] introduces two properties which it can use to prove the algorithm's correct behavior. A *common prefix* property and a *chain-quality* property. Common prefix, as you might guess, states that all correct processes will have the same chain, up to some point. Specifically, if k is the number of blocks ignored at the end of the chain, the probability that the prefixes before the ignored blocks are the same grows exponentially with k. This property is the source of the result that the fraction of computation power which is malicious and the network delay are inversely related. That is, a slow network (relative to the PoW time) cannot tolerate as strong an adversary as a fast one. The chain quality property proves that the fraction of malicious blocks in the chain decreases as the ratio of malicious computing power decreases. This seems obvious, but it is important to prove that this holds! With these two properties, the paper shows that Bitcoin can tolerate only a negligible amount of malicious behavior in solving Byzantine Agreement, though it then presents a new BA algorithm which can tolerate 1/3 of the compute power in the network being faulty.

The apparent weakness of the paper is that it performs a static analysis. It makes some synchrony assumptions which may or may not affect its results. More importantly, though, it assumes a static network. That is, the proofs of correctness do not handle systems where processes can join and exit while the protocol is running. Such proofs are much harder, of course, and working in a static system is an important step to working in such a system, but remember to understand the applicability of results. (Also remember that there may be newer work proving more complex cases.)

Aside: One other interesting observation in [1] is that Bitcoin achieves "anonymity" by replacing it with "pseudonymity". That is, while your real name may not be attached to your money, your various transactions may be connected to each other, and any party that finds your identity from one may be able to associate you with all your transactions. The Bitcoin paper suggests that using different key pairs for different transactions may reduce this risk, but keys can be associated between transactions involving the same coin, so there is still some risk.

3 Proof of Work vs. Proof of Stake

A blockchain protocol at its core must decide which block will be next on the chain. Since communication is asynchronous, processes may have different sets of transactions to commit. If we simply allowed any process to add a new block, an attacker could quickly generate a large set of of dummy transactions, collect them in blocks, and hash the chain, taking over the chain. By inserting a few malicious transactions, such as double-spends, this could invalidate the usefulness of the cryptocurrency.

To prevent this, Bitcoin (and most other blockchains to date) add one more element to each block: a *nonce*, which is cryptography speak for a "number used once". That is, this is a number used to generate the block which has no further meaning. To add a block to the chain, a process must compute a value for the nonce that gives the hash a particular property. For Bitcoin, this property is that the hash must start with a certain number of zeros. Since any change in the input to a hash function leads to an unpredictable change in the output, finding a nonce which gives the hash value the desired property is typically a brute force search through the space of integers.

Since an attacker cannot add blocks to a chain more quickly than the rest of the network (by the bound on faulty computational power), the "correct" chain will be longer, and thus accepted by the system. In practice, since the algorithm is probabilistic, malicious processes may be able to build a longer branch and replace valid transactions, which are now considered not to have happened. The idea is that the longer the branch rolled back, the lower the probability that the malicious nodes will be able to find enough blocks fast enough. This means that in practice, you will want to wait until a transaction is several blocks deep in the chain so that the probability of the transaction being rolled back is very small.

3.1 Proof of Stake

Proof of Work seems to work well, and is now (relatively) firmly established in practice, as Bitcoin, Ethereum, etc. rely on it to build their blockchains. Unfortunately, PoW incentivizes spending more and more computational power simply on maintaining the currency system. This has led to huge amounts of computing hardware and energy being used, arguably to no purpose. Maintaining a currency system is, in fact, a great societal good, but the value of these independent currency systems is less obvious. Bitcoin currently uses more energy than most countries and Ethereum about a third as much[3]. This has led to such unfortunate effects as keeping open coal power plants that might otherwise close[4].

To avoid this energy consumption, there is another solution for determining which block is added to a blockchain. Called "Proof of Stake", the consensus problem of choosing a next block is done by a weighted random choice. Each process that is trying to add the next block pays a "stake", typically in the cryptocurrency built on the blockchain. Higher stakes have a higher chance to have their block accepted, but if a process publishes a bad block, it may lose its stake. There are different mechanisms for returning a stake after a transaction has become stable, or deep enough in the chain that it is unlikely to be rolled back.

- Put up a stake
- Elect a leader (probabilistic, weighted by values of stakes)

[3] `https://www.forbes.com/advisor/investing/cryptocurrency/bitcoins-energy-usage-explained/`, `https://blog.ethereum.org/2021/05/18/country-power-no-more/`

[4] `https://arstechnica.com/tech-policy/2021/09/old-coal-plant-is-now-mining-bitcoin-for-a-utility-company/`

- Leader adds a new block
- Other processes verify new block, penalize leader if invalid
- Repeat

One possible algorithm for electing a leader is that the hash of the last block in the chain is interpreted as an index into the list of all coins. Whoever holds the coin with that index is the next leader and determines the next block.[2]

There is an obvious attack here: If you can control a large fraction of the total currency, you can win leader very often, building malicious chains of blocks to override correct nodes. It might seem that the penalty for bad blocks would counteract this, but if you can control a large number of nodes, each can put up a small stake, giving you a large probability of winning with a low penalty if one node is detected as bad. This is prevented by assuming that no adversary controls more than half of the currency, instead of assuming that they do not control more than half of the compute power.

Thus, the balance with PoS is that there is much lower cost to build the blockchain, but it opens a new avenue of attack. Simply by buying enough of the cryptocurrency, you can take over the blockchain and (probably) build chains that allow you to double-spend. This is potentially difficult, as one entity controlling more than half of the total currency pool should be hard. But it might be slighly more feasible than one entity controlling more than half of the world's computing power. This is especially true of new coins, since the creators could start with a large number of coins, giving them power over the blockchain.

One possible solution is to require nodes adding blocks to put up a financial stake, but not let the probability that a given node gets to build and publish the next block be dependent on the value of the stake. That is, participating nodes are chosen uniformly at random to build the next block and put up a fixed stake. To be a candidate, you must have sufficient currency to meet the staking requirement[5]. I think this is how Ethereum works, but am not 100% certain.[6]

Proof of Stake cryptocurrencies are very recently becoming more popular, so we have less real-world experience and data on its robustness. Ethereum, the second-most-popular cryptocurrency currently in existence, just switched to PoS instead of PoW in Fall 2022.

References

[1] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[2] Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities. *IEEE Access*, 7:85727–85745, 2019.

[5] As of 4/28/23, this is 32ETH, worth roughly $60,000USD.

[6] `https://arxiv.org/pdf/2003.03052.pdf` has a fairly well written, though still very technical, discussion of the algorithmic underpinnings of PoS Ethereum.