Cryptocurrencies: Protocols for Consensus

Andrew Lewis-Pye *

February 20, 2020

The novel feature of Bitcoin [N+08] as a currency is that it is designed to be *decentralised*, i.e. to be run without the use of a central bank, or any centralised point of control. Beyond simply serving as currencies, however, cryptocurrencies like Bitcoin are really protocols for reaching consensus over a decentralised network of users. While running currencies is one possible application of such protocols, one might consider broad swathes of other possible applications. As one example, we have already seen cryptocurrencies used to instantiate *decentralised autonomous organisations* [KOH19], whereby groups of investors come together and coordinate their investments in a decentralised fashion, according to the rules of a protocol that is defined and executed 'on the blockchain'. One might also envisage new forms of decentralised financial markets, or perhaps even a truly decentralised world-wide-web, in which open-source applications are executed by a community of users, so as to ensure that no single entity (such as Google or Facebook) exerts excessive control over the flow of personal data and other information.

There are many questions to be answered before we can talk with any certainty about the extent to which such possibilities can be realised. Some of these questions concern human responses, making the answers especially hard to predict. How much appetite does society have for decentralised applications, and (beyond the possibilities listed above) what might they be? In what contexts will people feel that the supposed advantages of decentralisation are worth the corresponding trade-offs in efficiency? There are also basic technical questions to be addressed. Perhaps the best known of these is the so called *scalability* issue: Can cryptocurrency protocols be made to handle transactions at a rate sufficient to make them useful on a large scale?

In this paper, we will describe how Bitcoin works in simple terms. In particular, this means describing how the Bitcoin protocol uses hard computational puzzles in order to establish consensus as to who owns what. Then we will discuss some of the most significant technical obstacles to the large scale application of cryptocurrency protocols, and approaches that are being developed to solve these problems.

Bitcoin and Nakamoto Consensus

The Bitcoin network launched in January 2009. Since that time, the total value of the currency has been subject to wild fluctuations, but at the time of writing is in excess of $170 billion.[1] Given the amount of attention received by Bitcoin, it might be surprising to find out that consensus protocols have been extensively studied in the field of distributed computing since at least the 1970s [Lyn96]. What differentiates Bitcoin from previous protocols, however, is the fact that it is a *permissionless* consensus protocol, i.e. it is designed to establish consensus over a network of users that anybody can join, with as many identities as they like in any role. Anybody with access to basic computational hardware can join the Bitcoin network, and users are often *encouraged* to establish multiple public identities, so that it is harder to trace who is trading with whom.

*The author is a professor of mathematics at the London School of Economics. His email address is a.lewis7@lse.ac.uk.

[1] For an up-to-date value, see coinmarketcap.com.

It is not difficult to see how the requirement for permissionless entry complicates the process of establishing consensus. In the protocols that are traditionally studied in distributed computing, one assumes a fixed set of users, and protocols typically give performance guarantees under the condition that only a certain minority of users behave improperly – 'improper' action might include malicious action by users determined to undermine the process. In the permissionless setting, however, a single user can establish as many identities as they like. Executing a protocol that is only guaranteed to perform well when malicious users are in the minority, is thus akin to running an election in which people are allowed to choose their own number of votes.

In the permissionless setting, one therefore needs a mechanism for weighting the contribution of users that goes beyond the system of 'one user, one vote'. The path taken by Bitcoin is to weight users according to their computational power. This works because computational power is a scarce and testable resource. A user might be able to double their number of identities in the system at essentially zero cost, but this will not impact their level of influence. To do that, they will need to increase their computational power, which will be expensive.

Before we see how Bitcoin achieves this in more detail, we will need to get a clearer picture of how one might go about running a (centralised or decentralised) digital currency in the first place. To explain that, we will need some basic tools from cryptography.

Basic tools from cryptography

The two basic tools that we will need from cryptography are *signature schemes* and *hash functions*. Luckily, we can entirely black-box the way in which these tools are implemented. All that is required now is to understand the functionality that they provide.

Signature schemes. Presently, most cryptocurrencies use signature schemes that are implemented using Elliptic Curve Cryptography. The functionality provided by these signature schemes is very simple. When one user wishes to send a message to another, the signature scheme produces a *signature*, which is specific to that message and that user. This works in such a way that any user receiving the message together with the signature can efficiently verify who the message came from. So the use of an appropriate signature scheme means one cannot produce 'fake' messages purporting to be from other users.

Hash functions. Hash functions take binary strings of any length as input, and produce strings of a fixed length as output. Normally, we work with hash functions which produce 256 bit strings, and the 256 bit output is referred to as the hash of the input string. Beyond that basic condition, a hash function is designed to be as close as possible to being a random string generator (subject to the condition that the same input always gives the same output): Informally speaking, the closer to being a random string generator, the better the hash function. This means that a good hash function will satisfy two basic properties:

(a) Although in theory the function is not injective, in practice we will never find two strings that hash to the same value, because there are 2^{256} possible outputs.

(b) If tasked with finding a string that hashes to a value with certain properties, there is no more efficient method than trying inputs to the hash function one at a time, and seeing what they produce.

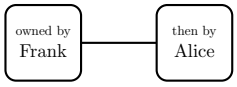
So if we are working with a good hash function, and we are tasked with finding a string of a certain length that hashes to a value starting with 10 zeros, then there is no more fundamentally efficient method than just plugging in input values, until we find one that works.

Implementing a centralised digital currency

As we have already said, it is the aim of Bitcoin to be decentralised. To understand what difficulties we face in implementing a decentralised digital currency, however, it is instructive to consider first how one might implement a *centralised* digital currency, which

works *with* the use of a central bank. Once that simple case is dealt with, we can properly analyse what difficulties arise in the decentralised case.

Presumably, we will want our currency to be divided into units, or *coins*. For the sake of simplicity, we will start by concentrating on what happens to a single coin, and suppose that this coin is indivisible. So the owner of the coin can either spend the whole coin or nothing – they are not allowed to spend half the coin. In that case, we might have the 'coin' simply be a *ledger* (i.e. an accounting record), which records its sequence of owners. A coin is thus a binary sequence, which could be visualised as below.



For now, do not not worry about how Frank came to own this coin in the first place – we will come to that later. Instead, let us consider what needs to happen when Alice, who presently owns the coin, wants to transfer it to another user. In the presence of a central bank this is simple: Alice can form a new version of the coin recording that it now belongs to the new user, Bob say, and send that new version of the coin to the central bank. In order that the central bank can be sure that the extension to the ledger really was created by Alice, though, she will need to add her signature – we will picture the relevant signature as a little black box attached to the bottom right corner of that part of the ledger. Of course, if Alice has to add her signature now, Frank will also have had to add his signature when he transferred the coin to Alice. The signature added to each extension of the ledger can be seen as testimony by the previous owner that they wish to transfer the coin to the new user. The new version of the coin can then be represented as below.

When the central bank sees the new version of the coin, they can check to see that the signature is correct, and, if so, record the transaction as *confirmed*. The use of a signature scheme therefore suffices to ensure that only Alice can spend her coin. This is not the only thing we have to be careful about though. We also need to be sure that Alice cannot spend her coin twice. In the presence of the central bank, this is also simple. Suppose Alice later creates a new version of the coin, which transfers the coin to another user Charlie instead. In this case, the central bank will see that this transaction conflicts with the earlier one that they have seen, and so will reject it.

This simple protocol therefore achieves two basic aims:

1. Only Alice can spend her coin, and;
2. Alice cannot 'double spend'.

So what changes when we try to do without the use of a central bank? Let us suppose that all users now store a copy of the coin. When Alice wishes to transfer the coin to Bob, she forms a new version of the coin, together with her signature, as before. Now, however, rather than sending it to the central bank, she simply sends the new version to various people in the network of users, who check the signature and then distribute it on to others, and so on. In this case, the use of signatures still suffices to ensure that only Alice can spend her coin. The issue is now that it becomes tricky to ensure that Alice cannot spend her coin twice. Alice could form two new versions of the coin, corresponding to two different transactions. If we could be certain that all other users saw these two versions in the same order, then there would not be a problem, as then users could just agree not to allow the second transaction. Unfortunately, we have no way of ensuring this is the case.[2]

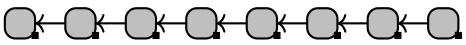
Removing the central bank

From the discussion above, it is clear that we need a protocol for establishing irreversible consensus on transaction ordering. To describe how this can be

[2]The reader is encouraged to convince themself that there is no simple solution here. For example, it might be tempting to think that one should cancel both if one sees contradictory transactions, but this will allow Alice to invalidate transactions deliberately after they are considered to have cleared.

achieved, we will initially describe a protocol that differs from Bitcoin in certain ways, and then we will describe what changes are required to make it the same as Bitcoin later.

Previously, we simplified things by concentrating on one coin. Let us now drop that simplification, and have all users store a *universal ledger*, which records what happens to all coins. We can also drop the simplification that coins are indivisible if we want, and allow transactions which transfer partial units of currency. So, according to this modified picture, each user stores a universal ledger, which is just a 'chain' of signed transactions. Each transaction in this chain might now follow an unrelated transaction, which transfers a different coin (or part of it) between a different pair of users: The universal ledger is just a chain of transactions recording all transfers of currency that occur between users.



The reader will notice that in the picture above, we have each transaction *pointing to* the previous transaction. We should be clear about how this is achieved, because it is important that we create a tamper proof ledger: We do not want a malicious user to be able to remove intermediate transactions and produce a version of the universal ledger which looks valid. What we do is to have each signed transaction include the hash of the previous transaction as part of its data. Since hash values are (in effect) unique, this hash value serves as a unique identifier.

What happens next is the key new idea:

A We specify a computational puzzle corresponding to each transaction, which is specific to the transaction, and which can only be solved with a lot of computational work. The puzzle is chosen so that, while the solution takes a lot of computational work to find, a correct solution can easily (i.e. efficiently) be verified as correct. The solution to the puzzle corresponding to a given transaction is called a 'proof-of-work' (PoW) for that transaction.

B We insist that a transaction cannot be included in the universal ledger, unless accompanied by the corresponding PoW.

Do not worry immediately about precisely how the PoW is specified – we will come back to that shortly. Now when Alice wants to spend her coin, she sends the signed transaction out into the network of users, all of whom start trying to produce the necessary PoW. Only once the PoW is found can the transaction be appended to the universal ledger. So now transactions are added to the chain at the rate at which PoWs are found by the network of users. The PoWs are deliberately constructed to require time and resources to complete. Exactly how difficult they are to find is the determining factor in how fast the chain grows.

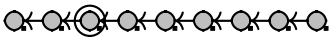
Of course, the danger we are concerned with is that a malicious user might try to form alternative versions of the ledger. How are we to know which version of the ledger is 'correct'? In order to deal with these issues, we make two further stipulations (the way in which these stipulations prevent double spending will be explained shortly):

C We specify that the 'correct' version of the ledger is the longest one. So when users create new transactions, they are asked to have these extend the 'longest chain' of transactions (with the corresponding PoWs supplied) they have seen.

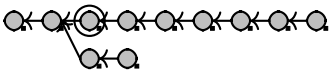
D For a certain *security parameter* k, a given user will consider a transaction t as 'confirmed' if t belongs to a chain C which is at least k transactions longer than any they have seen that does not include t, and if t is followed by at least k many transactions in C.

The choice of k will depend on how sure one needs to be that double spending does not occur. For the sake of concreteness, the reader might think of $k = 6$ as a reasonable choice.

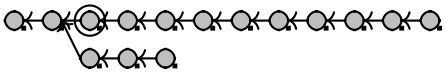
These are quite simple modifications. How do they prevent double spending? The basic idea is as follows. Suppose that at a certain point in time, Alice wants to double spend. Let us suppose that the longest chain of transactions is as depicted below, and that the confirmed transaction t that Alice wants to reverse is third one (circled).



In order to reverse this transaction, Alice will have to form a new chain that does not include t. This means branching off before t, and building from there.



For people to believe it, however, this new chain will have to be the longest chain. The difficulty for Alice is that while she builds her new chain of transactions, the *rest of the network combined* is working to build the other longer chain.



So long as Alice does not have more computational power than the rest of the network combined, she will not be able to produce PoWs faster than they can. Her chain will therefore grow at a slower rate than the longest chain, and her attempt to double spend will fail.[3] So long as no malicious user (or coordinated set of users) has more power than the rest of the network combined, what we have achieved is a tamper proof universal ledger, which establishes irreversible consensus on transaction ordering, and which operates in a decentralised way.

To finish this section, we now fulfil some earlier promises. We have to explain how PoWs are defined, what changes are necessary to make the protocol like Bitcoin, and how users come to own coins in the first place.

Defining PoWs

In fact, it will be useful to define PoWs for binary strings more generally – of course transactions are specified by binary strings of a particular sort. To do this we fix a good hash function h, and work with a difficulty parameter d, which (is not to be confused with the security parameter k and) can be adjusted to determine how hard the PoW is to find. For two strings x and y, let xy denote the concatenation of x and y. Then we define a PoW for x to be any string y such that $h(xy)$ starts with d many zeros. Given the properties of a good hash function described earlier, this means that there is no more efficient way to find a PoW for x than to plug through possible values for y, requiring 2^d many attempts on average. The expected time it will take a user to find a PoW is therefore proportional to the rate at which they can process hash values, and for larger d the PoW will be harder to find. Defining PoW in this way also means that the process by which the network as a whole finds PoWs can reasonably be modelled as a Poisson process: In any second there is some independent probability that a PoW will be found, and that probability depends on the rate at which the network as a whole can process hashes.

Using blocks of transactions

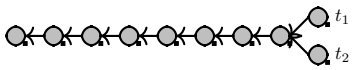
The most significant difference between the protocol we have described and Bitcoin, is that in Bitcoin the ledger does not consist of individual transactions, but *blocks* of transactions (hence the term 'blockchain'). Each block is a binary string, which contains within its data a few thousand transactions,[4] together with a hash value specifying the previous block. So now, individual transactions are sent out into the network, as before. Rather than requiring a PoW for each individual transaction, however, Bitcoin asks users to collect large sets of transactions into blocks, and only requires one PoW per block. The main reason[5] for this is worth understanding properly, because it also relates quite directly to the issue of scalability, which we will discuss in the next section. The key realisation here, is that we have to take careful account of the fact that the underlying communication network

[3] A caveat is that finding PoW is best modelled as probabilisitic. So there will be some chance Alice succeeds in double spending, but it will be small.

[4] At the time of writing the monthly mean is just over 2000 transactions per block.

[5] There is a second reason. We want the rate at which PoWs are found, rather than the rate at which users wish to execute transactions, to be the determining factor in how fast the chain grows. One PoW per transaction therefore means *requiring* a queue of transactions: If there is no queue and if users wish to execute x many transactions each hour, then x many transactions will be added to the chain each hour, and it will be the rate at which users wish to execute transactions that determines how fast the chain grows.

has *latency*, i.e. it takes time for messages to propagate through the network. This latency becomes especially problematic when we work at the level of individual transactions, since they are likely to be produced at a rate which is high compared to network latency. For the sake of concreteness, it may be useful to work with some precise numbers. So, as an example, let us suppose that it takes 10 seconds for a transaction to propagate through the network of users. Suppose that we are using the protocol as defined previously, so that PoWs are required for individual transactions, rather than for blocks. To begin with, let us suppose that the difficulty parameter is set so that the network as a whole finds PoWs for transactions once every 10 minutes on average. Consider a point in time at which all users have seen the same longest chain C, and consider what happens when a PoW for a new transaction t_1 is found by a certain user, so that t_1 can be appended to C. The PoW for t_1 then begins to propagate through the network. The crucial observation is that there is then the following danger: During those 10 seconds of propagation time, there is some chance that another user, who has not yet seen the new extended version of the ledger, will find a PoW for another transaction t_2. In this case, we now have an honestly produced *fork*, which splits the honest users.



While some users will be looking to find PoWs to extend one version of the chain, others will be working to extend the other. At least briefly, this makes it slightly easier for a malicious user to double spend, because now they only have to outcompete each component of the divided network.

In that example the chance of a fork is quite low, because PoWs are only produced once every 10 minutes on average, while propagation time is 10 seconds. If we have a PoW produced every minute on average, however, then the appearance of a fork will now be 10 times as likely. The problem we have is that, practically speaking, we will need transactions to be processed at a much higher rate than one per minute: Bitcoin can process 7 transactions a second and this is generally regarded as being unacceptably slow for large scale adoption. If PoWs are being produced at a rate of 7 per second, then we will not only see forks of the kind described above. We will see forks within forks within forks, with different honest users split between many different chains, and the security of the protocol will be dramatically compromised. This problem is avoided by using blocks, because blocks of transactions can be produced much more slowly: In Bitcoin the difficulty of the PoW is adjusted so as to ensure that one block is produced every 10 minutes on average. This means that, most of the time, all honest users will be working to extend the same chain.

Minting new coins

In Bitcoin, the users who look to provide the PoW for blocks of transactions are referred to as 'miners', and the process of searching for PoW is called 'mining'. Now, though, we have a problem of incentives to deal with. Mining costs money. There are hardware and electricity costs, amongst others. If the system is to be secure against double spending, then we certainly need lots money to be spent on mining – the security of the system is directly determined by how much it would cost a malicious user to establish more mining power than the rest of the network. To incentivise them to mine, this means that miners need to be paid in one way or another, and it is here that it is rather convenient that we happen to be designing a currency. In the context of running a currency, the solution becomes simple: We reward miners for finding PoWs by giving them currency.[6] This also, rather neatly, solves the problem as to how users come to own coins in the first place. It is when miners find a PoW that they are assigned previously unowned units of currency.

[6]It is often asked whether other forms of permissionless blockchain will have more impact than cryptocurrencies. Once one has the latter providing a tamper proof ledger, this can be used for other applications. Without using a cryptocurrency, however, the task of motivating users to follow protocol will have to be achieved by means other than payment in currency.

The issue of scalability

There are various technical issues that need to be addressed before cryptocurrencies see large scale adoption. Among the less serious of these is that Bitcoin requires fully participating users to download the *entire* ledger (presently over 200GB). Much more significant is the fact that proof-of-work protocols are energy intensive, to the point that recent estimates show Bitcoin consuming more energy than the nation of Switzerland.[7] The question that has received most attention, however, is how to increase transaction rates: While Visa is capable of handling more than 65000 transactions per second, Bitcoin can presently process 7 transactions per second. In this section, we will explain why Bitcoin processes transactions so slowly, and some proposed solutions.

The two transaction rate bottlenecks

There are two fundamental bottlenecks that limit transaction rates for cryptocurrency protocols such as Bitcoin.

The latency bottleneck. The Bitcoin protocol limits the size of blocks to include a few thousand transactions, and the PoW difficulty setting is adjusted every couple of weeks, so that blocks are produced once every 10 minutes on average. These two factors – the cap on the size of blocks and the fixed rate of block production – directly result in the limited transaction rate described above. To increase transactions rates, though, it is tempting to think that one could simply increase the size of blocks, or have them produced more frequently. To increase the transaction rate by a factor of 600, why not have blocks being produced once per second? In fact, the issue here is precisely the same as the motivation for using blocks in the first place, which we discussed in detail previously.[8] Our earlier discussion considered individual transactions, but precisely the same argument holds for blocks of transactions: The fact that the network has *latency* (blocks take a few seconds to propagate through the network) means that whenever a block is produced, there is also the possibility of an honestly produced fork in the blockchain. If we double the rate of block production, then we double the probability of that fork. If we were to have a block produced once per second on average, then we would see forks within forks within forks, and the protocol would no longer be secure.[9] Essentially the same analysis holds in the case that we increase the size of blocks, because doing so increases propagation time. This increase in propagation time similarly increases the probability of a fork.

The processor bottleneck. A basic feature of Bitcoin that distinguishes it from centrally run currencies is that all fully participating users are required to process all transactions. For some applications of blockchain technology, however, one might want to process many millions of transactions per second.[10] To achieve this (even if one solves the latency bottleneck), one needs to deal with the fundamental limitation that transactions can only be processed as fast as can be handled by the slowest user required to process all transactions. The prospect of a decentralised Web 3.0 in which all users have to process all interactions must surely be a non-starter. So how can one work around this? Limiting the users who have to process all transactions to a small set with such capabilities constitutes a degree of centralisation. Another possibility is not to require *any* users to process all transactions. For example, one might consider a process called 'sharding', whereby one runs a large number of blockchains that allow limited interactions between them, while requiring each user individually to process transactions on a small set of blockchains at any given time.

Solutions in three layers

There are a multitude of mechanisms which have been proposed with the aim of increasing transaction

[7] See https://www.cbeci.org/comparisons/.

[8] For a more detailed analysis, we refer the reader to [DW13].

[9] Of course, it might still be a good idea to increase the rate by a lower factor.

[10] It is a simplification to talk only in terms of the number of transactions. Transaction complexity is also a factor.

rates. They can be classified as belonging to three *layers*.

Layer 0. These are solutions that do not involve modifying the protocol itself, but aim instead to improve on the underlying infrastructure used by the protocol. Layer 0 solutions range from simply building a faster internet connection, to approaches such as Bloxroute [KBKS18], which change the way in which messages propagate through the network. At this point, layer 0 solutions are generally best seen as approaches to dealing with the latency bottleneck.

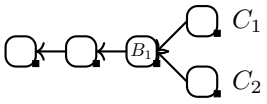
Layer 1. These are solutions which involve modifying the protocol itself, and can be aimed at dealing with either the latency bottleneck or the processor bottleneck.

Layer 2. These are protocols that are implemented *on top of* the underlying cyrptocurrency. So the underlying cryptocurrency is left unchanged, and one runs an extra protocol which makes use of the cryptocurrency's blockchain. Generally, the aim is to outsource work so that most transactions can take place 'off-chain', with the underlying cryptocurrency blockchain being used (hopefully rarely) to implement conflict resolution. To make these ideas more concrete, we will later explain the basic idea behind the Lightning Network, which is probably the best known Layer 2 solution. Layer 2 solutions are generally aimed at solving the processor bottleneck.

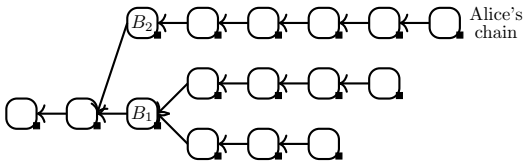
To finish this section, we will describe two well known scalability solutions. Due to the limited available space, we will not say anything further about Layer 0 solutions. We will briefly discuss a Layer 1 solution called the GHOST protocol [SZ15], which aims at dealing with the latency bottleneck. Then we will explain the basic idea behind the Lightning Network [PD16], already mentioned above as a Layer 2 solution aimed at solving the processor bottleneck.

The GHOST protocol

Recall that the latency bottleneck was caused by forks: While Bob is waiting for confirmation on a transaction in which Alice sends him money, a fork in the blockchain may split the honest users of the network. Suppose that the transaction is in the block B_1 in the picture below.



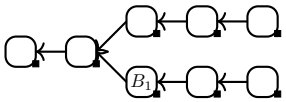
If the honest users are split between chains C_1 and C_2, then these will each grow more slowly than if there was a single chain. This makes it easier for Alice to form a longer chain.



The solution proposed by the GHOST (Greedy Heaviest Observed SubTree) protocol is simple. Rather than selecting the longest chain, we select blocks according to their total number of descendants. This means selecting the chain inductively: Starting with the first block (the so-called 'genesis' block), we choose between children by selecting that with the greatest total number of descendants, and then iterate this process to form a longer chain, until we come to a block with no children. This way B_1 will be selected over B_2 in the picture above, because B_1 has seven descendants, while B_2 only has five. So the consequence of using the GHOST protocol is that forks *after* B_1 do not matter, in the sense that they do not change the number of descendants of B_1, and so do not increase Alice's chance of double spending. We can increase the rate of block production and, although there will be an increase in the number of forks, Alice will still require more computational power than the rest of the network combined to double spend.

Unfortunately, however, this modified selection process only gives a partial solution to the latency

bottleneck. The reason is that, while forks *after* B_1 now do not matter (for confirmation of B_1), forks *before* B_1 still do. To see why, recall that, in order to be confirmed, B_1 must belong to a chain which is longer by some margin than any not including B_1.



If blocks are produced at a rate which is low compared to the time it takes them to propagate through the network, then such (possibly honestly produced) ties are unlikely to persist for long – before too long, an interval of time in which no blocks are produced will suffice to break the tie. If the rate of block production is too fast, however, then such ties may extend over long periods. This means long confirmation times.

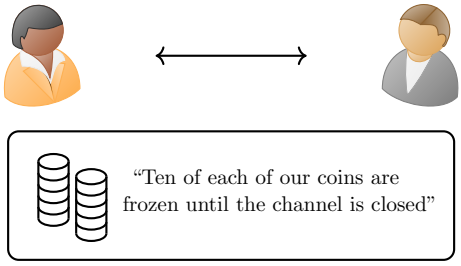
In summary, the GHOST protocol allows us to increase the rate of block production without decreasing the proportion of the network's computational power that Alice will need to double spend. If we increase the rate too much, however, this will result in extended confirmation times.

The Lightning Network

In order to explain the Lightning Network, we first need to discuss 'smart contracts'.

Smart contracts. So far, we have considered only very simple transactions, in which one user pays another in a straightforward fashion: Alice transfers funds to Bob, in such a way that Bob's signature now suffices to transfer the funds again. Bitcoin does allow, though, for more sophisticated forms of transaction. One might require two signatures to spend money, for example, or perhaps any two from a list of three signatures – so now units of currency might be regarded as having multiple 'owners'. In such a situation, where there are many forms a transaction could take, how is Alice to specify the transaction she wants to execute? The approach taken by Bitcoin is to use a 'scripting language', which allows users to describe how a transaction should work. While Bitcoin has a fairly simple scripting language, other cryptocurrencies, such as Ethereum [W+14], use scripting languages which are sophisticated enough to be *Turing complete* – this means that transactions can be made to simulate any computation in any programming language. As a mathematically minded example, (in principle) one might publish a transaction to the blockchain which automatically pays one million units of currency to anybody who can produce a (suitably encoded) proof of the Riemann Hypothesis![11] This is also a functionality whose significance depends on the information available to such computations: If reliable information on stock markets and cryptocurrency prices were to be recorded on the blockchain, then it would immediately become possible to simulate futures, options, and essentially any financial product that can be programmed using the given information. For our purposes now, the point is this. Transactions can be specified to work in much more sophisticated ways than simply transferring currency from one user to another.

A bidirectional payment channel. The aim of the Lightning Network is to allow most transactions to take place 'off-chain'. This is achieved by establishing an auxiliary network of 'payment channels'. Before coming to the network as a whole, let us consider briefly how to implement an individual channel between two users.[12]

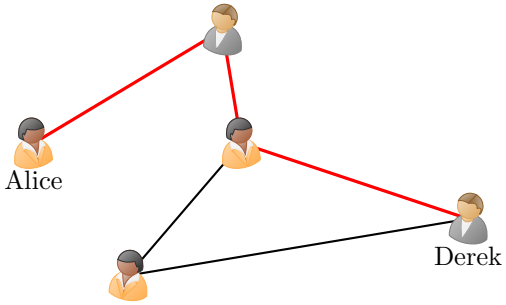


[11] While this is not presently realistic, it could soon be feasible through the use of smart contracts such as Truebit [TR18].

[12] There are a number of ways to implement these details. The Lightning Network is built specifically for Bitcoin, which means that it is designed with the particular functionalities provided by the Bitcoin scripting language in mind. For the sake of simplicity, however, we shall consider building a payment channel on top of a blockchain with a Turing complete scripting language.

So let us suppose that Alice and Bob wish to set up a payment channel between them. To initiate the channel, they will need to send one transaction to the underlying blockchain. This transaction is signed by both of them, and says (in effect) that a certain amount of each of their assets should be frozen until the payment channel is 'closed' – closing the channel has a precise meaning that we will discuss shortly. For the sake of concreteness, let us suppose that they each freeze ten coins. Once the channel is set up, Alice and Bob can now trade off-chain, simply by signing a sequence of timestamped IOUs. If Alice buys something for three coins from Bob, then they both sign a timestamped IOU stating that Alice owes Bob three coins. If Bob then buys something for one coin from Alice, they both sign a (later) timestamped IOU stating that Alice now owes Bob two coins. They can continue in this way, so long as neither ever owes the other more than the ten coins they have frozen. When either user wants to close the channel, they send in the most recent IOU to the blockchain, so that the frozen coins can be distributed to settle the IOU. We must guard against the possibility that the IOU sent is an old one, however. So, once an attempt is made to close the channel, we allow a fixed duration of time for the other user to counter with a more recent IOU.

The network. The bidirectional payment channel described above required one transaction in the blockchain to set up, and a maximum of two to close. The system really becomes useful, however, once we have established an extensive network of payment channels.



Suppose now that Alice wishes to pay Derek, but that they have not yet established a payment channel. They *could* set up a new channel, but this would require sending transactions to the blockchain. Instead, Alice can pay Derek via Bob and Charlie, if those existing channels are already in place. Of course, we have to be careful to execute this so that no middleman can walk away with the money, but this can be achieved fairly simply, with the appropriate cryptographic protocols.

Discussion

Academically, the study of permissionless distributed computing protocols is in its early phases, and is fertile territory for theoreticians, with much work to be done. Recent work [LPR20, GKL15, PSS17] has begun the process of establishing the same sort of framework for the rigorous analysis of permissionless protocols as was developed for permissioned protocols over many years. The hope is that, through the development of appropriate frameworks, a theory can be developed that probes the limits of what is possible through the development of impossibility results, as well as the formal analysis of existing protocols. Although the Bitcoin protocol was first described more than a decade ago, the original paper did not provide a rigorous security analysis. Since then a number of researchers have done great work towards providing such an analysis [Ren19, GKL15, PSS17], but the development of appropriate frameworks for security analysis remains an ongoing task. In addressing the issue of scalability, and in dealing with the substantial issues of privacy and transparency which arise in connection with the use of cryptocurrencies, there is also plenty of scope for the use of more advanced cryptographic methods such as succinct zero-knowledge proofs [BSBHR18].

Of course, there are many questions and issues that we have not had space to discuss. For example, it remains an ongoing task to develop a thorough *incentives* based analysis of Bitcoin and other protocols: The protocol may behave well when only a minority of users (weighted by computational power) behave badly, but are the other 'honest' users properly incentivised to follow the protocol? Is following the

protocol a Nash equilibrium according to an appropriate set of payoffs? In fact, these questions have been shown to be somewhat problematic for Bitcoin. There are contexts in which miners are incentivised to deviate from the protocol [ES14], and the infrequent nature of miner rewards also means that miners are incentivised to form large 'mining pools'. Today, a small handful of mining pools carry out the majority of the mining for Bitcoin, meaning that control of the currency is really quite centralised.

Earlier on, we briefly mentioned the significant issue that proof-of-work protocols are energy intensive. A viable alternative to proof-of-work may be provided by 'proof-of-stake' (PoS): With a PoS protocol users are selected to update state (i.e. to do things like publish blocks of transactions) with probability proportional to how much currency they own, rather than their computational power. PoS protocols face a different set of technical challenges [LPR20]. There are good reasons to believe, however, that as well as being energy efficient, PoS protocols may offer significant benefits in terms of increased security and decentralisation.

At this point it seems likely that very substantial increases in transaction rates will be made possible over time through a combination of approaches. At least in the short to medium term, however, if we are to see large scale adoption of cryptocurrencies, then one might conjecture that this is likely to be in applications such as the financial markets, where computational efficiency is important to a point, but where market efficiencies are key.

References

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev, *Scalable, transparent, and post-quantum secure computational integrity*, 2018. `https://eprint.iacr.org/2018/046`.

[DW13] Christian Decker and Roger Wattenhofer, *Information propagation in the bitcoin network*, Ieee p2p 2013 proceedings, 2013, pp. 1–10.

[ES14] Ittay Eyal and Emin Gün Sirer, *Majority is not enough: Bitcoin mining is vulnerable*, International conference on financial cryptography and data security, 2014, pp. 436–454.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos, *The bitcoin backbone protocol: Analysis and applications*, Annual international conference on the theory and applications of cryptographic techniques, 2015, pp. 281–310.

[KBKS18] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer, *bloxroute: A scalable trustless blockchain distribution network whitepaper*, IEEE Internet of Things Journal (2018).

[KOH19] Daniel Kraus, Thierry Obrist, and Olivier Hari, *Blockchains, smart contracts, decentralised autonomous organisations and the law*, Edward Elgar Publishing, 2019.

[LPR20] Andrew Lewis-Pye and Tim Roughgarden, *Resource pools and the cap theorem*, submitted (2020).

[Lyn96] Nancy A Lynch, *Distributed algorithms*, Elsevier, 1996.

[N+08] Satoshi Nakamoto et al., *Bitcoin: A peer-to-peer electronic cash system.(2008)*, 2008.

[PD16] Joseph Poon and Thaddeus Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat, *Analysis of the blockchain protocol in asynchronous networks*, Annual international conference on the theory and applications of cryptographic techniques, 2017, pp. 643–673.

[Ren19] Ling Ren, *Analysis of nakamoto consensus*, Cryptology ePrint Archive, Report 2019/943.(2019). https://eprint. iacr. org ?, 2019.

[SZ15] Yonatan Sompolinsky and Aviv Zohar, *Secure high-rate transaction processing in bitcoin*, International conference on financial cryptography and data security, 2015, pp. 507–527.

[TR18] Jason Teutsch and Christian Reitwießner, *Truebit: a scalable verification solution for blockchains*, 2018.

[W+14] Gavin Wood et al., *Ethereum: A secure decentralised generalised transaction ledger*, Ethereum project yellow paper **151** (2014), no. 2014, 1–32.