ARTICLE

k-root-n: An efficient algorithm for avoiding short term double-spending alongside Distributed Ledger Technologies such as Blockchain

Author: [REDACTED]

Correspondence: [REDACTED]

Abstract

Blockchains such as bitcoin rely on reaching global consensus for the distributed ledger, and suffer from well know scalability problems. We propose an algorithm which can avoid double-spending in the short term with just O(√n) messages, relying on the fact that the velocity of money in the real world has coins circulating through at most a few wallets per day. The k-root-n algorithm is suitable for an environment with synchronous or asynchronous (but fairly low latency) communication and with Byzantine faults. The "k-root-n" algorithm presented should be practical to avoid double-spending with arbitrarily high probability, while feasibly coping with all world commerce, provided there is a suitable cap on failure modes and network latency. It is resistant to Sybil attacks even well beyond 50%. This k-root-n algorithm is less efficient in the long term, once money circulates through a significant proportion of the world's wallets, and should, therefore, preferably be used as a complement, and not a replacement, to a global distributed ledger. With this two-tier system, global consensus on the ledger may be reached periodically and with a considerable time lag, while money may be safely spent, with rapidly confirmed transactions, in-between. The algorithm works best if every wallet is also a node which is online with fairly high availability.

Keywords: *blockchain, bitcoin, cryptocurrency, distributed ledger technology*

Introduction

In blockchains such as bitcoin, all n nodes reach Nakamoto Consensus [1] on each block of transactions, creating a scalability problem [2] [3] [4] which notoriously limits the entire bitcoin network to a few transactions per second while consuming massive power [5]. Nakamoto Consensus is considered the first digital currency algorithm to solve the double-spending problem without the need of a trusted authority or central server. It can cope with Byzantine faults (e.g. [6] [7]) including a Sybil attack [8] of up to 50% dishonest nodes. However, it requires O(n) communication messages per transaction which limits its scale.

In practice, bitcoin transactions suffer from a lag time of between 15 minutes to several hours before being included in a block on the bitcoin blockchain [9] (this lag time has a complex dependency on how high a fee is offered by transaction participants to the miner [10]), and then an hour longer to reach the commonly desired threshold of six-block confirmation [11]. Thus, there is typically a several hour latency, before received bitcoin transfers are confirmed and are safe to re-spend.

At the time of writing, the typical fee paid to the miner for a single bitcoin transaction, is tens of thousands of Satoshi or about US$0.50 - $5 [9], a price similar to most domestic bank transactions.

There are ongoing efforts to redesign the blockchain algorithm itself for greater scalability such as SCP [12], Algorand [13], Bitcoin-NG [14] all of which involve selecting a subset of users (committees, or a rotating leader) in various configurations to reduce the number of messages required to reach consensus. In an alternative approach, a subset of nodes transact with each other off-chain for a time [15] as in the Lightning Network [16].

In this paper we consider an approach to protecting against a double spending [26], possibly combined with a Sybil attack, without the need for global consensus. There are other forms of attacks not considered in this paper such as eclipse attacks [17], routing attacks [18], attacks based on time advantage [19], incentive attacks [20], quantum computing attacks [21]. Relevant surveys and discussions of attack formats are [22] [23]. Some previous discussions of avoiding double spending include [24] [25].

We propose an alternative approach, a scalable low-latency algorithm which can run in parallel to a global consensus mechanism such as blockchain, protecting against double-spending in the short-term, while the n nodes are working to reach consensus on transactions, possibly with a lag of some hours from the transaction time. With this algorithm, we can also accept a situation where consensus is achieved infrequently. Therefore we could accept longer blockchain blocks which are created every hour, or every few hours, rather than bitcoin's average of 10 minutes, thus increasing blockchain's transactions per second [27], while compensating for the longer latency with our alternative algorithm for preventing short-term double spending.

For example, each morning the nodes may reach consensus on the valid transaction histories and wallet balances as of the preceding midnight GMT, and they may do so asynchronously, reaching the consensus by, say, 6 am the next morning. For example, in the specific case of bitcoin, by 6 am all the transactions from the previous day will typically have achieved six-block verification and may be considered final. We now present an algorithm to allow fast transactions in the 30 hours from say Sunday midnight, till Tuesday 6am, when consensus is finalized for the ledger as of Monday midnight. Thus in this example we allow for a situation where the distributed ledger is relaxed, relative to bitcoin, to reach consensus every 24 hours with a 6-hour lag.

This 24-hour cycle and 6-hour lag is just a useful example. The purpose of the present algorithm is that global consensus on transactions is only reached periodically, and with a considerable lag from the transaction time, and, in the meantime, we propose a solution to allow people to trade, in particular to pass on received coins safely, with next to zero latency.

The proposed algorithm, called $k\sqrt{n}$ or "k-root-n", can avoid double-spending in the short to medium term, while there is no global consensus on the ledger, with an arbitrarily high probability of detecting double-spend, with just O($\sqrt{n}$) messages per transaction. This is under the assumption that specific money balances only circulate through O(constant) wallets in 24 hours. This assumption is realistic, as in the real economy money circulates with a velocity measured in one or two transactions per month [28], and bitcoin specifically is already practically constrained by lag times to circulating a few times per day, and in practice rarely more than once or twice a day.

In this algorithm, every transaction can eventually be on-chain. But the initial transaction verification is off-chain, allowing transactions to continue off-chain at high speed, while waiting for the blockchain to catch up. The algorithm only involves O($\sqrt{n}$) nodes and messages in each transaction; typically, we will assume that we pick a number of verification messages around $10\sqrt{n}$.

Overview of $k\sqrt{n}$ random double-spending detection

Suppose we have n nodes, each of which is also a wallet, connected to a network, and they achieved consensus on the global distributed ledger (or at least on the balance of each node) using blockchain (or another algorithm) as of some time in the recent past, a time we shall call the **consensus checkpoint**.

We assume for now that every wallet is also a node, which is online and available most of the time, and provides basic verification services to the network. The core idea is that any honest node that wants to verify that the funds it receives have not been double-spent, will demand from the sender the "pedigree" of those funds, namely the sender's own transaction history since the last global consensus, and in case the sender is depending on incoming funds to have balance to cover the transaction, the receiver will recursively demand the source of funds right back to funds that were available as of the last consensus checkpoint.

Since querying all the nodes to verify each transaction is prohibitive, an honest node will check the pedigree of each inbound transaction with a random $k\sqrt{n}$ other nodes. We will see that on average if two nodes query a random $k\sqrt{n}$ nodes, the probability of zero common nodes is extremely small for suitable k, even if some proportion of nodes are failing or malicious.

Each honest node provides verification services by keeping a history of the transaction pedigrees it has been asked to verify, so that when two honest nodes query random nodes, any one common queried honest node will immediately raise the alarm if the two receiving nodes are victims of a double spending attempt.

Here k is a small number greater than 1, we will generally assume k=10. Assuming we are in an environment with Byzantine faults, say 10% of nodes may not respond due to node or network failure, and assuming up to 50% of nodes are malicious, we would have an effective $\underline{k}$=4.5 i.e. $k\sqrt{n}$ responsive, honest nodes. When two honest nodes each receive funds and successfully query $4.5\sqrt{n}$ honest nodes each, this $\underline{k}$=4.5 is sufficient to ensure an average of >20 common, honest, responsive nodes. We will see that there is a probability of just ~10^{-9} of zero common, honest, responsive nodes. Therefore, the chances of getting away with double-spending are negligible, and there is a probability very close to 1 that any double spending will be detected as soon as both branches of the spend reach honest nodes.

The penalty for double-spending should be at least forfeiting the wallet, so if each wallet has a minimum stake $\boldsymbol{m}$ of $1, and each transaction is limited to well under $1 billion, say to a maximum $\boldsymbol{M}$=$1,000,000, then there is no expected gain from double-spending, as there is a probability of just ~10^{-9} of not getting caught.

Of course, a dishonest node may not be checking its inbound transactions or may be maliciously collaborating with other nodes. This is why the receiving honest node need to check not only the transaction history of the immediate sender for forked history/double-spending, but also to recursively check any of sender's sender's transactions, at least in any case that the immediate sender is relying on the sender's sender (recursively) to have balance to cover the current transaction. That is, the receiver will treat any inbound transaction (since the last global consensus of the network) as suspicious, and if any of its source transactions are critical to providing cover for the current transaction, that transaction will be recursively validated before the transaction is accepted. We call this recursive tree of inbound transactions, the **pedigree** of the transaction. This is why the k-root-n algorithm is less efficient for the long term, as the pedigree of transactions may become large over a longer period of time.

To see how well a $10\sqrt{n}$ algorithm scales, assume n=10 billion people (the projected world population for 2050 [29] and much more than bitcoin's current 32m wallets [30]). Suppose people are each transacting on average once per hour, 24-hours per day (higher than the average rate of commerce). Each transaction will involve messages to $10\sqrt{n}$=1 million nodes. We will see that this gives a probability of $p \approx 10^{-9}$ of getting away with double-spending, even if half the nodes are fraudulent and 10% of the nodes are unavailable (i.e. $4.5\sqrt{n}$ honest, responsive, validating nodes). So, each transaction only burdens 1 out of 10,000 nodes, and with 10 billion transactions per hour globally, or 2.77million transactions per second globally, each node has to be involved in just 278 transactions per second, which is feasible for a modern computer (especially in 2050).

Thus, it seems practical that this algorithm could securely cope not only with Visa/Mastercard volumes but with all the commerce in today's world and in the foreseeable future. Visa's volumes have been widely misquoted in bitcoin articles as 24,000 per second, although that appears to be mythical [31], with apparently more reliable sources talking about 78.95 billion Visa transactions in the first half of 2018 [32] which averages 5,000 per second, although peak times would of course be higher.

We now introduce some definitions, and then specify the k-root-n algorithm more formally.

Formal preliminaries to the k-root-n algorithm

Definition 1 A **transaction** $T=(x,t,S,R,s_1,s_2)$ is an agreement to transfer a balance from a sender node/wallet to a receiver node/wallet, comprising a positive amount $x=x[T]$, a time stamp $t=t[T]$, identifiers (public keys) of the sender user $S=S[T]$ and the receiver user $R=R[T]$ and their respective digital signatures s_1 and s_2 of the data tuple (x,t,S,R). ■

Definition 2 The **balance** $b[u,t_1]$ of a user u at time t_1 assuming s/he had a balance of b_0 at the time t_0 of the last known global consensus is $b_0 + \sum_{t_0<t[T_i]<t_1,R[T_i]=u} b[T_i] - \sum_{t_0<t[T_i]<t_1,S[T_i]=u} b[T_i]$ i.e. the last known global consensus balances plus all received amounts, minus all spent amounts. A transaction is fraudulent if it would leave the sender with a balance below a minimal balance of m. ■

Definition 3 The **Lineage** ($LIN[T]$ of a Transaction T) is the Transaction History for the Sender $u[T]$ from the last known consensus checkpoint at time t_0 before $t[T]$, and up to the time of T $LIN[T] = \{T_i \mid t_0 < t[T_i] < t[T]\,,\ \mathrm{S}[T_i] = \mathrm{S}[\mathrm{T}] \vee \mathrm{R}[T_i] = \mathrm{S}[\mathrm{T}]\}$. ■

These are the transactions relevant to establishing that user has sufficient balance to afford $\boldsymbol{T}$.

Definition 4 The **Critical Lineage** ($CLIN[T]$ of a transaction T) is an ordered list of transactions whose elements are a minimal subset of $LIN[T]$ critical to the balance which allows the sender to afford T. Formally, suppose a Sender S makes a payment of amount x in a Transaction T and suppose s's last known consensus balance was b_0 and suppose the set of transactions which s participated in since the last Global Consensus, $LIN[T]$ includes inbound payments $r_1 \ldots r_n$ in descending order of amount (and chronologically when equal) and outbound payments $s_1 \ldots s_m$. Now the critical inbound payments are the smallest subset $CLIN[T]=(r_1 \ldots r_j)$ of inbound payments with j minimal such that $b + \sum_i^j r_i - \sum_i^m s_i \geq x + m$. ■

Thus $\{r_1 \ldots r_k\}$ is a minimal subset of inbound transactions which are sufficient to provide balance coverage for this payment of $\$x$, given that the Sender has opening balance $\$b$ and has spent the s_i. Validating by Receiver of these critical inbound payments $CLIN[T]$ of Sender is sufficient to ensure

Sender can afford $\$x$, even if the other inbound payments $r_{j+1}...r_n$ derive directly or indirectly from fraud. Therefore, the minimum due diligence of the receive $R[T]$ is to check that $LIN[T]$ is complete and then recursively check the lineage of each transaction in $CLIN[T]$. We now formalize this recursive set of transactions.

Definition 5 The **Pedigree** ($PED[T]$ of a Transaction T) - is a set of transactions defined as the recursive closure of all transactions reachable from T using $CLIN[T]$. To compute this:

- Start with the Set of $CLIN[T]$.
- Recursively for each new T_1 added for the first time to the Set add also $PED[T_1]$ to the Set.
- That's it. ■

It may also be helpful to think of $PED[T]$ as the nodes of a directed acyclic graph for each transaction showing recursively the inbound transactions, since the last known global consensus, that the sender depended on for covering the transaction.

k-root-n algorithm

Suppose we have n nodes, each of which is also a wallet, connected to a network, and the nodes achieve consensus on the global ledger (or at least on the balance of each node) as of some time in the recent past, a time we shall call the **consensus checkpoint**. Each consensus checkpoint may become **known** with some latency after the time which the consensus ledger relates to.

The nodes transfer balances to each other by mutually digitally signing transactions. According to the algorithm, each receiving honest node, before accepting and signing a transaction T, will do the following

- Demand sender's recursive list of dependent transactions $PED[T]$ and a Transaction History LIN[T_1] for each T_1 in $PED[T]$.
- Validate that each such transaction was properly formed and signed by known nodes, and had balance to cover it based on the last known global consensus balance plus the provided LINs.

- Randomly choose $k\sqrt{n}$ (or the nearest integer) **validating nodes** on the network, send each (directly or by communicating through a cascading tree of nodes) all the LINs and ask the validating nodes to validate that they have not seen any alternative LIN history for any of the transactions in this recursive list *PED[T].*
    - Any honest node receiving this request will validate that they have not seen a contradictory LIN history for any of the transactions. The nodes then store every LIN transaction history they are asked to validate till the next achieved global consensus, for future validation.
- If the **validating nodes** have seen an alternative history, they inform the receiver and the transaction should be rejected. Proof of fraud for the wallet with two alternative histories is broadcast to all nodes. The wallet will be blacklisted and any balance forfeited.
    - In such a case the two honest receivers will also check if they consulted any other validating nodes in common, and if they did and that node failed to report the double-spending, it should also be blacklisted.
- Otherwise the transaction is accepted by receiver.

Periodically all nodes will take the time to reach a global consensus on the distributed ledger e.g. using Nakamoto Consensus. All recipients will request that the transactions they received be added to the global ledger. In the case a node has double spent it will be disqualified. All fraudulent transactions will be iteratively removed from the ledger and any transaction which has any fraudulent transaction in its pedigree. These concepts are now defined more formally.

Definition 6 Fraudulent transaction (A) any transaction T where the sender provided the receiver with a *LIN[T]* which was missing any transaction which the sender had signed; and (B) any transaction which the sender subsequently failed to disclose when providing a *LIN* of a later send. ■

Thus, if Malory sends money to Alice and later double spends by sending the same money to Bob without disclosing to Bob the earlier payment to Alice, both payments are considered fraudulent. It is not sufficient to cancel the second transaction, the one which directly involved the fraud, as Alice may be a co-conspirator of Malory, and Bob the only victim. Cancelling both transactions ensures there is a significant penalty for fraud. This does mean that in theory Bob could lose out due to

becoming a victim of double spending in retrospect, but in practice this arrangement ensures that double spending has a negative expected value and so is very unlikely to occur at all.

Definition 7 An **Invalid transaction** is a transaction which is not fraudulent but where the sender in retrospect did not have balance to cover the transaction, after removing fraudulent transactions. Equivalently, these are transactions which turn out to have a fraudulent transaction in their pedigree. ■

When achieving global consensus, invalid transactions must be iteratively identified till all invalid transactions are identified and they are all excluded from the global ledger. The process must be iterative, as invalidating one transaction may cause the receiver to have had no cover for subsequent spends thereby invalidating further transactions.

In the k-root-n algorithm there is a need for honest nodes to be online most of the time. It is recommended to have a protocol where an honest node commits to a Service Level Agreement (SLA e.g. [33] [34]) of say u=90% uptime and a node which doesn't comply may receive warnings and eventually financial penalties or disqualification by consensus of all nodes. A node which tries to consult $k\sqrt{n}$ nodes and receives less than $uk\sqrt{n}$ responses in a specified target latency time, should pick other nodes and retry till receiving the target $uk\sqrt{n}$ validations.

An example of detection of double spending via k-root-n algorithm

Suppose during Monday morning the network reaches consensus that as of Sunday midnight the balances on the distributed ledger after all valid transactions were

Chuck (malicious)	$100
Mallory (malicious)	$100
Alice (honest)	$100
Bob (honest)	$100

The ledger also shows that at Sunday midnight there were a total of n valid nodes, each with at least a minimum stake of m=$1.

We analyze the scenario where Chuck conspires with Mallory to double-spend, by giving the same money to Alice and also to Bob. The payment to Bob will be passed by Chuck via co-conspirator Mallory, in an effort to conceal the double-spend.

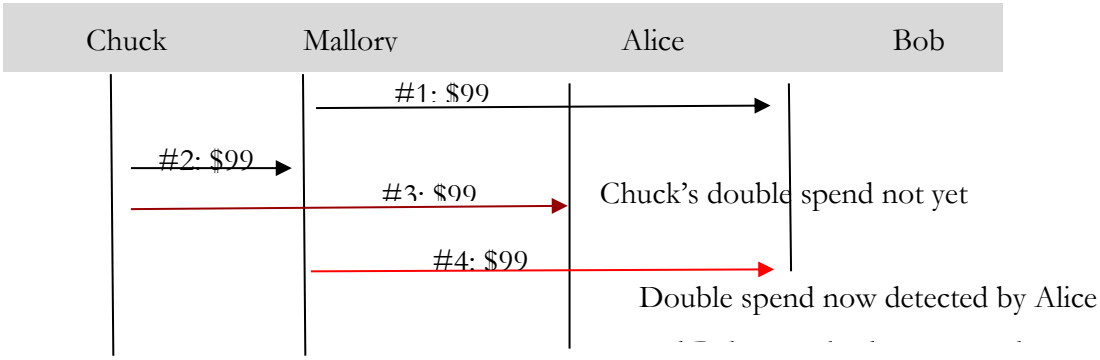

Figure 1: Example of detecting an attempted double spending

1. Mallory sends $99 to Bob (in exchange for some goods or services). Mallory declares her transaction history from the last consensus, which is empty, so she has $99 to spare. Bob first confirms Mallory had $100 as of the last known consensus checkpoint. Bob being honest then queries $k\sqrt{n}$ network nodes (either directly or through a cascading tree of nodes) to confirm that none of them have heard of Mallory signing any other transactions since consensus. They have not. Bob accepts the $99 and they both digitally sign the transaction and submit it for eventual inclusion on the main distributed ledger.
2. Chuck sends $99 to Mallory. Mallory being malicious and complicit with Chuck tells no one about this transaction. They both sign the transaction and may or may not submit it to the main ledger. Chuck is passing this $99 through Mallory attempting to mask the double-spending he is planning. He might potentially pass this money through further nodes.
3. Chuck now sends $99 to Alice in return for some value. This is a fraudulent double-spend. He tells Alice fraudulently that he has no other transactions since the last consensus. Alice being honest queries $k\sqrt{n}$ network nodes. They all tell Alice that they are not aware of any forked transaction histories (since transaction #2 was not broadcast) for Chuck, and so Alice accepts the payment. Thus, the double-spend is not yet detected (until both instances of double-spent money reach honest nodes).
4. Mallory gives another $99 to Bob in return for some value, and provides Bob with a copy of Mallory's transaction history since consensus namely transaction #1 (-$99 which Bob already

knows about) and transaction #2 (+$99) thereby evidencing Mallory's balance of $100 allowing Mallory to spend $99. At this point, Chuck's double-spent money has, via Mallory, reached the honest Bob.

- Bob notices that Mallory has $100 but only when depending on money from Chuck (so transaction #2 is in the critical lineage $CLIN[\#2]$ and therefore in $PED[\#2]$). Bob will, therefore, want to validate transaction #2 and will require Mallory to provide Chuck's transaction history $LIN[\#2]$ as part of the pedigree. (Further, in case Chuck, in turn, was relying on incoming transactions for his balance in transaction #2, which is not the case here, Bob would recursively ask for sender's sender's sender's transaction history until he has a transaction history for every transaction since the last consensus which is required to justify Mallory's balance sufficiently to cover the current transaction).
- Chuck now queries $k\sqrt{n}$ random network nodes providing both Mallory's and Chuck transaction history (and any other recursively requested history).
- Some of these nodes (k^2 on average but at least 1 with an incredibly high probability) had previously been told about Chuck's alternative transaction history of transaction #3 where he gave $99 to Alice. They raise the alarm of double-spending and broadcast a fraud-proof. The proof of fraud comprises the two divergent transaction histories both signed by Chuck.
- Bob rejects the fraudulent transaction.
    - Chuck has his wallets blacklisted and forfeits his $1 minimum stake.
    - The fraudulent transaction from Chuck to Mallory (which was later hidden from Alice) will also be rejected from the distributed ledger.
    - Preferably the network should ask Mallory to evidence that he queried $k\sqrt{n}$ network nodes and when failing to do so, Mallory should also be blacklisted and forfeit his balance.
    - Alice and Bob should compare notes and find all the common nodes they had consulted and ensure none of them failed to report the double-spending. If any common node failed to raise the alarm that node should also be blacklisted for fraud, with proof of fraud showing that the node received two

alternative histories of Chuck and in both cases approved them (such approvals being signed by the node evidencing the verification fraud).

The next morning consensus is established again around the following balances:

End balances

Chuck (malicious)	$1 (blacklisted with balance forfeited)
Mallory (malicious)	$1 (blacklisted with balance forfeited)
Alice (honest)	$199
Bob (honest)	$199

Once this new consensus checkpoint is known, future senders need only provide shorter transaction histories back to the newer consensus checkpoint.

Algorithm correctness

Theorem 1 For any two honest nodes receiving and successfully validating payments with $k/\sqrt{n}$ random nodes each, there will be an average of k^2 common nodes queried by both (any one of which can detect double-spending and raise the alarm).

Proof The first honest node queries randomly $k\sqrt{n}$ nodes representing a proportion $k/\sqrt{n}$ of all n nodes. So, when the second honest node queries $k\sqrt{n}$ random nodes, on average a proportion of $k\sqrt{n} * (k/\sqrt{n}) = k^2$ will overlap. ■

This result is key to the strength of the algorithm, as both transactions involve $O(\sqrt{n})$ validating nodes, and the expected value of the number of overlapping nodes is significant allowing any double spend to be detected.

Since it only requires one common node to detect fraud, we now focus on the probability of at least one node in common versus the probability of zero common nodes, which we require to be very small.

Lemma 1 The probability $p_0(n,r)$ of zero clashes (zero common nodes) between two random sets of $r = k\sqrt{n}$ nodes, satisfies $p_0(n,r) < e^{-k^2}$ with $p_0(n,r) \approx e^{-k^2}$ for large n and $r \ll n$.

Proof

First $p_0(n,r) = \frac{\binom{n-r}{r}}{\binom{n}{r}}$ since there are $\binom{n}{r}$ ways for the second node to choose r validating nodes of which $\binom{n-r}{r}$ combinations involve zero of the same r validating nodes which the first node chose. So

$$p_0(n,r) = \frac{\binom{n-r}{r}}{\binom{n}{r}} = \frac{(n-r)!r!(n-r)!}{r!(n-2r)!n!} = \frac{(n-r)\ldots(n-2r+1)}{n\ldots(n-r+1)} < \left(\frac{n-r}{n}\right)^r = (1-\frac{r}{n})^r \text{ with } \approx \text{ for } r \ll n$$

Now let $r = k\sqrt{n}$ and we can approximate

$$p_0(n,r) = \left(1-\frac{k\sqrt{n}}{n}\right)^{k\sqrt{n}} = \left(\left(1-\frac{k}{\sqrt{n}}\right)^{\sqrt{n}}\right)^k < (e^{-k})^k = e^{-k^2} \text{ again with } \approx \text{ for the limit of large } n \quad \blacksquare$$

So e^{-k^2} is a safe upper bound for p_0 and a good approximation in the realistic case of large n and small k. By substitution we can see that $\underline{k}=4.5$ gives $p_0 \sim 10^{-9}$ for all large n, and for convenience we therefore typically assume $\underline{k}=4.5$. We must take k large enough to allow for the level of Byzantine faults in the network. A typical practical value would be $k=10$ to allow for 10% unavailable nodes and 50% fraudulent nodes so that we have $\underline{k}=4.5$ of honest available nodes. 10% unavailability seems generous for most modern networks while 50% fraudulent nodes is typically anyway the most supported by the Distributed Ledger Technology used for global consensus.

The appendix shows values of $p_0(n,r)$ and confirms that the approximation is excellent for large n and small k while providing a valid upper bound in all cases.

Now, double spending with co-conspirators is in itself of no value as the co-conspirators will not provide any value in return for a payment which they know is fraudulent and will be later rejected from the global ledger. Therefore, the algorithm depends on ensuring a negative expected value

when double spent money arrives directly or indirectly at honest nodes, catching the fraud in time before honest nodes naively provide value in return for fraudulent payments.

Theorem 2 Let M be the maximum transaction amount, m be the minimum wallet balance, n be the number of valid nodes as of the last global consensus, h be the proportion of nodes assumed to be honest, and u be the proportion of uptime required from nodes. Assuming the network is designed such that

$$p_0\left(n, \underline{k}\sqrt{n}\right) < \frac{m}{M+m} \approx \frac{m}{M},$$

where $\underline{k} = khu$, the expected return on any combination of double spending to honest nodes is negative.

Proof The maximum amount of a double-spend transaction is the maximum transaction amount M. By Lemma 1, the probability of two honest nodes not detecting a double spend is $p_0\left(n, \underline{k}\sqrt{n}\right)$, in which case there is a gain of M so . In case a double spend is detected, the double-spender will at least forfeit the minimum wallet balance m. Therefore, the maximum expected gain is

$$p_0\left(n, \underline{k}\sqrt{n}\right) M - \left(1 - p_0\left(n, \underline{k}\sqrt{n}\right)\right) m.$$

Given $p_0\left(n, \underline{k}\sqrt{n}\right) < \frac{m}{M+m} \approx \frac{m}{M}$ this expected value is negative. ■

As discussed, practical values are m=\$1, M=\$1,000,000, $\underline{k}$=4.5 which gives $p_0 \sim 10^{-9} \ll \frac{m}{M}$. Assuming h=50% honest nodes (the algorithm can handle less than 50% honest nodes, but blockchain cannot) and u=90% uptime, we need k=10 to ensure a negative expected value of any double spend.

Algorithm message space complexity

The number of messages per transactions is $k\sqrt{n}$. These may be transmitted directly or cascaded through a tree of nodes to avoid the receiving node becoming a network bottleneck.

Before discussing message size, we introduce a couple of definitions.

Definition 8 The **critical inbound transaction size** $j[T]$ is the number of inbound transactions (since the last global consensus) which a sender relies on for their balance when spending money in a transaction t that is $j[T] = |CLIN[T]|$. ■

An upper bound for $j[T]$ is the total number of inbound transactions the node has participated in since the last global consensus. In most cases, j might be zero. That is, a person spending money typically already had that money that morning. In extreme cases though, v might be large, for example a grocery store starting the day with zero balance, accepting hundreds of small transactions, and spending all their accumulated money on a large capital item or on payroll that same evening. The large spend may depend on every one of the small inbound transactions.

Definition 9 The **critical velocity of money** $v[T]$ for a transaction T is the maximum depth of the recursion in $PED[T]$. ■

$v[T]$ is the number of nodes the specific balance of coin circulated through in the time period between consensus checkpoints, until it landed in transaction T, where it is only considered circulation if the nth transaction was dependent for its balance on the $(n-1)$th. v is generally assumed to be small, most often 1 and only rarely more than 2. It is defined more narrowly than the economic concept of the velocity of money [35] which includes all circulation of currency, whether critical to the spender's balance or not, and the velocity of money itself tends to average a very modest rate of 4-11 per year [36] so that $v>2$ in a single day between consensus checkpoints would be rather rare. That is, in real commerce in say 24 hours, a specific amount of currency will rarely change hands more than once or twice and at most a handful of times.

Now, the number of transactions in the pedigree of a transaction T is the size of all transactions in the pedigree, and we can see that $|PED[T]| = o(\bar{j}^{\bar{v}})$ where $\bar{j}$ and $\bar{v}$ represents the maximum values of j and v for transactions in the pedigree recursion.

In the vast majority of transactions, we expect $v=1$ and occasionally $v=2$, but rarely more, while j will probably most often be 0, but may hit a few hundred occasionally. Thus, the message sizes in realistic commerce will typically be small, but on rare occasions we may have tens thousands of

transactions and require some megabytes of message size, which is still quite a practical message size for a modern network.

However, if the algorithm is continuously used for days and weeks without a global consensus, the message size v may grow prohibitively large.

Sybil attack

In a Sybil attack, a fraudster can create a large number of fraudulent nodes hoping to reduce the chance of two honest nodes detecting the fraudster's double-spending. We already saw that a ~50% attack does not provide a positive expected value of double spending with the recommended network parameters. What about a still larger attack?

It should be noted that the global consensus algorithm will typically fail with a 51% attack [37], but regardless we investigate whether such an attack could pay off in the k-root-n algorithm.

Suppose again that m=\$1 and M=\$1m and k=10. Suppose there are initially n nodes all of which are honest, and the fraudster creates another n fraudulent nodes, for control of 50%, and suppose further that 10% of nodes are unavailable. So, as we already saw that $\underline{k}=4.5$ and the fraudster has successfully reduced $p_0 \approx 10^{-44}$ to $p_0 \approx 10^{-9}$. But with $M/m=10^6$ there is no incentive to double-spend with $p_0 \approx 10^{-9}$.

In fact, as seen in the Appendix, the fraudster needs to get $\underline{k}<3.5$ approximately, to obtain $p_0>10^{-6}$ and achieve a positive expected value for double-spending. For this, the criminal would need $\sim 2n$ fraudulent nodes. But now the fraudster has another problem. The loss from a single unsuccessful double-spend is not limited to forfeiting the double-spending wallet, but also the loss of all the fraudulent nodes that failed to detect the double-spend, namely according to Theorem 1, $(10\text{-}3.5)^2 = 42.25$ nodes for a loss of at least 42.25m. Thus, even in this case, double spending will have a negative expected value.

Consider more generally that a fraudster creates $(f\text{-}1)n$ fraudulent nodes for a total of $\underline{n}=fn$ nodes. Now, when a user consults $k\sqrt{\underline{n}} = k\sqrt{(fn)}$ nodes, a proportion of $1/f$ of the nodes, or $k\sqrt{(n/f)}$ nodes, will be genuine, this being a proportion $k/\sqrt{(fn)}$ of all the n honest nodes. Two honest nodes

will therefore have an expectation of consulting $(k/\sqrt{f}n)(\ kn/\sqrt{f}) = k^2/f$ common honest nodes, i.e. $\underline{k}=k/\sqrt{f}$. They will also consult on average $k^2- k^2/f$ dishonest nodes, and if the double-spending is caught these $k^2(1-1/f)$ nodes will be disqualified.

So the expected payoff from a single double-spend is $p_0(\underline{k}, n)M \approx e^{-k^2/f}M$ against an expected cost of
$(k^2(1-1/f)+1)m \approx k^2m$ (the fraudulent nodes who fail to report the double-spending, plus one for the double-spending wallet) for every unsuccessful transaction, against a setup cost of $(f-1)nm$.

In practice, the number of fraudulent nodes required for double-spending to pay off is huge. For $k=10$ we can find numerically that we need approximately $f>10.5$ for a positive payoff! With say $f=11$ the fraudster would have to create a massive $10n$ fake nodes to control ~91% of the network, this at a cost of $\$10nm$, say $\$10m$ for n=1 million. Now $\underline{k}=k/\sqrt{f}\approx 3$ and so $p_0 = e^{-k^2/f} = e^{-9} \approx 0.00012$. So a double-spend of M=$1,000,000 would have an expected value of $120 while the expected cost would be losing $k^2(1-1/f)+1 = 91$ nodes at a cost of $91m=\$91$ giving an expected profit of $29. So, after such an extreme attack, a single fraudulent transaction does have a positive expected value.

However, even this strategy is doomed to fail in realistic scenarios. If n=1 million, the cost of the setup of 10 million nodes would be $10 million and the user would have to repeat the double-spending three hundred thousand times in order to recoup the initial investment. However, they would lose on average 91 nodes each time they fail, meaning they would lose the vast majority of the fraudulent nodes before recouping their investment and so the whole scheme is not feasible.

Now if we increase f further say $f \approx k^2 = 100$ then fraud can pay off. p_0 gets closer to 1 and the fraudster earns a payback tending to M as f increases. However, this requires creating $O(100n)$ nodes to dominate ~99% of the network. Several strategies can help to defend against such an extreme attack including

- Reducing M/m. Reducing M will also force honest people to have more wallets increasing n.
- Increasing k

- Biasing the $k\sqrt{n}$ random nodes towards nodes that have been around for longer or have higher balances.
- Monitoring for suspicious behavior such as the creation a huge number of wallets with minimum stake.

In summary, we have found that with the recommended parameters of k, m, M, the algorithm is immune to 51% attacks and even 90% attacks, and in fact resilient to all but the most extreme of Sybil attacks.

Variations on the algorithm for further research

k-root-n without global consensus

There would be an option of running k-root-n on its own without any common consensus on a ledger. As time goes on, j increases slowly and the verifications will become exponentially heavier. But also, each node can cache everything it knows about other nodes' verified transaction histories. Over time if money circulates throughout the entire network, every node will end up verifying every transaction at some time or another just once with $k\sqrt{n}$ other nodes, creating in the long term a complexity of $kn\sqrt{n}$ per transaction which seems unattractive. However, there is room for a lot of optimization which could make this approach of standalone k-root-n feasible.

Nodes v wallets

The assumption so far is that every wallet is a node and that providing node verification services is part of the cost of being a wallet. This may be feasible as we rapidly move to a world where all devices are online all the time, but it could also be a limitation.

There could be an alternative variation of the algorithm where not every wallet is a node. This may be helpful, as people may want their wallets to be offline or to be stored on a machine with limited processing power, bandwidth or memory. In this scenario, nodes might be paid a fee to provide verification services with a penalty for failing to report a double-spend they were aware of. The nodes could be the same machines as the nodes of the underlying blockchain. Further research is required to formally define such a network.

Forced validation

An alternative idea may be considered where even dishonest nodes are forced to consult $O(\sqrt{n})$ nodes. It is critical in this situation that the dishonest nodes are not given the opportunity to select which nodes they consult as they could pick collaborating dishonest nodes. So, we may introduce a pseudorandom formula to dictate which nodes are consulted, while also ensuring balancing the load between all nodes. The idea in this situation is that if Alice sends money to Bob, and Bob sends the same money to Charlie, then Charlie will again ask $k\sqrt{n}$ nodes to validate that Bob didn't double spend, but Charlie will not need to ask the network to validate the transaction from Alice to Bob. Instead Charlie will simply ask Bob to see the $k\sqrt{n}$ digital signatures for the appropriate nodes that signed off on the transaction with Alice, and thus Bob can verify that Bob indeed consulted the right set of $k\sqrt{n}$ and received all of their approval, creating less traffic and processing demands on the network.

We therefore propose that when two people do a transfer, they must notify a formulaically determined pseudorandom selection of $k\sqrt{n}$ other nodes and get each of their digitally signed approval. The pseudorandom selection is based on a predetermined formula which is known to all and takes as input e.g. sender's ID and timestamp. Preferably we take timestamp to the second or minute (rather than a more fine-grained time slot) to reduce sender's ability to pick and choose a specific time when the pseudorandom formula happens, in order to limit their ability to select many fraudulent nodes. As before, each of those nodes if honest will check that the sender has not double-spent.

Now for the recursive check of sender's sender etc. as needed, the receiver can simply check that all the recursive transactions have the necessary sign-off from all the nodes as determined by the pseudorandom formula. Thus, the receiver does not have to burden the network with validating the recursive transactions. Such an algorithm will scale better over longer periods of time.

This scheme suffers from some clear vulnerabilities. There is a high chance in a real network that some nodes are not available and so the sender could feasibly calculate which k^2 nodes would detect his double-spending and simply claim that those specific nodes were not available. This would have to be mitigated by common monitoring of node availability or the honest nodes will self-monitor so that anyone can later validate the claim that a certain node was unavailable at a certain time.

The sender may also have multiple wallets and multiple available time slots allowing them some choice of sending node and time slot with which choice they can try to plan a double-spend without any clashes by choosing the specific wallet and time slots where they can double-spend without any clash of the honest pseudorandom nodes. Of course, if we choose high enough k we can make this infeasible, for example with $\underline{k}=10$, $p_0=3.70x10^{-44}$ so the user would have to consider $O(10^{44})$ combinations of wallets and time slots to find one with no clashes to an earlier transaction, which is not feasible.

Therefore, it should be feasible to design an algorithm where each node (honest or not) is forced to consult a specific set of $k\sqrt{n}$ nodes based on a function of the sender and time slot, and where there is no feasible way to create a positive expected value of double spending.

Conclusion

Reaching consensus for a distributed ledger is expensive and may involve a long lag time. We have explored a two-tier system where the primary algorithm ensures global consensus is reached for the distributed ledger, but perhaps only periodically and with high latency. In the meantime, a secondary k-root-n algorithm allows parties to transact rapidly and protect against double-spending with a more efficient $O(\sqrt{n})$ probabilistic algorithm which involves validating each transaction, and recursively the transactions it depends on (the transaction's pedigree) with a random selection of $k\sqrt{n}$ nodes. We showed that it is feasible for such a network to cope with all the world's commerce, while always having a negative expected value of double spending, and being resistant to even aggressive Sybil attacks.

Further research is required to investigate the practicality of each wallet being a highly available node, or to develop the idea of separating wallets and nodes, as well as the alternative idea of choosing validating nodes formulaically.

Acknowledgments

[REDACTED]

Appendix: Table of k and p_0

This table shows p_0, the chances of zero clashes when two honest nodes consult $k\sqrt{n}$ random nodes, for various values of k (in practice we should use $\underline{k}$ net of any fraudulent and unavailable nodes) for a couple of values of n. We see that for large n, e^{-k^2} gives an excellent approximation for p_0 and for small n and large k there is divergence from the estimate, but in the direction of the probabilities being even smaller than the estimate as per Lemma 1. Thus, in all cases we can safely ignore n and plan our network based on $p_0 = e^{-k^2}$. We also see that p_0 drops rapidly with small increases in k, meaning we can substantially increase the certainty of no double spending, and further increase the negative expected value of double spending, by modestly increasing k and hence $\underline{k}$.

$\underline{k}$	$p_0(n=10^4, \underline{k}\sqrt{n})$	$p_0(n=10^{10}, \underline{k}\sqrt{n})$	e^{-k^2}
1	0.36	0.37	0.37
1.5	0.1	0.11	0.11
2	0.017	0.018	0.018
2.5	0.0016	0.0019	0.0019
3	9.30E-05	1.20E-04	1.20E-04
3.5	3.10E-06	4.80E-06	4.80E-06
4	5.80E-08	1.10E-07	1.10E-07
4.5	6.10E-10	1.60E-09	1.60E-09
5	3.70E-12	1.40E-11	1.40E-11
5.5	1.20E-14	7.30E-14	7.30E-14
6	2.30E-17	2.30E-16	2.30E-16
6.5	2.30E-20	4.50E-19	4.50E-19
7	1.30E-23	5.20E-22	5.20E-22
7.5	3.70E-27	3.70E-25	3.70E-25
8	5.60E-31	1.60E-28	1.60E-28
8.5	4.60E-35	4.20E-32	4.20E-32
9	1.90E-39	6.60E-36	6.60E-36
9.5	4.10E-44	6.30E-40	6.40E-40
10	4.40E-49	3.70E-44	3.70E-44

Bibliography

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2009.

[2] S. S. R. Network, "The Limits to Blockchain? Scaling vs. Decentralization.," *Cybersecurity, Privacy & Networks eJournal,* 2019.

[3] "Scalability https://en.bitcoin.it/wiki/Scalability," Bitcoin wiki, 2015.

[4] J. Garzik, "Making decentralized economic policy http://gtf.org/garzik/bitcoin/BIP100-blocksizechangeproposal.pdf," 2015.

[5] A. d. Vries, "Bitcoin's Growing Energy Problem," *Joule,* vol. 2, no. 15, pp. 801-805, 2018.

[6] R. Canneti and T. Rabin, "Optimal Asynchronous Byzantine Agreement," *Technical Report #92-15, Computer Science Department, Hebrew University,* 1992.

[7] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans,* 1999.

[8] J. R. Douceur, "The Sybil Attack," *Peer-to-Peer Systems. Lecture Notes in Computer Science.,* vol. 2429, p. 251–60, 2002.

[9] L. S.A., "Blockchain Unconfirmed Transactions. https://www.blockchain.com/btc/unconfirmed-transactions".

[10] D. Azzolini, F. Riguzzi and E. Lamma, "Studying Transaction Fees in the Bitcoin Blockchain with Probabilistic Logic Programming," *Information,* 2019.

[11] B. Wiki, "Irreversible Transactions https://en.bitcoin.it/wiki/Irreversible_Transactions".

[12] L. e. a. Luu, "SCP: A Computationally-Scalable Byzantine," *IACR Cryptology,* 2015.

[13] Y. Gilad, R. Hemo, S. Micali, G. Vlachos and N. Zeldovich, "Algorand: Scaling Byzantine Agreements," *SOSP '17 Proceedings of the 26th Symposium on Operating Systems Principles,* pp. 51-68, 2017.

[14] I. Eyal, A. Efe Gencer, E. Gün Sirerr and R. van Renesse, "Bitcoin-NG: A Scalable Blockchain Protocol," *e Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16),* 2016.

[15] L. Apeltsin, "A CryptoCubic Protocol for Hacker-Proof Off-Chain Bitcoin Transactions," *Cornell University ,* vol. arXiv:1408.2824, 2014.

[16] J. P. a. T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," 2019.

[17] E. Heilman, A. Kendler, A. Zohar and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," *SEC'15 Proceedings of the 24th USENIX Conference on Security Symposium,* pp. 129-144, 2015.

[18] M. Apostolaki, A. Zohar and L. Vanbever, "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies," *IEEE Symposium on Security and Privacy,* 2017.

[19] C. Pinzón and C. Rocha, "Double-spend Attack Models with Time Advantange for Bitcoin," *Electronic Notes in Theoretical Computer Science,* vol. 329, 2016.

[20] E. Ittay, G. Peter, J. Aljosha, M. Sarah, S. Nicholas, T. Itay, W. Edgar and Z. Alexei, "Pay-To-Win: Incentive Attacks on Proof-of-Work Cryptocurrencies," *IACR Cryptology ePrint Archive,* 2019.

[21] I. Stewart, D. Ilie, A. Zamyatin, S. Werner, M. F. Torshizi and W. J. Knottenbelt, "Committing to quantum resistance: a slow defence for Bitcoin against a fast quantum computing attack," *Royal Society Open Science,* 2018.

[22] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang and A. Mohaisen, "Exploring the Attack Surface of Blockchain: A Systematic Overview," *arXiv,* vol. 1904.03487, 2019.

[23] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais and S. Čapkun, "Misbehavior in Bitcoin: A Study of Double-Spending and Accountability," *ACM Transactions on Information and System Security (TISSEC),* vol. 18, no. 1, 2015.

[24] C. Pérez-Solà, S. Delgado-Segura, G. Navarro-Arribas and J. Herrera-Joancomartí, "Double-spending prevention for Bitcoin zero-confirmation transactions," *International Journal of Information Security,* vol. 18, no. 4, p. 451–463, 2019.

[25] K.-Y. Kang, "Cryptocurrency and Double Spending History: Transactions with Zero Confirmation," *Munich Personal RePEc Archive (MPRA),* vol. 96875, 2019.

[26] U. W. Chohan, "The Double Spending Problem and Cryptocurrencies," *Banking & Insurance Journal, Social Science Research Network (SSRN),* 2017.

[27] J. Göbel and A. E. Krzesinski, "Increased block size and Bitcoin blockchain dynamics," *27th International Telecommunication Networks and Applications Conference (ITNAC),* 2017.

[28] N. R. Ericsson, D. F. Hendry and S. B. Hood, "Milton Friedman and Data Adjustment," *IFDP Notes, Board of Governors of the Federal Reserve System.*

[29] "The World Population Prospects 2019: Highlights," *Population Division of the UN Department of Economic and Social Affairs,* 2019.

[30] A. Lielacher, "How Many People Use Bitcoin in 2019?," *Bitcoin Market Journal,* 2019.

[31] K. Sedgwick, "No, Visa Doesn't Handle 24,000 TPS and Neither Does Your Pet Blockchain," *https://news.bitcoin.com/no-visa-doesnt-handle-24000-tps-and-neither-does-your-pet-blockchain/*.

[32] "Global General Purpose Cards - Midyear 2018," *The Nilsen Report,* vol. 1140, no. https://nilsonreport.com/upload/issues/1140_0321.pdf, p. 7, October 2018.

[33] J. Skene, D. D. Lamanna and W. Emmerich, "Precise Service Level Agreements," *ICSE '04 Proceedings of the 26th International Conference on Software Engineering,* pp. 179-188, 2004.

[34] D. Lamanna, J. Skene and W. Emmerich, "SLAng: A Language for Defining," *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS),* 2003.

[35] I. Fisher, The Purchasing Power of Money, 1911.

[36] Federal Reserve Bank of St. Louis, "Velocity of M1 Money Stock," *Fred Economic Data,* no. https://fred.stlouisfed.org/series/M1V, 2019.

[37] M. Bastiaan, "Preventing the 51 %-Attack : a Stochastic Analysis of Two Phase Proof of Work in Bitcoin," no. https://www.hmbastiaan.nl/martijn/docs/2015/preventing-the-majority.pdf, 2015.