Cryptocurrencies - Protocols for Consensus
Professor Andrew Lewis-Pye

21 October 2020

This talk is going to be about cryptocurrencies. So, in particular that means I'm going to talk about Bitcoin, because Bitcoin is presently the best-known cryptocurrency. By the end of the talk, though, my aim is going to be much more general than that. I'm going to talk about what cryptocurrencies are, what some of the president limitations are, how those limitations might be overcome, and more generally about how the future of cryptocurrencies might play out.

Bitcoin has been with us for a few years now. It was launched in 2009 and started being reported on in the mainstream media fairly soon after that. Right from the start, though, it's been fairly controversial. For many people, when they first hear about Bitcoin, the immediate reaction is that this is just some sort of elaborate Ponzi scheme. Then, at the other end of the spectrum, there are many people for whom Bitcoin is just the beginning of a decentralisation revolution, which is about more than just currency. This is a revolution that, much more generally, is about the control of information and who controls the information. So, the basic idea is that, while Bitcoin is first presented as a currency, in fact it provides methods which can be applied to other scenarios. The initial aim of Bitcoin is to show how a *currency* can be run in a decentralised fashion without any point of centralised control — but the same methods can then be applied to other contexts, like the World Wide Web. In the context of the World Wide Web, the basic aim might be to try and ensure you don't have a small number of very powerful companies (such as Google and Facebook) controlling all of the information in ways that aren't very transparent.

Wherever you lie on that spectrum, though — whether you think that the currencies have significant potential applications, or not — one thing we can all agree on is that cryptocurrencies presently have a number of serious issues to be solved if they are to be useful on a large scale. It is a problem for Bitcoin, for example, that it has been extremely volatile. Another serious issue is often referred to as the 'scalability problem': Bitcoin can only process a small number of transactions per second. Visa can process something of the order of 50,000 transactions a second when needed. Bitcoin, on the other hand, can process something like seven transactions a second. So, this is a serious issue. If your ambition is to create a cryptocurrency which is viable as a serious large-scale currency then you're going to want to see improvements in your transaction rates or several orders of magnitude. If you're more ambitious than that, and if you want to see a decentralised web 3.0, then you're going to need to be able to process many millions or billions of transactions a second.

So those are a couple of serious issues, and there are others. What I'll do towards the end of this talk is to describe some of the most serious issues that cryptocurrencies are facing, and then we'll briefly discuss some of the potential solutions.

Before we do that though I want to take a step back. Before we get on to talk about where cryptocurrencies are heading, I want to try and describe in very simple terms what cryptocurrencies are, and how cryptocurrencies work. Then once we've gone through some of the basics, we'll get on to talk a little bit about the future of cryptocurrencies.

Some Basic Tools from Cryptography

So, first of all, let's take that step back and let's see how the Bitcoin protocol works. In order to describe the protocol, though, we'll need a couple of very basic ideas from cryptography — I promise this is not going to be a technical talk, but in order to get anywhere we do need just a couple of very basic ideas from cryptography.

Hash Functions

The first basic idea we need from cryptography is hash functions. Hash functions are very simple things, at least in terms of the service they provide, which is all we need to know about them right now. They are functions, which means they take inputs, and they give outputs. The form of input that a hash function takes is just any piece of data. You can think of data as being represented by sequences of ones and zeros, so by *binary strings*. So, what a hash function does is to take any binary string as input, and to give a binary string of a *fixed length* as output. Normally we work with hash functions which give outputs that are 256 bits long. Then it has one other important property. Whenever you feed in the same input, you have to get the same output, but, other than that, a hash function acts essentially like a random string generator. So, if I feed in the same input several times, I'm going to get the same output each time. If I then go back and I change the input by even a single bit, however, then the hash function acts like a random string generator and will very probably produce something entirely different.

An immediate consequence of that is that you are unlikely ever to find two inputs which hash to the same value. The reason is very simple. It's just because the number of possible outputs is enormous. The number of possible outputs is the number of 256-bit strings, so that's 2 to the power of 256, which is more than there are atoms in the universe. So, if we feed in two different inputs and if this function is acting essentially like a random string generator, then the probability these two different inputs happen to map to exactly the same output is astronomically small.

So that's our first basic tool, hash functions, and they are essentially just random string generators that always give outputs of the same length.

Digital Signatures

Then our second basic tool is digital signatures. Again, the functionality provided by digital signatures is extremely simple. The underlying cryptography might be clever, but we can black box all of that. All we care about is the way in which they are used, which is as follows.

Let us suppose that Alice wants to send a message to Bob. Again, the message is just a piece of data, so you can think of it as being a binary string. In that context, cryptography provides us with a clever process that Alice can go through, whereby she takes her message and she uses this clever process to produce a special extra piece of data, which is specific to the message, and which we call her signature for the message. The way in which this works, is that if she then sends the message together with the signature to Bob, then through the magic of cryptography, Bob can be sure that the message came from Alice. So, if Alice wants to send a message to Bob, she goes through this special process which produces this extra-special piece of data, we call her signature for the message, which is specific to the message.

It might be somewhat surprising that this can be done, but it can be done, and the functionality achieved is very simple: When someone sends a message, the receiver can be sure who the message came from.

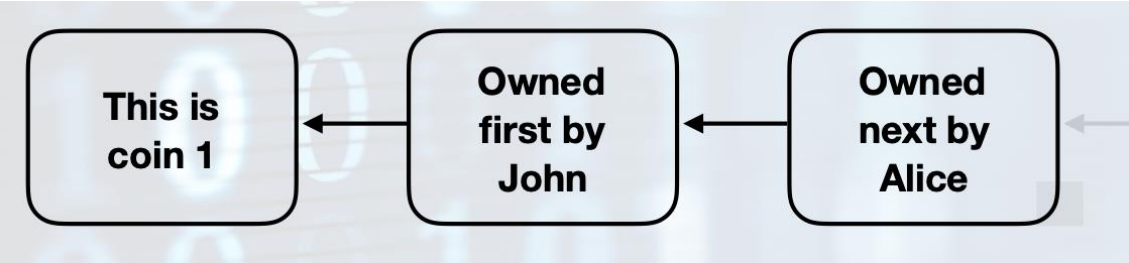
So, those are our two basic tools. We have hash functions, which are just random string generators. Then we have digital signatures and the functionality they provide is just that people can be sure who messages come from.

How To Run a Cryptocurrency

Now we have these two basic tools, let's think about how we might design our own cryptocurrency. The whole point of Bitcoin is that we want it to be *decentralised*. So, ultimately, we want to find a protocol which works without the use of a central bank. The motivation here is that centralised control — or too much centralised control — can be a bad thing. Those in control can be corrupted or otherwise compromised, and decentralised systems can be more robust. So, ultimately, we want to find a protocol which works without the use of a central bank. Just to keep things simple to start with, though, let's start off by imagining how things would work *with* the use of a central bank. Then we'll think about how to remove the use of the central bank later on.

It's presumably also the case that our currency is going to be divided up into multiple units of currency — the equivalent of many pounds or many dollars. Again, just to keep thing simple, let's start off by concentrating on what happens to a single unit of currency, a *coin* let’s say, and let's imagine that coin is indivisible.

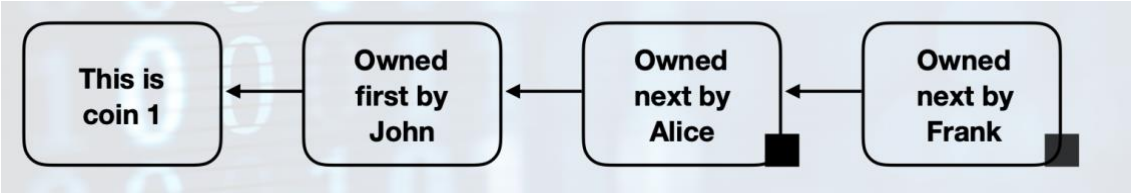
In that situation we might run things as follows. We might insist that every user of the currency keeps a *ledger* for this coin, and that ledger might look something like the figure below.



So, this is a version of the ledger that records the fact that the coin is presently owned by Alice. It also records the fact that John was the first owner of the coin. Let’s not immediately worry about how John came to end the coin in the first place. That's a slightly separate issue.

If Alice now wants to spend her coin, what she could do is to form a new version of the ledger, recording the fact that the coin is now owned by Frank. Let us suppose she wants to give Frank this coin in return for some chickens. Then she forms this new version of the ledger and sends it to the central bank. The central bank then distributes the new version of the ledger to other users. In order that only Alice can spend her coin, though, we better require that her digital signature is added to this new version of the ledger. That way only she can spend the coin, because only she can add the necessary signature. I’m picturing that as a little square block at the end of the new ledger. And, of

course, if Alice had to sign the new version of the ledger when she gave her coin to Frank, then John would had to sign the previous versions ledger when he gave the coin to Alice.



So, when Alice wants to spend her coin, she forms a new version of the ledger giving the coin to Frank. She then forwards this new signed version of the ledger to the central bank. They then check that everything is in order, that the signature is correct, and so on, and then the central bank can forward this new version of the ledger to all users. When Frank sees that the new version of the ledger has been distributed, he's happy to give Alice her chickens.

What have we achieved with this process? First of all, it is clear that only Alice can spend her coin, because only Alice can add the necessary digital signature. It is also clear that she can't spend the coin twice, because if she forms one version of the ledger giving the coin to Frank, and forwards that to the central bank, and then later on she forms a new version of ledger giving the coin to Derek, and then she forwards that one to the central bank too, then the central bank would object! So this gives us a rudimentary functioning currency.

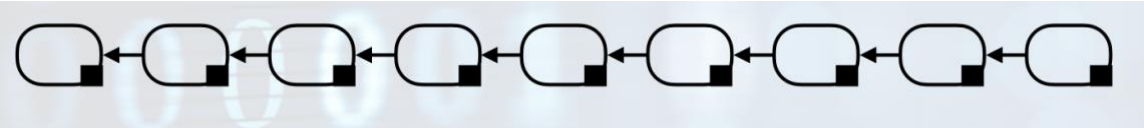
Now let us think what happens *without* use of the central bank. Let's run things in a fairly similar way, and let's suppose that when Alice wants to spend her coin, she now forms a new version of the ledger giving the coin to Frank, as before. She signs that new version of ledger. Now, though, she can't give that new version to the central bank, because there is no central bank. So, what she does is she starts distributing the new version of the ledger directly to other users of the currency. Those other users of the currency have to check that this transaction is correct — that everything is in order, that the signature is correct, and so on. Once they've done that, they will then pass the new ledger on to other users of the currency, who will proceed in the same way, and so on.

Now what goes wrong? Well, it's still the case in this situation that only Alice can spend her coin because only Alice can add the signature that is required. The problem now is that she might be able to spend the coin twice. What Alice could now do, is to form one version of the ledger giving the coin to Frank, and another version of the ledger giving the coin to Derek. Then the question is, how are we to know which version of the ledger is correct?

There are all sorts of little tricks you might try to get around the issue. For example, you might stipulate that users should believe the version they see first. Then what Alice could do is to form one version of ledger giving the coin to Frank, show some people that version first, and then form another version of the ledger giving the coin to Derrick, showing other users that version first. Or you might decide, "Well, if Alice is signing these contradictory versions of the ledger then she is clearly not acting honestly, so what we should do is just invalidate both of those transactions, and maybe we'll also cancel Alice's coin, to punish her." The problem then, is that Alice might first form a new version of ledger, giving the coin to Frank. At this point, Alice has done nothing wrong, so Frank is happy to her the chickens. Then, once she has the chickens, Alice can form a contradictory version of the ledger, giving the coin to someone else. This would then cancel the transaction giving the coin to Frank, meaning that Alice is able to take Frank's coin away. So that doesn't work either.

In fact, there really is no extremely simple fix. So, what can we do to avoid double spending? Let us take it in stages.

First of all, we were working according to the assumption that we were considering a single coin. That was just for the sake of simplicity, so let's now drop that assumption. Let's have a universal ledger which works essentially the same way as before, except that now it records what happens to *all* coins. So now, all users of the currency keep a universal ledger which records what happens to all coins. That ledger might look something like below.

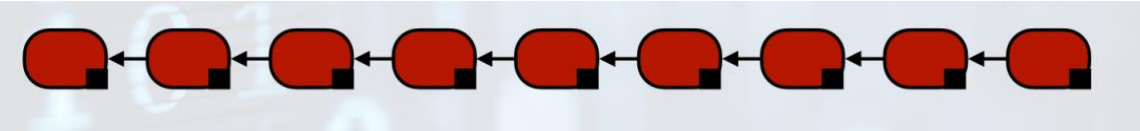


I should say at this point, that initially the way I'm setting things up here is slightly different than Bitcoin. We'll start a bit differently, and then we'll make it like a Bitcoin later on.

Let's also imagine that the way this this ledger works is that each transaction here specifies its predecessor in some way. So, the third transaction here specifies somewhere within its data that the second transaction is its predecessor. What that means is that if I'm given a valid version of the ledger, then I can't form another valid versions of ledger by, say, just removing the second transaction, and having a third one point to the first. So, we run the ledger in such a way that one can't easily tamper with the order of transactions in the ledger.

Now, what we could do to avoid double spending is to specify what we'll call a *proof-of-work* for each transaction. The proof-of-work for a transaction is just the outcome of some hard computational task, which is specific to that transaction — so that is some task which it would take the average computer a long time to carry out. Then we agree that one can only append a transaction to the ledger once the corresponding proof of work for the given transaction has been completed. If that initially sounds a little bit mysterious, don't worry. We'll specify later exactly how the proof-of-works should be defined.

So, we specify that a certain proof-of-work has to be completed before each transaction can be added to the ledger. Now what happens is that when Alice wants to spend her coin, she sends the transaction out into the network of users as a whole, and everybody starts working to trying to find the corresponding proof of work. Once somebody finds the proof-of-work, then the transaction can be put into the ledger. The way I'm depicting this in the figure below, is that once the corresponding proof of work has been found the transaction turns red. So red transactions can be added to the ledger.



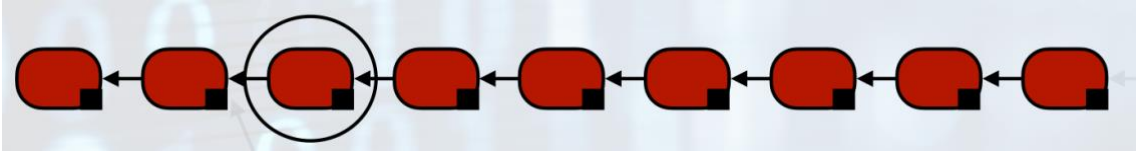
So far it might not seem that we have achieved very much. All we really seem to have done is to ensure that it's harder to add transactions into the ledger. Now, though, we just have to make two further specifications and we'll have established a protocol that suffices to avoid double spending.

First, we specify that the *correct* version of the ledger is always the longest one — that's the one with the most proof-of-work attached. So, if any user sees two different versions of the ledger, the one they will regard as correct is the longest one.

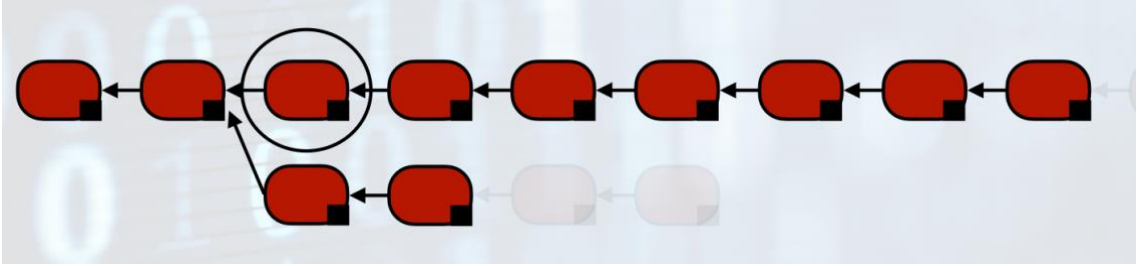
Second, we specify that a transaction is considered as *confirmed* once it's in the (correct) ledger with the appropriate proof of work, and once it's followed by *sufficiently* many transactions — what

sufficiently many means here will depend upon the level of security one wants. More security means being followed by more transactions.

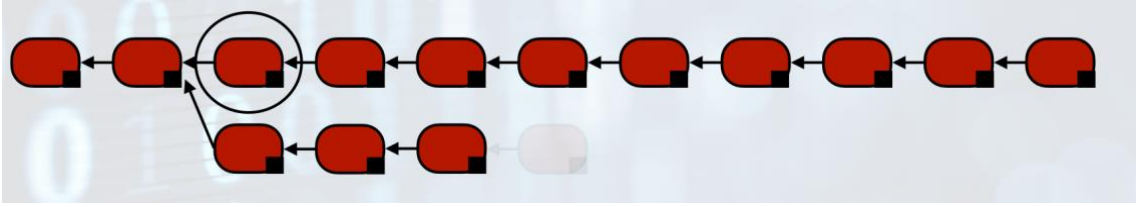
Why would these specifications suffice to avoid double spending? Let's imagine that I'm Frank and that Alice is transferring her coin to me, in return for some chickens. Suppose that the longest version of the lecture is the one in the figure below, and that the transaction I care about is the third one (circled).



If Alice wants to double spend, then she's going to have to form a new version of the ledger that doesn't include the transaction I care about. In order for anybody to believe that new version, though, it is going to have to be the longest version of the ledger. So what Alice is going to have to do, is to fork off before the transaction I care about, and she's going to have to start trying to form a new longer version as a ledger which branches off in this new direction.



For each transaction she adds in, she has to find the corresponding proof of work. She can't use transactions appearing after the one I care about, because they specify their predecessors. The problem for Alice is that while she does that, the rest of the network combined is working to extend the longest chain.



So, ultimately, in order for Alice to catch up and produce the longest chain, she's going to have to be producing proofs-of-work faster than the rest of the network combined! In order for her to double spend, in other words, she's going to need more computational power from the rest of the network combined. So that's the guarantee: A malicious user won't be able to double spend unless they have more computational power than less than network combined. So long as that's not the case then what we established is a secure and tamper proof ledger, which operates an entirely decentralised way.

Tying Up Loose Ends

There are some details left to tie up here. First of all, let us call the people looking for the necessary proof-of-work 'miners'. A first issue is that producing all these proofs-of-work will be expensive (it better be, otherwise, a malicious user will be able to afford to do too much of it)! There are substantial

hardware, electricity and other costs involved. Miners won't do their job for free, so we better pay them for their effort. How can we do that? Well, it's at this point that it is convenient that we are defining a currency. What we can do is simply to give miners some units of currency whenever they find an appropriate proof-of-work. So that is why people 'mine' Bitcoin.

A second detail to be tied up here is that so far, I've considered a version of the ledger in which we append individual transactions. Actually, it turns out if we work that way, we'll end up with all sorts of timing issues. The basic problem is that the underlying communication network has some *latency,* which means that it takes time for messages to travel across the network. Transactions will normally be produced at a rate which is high compared to network latency. As a result of that latency, users will tend to see transactions arriving in different orders, and this will often mean that honest users get split between several competing longest chains. This ends up harming the security of the protocol. It therefore turns out to work much better if we have the miners group transactions together into large blocks of transactions (with maybe a few thousand transactions per block). Then, rather than requiring a proof-of-work for each transaction, now we just require a proof-of-work for each block. That's the way that Bitcoin works, and the difficulty of the proof-of-work is adjusted so that on average we have one new block produced once every ten minutes.

One final detail to tie up is that I said earlier that I would specify precisely how each proof-of-work should be defined. We can do this by using an agreed-on hash function. To specify the proof-of-work required for a block of transactions, we take the data for the block and then we agree that the required proof of work is a *nonce* for the block: A nonce is a binary string that we can append to the block data so that the extended sequence hashes to a string ending in k many zeros — here k is a variable which is chosen so as to tailor the difficulty of finding the nonce. If k is one, then on average this means that we are going to have to try two nonces before we find one that works. If k is two, then on average we will have to try four different nonces. In the general case, the average number of tries is two to the kth power. If we make k large, then finding a nonce for the block becomes a hard-computational task. By playing with different values for k, we can make the task of finding a nonce (i.e. the task of finding a proof-of-work) precisely as easy or hard as we want it to be.

So that's basically how Bitcoin works. Now let's think a bit about some of the limitations of the protocol.

Limitations of Bitcoin

Of course, Bitcoin has been a tremendous success in many ways, and presently has a market capitalisation of around $200 billion. It is important to properly understand its limitation, though, and there are quite a few.

First of all, there is the fact that Bitcoin mining is extremely energy intensive. Recent estimates show Bitcoin mining consuming more energy that the entire nation of Switzerland! In the present climate situation that certainly seems like something to be avoided if possible.

One can also ask interesting questions about the security of the protocol. So far, Bitcoin has proved to be very robust, and we argued previously that it should be secure so long as no attacker has more computational power than the rest of the network combined. So that sounds pretty safe. Things become a little less straightforward, however, when one performs a more careful analysis, and thinks in terms of attacks which aren't designed just to double spend, but rather to destroy the currency — attacks which are designed to repeatedly double spend and so stop the currency functioning at all. In that context, we know that an attacker will need more than 50% of the total mining power. But then one can ask, what percentage of the total value of the currency would it cost to buy that much

computational power? If it only costs 1% of the total value of the currency, then this potentially leaves things open to a devastating attack, and there are strong reasons why a figure of this sort of order has to hold for proof-of-work protocols like Bitcoin. So far nobody has attempted such a thing. Why would they spend all that money to bring down Bitcoin? If this was a vital part of our infrastructure, perhaps that is a different scenario.

Then a third extremely serious problem is the issue of transaction rates, which I mentioned right at the start of the talk. Visa can process something of the order of 50,000 transactions a second when needed. Bitcoin can handle seven. If you want Bitcoin or other cryptocurrencies to be operating in the mainstream, then this is very clearly something that has to be solved.

So those are some of the limitations, and there are others. Now let's take a brief look at possible solutions.

For the first two of the limitations that we listed there, a solution that is already very well understood is to replace proof-of-work with something that is called 'proof-of-stake', and which is probably best explained by comparing how it works with proof-of-work. In a proof-of-work protocol like Bitcoin, certain users are selected and are allowed to update state by appending new blocks to the ledger. For each user, the probability that they get to append the next block, i.e. the probability that they find the next proof-of-work, is proportional to their computational power. In a proof-of-stake protocol, we select users to update state (e.g. produce blocks) in the same way, but now they are selected with probability proportional to how much currency they own. Proof-of-stake protocols bring with them their own set of problems, but they are now quite well understood. They solve the energy issue and are potentially also much more secure than proof-of-work protocols.

For the third issue of transaction rates, we need to understand the problem a little better before we can talk about solutions. Why should Bitcoin have such low transaction rates in the first place?

It turns out that there are two bottlenecks here. Two bottlenecks, and then solutions in three layers — we'll come to the solutions in a bit. Roughly speaking, the first bottleneck stems from *network latency*, by which we mean that it takes time for information to travel across the underlying communication network. If Bitcoin blocks are produced too fast, then this network latency causes confusion, and causes the protocol to be less secure. We'll see why in more detail in a moment. Then the second bottleneck stems from the very basic fact that if all nodes of the network are required to verify and record all transactions, then you can only go as fast as your slowest node.

Now let's briefly consider each of those bottlenecks in more detail. As I said, the first scaling bottleneck is caused by the fact that it takes time for information to travel across the underlying communication network. When a Bitcoin miner appends a new block to the ledger, it takes time for this new block to propagate through the network. When two blocks are found almost simultaneously, this means that some nodes of the network will see one block first, while other nodes will see the other block first. This causes different versions of the 'longest chain' and splits the honest nodes of the network. Now some honest miners will be working to extend one chain, while others will be working to extend another. This makes it easier for a malicious user to double spend. With Bitcoin these splits happen quite infrequently, but if you double the rate at which blocks are produced, then they will happen twice as much. If we produce a block every 5 seconds or so, then we would see forks within forks within forks, and chaos would ensue. So this is the first scaling bottleneck: Network latency means blocks cannot be produced too fast without sacrificing security.

Then the second scaling bottleneck is perhaps more basic, more fundamental. So long as all users have to verify all transactions, this limits the rate at which they can be processed. How significant this limitation is depends upon the level of your ambition. If you only want to process a thousand

transactions a second, then it might suffice to overcome the first scaling bottleneck. In a decentralised Web 3.0, however, we couldn't possibly have users verifying the actions of all users! So, for that sort of application, one is going to have to deal with the second scaling bottleneck as well.

Then, to talk just briefly about the solutions, we can think of these as existing in three layers.

Layer 0 solutions involve improvements to the underlying infrastructure used by the protocol. If internet speeds improve then this increases transaction rates. Layer 0 solutions help overcome the first scaling bottleneck.

Layer 1 solutions are those which involve changes to the underlying protocol itself. These might be aimed at overcoming either the first or second scaling bottleneck. Since Bitcoin was launched over ten years ago, many alternative protocols have been suggested and developed, and many of those have much higher transaction rates than Bitcoin. 'Sharding' is another interesting layer 1 solution, which involves running multiple blockchains simultaneously.

Then layer 2 solutions are perhaps the most potent. These are solutions which 'sit on top' of the underlying cryptocurrency. So, you take an underlying cryptocurrency, like Bitcoin. Then, without necessarily making any changes to that cryptocurrency itself, you build an auxiliary protocol on top, which makes use of the underlying cryptocurrency blockchain. The basic idea is that most interactions between users should be able to take place "off-chain" and within this auxiliary protocol, so long as those off-chain interactions produce enough evidence for disputes to be resolved on chain if necessary. A famous example of a layer 2 solution is the Lightning Network, which has been built to operate on top of Bitcoin, and which allows for high transaction rates.

The Future of Cryptocurrencies

To finish with, let's just very briefly talk about what the future holds. Making predictions is made especially hard by the human element to all of this, but we can talk with some authority about what should be technologically possible. Then how things pan out will depend on the human element also — one of the big questions here is as to how much appetite there is in the mainstream for decentralised technologies.

One of the main limiting factors for Bitcoin has been this question of transaction rates. Can very high transaction rates be achieved? The answer is "yes, absolutely". The way in which this plays out, though, will depend on the human element. If users stay loyal to Bitcoin and it remains the number one cryptocurrency, then truly fundamental changes to the underlying protocol are difficult. In this case you are really relying on layer 2 solutions such as the Lightning Network to facilitate high transaction rates. If, on the other hand, some combination of layer 1 and layer 2 solutions is possible, i.e. if we can replace Bitcoin with a more efficient protocol *and* build layer 2 solutions on top of that, then this allows for a much stronger solution in terms of transaction rates.

What role will cryptocurrencies play in the future? Many proponents of Bitcoin talk excitedly about it replacing fiat currencies such as the pound and the dollar. I think the truth is that this isn't likely to happen in the short term, and if it did it would be a scary thing. With currencies such as the pound, governments and central banks have all sorts of ways of responding to changing economic environments. They can do things like applying quantitative easing, or changing interest rates, and in this way, they can stimulate the economy when deemed necessary, and reign things in at other times. Such mechanisms for control are presumably necessary, and it's not clear what exactly would be their replacement in a fully decentralised economy.

If there is strong appetite for decentralised applications, though, then what we'll see is cryptocurrencies taking on a role that exists alongside existing fiat currencies. These are protocols that provide methods of decentralisation that can be applied to contexts such as the world wide web, and the financial markets. Whenever one looks to apply these decentralisation techniques, there are efficiency trade-offs, in the sense that decentralised solutions will normally be less computationally efficient. So, at least in the short term, if we are to see the mainstream adoption of cryptocurrencies then one might expect it to be in applications such as the financial markets, where computational efficiency is important to a point, but where market efficiencies are key.