The Economics of FinTech

Steven Kou

Department of Finance

Questrom School of Business

Boston University

This version Spring 2020

Chapter 2

Introduction to Blockchains and Cryptocurrencies

.

2.1 Blockchains, Smart Contracts, and Cryptocurrencies

There are many books and articles on blockchains, smart contracts and cryptocurrencies. Why do we need another survey? There are at least two reasons for writing this chapter. First, many of the people who write about blockchains and cryptocurrencies are computer scientists, and they might write in a style that is difficult for outside people, e.g. economists, to understand. Of course, there are many articles and books in the general media that are readable to economists, but, arguably, some of the authors knows very little about blockchains, as they perhaps have never written a single line of blockchain programming code. For example, had they done the programming, instead of making a statement such as public blockchains will survive but not crypocurrencies, they should have realized that running a code on a public blockchain most likely requires a certain amount of the cryptocurrency associated with the blockchain (e.g. the "gas" in the Etherium network in terms of Ether) to pay for every operations within the code, such as plus, minus, multiplication, sorting, etc. Therefore, public blockchains (that supports smart contracts) and cryptocurrencies (that pay for programming codes) go together: Either they will go burst together, or they will all survive together.

Second, we focus on problems that may be of interests to data scientists and economists. For example, very few survey papers discuss the detail connection

between the gambler's ruin problem, which is a classical problem in applied probability, and the solution to the prevention of the double spending problem, which is central to design blockchains; but we do spend a section on this.

There are two types of blockchains: (1) Permissoned blockchains (including private and semi-private blockchains), which are not open to the general public. Examples include Hyperledger, Ripple, R3 Corda and Quorum (by JP Morgan). (2) Permissionless(or public) blockchains, which are for public use. Examples include Bitcoin and Etherium networks. Cryptocurrencies are not essential for permissioned blockchains, but are crucial for permissionless (public) blockchains.

For simplicity, we focus only on permissionless (public) blockchains. Consequently, we shall omit the words permissionless and public if not confusion arises.

2.1.1 The Problem of Double Spending

There are two main difficulties to create anonymous digital currency. First, as people can easily copy music and movie files, how to prevent people from copying digital currency token electronically? Secondly, how to prevent the double spending problem in which a single digital currency token can be spent more than once to settle liabilities. The first problem can be solved by using crystallography. The second problem can be solved easily if there is a central party to verify transactions. The first centralized anonymous digital currency was proposed in Chaum (1983), which requires a central party such as a bank; in 1995, he implemented the idea commercially as Digicash.

However, how to create a decentralized digital currency was a long standing open problem for many years, because without a central party it is much more difficult to solve the double spending problem, as we shall explain in details in the next section.

Double spending means the same money is spent twice. For example, after B delivers the good or service to A, A may revise the original transaction record, and claims that the money is paid to account A' (which could just be another account owned by A) rather than to B. A traditional way to solve this problem is to have a central authority, e.g. a bank, to verify that the check that A writes to B is legitimate in the sense that A has sufficient money in the account to pay to B, and record this transaction in the account of A. However, this becomes a very difficult problem in a distributed network without a central authority. In fact, this is a special case of reaching consensus in a distributed network, which has shown to be impossible under traditional deterministic consensus framework; see, e.g.,the Byzantine Generals Problem in Lamport et al. (1982) and the impossibility result in in Fischer et al. (1985). The problem can be solved if one can accept a probabilistic solution (instead of a deterministic solution); see Ben-Or (1983), Micali and Feldman (1997), Gilad et al. (2017).

2.1.2 The Basic Idea of Nakamoto

A revolution started in Nakamoto (2008) was that the two problems may be solved by using blockchains and "mining" (a form proof-of-work).

The concept of blockchains was first proposed in Haber and Stornetta (1991), which originates the concept of using cryptographic hash functions to link and timestamp blocks of data, whence the name "blockchain." A blockchain is a special case of "distributed ledger," where the word distributed means that there may be no central party and each node can create transactions individually. The word ledger means an accounting book or, rather more accurately, a database. More precisely, a blockchain (whether public or private) is a decentralized (peer to peer) distributed database that is used to record all transactions which can be viewed by every users, thus allowing people to audit transactions in a transparent and inexpensive way. Just like Internet is a collection of individual computers exchanging information via the web, a blockchain network consists of individual nodes, where each node can contain a full copy of blockchain and may even run a programming. The records cannot be easily altered retroactively; in fact, any alternation of the records will trigger the alteration of all subsequent blocks. The concept of proof-of-work was suggested in Dwork and Naor (1992) to combat junk emails and was implemented independently and commercially as "Hashcash" in 1997 (see Back (2002)).

The innovation in Nakamoto (2008) is twofold. First, the Nakamoto's innovation brought together the above two concepts together to solve the double spending problem for permissionless public blockchains. Secondly, the proposed concept was implemented in 2009 as the core component of the first cryptocurrency, Bitcoin. In particular, in the next section we will see that unless there is a collusion of significant users of the network it is statistically impossible to change the records; furthermore, a blockchain confirms, with very high probability, that each unit of value was transferred only once, solving the double spending problem in the absence of a central intermediary.

In a proof-of-work based public blockchain, the next block is chosen according to the winner of the mining effort. In addition to proof-of work, there is another way to reach distributed consensus called proof-of-stake, in which the creator of the next block is chosen via various combinations of random selection, wealth and seniority. In this chapter we shall focus on proof-of-work.

2.1.3 Smart Contracts

The name and the concept of "smart contracts" were first proposed in Szabo (1994), which are contracts that can be executed online automatically without a central authority. Some limited smart contracts can be executed with non Turing-complete script code in the Bitcoin network; non-Turing complete implies, for example, one cannot be sure about the existence of an infinite loop within a code unless the coding is executed. In general, Turing-complete languages can be applied more broadly, while non Turing-complete languages have only specific uses.

Another breakthrough came in late 2013 when Vitalik Buterin (born in 1994) extends the idea of Bitcoin by introducing "gas" to create the Ethereum network where people can write smart contracts more generally. The Ethereum network supports nearly Turing-complete languages, such as Solidity. The cryptocurrency generated and circulated on the Ethereum platform is Ether (with the trading symbol ETH). The gas in the Etherium network refers to charges in terms of Ether for every operations within the code, such as plus, minus, multiplication, sorting, etc. For example, if one creates an infinite loop by using the Solidity programming language in the Ethereum network, the loop will die out eventually as it will run out of gas. This is a very important technological advance, as many types of clerk works (such as public notary, import and export paper works, certain legal and accounting documentations) can be programmed as smart contracts, which can be tracked and executed automatically on the Ethereum network.

For Internet, developers can use programming lauguages such as JavaScript to design web pages, while consumers can use web browsers such as Chrome to interact with the web. Similarly, developers can use the Solidity programming language to write smart contracts on the Ethereum network, and use web3.js to create an interface to access and to run smart contracts; consumers can either use Metamask (an Chrome add-on) or Mist Browser to interact with the Ethereum network. We will discuss Solidity programming later.

2.1.4 Programming Smart Contracts

To see how effective a smart contract can be implemented via the Ethereum network, consider a simple example of running a lottery or raffle event. Every one from a group of people each contributes $1 to a central account. Then the manager of the event will randomly draw a winner, and will give all the collected money in the central account to the winner. One faces several obstacles, if one wants to implement this in traditional ways. First, how to find a trustful central account, so that people can be certain that the money in the account will not stolen or disappear for other unwanted purposes. Second, how to make sure that the winner will indeed get the money paid, as it is stated in the rule. Third, how to do the computer programming to link each person's account to the central account. One can easily image the amount of legal/clerk efforts and thousands of lines of computer codes that need to make this happen.

However, using the Solidity programming, the most popular Ethereum programming language, all these can be achieved in *less than 30 lines of coding*. More precisely, the Solidity programming just instructs each user account to contribute a fixed unit of ETH to the central account; then, the programming, upon the command of the manager (who has no control of the central account), randomly draws a user account and wires all the money (in terms of ETH) in the central account to the lucky user account. Once the contract is written and implemented, the contract is strictly enforced, as no one can change it in a probabilistic sense. Furthermore, all the accounts and users involved can be anonymous, and all transaction records are stored in the network that can be

retrieved by the general public.

Indeed, one can do even better things using smart contracts in the Ethereum blockchain network. For example, one can implement a smart contract for a crowdfunding event, e.g. by issuing initial coin offering (ICO) in the network. More precisely, an inventor of a new type of wheel can ask investors to buy some tokens from the inventor; in return, holders of the tokens can share the profits from selling of new wheels. Furthermore, rules can be written in the smart contract so that some major decisions will be decided by the vote from the token holders, with the vote being conducted on the blockchain. Amazingly, the core part of the code of this type of smart contract can be less than 100 lines!

In short, a smart contract is an automatic way to store data and transfer money according to a set of prescribed rules without a central intermediary. The money associated with the transfers are cryptocurrencies and the shares associated with the ownerships are called tokens (in analogy with stocks). Currently, there are over 1000 crytocurrencies and even more tokens traded in exchanges; see, e.g, the list on coinmarketcap.com. Some of crypocurrencies are based on public blockchains, such as Bitcoin and Ether, and others on private blockchains, such as Ripple. In fact, one can buy crytocurrencies from online exchanges or at ATM machines worldwide, just like buying standard financial securities or foreign currencies. All cryptocurrencies share three important features. First, a payment from one user to another is processed in a decentralized way without any intermediary. Second, all transaction records are stored in the networks and can be viewed by every user. Third, they allow anonymous payments.

We will soon how to achieve these three features via blockchains in the next section, and how to solve the double spending problem with these features. However, before we do that we need to discuss the advantages and disadvantages of cryptocurrencies.

2.1.5 Pros and Cons of Cryptocurrencies

There are many papers and media articles discussed pros and cons of cryptocurrencies. Using cryptocurrencies as a payment method has several benefits. First, as pointed out in Harvey (2016), the core innovation of cryptocurrencies like Bitcoin is the ability to publicly verify ownership, instantly transfer the ownership, and to do that without any trusted intermediary. Therefore, cryptocurrencies reduce the cost of transferring ownership. Also, the blockchain technology makes the ledger easy to maintain, reduces the cost of networking (see Catalini and Gans (2016)), and is robust against attackers. The distributed ledger can result in a fast settlement that reduces counterparty risk and improves market quality (see Khapko and Zoican (2018)). Furthermore, since the transaction is recorded to the blockchain anonymously, cryptocurrencies help to protect the privacy of their users. The underlying technology of cryptocurrencies may one day strengthen the menu of electronic payments options, while the use of paper currency is phased out (see Rogoff (2015)).

There are also some criticisms of cryptocurrencies. First, a payment system

with cryptocurrencies lacks a key feature, the confidence that one can get his money back if he is not satisfied with the goods he purchased. As pointed out in Grinberg (2011), Bitcoin is unlikely to play an important role in the traditional e-commerce market, since consumers typically do not care about the anonymity that Bitcoin provides; instead, they prefer a currency they are familiar with for most goods and services, and they want fraud protection. Second, unlike government-backed systems, Bitcoin does not have identity verification, audit standards, or an investigation system in case something bad happens. For instance, one may lose all his deposit in cryptocurrencies should his password get stolen, and there is no remedy. Furthermore, Blockchain systems are not as trustworthy as they seem to be. Without an intermediate, individuals are responsible for their own security precautions. Finally, it is difficult to value cryptocurrencies like Bitcoin.

Here we want to point out that despite significant drawbacks of cryptocurrencies, it is generally agreed that the blockchain technologies are here to stay. However, blockchain technologies automatically generate cryptocurrencies for the purpose of charging the services provided by the system (such as fees, called "gas," incurred by every operations in all programming codes which are run on the Ethereum network), crediting essential services to the system (such as the verification services provided by miners), and of exchanging credits for services. Therefore, cryptocurrencies will not disappear as long as blockchain technologies exist. Thus, designing suitable stable coins is essential for the blockchain system to perform financial functions efficiently and for doing asset allocation across different cryptocurrencies generated by different blockchain systems. In this regard, governments can provide an essential role in helping design a better financial ecosystem of blockchains.

2.2 Solving the Double Spending Problem

To begin our discussion, image that Account A wants to send 0.1 ETH to Account B. After Account A hits the send button in Metamask or Mist Browser (after verifying the password), a record of transaction is created, which consists of the serial number of the transaction, to and from accounts numbers, the amount of money, and a digital signature based on Account A's private key showing that the transaction is indeed a genuine transaction initiated by Account A. Now things get more interesting, the system will wait for a period of verification called "mining", ranging from a few minutes to a few hours. The mining part is crucial for the blockchain to solve the double spending problem and to verify the ETH money is real.

2.2.1 Basic Idea

The main idea of the Bitcoin blockchain to solve the double spending problem is to combine the concept of the proof-of-work, economic incentives, and a probabilistic solution. The key idea goes as follows.

(1) After Account A sends some money to Account B, the transaction is added to a mempool of all unconfirmed transactions, waiting for miners to confirm.

(2) A miner can choose any transactions (typically sorted by the transaction fees the originators of the transactions are willing to pay) to form a block (about the size of 1MB)[1].

(3) A miner first verifies that all the transactions within the block are consistent with the existing confirmed blocks in the blockchain, and then try to add the new block to the existing blockchain.

(4) To add the new block, the miner needs to show a proof-of-work by solving a (somewhat random) cryptography puzzle based on a hash function. This takes a random time τ to solve the puzzle. Whoever solves the puzzle first will get a (large) mining reward (currently 12.5 Bitcoins in 2018)[2] and the corresponding block will be added to the blockchain. In addition, the successful miner will get all the transaction fees in the block (like a tip). We will discuss the details of the hash function and the cryptography puzzle in the next subsection.

There are several rules of the Bitcoin blockchains: (a) By design, the difficulty of the puzzle is adjusted periodically so that the expectation of τ is about 10 minutes for Bitcoin network[3] (and about 15 seconds for the Ethereum network). More precisely, in the Bitcoin network the rate is recalculated every 2,016 blocks to a value, so that the previous 2,016 (=6*14*24) blocks would have been generated in exactly 14 days had everyone been mining at this difficulty. (b) The longest chain, which has the most mining work devoted to it, will be the "official" chain, called active chain. (c) The only way to generate Bitcoins is through mining; the mining reward was 50 Bitcoins per block in the beginning, and gets halved every 210,000 blocks (about every 4 years, as $210,000/(6 \times 24 \times 365) = 3.996$). (d) The total supply of the Bitcoins is fixed at 21 million. Indeed, no new Bitcoin will be produced when the unit of Bitcoin reward is less than 10^{-8} (around the year 2140), after which the only mining rewards are transaction costs[4]. (f) The hash function has the property that any tiny change to the original input of transactions will lead to a big change to the output. Thus, the only way to change the original transaction is to mining a new competing block (called a folk) that includes the modified transaction, and hope that the competing block will become part of the longest chain after lots of mining effort.

When a transaction is broadcast and sent to the mempool, initially it has 0 confirmation before the mining. After the successful mining, the transaction is added to the official blockchain and the confirmation number becomes 1. The confirmation number is increased by one, after each new block is added

[1]The average BTC Transaction size is around 495 Bytes, yielding about 2020 transaction per block.

[2]Bitcoin's public ledger was started on January 3rd, 2009 at 18:15 UTC. The first transaction recorded in the genesis block was a single transaction paying the reward of 50 new bitcoins to its creator.

[3]Since $2020/600 = 3.37$, the limit of transactions in the Bitcoin network is about 3.37 transactions per second.

[4]For details, see, e.g., https://en.bitcoin.it/wiki/Controlled_supply.

sequentially.

Due to this very creative design, which combines both economic incentives (i.e. Bitcoin rewards and transaction fees) and mining effort (i.e. proof of the work), the Bitcoin blockchain solves the double speeding problem in a probabilistic way. More precisely, it can be shown in the next section that if a merchant waits for z confirmations before acknowledges the payment, the chance of double speeding is geometrically small, as z becomes large. The derivation of this result is closely related to the classical gamblers' ruin problem, which is discussed in great details in Section 11 of the original paper by Satoshi Nakamoto. A rule of thumb is $z = 6$; but for small transaction, a smaller value such as 2 or 3 may be chosen.

Note that economic incentives are very important to compensate the mining effort; otherwise, very few people will do mining, and the double spending problem cannot be solved. Thus, a blockchain system and its associated cryptocurrency go together. One cannot talk about blockchain by disregarding its cryptocurrency.

2.2.2 Hash Function and the Proof-of-Work

In this subsection we shall give some details about the hash function and the cryptography puzzle used in the proof-of-work for the Bitcoin blockchain. Note that instead of using proof-of-work, one can use proof-of-stake, which will be used in the next generation Ethereum network.

A hash function is a deterministic function that maps a string (or text) to a number. To be useful for blockchain applications, a hash function must satisfies some properties: (a) It should be very easy to compute the hash function, even if the original string is long. (b) It must be sensitive to a small change of the input. (c) It will be very difficult to figure out the input, given the output of the hash function. (d) Given an output of certain input, it will be time consuming to find another input such that the difference between the two outputs is smaller than a given level.

Here are some examples of SHA-256, the hash function used by Bitcoin[5]. If I type the word "FinTech," the hash function gives a 64 digit hexadecimal number

$$538372d293e8eefec27096631ac39d9680f55ca0fcfd87f5c6a9a616a7b4a9b2.$$

If I type a similar word "1FinTech," the hash function yields

$$70aa798f2683f4af0017bc8859f7fba5ed1fcdbf1fd13a70c95aa074df43a388,$$

a very different output. In fact, I can copy the entire book of "The Adventures of Sherlock Holmes" (from http://www.gutenberg.org/cache/epub/1661/pg1661.txt), and get a (simple!) hash function output

$$913aece12638e921dd69b9a1b60c1148184637071e8002e034a8518f268b0aef.$$

[5]One can use the web page https://anders.com/blockchain/hash to generate the hash function, and see a demonstration of blockchains.

However, if I make a tiny change of the first letter in the beginning sentence, "Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur Conan Doyle," in the book from "P" to "p", the hash function output becomes drastically different:

$$bfe331c8c1c779c1014bc9b2f4a2d0842e50b567da923ba412532cc048a9ca51.$$

Furthermore, the whole computation takes almost no time, at least I did not notice any time lapse in my computer. Note that the SHA-256 can handle a huge amount of input data, as $2^{256} = 1.158 * 10^{77}$, while, according to an estimate by Lyman and Varian, the US Library of Congress had 300 million books (208 terabytes, 1 terabytes being 10^{12} bytes) in 2000.

From the above examples, we have an understanding of how the hash function SHA-256 meets the requirement (a)-(c) above. To see how the hash function meets (d), we use the following example related to the real operation of Bitcoin network, in particular the cryptography puzzle used in the proof-of-work for Bitcoin blockchain.

Consider a string of "1" and 64 zeros pasted together leading to a new string 1 with 64 zeros after, i.e. we have 1||P, where P is a string of 64 zeros and the notation || means concatenation. In the real blockchain application, 1 indicate the series number of the block, and the 64 zeros is the initial input of the hash function for the first block. Then the hash function output is[6]

$$0a2a55b65844afad47168cf3f371458ae75167492438485adab7c48c2ca88400.$$

Now the cryptography puzzle is to find another number n, such that when we insert n to the original string to get a the new string, which is 1||n||P, the new hash function is so close to zero that the first 4 digits of the output should be 0000. It takes several seconds in my computer to find out that, with n=11316, we have a hash output for the new string to be

$$000015783b764259d382017d91a36d206d0600e2cbb3567748f46a33fe9297cf.$$

In the real Bitcoin blockchain, n is called nonce, and one has to find the nonce to solve the puzzle with about 17 zeros (which is adjusted periodically), not just 4 zeros!

In the real world, one needs to buy special computers, with powerful graphics processing units (GPU), called mining machines, to solve this puzzle. That is why a special word "hash power" is often used to indicate the computation power to solve puzzles related to hash functions. Of course, the number of zeros required in the puzzle is adjusted periodically so that the expectation of a successful mining in the whole Bitcoin blockchain is about 10 minutes.

Each new block of the Bitcoin blockchain consists a string b||n||D||P, where b is the series number of the block, n is the nonce to be mined, D is the new text string (new transaction data) in this block, and P is the previous hash output.

[6] See https://anders.com/blockchain/blockchain

The mining cryptography puzzle is to find n, such that the hash output begins with the required number of zeros.

For example, if in addition to the previous block 1, which has b= 1, n= 11316, D empty, and P a string of 64 zeros, we add blocks 2 to 5, all have empty D. Then, we need to do the mining from block 2 to 5. In particular, the hash P, which is obtained from block 1,

$$000015783b764259d382017d91a36d206d0600e2cbb3567748f46a33fe9297cf,$$

becomes the previous hash in block 2, and nouce needs to mine the string b= 2, D empty, and P is $n = 35230$. Continuing do this, we find the nouces for block 3, 4, 5 are 12937, 35990, and 56265, respectively.

It is now clear how the blockchains are linked together, as each new block contains the hash output from the immediate precede block. A tiny change in any block will change all the subsequent blocks. For example, if we change the data in the second block from empty to "FinTech," then we have to do redo the mining for block 2 to get a new nouce 95295. But this will leads to a change in the output hash in block 2, which also changes the input hash in block 3. Consequently, redo the mining in blocks 3 yields the new nouce 141578. Continuing do this lead to the nouces for blocks 4 and 5 to be 124307 and 6160, respectively.

In the real world blockchains, D in each block contains the coinbase information on who got the last mining award, and all transaction information within the block; P in each block contains the hash information based on a Merkle tree[7]. Then copies of the blockchains are distributed among many peers. Each peer will only need to check the hash in the last block to see whether there is any differences between these copies.

2.2.3 Mining, Double Spending, and Gamblers' Ruin Problem

We shall present the original approximation in Section 11 of the original paper in Nakamoto (2008), showing heuristically that with the honest people control most of the computing power, the probability of double spending diminishes exponentially fast, as the number of required confirmation, z, increases. We show the argument below.

Suppose A tries to create a double speeding, by first sending money via the Bitcoin blockchain network to B. At time 0, A first broadcasts the transaction from A to B to the official blockchain mempool, waiting for the transaction to be official confirmed by miners. In the meantime, at time 0, A secretly create another transaction from A to A', which is just another account owned by A, but does not broadcast the secret transaction. A uses the computer power to

[7] A Merkle tree is a way to process and to store the previous hash information more efficiently. More precisely, each branch of the tree has two nodes and the hash of the two combined hash is passed on to the next branch. In this way, one only needs to check the most recent block to see if there is any change to the previous blocks, and one can trace back a transaction using $O(\log_2(n))$ operations, rather than $O(n)$.

mine blocks contains the secret transaction. After waiting for z confirmations, B delivers the goods/service to A. Immediately, A broadcasts the secret blocks containing the transactions to A', and hopes that the secret transaction and the chain that A secretly created will become part of the longest chain, thus creating an official revising of the original transaction to B.

Assume that there are two groups of people, dishonest people who support A and honest people who support B, respectively. Honest people will only mine the real blocks, while the dishonest people only mine the fake (secret) blocks. Furthermore, we assume that the longest chain is determined exactly without any ambiguity.

Let τ_i be the mining time to get the ith block for the real chain after time τ_{i-1} (with $\tau_0 = 0$), and τ_i' be the mining time for the ith block for the fake (double spending chain) after time τ_{i-1}' (with $\tau_0' = 0$). A reasonable assumption, due to the approximate memoryless of hash function, is that τ_i and τ_i' are all independent and identically distributed random variables with exponential distribution with rate α and α', respectively.

Note that due to the almost randomness of the hash function, we can also assume that α and α' are proportional to their computing power for hash functions, i.e. $\alpha = Ch$ and $\alpha' = Ch'$ for the same constant C, where h and h' are computing power for hash functions.

Let p be the probability of the honest people mining one block faster than the dishonest people; more precisely.

$$p = P(\tau < \tau') = \frac{\alpha}{\alpha + \alpha'} = \frac{h}{h + h'}, \quad q := 1 - p = P(\tau' < \tau) = \frac{h'}{h + h'}.$$

Here we use the fact for two independent exponential random variables τ and τ' with rates α and α', respectively, we have

$$P(\tau < \tau') = \frac{\alpha}{\alpha + \alpha'}.$$

To show this note that

$$\begin{aligned} P(\tau < \tau') &= \int_0^\infty P(\tau < \tau' | \tau = t) \alpha e^{-\alpha t} dt \\ &= \int_0^\infty P(t < \tau') \alpha e^{-\alpha t} dt \\ &= \int_0^\infty e^{-\alpha' t} \alpha e^{-\alpha t} dt \\ &= \frac{\alpha}{\alpha + \alpha'}. \end{aligned}$$

Assuming the honest group has more computer hash power, $h > h'$, then $p > 1/2, \quad q < 1/2$. Thus, the dishonest people are playing a game with unfavorable odds.

There are only two ways for the dishonest people to win the race: (a) When B gets z confirmations, A has already mined a secret chain of length at least

z. Here for simplicity, we assume that when there is a tie of equal blocks, the dishonest people win the race and their dishonest chain become the longest chain. (b) When B gets z confirmations, A has mined a secret chain of length less than z, but A eventually catches up with B by having the longer chain sometime in the future, even with the unfavorable odds and unfavorable initial conditions.

To find out the probability of a successful attack, i.e. the dishonest people winning the race, we need to introduce some notations first. Define $N(t)$ to be the number of new blocks mined by time t by the honest people and $N'(t)$ the number of new blocks mined by time t by the honest people. More precisely,

$$N(t) = \max\{n \geq 0 : \quad S_n < t\}, \quad S_n = \sum_{i=1}^{n} \tau_i;$$

$$N'(t) = \max\{n \geq 0 : \quad S'_n < t\}, \quad S'_n = \sum_{i=1}^{n} \tau'_i.$$

Then the probability of a successful attack, Q, is simply the sum of two probabilities,

$$Q = P\left\{N'(S_z) \geq z\right\} + \sum_{k=0}^{z-1} P\left\{N'(S_z) = k\right\} q_{z-k},$$

where q_{z-k} is the event that an attacker chain catch up the main block chain, even if the attacker's chain is $z-k$ blocks behind. Indeed, the first term relates to the event (a), and the second term is the probability of the event (b).

To compute q_n, note that this is equivalent to the gamblers' ruin problem with starting point $\$n$ ever goes to zero, with winning probability being p and losing probability being $q = 1 - p$. A standard result from the gamblers' ruin problem, which will be reviewed in the appendix, based on a simple recursion yields that $q_n = (q/p)^n$, $q < 1/2$.

Now Nakamoto (2008) uses the approximation

$$S_z \approx E\left[S_z\right] = \frac{z}{\alpha}.$$

Thus, $N'(S_z)$ can be approximated by a Poisson random variable with mean $\frac{z}{\alpha}\alpha' = zq/p$. Therefore,

$$\begin{aligned} Q &= 1 - P\left\{N'(S_z) < z\right\} + \sum_{k=0}^{z-1} P\left\{N'(S_z) = k\right\} q_{z-k} \\ &\approx 1 - \sum_{k=0}^{z-1} P\left\{N'(\frac{z}{\alpha}) = k\right\} + \sum_{k=0}^{z-1} P\left\{N'(\frac{z}{\alpha}) = k\right\} q_{z-k} \\ &= 1 - \sum_{k=0}^{z-1} e^{-zq/p} \frac{(zq/p)^k}{k!} (1 - (q/p)^{z-k}), \end{aligned}$$

which decay geometrically in z.

For example, if $q = 0.1$, then the above approximation for the double spending probability yields 0.0009137, 0.0002428, 0.0000647, 0.0000173 for $z = 5, 6, 7, 8$, respectively. A rigorous derivation and an exact calculation are given recently in Grunspan and Perez-Marco (2018).

It is also difficult to know what the true q is. The default conclusion in Nakamoto (2008) is that as long as q is smaller than $1/2$, it will be difficult for an attacker to have a successful attempt. Sometimes, even with q as low as $1/4$ there still could be a successful attack; see, e.g., Eyal and Sirer (2014). In addition, in reality there could be the third group of people who just follow the trend of the perceived longest blockchain, and there could be disagreement on which one is the longest chain due to various stochastic noises, e.g. time latency and network outrages. It is an interesting open problem to study the double spending probability under more realistic models.

Appendix: Gambler's Ruin Problem

Consider a Markov chain $\{X_n\}$, with $n \geq 0$ being the time, on integers $i = 0, \pm 1, \pm 2, ...$ with transition probabilities

$$P_{i,i+1} = p, \ P_{i,i-1} = 1 - p, \ \ i = 0, \pm 1, \pm 2, \ \ 0 < p < 1.$$

This can be interpreted as a model for Gambler's winnings, with p being the winning probability.

Suppose we win one dollar with probability p, and lose one dollar with probability $1 - p = q$. First of all, this is a Markov chain with

$$P_{00} = P_{NN} = 1,$$

$$P_{i,i+1} = p, \ P_{i,i-1} = q = 1 - p, \ \ i = 1, ..., N - 1.$$

It is easy to see that $\{0\}$ and $\{N\}$ are recurrent, as they are absorbing states; and the other states are transient. Therefore, after some finite amount of time periods, the gambler will either get $\$N$ or go broke.

Question: Starting with $\$i$, let Q_i be the probability that the gambler's fortune will reach $\$N$ before reaching 0. What is Q_i?

To solve this question, note that by conditioning on the first transition we have a recursion

$$Q_i = pQ_{i+1} + qQ_{i-1}, \ \ i = 1, ..., N - 1.$$

To solve this recursion, note that

$$Q_{i+1} - Q_i = \frac{q}{p}(Q_i - Q_{i-1}), \ \ i = 1, ..., N - 1.$$

$$Q_0 = 0, \ \ Q_N = 1.$$

Thus,

$$Q_{i+1}-Q_i = \left(\frac{q}{p}\right)^2 (Q_{i-1}-Q_{i-2}) = \cdots = \left(\frac{q}{p}\right)^i (Q_1-Q_0) = \left(\frac{q}{p}\right)^i Q_1, \;\; i = 1, ..., N-1.$$

Adding them together, we have

$$Q_i - Q_1 = Q_1 \left\{ \left(\frac{q}{p}\right) + \left(\frac{q}{p}\right)^2 + \cdots + \left(\frac{q}{p}\right)^{i-1} \right\},$$

yielding

$$Q_i = \left\{ \begin{array}{ll} \frac{1-(q/p)^i}{1-(q/p)} Q_1, & \text{if } q \neq p \\ iQ_1, & \text{if } q = p \end{array} \right\}.$$

What is Q_1? Since $Q_N = 1$, we have

$$Q_1 = \left\{ \begin{array}{ll} \frac{1-(q/p)}{1-(q/p)^N}, & \text{if } q \neq p \\ 1/N, & \text{if } q = p \end{array} \right\}.$$

In summary, we have

$$Q_i = \frac{1-(q/p)^i}{1-(q/p)^N}, \;\; \text{if } p \neq 1/2; \quad Q_i = i/N, \;\; \text{if } p = 1/2.$$

Note that as $N \to \infty$

$$Q_i \to \left\{ \begin{array}{ll} 1-(q/p)^i & \text{if } p > 1/2 \\ 0 & \text{if } p \leq 1/2 \end{array} \right\}.$$

This is the probability that a person starting with i will never go bankrupt (i.e. reach 0).

Now in the double spending problem for the blockchain, the probability of a successful double spending, starting with n blocks behind, is a person starting with n will ever reach 0. Thus, the required probability is

$$1 - Q_n = (q/p)^n, \quad q < 1/2.$$

Exercise 1. One extension of the model of the double spending problem for the blockchain is to allow the attacker to give up after the attacked is n blocks behind. This is related to the following problem. Suppose that the gambler continues to bet until either wins n dollars or loses m dollars. What is the probability that the gambler quits as a winner?

Exercise 2. One extension of the model of the double spending problem for the blockchain is to allow the winning probability depending on the state variable. This is related to the following problem. Suppose in the gambler's ruin problem that the probability of winning depending on the gambler's current fortune, i.e. p_j is the probability that the gambler wins a bet when the wealth is j. Compute Q_i.

Bibliography

Back, A. (2002): "Hashcash - A Denial of Service Counter-Measure," Working Paper.

Ben-Or, M. (1983): "Another advantage of free choice: completely asynchronous agreement protocols," *ACM PODC 1983*.

Catalini, C. and J. S. Gans (2016): "Some Simple Economics of the Blockchain," *NBER Working Paper*, http://www.nber.org/papers/w22952.

Chaum, D. (1983): "Blind Signatures for Untraceable Payments," Working Paper.

Dwork, C. and M. Naor (1992): "Pricing via Processing or Combatting Junk Email," *CRYPTO 1992*, 139–147.

Eyal, I. and E. Sirer (2014): "Majority is not Enough: Bitcoin Mining is Vulnerable," *Financial Cryptography and Data Security 2014*.

Fischer, M. J., N. A. Lynch, and M. S. Paterson (1985): "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM*, 32, 374–382.

Gilad, Y., R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich (2017): "Algorand: Scaling Byzantine Agreement for Cryptocurrencies," *SOSP 2017*.

Grinberg, R. (2011): "Bitcoin: An Innovative Alternative Digital Currency," https://papers.ssrn.com/abstract=1817857.

Grunspan, C. and R. Perez-Marco (2018): "Double Spend Races," Working Paper.

Haber, S. and W. S. Stornetta (1991): "How to time-stamp a digital document," *Journal of Cryptology*, 3, 99–111.

Harvey, C. R. (2016): "Cryptofinance," https://papers.ssrn.com/abstract=2438299.

Khapko, M. and M. Zoican (2018): "Smart Settlement," https://papers.ssrn.com/abstract=2881331.

Lamport, L., R. Shostak, and M. Pease (1982): "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, 4, 387–389.

Micali, S. and P. Feldman (1997): "An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement," *SIAM Journal of Computing*, 26, 873–933.

Nakamoto, S. (2008): "Bitcoin: A Peer-to-Peer Electronic Cash System," https://bitcoin.org/bitcoin.pdf.

Rogoff, K. (2015): "Costs and Benefits to Phasing out Paper Currency," *NBER Macroeconomics Annual*, 29, 445–456.

Szabo, N. (1994): "Smart Contracts," Working Paper.