k-root-n: An efficient O(√n) algorithm for avoiding short term double spending in Distributed Ledger Technologies such as Blockchain

Author: Zvi Schreiber (zvi@zvi.net)

Abstract

Blockchains such as bitcoin rely on reaching global consensus for the distributed ledger, and suffer from a well know scalability problem. 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 generally circulating through at most a few wallets per day. The algorithm should be practical to avoid double spending with arbitrarily high probability, while feasibly coping with all the commerce in the world. This k-root-n algorithm is less effective in the long term once money circulates through a significant proportion of the world's wallets, and should therefore be used as a complement to a global consensus. Thus, global consensus can be reached periodically and with a considerable lag, while money can be safely spent with quick transactions in-between.

Introduction

In blockchains such as bitcoin, all *n* nodes reach Nakamoto Consensus (Nakamoto, 2009) on each block of transactions creating a scalability problem (Network, 2019) which famously limits the entire bitcoin network to a few transactions per second while consuming massive power. Nakamoto Consensus takes O(*n*) communication messages which limits its scale, although is already a huge improvement on traditional consensus algorithms like Paxos (Lamport, 1998) which take at least O(n^2) messages.

In practice, bitcoin transactions suffers from a lag time of between 15 minutes to several hours before being included in a block on the bitcoin blockchain (this lag time partly depending on how high a fee is paid), then an hour longer more to reach the commonly desired threshold of six block confirmation. Thus, there is a several hour latencies before received bitcoins can be respent.

At the time of writing the typical fee paid to the miner for a single Bitcoin transaction is tens of thousands of Satoshis or about $0.50 - $5, more expensive and slower than most domestic bank transactions.

We propose a scalable low-latency solution 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 reach consensus on transactions with a lag of some hours from the transaction time. With this algorithm we can also accept a situation where consensus is achieved infrequently (e.g. we could accept longer blocks which are created every hour, or every few hours rather than bitcoin's average of 10 minutes). The consensus itself could be the bitcoin blockchain or another blockchain or other consensus blockchain algorithms such as SCP (Loi Luu, 2015) or a traditional algorithm like PBFT (Liskov, 1999).

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 6am the next morning. Or in bitcoin terms we assume that a transaction is processed and verified within 6 hours and we take the chain up to the 6-th last block as known at 6am as a final record of all transactions dated up to the midnight before.

This timing with a 24-hour cycle and 6-hour lag is just a useful example. The purpose of the algorithm is that global consensus on transactions may only be 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.

We introduce a probabilistic algorithm called k-root-n which can avoid double-spending in the short to medium term, while there is no global consensus on the ledger, with arbitrarily high probability of detecting double spend, with just $O(\sqrt{n})$ messages per transaction. This is under the realistic assumption that specific money balances only circulate through O(constant) nodes in a 24-hour period. This assumption is realistic as in the real economy money circulates with a velocity measured in transactions per year and bitcoin is constrained by lag times to circulating a few times per day and in practice much more rarely on average.

In this algorithm the transactions are not off-chain and not limited to finite subset of user as in other solutions like the Lightning Network (Dryja, 2019). Every transaction can be on-chain. But the transaction verification is low latency allowing transactions to continue off-chain while waiting for the blockchain to catch up. And the algorithm only involves $O(\sqrt{n})$ nodes, typically we will assume that we pick a number of verification messages around $10\sqrt{n}$.

To see how well the k-root-n algorithm scales, assume n=10 billion people (the projected world population for 2050 and much more than bitcoin's current 32m wallets) transacting each on average once an 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 nodes validating). 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 278 transactions per second which is feasible for a modern computer. Thus, it seems practical that this algorithm could securely cope not only with Visa/Mastercard volumes but in fact with all the commerce in today's world and in the foreseeable future.

Basic idea of $\sqrt{n}$ random double spending detection

We assume for now that every wallet is also a node which is online and provides basic verification services to the network. The basic idea is that any honest node that want to make sure the funds they received have not been double spent, will query $k\sqrt{n}$ random nodes and share with each of them the transaction history (since the last global consensus) of the person sending them funds. Here k is a small number greater than 1, we generally recommend and assume k=10 for an effective $\underline{k}$=4.5 after deducting 10% non-respondent nodes and up to 50% fraudulent nodes.

The communication with $k\sqrt{n}$ random nodes can be directly or preferably cascading the query through a smaller number of nodes in a simple tree to avoid network bottlenecks of one node making $k\sqrt{n}$ networks calls at the same instant.

On average for any two honest nodes receiving a payment 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. To see this, the first honest node queried randomly $k\sqrt{n}$ nodes which is 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 $k\sqrt{n} * (k/\sqrt{n}) = k2$ will overlap.

Choosing say $\underline{k}$=4.5 is sufficient that there are on average 25 common honest responsive nodes and we will see that there is only a probability of $\sim 10^{-9}$ of zero common honest responsive nodes and therefore the chances of getting away with double spending are negligible. Also, the penalty for double spending is at least the minimum stake so if each wallet has a minimum stake m of $1, and each transaction is limited to well under $1 billion, say to a maximum M=$1,000,000, there is no expected benefit of double spending.

Critically though, a dishonest node may not be checking their inbound transactions. Therefore, the honest nodes need to check not only the transaction history of their immediate sender for forking/double spending, but also to recursively check any of sender's sender's, in any case that the immediate sender is relying on the sender's sender (etc.) 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 it is critical to providing cover for this transaction it will be recursively validated before the transaction is accepted.

There is an assumption here that each wallet must function as a node, must be online most of the time, and that each wallet has a minimum balance m (proof of stake threshold) so that there is always something to be lost by fraud, while every transaction has some maximum M so there is a limit on what can be gained by a single fraudulent transaction. It is also recommended that each wallet has a maximum balance (which could be simply $m+M$) so that it will not be possible to gain a high proportion of the wallets with an infinitesimally small proportion of the wealth.

And again, there is an assumption that the entire network reaches consensus on all transactions periodically e.g. with Blockchain (e.g. every 24 hours and with some hours of latency). This protocol provides performant transaction security in-between, while the participants must trust that any valid transaction will eventually make its way into the main blockchain or other main consensus algorithm which is the only long term record of transactions. The reason for this is that over a long time this k-root-n algorithm will slow down as money circulates through many nodes and the recursive checking becomes expensive.

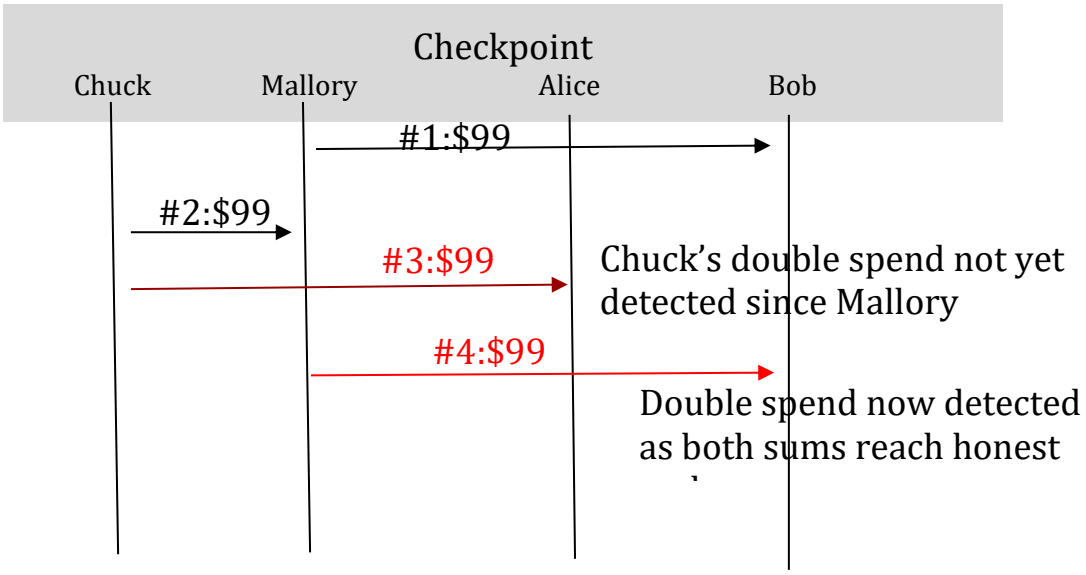
Example

During Monday morning the network eventually reaches consensus that as of Sunday midnight the balances after all transactions were

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

At that time there were a total of *n* valid wallets each of which with a minimum stake of m=$1.

We analyze this scenario where Chuck conspires with Mallory to double spend by giving the same money to Alice directly and to Bob via Mallory in an effort to conceal the double spend.



1. Mallory gives $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 consensus. Bob being honest then queries $k\sqrt{n}$ network nodes (either directly or through a cascading tree) to confirm that none of them have heard of Mallory doing 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 gives $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 gives $99 to Alice. 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 reaches honest nodes).

4. Mallory gives another $99 to Bob 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 (transaction #2). Bob will therefore want to validate transaction #2 and will require Mallory to provide Chuck's transaction history. (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 which is needed 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 #3 where he give $99 to Alice. They raise the alarm of double spending and broadcast a fraud proof. The proof of fraud is two different histories both signed by Chuck.
    - Bob rejects the transaction.
    - Since Alice had reported her transaction we can assume that Mallory is malicious. Mallory and Chuck have their wallets blacklisted and both forfeit their $1 minimum stake.
    - As an extra recommended step, Alice and Bob might compare notes and find all the common nodes they had consulted and make sure none of them failed to report the double spending. If they did that node should also be blacklisted for fraud, with proof of fraud showing that the node received two alternative histories of Alice and in both cases approved them (such approvals being signed by the node evidencing the fraud).

The next morning (or after some hours of lag time on a sliding scale) 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 consensus is reached, future senders need only provide shorter transaction histories back to the newer consensus.

Uptime

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) of say 90% uptime and a node which doesn't comply may receive warnings and eventually financial penalties or disqualification by consensus of all nodes.

Technical details

Variables

- ***n*** Number of valid nodes as of last consensus
- ***k*** A number >1 where we choose a convention of querying $k\sqrt{n}$ nodes for validation
- ***k*** "Effective k" which is *khu*
- **m** Minimum allowed wallet transactions, typically $1
- **M** Maximum allowed transaction (where preferably the maximum wallet balance is also *M+m*)
- ***h*** Proportion of nodes assumed to be honest. Typically, assumed to be at least 50%. The algorithm can work with less but the underlying Blockchain probably can't.
- ***u*** Proportion of uptime required from nodes, typically 90%
- ***v*** The maximum number of inbound transactions a node commonly participates in during the time period between consensus (or more accurately only the inbound transactions which the node depends on for a subsequent spend)
- ***w*** The maximum number of nodes a specific balance of coin commonly circulates through in time period between consensus (where it is only considered circulation of the *n*th transaction was dependent for its balance on the *(n-1)*th). *w* is generally assumed to be small e.g. around 2. It is related to the economics concept of the velocity of money but defined more narrowly.

Glossary

- **Node/wallet** - a computing device that represents the interests of a User acting as their digital wallet storing their public and private key, transactions and balances, and executing their Transactions with other wallets. Also acts as a Node in the network providing validation services to the rest of the network by storing various Transaction Lineages it has seen for other users and raising the alarm if a forked Transaction Lineage is spotted.
- **User** a human or other legal Person such as a company who owns a Wallet.
- **Transaction** A Transaction *t* is an agreement between two User's Wallets to transfer a specific amount of money from one Wallet called the Sender *S[t]* to the other called the Receiver *R[t]* with at a minimum the timestamp, amount, and digital signatures of both Wallets.
- **Consensus Checkpoint** - a periodic (e.g. daily at midnight) checkpoint where all honest nodes reach universal agreement on the transaction histories of all wallets as of that given

time and therefore also of the Balances of all Wallets at that time, although the time at which the **Global Consensus is Achieved** may be some time (e.g. a few hours) later due to the network time involved in running the consensus algorithm.

- **Consensus Latency** Is the lag time from the time for which Global Consensus is achieved, and the time at which consensus is achieved (e.g. in our examples 6 hours from midnight to 6am each day)
- **Transaction History** A chronologically ordered list with all transactions for a certain user u from the previous Consensus Checkpoint up to some point in time.
- **Lineage** (***LIN[t]*** of a Transaction t) is the Transaction History for the Sender of t *S[t]* from the last Global Consensus achieved prior to the time of t, and up to the time of t. The Lineage establishes Sender's balance to cover the Transaction t and should be checked by any honest Receiver *R[t]* before signing Transaction t.
- **Critical Lineage** (***CLIN[t]*** of a transaction t) is a set of transactions and a subset of ***LIN[t]*** (i.e. a list with the same order but with deletions) critical to the balance which allows t. Suppose a Sender S makes a payment of $\$x$ in a Transaction t and suppose S's last consensus balance was $\$y$ and suppose the list of transactions S participated in since the last Global Consensus, *LIN[t]* includes inbound payments $r_1 \ldots r_n$ in descending order of size (and chronologically when equal) and outbound payments $s_1 \ldots s_m$. Now the critical inbound payments are the smallest subset of k inbound payments ***CLIN[t]=*** $\{r_1 \ldots r_k\}$ such that $y - \sum_i^m s_i \ + \sum_i^k r_i \ \geq x \ + \ m$. I.e. $\{r_1 \ldots r_k\}$ are the smallest subset of inbound transactions which are sufficient to provide balance coverage for this payment of $\$x$ given that the Sender has value $\$y$ and 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 $r_{k+1} \ldots r_n$ derive directly or indirectly from fraud) and 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 ***CLIN[t]***.
- **Critical Recursive Lineage** (***CLIN*[t]*** of a Transaction t) - is a set of transactions defined as all transactions reachable on the directed graph whose edges for every t are 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 (which can be stored in a global variable) add also ***CLIN*[*** t_1***]*** to the Set
    - That's it

  This gives a list of all the transactions which ultimately t depends on in the sense that if any one of those is invalidated the transaction t won't be covered (whereas any other transaction in the entire system since the last Global Consensus can be invalidated without invalidating t).

- **Validating a Transaction Lineage LIN[t]** Involves checking that S[t] has signed LIN[t] and that all the transactions t1 in LIN[t] are properly formed and signed and then checking with $k\sqrt{n}$ random nodes that they have not seen any alternative transaction histories from S[t].

- **Transaction Receiver Due Diligence** The checks an honest Node will perform before Receiving and signing a new Transaction *t*. Specifically they will Validate the Transaction Lineage LIN[t] of t and the Transaction Lineage LIN[t_1] for every transaction t_1 in CLIN*[t].

Data Details

A **node/wallet** stores the following data

- Public and private key
- List of all Transactions ever where this Node is Sender/Receiver
- List of valid Wallets on the network and their Balances as of last Consensus Checkpoint
- List of Transaction Histories for various people on the Network including
    - The histories ***CLIN*[t]*** for any inbound Transaction since the last Consensus Checkpoint
    - Any Transaction History which anyone on the Network has asked this Node to validate since the last Consensus Checkpoint

API Details

The APIs of a wallet/node are

- Inbound transaction initiated by Sender node to Receiver node on behalf of Sender's User
    - Sender public key
    - Amount (which must be less than the sender's balance minus *m*)
    - Timestamp
    - *CLIN*[t]* and a full transaction history for each transaction therein
    - Response
        - Approval (may require Receiver's User approval); or
        - Rejection
- Validate transaction for third party
    - Details of Transaction t
    - Recursive list of Transaction Histories ***CLIN*[t]*** which this transaction Sender depends on for the balance to cover the Transaction
    - Optionally a list of further nodes that this node is requested to cascade the request to (in case it is prohibitive for one node to communicate to $k\sqrt{n}$ nodes directly)
    - Response
        - OK; or
        - Reject (in case fraud detected)
        - Optionally responses relayed from cascaded nodes with their signature
- Notification of fraudulent node
    - Public key of fraudulent node
    - Proof of fraud
        - Double spend: two alternative Transaction histories signed by same node

        - Failure to validate: Two alternative transaction histories sent to the node and both approved by node
- Not specified here are the protocols for reach periodic global consensus on transactions and balances

All messages will have a hash digest, sender's signature, and all responses will include the request digest, response data, response digest, and responder's signature.

Before signing a transaction, an honest receiver will do the following

- Receive sender's recursive list of dependent transactions *CLIN*[t]* and a Transaction History for each. Validate that each transaction was properly formed and signed, and had balance to cover it (assuming till the next steps that the histories are honest).
- Randomly choose $k\sqrt{n}$ valid nodes, send each (directly or through a cascading tree of nodes) all the Transactions Histories and ask them to validate that they have not seen any different history for any of the senders of any of the transactions in this recursive list *CLIN*[t].*
    - The nodes validate that they have not seen a contradictory history for any of the same senders. They store every transaction history they are asked to validate for future validation.
- If they have seen an alternative history the transaction is cancelled and the double spending wallet is broadcast with proof to all nodes for blacklisting. Any wallet that could have reported the double spending should also be blacklisted.
- Otherwise the transaction is approved. The same k√n nodes are updated provided the updated history for the sender/receiver with the new transaction

Complexity and security

Message Complexity

While this recursive check may add overhead, in 24 hours a real currency will only change hands at most a handful of times (in fact a typical velocity of real money is changing hands order of magnitude once per month) and therefore this overhead is small. That's why this k-root-n method of preventing double spending is not practical for the long term and it's critical to use this technique only for some hours or days, but eventually as money changes hands many times this algorithm will slow down and it's then important to reach a global consensus and reset the transaction histories.

Thus the message size is bounded by (and probably much less) than $O(v^w)$ where v is the number of inbound transactions a node participates in the time period between consensus (or more accurately only the inbound transactions which the node depends on for a subsequent spend), and w is the number of nodes a specific balance circulates through in the same time period, again only to the extent that it is relied on for a spend. In practice w is likely to be mostly ≤ 2. v can occasionally be bigger, in the case of a merchant who receives many small payments and then uses the aggregate balance to pay a supplier all in the same cycle. Still in most transactions, payers will have balance to

cover their spend as of the last global consensus and there will be no need to recursively validate their inbound transactions at all. So most message sizes will have a reasonable bound measured in say tens of kilobytes including at most a few dozen recursive transactions.

Security and choice of k

We may compute the probability of zero clashes (common nodes) between two random sets of r=$k\sqrt{n}$ nodes to be:

$$p_0(n,r) = \frac{\binom{n-r}{r}}{\binom{n}{r}} = \frac{(n-r)!\,r!\,(n-r)!}{r!\,(n-2r)!\,n!} = \frac{(n-r)\dots(n-2r+1)}{n\dots(n-r+1)} \approx (1-\frac{r}{n})^r$$

For large *n* we can approximate further

$$p_0(n,r) \approx \left(1-\frac{k\sqrt{n}}{n}\right)^{k\sqrt{n}} = \left(\left(1-\frac{k}{\sqrt{n}}\right)^{\sqrt{n}}\right)^k \to (e^{-k})^k = e^{-k^2}$$

By substitution we can see that $\underline{k}$=4.5 gives p~10^{-9} for all large n and for convenience we therefore typically assume $\underline{k}$=4.5 then k=10 to allow for 10% unavailable nodes and 50% fraudulent nodes.

The probability of exactly *c* clashes can be written as

$$p_c(n,r) = \frac{\binom{r}{c}\binom{n-r}{c}}{\binom{n}{r}}$$

Note that even if 51% or more of nodes are dishonest, the system can still work. For example, if *k*=6 and 51% or so of nodes are dishonest we have an effective $\underline{k}$=3 which still gives p≈0.00012 for sizeable *n* and of course we can always pick a higher k. However, the Global Consensus algorithm is likely going to fail with 51% dishonest nodes.

In general $\underline{k}=khu\sqrt{n}$ is the expected number of honest nodes who will respond to a verification request where *u* is the proportion of uptime of honest nodes. Therefore, if *m* is the wallet minimum (and therefore the minimum penalty for being caught in a double spend), *M* is the wallet maximum and therefore the maximum gain from a double spend we have

$$p_0(n,khu\sqrt{n})M\ -(1-p_0(n,khu\sqrt{n}))m\ \approx p_0(n,khu\sqrt{n})M\ -m$$

is the expected payoff from a double spend attempt.

W must design our network so that $m/M\ >\ p_0(n,khu\sqrt{n})$. p_0 is not sensitive to *n* (above a certain number) so typical practical numbers which satisfy the above with a couple of orders of magnitude to spare would be *k=10, h>0.5, u=0.9,* $\underline{k}$*=4.5,* $p_0(n,4.5) \approx (1.5)10^{-9}$ and therefore we might choose *m*=$1, *M*=$100 million.

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.

Suppose again that m=\$1 and M=\$1m and k=10. Suppose there are n honest nodes and the fraudster creates another n fraudulent nodes, for control of 50% , and suppose further that 10% of nodes are unavailable. So, $\underline{k}$=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 they would need to get $\underline{k}<3.5$ to make double spending have a positive expected value. For this the criminal would need about $2n$ fraudulent nodes. But now they have another problem. The loss from a single unsuccessful double spend would be not only be the loss of the double-spending wallet but also the loss of all the fraudulent nodes that failed to detect the double spend namely 6.5^2 nodes.

Consider more generally that a fraudster creates $(f-1)n$ 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 them* or $k\sqrt{(n/f)}$ nodes will be genuine this being a proportion $k/\sqrt{(fn)}$ of all real nodes. Two honest nodes will therefore have an expectation of consluting $(k/\sqrt{fn})(\ kn/\sqrt{f}\ ) = k^2/f$ 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 $p_0(\underline{k}, n) \approx e^{-k^2/f} M$ against an expected cost of $(k^2(1-1/f)+1)m \approx k^2 m$ (the fraudulent nodes who fail to report the double spending +1 for the double spending wallet) for the for every unsuccessful transaction and 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 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.

However, even this strategy is doomed to fail. 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 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 gets a payback tending to M as f increases. However this requires creating O($100n$) nodes to dominate O(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.

In summary we have found that with the right parameters of *k, m, M*, the algorithm is immune to 51% attacks and very resilient to all but the most extreme of Sybil attacks.

Alternative ideas

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.

There could be a scenario 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.

Alternative node selection

Instead of purely random selection of $k\sqrt{n}$ nodes there are some alternative ideas.

- Preferring richer nodes. This would increase the cost of attacking with dishonest nodes. But it would mean that rich nodes have to handle more than $O(\sqrt{n})$ traffic
- An algorithm to select nodes based on buyer/seller's public key, time etc. This would increase the clashes and allow fewer nodes to be queried. But a dishonest node may be able to use this to predict which nodes an honest receiver will consult and they could perhaps engineer a transaction by picking one of many sender keys or one of many time slots in order to avoid the receiver detecting double spending.

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 and so we assume a pseudorandom formula is used 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 spend the money 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 these himself creating less traffic and processing demands on the network.

We therefore propose that when two people do a transfer they must notify a pseudorandom selection of $k\sqrt{n}$ other nodes and get each of their digitally signed approval. The pseudorandom selection is based on a predetermined pseudorandom formula which is known to all and takes as input e.g. sender 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 to pick all 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 receive does not have to trouble the network with validating the recursive transactions.

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 nodes/time to 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 pseudorandom nodes. Of course, if we choose high enough k we can make this infeasible for example with $k=10$ $p_0=3.70E-44$ 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.

Acknowledgements

Thanks Joshua Fox, Benjamin Fox, Aviv Zohar, for their valuable constructive feedback.

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) and 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 some divergence from the estimate, but in the direction of the probabilities being even smaller. Thus, in all cases we can safely ignore n and plan our network based on $p^0 = e^{-k^2}$.

$\underline{k}$	$p_0(n=10^4, \underline{k})$	$p_0(n=10^{10}, \underline{k})$	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

Dryja, J. P. (2019). The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments.

Lamport, L. (1998, May). The Part-Time Parliament. *n ACM Transactions on Computer Systems, 16*, 133-169.

Liskov, M. C. (1999). Practical Byzantine Fault Tolerance. *e Proceedings of the Third Symposium on Operating Systems Design and Implementation*.

Loi Luu, V. N. (2015). SCP: A Computationally-Scalable Byzantine. *IACR Cryptology*.

Nakamoto, S. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System.

Network, S. S. (2019). The Limits to Blockchain? Scaling vs. Decentralization. *Cybersecurity, Privacy & Networks eJournal*.