T.E.T.O DRAFT

Paul Firth
Original author Max Kaye

February 20, 2016

paul@wildbunny.co.uk

Abstract

Trustless Eventual Total Order, or T.E.T.O builds on Satoshi Nakamoto's Bitcoin, allowing individual transactions to be totally ordered in sequence with the same or better asymptotic ordering finality as Bitcoin's blockchain. Further it allows the elimination of mining pools by allowing miners to specify their own difficulty, thereby affording greater decentralisation.

Introduction

The philosophy of *TETO* is that the fundamental goal in p2p cryptocurrencies is to find a trustless, deterministic ordering for all transactions. *Bitcoin*[2] gave us a trustless, deterministic ordering for a *subset* of all valid blocks, and therefore a partial ordering for all transactions - orphaned blocks contain largely redundant data anyway, so very rarely are transactions actually lost. Moreover, the ordering of each block in bitcoin has asymptotic finality, in that the more blocks which get built 'on top' of any given block, the harder the ordering of that block is to change, and that hardness is superlinear in the number of subsequent blocks. *TETO* takes this idea and extends it to work with individual transactions in a one transaction per block configuration. *TETO* blocks have a specifiable difficulty parameter, so the *Longest Chain Rule* in Bitcoin becomes the *Largest Cumulative Difficulty* rule in *TETO*.

TETO utilities *Proof-of-Work*[1] in the same way as Bitcoin as a unforgeable analogue to elapsed time, to prevent against *Sybil-Attack*[3] and to provide an objective measure for ordering blocks in the absence of any clock based timestamping. A key difference from previous work, is that *TETO* utilises a *Directed Acyclic Graph* (DAG from now on) instead of a chain, and never discards branches which would have been orphaned under a blockchain design, which improves the overall efficiency of the system as noted by Sompolinsky and Zohar[4]. This has the implication that double-spends[5] must also be allowed to persist in the DAG rather than being orphaned. These last two points are a critical result of the fact that there are no mining pools in *TETO*, instead every individual user is their own miner (and critically, the *only one* who can mine their transaction) so, all valid transactions must be allowed to enter the DAG to prevent users having to constantly retry sending transactions which would have otherwise been orphaned. Additionally, only one transaction is permitted per block which, we argue allows a lower bound to be placed on transaction acceptability.

Previous work

Satoshi Nakamoto's *Bitcoin*[2] solved the double spending problem by using an objective measure for transaction ordering known as the longest chain rule (LCR), which allows a subset of all blocks to be totally ordered in sequence in the absence of any timestamping service and transactions to have a partial ordering (they are not ordered within blocks). Because each block must contain a reference to the previous block, attempting to alter historical blocks (or to extend a historically orphaned chain) would require an attacker to redo all the PoW from the historical block up to the current head block of the chain, making the design highly resilient to so called long and short range attacks prevalent in *Proof of Stake*[8].

Sompolinsky and Zohar's GHOST branch selection rule is shown to increase the security of Bitcoin by including the work of orphaned branches in the selection of the best block. By doing so, it increases the amount of work an attacker must do to outpace the network. However, because orphan references are not stored within each block, the system is vulnerable to long range attack whereby blocks can be added to historical orphans after the fact which can change contemporary ordering.

Sergio Demian Lerner's *Dagcoin*[6] and Serguei Popov's *Tangle*[7] describe a system whereby individual users mine their own transactions in a similar one transaction per block configuration. Both designs use a DAG of transactions, with no deterministic ordering, therefore requiring a probabilistic confirmation model; since transactions can be spent before they arrive. Neither design features a mining incentive, which we argue is essential for a *Nash Equilibrium* to be established.

Mining eligibility

Only the sender of a transaction can add their transaction to the DAG (or mine) in TETO, this is enforced by requiring the submitted PoW to be signed by the sender of the transaction. Doing this means the transaction cannot be mined by anyone else without being re-signed by the original sender. This eliminates mining pools because private keys of each miner would need to be given to the pools in order for them to operate. We also eliminate the *need* for mining pools as described later on.

TETO's DAG

The constrained structure of TETO's DAG is critical to deterministic ordering. Each block/transaction contains a reference to one parent transaction and optionally, one *uncle* transaction. An uncle is defined as any transaction *from another branch* with a common ancestor which is at a lower cumulative difficulty than the transaction referencing it.

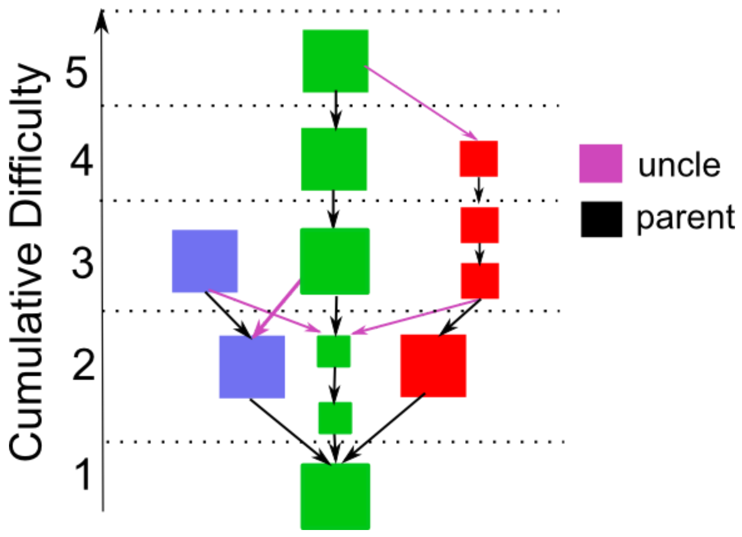

Figure 1

Figure 1 shows three branches of transactions, blue, green and red which have a common ancestor at cumulative difficulty 1. The black arrows show the child/parent relationship and the purple arrows show the child/uncle relationship. Note that not all valid uncles are illustrated, for the sake of clarity.

Unlike traditional blockchain designs, which discard branches should they have a lower LCR score, branches in TETO are allowed to persist, along with any double spends they might contain. This means that there will be multiple head transactions at the same time, unlike in bitcoin where only one head is allowed to persist (although, it can still get orphaned in bitcoin if there is competition between chains).

Cumulative difficulty includes uncles

By including uncles in the cumulative difficulty scoring, TETO enjoys the same increased security as Sompolinsky and Zohar’s GHOST protocol, but with the added advantage that it is not possible to perform a long range attack by altering past history - more on this later.

Eventual total ordering

Ordering transactions deterministically is key to TETO. Transactions are ordered from the genesis transaction to the head transaction with the *highest cumulative difficulty* score; this is analogous to the LCR in bitcoin. The ordering works by following the dependency relationships created by the parent and uncle references. Specifically both parent and uncle transactions must be included in the ordering before the transaction that referenced them. The algorithm recursively orders *parent transactions*, then *uncle transactions* at every node in the DAG. This order of traversal naturally gives ordering priority to direct parents over uncles.

```
def order_from(early_node, late_node, carry=None):
        carry = [] if carry is None else carry
        if early_node == late_node:
            return [late_node]
        if late_node.parent_hash == 0:
            raise Exception('Root block encountered unexpectedly while ordering graph')
        main_path = exclude_from(Graph.order_from(early_node, late_node.parent), carry)
        aux_path = exclude_from(Graph.order_from(early_node, late_node.uncle), carry + main_path) if
 late_node.uncle is not None else []
        return main_path + aux_path + [late_node]
```

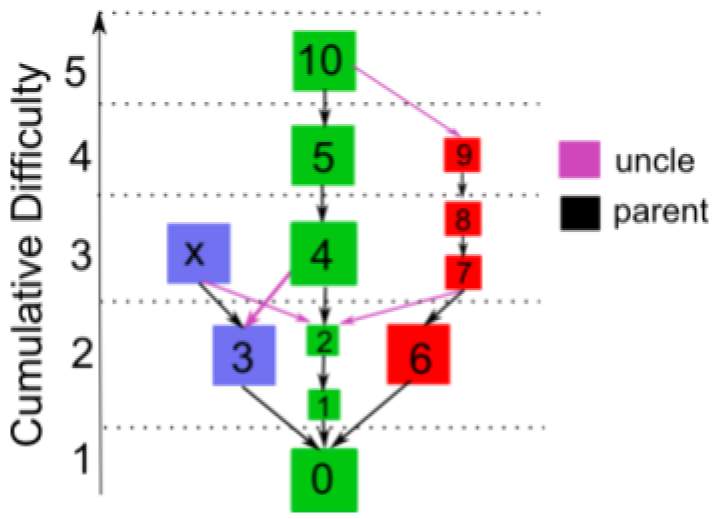

Figure 2

Figure 2 shows an example of the ordering in action. The second blue transaction (x) is not included because it was not referenced yet - more on this later.

If there are several heads with the same cumulative difficulty score, the tiebreaker rule is to chose the head with the lowest numerical transaction ID to order towards.

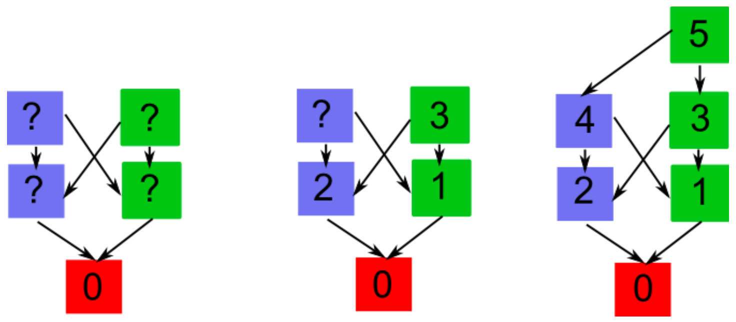

Figure 3

Figure 3 left shows a situation where two branches have the same score. Figure 3 middle shows the resolution by picking the transaction with the lower ID (green in this case). Figure 3 right shows a new transaction gets added to the green branch, which enables final blue transaction to be included (as it was referenced as an uncle of the green branch).

The ordering is a total ordering because every transaction is sequenced numerically from the genesis transaction to the transaction with highest cumulative difficulty. Unlike in bitcoin, where orphaned blocks are discarded, leading to a total ordering of a *subset* of all blocks, in TETO, uncle references *include* orphans in the ordering, at a lower priority than direct ancestors. Ordering is *eventual* because ordering strength increases with every subsequent transaction in the same branch; the more work which gets appended to the branch, the stronger the ordering becomes. Ordering at the head of the tree is subject to a greater probability of change than ordering further down the tree. **<show strength increase of ordering over time>**

Uncles are critical

Uncle references are completely critical to TETO, because they prevent transactions from being inserted into past history, which would have dire consequences for contemporary transactions. It is not enough to have a rule which disallows transactions from being appended to anything but the head with the highest score because, if the only connections between transactions were parent references, there is no objective evidence that any given transaction was originally part of the historical chain; it might have just been appended maliciously to the head of an old branch in order to insert a double spend. This kind of malicious action would only be detectable by nodes which have a full history of transactions. In blockchain designs this is prevented by the requirement for there to be a full chain of blocks containing a reference from child to parent all the way back to any orphaned chain which might have blocks appended to it, meaning any attacker needs to redo all the PoW from the historical point to the head block.

In TETO, uncles are used to act as evidence of historical branches. Indeed, without an uncle reference from a branch which is part of the path of the largest cumulative difficulty to a lesser branch, that lesser branch will *not be included* in the ordering.

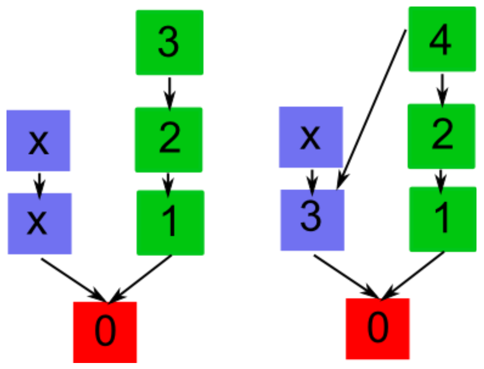

Figure 4

Figure 4, left shows an example where the longest chain has no uncle reference to the blue branch and consequently, the blue branch is not included in the ordering and therefore ***excluded completely from the DAG****. On the right, we see that the best transaction has an uncle reference to the first blue transaction, which thereby includes the lower part of the blue branch in the main ordering. Again, the upper part of the blue branch is not included until it (or any future parents of it) receive an uncle reference from a branch in the main ordering.*

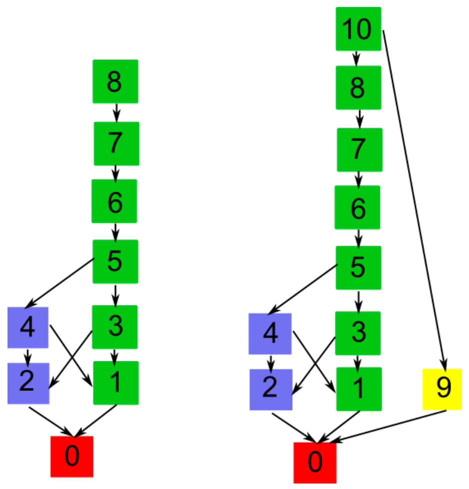

Figure 5

Figure 5 shows situation where the DAG starts off with a longest chain (left) and a malicious user attempts to insert the yellow double spend transaction into past history and includes it in the main chain by creating a new transaction (the best green transaction) with an uncle reference to it. The numbering in the transactions show the resulting deterministic ordering. In order for the attacker to succeed with his past history insertion, the yellow branch would need to be longer than the green branch, which would require extending the yellow branch by the sum of all the PoW in the green branch. The reason for this is that the ordering always goes from the genesis transaction to the transaction with the highest cumulative difficulty; the yellow transaction cannot possibly have the highest cumulative difficulty and furthermore, it is not referenced (as an uncle) by any transactions from the longest branch (since it didn't exist at the time), so it is *only given priority over the final green transaction and none of the others.*

Double spends

Because all referenced branches are allowed to persist in the tree, double spends must be allowed to persist as well. However, even if two conflicting transactions are included in the ordering, the 2nd transaction in the ordering will be invalid and as such it will not be applied. For example, the 2nd conflicting transaction will attempt to spend the same output as the first and would simply be disallowed, but critically subsequent transactions that reference this as a parent will be processed normally as long as they themselves are valid. It is possible that a double spend could lead to a cascade of invalid transactions; but that cascade is limited in effect to the chain of spends of the particular outputs involved in the double spend.

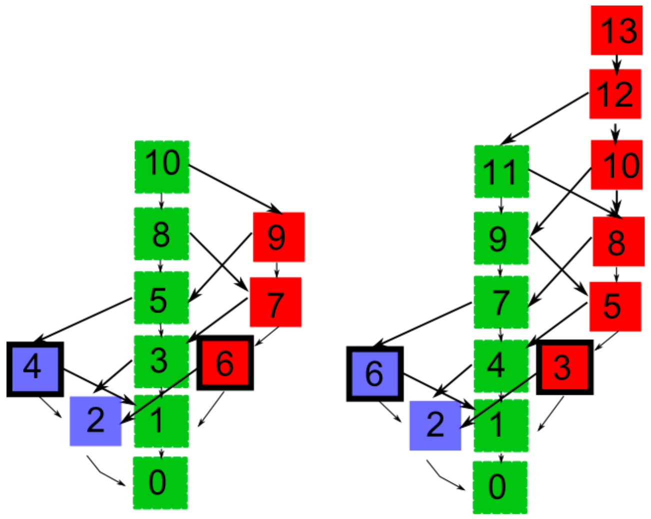

Figure 6

Figure 6 left shows a double spend (the transactions with the dark outline are conflicting) and the resulting ordering which applies the 2nd blue *transaction before the* first red *transaction. Figure 6 right shows the situation some transactions later where the red branch has been extended and has the largest cumulative difficulty, the resulting order of the conflicting transactions changes to favour the branch with the higher difficulty This is equivalent to the LCR rule in bitcoin orphaning a previously majority chain in favour of a new one at a better height.*
<prove it>

Ambiguous double spends?

It is possible for a double spend to exist at the same cumulative difficulty as the legitimate transaction.

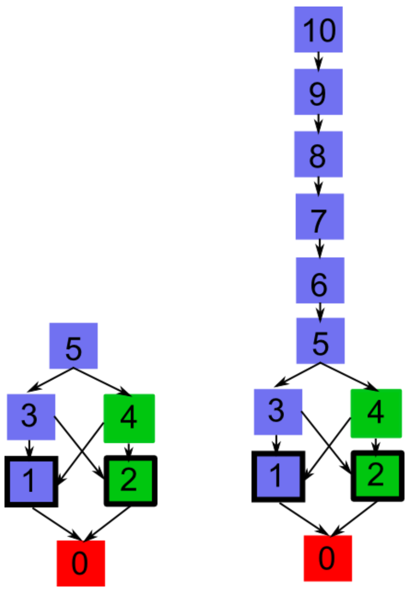

Figure 7

*Figure 7 left shows a double spend (*green order 2*) which has a very close numerical ordering to the legitimate transaction (*blue order 1*). The top blue transaction favours the blue branch by having it as a direct parent, and the green branch as an uncle, which is what gives priority for the legitimate transaction over the double spend. Figure 7 right shows the situation some time later as more transactions have been appended to the blue branch. Although the ordering of the legitimate transaction and the double spend remain very close numerically, the risk of the ordering being reversed decreases with every subsequent transaction, and in fact, would require PoW expenditure of the sum of all the blue transactions.*

Temporary partitions

It is possible, due to network latency that a temporary partition could develop where different parties observe a different *best transactions* which they order towards.

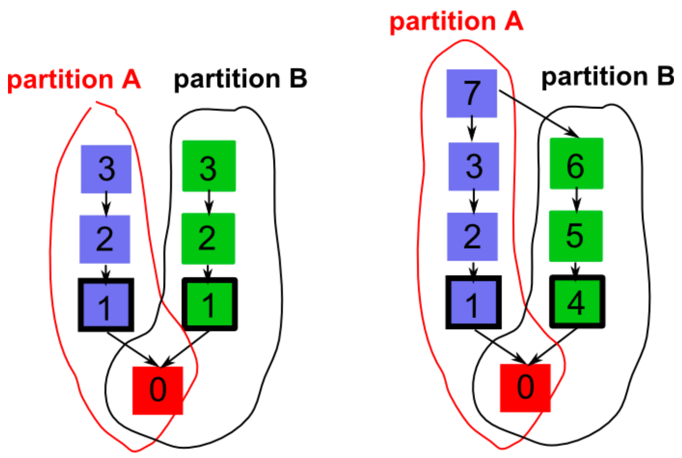

Figure 8

*Figure 8 left shows two partitions of the network which appear to different observers each separately containing what appears to be the best transaction. Figure 8 right shows an attacker using his knowledge of both branches to orchestrate a double spend by placing conflicting transactions (*outlined in black*) in both and waiting for a merchant to accept one of his before uniting the two branches by connecting them together with an uncle reference and therefore solidifying the ordering in favour of his prefered spend (*blue order 1*).*

However, because TETO's miners are incentivised to include uncle references (more on this later), the attacker has to rely on the rest of the network either not transacting at all during his attack, or having no knowledge of both branches. Once the branches are united by an uncle reference the attacker is reduced to attempting to outpace the network to change the ordering in his favour.

Mining incentive and the Nash equilibrium

The security of the TETO network depends upon the DAG converging and not diverging. Specifically, it must be optimal for any given miner to increase the length of the DAG and not the width. To this end, TETO only pays a mining subsidy down the path of the direct ancestors of the best transaction, and not down the path of the uncles. Doing this has the same effect for the mining subsidy as orphaning branches in Bitcoin does, except that only the subsidy gets 'orphaned' and not the other transactions in the uncle branch.

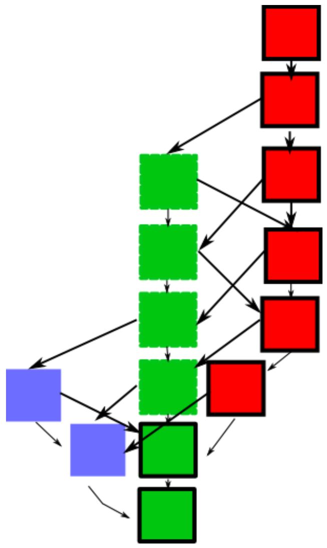

Figure 9

Figure 9 shows the direct ancestors (outlined) on the path of largest cumulative difficulty, which are the only transactions which are awarded their mining subsidy.

The overall security of the DAG decreases linearly in the number of partitions, so it is essential that the DAG in TETO be as narrow as possible at all times, however this is at odds with the rational behaviour of individual miners, because although including an uncle reference (to a previously unreferenced uncle) narrows the DAG, doing so leads to less timely confirmation for the transaction at hand, since the uncle reference is an indication that the uncle be ordered *before* the transaction referencing it.

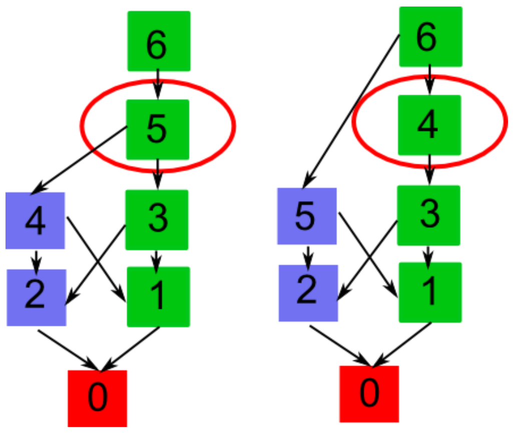

Figure 10

Figure 10 left shows a situation where the circled candidate transaction includes an uncle reference to the best blue transaction, which gives ordering priority to the uncle over it. Figure 10 right shows the situation when the candidate transaction doesn't include an uncle reference, leaving the subsequent transaction to include one - notice the resulting numerical ordering favours this behaviour? This implies that for timely confirmation, transactions would be advised not to include uncle references, but this behaviour is detrimental to the overall DAG, because uncles cause the DAG to converge.

It is clear that the network needs to incentivise uncle references and thanks to the way the cumulative difficulty scoring *includes* uncles, this is in fact the case. By including an uncle reference (to previously unreferenced uncle) from a transaction hoping to earn a mining subsidy, that miner increases the chances his transaction will remain in the path of highest cumulative difficulty, because his cumulative difficulty score will then include the difficulty of the uncle branch. This aligns the rational behaviour of the individual miner with behaviour that benefits the entire network, thus establishing the critical Nash equilibrium.

Selectable mining difficulty

Miners in TETO can choose the difficulty of the PoW they solve from anywhere between two bounds:

- Base difficulty, designed to double every 18 months, in line with Moore's law [10]
- Max difficulty, adjusting to keep issuance at a constant rate

The reward for solving the PoW at the chosen difficulty scales linearly with the chosen difficulty upto a maximum of N coins **<define N>.** This enables both normal users of the currency and professional miners to participate at the same time. Base difficulty is set so as to require X **<define X>** seconds of computation on a contemporary CPU **<give example>**.

Selectable mining difficulty eliminates the need for mining pools by reducing the variance in mining rewards. This also enables applications such as websites, or computer games to monetise their users by solving tiny difficulty PoWs to generate a reward paid to the site/game owner.

Difficulty **is the expected number of hashes required to solve a given PoW. The number of hashes required to solve 10 blocks with difficulty 2, is the same on average as the number of hashes required to solve 1 block of difficulty 20.**

Let each hash be an independent Bernoulli trial *x*. Each trial has an expected probability of success *p* of *1/d*, where *d* is the chosen difficulty, which is also the expectation *E*. The expected number of trials until *X* success *E(X)* is a negative binomial random variable. Recalling that expectation of a sum of random variables is the sum of their expectations:

$$E(x) = \frac{1}{p}$$

$$E(X) = \sum_{i=1}^{X} \frac{1}{p} = \frac{X}{p}$$

$$p = 0.005$$
$$X = 1$$

$$E(X) = \frac{1}{0.05} = 20$$

$$p = 0.5$$
$$X = 10$$

$$E(X) = \frac{10}{0.5} = 20$$

Bounding transaction acceptability

In contrast to PoW blockchains, such as Bitcoin, it is possible to place a lower bound on when it is safe to accept a transaction as confirmed in TETO by using the simple intuitive rule which says that a transaction is confirmed when it becomes 'buried' under an amount of cumulative difficulty equal in cost to the value of the transaction.

By forcing transaction senders to submit PoW solutions with a difficulty proportional to the number of transaction outputs, we reduce the problem space of the solution compared to blockchain designs because the attacker must expend more work to attack more victims simultaneously within one transaction.

Because mining is a zero sum game overall, the *electricity cost of any given PoW solution is equal to the mining subsidy on average*, so as long as a given transaction is buried under an amount of cumulative difficulty for which the subsidy of solving a PoW would be greater than the value of the transaction, any rational attacker would be better off just taking the mining subsidy instead of attempting to outpace the network in order to double spend.

Let *r* be the probability that an attacker can pull this attack off, *B* is the subsidy at stake during his attack and *v* is the value of the transaction he is attempting to double spend. A merchant is safe to accept a transaction as long as:

$$v < B(\frac{1}{r} - 1)$$

If the attacker doesn't pull off his attack by definition he will not have replaced the best transaction and as such will lose his mining subsidy, making the attack an overall loss due to the way the subsidy pays out by direct ancestors only.

The bound is a lower bound because the variance in PoW solution times and the efficiency of scale of different mining operations will introduce a margin of error.

Conclusion

We have presented TETO, which to our knowledge is the first trustless solution to the total ordering problem for transactions in a p2p system. We have shown that the ordering has asymptotic finality and that the system as a whole achieves a Nash equilibrium. TETO also overcomes limitations of previous work particularly in achieving resilience to long range attacks. Mining pools are also eliminated by enforcing the rule that only the sender of a transaction can mine their transaction. Allowing miners to select their own difficulty eliminates the *need* for pools by reducing the variance in subsidy. TETO is the first DAG based cryptocurrency to robustly support a mining subsidy that we are aware of. TETO is also the first cryptocurrency to place a clearly defined lower bound on transaction acceptability.

Future work

Analyse the behaviour of TETO in an environment where the mining subsidy is replaced by transaction fees alone. Explore the computational cost associated with the ordering process. Consider the existence of light network nodes, and or pruning of the DAG.

References:

[1] Jakobsson, Markus; Juels, Ari (1999). "Proofs of Work and Bread Pudding Protocols". *Communications and Multimedia Security* (Kluwer Academic Publishers): 258–272.

[2] Satoshi Nakamoto (2008). “Bitcoin: A Peer-to-Peer Electronic Cash System”

[3] Douceur, John R. (2002). "The Sybil Attack" *International workshop on Peer-To-Peer Systems*.

[4] Yonatan Sompolinsky,Aviv Zohar (2013) “Accelerating Bitcoin’s Transaction Processing Fast Money Grows on Trees, Not Chains“

[5] Mark Ryan (2010). "Digital Cash". *School of Computer Science, University of Birmingham.*

[6] Sergio Demian Lerner (2015). “DagCoin” https://bitslog.files.wordpress.com/2015/09/dagcoin-v41.pdf

[7] Serguei Popov (2015). “The tangle” http://188.138.57.93/tangle.pdf

[8] Andruiman (2015). “Forging algorithms: multi-strategy forging and related security issues”

https://github.com/ConsensusResearch/articles-papers/blob/master/multistrategy/multistrategy.pdf

[9] Ittay Eyal and Emin G¨un Sirer (2014). “Majority is not Enough: Bitcoin Mining is Vulnerable“
http://www.cs.cornell.edu/~ie53/publications/btcProcFC.pdf

[10] Gordon Moore (1965). “Moore's law” https://en.wikipedia.org/wiki/Moore’s_law