Blockchain Basics
A Dive into Ethereum and Bitcoin

Stephen M. Lee

smlee@memphis.edu

Abstract. Public blockchains and their corresponding cryptocurrencies have captured the imagination of many in recent years. Since the release of Bitcoin, several other projects, most notably one called Ethereum, have extended the key insights to allow code of arbitrary complexity to run securely on these public networks. This paper describes the main problem that they seek to solve and then proceeds to compare and contrast the implementations of Bitcoin and Ethereum. Finally, I suggest an evaluation of the Ethereum blockchain as a distributed system based on how well it makes resources available and provides user transparency, openness, and scalability.

1 Introduction

The original Bitcoin paper [8] was released in 2008 under the pseudonym Satoshi Nakamoto. Here, the author introduces a novel way of enabling digital, peer-to-peer financial transactions without double-spending attacks.[1] This is the major contribution of blockchain technology. Traditionally, this problem is avoided through the use of banks and other intermediaries (i.e. Paypal, Venmo) who ensure that users don't spend more than they actually have. With Bitcoin, however, double spending is prevented through a clever combination of cryptography and game theory, opening up a new world of opportunities for peer-to-peer digital transactions. Opportunity has a cost, however. While public blockchains can truly allow for secure peer-to-peer digital transactions without any mediating party, they are expensive: both computationally and financially. As will be discussed in more detail, there is a necessary limit to the number of transactions that can be included in each block – this condition can create a "bidding war" over transaction fees to get your transaction counted in a reasonable time frame. Further, whereas in most traditional distributed systems, adding more computer resources can increase performance and throughput, blockchains add security when more computing power joins the network[2]. This is to say that

[1] A double-spend attack is when some attacker, Bob, spends the same units of a digital currency with multiple people, Alice and Cate. For example, Bob sends an electronic message to Alice that says he is giving her 5 dollars, but Bob sneakily sends the same exact message to Cate. In this case, Bob is telling people that he has spent 10 unique dollars, but without someone checking, he has actually only spent 5.

[2] Suppliers of this computing power are called miners in the traditional Proof-of-Work consensus algorithms. These will be discussed in more detail below.

there can be no increase in transaction throughput as more computers join the network: it's baked into the protocol.

There are efforts to scale transaction throughput with new consensus mechanisms, however, the leading alternatives are still not in production. Additionally, blockchains have taken various other forms besides Bitcoin. Most notably, and the main focus of this paper, is Ethereum. This network was first introduced by a University of Waterloo student named Vitalik Buterin as a whitepaper[1] and officially went live in 2015. Here, he extended some of they core Bitcoin concepts to allow for more expressive scripting languages. While Bitcoin's language is a simple stack-based language that isn't touring complete, Ethereum does in fact offer touring-complete expressions. This change allows for creating secure "smart-contracts" that can process arbitrarily complex transactions.[3] Again though, opportunity comes at a cost. This more expressive blockchain has led to numerous security issues: bugs can now take the form of actual code bugs or incentive bugs – a bug that arises through poor mechanism design that allows an attacker to game your smart-contract system. Further, since the code released onto a blockchain is immutable, patching bugs is a challenge.

With all of this in mind, blockchain technology does seem to offer a truly unique and valuable asset: secure peer-to-peer digital transactions without the need for an intermediary. However, until the cost of using these networks decrease, the status-quo is often cheaper.

2 Related Work

After the release of the original Bitcoin paper, not much additional work was done until the Ethereum white and subsequent yellow [12] papers. The Ethereum white paper laid out the conceptual framework for a more expressive blockchain and, notably, introduced the concept of "gas". In practice, paying for gas means that a user must pay a transaction fee on the Ethereum network. This major new piece in the blockchain is added to guarantee that a given smart contract will eventually halt and not send the mining node into an infinite loop. Further, the amount of gas that a user spends on a transaction is directly related to the number of computational steps that the program takes – in this way, each opcode call to the Ethereum Virtual Machine has a corresponding amount of gas that it uses. The analogy here is intended to be just like buying gas for a car: one gallon will not take you the same distance going uphill as it would going downhill.

The Ethereum yellow papers spell out actual implementation details for each version of Ethereum, but does not necessarily constitute formal research.

More recently, however, academic researchers have began contributing. This includes analyzing blockchain in asynchronous networks [9], research into various other byzantine fault-tolerant frameworks [10] [11], and work that injects tra-

[3] For example, on the Ethereum network you could create a smart-contract system that allows a hotel to manage and rent it's room inventory. In this way, in Ethereum you can program smart-contracts that have a very heavy object-oriented feel.

ditional database and distributed computing ideas to better handle blockchain information [7].

While work in this field is nascent, the large majority of research has come from the open source community and the Ethereum Foundation. In particular, several alternatives to the Proof-of-Work algorithm – most notably Proof-of-Stake – are in research and development phases. Further, scalability research also includes attempts to shard a blockchain across nodes and to build "side-chain" options that don't directly record to the main network, but periodically write a Merkle Tree hash of all their previous activity to the main chain.

3 Design - Block Basics

For the scope of this paper, I will focus on the Bitcoin and Ethereum implementations of a public blockchain. Since they both have slightly different levels of expressiveness, there are some subtle, and some profound differences. In general, this section will proceed by describing how both systems track account balances, what fields are included in a single transaction, and how transactions are included into a block.

Accounts and Balances

On a public blockchain, coin balances are associated with a public account number, which, for Ethereum, is represented has 160 bit hexadecimal number.[4] This public account is built using public/private key cryptography, and is necessary to provide digital signatures for each transaction. After generating a public/private key pair, several necessary methods are necessary: 1) it must be possible to sign a message with your private key, and 2) it must be easy for someone else to verify the message using your public key. All told, the API in pseudocode has the following form:

```
1. function generateKeys(keySize)
     returns (privateKey, publicKey)

2. function sign(message, privateKey)
     returns (signedMessage)

3. function verify(publicKey, message, signedMessage)
     returns (boolean)
```

This set of methods allows a user to uniquely sign a transaction while enabling any other user on the network to verify that the transaction is in fact valid using only the senders public address. Both Ethereum and Bitcoin use implementations of an elliptical curve digital signature algorithm to generate

[4] For example, one of my public Ethereum addresses is 0x130B6195560119579e69b2c787C6D14906Ff91f9.

these keys, however, the details are outside the scope of this paper and may well be taken as cryptographic magic.

While similar so far, Ethereum and Bitcoin differ substantially on how they handle current account balances. For Ethereum, balances are stored in a "global state", where each account address is mapped to its current account state using a Merkel Patricia tree.[5] Thus, each new transaction updates the relevant account address mapping. Note that in addition to keeping track of standard balances of Ether, the increased scripting complexity of Ethereum also allows for storing other bits that can correspond to balances of transferable tokens, reservations, digital collectibles, or nearly any other state that can be conceived of with traditional object-oriented programming. This difference allows Ethereum to store states of arbitrary complexity. Officially, the state of an address in Ethereum is given by the following fields:

- **nonce**: This is a positive integer value that denotes the number of transactions previously sent from the account address. It is through incrementing this value that double-spending is prevented.
- **balance**: A positive integer value that records the number of Wei stored in the account, where 1 Ether equals 10^{18} Wei. Wei is the smallest possible unit of exchange while Ether is the unit often denoted for trade or exchange.
- **storageRoot**: A 256 bit value that maps to the accounts aforementioned Merkel Patricia tree.
- **codeHash**: A hash of the account's code that will run when a message is sent to it. This code is created upon construction and cannot be modified. The machine that runs the code is called the Ethereum Virtual Machine (EVM). An example of an account with code is a so-called "smart contract". Each contract can be conceptually thought of as an object in traditional object oriented design, and each contract is stored latent in the EVM until an external message calls it. In this way, every time you hear the term "smart contract", the phrase "stored program" is perhaps more accurate.

On the contrary, Bitcoin keeps track of balances by recording so-called "unspent transaction outputs" (UTXO). For Bitcoin, each transaction consists of "inputs" and "outputs", where the sum of the inputs must equal the sum of the outputs, and further, each input must be a valid UTXO from a previous transaction. After an UTXO is used as input, it is removed from the pool of valid UTXOs.[6] Through these simple interactions, Bitcoin is able to prevent double spending by ensuring that no transaction output is spent twice. We can imagine a transaction on the Bitcoin network visually in Figure 1 below.

[5] See https://ethereum.github.io/yellowpaper/paper.pdf.
[6] For video description, see https://www.coursera.org/learn/cryptocurrency/home/welcome.

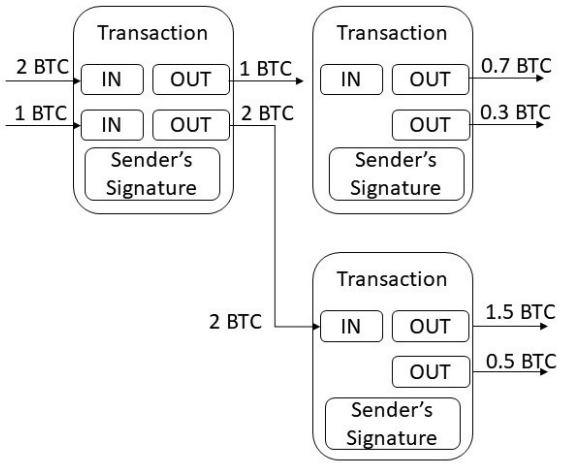

Fig. 1. An example of a transaction on the Bitcoin network.

Transactions

Transactions on both Ethereum and Bitcoin are atomic in that they are all-or-nothing: if something fails, then the transaction is reverted. Further, there are no partial states. That is about the extent of their similarities, however. The figure above shows visually what information is included into each Bitcoin transaction - the unspent output from a previous transaction along with the new outputs to be sent to other accounts. Transaction fees are optionally included by sending more value in transaction inputs than outputs.

In contrast, a transaction in Ethereum feels a bit more intuitive on the surface: an owner sends some optional value (in Wei) to a recipient, and the account state updates the balances accordingly. However, to prevent double-spending (as mentioned above), the sender must also include their current "nonce" value that represents the number of transactions that the account has made previously. Since accounts can be stored programs i.e. smart contracts, each transaction also has an optional field for raw data that can be sent and processed by an account. For example, consider an address that stores complied bytecode that corresponds to the following pseudocode:[7]

[7] This pseudocode example is written using syntax from the Solidity programming language, which compiles into machine code for the Ethereum Virtual Machine (EVM).

```
contract ERC20TokenExample {

  mapping (address => uint256) balances;
  string name = "COMP7212-Token";

  function transfer(address _to, uint256 _value) public returns (bool) {
    /* first make sure sender has enough */
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    return true;
  }

  function balanceOf(address _addr) public returns (uint256) {
    return balances[_addr];
  }

  function mint() public payable returns (bool) {
    address sender = msg.sender;    // address of the calling account
    uint256 value = msg.value;      // value carried with the message
    balances[sender] += value;      // create new 'tokens'
    return true;
  }

  ...
}
```

In the above example, if a sender wants to create tokens for herself, she can create a new transaction that sends some amount of Wei (i.e. the base unit of Ether) to the stored bytecode of the above contract and, importantly, include a data segment that specifically tells the program to run the process using the function "mint()". In this way, a human with an account address can send value (i.e. Wei) to an address that is purely governed by immutable code. To finish up the book-keeping, a clever and slightly devious reader may ask about preventing malicious transactions that include infinite loops. Rephrased, how does Ethereum ensure that each transaction will eventually halt? The answer is to also include the parameters *gasLimit* and *gasPrice*, which together set an upper bound on the number of computational steps that the sender is willing to pay for. These additional parameters do three things: 1) they require that every transaction on the Ethereum network pay a transaction fee, 2) they ensure that even in the case of malicious or faulty code, the transaction will eventually halt, and 3) they make network attacks costly. Summarizing, the following fields are included for each transaction on the Ethereum network:

- **to**: The address of the message recipient.
- **value**: The amount in Wei to transfer to the recipient, if any. This field can be zero if the transaction is merely calling a stored program i.e. smart contract in order to execute some request.

- **nonce**: The number of previous transactions made by the sending address.
- **gasPrice**: The price in Wei the sender is willing to pay per unit of gas.
- **gasLimit**: The maximum number of gas the sender is willing to pay for. Here, each OPCODE that acts on the Ethereum Virtual Machine corresponds with a specific amount of gas. In this way, setting the gas limit is akin to setting the maximum number of computational steps the sender is willing to try. If the transaction completes successfully, and there is gas left over, the sender is refunded the difference.
- **signatureInfo**: The sender also includes three values, r, s, and v, that can uniquely sign for the transaction. These three values are the Ethereum version of the "signedMessage" described in the digital signature API above, and can be uniquely combined with the message (in this case, the message is the hash of the transaction information) to prove that the sender owns that address.
- **data** (optional): A bytes array of technically unlimited size (although practically constrained by the gas limit) that is passed to the receiving address. If the receiving address has code that can read the data, then it is processed and both account state's are updated. If something fails, or the transaction runs out of gas, then the transaction is aborted and both account states are reverted to their initial conditions.

Visually, we can imagine an Ethereum transaction as in Figure 2 below:

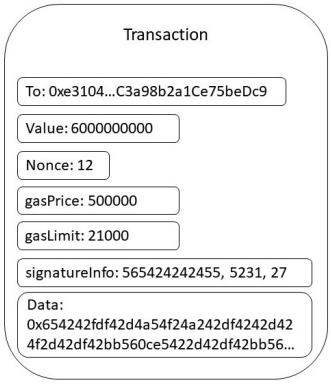

Fig. 2. An example of a transaction on the Ethereum network.

Blocks

Having built a foundation for both Ethereum and Bitcoin with respect to accounts addresses, balances, and transactions, we can now consider how

transactions are processed. Since both networks are public, permissionless, peer-to-peer systems, processing cannot rely on transactions *always* being received by all nodes, nor can it rely on transaction begin received in any sort of sequential order. This is to say that transactions are broadcast to the network, with no real guarantee of processing. Further, since both of these networks are overlaid on existing TCP/IP protocols, they must be resilient to dropped messages. In practice, this may mean that no single node has even heard of every transaction request, however, as long as a majority of the nodes receive an overlapping portion of transaction requests, then those can be processed.

But how exactly are transactions processed? By creating a new "block" that contains a reference to the previous block, some number of processed transactions, and some additional information that makes it difficult to falsify transactions. A simplified example of the minimum necessary block contents is shows in Figure 3 below:

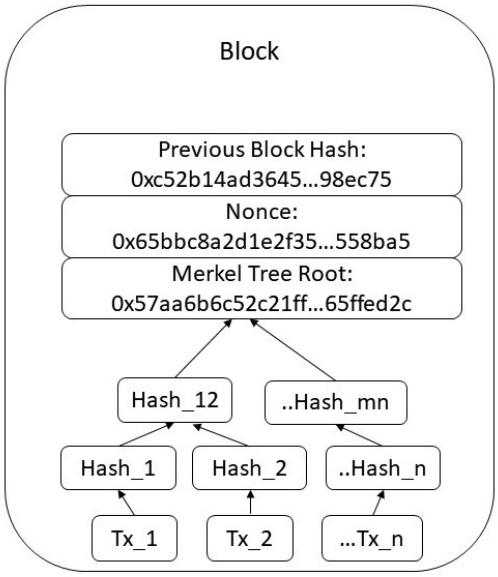

Fig. 3. Example of fields in a Bitcoin block.

Notice that transactions are collapsed into what is called a "Merkel Tree" by taking the cryptographic hash of neighboring transactions until a single root

value is found. This value, called the Merkel Tree Root, uniquely identifies the transaction contents of the block - if any single transaction changes, then the Merkel Tree Root will also change. As we will see in more detail below, this property is crucial to providing transaction security and immutability to each block.

Proof of Work

Note, also, the presence of the nonce in the block header. While it is tempting to think of this in the same way as the nonce in an Ethereum transaction, they are very different. As will be discussed in more detail below, the nonce is a number that provides a "proof of work" for that block. Formally, the nonce is a value that, when combined and cryptographically hashed with the rest of the block header information, will output a value with a certain number of leading zeros that correspond to a level of "block difficultly". The key insight into this nonce value is that there does not exist a faster way of finding a valid nonce than random guessing, yet it is trivial to verify after one has been found. This property stems from the fact that an effective cryptographic hash acts a scrambler of sorts, whereby any input value of arbitrary length is mapped to a unique[8] output of fixed length. For example, using the SHA256 hashing algorithm, we can see the hashed output of several inputs:

SHA256("hashThis") = 0x8171ba87af5f0615160c4d1327d1cae61b3acecdf7e586c0ef10f1b1fdc1b65e
SHA256("HashThis") = 0x9e11568603cae166a8fbf155a37b3b722c1fe94900038335f957a50468c0e1b2
SHA256("HASHTHIS") = 0x6582d74cadd61c52f3235312c859684fd9125db49429f9f242be5964d5fcdc73

The key insight here is that the outputs are unpredictable, even given very subtle changes to the input. This fact enables the possibility of a "game" whereby competitors try to be the first to find a valid nonce to put into the block header. Using the above as an example, a valid nonce could have the following property:

SHA256(blockHeader + nonce) = 0x**0000007**af5f0615160c4...f10f1b1fdc1b65e

Notably, the output hash contains some number of leading zeros. The exact number of zeros is dictated by the protocol, and can update itself over time depending on the number of competitors trying to guess a valid nonce: this is to say that the number of zeros is set to ensure that one block is found, on average, every t units of time. For Bitcoin, this target is one block every 10 minutes, and for Ethereum this is roughly one block every 13 seconds. Implicitly, we recognize that, all things being equal, the more nonce guesses that are made per minute, the faster a valid nonce could be found. Thus, as

[8] Pseudo-unique, more technically. There is always a chance of collision, however, with a good cryptographic hash function, the probability is so small as to be considered impossible.

more competitors enter the game, the puzzle must get appropriately harder in order to maintain a consistent time between blocks. These concepts can be seen visually in the included code demo. It may seem like we made some leaps here, and that's because we did. To understand these ideas more fully, we must consider the following:

- How do you get competitors to agree that a fellow competitors block is actually valid? Why don't they just ignore every other block that they didn't find?
- How do you get competitors to actually participate? If it takes energy to guess a nonce, why would they spend real money to do that?

4 Design - Block Chaining

To summarize up to this point: we have seen how both Bitcoin and Ethereum treat accounts and balances, how they handle transactions between accounts, and how those transactions are processed into blocks that periodically update the global state. Namely, public/private key cryptography is used to sign messages or transactions on behalf of a sender, and, after processing, value can be securely transferred between accounts. This section will proceed by explaining how subsequent blocks are linked or "chained" together.

Hash Pointers

As shown in the section above, blocks process transactions that update the state of the relevant accounts. We can, therefore, consider each block as a state transition in that it only reveals a glimpse of total state. How, then, can we see the big picture? By considering the entire history of blocks *in sequential order*. To do this, we simply include in each block a hash pointer to the previous block and create a sort of append-only linked list. Visually we can see this in Figure 4 below:

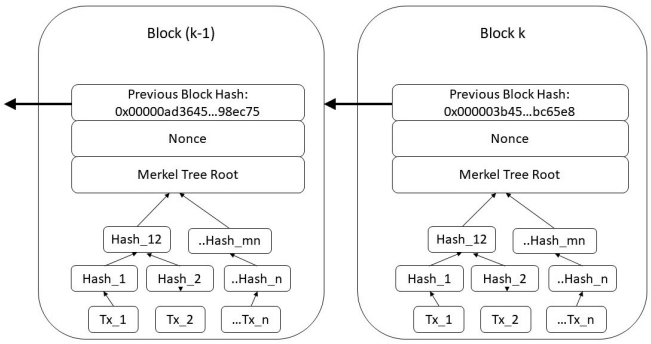

Fig. 4. Hash pointers in a blockchain.

With this in mind, each new block that processes transactions must point to what was previously the last block in the chain. Thus, the entire set of updates to the state are given by the entire blockchain, and together they uniquely define the current state.

Longest Chain

As we already discussed, a public blockchain network must be able to work even if messages or transactions are dropped, and allow peers in the network to come in and out of service as desired. This is to say that the network cannot rely on all peers in the network to see the same exact pool of broadcast transactions. Thus, we can easily imagine cases where several peers find a new block (with different transaction sets) at approximately the same time and broadcast their new node to the network for verification. When this happens, some peers, but not all, could see a block A that is valid, and some other peers could see a block B or block C that are also valid. When this happens, the chain is said to "fork" (more specifically, it creates a soft-fork). In these cases, the protocol uses the following lemma to decide which of these forks is the valid chain:

Lemma: The longest or heaviest chain is always the valid chain.

Here, the main measure for the heaviest chain is the one with the "most" proof-of-work as defined by the total difficulty of solutions. Recall from above that difficulty in proof of work governs how many leading zeros must be present when the nonce value is hashed with the rest of the information in the block. Thus, the chain of blocks with the most difficulty is also the chain that has the most competitors looking for nonce values. We can formalize this slightly by calling the competitors "miners" and describing their effort in terms

of hashes-per-minute.[9] Using these terms, miners will always choose to add a block to the longest (heaviest) chain, and that corresponds to the chain that has the most competition. Visually, we can see the image below that represents a blockchain with a couple small forks that are ultimately abandoned for the longest proof-of-work chain.

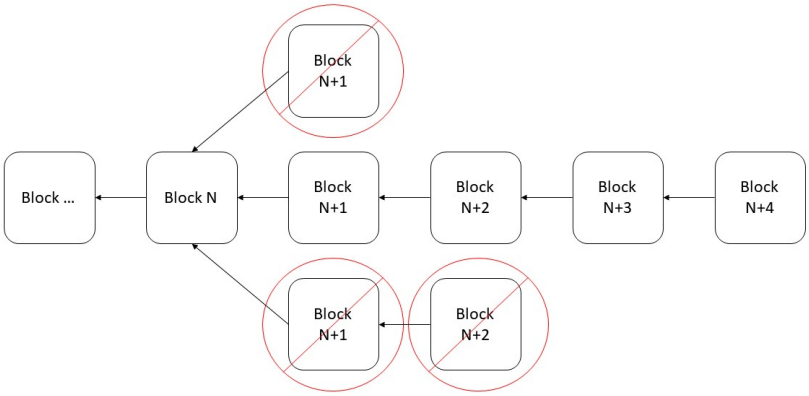

Fig. 5. An example of abandoned forks.

Block Size Limit

Before describing the game played by miners, we need to consider the following case: what is stopping miners from colluding and simply ignoring valid blocks sent from other miners so that they can try and include all of the transactions themselves? This way, they could keep all of the transaction fees for themselves. The answer to this is to impose a limit on the total size of each block. For Bitcoin, this is referred to as the block size limit, and was originally set at 1MB per block. This, however, has been a major source of disagreement. For Ethereum, the corresponding concept is a bit more fluid. Recalling the concept of gas described above,[10] Ethereum sets a maximum gas limit for each

[9] We can think of hashes per minute as guesses per minute. Each miner's computer will rapidly try to find a valid nonce that satisfies the block difficulty. Doing this can use up considerable computational resources, and therefore we can easily imagine that as more miners join the game, the more total guesses that can be tried each minute.

[10] Due to the increased scripting complexity of Ethereum, gas is introduced as a means of ensuring transactions will eventually halt. In short, each OPCODE call performed on the Ethereum Virtual Machine takes an amount of gas that corresponds with its

block. This effectively sets an upper limit on the total state changes from one block to the next.

Ultimately, these upper limits create a situation whereby it is more profitable for miners to accept a valid block and get a "head-start" on looking for the next one than it is for them to stonewall acceptance in hopes of including all of the transactions into their own block.

Block Rewards, Incentives, and Security

The final piece to the puzzle is the miner reward. The Bitcoin protocol set in place a special rule that allows the first transaction in a block to actually create new coins, and further, the miner that discovers the block is able to own those coins. A very similar rule in set for Ethereum. One major difference to note is that the reward for Bitcoin is set to pay a fixed total amount - in practice, this means that the block reward is cut in half approximately every four years until the ecosystem will be forced to rely entirely on transaction fees.[11]

This reward scheme creates a game for miners: be the first to find a block with a valid proof-of-work and pay yourself a reward. Additionally, the nature of proof-of-work as explored above ensures that current blocks are valid (otherwise the fellow miners will not accept the block), and that previous blocks cannot (feasibly) be rewritten or edited by a malicious actor.[12] We can see this is the case by imagining that a malicious miner wanted to attack the network by sneaking a fake transaction into an old block. To do this convincingly, they would not only need to find a new valid nonce for that block, but also find a new valid nonce for each subsequent block - and do all of that before any other miner in the network found a single valid nonce to add to the longest chain.

---

computing complexity. So the gasLimit can essentially be thought of as paying for a maximum number of computational steps for your transaction.

[11] There are some interesting economics that arise with this implementation decision, and, in short, they are not encouraging. Basically, the lack of a block reward with create a congested queuing game in which only transactions with a "sufficiently high" transaction fee will be processed. This condition necessitates the threat of a long enough delay such that a user will feel the need to "bribe" the miners to include their transaction. If there is no delay for processing transactions, then users will not pay a fee, and miners will not spend their energy to mine blocks as it will be unprofitable. For more details see this working paper: https://www.dropbox.com/s/siidwx99dwgxu9e/Huberman%2CLeshno%2CMoallemi-Monopoly_without_Monopolist%3BEconomic_Analysis_of_Bitcoin.pdf?dl=0

[12] There of course are attacks, both simple and sophisticated that can be done on the network. The most obvious is dubbed a 51% attack and basically involves one miner or group of colluding miners to control a majority of the hash power i.e. they must be able to make more guesses than the rest of the network combined. In this case, they can, on average, confidently generate a new block before the rest of the network. This allows them, in theory, to not only find the new block, but also go back and update old block with fake transactions.

In other words, by including the root of the Merkel Tree, and defining some difficulty standard for the proof-of-work, any attack must redo the work. In more concrete terms, if a miner wanted to slip a fake transaction into the previous block, they would need to find two valid nonce values before anyone else finds one valid nonce. If they wanted to fake a transaction into the k^{th} previous block, they would need to find $k + 1$ valid nonces before any other miner found a single valid nonce. It is this property that leads people to describe the blockchain as immutable or, more accurately, tamper-evident.

5 Design - Scaling

As described above, there are throughput limitations for both Bitcoin and Ethereum that are written into the protocols. Namely, each block has a limit to the number of transactions that can be included, and additionally, the proof-of-work difficulty is set such that, on average, blocks are not found "too often".[13] That being said, the Ethereum Foundation is leading an effort to research solutions to this scalability and usability problem.

Limitations to Proof of Work

The original proof-of-work as laid out in the original Bitcoin whitpaper requires a tremendous amount of duplicated effort: essentially, some can argue, the consensus game is to focus computation power on solving a very difficult, but ultimately arbitrary problem. As a result, mining Bitcoin currently uses about as much energy as the country of Austria.[14] Further, special dedicated hardware is now the standard in Bitcoin mining, as they can check hash values faster than a GPU, making it only feasible for wealthy individuals who can afford to have the hardware manufactured. Notably, Ethereum does not have this issue as their implementation of proof-of-work is considered "memory hard",[15] and therefore, miners can feasibly compete using state of the art GPUs.

Other Consensus Algorithms

The leading alternative to proof-of-work is so-called proof-of-stake. According to the official Ethereum GitHub account, proof-of-stake is, "a category of consensus algorithms for public blockchains that depend on a validator's economic stake in the network".[16] Since this removes the costly mining dependency of finding a valid proof-of-work, many believe this is the leading

[13] For Ethereum, this is about one block every 12 seconds, and for Bitcoin, this is about one block every 10 minutes.
[14] See https://digiconomist.net/bitcoin-energy-consumption
[15] For brevity I'm intentionally waving my hands about this. For more information, see https://github.com/ethereum/wiki/wiki/Mining.
[16] See https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs.

direction for the future of public blockchains. Importantly, if a validator is found to be dishonest, the protocol can internally penalize them by confiscating the staked coins: this is another major advantage of these consensus algorithms. In proof-of-work, if a malicious miner acquired enough hardware to cheat the network, the network can only fork away from that chain, but it cannot feasibly confiscate the miner's hardware, leaving open the possibility of another attack.

The first major implementation of proof-of-stake is scheduled for early 2019 in the "Metropolis" update to Ethereum, although, this is expected to be a hybrid system that combines both proof-of-work and proof-of-stake until the latter is sufficiently stable. While researchers are looking for other, less energy intensive, consensus algorithms that are Byzantine Fault Tolerant (BFT), academic work is largely limited to survey papers as in [6] and [5].

Sharding

Without going into too many details, the idea with sharding a blockchain network is to allow only a subset of nodes to verify each transaction - rather than requiring that every node be able to process and verify every transaction.[17] While research into this area is still nascent, one such proposal is to split the shards into static, domain-based shards as in [3]. Toward this end, there is not currently any working implementation of a sharded blockchain network.

Side-Chains

One of the most promising areas of scalability is to process a bulk of transactions outside of the main network, and only periodically write the Merkel Tree root as a checkpoint of activity. One such idea is to create so-called "plasma chains", as introduced in [2]. The analogy for this is akin to using a bar tab: if I go in and make purchases, rather than charge me as I go (which is inefficient for a busy bartender), they keep my tab, and, when I'm done, I pay for the final sum. Challenges in this area are primarily focused around proving malicious behavior and subsequently exiting a side-chain without loss of funds.

Analysis and Conclusion

Blockchain offers a truly new possibility: sending secure peer-to-peer digital transactions without the need for a trusted intermediary. In this sense, it can be thought of as a distributed system for securely and immutably tracking value as it changes hands. Keeping the four goals of distributed systems in mind, we can evaluate public blockchains, as they currently exist:

[17] For more reading on this, please visit https://github.com/ethereum/wiki/wiki/Sharding-FAQs.

- **Make resources available** (4/10): Public blockchains provide access (albeit limited by some standards) to processing, memory, and storage. In fact, the key advantage is as a "tamper-evident" ledger for storage, and as a secure means of processing updates (in the form of transactions) to that ledger. With this in mind, we can also consider trust and security as key resources that are made available.
- **Transparency** (2/10): Transparency of Ethereum and Bitcoin are both very low - using either system is bulky when compared to the current state of the art with Financial Tech (for example, the ease of using ApplePay). With so many of the core implementation details still being actively researched, transparency will likely take a while to improve to a point of mass adoption - if it ever does.
- **Openness** (9/10): Due to the open source nature of development, these systems are very open with respect to interoperability, portability, and extensibility. Many developers from all over the world contribute to research, core development, and application development for these networks.
- **Scalability** (3/10): While both Bitcoin and Ethereum are in use across the world (i.e. geographically scalable), they have very limited throughput, and are consequently a relatively expensive way of making small value transactions. One interesting thing to note is the concept of "blockchain governance". Due to the peer-to-peer security offered, many are interested in creating so-called "Distributed Autonomous Organizations (DAOs)". Thus, while in it's infancy, administrative scaling could be an interesting area to watch.

With this in mind, the wave of interest in distributed cryptocurrencies may sometimes miss the important point: the underlying public blockchain technology that enables the currencies to exist. From an application standpoint, potential uses include any system where trusted intermediation is necessary, expensive, and can be automated with code. This is to say that a public blockchain system is not as helpful if parties all trust each other; if human or company intermediation is easy and inexpensive; and finally, you shouldn't use a blockchain if ambiguity is necessary. That being said, possible applications include shipment tracking in a large supply chain, renting out hotel room access without using the likes of Expedia and Priceline, and enforcing simple legal agreements in freelance work.

References

1. Buterin, Vitalik. “A Next-Generation Smart Contract and Decentralized Application Platform-Ethereum Whitepaper”, 2014.”
2. Buterin V., Poon J.: “Plasma: Scalable Autonomous Smart Contracts” (2017)
3. H. Yoo, J. Yim and S. Kim : “The Blockchain for Domain Based Static Sharding,” 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), New York, NY, 2018, pp. 1689-1692.
4. Hull, Richard. “Blockchain: Distributed Event-based Processing in a Data-Centric World.” Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems. ACM, 2017.
5. L. M. Bach, B. Mihaljevic, M. Zagar, “Comparative analysis of blockchain consensus algorithms”, Information and Communication Technology Electronics and Microelectronics (MIPRO) 2018 41st International Convention on, pp. 1545-1550, 2018.
6. L. S. Sankar, M. Sindhu and M. Sethumadhavan, “Survey of consensus protocols on blockchain applications,” 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, 2017, pp. 1-5.
7. Maiyya, Sujaya, et al. “Database and distributed computing fundamentals for scalable, fault-tolerant, and consistent maintenance of blockchains.” Proceedings of the VLDB Endowment 11.12 (2018): 2098-2101.
8. Nakamoto S. : Bitcoin: A Peer-to-Peer Electronic Cash System. (2008)
9. Pass R., Seeman L., Shelat A.: Analysis of the Blockchain Protocol in Asynchronous Networks. Advances in Cryptology – EUROCRYPT 2017. Lecture Notes in Computer Science, vol 10211. (2017).
10. Sousa, Joao, Alysson Bessani, and Marko Vukolic. ”A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform.” 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2018.
11. Vukolić, Marko. “The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication.” International Workshop on Open Problems in Network Security. Springer, Cham, 2015.
12. Wood, Gavin. “Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper 151 (2014).” (2014).