CRYPTO WORDS

CY18 Q4 November

A collection of Bitcoin commentary from the brightest minds in the crypto community.

Contents

Goals and Scope

Crypto Words is a journal of Bitcoin commentary, established February 13, 2019. Its purpose is to document and advance commentary and research in disciplines of particular interest to the Bitcoin community. The journal is broad in scope, publishing content from original research, essays, blog posts, and tweetstorms from a wide variety of fields, especially governance, technology, philosophy, politics, and economics, but also legal theory, history, criticism, and social or cultural analysis. Its broader mission is to capture the conversations and think pieces in the Bitcoin space for current and future researchers. *Crypto Words hopes to* continue and expand the tradition established by publications such as the *Journal of Libertarian Studies* and *Libertarian Papers*.

History

There exists a gap in Bitcoin publishing. For authors with commentary and scholarly papers on topic, the choice of publication outlets is relatively limited. The number of journals that serve as outlets for crypto research is in any event too small, as the number of crypto thinkers continues to grow with every market cycle.

This generation of Bitcoin thinkers have limited places to submit thought pieces for publication. Content is scattered across the web, and in some cases behind paywalls which prevent the free flow of information. With the advent of the Twitter and blogging, authors also now have the option of self-publishing: they post the content to their own site or some private site, link it in a blog post, or post a working paper. But this is obviously not the best way to document and publish. What is needed is a journal that takes full advantage of the possibilities of the digital age as a go to resource for think pieces in the crypto space.

Enter *Crypto Words*. Published independently, *Crypto Words* is a journal that welcomes submissions on a range of topics of interest to the crypto community. In addition to conventional research articles, we welcome review essays blog posts, tweets as well as papers in other formats, such as distinguished lectures. Finally, wherever possible, content on this site is licensed under a Creative Commons Attribution 4.0 License. Authors retain ownership without restriction of all rights under copyright in their articles. *Crypto Words* is open access, and we encourage readers to "read, download, copy, distribute, print, search, or link to the full texts of these articles…or use them for any other lawful purpose." We want our ideas read, spread, and copied. We welcome discourse and debate.

Support Crypto Words

The posts and journals published here have been carefully curated and crafted as a true labor of love. If you've found any of this content useful here's how to show your thanks and keep the project going.

Send Bitcoin tippin.me Send CashApp Send PayPal

Spread the word

Have a website or use social networking sites like Twitter, Facebook, or LinkedIn? Please consider sharing the content found on Crypto Words or linking to https://cryptowords.github.io.

Follow us on social media

We post regularly on Twitter and use it as our main form of communication. — We don't rapid fire posts but add commentary where we see fit. Posts are typically links to our content here, trolling nocoiners, sarcastic remarks, and other things regarding development of this site.

If these sorts of things interest you, follow along on:

Twitter

Subscribe to our newsletter

We publish our journal monthly and share it via Twitter and via newsletter. Consider subscribing to the newsletter. If you're not on Twitter all day, it might make sense to subscribe so you never miss a publication.

Our pledge

- We will never sell you out.
- We will never shill you shitcoins.
- We will only deliver what is promised.

Subscribe

Transaction count is an inferior measure

By Nic Carter

Posted November 10, 2018

It is popular to measure Bitcoin by looking at its daily transaction count or 'tps'— transactions per second. New blockchains often advertise tps rates in the millions or billions and sneer at Bitcoin's measly 3 - 4 tps rate (it fluctuates between 200,000 and 350,000 transactions per day).

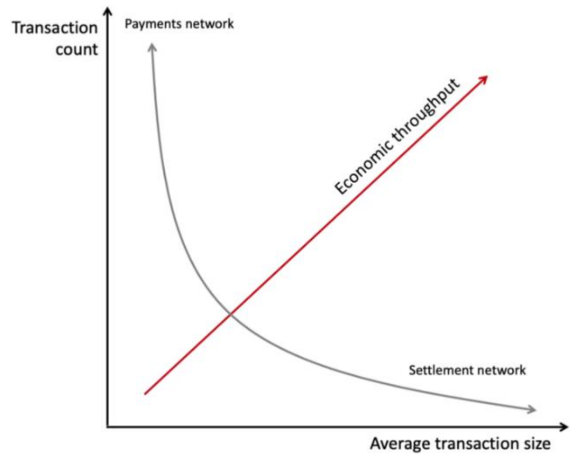
But this is an incomplete picture. There are at least three important variables to consider when holistically appraising a value transfer system, and the transaction rate is just one of them:

- Transaction capacity (tps)
- Typical transaction characteristics (transaction size)
- Settlement assurances

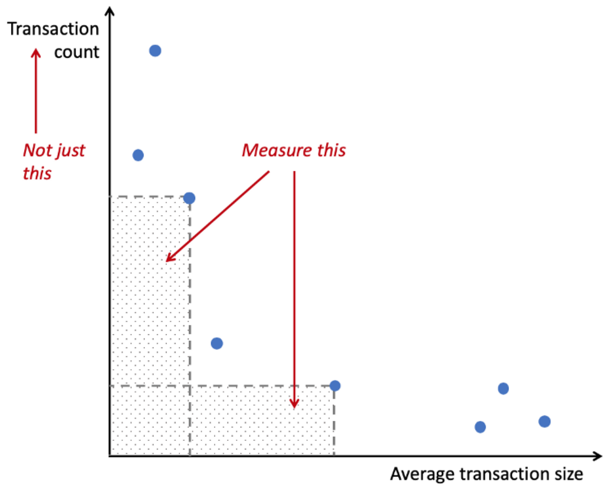
Together, transaction rate and average transaction size give you the economic throughput of the system; a measure of its financial bandwidth per unit of time. The settlement assurances tell you what *sort* of a system it is. How certain are you that you won't face a chargeback or be defrauded? Does your transaction settle immediately, like physical cash? Or is there a 90-day chargeback period prior to settlement, as with most credit cards?

Even though Bitcoin is virtual, it is closer to a physical settlements system than a deferred-settlement system like credit cards. Bitcoins are virtual bearer instruments: if you possess the private key which unlocks an unspent output, you are the owner of those coins and entitled to do with them whatever you like. That asset is no one's liability. In this respect, it more closely resembles physical cash or a commodity like gold. There is no recourse if you lose your coins or send them to the wrong address. Of course, recourse has its advantages in certain contexts; and the overhead for dealing with reversed transactions and fraud is exactly why credit fees exist. The costs of credit fraud and the remediation infrastructure are passed on to end users in the form of merchant fees. That's why you can't typically do a credit transaction for less than a dollar or two.

So how do you consistently compare dissimilar value-transfer systems like Bitcoin, Paypal, Visa, SWIFT, and physical cash? I'd recommend broadening your focus beyond simple transaction count. That's only one component of economic throughput.



The chart below shows an idealized comparison of various value transfer systems along the axes of transaction count and typical transaction size. To simplify, payment networks consist of many small transactions, whereas settlement networks will exhibit fewer, but larger transactions (in dollar terms).

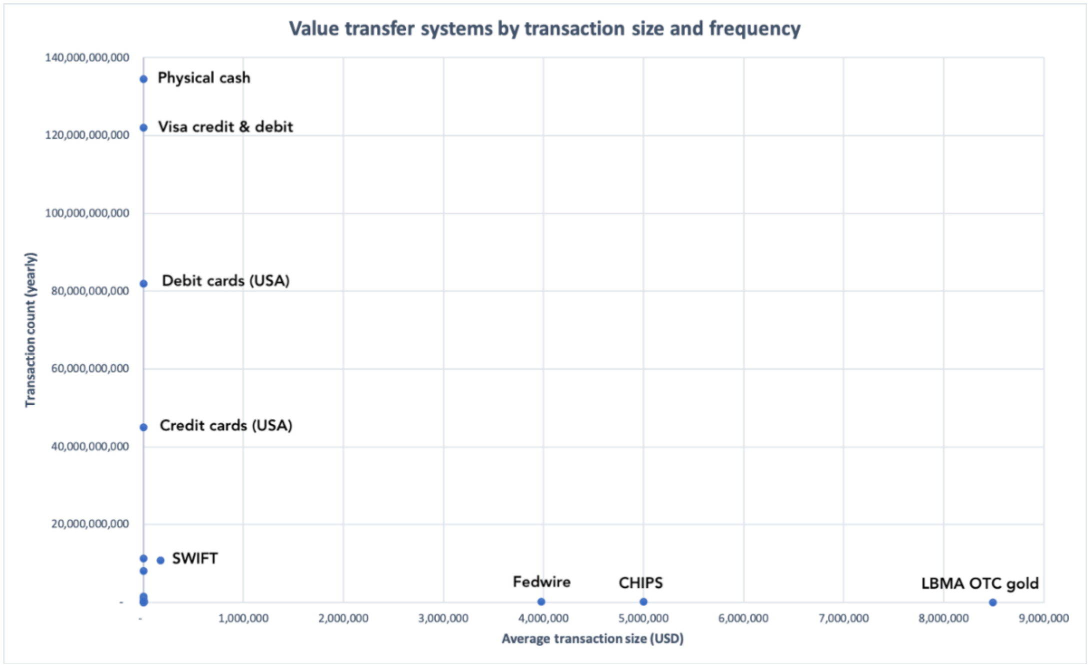


What we should really be interested in is the area under each point: that's **average transaction size * the number of transactions**, which gives us the value flowing through the system per unit of time.

To further simplify, what you find if you aggregate data from a variety of networks is that they tend to cluster at one end or the other.

I'm oversimplifying of course, but you get the idea. Let's plug some real numbers into the idealized model. I went ahead and found data for physical cash transactions, credit/debit transactions, mobile p2p payments, various popular cryptocurrencies, gold settlements, and various settlement and financial infrastructure systems (ACH, Fedwire, CHIPS, SWIFT).

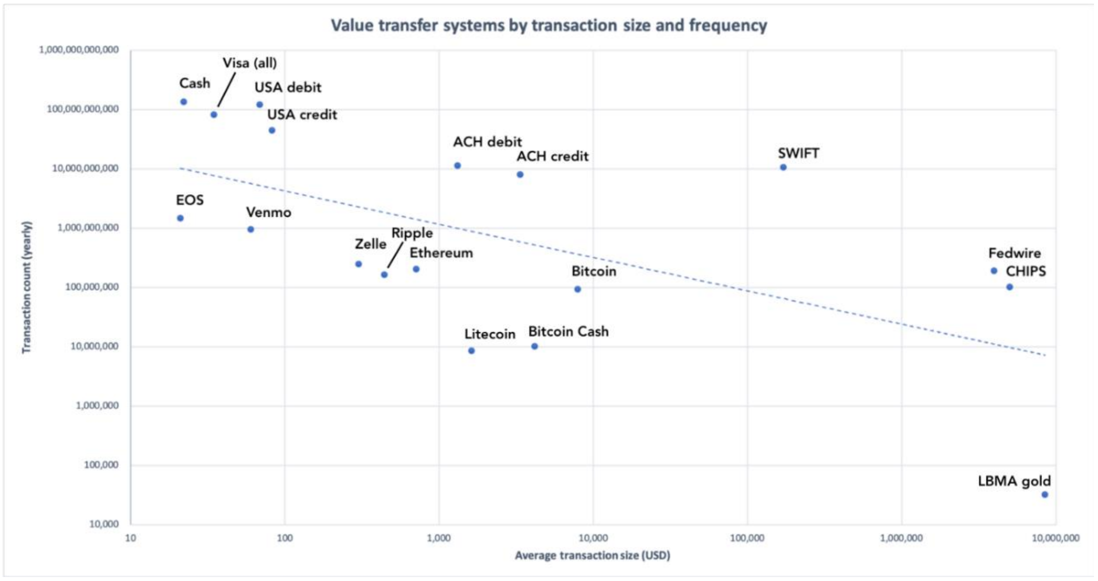
The real numbers chart looks like an even more exaggerated version of the L in my toy model, because the systems vary on many orders of magnitude along the two axes.



Sources in appendix

On the top left, you have physical cash and Visa credit/debit transactions, which have extremely high volume and smallish per-transaction sizes. On the bottom right, you have the massive settlement networks, with the gold settlements on the London market taking the crown for per-transaction size. One caveat: SWIFT is a messaging service, so value does not really flow through the system. The numbers listed for SWIFT are mostly academic in nature.

Let's take the log view so we can cram everything in there.



Blue dotted line represents the exponential best fit

This is the most comprehensive view. It demonstrates clearly that you have settlement-like networks towards the bottom right, and payment-like networks on the top left. Bitcoin is edging closer and closer to the bottom right, as base-layer transactions encode progressively more value. Let's exclude the megalarge settlement networks and focus on the payments networks.

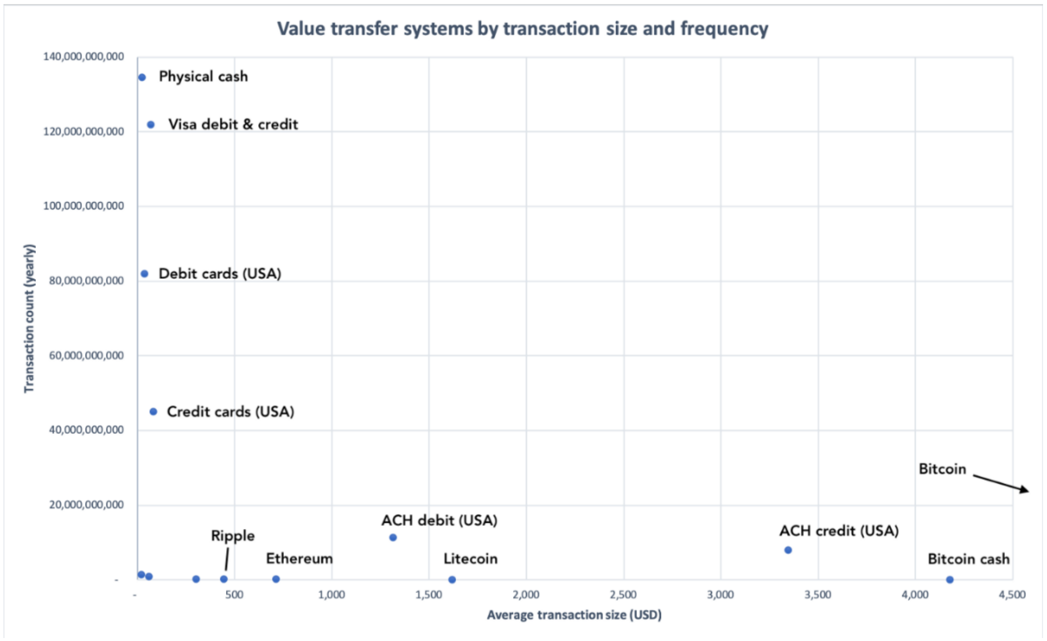


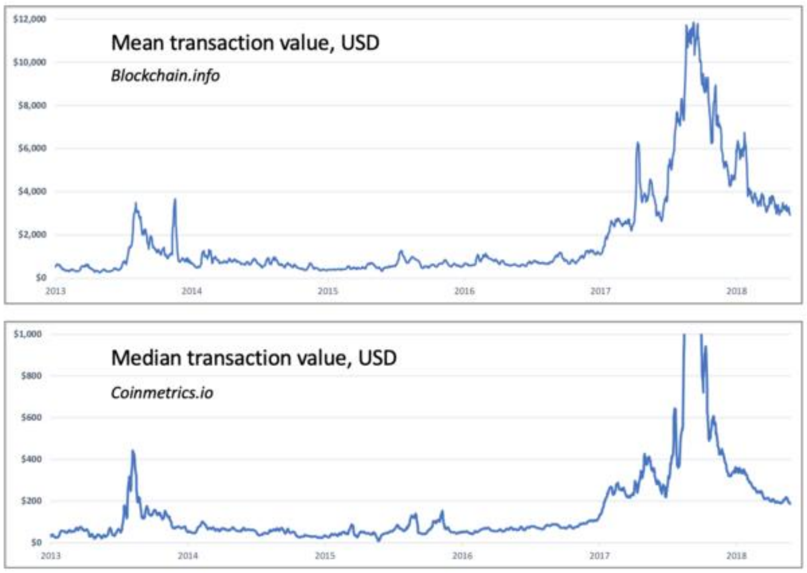
Chart truncated to show a subset of the data

As you can tell, most cryptocurrencies can barely lift themselves off the X axis — they don't have the throughput, or crucially, the adoption, to work as mainstream payments networks.

So what is Bitcoin for, anyway?

As shown on the chart, Bitcoin transactions tend to be quite large. It's hard to know the precise number, but your average transaction will be in the thousands of dollars, possibly tens of thousands. Your median transaction is well over $100.

Container ships, not parcels



Slide from 'Bitcoin as a novel economic system,' my presentation at Baltic Honeybadger 2018

Large transactions are partly out of necessity: exchanges will often batch together many cashouts to investors into a single transaction, to reduce the per-payment overhead, which increases transaction size; and fees in the single-digit dollars mean that only larger transactions are worth it from a fees perspective.

More saliently, though, the strong assurances of global final settlement that Bitcoin provides are simply not required for low-value transactions. Typical usage of the system tends to be *scaled to the assurances that the system provides* . You don't drive an armored personnel carrier down the street to get a packet of cigarettes— you don't need those security guarantees. You don't wear body armor when you walk to work. You don't put your petty cash in a safe and handcuff it to yourself when you buy groceries.

What critics miss when they fixate on TPS is the simple fact that the users of these systems tend to have a good idea of what they want from them. Low-stakes, small value transfers with some reversibility guarantees work just fine on Venmo, Paypal, or Visa. Yes—these don't work for the unbanked, but then again virtually no financial infrastructure does. This stuff takes a long time to build, as does the trust in the system.

What a low bandwidth-in-bytes but high bandwidth-in-dollars system like Bitcoin works for is large, inter-jurisdictional, money-for-enemies transactions where mutual trust is lacking and rapid settlement is desired. If mutual trust is present, or deferred settlement is acceptable, or the payment size is very small, base-layer-bitcoin is not the system for you! Vanilla Bitcoin is useful for a tiny fraction of all of the payments or value transfers that happen on a given day. And that's fine!

Bitcoin lives on the bottom right size of the size/frequency chart. Those 3 or 4 transactions which happen per second on Bitcoin are exactly what its users are after. That very particular, idiosyncratic use case doesn't really overlap with systems like Visa, Paypal, Alipay, or Venmo. They aren't teleologically comparable: *they aren't trying to do the same thing.*

Is it fair to compare them? Yes, but only if you are mindful of the tradeoffs made by each system.

"Why can't fruit be compared"—Lil Dicky

Additionally, Bitcoin's near-immediate physical settlement opens up a large design space allowing developers to add cheaper, more convenient payments networks with deferred settlement. This is Lightning and other second-layer solutions: cheap, near-instantaneous transfers which periodically settle to the base layer. The truth is that not all transactions merit final settlement within minute of occurring; for these petty cash situations you are better off using a higher layer. Bitcoin—and increasingly, Ethereum too—is developing precisely with this in mind.

So, in short, value transfer systems vary along at least three major axes, not just one. The response to "Our system does 500,000 TPS" is "at what cost?" Are you deferring settlement? Do you have a single validator? Do you require that transactors be part of the US-controlled financial system?

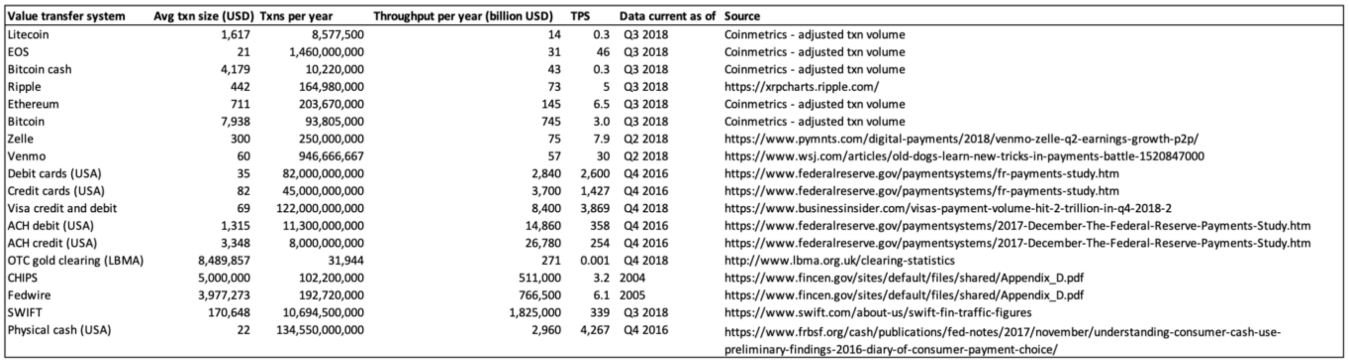
In my view, economic throughput—the dollar equivalent value flowing through the system per unit of time—is a far more comprehensive view of the system than the simple message-send-rate (TPS). Measure the area under the point, not just the Y value. From that more holistic perspective, Bitcoin is a single order of magnitude away from Visa.

Appendix

For data collection, recency was sought. Data for major cryptocurrencies comes from coinmetrics.io. Both adjusted transaction volume and transaction count were smoothed on a trailing 30-day basis and annualized. Raw or nominal transaction volumes for cryptocurrencies are noisy and bias the figures upwards, and so Coinmetrics performs transparent adjustments and de-noising to render the figures more conservative.

Data for legacy financial systems is found in 10K reports, news articles, and academic studies and annualized. Figures for CHIPS and Fedwire are somewhat out of date and hence should be interpreted as rough order of magnitude estimates.

SWIFT data is taken from the most recent quarter and annualized but does not imply settlement volume but rather financial messaging.



Value transfer system	Avg txn size (USD)	Txns per year	Throughput per year (billion USD)	TPS	Data current as of	Source
Litecoin	1,617	8,577,500	14	0.3	Q3 2018	Coinmetrics - adjusted txn volume
EOS	21	1,460,000,000	31	46	Q3 2018	Coinmetrics - adjusted txn volume
Bitcoin cash	4,179	10,220,000	43	0.3	Q3 2018	Coinmetrics - adjusted txn volume
Ripple	442	164,980,000	73	5	Q3 2018	https://xrpcharts.ripple.com/
Ethereum	711	203,670,000	145	6.5	Q3 2018	Coinmetrics - adjusted txn volume
Bitcoin	7,938	93,805,000	745	3.0	Q3 2018	Coinmetrics - adjusted txn volume
Zelle	300	250,000,000	75	7.9	Q2 2018	https://www.pymnts.com/digital-payments/2018/venmo-zelle-q2-earnings-growth-p2p/
Venmo	60	946,666,667	57	30	Q2 2018	https://www.wsj.com/articles/old-dogs-learn-new-tricks-in-payments-battle-1520847000
Debit cards (USA)	35	82,000,000,000	2,840	2,600	Q4 2016	https://www.federalreserve.gov/paymentsystems/fr-payments-study.htm
Credit cards (USA)	82	45,000,000,000	3,700	1,427	Q4 2016	https://www.federalreserve.gov/paymentsystems/fr-payments-study.htm
Visa credit and debit	69	122,000,000,000	8,400	3,869	Q4 2018	https://www.businessinsider.com/visas-payment-volume-hit-2-trillion-in-q4-2018-2
ACH debit (USA)	1,315	11,300,000,000	14,860	358	Q4 2016	https://www.federalreserve.gov/paymentsystems/2017-December-The-Federal-Reserve-Payments-Study.htm
ACH credit (USA)	3,348	8,000,000,000	26,780	254	Q4 2016	https://www.federalreserve.gov/paymentsystems/2017-December-The-Federal-Reserve-Payments-Study.htm
OTC gold clearing (LBMA)	8,489,857	31,944	271	0.001	Q4 2018	http://www.lbma.org.uk/clearing-statistics
CHIPS	5,000,000	102,200,000	511,000	3.2	2004	https://www.fincen.gov/sites/default/files/shared/Appendix_D.pdf
Fedwire	3,977,273	192,720,000	766,500	6.1	2005	https://www.fincen.gov/sites/default/files/shared/Appendix_D.pdf
SWIFT	170,648	10,694,500,000	1,825,000	339	Q3 2018	https://www.swift.com/about-us/swift-fin-traffic-figures
Physical cash (USA)	22	134,550,000,000	2,960	4,267	Q4 2016	https://www.frbsf.org/cash/publications/fed-notes/2017/november/understanding-consumer-cash-use-preliminary-findings-2016-diary-of-consumer-payment-choice/

Building a fundamental piece of technology that will bring Bitcoin to the next billion users? Reach out: castleisland.vc .

How Does Distributed Consensus Work?

##An overview of key breakthroughs in blockchain technology — and why Nakamoto Consensus is such a big deal

By Preethi Kasireddy

Posted November 13, 2018

Distributed systems can be difficult to understand, mainly because the knowledge surrounding them *is* distributed. But don't worry, I'm well aware of the irony. While teaching myself distributed computing, I fell flat on my face many times. Now, after many trials and tribulations, I'm finally ready to explain the basics of distributed systems to you.

Blockchains have forced engineers and scientists to re-examine and question firmly entrenched paradigms in distributed computing.

I also want to discuss the profound effect that blockchain technology has had on the field. Blockchains have forced engineers and scientists to re-examine and question firmly entrenched paradigms in distributed computing. Perhaps no other technology has catalyzed progress faster in this area of study than blockchain.

Distributed systems are by no means new. Scientists and engineers have spent decades researching the subject. But what does blockchain have to do with them? Well, all the contributions that blockchain has made wouldn't have been possible if distributed systems hadn't existed first.

Essentially, a blockchain is a new type of distributed system. It started with the advent of Bitcoin and has since made a lasting impact in the field of distributed computing. So, if you want to really know how blockchains work, a great grasp of the principles of distributed systems is essential.

Unfortunately, much of the literature on distributed computing is either difficult to comprehend or dispersed across way too many academic papers. To make matters more complex, there are hundreds of architectures, all of which serve different needs. Boiling this down into a simple-to-understand framework is quite difficult.

Because the field is vast, I had to carefully choose what I could cover. I also had to make generalizations to mask some of the complexity. Please note, my goal is not to make you an expert in the field. Instead, I want to give you enough knowledge to jump-start your journey into distributed systems and consensus.

After reading this post, you'll walk away with a stronger grasp of:

- What a distributed system is,
- The properties of a distributed system,
- What it means to have consensus in a distributed system,
- An understanding of foundational consensus algorithms (e.g. DLS and PBFT), and
- Why Nakamoto Consensus is a big deal.

I hope you're ready to learn, because class is now in session.

What Is a Distributed System?

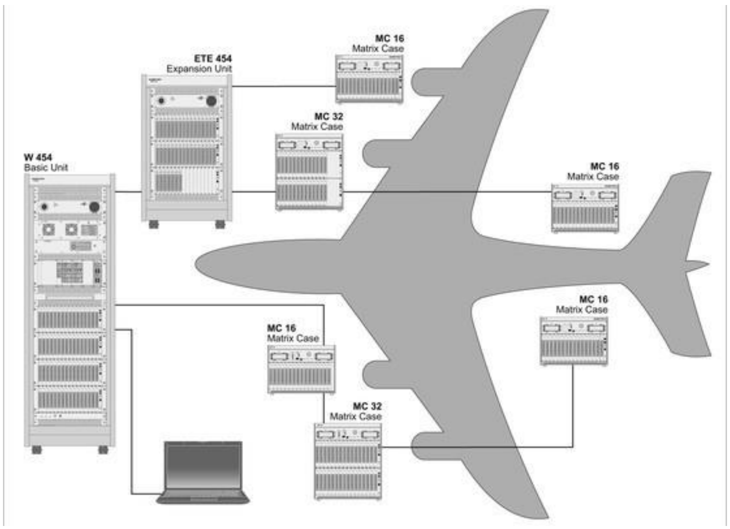
A distributed system involves a set of distinct processes (e.g., computers) passing messages to one another and coordinating to accomplish a common objective (i.e., solving a computational problem).

A distributed system is a group of computers working together to achieve a unified goal.

Simply put, a distributed system is a group of computers working together to achieve a unified goal. And although the processes are separate, the system appears as a single computer to end-user(s).

As I mentioned, there are hundreds of architectures for a distributed system. For example, a single computer can also be viewed as a distributed system: the central control unit, memory units, and input-output channels are separate processes collaborating to complete an objective.

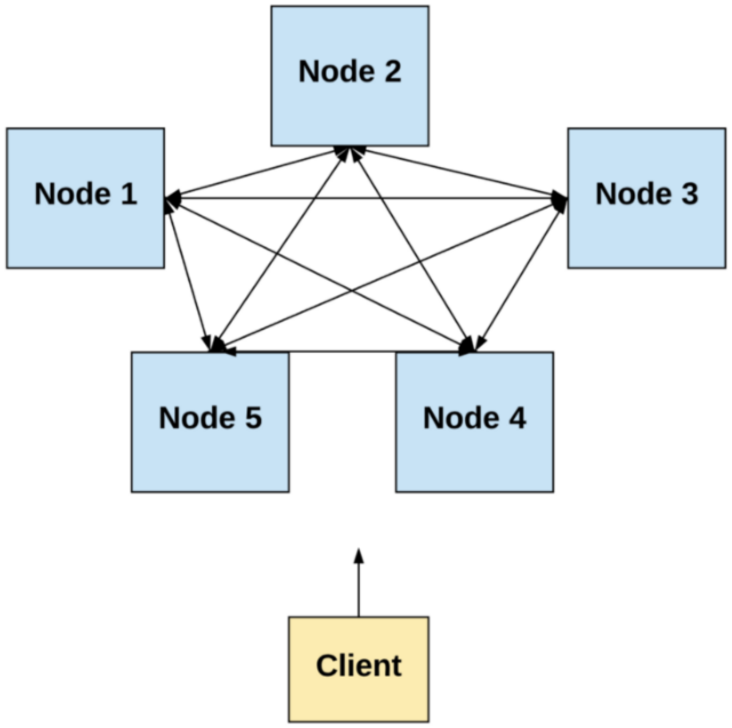
In the case of an airplane, these discrete units work together to get you from Point A to Point B:



Source: https://www.weetech.de/en/news-info/tester-abc/distributed-system-1/

In this post, we'll focus on distributed systems in which processes are spatially-separated computers.

Diagram by author.



Note: I may use the terms "node," "peer," "computer," or "component" interchangeably with "process." They all mean the same thing for the purposes of this post. Similarly, I may use the term "network" interchangeably with "system."

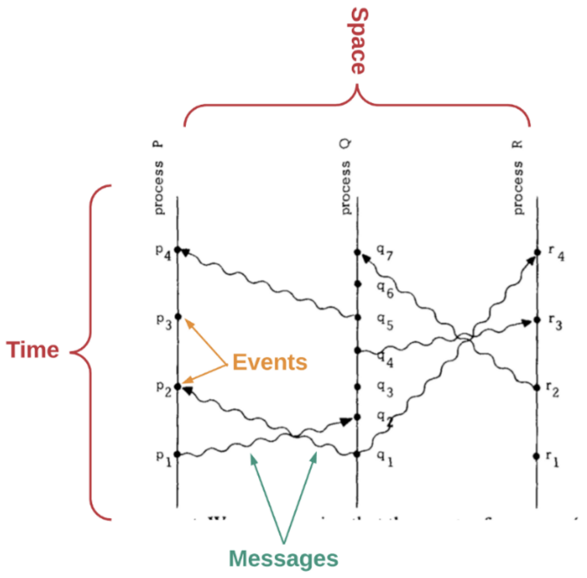
Properties of a Distributed System

Every distributed system has a specific set of characteristics. These include:

A) Concurrency

The processes in the system operate concurrently, meaning multiple events occur simultaneously. In other words, each computer in the network executes events independently at the same time as other computers in the network.

This requires coordination.



Lamport, L (1978). Time, Clocks and Ordering of Events in a Distributed System

B) Lack of a global clock

For a distributed system to work, we need a way to determine the order of events. However, in a set of computers operating concurrently, it is sometimes impossible to say that one of two events occurred first, as computers are spatially separated. In other words, there is no single global clock that determines the sequence of events happening across all computers in the network.

In the paper "Time, Clocks and Ordering of Events in a Distributed System," Leslie Lamport shows how we can deduce whether one event happens before another by remembering the following factors:

1. Messages are sent before they are received.
2. Each computer has a sequence of events.

By determining which event happens before another, we can get a partial ordering of events in the system. Lamport's paper describes an algorithm which requires each computer to hear from every other computer in the system. In this way, events can be totally ordered based on this partial ordering.

However, if we base the order entirely upon events heard by each individual computer, we can run into situations where this order differs from what a user external to the system perceives. Thus, the paper shows that the algorithm can still allow for anomalous behavior.

Finally, Lamport discusses how such anomalies can be prevented by using properly synchronized physical clocks.

But wait—there's a huge caveat: coordinating otherwise independent clocks is a very complex computer science problem. Even if you initially set a bunch of clocks accurately, the clocks will begin to differ after some amount of time. This is due to "clock drift," a phenomenon in which clocks count time at slightly different rates.

Essentially, Lamport's paper demonstrates that time and order of events are fundamental obstacles in a system of distributed computers that are spatially separated.

C) Independent failure of components

A critical aspect of understanding distributed systems is acknowledging that components in a distributed system are faulty. This is why it's called "fault-tolerant distributed computing."

It's impossible to have a system free of faults. Real systems are subject to a number of possible flaws or defects, whether that's a process crashing; messages being lost, distorted, or duplicated; a network partition delaying or dropping messages; or even a process going completely haywire and sending messages according to some malevolent plan.

It's impossible to have a system free of faults.

These failures can be broadly classified into three categories:

- **Crash-fail**: The component stops working without warning (e.g., the computer crashes).
- **Omission**: The component sends a message but it is not received by the other nodes (e.g., the message was dropped).
- **Byzantine**: The component behaves arbitrarily. This type of fault is irrelevant in controlled environments (e.g., Google or Amazon data centers) where there is presumably no malicious behavior. Instead, these faults occur in what's known as an "adversarial context." Basically, when a decentralized set of independent actors serve as nodes in the network, these actors may choose to act in a "Byzantine" manner. This means they maliciously choose to alter, block, or not send messages at all.

With this in mind, the aim is to design protocols that allow a system with faulty components to still achieve the common goal and provide a useful service.

Given that every system has faults, a core consideration we must make when building a distributed system is whether it can survive even when its parts deviate from normal behavior, whether that's due to non-malicious behaviors (i.e., crash-fail or omission faults) or malicious behavior (i.e., Byzantine faults).

Broadly speaking, there are two types of models to consider when making a distributed system:

1) Simple fault-tolerance

In a simple fault-tolerant system, we assume that all parts of the system do one of two things: they either follow the protocol exactly or they fail. This type of system should definitely be able to handle nodes going offline or failing. But it doesn't have to worry about nodes exhibiting arbitrary or malicious behavior.

2A) Byzantine fault-tolerance

A simple fault-tolerant system is not very useful in an uncontrolled environment. In a decentralized system that has nodes controlled by independent actors communicating on the open, permissionless internet, we also need to design for nodes that choose to be malicious or "Byzantine." Therefore, in a Byzantine fault-tolerant system, we assume nodes can fail or be malicious.

2B) BAR fault-tolerance

Despite the fact that most real systems are designed to withstand Byzantine failures, some experts argue that these designs are too general and don't take into account "rational" failures, wherein nodes can deviate if it is in their self-interest to do so. In other words, nodes can be both honest and dishonest, depending on incentives. If the incentives are high enough, then even the majority of nodes might act dishonestly.

More formally, this is defined as the BAR model—one that specifies for both Byzantine and rational failures. The BAR model assumes three types of actors:

- **Byzantine:**Byzantine nodes are malicious and trying to screw you.
- **Altruistic:**Honest nodes always follow the protocol.
- **Rational:**Rational nodes only follow the protocol if it suits them.

D) Message passing

As I noted earlier, computers in a distributed system communicate and coordinate by "message passing" between one or more other computers. Messages can be passed using any messaging protocol, whether that's HTTP, RPC, or a custom protocol built for the specific implementation. There are two types of message-passing environments:

1) Synchronous

In a synchronous system, it is assumed that messages will be delivered within some fixed, known amount of time.

Synchronous message passing is conceptually less complex because users have a guarantee: when they send a message, the receiving component will get it within a certain time frame. This allows users to model their protocol with a fixed upper bound of how long the message will take to reach its destination.

However, this type of environment is not very practical in a real-world distributed system where computers can crash or go offline and messages can be dropped, duplicated, delayed, or received out of order.

2) Asynchronous

In an asynchronous message-passing system, it is assumed that a network may delay messages infinitely, duplicate them, or deliver them out of order. In other words, there is no fixed upper bound on how long a message will take to be received.

What It Means to Have Consensus in a Distributed System

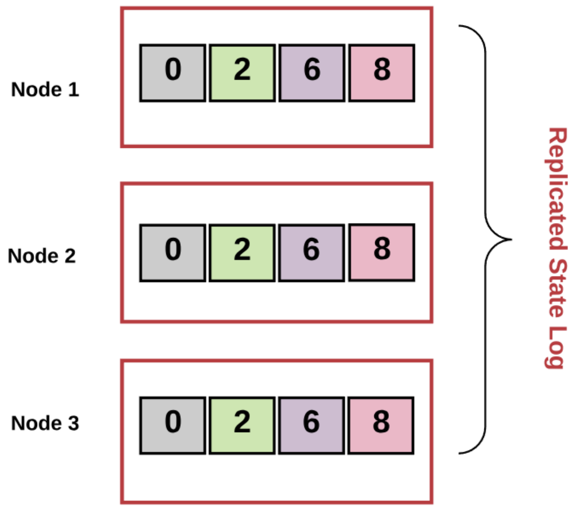
So far, we've learned about the following properties of a distributed system:

- Concurrency of processes
- Lack of a global clock
- Faulty processes
- Message passing

Next, we'll focus on understanding what it means to achieve "consensus" in a distributed system. But first, it's important to reiterate what we alluded to earlier: there are hundreds of hardware and software architectures used for distributed computing.

The most common form is called a replicated state machine.

Replicated State Machine



A replicated state machine is a deterministic state machine that is replicated across many computers but functions as a single state machine. Any of these computers may be faulty, but the state machine will still function.

Left: Diagram by author.

In a replicated state machine, if a transaction is valid, a set of inputs will cause the state of the system to transition to the next state. A transaction is an atomic operation on a database. This means the operations either complete in full or never complete at all. The set of transactions maintained in a replicated state machine is known as a "transaction log."

The logic for transitioning from one valid state to the next is called the "state transition logic."

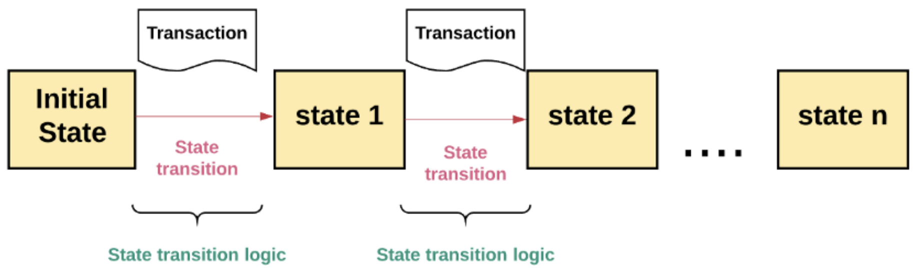


Diagram by author.

In other words, a replicated state machine is a set of distributed computers that all start with the same initial value. For each state transition, each of the processes decides on the next value. Reaching "consensus" means that all the computers must collectively agree on the output of this value.

In turn, this maintains a consistent transaction log across *every* computer in the system (i.e., they "achieve a common goal"). The replicated state machine must continually accept new transactions into this log (i.e., "provide a useful service"). It must do so despite the fact that:

1. Some of the computers are faulty.
2. The network is not reliable and messages may fail to deliver, be delayed, or be out of order.
3. There is no global clock to help determine the order of events.

And this, my friends, is the fundamental goal of any consensus algorithm.

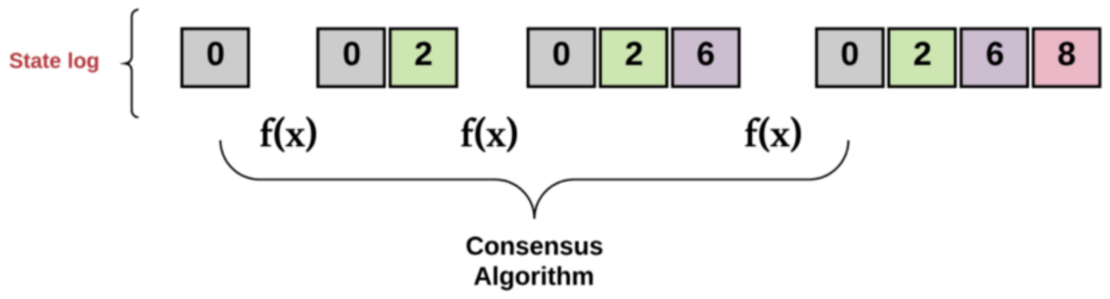


Diagram by author.

The Consensus Problem, Defined

An algorithm achieves consensus if it satisfies the following conditions:

- **Agreement**: All non-faulty nodes decide on the same output value.
- **Termination**: All non-faulty nodes eventually decide on some output value.

*Note: Different algorithms have different variations of the conditions above. For example, some divide the **Agreement** property into **Consistency** and **Totality** . Some have a concept of **Validity** or **Integrity** or **Efficiency** . However, such nuances are beyond the scope of this post.*

Broadly speaking, consensus algorithms typically assume three types of actors in a system:

1. **Proposers**, often called leaders or coordinators.

2. **Acceptors**, processes that listen to requests from proposers and respond with values.
3. **Learners**, other processes in the system which learn the final values that are decided upon.

Generally, we can define a consensus algorithm by three steps:

Step 1: Elect

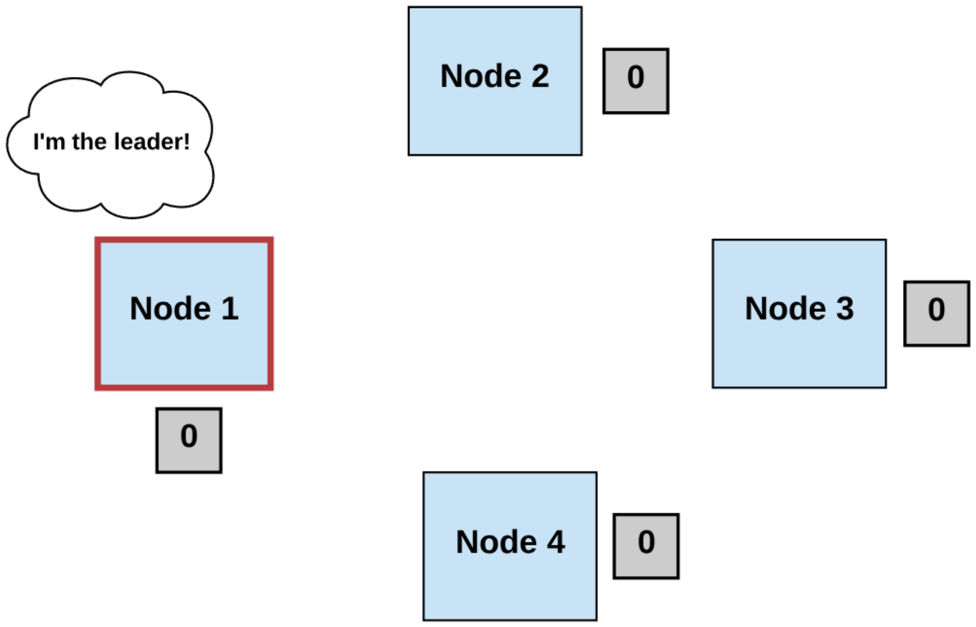
- Processes elect a single process (i.e., a leader) to make decisions.
- The leader proposes the next valid output value.

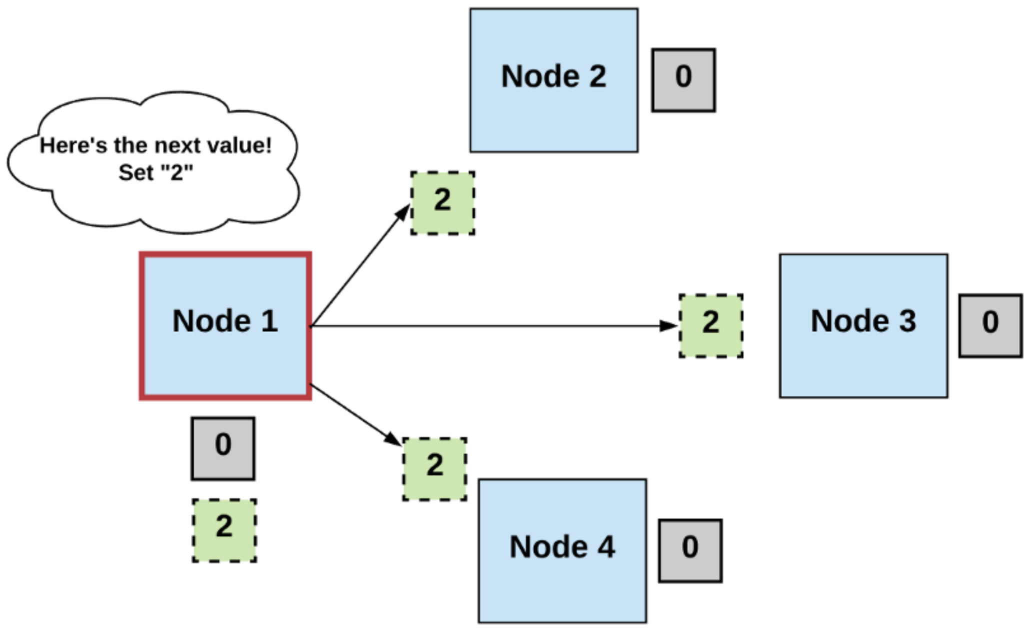
Step 2: Vote

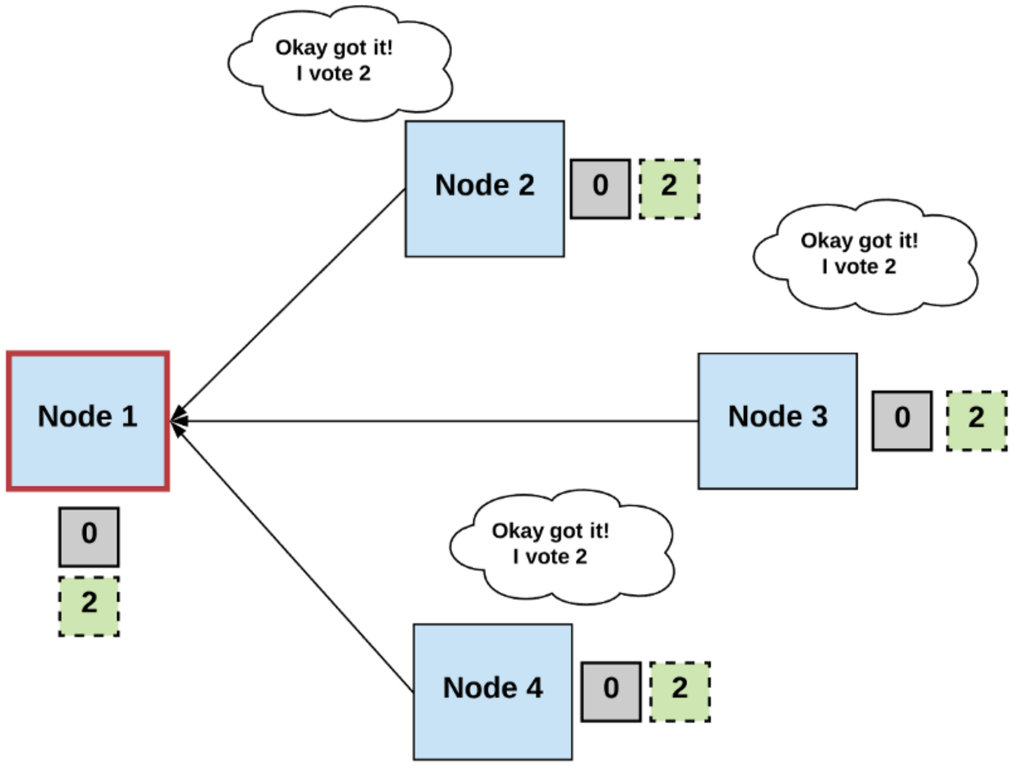
- The non-faulty processes listen to the value being proposed by the leader, validate it, and propose it as the next valid value.

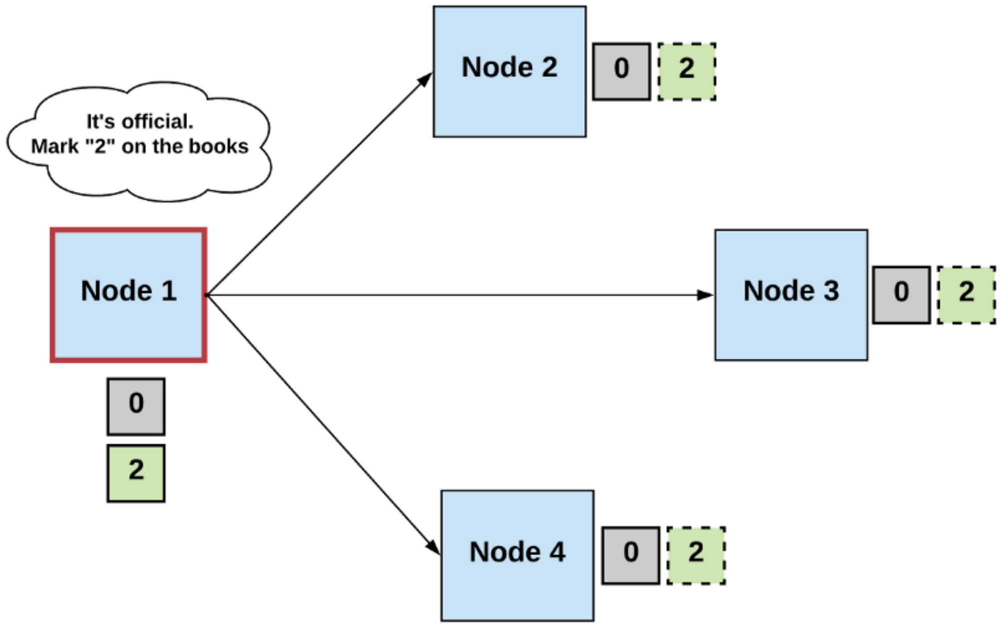
Step 3: Decide

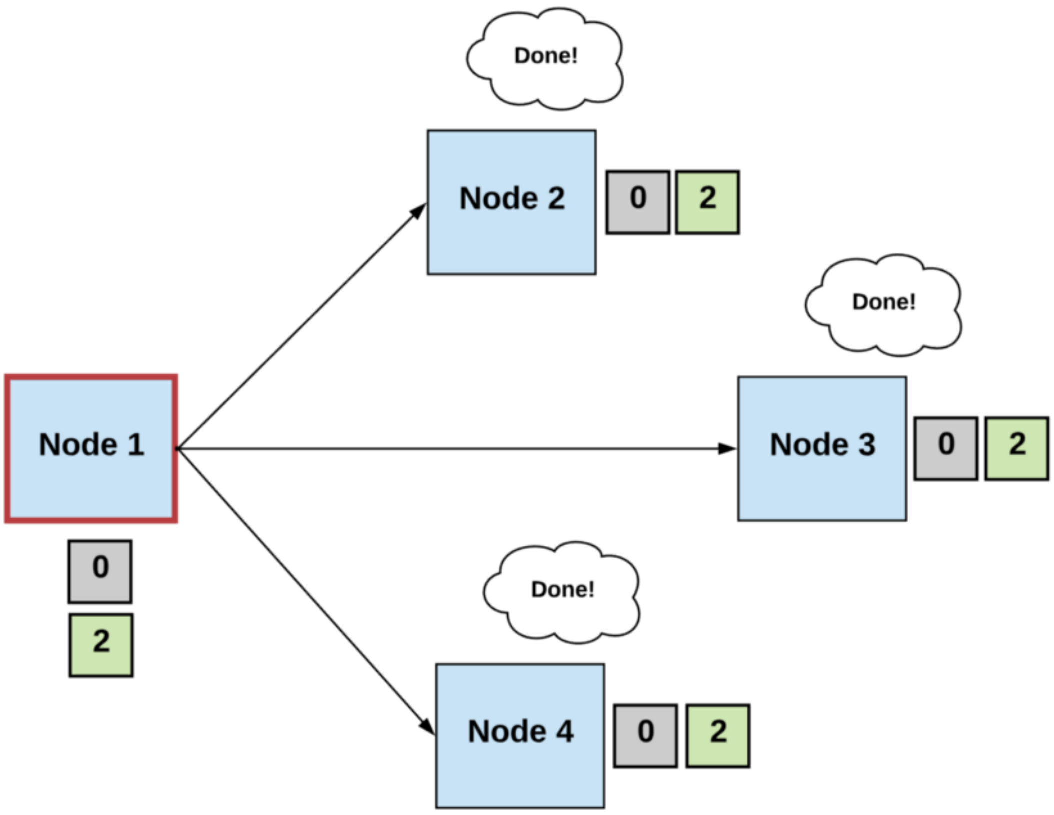
- The non-faulty processes must come to a consensus on a single correct output value. If it receives a threshold number of identical votes which satisfy some criteria, then the processes will decide on that value.
-
- Otherwise, the steps start over.




Here's the next value! Set "2"
Node 1
0
2
2
Node 2
0
2
Node 3
0
2
Node 4
0


Okay got it! I vote 2
Node 2
0
2
Okay got it! I vote 2
Node 1
0
2
Node 3
0
2
Okay got it! I vote 2
Node 4
0
2





Diagrams by author.

It's important to note that every consensus algorithm has different:

- Terminology (e.g., rounds, phases),
- Procedures for how votes are handled, and
- Criteria for how a final value is decided (e.g., validity conditions).

Nonetheless, if we can use this generic process to build an algorithm that guarantees the general conditions defined above, then we have a distributed system which is able to achieve consensus.

Simple enough, right?

FLP impossibility

… Not really. But you probably saw that coming!

Recall how we described the difference between a synchronous system and asynchronous system:

- In synchronous environments, messages are delivered within a fixed time frame
- In asynchronous environments, there's no guarantee of a message being delivered.

This distinction is important.

Reaching consensus in a synchronous environment is possible because we can make assumptions about the maximum time it takes for messages to get delivered. Thus, in this type of system, we can allow the different nodes in the system to take turns proposing new transactions, poll for a majority vote, and skip any node if it doesn't offer a proposal within the maximum time limit.

But, as noted earlier, assuming we are operating in synchronous environments is not practical outside of controlled environments where message latency is predictable, such as data centers which have synchronized atomic clocks.

In reality, most environments don't allow us to make the synchronous assumption. So we must design for asynchronous environments.

If we cannot assume a maximum message delivery time in an asynchronous environment, then achieving termination is much harder, if not impossible. Remember, one of the conditions that must be met to achieve consensus is "termination," which means every non-faulty node *must decide* on some output value.

This is formally known as the "FLP impossibility result." How did it get this name? Well, I'm glad you asked!

Even a single faulty process makes it impossible to reach consensus among deterministic asynchronous processes.

In their 1985 paper "Impossibility of Distributed Consensus with One Faulty Process," researchers Fischer, Lynch, and Paterson (aka FLP) show how even a single faulty process makes it impossible to reach consensus among deterministic asynchronous processes. Basically, because processes can fail at unpredictable times, it's also possible for them to fail at the exact opportune time that prevents consensus from occurring.

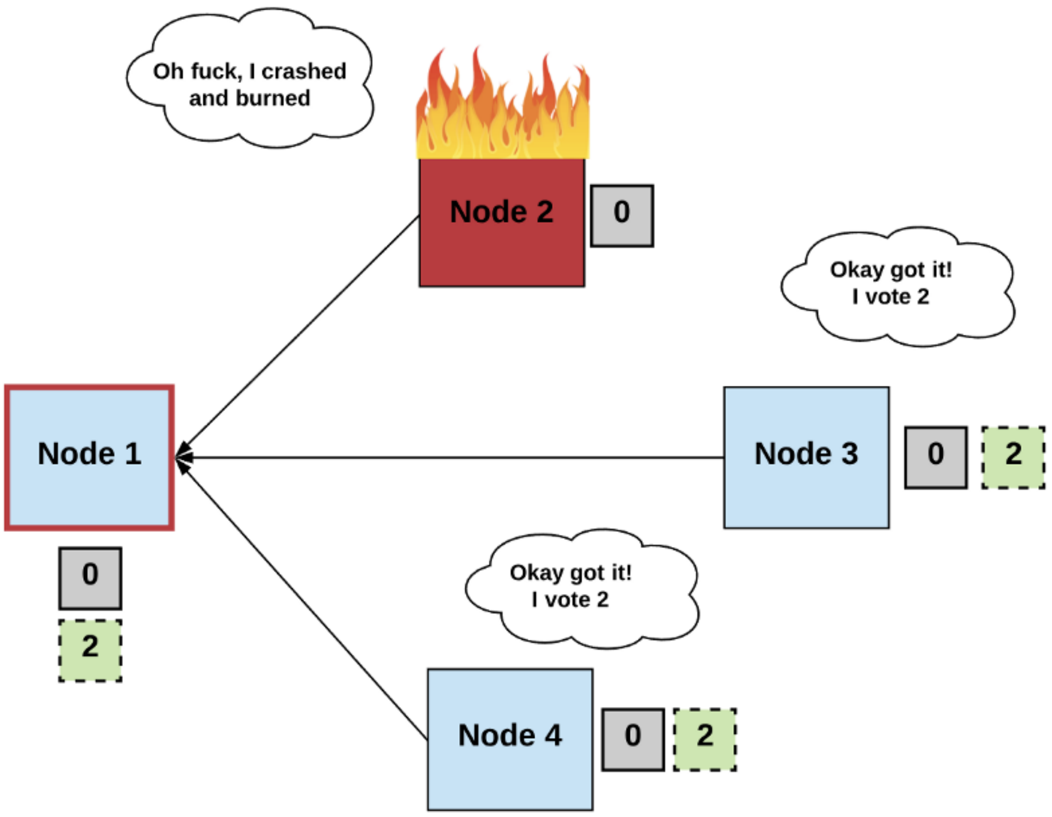


Diagram by author.

This result was a huge bummer for the distributed computing space. Nonetheless, scientists continued to push forward to find ways to circumvent FLP impossibility.

At a high level, there are two ways to circumvent FLP impossibility:

1. Use synchrony assumptions.
2. Use non-determinism.

Let's take a deep dive into each one right now.

Approach 1: Use Synchrony Assumptions

I know what you're thinking: What the heck does this even mean?

Let's revisit our impossibility result. Here's another way to think about it: the FLP impossibility result essentially shows that, if we cannot make progress in a system, then we cannot reach consensus. In other words, if messages are asynchronously delivered, termination cannot be guaranteed. Recall that termination is a required condition that means every non-faulty node must eventually decide on some output value.

But how can we guarantee every non-faulty process will decide on a value if we don't know when a message will be delivered due to asynchronous networks?

To be clear, the finding does not state that consensus is unreachable. Rather, due to asynchrony, consensus cannot be reached in a fixed time. Saying that consensus is "impossible" simply means that consensus is "not always possible." It's a subtle but crucial detail.

One way to circumvent this is to use timeouts. If no progress is being made on deciding the next value, we wait until a timeout, then start the steps all over again. As we're about to see, this is what consensus algorithms like Paxos and Raft essentially did.

Paxos

Introduced in the 1990s, Paxos was the first real-world, practical, fault-tolerant consensus algorithm. It's one of the first widely adopted consensus algorithms to be proven correct by Leslie Lamport and has been used by global internet companies like Google and Amazon to build distributed services.

Paxos works like this:

Phase 1: Prepare request

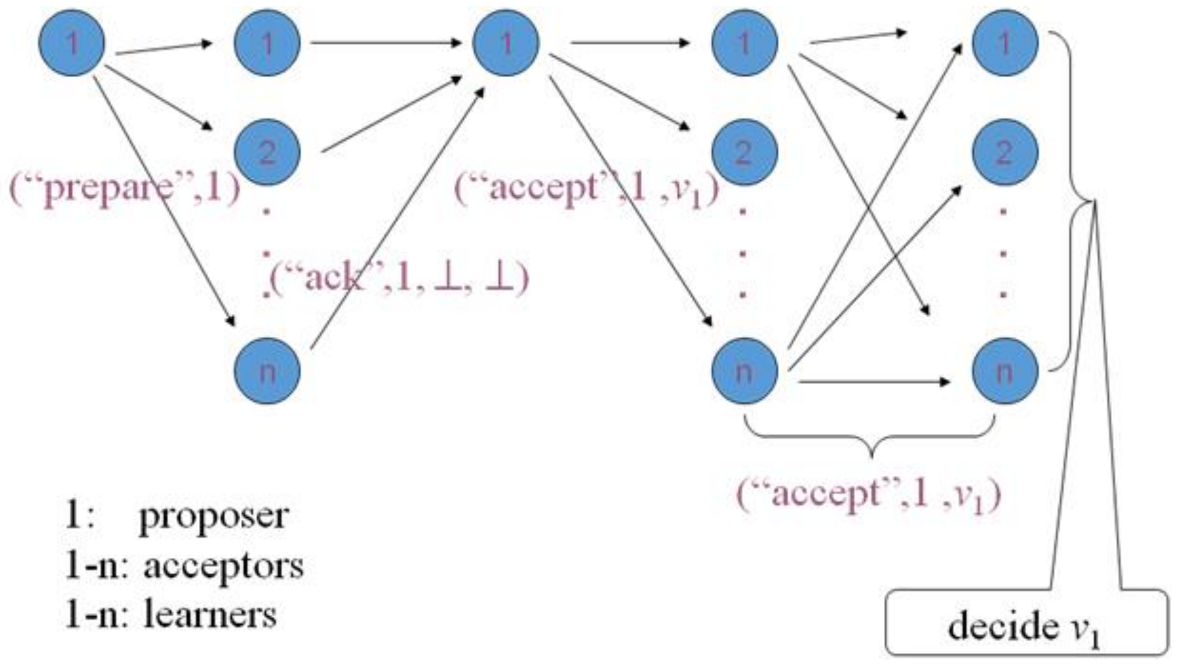
1. The proposer chooses a new proposal version number (n) and sends a "prepare request" to the acceptors.
2. If acceptors receive a prepare request ("prepare," n) with n greater than that of any prepare request they had already responded to, the acceptors send out ("ack," n, n', v') or ("ack," n, ^ , ^).
3. Acceptors respond with a promise not to accept any more proposals numbered less than n.
4. Acceptors suggest the value (v) of the highest-number proposal that they have accepted, if any. Or else, they respond with ^.

Phase 2: Accept request

1. If the proposer receives responses from a majority of the acceptors, then it can issue an accept request ("accept," n, v) with number n and value v.
2. n is the number that appeared in the prepare request.
3. v is the value of the highest-numbered proposal among the responses.
4. If the acceptor receives an accept request ("accept," n, v), it accepts the proposal unless it has already responded to a prepare request with a number greater than n.

Phase 3: Learning phase

1. Whenever an acceptor accepts a proposal, it responds to all learners ("accept," n, v).
2. Learners receive ("accept," n, v) from a majority of acceptors, decide v, and send ("decide," v) to all other learners.
3. Learners receive ("decide," v) and the decided v.



Source: *https://www.myassignmenthelp.net/paxos-algorithm-assignment-help*

Phew! Confused yet? I know that was a quite a lot of information to digest.

But wait… There's more!

As we now know, every distributed system has faults. In this algorithm, if a proposer failed (e.g., because there was an omission fault), then decisions could be delayed. Paxos dealt with this by starting with a new version number in Phase 1, even if previous attempts never ended.

I won't go into details, but the process to get back to normal operations in such cases was quite complex since processes were expected to step in and drive the resolution process forward.

The main reason Paxos is so hard to understand is that many of its implementation details are left open to the reader's interpretation: How do we know when a proposer is failing? Do we use synchronous clocks to set a timeout period for deciding when a proposer is failing and we need to move on to the next rank? 🤷‍♀️

In favor of offering flexibility in implementation, several specifications in key areas are left open-ended. Things like leader election, failure detection, and log management are vaguely or completely undefined.

This design choice ended up becoming one of the biggest downsides of Paxos. It's not only incredibly difficult to understand but difficult to implement as well. In turn, this made the field of distributed systems incredibly hard to navigate.

By now, you're probably wondering where the synchrony assumption comes in.

In Paxos, although timeouts are not explicit in the algorithm, when it comes to the actual implementation, electing a new proposer after some timeout period is necessary to achieve termination. Otherwise, we couldn't guarantee that acceptors would output the next value, and the system could come to a halt.

Raft

In 2013, Ongaro and Ousterhout published a new consensus algorithm for a replicated state machine called Raft, where the core goal was understandability (unlike Paxos).

One important new thing we learned from Raft is the concept of using a shared timeout to deal with termination. In Raft, if you crash and restart, you wait at least one timeout period before trying to get yourself declared a leader, and you are guaranteed to make progress.

But Wait… What about 'Byzantine' Environments?

While traditional consensus algorithms (such as Paxos and Raft) are able to thrive in asynchronous environments using some level of synchrony assumptions (i.e. timeouts), they are not Byzantine fault-tolerant. They are only crash fault-tolerant.

Crash-faults are easier to handle because we can model the process as either working or crashed—0 or 1. The processes can't act maliciously and lie. Therefore, in a crash fault-tolerant system, a distributed system can be built where a simple majority is enough to reach a consensus.

In an open and decentralized system (such as public blockchains), users have no control over the nodes in the network. Instead, each node makes decisions toward its individual goals, which may conflict with those of other nodes.

In a Byzantine system where nodes have different incentives and can lie, coordinate, or act arbitrarily, you cannot assume a simple majority is enough to reach consensus. Half or more of the supposedly honest nodes can coordinate with each other to lie.

For example, if an elected leader is Byzantine and maintains strong network connections to other nodes, it can compromise the system. Recall how we said we must model our system to either tolerate simple faults or Byzantine faults. Raft and Paxos are simple fault-tolerant but not Byzantine fault-tolerant. They are not designed to tolerate malicious behavior.

The 'Byzantine General's Problem'

Trying to build a reliable computer system that can handle processes that provide conflicting information is formally known as the "Byzantine General's Problem." A Byzantine fault-tolerant protocol should be able to achieve its common goal even against malicious behavior from nodes.

The paper " *Byzantine General's Problem* " by Leslie Lamport, Robert Shostak, and Marshall Pease provided the first proof to solve the Byzantine General's problem: it showed that a system with **x** Byzantine nodes must have at least **3x + 1 total nodes** in order to reach consensus.

Here's why:

If **x** nodes are faulty, then the system needs to operate correctly after coordinating with **n minus x** nodes (since x nodes might be faulty/Byzantine and not responding). However, we must prepare for the possibility that the **x** that doesn't respond may not be faulty; it could be the **x** that *does* respond. If we want the number of non-faulty nodes to outnumber the number of faulty nodes, we need at least **n minus x minus x > x**. Hence, **n > 3x + 1** is optimal.

However, the algorithms demonstrated in this paper are only designed to work in a synchronous environment. Bummer! It seems we can only get one or the other (Byzantine or Asynchronous) right. An environment that is both Byzantine and Asynchronous seems much harder to design for.

Why?

In short, building a consensus algorithm that can withstand both an asynchronous environment *and* a Byzantine one is… well, that would sort of be like making a miracle happen.

Algorithms like Paxos and Raft were well-known and widely used. But there was also a lot of academic work that focused more on solving the consensus problem in a Byzantine + asynchronous setting.

So buckle your seatbelts…

We're going on a field trip…

To the land of…

Theoretical academic papers!

Okay, okay—I'm sorry for building that up. But you should be excited! Remember that whole "making a miracle" thing we discussed earlier? We're going to take a look at two algorithms (DLS and PBFT) that brought us closer than ever before to breaking the Byzantine + asynchronous barrier.

The DLS Algorithm

The paper "Consensus in the Presence of Partial Synchrony" by Dwork, Lynch, and Stockmeyer (hence the name "DLS" algorithm) introduced a major advancement in Byzantine fault-tolerant consensus: it defined models for how to achieve consensus in "a partially synchronous system."

As you may recall, in a synchronous system, there is a known fixed upper bound on the time required for a message to be sent from one processor to another. In an asynchronous system, no fixed upper bounds exist. Partial synchrony lies somewhere between those two extremes.

The paper explained two versions of the partial synchrony assumption:

- Assume that fixed bounds exist for how long messages take to get delivered. But they are not known a priori. The goal is to reach consensus regardless of the actual bounds.
- Assume the upper bounds for message delivery are known, but they're only guaranteed to hold starting at some unknown time (also called "Global Standardization Time," GST). The goal is to design a system that can reach consensus regardless of when this time occurs.

Here's how the DLS algorithm works:

A series of rounds are divided into "trying" and "lock-release" phases.

1. Each round has a proposer and begins with each of the processes communicating the value they believe is correct.
2. The proposer "proposes" a value if at least **N − x**processes have communicated that value.

3. When a process receives the proposed value from the proposer, it must lock on the value and then broadcast that information.
4. If the proposer receives messages from **x + 1** processes that they locked on some value, it commits that as the final value.

DLS was a major breakthrough because it created a new category of network assumptions to be made—namely, partial synchrony—and proved consensus was possible with this assumption. The other imperative takeaway from the DLS paper was separating the concerns for reaching consensus in a Byzantine and asynchronous setting into two buckets: safety and liveness.

Safety

This is another term for the “agreement” property we discussed earlier, where all non-faulty processes agree on the same output. If we can guarantee safety, we can guarantee that the system as a whole will stay in sync. We want all nodes to agree on the total order of the transaction log, despite failures and malicious actors. A violation of safety means that we end up with two or more valid transaction logs.

Liveness

This is another term for the “termination” property we discussed earlier, where every non-faulty node eventually decides on some output value. In a blockchain setting, “liveness” means the blockchain keeps growing by adding valid new blocks. Liveness is important because it’s the only way that the network can continue to be useful—otherwise, it will stall.

As we know from the FLP impossibility, consensus can’t be achieved in a completely asynchronous system. The DLS paper argued that making a partial synchrony assumption for achieving the liveness condition is enough to overcome FLP impossibility.

Thus, the paper proved that the *algorithms don’t need to use any synchrony assumption to achieve the safety condition*.

Pretty straightforward, right? Don’t worry if it’s not. Let’s dig a little deeper.

Remember that if nodes aren’t deciding on some output value, the system just halts. So, if we make some synchrony assumptions (i.e., timeouts) to guarantee termination and one of those fails, it makes sense that this would also bring the system to a stop.

But if we design an algorithm where we assume timeouts (to guarantee correctness), this carries the risk of leading to two valid transaction logs if the synchrony assumption fails.

Designing a distributed system is always about trade-offs.

This would be far more dangerous than the former option. There's no point in having a useful service (i.e., liveness) if the service is corrupt (i.e., no safety). Basically, having two different blockchains is worse than having the entire blockchain come to a halt.

A distributed system is always about trade-offs. If you want to overcome a limitation (e.g., FLP impossibility), you must make a sacrifice somewhere else. In this case, separating the concerns into safety and liveness is brilliant. It lets us build a system that is safe in an asynchronous setting but still needs some form of timeouts to keep producing new values.

Despite everything that the DLS paper offered, DLS was never widely implemented or used in a real-world Byzantine setting. This is probably due to the fact that one of the core assumptions in the DLS algorithm was to use synchronous processor clocks in order to have a common notion of time. In reality, synchronous clocks are vulnerable to a number of attacks and wouldn't fare well in a Byzantine fault-tolerant setting.

The PBFT Algorithm

Another Byzantine fault-tolerant algorithm, published in 1999 by Miguel Castro and Barbara Liskov, was called "Practical Byzantine Fault-Tolerance" (PBFT). It was deemed to be a more "practical" algorithm for systems that exhibit Byzantine behavior.

"Practical" in this sense meant it worked in asynchronous environments like the internet and had some optimizations that made it faster than previous consensus algorithms. The paper argued that previous algorithms, while shown to be "theoretically possible," were either too slow to be used or assumed synchrony for safety.

And as we've explained, that can be quite dangerous in an asynchronous setting.

In a nutshell, the PBFT algorithm showed that it could provide safety and liveness assuming **(n-1)/3**nodes were faulty. As we previously discussed, that's the minimum number of nodes we need to tolerate Byzantine faults. Therefore, the resiliency of the algorithm was optimal.

The algorithm provided safety regardless of how many nodes were faulty. In other words, it didn't assume synchrony for safety. The algorithm did, however, rely on synchrony for liveness. At most, **(n-1)/3** nodes could be faulty and the message delay did not grow faster than a certain time limit. Hence, PBFT circumvented FLP impossibility by using a synchrony assumption to guarantee liveness.

The algorithm moved through a succession of "views," where each view had one "primary" node (i.e., a leader) and the rest were "backups." Here's a step-by-step walkthrough of how it worked:

1. A new transaction happened on a client and was broadcast to the primary.
2. The primary multicasted it to all the backups.
3. The backups executed the transaction and sent a reply to the client.
4. The client wanted **x + 1** replies from backups with the same result. This was the final result, and the state transition happened.

If the leader was non-faulty, the protocol worked just fine. However, the process for detecting a bad primary and reelecting a new primary (known as a “view change”) was grossly inefficient. For instance, in order to reach consensus, PBFT required a quadratic number of message exchanges, meaning every computer had to communicate with every other computer in the network.

Note: Explaining the PBFT algorithm in full is a blog post all on its own! We’ll save that for another day ;).

While PBFT was an improvement over previous algorithms, it wasn’t practical enough to scale for real-world use cases (such as public blockchains) where there are large numbers of participants. But hey, at least it was much more specific when it came to things like failure detection and leader election (unlike Paxos).

It’s important to acknowledge PBFT for its contributions. It incorporated important revolutionary ideas that newer consensus protocols (especially in a post-blockchain world) would learn from and use.

For example, Tendermint is a new consensus algorithm that is heavily influenced by PBFT. In their “validation” phase, Tendermint uses two voting steps (like PBFT) to decide on the final value. The key difference with Tendermint’s algorithm is that it’s designed to be more practical.

For instance, Tendermint rotates a new leader every round. If the current round’s leader doesn’t respond within a set period of time, the leader is skipped and the algorithm simply moves to the next round with a new leader. This actually makes a lot more sense than using point-to-point connections every time there needs to be a view-change and a new leader elected.

Approach 2: Non-Determinism

As we’ve learned, most Byzantine fault-tolerant consensus protocols end up using some form of synchrony assumption to overcome FLP impossibility. However, there is another way to overcome FLP impossibility: non-determinism.

Enter: Nakamoto Consensus

As we just learned, in traditional consensus, **f(x)** is defined such that a proposer and a bunch of acceptors must all coordinate and communicate to decide on the next value.

Traditional consensus doesn't scale well.

This is too complex because it requires knowing every node in the network *and* every node communicating with every other node (i.e., quadratic communication overhead). Simply put, it doesn't scale well and doesn't work in open, permissionless systems where anyone can join and leave the network at any time.

The brilliance of Nakamoto Consensus is making the above probabilistic. Instead of every node agreeing on a value, f(x) works such that all of the nodes agree on the *probability* of the value being correct.

Wait, what does that even mean?

Byzantine-fault tolerant

Rather than electing a leader and then coordinating with all nodes, consensus is decided based on which node can solve the computation puzzle the fastest. Each new block in the Bitcoin blockchain is added by the node that solves this puzzle the fastest. The network continues to build on this timestamped chain, and the canonical chain is the one with the most cumulative computation effort expended (i.e., cumulative difficulty).

The longest chain not only serves as proof of the sequence of blocks, but proof that it came from the largest pool of CPU power. Therefore, as long as a majority of CPU power is controlled by honest nodes, they'll continue to generate the longest chain and outpace attackers.

Block rewards

Nakamoto Consensus works by assuming that nodes will expend computational efforts for the chance of deciding the next block. The brilliance of the algorithm is economically incentivizing nodes to repeatedly perform such computationally expensive puzzles for the chance of randomly winning a large reward (i.e., a block reward).

Sybil resistance

The proof of work required to solve this puzzle makes the protocol inherently Sybil-resistant. No need for PKI or any other fancy authentication schemes.

Peer-to-peer gossip

A major contribution of Nakamoto Consensus is the use of the gossip protocol. It's more suitable for peer-to-peer settings where communication between non-faulty nodes can't be assumed. Instead, we assume a node is only connected to a subset of other nodes. Then we use the peer-to-peer protocol where messages are being gossiped between nodes.

Not "technically" safe in asynchronous environments

In Nakamoto consensus, the safety guarantee is *probabilistic* — we're growing the longest chain, and each new block lowers the probability of a malicious node trying to build another valid chain.

Therefore, Nakamoto Consensus does not technically guarantee safety in an asynchronous setting. Let's take a second to understand why.

For a system to achieve the safety condition in an asynchronous setting, we should be able to maintain a consistent transaction log despite asynchronous network conditions. Another way to think about it is that a node can go offline at any time and then later come back online, and use the initial state of the blockchain to determine the latest correct state, regardless of network conditions. Any of the honest nodes can query for arbitrary states in the past, and a malicious node cannot provide fraudulent information that the honest nodes will think is truthful.

Nakamoto Consensus does not technically guarantee safety in an asynchronous setting.

In previous algorithms discussed in this post, this is possible because we are deterministically finalizing a value at every step. As long as we terminate on each value, we can know the past state. However, the reason Bitcoin is not "technically" asynchronously safe is that it is possible for there to be a network partition that lets an attacker with a sufficiently large hash power on one side of the partition to create an alternative chain faster than the honest chain on the other side of the partition — and on this alternative chain, he can try to change one of his own transactions to take back money he spent.

Admittedly, this would require the attacker to gain a lot of hashing power and spend a lot of money.

In essence, the Bitcoin blockchain's immutability stems from the fact that a majority of miners don't actually want to adopt an alternative chain. Why? Because it's too difficult to coordinate enough hash power to get the network to adopt an alternative chain. Put another way, the probability of successfully creating an alternative chain is so low, it's practically negligible.

Nakamoto vs. Traditional Consensus

For practical purposes, Nakamoto Consensus is Byzantine fault-tolerant. But it clearly doesn't achieve consensus the way consensus researchers would traditionally assume. Therefore, it was initially seen as being completely out of the Byzantine fault-tolerant world.

By design, the Nakamoto Consensus makes it possible for any number of nodes to have open participation, and no one has to know the full set of participants. The importance of this breakthrough can't be overstated. Thank you, Nakamoto.

Simpler than previous consensus algorithms, it eliminates the complexity of point-to-point connections, leader election, quadratic communication overhead, etc. You just launch the Bitcoin protocol software on any computer and start mining.

This makes it easily deployable in a real-world setting. It is truly the more "practical" cousin of PBFT.

Conclusion

And there you have it—the brief basics of distributed systems and consensus. It has been a long, winding journey of research, roadblocks, and ingenuity for distributed computing to get this far. I hope this post helps you understand the field at least a tiny it better.

Nakamoto Consensus is truly an innovation that has allowed a whole new wave of researchers, scientists, developers, and engineers to continue breaking new ground in consensus protocol research.

And there's an entirely new family of protocols being developed that go beyond Nakamoto Consensus. Proof-of-Steak, anyone? ;) But I'll save that for the next post—stay tuned!

Note: For the purposes of not turning this into a book, I skipped MANY important papers and algorithms. For example, Ben Orr's Common Coin also used probabilistic approaches but was not optimally resilient. Other algorithms like Hash Cash also used PoW but for limiting email spam and denial-of-service attacks. And there are so many more traditional consensus protocols that I left out! I feel the above is good enough to help you get a great grasp on consensus in a traditional setting vs. Nakamoto. See ya in the next post!

Special thanks to Zaki Manian for putting up with all my questions on distributed consensus.

Exploring Liquid Sidechain

Joe Kendzicky

Posted November 15, 2018

Overview

In 2014, a group of notable Bitcoin developers published *Enabling Blockchain Innovations with Pegged Sidechains*. This whitepaper outlined a very high level framework on how sidechains might look, but lacked significant technical specifications with regard to implementation. Two years later, a group of Blockstream researchers released *Strong Federations: An Interoperable Blockchain Solution to Centralized Third-Party Risks*, describing how many of the aforementioned concepts could be implemented utilizing a semi-trusted Byzantine fault tolerant consensus mechanism referred to as **strong federations**. Strong federations fall somewhere between the extensive decentralization guarantees provided through Nakamoto style consensus, and the significant trust implications of centralized exchange services. In this article, we explore one of the first iterations of federated two-way pegs: *Liquid*, a settlement network primarily directed towards exchanges and other liquidity providers.

What do sidechains provide?

Sidechains allow us to take an asset that traditionally exists on a remote chain, and transfer it onto a different blockchain, without distorting the asset's integrity. For example, imagine there were a way to move bitcoin onto the Ethereum network. Though it is impossible to actually move a BTC onto another chain, we could create a synthetic ETH token collateralized by real bitcoin on the mainchain. Anyone holding the ETH token could exchange it at any moment for real BTC, and thus the two assets would exhibit a 1:1 (or near 1:1) peg.

Having this capability opens the door to many different form of unique cross-chain interoperability features, such as executing smart contracts across different blockchains. Sidechains always involve a *source* and *destination* blockchain. A peg is established between the chains, meaning assets from the source are converted into entirely new ones on the destination, at a fixed ratio (usually 1:1). Since the destination sidechain is its own independent blockchain, destination tokens can be transferred freely among network users, just as they would inside of the source network. Users are free to reconvert their sidechain tokens back into mainchain ones at their full discretion. Usually these destination tokens provide beneficial features that the source chain cannot: privacy, Turing-completeness, fast settlement or low fees. In our scenario, Bitcoin serves as the source blockchain, with Liquid as the destination.

Functionaries

To begin, let’s first start off by defining two important figures within the Liquid ecosystem:

1) **Watchmen**- play an important role serving as gatekeepers, regulating the transition of assets between the source and destination blockchains.

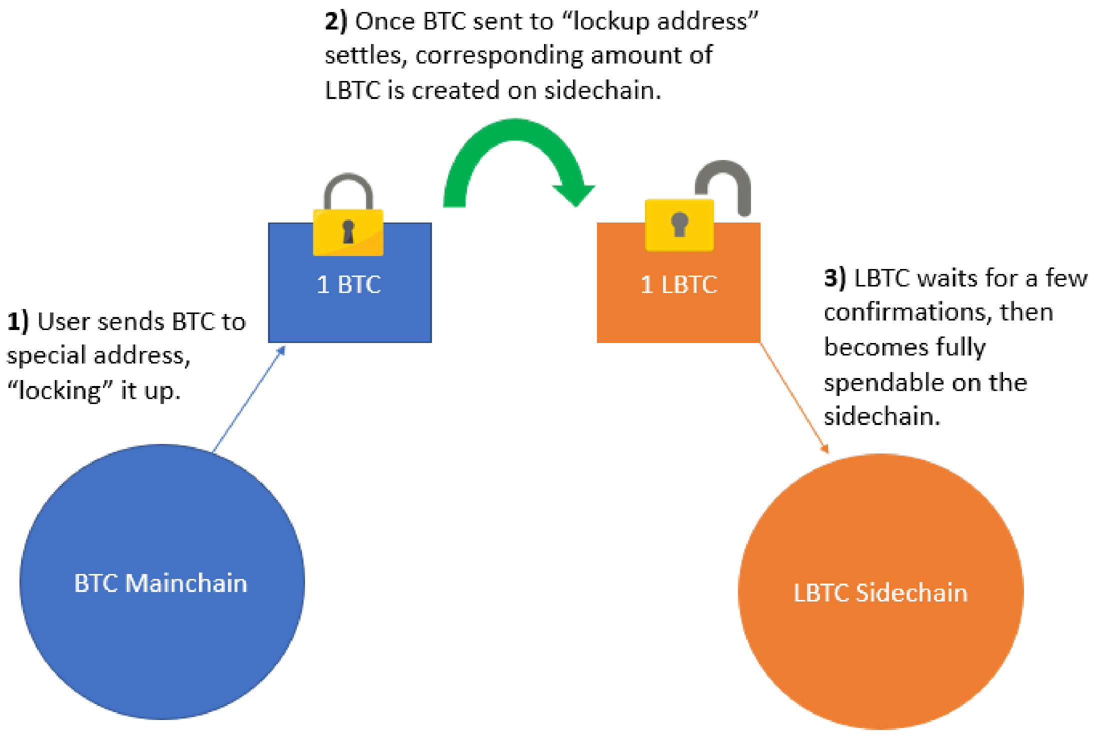
2) **Blocksigners**- contribute to the creation of new blocks, similar to miners in proof-of-work systems and validators in proof-of-stake systems

Both of the above parties fall under the collective roof of *functionaries*. Functionaries are a privileged set of network actors who mechanically perform defined network operations. By privileged, we are referring to a fixed group of permissioned actors who have the capacity to fulfill these roles. Contrast this to open system like Bitcoin, where any network participant has the ability to contribute to the consensus process through open mining operations.

Entry into the sidechain is a process referred to as a “peg-in”, where Bitcoin users deposit BTC to a specially constructed address, called a pay-to-script-hash (P2SH). Traditional Bitcoin scripts usually just require the spender to produce a valid signature from a private key. This private key corresponds to the public key in which the funds are encumbered. P2SH requires the spender to produce a few more pieces of information outside of just a digital signature. The scripts required to spend the LBTC reads: *<11 of 15 Multisig> OR <4-week timeout + 2-of-3 Multisig>*. We will touch on this construct and its importance in later segments.

Once bitcoins are sent to the P2SH address, the depositing party loses sovereignty over them. A confirmation period ensues to guarantee settlement of the funds, preventing against the risk of a chain reorganization or double spend attack. Without sufficient confirmation that the pegged-in BTC has settled, an attacker might be granted LBTC prematurely on the Liquid sidechain, while their BTC manages to “escape” on the mainchain through a reorg or doublespend. Such an event would lead to a situation where LBTC and BTC no longer maintain a 1:1 pegged ratio, thereby granting the attacker free money.

Liquid requires 102 BTC confirmations for the transaction in order for the LBTC to become available. This seemingly ambiguous number exists for 2 reasons:


2) Once BTC sent to "lockup address" settles, corresponding amount of LBTC is created on sidechain.
1 BTC
1 LBTC
1) User sends BTC to special address, "locking" it up.
3) LBTC waits for a few confirmations, then becomes fully spendable on the sidechain.
BTC Mainchain
LBTC Sidechain

1. **100 blocks** as overkill to guarantee that a reorganization is probabilistically impractical. This is the same depth that the Bitcoin protocol requires miners to wait before being able to spend their coinbase rewards when they solve a new block. While 6 confirmations is usually considered sufficiently strong for a BTC transaction, there have been a few rare occasions of orphaned blocks extending much deeper (though this usually only happens during some obscure bug interval). For example, the March 12, 2013 fork extended 31 blocks, while the 2010 value overflow bug extended an orphan 53 blocks.
2. **2 blocks** to ensure every node agrees on the 100th block. Given our first principle, conflicting views could exist amongst the network nodes as to which block that *100th* block is, were a chain split to happen after the 99th mined block. By extending depth an additional 2 blocks, we allow the blockchain some leeway time to re-converge if necessary.

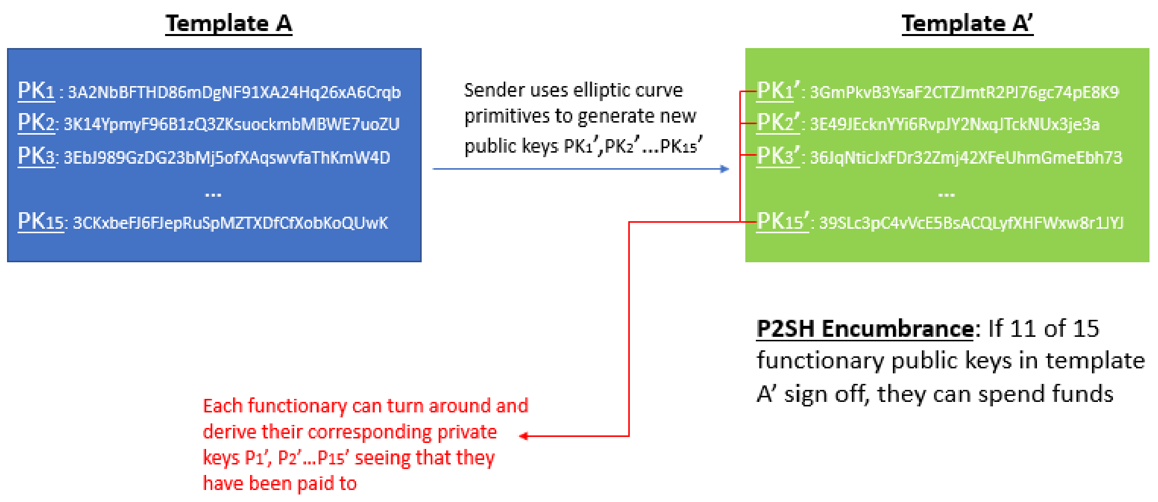
It is important to understand that the BTC on the Bitcoin chain is not destroyed. Rather they remain "frozen" in the multisignature account until the corresponding LBTC are revoked off the destination chain, and proof has been illustrated that such action has taken place.

Examining LBTC Creation

BIP 32 introduced a set of important features that allows for hierarchical wallet structures. Hierarchical frameworks are useful because they allow for deterministic construction of new public keys (and therefore unlimited new BTC addresses) without needing underlying exposure to the public key's corresponding private key. This is a great feature for online merchants wanting to incorporate Bitcoin payments at point of sale. They can upload something called the master private key into their web server, and compute sub-domains of new bitcoin addresses directly from this master public key. Previously, they would have had to either expose their private key onto the server in order to derive new addresses (dangerous) or generate a bunch of new private/pub key pairs offline, then manually import the addresses over to the server (inefficient).

We can think about the construction of new LBTC in a somewhat similar fashion, though some technical differences remain. Essentially what the user is doing during a Liquid peg-in transaction is performing some cryptographic tricks to derive a new public key (call this *A*') directly from each functionary's existing public key (call this *A*), in a similar way that BIP 32 allows one to derive a child public key from a parent public key without needing involvement from the corresponding private keys. The user then sends their BTC to a special P2SH address that references all these new *A'* addresses. This P2SH transaction reads very analogously to a multisignature clause. It says "if 11 functionaries (out of the 15 possible) prove ownership of the *A'* public keys outlined in this template, by collectively signing off with their corresponding *a'* private keys, they can unlock and spend the funds."

After issuing the Bitcoin P2SH transaction, the user needs to actually let the functionaries know that the transaction occurred. Up to this point, the functionaries still have no idea that they've been paid; everything has been done on the user end. The user now needs to turns around and hand this new template of *A'* to the functionaries, so that they can reverse engineer the corresponding a' and gain access to the locked up BTC.


Template A
PK1 : 3A2NbBFTHD86mDgNF91XA24Hq26xA6Crqb
PK2: 3K14YpmyF96B1zQ3ZKsuockmbMBWE7uoZU
PK3: 3EbJ989GzDG23bMj5ofXAqswvfaThKmW4D
...
PK15: 3CKxbeFJ6FJepRuSpMZTXDfCfXobKoQUwK
Sender uses elliptic curve primitives to generate new public keys PK1',PK2'...PK15'
Template A'
PK1': 3GmPkvB3YsaF2CTZJmtR2PJ76gc74pE8K9
PK2': 3E49JEcknYYi6RvpJY2NxqJTckNUx3je3a
PK3': 36JqNticJxFDr32Zmj42XFeUhmGmeEbh73
...
PK15': 39SLc3pC4vVcE5BsACQLyfXHFWxw8r1JYJ
P2SH Encumbrance: If 11 of 15 functionary public keys in template A' sign off, they can spend funds
Each functionary can turn around and derive their corresponding private keys P1', P2'...P15' seeing that they have been paid to

The way the user goes about doing this is by creating a LBTC transaction from scratch. The inputs to this transaction will look a bit funky, because they reference no previous LBTC output (think coinbase transactions when miners mine a new BTC block). Instead, the LBTC transaction embeds the original BTC transfer as its input, illustrating proof that the user previously surrendered their BTC rights into the hands of the watchmen. Seeing that they've been paid via the A' public keys and the 102-block confirmation period has passed, these functionaries can reverse engineer back to its corresponding private keys a' and assume control over those funds. With all the boxes now checked, the functionaries will validate the peg-in LBTC transaction by including it into a Liquid block.

Once the new LBTC are unlocked onto the Liquid sidechain, users are free to engage in transactional behavior at their full discretion. Very similar to the traditional Bitcoin blockchain, LBTC are governed by the underlying private keys of their owners, are immutable digital assets, and publicly auditable through a distributed ledger. LBTC also possess certain characteristic advantages over the tradition BTC network. Examples include resistance to reorganizations, 1-minute block times, deterministic block intervals (instead of dynamic block production), and ring confidential transactions.

When users are ready to convert their LBTC back into BTC, they engage in the "peg-out" process. This involves transferring their LBTC to a Liquid address where they wait for 2 block confirmations. Because there is no threat of reorganizations occuring on the Liquid network, we don't have to wait the meticulous 100-block period as we did during the peg-in process. After 2 Liquid blocks have ensued, watchmen sign off the *11* of *15* transaction on the Bitcoin mainchain, transferring money to the entitled owner, and subsequently destroying the LBTC in the process.

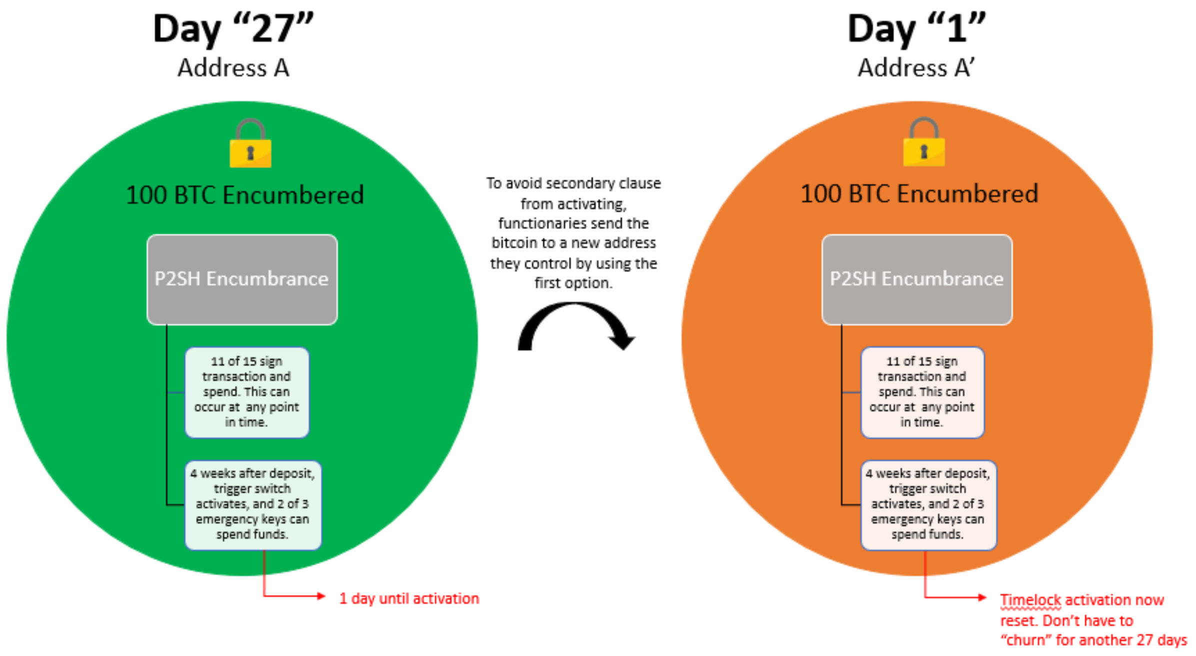
Peg-Out Transactions

When funds are ready to be transferred out of the Liquid sidechain and back into the Bitcoin network, the LBTC needs to be destroyed and the corresponding BTC released from multisig via the Watchmen efforts. In an effort to preserve privacy and mitigate security risks, functionaries control a Peg-out Authorization Key list that ensures outgoing BTC transactions are sent to a bitcoin address controlled by an authorized party. However, this creates an unfortunate situation where any user needing to redeem their LBTC to BTC must do so through an exchange or partner with a block signer. Essentially what we have happening is the user sends LBTC to a burn-style address, then waits 2 confirmations for that LBTC transaction to settle. The exchange sees the transaction as confirmed, pays out the user from their own hot wallet, and is subsequently "reimbursed" by receiving the BTC from the functionary multisig.

The intention of this strategy is aimed to reduce the risk of theft on the sidechain, which would consequently lead to theft on the mainchain. Peg-out addresses which the Watchmen pay out to are air-gapped cold addresses maintained by the exchanges. These addresses are constructed in a way that make it incredibly difficult to compromise the underlying private key.

Emergency Recovery Process

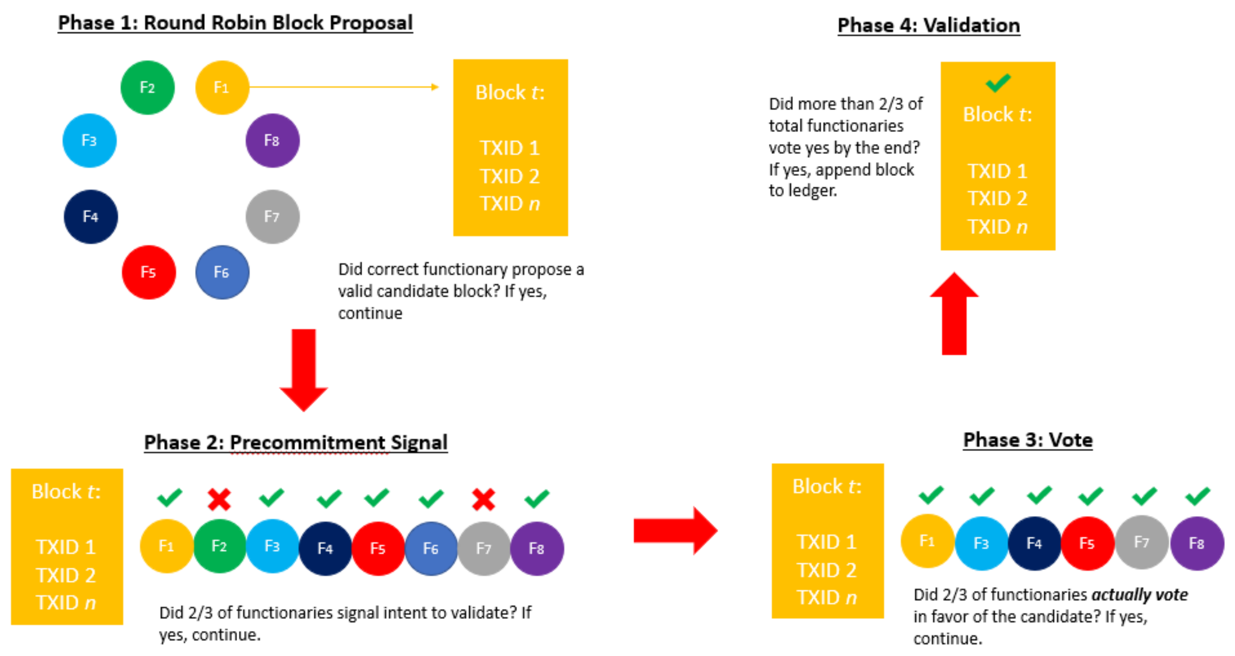
Remember how we earlier touched on the *<11 of 15 Multisig> OR <4-week timeout + 2-of-3 Multisig>_P2SH that keeps our Bitcoin locked up? The < _4-week + 2-of-3 Multisig>* clause serves as an emergency recovery component in a situation where ⅓ of the network goes offline or begins engaging in nefarious behavior. If such an event were to take place, the network would stall out and fail to produce additional blocks, but the Bitcoin on the mainchain would be stuck in escrow if the 11 of 15 component is not capable of being overcome. So, we introduce a second clause that consists of a completely different set of 3 emergency keys which can be used if (and only if) the network sits idle for 4 weeks, and we only need 2 of those 3 signers to sign off and move funds out. These keys are controlled by a totally different set of functionaries (undisclosed who these participants are for security reasons, but presumed to be geographically distributed attorneys) and can only be utilized after the 4 week lapse. Because of this, functionaries need to “churn” the locked BTC on the 27th day by sending it to to a new peg-in address they control, as a way to reset the timelock. Otherwise, 2 of the 3 emergency component would go into effect, and functionaries could then collude and steal user deposits.


Day “27”
Address A
100 BTC Encumbered
P2SH Encumbrance
11 of 15 sign transaction and spend. This can occur at any point in time.
4 weeks after deposit, trigger switch activates, and 2 of 3 emergency keys can spend funds.
1 day until activation
To avoid secondary clause from activating, functionaries send the bitcoin to a new address they control by using the first option.
Day “1”
Address A’
100 BTC Encumbered
P2SH Encumbrance
11 of 15 sign transaction and spend. This can occur at any point in time.
4 weeks after deposit, trigger switch activates, and 2 of 3 emergency keys can spend funds.
Timelock activation now reset. Don’t have to “churn” for another 27 days

Consensus Algorithm

In Bitcoin, miners bundle transactions into aggregated units called candidate blocks, with each candidate displaying a unique header. These miners subsequently perform redundant SHA-256 hashing on such headers, tweaking the nonce field element after each iterative hash, until someone in the group arbitrarily produces a digest below a predefined target. The winning party broadcasts their solution to the rest of the network, who can validate its authenticity. If the rest of the network is satisfied by the proof, they simultaneously adopt the winning party's candidate block as the "official" ledger state.

Liquid does not depend on a dynamic hashing contest to reach network consensus. Rather, it involves a Byzantine fault tolerant, round-robin multisignature scheme amongst a privileged group of *blocksigners* (the second group of functionaries we touched on earlier). These blocksigners inherit a role similar to miners in PoW, or validators in PoS; their job is to contribute to forward propagation of the network by creating new blocks, and thereby validate transactions. By limiting the number of candidates capable of contributing to consensus rather than remaining an open consortium, the network is able to achieve universal consensus with lower latency, faster block intervals, and lower computational burdens. In many respects, the consensus algorithm replicates features found in the Ripple RCPA protocol, described here. The system involves a round-robin style where blocksigners rotate turns proposing candidate blocks. After the blocksigner issues a candidate, their peers signal intent to validate the block through a precommitment scheme. At the conclusion of the precommitment round, if 2/3 or more of the group has signaled favorably, each member proceeds by actually signing the *k* of *n* block with their key. If the number of signees is greater than 2/3 then the block is accepted by the network and written to ledger. Note that the number of blocksigners who commit to sign and those who *actually* end up signing may be slightly different. For example, if someone loses connection midway through the process. The system rotates to a different block signer who proposes a new candidate block, and the process repeats.


Phase 1: Round Robin Block Proposal
F1
F2
F3
F4
F5
F6
F7
F8
Block t:
TXID 1
TXID 2
TXID n
Did correct functionary propose a valid candidate block? If yes, continue
Phase 2: Precommitment Signal
Block t:
TXID 1
TXID 2
TXID n
Did 2/3 of functionaries signal intent to validate? If yes, continue.
Phase 3: Vote
Block t:
TXID 1
TXID 2
TXID n
Did 2/3 of functionaries actually vote in favor of the candidate? If yes, continue.
Phase 4: Validation
Did more than 2/3 of total functionaries vote yes by the end? If yes, append block to ledger.
Block t:
TXID 1
TXID 2
TXID n

A compelling feature of the Liquid BFT is that a reorganization after two block's depth is impossible (assuming no malice or bugs). If a candidate block fails to reach its designated threshold, or the selected functionary for that 1-min time frame fails to perform their duty, no block for that interval is produced. Those transactions then carry over into the next candidate round. Users are not at risk of the network forking into two conflicting chains. The tradeoff to this scenario is that the network has the potential to stall out if blocksigners continually fail to reach the designated threshold required to append a new block.

Privacy

Liquid offers privacy features to users by embedding Ring Confidential Transactions directly into the protocol. Ring CT shields important metadata such as the transactional output values along with the destination/recipient addresses. Similar to Zcash, Liquid users have the ability to broadcast transactions on a transparent or confidential basis. However, the protocol automatically issues transactions on an opaque basis by default; users must manually configure transactions if they want the metadata to be publicly exposed. When dealing with CT's, only the participants that have access to the what is known as the *blinding key* are capable of viewing such information. This key is limited to the sender and recipient of the transaction, but can be passed along to other parties as they like.

The benefit to such a system is that it allows network participants selective disclosure. Selective disclosure gives network participants the ability to preserve the integrity of information that may be sensitive to their business or trading operations, while simultaneously offering the ease and flexibility of sharing this data in a frictionless manner with authorized parties (business partners, auditors, government officials etc). A great example of where CT might prove beneficial is for traders who are looking to place large orders. Since the Bitcoin blockchain is available for public audit, external actors can witness the movement of large funds to an exchange and front run, driving up the order book and forcing the trader to incur slippage.

Challenges

While there are many positive implications of LBTC, it also comes with its share of tradeoffs, especially for node operators. Some of the biggest performance limitations include:

- 1MB block size (relatively low)
- 10x CPU requirements over BTC
- 10x bandwidth requirements over BTC
- 10x transaction sizes compared to BTC due to CT features. However, drastic improvements are on the horizon with the implementation of bulletproofs (see XMR)
- UTXO sets could grow exponentially larger
- Synchronization and new node bootstrapping could get significantly more difficult and computationally expensive
- Higher trust implications (discussed in later segments)

Issued Assets

Another interesting feature Liquid provisions are Issued Assets. Issued Assets are analog to the ERC-20 standard commonly found on the Ethereum network, and are uniquely native to the Liquid sidechain. Any sidechain participant has the ability to issue and transfer their own Issued Asset with other users inside the Liquid ecosystem. The provisioners can elect for supply to remain permanently fixed, or maintain discretionary autonomy over its future inflation schedule. This is accomplished through a feature called the reissuance token, which subsequently can be constructed using a multisignature scheme and dispersed among numerous parties. Different scenarios where these assets may come into play is if an issuer wants to distribute some form of tokenized security across a highly liquid and interconnected network: gold storage tokens, fiat backed tokens, shares in a company, collectables, ETF-related products etc.

Risk Mitigation

If not already obvious, some attack vectors lay present. Functionaries are not only required to communicate with one another, but sometimes depend on these communication broadcasts in order to determine **how they themselves should act**

in specific situations. For example, during the precommitment of the consensus algorithm, if a blocksigner themselves signals “Yes” for a specific candidate block, but sees that not enough signers signaled accordingly to meet the *X* target, they will sit out during the actual signing phase. If an attacker is capable of infiltrating more than 1/3 of the blocksigners, they could force a network stall out simply by refusing to vote, and preventing the require 2/3 consensus to be achieved in order to create a block. If they are capable of corrupting 2/3 of the blocksigners, they could validate nefarious blocks.

To prevent against such threats, when new functionaries join the network, Blockstream provides these agents with proprietary hardware to conduct network operations. This hardware consists of two components: a server with precompiled software to perform network operations, and a “offline” hardware module to store functionary keys. These hardware boxes are designed in a way that purposely limits their vulnerabilities to network stack exploits which could compromise the consensus process. Each functionary server is presumably stored in a secure, anonymous location, and only communicates to other functionary servers via Tor. Tor connections provide security by obfuscating the server’s IP address, reducing the risk of physical theft or tampering while simultaneously hedging against potential DDOS attacks on the server. RCP calls are severely restricted and the hardware is specifically configured to only allow incoming SSH connections when a button on the box is pressed. Meaning to make any software or network alterations, you need to be in physical location of the hardware, severely limiting attack vectors.

The most obvious threat to the Federation is the group of functionaries themselves. At the end of the day, users transacting across the Liquid network are forced to trust the functionaries. If more than ⅔ of the functionaries collude, they have the ability to steal user funds. However, given the game-theoretic incentive models at play, the probability of this happening seems relatively low. First and most obvious, these functionaries are not anonymous parties but publicly recognized institutions that operate in jurisdictions where laws exist. The probability of a set of functionaries openly colluding to steal user funds without being subject to legal recourse are slim to none. Second, with regards to covert collusion (playing it off as a hack), exchanges would naturally be disincentivized to steal from their customers, thereby driving away core business revenues (exchange fees), unless the payoff from the attack outweighed the cost of collusion plus opportunity cost of capital relinquished from future business revenues. Consider also the fact that the cost of collusion grows logarithmically with respect to number of total parties colluding. Unless the Liquid network grows to a considerable value, it is highly improbable scenario this happens, given how much money these businesses are making on trading fees.

Workflows

Liquid's audience will primarily cater towards exchanges, market makers and other large liquidity providers, along with their users. While Issued Assets might offer compelling opportunities for asset issuance, the scripting capabilities offered by other blockchain systems and higher levels of throughput make it a highly competitive and saturated market. Traders on the other hand are in fervent demand for quicker asset settlement, illustrated through the radical price discrepancies that exist across premier exchanges. Being able to move significant volume across platforms will reduce friction and costs for traders and liquidity providers alike, while simultaneously improving overall market efficiency by reducing spreads between exchange indexes.

Conclusion

While not the most bleeding edge implementation of sidechain technology, Liquid addresses a legitimate and current need in the cryptoasset ecosystem by allowing traders and exchanges to provision liquidity and settle transactions in short order. Benefits include embedded privacy through confidential transactions, higher throughput, faster settlements times, and ability to issue custom assets. Drawbacks include increased computational costs to node operators, higher degrees of trust than Nakamoto consensus, and counterparty risk of stolen or irrecoverable funds. If successful, Liquid validates the sidechain model, and opens the Bitcoin blockchain for increased functionality in further implementations. This no doubt comes with certain risks, but is a development challenge worth pursuing given the non-coercive nature of this technology.

Many thanks to Allen Piscitello from Blockstream , Pietro Moran and Hasu for their contributions to this piece!

Cryptic History: The Cypherpunks

William Smith

November 19, 2018

“[T]he question at once arises whether it would not be equally desirable to do away altogether with the monopoly of government supplying money and to allow private enterprise to supply the public with other media of exchange it may prefer” F.A. Hayek

The sixteen square inches of inked fabric constituting a federal reserve note represents power, not value. It wasn’t always this way. Money proper, as evolved through the intertwining processes of exchange, holds value because of what it represents: all other goods. The Regression Theorem, developed by early 20th century economist Friedrich Wieser and his student Ludwig von Mises, demonstrates this process. Money, when examined historically, derives its value from comparisons to commodity goods. Money is intimately connected to barter. But government has usurped this role, replacing commodity value for the value of violence.

Money is a government service, a service facilitated by graft and perpetuated by surveillance.

First, inflation is theft. When the government artificially increases the supply of money the purchasing power of all money decreases across the board. Real wealth and savings evaporate. Since the creation of the Federal Reserve in 1913, the U.S. Dollar has lost 95 percent of its value. That’s generational wealth simply stolen.

Exchange surveillance is another manifestation of the government money monopoly. As a third party with political interest, any exchange using the government system comes under bureaucratic scrutiny. Again, it wasn’t always this way. The National Banking Acts of 1863 and 1864 taxed private money out of existence. Now, the U.S. Government holds the corner on money and therefore exchange.

The 20th century is dotted with exchange intervention: FDR’s 1933 gold seizure, the War on Drugs, and modern day civil asset forfeiture to name a few. Morality aside, arrested exchanges decrease societal wealth. In economics, this is called deadweight loss: both parties lose out on a mutually beneficial exchange.

These costs ignited opposition, but the revolt had limited tools. Then came the computer age.

The Cypherpunk Movement

Bill Clinton announced the end of "big government" in 1996. He was a little late or perhaps a tad ignorant.

The Cypherpunks, a conglomerative anarchist oozing of tech enthusiasts, hardcore libertarians, radical leftists, and college dropouts were already putting Clinton's claim into code. Living in communal dwellings in Silicon Valley and elsewhere, the Cypherpunk's were Spaniards in the New World—coding was their steel.

To be reductionist, the Cypherpunks were Orwellian. They watched with tempered expectations as the internet flowered into being: communication on such scales had obvious drawbacks and demanded a reexamination of privacy. As prominent Cypherpunk Eric Hughes wrote in his *Manifesto* :

We must defend our own privacy if we expect to have any. We must come together and create systems which allow anonymous transactions to take place. People have been defending their own privacy for centuries with whispers, darkness, envelopes, closed doors, secret handshakes, and couriers. The technologies of the past did not allow for strong privacy, but electronic technologies do. We the Cypherpunks are dedicated to building anonymous systems. We are defending our privacy with cryptography, with anonymous mail forwarding systems with digital signatures and with electronic money.

Modern cryptography is indebted, ironically, to the state it seeks to undercut. The surveillance systems that sprung up from 20th century conflicts, namely the *unconditionally secure* One-Time Pad, were foundational to the Cypherpunks. This technology eked out of government offices in the 1970s and into private hands—it bloomed.

The Cypherpunks were bullish on individual data protection. As the internet grew, so must security. Public and private key encryption, the children of two papers entitled "New Directions in Cryptography" (1976) and "Blind Signatures for Untraceable Payments" (1981), were the most notable steps toward repurposing cryptographic methods for individual data protection.

In math and prose, these papers described what is becoming the new normal for open gateway communication. "New Directions" and "Blind Signatures" identified an evergreen issue: electronic messages sent via a third party system can be and will be compromised. Encryption, furthermore, has real limits: both the key and encrypted message reside in the sendee's hands. So, how do you send the key without compromising the message or the parties involved?

As with most problems, the answer was waiting in Berkeley, California. Computer scientist David Chaum's "Blind Signatures" solved the sendee's problem by masking the signature before the data could be accessed. His cryptographic delivery system ensured that: 1) all parties remained anonymous, 2) messages would be verifiable to both parties and 3) messages would be recorded for public confirmation.

Chaum's blind signatures went on to be the basis of the eCurrency movement premptying today's crypto bonanza such as *Digicash,* Chaum's own digital currency; *Hashcash,* an anti-email spamming mechanism; and *Egold,* an electronic currency backed by gold bullion.

Yet all precursors piddled out. Why? Among other things, the issue of double spending. In digital currencies, double spending occurs when a cryptocurrency is used for two purchases. Without a third party like a bank to oversee transactions, it is possible to both spend and keep ones money. A solution was on the horizon, however.

Nakamoto

Ten years ago this month, an anonymous Cypherpunk named Satoshi Nakamoto sent a white paper to a Cypherpunk email list. His paper, "Bitcoin: A Peer-to-Peer Electronic Cash System," proposed a digital ledger as a solution to double spending. Colloquially called 'blockchain' but more accurately termed 'timechain,' Nakamoto created what amounts to a decentralized database system as an immutable digital ledger. Anyone using bitcoin can both see and monitor the ledger. Simply put, the blockchain prohibits double spending.

Nakamoto's design is the realization of Hugh's *Manifesto*: electronic money. By integrating Chaum's blind signatures into a timestamp system, Nakamoto created the world's first trustless, decentralized, peer-to-peer digital currency.

At present, Bitcoin is the fulfillment of the Cypherpunk movement. With private currencies, the state's two pronged trident of inflationism and surveillance are dulled considerably. Transactions cannot be infringed on by a third party. Opting out of fiat systems ends inflationism. Money is returning home.

Skeptic's Guide to Bitcoin: An Honest Account of Fiat Money

A framework for Bitcoin skeptics, part 1

By Su Zhu and Hasu

Posted November 21, 2018

This is part 1 of a 4 part series. See additional articles below

Image credit: Pexels

Discussions about Bitcoin often start from the presumption that fiat money is terrible and against the will of the people. We think a discussion about money and Bitcoin should start by acknowledging both—the good, and the bad—of the fiat system.

Bitcoin has an immense pull on all critics of the current fiat system. Bitcoiners tend to be “sound money”-advocates—meaning, they prefer money that can’t be created or destroyed by a central party and instead has a fixed or at least predictable supply, like gold, instead of one where the government can centrally manage the money supply.

While sovereign money has undeniably led to a lot of abuse, as evidenced by instances of hyperinflation, seizures or capital controls, the absence of all control brings problems of its own. In the example of the Euro countries entered a monetary union without a fiscal union. The result is a currency that strongly benefits the export-oriented nations like Germany while hurting net importers like Greece or Portugal.

Fiat is incredibly useful when people want their country to have monetary sovereignty. Ceding control over the money supply to a foreign institution (for example by pegging to an uncontrollable asset like gold or bitcoin) removes the ability to tailor the monetary and economic policy of your money to the needs of your economy.

Benefits of sovereign money

Fiat proponents would say that each country has its own economic problems that need personalized solutions. That’s why the UK, Norway or Sweden never joined the Euro and most of the southern European countries today regret it.

Monetary sovereignty, on the other hand, unlocks certain benefits for countries:

1. The government can better finance large social projects and benefits by printing more money, which is good for beneficiaries of such a welfare state.
2. The government can demonetize or freeze assets easily, which is good for anticrime (most people aren’t criminals but want to be protected from crime).
3. The government can manage fiat liquidity with the goal to kickstart the economy when needed and smoothen business cycles.
4. The government can keep the currency stable against a basket of goods and services to increase its usability in the economy
5. Countries can devalue their currency in international trade to boost their exports and to reduce their trade deficit

Contentious inflation

Because governments tend to create more money than they destroy, the value of fiat money erodes over a long enough time against all real assets. And since creating new money doesn’t create new wealth, the seigniorage from spending it first can be seen as an implicit tax on all other holders of the money. A small level of inflation is no side effect of fiat money but is actively targeted by central banks today.

Inflation incentivizes firms and households to invest into productive enterprises and makes it easier for the government to meet future obligations, as nominal debts slowly become easier to pay off when each unit of money loses a bit of purchasing power. It's also a great way to tax the shadow economy in places where compliance can be low and collecting taxes hard, like India, much of Africa and South America.

Is Fiat inevitable?

But the ability to tax through inflation is more than a tool for wealth redistribution inside an economy. It can also be used to gain an advantage in supranational competition. If China could endlessly allocate resources towards military production by printing more money, while the US has to allocate via taxation, it would end in a huge uphill battle for the US. In that scenario, China would effectively borrow money from their own citizens via implicit taxation but that money would flow back into their economy later from the spoils of a won war.

So while the observation of sound money advocates that the abolishment of the gold standard has led to more sustained warfare is not wrong, we can imagine the situation similar to the deterrence theory. While it might be better for the world at large to be on a sound-money standard (or have no nuclear weapons), the equilibrium would be unstable: once one player abandons it, he forces everyone else to follow him to a new, inferior equilibrium.

Another parallel is the rise of the nation-state itself, which quickly proved much stronger militarily than the feudal state as rulers were able to create national armies. The newly emerging nation-states dominated all older forms of political organization which forced the remaining actors to centralize into nation-states themselves.

Fundamental problems

While the fiat system has many positives, the last decades have revealed some fundamental problems. If you want your money supply to be flexible, you still have to decide who gets to manage it.

In our current system, central banks and governments have largely outsourced the creation and destruction of money to commercial banks. New money is created when these banks credit lender accounts without debiting someone else. For the liabilities they have towards their customers, they only need to keep a small fraction of actual deposits in reserve. This is known as fractional reserve banking.

Unfortunately, that causes the vital functions of a financial system, which are

1. to facilitate payments
2. to broker investments/loans
3. to provide insurance

to become hopelessly intertwined. If too many borrowers were to default on their loans, it could cause the entire system, including payments, to fail even for customers who never agreed to have their deposits invested in risky schemes in the first place.

It also creates a severe disconnect between risk and reward for the financial sector. When gambling is allowed to take place on the same balance sheet as the payments system and other crucial parts of our financial infrastructure, the government becomes obliged to bail out banks once they get into trouble from their risky lending activities. That drives the banks to even riskier behavior, knowing that more volatility can only increase their upside as potential losses are capped.

A new conversation on money

Even fiat proponents will agree that we have some fundamental problems in our current financial system. There are system-critical companies that cannot be allowed to go bankrupt because the different parts of the system are too interconnected and not modular. A crisis in one part can spill over into the rest of the economy and put the whole system at risk. This is not exactly how you should design world-critical infrastructure.

This is my favorite starting point for thinking about Bitcoin. It is more than “fiat vs sound money”. Bitcoin proponents are driven by a collective interest in exploring how we can evolve money for a better society, and how we can make the global financial system more stable and distributed. The creators of Bitcoin didn’t see how they could contribute to fixing the holes in the current system, because of powerful incumbents who have a strong interest in preserving the status quo. So they found a way to compete.

But it’s not a hostile takeover attempt. Bitcoin’s purpose is not to destroy, but rather to offer us freedom and choice, and in doing to spark a conversation about the nature of money and its crucial role in our society.

Continue with Part II: Unpacking Bitcoin’s Social Contract

This article is part of an ongoing series—the Bitcoin skeptic’s framework. Follow me on Twitter and Medium to be notified when the next episode hits.

- **Part I—An Honest Account of Fiat Money**
- Part II — Unpacking Bitcoin’s Social Contract
- Part III — Bitcoin and the Promise of Independent Property Rights

Acknowledgments

Thank you to Nic Carter , Ataraxia_Invest and Joe Kendzicky for their feedback.

How Cryptography Redefines Private Property

By Hugo Nguyen

November 26, 2018

This is Part 3 of a 5 part series

- Part 1 - The Anatomy of Proof-of-Work
- Part 2 - Bitcoin, Chance and Randomness
- Part 3 - How Cryptography Redefines Private Property
- Part 4 - Bitcoin's Incentive Scheme and the Rational Individual
- Part 5 - Bitcoin: Two Parts Math, One Part Biology

Cryptography is the art of protecting and breaking secrets. Bitcoin employs a special type of cryptography, called public-key cryptography, in order to facilitate its system of storing and transferring of value. It is through this mechanism that Alice can keep her bitcoins secure or send some of them to Bob, or that coinbase rewards are issued to the miners. The term "cryptocurrency" derives its name from this usage. In this third part of the Bitcoin Fundamentals series, we will explore the pivotal role public-key cryptography plays in the creation of digital hard money, as well as its ramifications on society at large.

To fully understand the significance of public-key cryptography, we must first take a few steps back and understand how our society is principally organized around the concept of property.

What is Property?

Property never has been abolished and never will be abolished. It is simply a question of who has it. And the fairest system ever devised is one by which all, rather than none, [are] property owners. — A. N. Wilson

Property has wide-ranging definitions. The idea of what constitutes property often changes to reflect contemporary beliefs and circumstances. For example, early discussions of property almost always exclusively referred to land. Since it was mostly land that was the source of constant conflicts and struggle for power, it was natural that land occupied the minds of ancient thinkers. Later, as society develops, property was expanded to include intangible assets such as patents and copyrights, and then expanded again to encompass all things deemed necessary for life and liberty.

Formally speaking, property is a legal concept defined and enforced by a local sovereignty. The specific implementation varies greatly depending on the political system, from monarchy to communism to democracy, among others.

Discussions on property dated back to Greek civilization. Plato and Aristotle, the two fathers of Western philosophy, laid much of the groundwork for the fierce and at times extremely bloody debates that followed in the subsequent 2,500 years.

Plato, likely inspired by the Spartans, was against private property of all sorts, including men's wives and children. He rejected the idea of "mine" and "not mine", "his" and "not his". According to him, private property corrupts the human soul: it fosters greed, jealousy and violence. An ideal Platonic society would eradicate private property entirely. This school of thought was later developed into a full-fledged system, most notably by Karl Marx, the father of communism.

Aristotle, Plato's best student, had other ideas. Aristotle rejected Plato's argument that common property would remove vices and violence, arguing that people who share stuff tend to fight more than those who own them personally. Aristotle also believed that private ownership is crucial to progress because people would only be incentivized to work hard for the things they own. Furthermore, only when private property rights are respected, people can have the opportunity to fully grow and afford to be virtuous. As the late historian Richard Pipes eloquently put it: "Human beings must have, in order to be."

Left: Plato (427–347BC) and Aristotle (384–322BC)

Plato's central argument against private property rested on morality. Aristotle also responded to Plato on the ground of morality. But some of Aristotle's arguments went beyond morality and into the realm of economic reality. (Interestingly, like his teacher, Aristotle was against trade and profit-seeking, so he did not develop a consistent economic framework.)

The discussions on property since Plato & Aristotle have evolved to include mainly four aspects: morality, politics, psychology, and economics [1]. Among these, psychological and economical arguments for private property probably bear the most weight, since they address the world as it is, not how it should be.

One of the fundamental questions regarding property is that **between property and sovereignty, which comes first?**

Some influential thinkers, such as Harrington & Locke, believed that property predated sovereignty. They believed humans intuitively understand property, even without the existence of a state.

To them the primary function of a state is to protect property rights. A sovereign state is judged on its merits to fulfill this responsibility. Locke went further and stated that individuals have the right to rebel against the state if it fails in its duty to preserve property rights. Adam Smith, the father of capitalism and modern economics, had a similar view. Smith recognized that property and government were dependent on each other, and argued that “civil government could not exist without property, as its main function is to safeguard property ownership”. (Smith differed with Locke, however, on whether property rights are “natural”—he believed property rights are “acquired” rights, not “natural” rights).

In contrast, other thinkers, most prominently Hobbes, believed it was the opposite, that property is a creation of the state. He regarded property simply as the product of authority and the acceptance of that authority [2].

There has been mounting evidence, particularly in the later half of the 20th century, that vindicated Harrington’s & Locke’s view. The 20th century saw a large-scale social experiment in attempting to abolish private property via the spread of communism. This experiment resulted in the Cold War and numerous proxy wars across the globe, culminated in the Cuban Missile Crisis in 1962, and ended dramatically when the Soviet Union finally collapsed in 1991. Following this collapse, the remaining communist states were split into two camps. Some switched to capitalism, as many Central & Eastern European countries did. Some retained only a nominal form of communism, as China & Vietnam did—in these countries the political regime purports to represent the “bourgeois”, but the economy functions much like that of a capitalist state. Almost all surviving communist states have adopted free-market ideology, and were forced to respect property rights (to a certain extent) in order to survive.

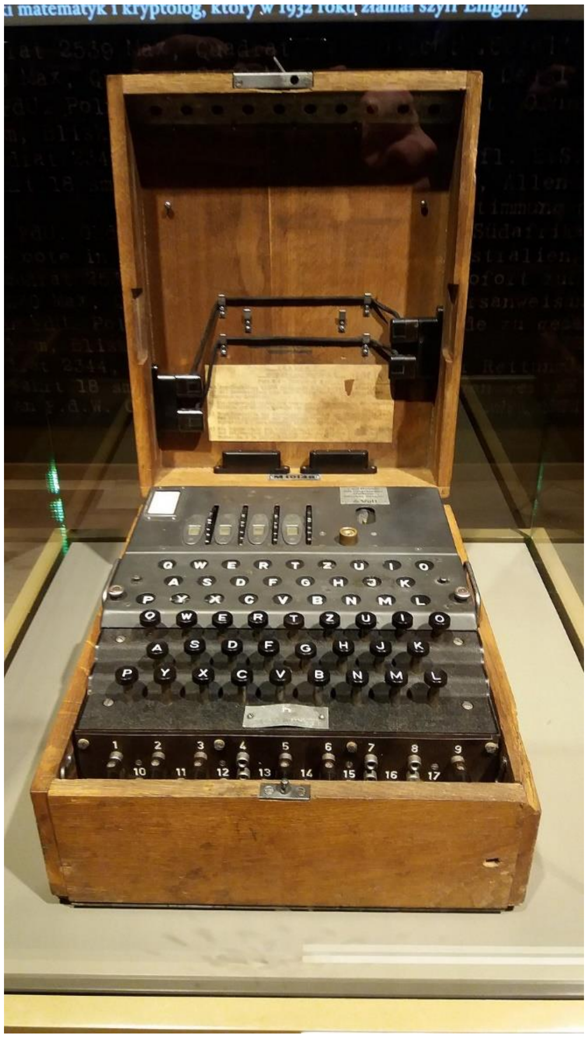
In summary, what the 20th century has shown us is that governments that failed in their primary role of protecting property ended up either dead or forced to eventually respect private property. This proves that the state is subservient to, and is a derivative of, property needs, not the other way around. Property, by and large, comes before the state .

Despite this, in the short to medium term, individuals do have to depend on the state for preserving their property rights, and suffer a great deal when it fails to do so. We will see how cryptography can help break this cycle of dependency. But first, let's quickly look at how modern cryptography came about.

Public-Key Cryptography

Until recently, the development of cryptography has been driven mainly by wars and rivaling states.

Prior to WW2, classical cryptography was more about obscurity than mathematical rigor. If there was any math involved, it was simply accidental. This changed with a couple of developments:



- The invention of the telegraph dramatically increased the amount of data that needed to be secured
- The invention of the radio heightened the need to encrypt public communication channels, as opposed to relying on stealth methods
- The invention of the computer brought incredible computational power to cryptanalysis; breaking codes became much easier

These developments culminated in WW2 and saw the Allies heroically break Germany's state-of-the-art cryptographic device, the Enigma machine, with notable help from Alan Turing. The breaking of Enigma decidedly concluded the era of classical ciphers (a cipher is a particular method of doing encryption and decryption) and called on cryptographers to look for stronger ciphers— ciphers that are based on a more solid, mathematical ground.

Left: End of an era: the Enigma machine— image by LukaszKatlewa, CC 3.0

Luckily, there were hints for where to look. One major problem with classical ciphers was the problem of secret keys distribution—a problem common to all symmetric ciphers (symmetric ciphers are ciphers that use the same key for encryption and decryption). Basically, with symmetric ciphers, since it is not recommended to reuse secret keys, the number of keys that need to be distributed is proportional to the amount of data that need to be secured—which as mentioned, exploded at an unprecedented rate. As a result, one big research area at the time was to find ways to cheaply and securely distribute keys at scale. The insight for asymmetric ciphers began when a few cryptographers flipped this problem on its head and pondered: do we actually need to distribute the secret keys at all?

This research direction eventually narrowed the search down to one-way functions, and it was not long until a brand new solution was found—one which does not require the distribution of secret keys. Diffie, Hellman & Merkle discovered public-key cryptography, the first ever *asymmetric* cipher, in 1976. One year later, Rivest, Shamir & Adleman published the first implementation, the now-famous RSA algorithm. Public-key cryptography was also independently discovered, but not implemented, a few years earlier by Ellis & Cocks in Great Britain. However, Ellis & Cocks' research was classified and not publicly acknowledged for 27 years.

Thus, public-key cryptography, today considered the greatest cryptographic achievement in two thousand years, was born. It took several decades for this technology to filter down to the masses, largely thanks to the efforts of the cypherpunk movement. The cypherpunks, foreseeing that the state's monopoly on this new technology creates a huge imbalance of power and a potential threat to individual liberty, took incredible risks to bring public-key cryptography to civilians. When they successfully did, the gates were flung wide open. For the first time in history, individuals could afford encryption as strong as the military's.

But it was not until the invention of Bitcoin that public-key cryptography realizes its full potential. Its application in digital hard money would redefine private property.

Privately-Owned Digital Goods: An Oxymoron?

The arrival of the personal computer and the Internet introduced a new asset class: digital goods. Digital goods are intangible digital bits of information, packaged in various encoding formats. Examples are Wikipedia articles, MP3 files, software, etc.

Digital goods ushered in a new era of information-sharing never before seen. Any piece of information could be instantly reproduced and distributed at scale, at almost zero marginal cost.

However, early on digital goods did not significantly change the nature of private property. As it turns out, digital goods are notoriously bad at staying private.

The problem with trying to establish exclusive ownership over digital goods is that they are infinitely copyable. But that did not stop people from trying anyway. The state and businesses' preferred method of imposing control over digital goods is through regulations. For example, the US created a complex export control & regulations program to prevent secrets, particularly knowledge of advanced technologies, from leaking to other states. In business, the Big 3 music labels and the Big 5 movie studios spend a large amount of money lobbying for anti-piracy laws, and suing people for violations. Similarly, big software companies learn to build patent war chests. In the process they exploit the patent system and cripple smaller companies. At best, regulatory solutions at enforcing digital exclusivity are expensive and ineffective. At worst, they create an environment where a few players can game the system to their advantage.

Non-regulatory solutions are equally inefficient. One such example is Digital Rights Management (DRM), an access-control technology designed to restrict the use of digital goods. DRM does provide some basic level of security against piracy. However, DRM suffers from a fatal flaw: the security only needs to be bypassed once, and there would be no recourse.

One area where we saw limited success at creating digital exclusivity came from the gaming industry. Game creators figured out that if they could create virtual goods (a subset of digital goods) within the games they designed, these goods could serve to add to the gameplay, as well as a nice bonus source of revenue. Since the game creators control 100% of the environment within which these virtual goods operate, they could impose restrictions on them that are not possible otherwise, including making them inaccessible or rare. However, in-game virtual goods have limited application, and game users are dependent on the game creators to not change the game rules.

So it might have seemed like digital goods and private property are at odds. If something is digital, it is almost impossible for it to be private or for someone to have sole possession of it. *Privately-owned digital goods* sounds like an oxymoron.

The inability to exclusively own digital goods means that everyone can take advantage of them. Individuals can use them to empower themselves, but the state is also able to become more powerful through data collection, surveillance, and propaganda—all made easier with digital goods. Digital goods, in their pure form, do not change the power structure between the individuals and the state.

But there is a way for digital goods to become private property, only if we can overcome 2 major hurdles:

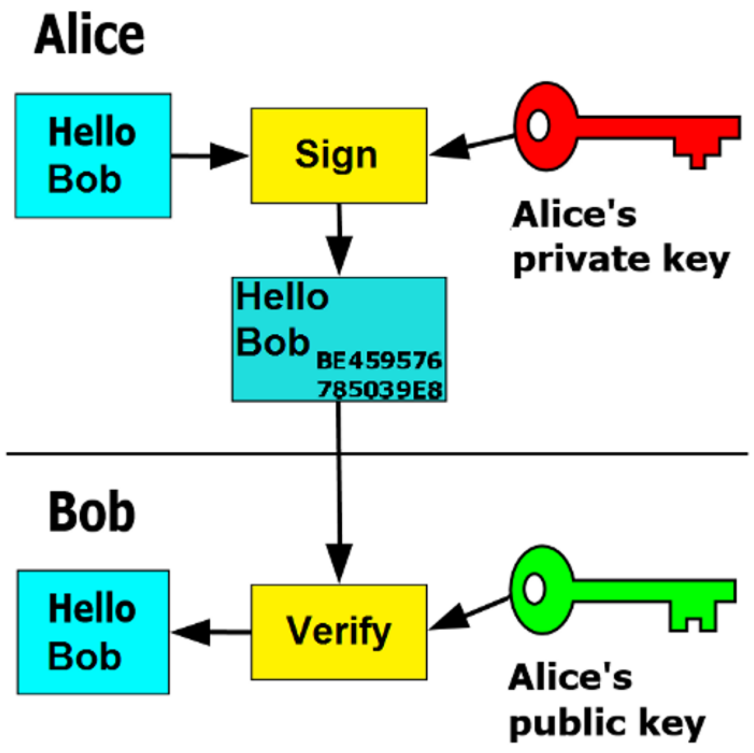
- **Scarcity**: the ease of reproducing and redistributing digital goods in unlimited quantity is precisely what makes exclusive ownership difficult.
- **Transferability**: the ability to dispose of an asset by sale is a crucial component of ownership. Without this, ownership loses much of its power.

It turns out that these two desired attributes are also the prerequisites of money. If we are able to create a digitally-scarce, transferable digital good, not only this good could be privately owned, it would become a great store-of-value and ultimately a new form of money. Digital "hard" money.

Bitcoin solves both problems. Bitcoin creates digital scarcity by imposing a fixed supply on itself in software (a.k.a. consensus rule), and then protecting that consensus rule with a large amount of energy (in the form of Proof-of-Work) through a decentralized network of mining machines. As long as mining is sufficiently decentralized, it would be nearly impossible to change this fixed supply rule or bring down the network. We discussed this key innovation in-depth in Part 1 of the series.

The problem of transferability, on the other hand, can be solved by using public-key cryptography. Specifically, a unique application of public-key cryptography is to generate unforgeable digital signatures. The digital asset could be coupled with a public-private key pair. This single key pair could then generate as many digital signatures as needed, each one represents a piece of the underlying asset. These digital signatures effectively "tokenize" the asset, making it transferable between individuals [4].

In short, public-key cryptography, together with digital scarcity, have given us digital hard money. Digital hard money inherits all the strengths of digital goods: instant, borderless, and impossible to regulate or control. Most importantly, this digital hard money can be privately owned.



Left: Digital signatures using public-private key pair — image courtesy of Wikipedia, CC 4.0

It is hard to overstate how big of a disruption this is.

For the first time in history, a form of private property exists that is completely independent of jurisdiction or the law [5]. Private keys and the bitcoins they control are private property *de facto*, not de jure.

Public-key cryptography's application in digital hard money greatly changes the social contract that has bound human societies for thousands of years: that individuals need the state for the protection of property, and the state in turn needs individuals to finance its continued existence. One has to wonder, if securing property is the primary function of a state, then will its role—and power—be significantly reduced in a world run on digital hard money?

In summary, public-key cryptography's application in digital hard money redefines private property and has the potential to uproot the very foundation of society as we know it. The consequences might take decades to play out, but when it happens, the impact will likely be incredibly far-reaching.

**This is part 3 of the Bitcoin Fundamentals series. Check out the full series here: part 1 , part 2 , part 3 , part 4 , and part 5 .*

Acknowledgments

Special thanks to Jimmy Song, Hasu, Nic Carter and Steve Lee for their extremely valuable feedback.

[1]: Property and Freedom, Richard Pipes

[2]: Property and Ownership , Jeremy Waldon

[3]: The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography , Simon Singh

[4]: It is very possible that public-key cryptography is the only way to tokenize a digital asset this way. Imagine implementing Bitcoin using a symmetric cipher.

[5]: Physical assets like gold can be and often is heavily regulated. For example, FDR outlawed & confiscated private gold during the Great Depression.

Thanks to Steve Lee, Hasu, Nic Carter, and Jimmy Song.

An Honest Explanation of Price, Hashrate & Bitcoin Mining Network Dynamics

Bitcoin mining update—Part 1 of 2

By Christopher Bendiksen

Posted November 29, 2018

This is part 1 of a 2 part series

- Part 1 **An Honest Explanation of Price, Hashrate & Bitcoin Mining Network Dynamics**
- Part 2 Beware of Lazy Research: Let's Talk Electricity Waste & How Bitcoin Mining Can Power A Renewable Energy Renaissance

As a part of our ongoing effort to keep the crypto community knowledgeable on the inner workings and development of the Bitcoin mining network, CoinShares Research just published our latest *Bitcoin Mining Report.*

Much has happened since our first report in June. In fact, more than could be reasonably covered in a single Medium post, so I decided to break out some key takeaways into two posts.

This first is part *'TL;DR';* part commentary on recent events; and part in-depth explanation of terms, methodology and the concept of Bitcoin creation costs.

The second summarizes our latest findings on the electricity mix of Bitcoin mining—a hotly contentious topic in its own right.

Let's start with the question that everyone is talking about:

Yes—Bitcoin miners are shutting down hardware and exiting the network. No—this is not going to cause a "death spiral."

Since the start of 2018, the price of bitcoin has declined from a peak of nearly $20,000 to approximately $4,000—an 80% fall. Miners receive their income in bitcoin and thus depend on exchanges—and exchange prices—to cover their expenses and (hopefully) turn a profit.

So what does this all mean for the network?

Before we go there, let's 'level set' with a brief primer on CAPEX, OPEX and ROI for non-MBAs. *(Skip ahead to the Results section under the photo if you're already familiar with these concepts.)*

Mining capex, opex and ROI for non-MBAs

For those unfamiliar with the lingo I am about to employ, when I use the term ***capex*** I am referring to ***capital expenditures,*** and when I use the term ***opex***I am referring to ***operational expenditures***. *Briefly explained:*

Capex describes all expenditures related to the acquisition of *capital*, such as mining gear, racks, property (if applicable) etc.

Opex describes all *ongoing* expenditures, like wages, electricity cost, rent (if applicable) etc.

I will also be using the term ***ROI***which simply means ***return on investment***. ROI is positive if an investment is profitable, and negative if it produces losses.

When establishing a mining operation you can structure it between two extremes:

On one end of the scale, you can rent all your equipment, housing, maintenance, etc. and pay-as-you-go for electricity. Under this setup, all expenditures are **opex** and your operation owns no capital. If nothing is contracted, you have no assets and no liabilities.

On the other end of the scale, you could decide to buy all of your gear, a plot of land, and fixed electricity and employment contracts. In this structure, all expenditures are pre-paid **capex**. Your operation owns capital (assets) but also has contractual liabilities.

In reality, most mining operations fall somewhere between these two extremes — a combination of both opex and capex.

For example, some may own gear, but rent space and pay as they go for electricity. Others rent gear, own land, and buy electricity on short contracts. The point is that many miners have both capex ***and*** opex components to their cost function, and our assumption is that ***on average***, most mining operations have a bit of both. More on this later.

Mining gear, like most other productive capital has a finite lifetime and will see its productivity deteriorate over time until it is either worn out or obsolete. This is the concept of **depreciation**. In practice, it means that mining gear must generate more free cash-flow over its lifetime than its acquisition cost plus ongoing capital costs in order to be profitable.

The value of the gear you have is therefore understood to depreciate from its acquisition cost down to either scrap cost or zero.

If your gear is productive for a long time, it has a long depreciation horizon; conversely if it is only productive for a short time, it has a short depreciation horizon.

With longer depreciation schedules, you have more time to spread out the total purchase price.

Depreciation is booked as an ongoing cost in accounting, but does not impact actual cash-flow as it is normally “front-loaded”— or in colloquial terms, prepaid—and therefore does not impact *cash-flow*, or by extension, *cash cost*.

Free cash-flow here refers to cash-flow that exceeds all of your opex. *Cash cost*, or the cost of capital, refers to the return you would have received if you put your money into a “risk-free” (lol) investment like US Treasury notes instead of investing in mining.

For miners, there are two cutoff levels for bitcoin prices that matter:

1. The first is their ***all-in ROI breakeven*** level, above which they make a profit on their investment, and below which they make a loss on their investment.
2. The second is their ***cash-cost breakeven***level, above which they are cash-flow positive but still potentially loss-making (ROI can still be negative if they never make enough cash to cover what they paid for their mining gear); and below which they are cash-flow negative and thus—depending on their industry view, risk appetite and capital levels—are likely to shut off mining gear entirely.

This is an important distinction because miners—*even if they are realizing a negative ROI and unable to recover their capex—will keep mining for as long as the gear is returning* ***any*** *positive cash-flow in order to recover* ***as much*** *of their capex as possible.*

As soon as opex exceeds income and mining gear becomes cash-flow negative, it no longer makes sense to even leave on, as it is now just burning cash.

This is when miners remove gear from the network*— not *at the point of negative ROI.

In our report we calculate a ***market-average all-in breakeven cost for creating one bitcoin***. This is our best approximate for the bitcoin creation cost above which the *market-average* miner will make a positive ROI.

At the risk of repeating myself ad nauseum here, ***this is not the level below which miners will turn off their gear.***

It is simply the level below which they will lose money on their investment, making them unlikely to remain players in the mining industry over time, unless they have bottomless pockets of investment capital available.

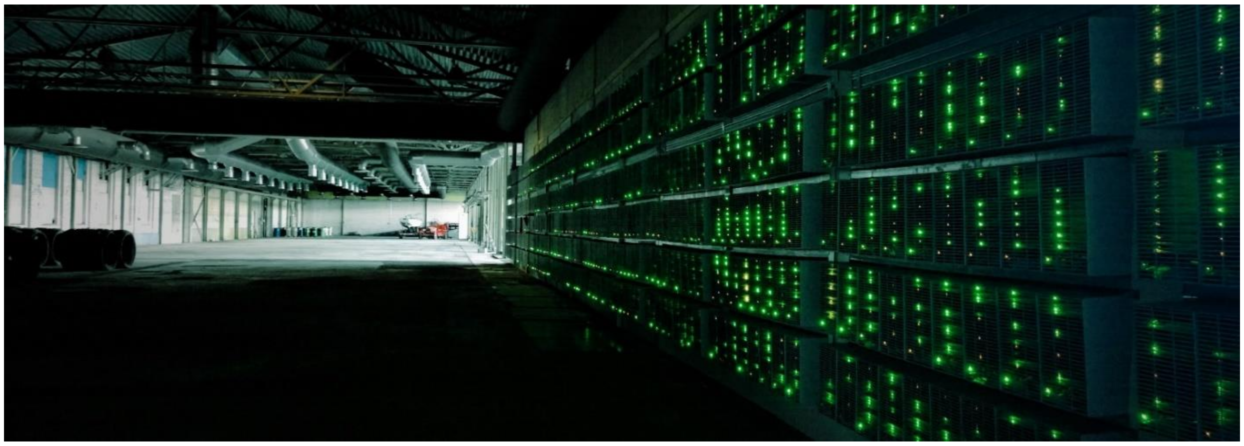


Photo by Marko Ahtisaari, used with permission under Creative Commons 2.0 license.

Now, the results…The Cost to Mine One Bitcoin:

Average 'all-in' cost: $6,800/btc | A *verage cash cost:* $3,400/btc

In our June report we estimated a *market-average all-in* cost of creation of approximately $6,500 per bitcoin. We arrived at this number by assuming a market-average capex based on all available pricing information, electricity cost of ¢5/kWh and an 18-month depreciation schedule (for a full treatment of the methodology and all assumptions we direct readers to the appendix of our June report).

I need to stress that ***this is an average figure*** . Not all miners are operating at these assumption levels. Some have it better, some have it worse.

The figure itself means that *if our assumptions are correct*, the average miner, at the network conditions present in June, would run a positive ROI at bitcoin prices above $6,500. The bitcoin price at the time was approximately $8,500.

In our latest report, *under the same exact assumptions*, we estimate the creation cost to now be approximately $6,800, an increase of $300. The current bitcoin exchange price is about $4,000—an entirely different situation altogether.

For reference, we estimate that the opex component of this all-in cost is exactly 50% of the total (at an 18-month depreciation schedule). That means the *market-average cash cost* at our assumptions is approximately $3,400. Pretty close to current bitcoin prices.

So what does that mean?

Essentially one of two things:

1. Either our assumptions are silly, or…
2. Many miners are currently feeling the squeeze, with inefficient mining gear and high-cost electricity miners likely to be forced off the network.

What does the data say?

Interestingly, the data suggests that the truth lies somewhere between the two — cliché, I know. But let's unpack that a little.

After peaking at a 2016-block average of almost 55 Exahash per second (EH/s) in late September, the hashrate has since fallen to approximately 40 EH/s, triggering the largest difficulty decrease in more than five years at the last adjustment.

At the time of writing, Bitcoinwisdom projects the next downwards adjustment to be even larger. The combined back-to-back decrease would be one of the largest in Bitcoin history, certainly the largest since the advent of large-scale professional mining around 2014.

Clearly, some miners are struggling. They have gear that is running below cash cost which means this gear is now being shut off.

It is also possible our market average assumed electricity cost of ¢5/kWh could be too high. Or perhaps our cooling cost assumptions are too high? While we can speculate about this, that is also ultimately all we can do.

An Important Note: Price and Hashrate Dynamics

Bitcoin is structured such that the hashrate follows price, slightly modulated by increases in gear efficiency. When the price increases, the hashrate increases, and when the price decreases the hashrate decreases.

Mining cost will always tend towards the price of bitcoin minus a narrow competitive margin. However, these dynamics are not instant, and there is an asymmetrical delay in the trailing effects.

Like any other capital-driven industry, the delay in the upwards drag results from the time difference between making an investment decision and when the gear is actually switched on.

For most players in bitcoin mining, this is on the order of months and depends on their proximity and relationships with the producers of mining gear.

But even for the producers themselves, there is significant delay. Chips must be ordered from the foundry; units must be assembled, shipped and installed. Only at the end of that process does the hashrate actually increase. In the meantime, the price could have increased even more, and much more rapidly than new gear could have possibly been employed.

The same dynamic does apply in the opposite direction. When cash cost falls below breakeven, there are no barriers preventing miners from immediately pulling the plug on their gear, meaning that mining gear can be shut off immediately in response to falling prices. *(Side bar: the notable exception to this are mining operations that bought fixed supply electricity contracts, thus forcing them to mine until they are insolvent).*

Hashrate will therefore lag price increases on the order of months, but respond much quicker to decreases in price.

"But I thought price follows hashrate?"

It doesn't. And moreover, how could that possibly be the case?

Miners are compensated in bitcoin, but incur costs in their local currency. Under a steady hashrate marketshare, the bitcoin exchange price is directly proportional to their payout.

At average competitive conditions, increasing your hashrate in a falling market will only make you lose money, as mining costs will increase in line with the difficulty increases caused by increasing hashrates.

Only when prices increase can the hashrate increase—in excess of efficiency gains—to compensate for increasing difficulty and costs.

In fact, I can only think of one scenario where the hashrate could act as some sort of 'floor' for prices. This is in the unlikely event where miners *are so well capitalised* that they could refuse to sell their bitcoins at market prices, and cover their costs with liquid capital from their balance sheet while continuing to mine at a loss. You be the judge of that likelihood.

So is the mining industry collapsing now?

No. There is nothing dramatic about what is currently happening. The net effect is that the highest marginal cost producers are booted off the market while the most efficient miners remain.

Difficulty resets to a lower level and the all-in cost of mining falls to a level where it is again right below the price of bitcoin.

The remaining miners then restart a new competitive cycle—both against each other, and against a new wave of prospective outsiders who believe they can mine profitably under new conditions. Through this process, mining migrates ever closer to the cheapest underlying conditions. It's an incredible spectacle of pure free-market dynamics.

Meanwhile, ***bitcoin issuance effectively remains the same*** . No fewer bitcoins are created, just as no more bitcoins were created during the period of price growth (with the exception of single-digit perturbations caused by the relative over- or-underperformance caused by the growth or shrinkage happening between difficulty resets).

This is a defining characteristic of bitcoin mining and sets it apart from all other commodities—it is also a fundamental driver of volatility.

Unlike other commodities where the output is modulated by price, the bitcoin issuance* cannot change *.

If the price of gold increases, production will increase until the marginal production cost again equals the market price (minus transport costs, which for bitcoin, are negligible compared to physical commodities). If the price falls, production will decrease until the same condition holds. This dampens volatility by increasing supply in rising price markets and reducing supply in falling markets.

No such effect in Bitcoin. Issuance is predefined and no market dynamics can significantly influence it.

So what happens next?

As price does its thing, hashrate will follow and settle into whatever new market conditions are in stall. Old, inefficient gear and high-cost producers are out; and until price increases again, the hashrate can only increase by miners lowering their opex. They can do this by sourcing cheaper electricity, installing more efficient mining equipment or generally cutting costs.

There is no such risk of mining collapse or any other click-bait nonsense you might read out there. ***The dynamic difficulty adjustment takes care of that by regularly resetting the difficulty so mining costs fall in tune with bitcoin's price.***

For such a collapse to occur, the bitcoin price would need to immediately plunge to near zero; thus triggering virtually the entire mining network to shut down; and therefore preventing the requisite blocks to reach the next difficulty reset from being mined for months or even years.

Perhaps possible, but not a likely scenario in our view.

Bitcoin isn't dead. Not this time nor the 326 times before.

Now I suggest you have a look at our report, ***some of our other findings might totally surprise you*** (see what I did there?).

Right: Link to report

CoinShares Research
The Bitcoin Mining Network
- Trends, Marginal Creation Cost, Electricity Consumption & Sources

CoinShares
@CoinSharesCo

The truth about bitcoin mining may surprise you: coinshares.co.uk/bitcoin-mining…

♡ 60 1:17 PM - Nov 28, 2018

35 people are talking about this

Disclaimer

Please note that this Blog Post is provided on the basis that the recipient accepts the following conditions relating to the provision of the same (including on behalf of their respective organisation).

This Blog Post does not contain or purport to be, financial promotion(s) of any kind.

This Blog Post does not contain reference to any of the investment products or services currently offered by members of the CoinShares Group.

Digital assets and related technologies can be extremely complicated. The digital sector has spawned concepts and nomenclature much of which is novel and can be difficult for even technically savvy individuals to thoroughly comprehend. The sector also evolves rapidly.

With increasing media attention on digital assets and related technologies, many of the concepts associated therewith (and the terms used to encapsulate them) are more likely to be encountered outside of the digital space. Although a term may become relatively well-known and in a relatively short timeframe, there is a danger that misunderstandings and misconceptions can take root relating to precisely what the concept behind the given term is.

The purpose of this Blog Post is to provide objective, educational and interesting commentary. This Blog Post is not directed at any particular person or group of persons. Although produced with reasonable care and skill, no representation should be taken as having been given that this Blog Post is an exhaustive analysis of all of the considerations which its subject matter may give rise to. This Blog Post fairly represents the opinions and sentiments of its author at the date of publishing but it should be noted that such opinions and sentiments may be revised from time

Disclaimer:

Share this journal early and often. Engage the authors and tell them what you think. We sharpen our position through discourse and debate.

DYOR | BTFD | HODL

Thanks for your attention and support. I appreciate your feedback and hope you enjoy this publication.

- @_joerodgers