Storm

Using weak proof-of-work for instant confirmations on Bitcoin Cash

awemany

August 30, 2019

Abstract

This describes an enhancement of the weak blocks idea to create a system capable of merging multiple weak blocks, hereby termed *delta blocks*. It describes in detail the workings of the proposed scheme, its benefits and its limits and finishes with a detailed simulation and prototype implementation that are both being made available with the publication of this document.

Contents

1 Introduction

One of the areas where there is a perceived need for improvement in Bitcoin is its ability to do safe, quick confirmations of transfers. This is the so-called instant transactions problem. The Bitcoin Cash (BCH) blockchain is referred to simply as Bitcoin throughout this document.

The author has worked on ways to improve instant confirmation security before, such as the Zero Confirmation Forfeit (ZCF) scheme[5] and believes that this approach, viable since the introduction of the OP_CHECKDATASIG/-VERIFY opcode, in conjunction with other proposed schemes such as Double Spend Proofs[4] is actually good enough to solve almost all needs for safe instant transactions in Bitcoin, especially for the common case of its use as a daily payment/transaction system of relatively small amounts. It would get us most of what we might need. It only needs some infrastructure work to make it transparent to the average user.

However, there are more comprehensive and general proposals trying to address the instant confirmation problem: At the Instant Transactions Workshop in Italy where ZCF was initially presented, a first preliminary look at the Avalanche pre-consensus system was presented as well[7, 8]. Resonance regarding Avalanche was positive, though with the impression that more time needed for analysis by many, as it touched a lot of critical areas. For instance, how Avalanche with its own consensus mechanism would interact with the existing POW system had not been sufficiently explored. One major problem that cropped up later if relying on past POW for deciding on blocks through Avalanche, and implementing Avalanche as a proper enforced (soft) fork, is that this could allow someone who has the resources to temporarily overpower the hash power of the network (something that has been demonstrated as a currently realistic concern during the November 2018 split) take the network over through the Avalanche mechanism.

After further thought, this hints at there being a fundamental problem with any preconsensus scheme that does not rely on the very same consensus mechanism as Bitcoin right now: *Current* POW.

It is certainly good to see that some proponents of Avalanche see this problem with past POW now[1] and it is a sign that those working on BCH are aware of and working in line with the overall incentive system of Bitcoin. Now, there may be ways to fix Avalanche with a different consensus scheme that appears not as directly gameable or less gameable than past POW. But this failure motivates one to look for alternative solutions to the instant confirmation problem.

It seems especially worthwhile to explore the idea of *weak blocks* further, as any scheme that uses current POW very naturally leads to weak blocks as the solution of the instant confirmation problem. Bitcoin is steered by POW, and if one does not want to change its overall structure and incentive system, partial solutions to the POW problem are *the* natural way to get faster responses from the system.

In the past, the author has worked on implementing the weak blocks scheme as envisioned by Peter Rizun[9] as a pull request to the Bitcoin Unlimited full node implementation[6].

One criticism directed at weak blocks is that without a consensus level enforcement of weak block validity, the remaining incentives (of better network propagation) are going to be too weak to be followed by miners for reliable instant transactions. Another criticism directed at weak blocks is that the weak block emission time is limited to something on the order of dozens of seconds, if one wants to avoid the mutual orphaning chaos caused by faster emission times. This rate however, is too slow to be really useful for the instant transaction regime.

To address these two more major concerns, an update to the weak blocks idea is therefore proposed. This update is named **Delta blocks**. Realizing the need for a catchy name and being inspired by Avalanche as a weather phenomenon in the most abstract sense, I hereby name the overall application of delta blocks to Bitcoin the **Storm** system.

2 Acknowledgements

First, I want to make it very clear that I did not invent the idea of merging weak blocks. For reference, the github user `duzy` (Duzy Chan) has pointed to this idea in my original pull request for weak blocks to the BitcoinUnlimited implementation[1].

Reddit user `cryptocached` was to my knowledge one of the first to clearly articulate and point out the problems with relying on past-POW in Avalanche, the problem which motivated to take a second look at my former weak blocks work.

What I merely claim is having thought this through in the context of Bitcoin's POW system, working it out in detail and maybe adding an original thought here and there. Perhaps I also should have done a more thorough literature review as there might be parallel developments in other coins. Pointers to relevant other work and constructive input are very welcome!

Special thanks go to Josh Ellithorpe, Peter Rizun, Andrew Stone, Andrea Suisani and Peter Tschipper for valuable input, review, fixes and clarifications.

I also like to thank Bitcoin Unlimited for some of the computational resources used for the preliminary full node test in Section 6.

[1] See `https://github.com/BitcoinUnlimited/BitcoinUnlimited/pull/856#issuecomment-351266248`

3 Overview

In the bird's eye view, delta blocks differ from weak blocks in one key aspect. Instead of having a single ancestor, like weak blocks do, they refer to any number of compatible blocks as ancestors, and result in the *merging* of the transaction sets of these compatible ancestors. On top of this merged set, a delta set of transactions is added, giving the blocks their name. Once deltablocks are implemented, they would cause a (relative) storm of deltablocks in the network, underlining the appropropiateness of the naming for the overall approach.

In the following sections, the specification of delta blocks and the thoughts that lead to them will be detailed. It will be too informal to be considered a definite specification, but it can certainly serve as the basis for that. Followed by this is a simulation of the supposed properties of delta block to make the case for them, which precedes a look at the prototype implementation made available with the publication of this document as a pull request to Bitcoin Unlimited.

3.1 Weak block

A *weak block* is a Bitcoin block that has less than the full POW required to be considered a valid (Strong) block by the Bitcoin network. Due to the nature of proof-of-work calculations, weak blocks are an unavoidable side product of mining for strong blocks, as partial solutions to the POW problem will appear with a frequency that depends inversely on the strength of the POW that a a weak block must meet. As they are a side product they cost the miners chasing a strong block nothing except additional communication to publish them to the network. This ties weak blocks intimately to the strong blocks which are part of the foundation of Bitcoin's incentive system. Producing weak and strong blocks is essentially one and the same operation and they are inseparable from each other.

A weak block in the sense of Rizun's paper can be tweaked to have a single link to its weak parent, which could be placed in the coinbase such as done in [6]. This refers to the parent weak block plus the differential of transactions that has been added since the last weak block, thus cutting down on network bandwidth and giving the miners an incentive to implement and follow them. It also allows to more easily identify the chains of weak blocks building on top of each other, allowing to easily implement the necessary notion of longest chain of weak proof of work.

It should be noted that a block being weak and a block having a link to the weak parent are orthogonal properties of blocks. If a miner finds a strong block in a weak blocks-enabled Bitcoin and mines using such a coinbase link, the strong block will also benefit from this differential transmission scheme.

Weak blocks of this kind have the downside that since the weak blocks are generated in parallel in the network, proof of work (POW) is going to be wasted in the weak block that will not make it into the weak blocks chain. This, however, can largely be remedied by going from the tree-like structure of weak blocks to a full directed acyclic graph (DAG) based approach:

3.2 Delta block

A *delta* block is defined to be a data structure that contains a block header, a set of weak ancestors and a set of transactions named the *delta set*. The POW in the header can be weak or strong.

In the current implementation, this set of ancestors is stored in the coinbase in the form of `OP_RETURN` data containing the weak ancestor block hash pointers.

It should be prominently noted that this set of ancestors is different to the `hashPrevBlock` pointer that is in the block header and refers to the previous strong block! The `hashPrevBlock` hash, other than being required to match for a given delta block, has no further bearing on the delta/weak blocks logic and is therefore mostly ignored in the discussion here.

Network representation The name *net(work) delta block* refers to the representation of the change set of an individual delta block on the network, which is optimized for transmission by leveraging the Graphene set reconciliation protocol available in BU.

3.3 Merged block

A delta block can be assembled from all its direct and indirect ancestors into a weak or strong block with a `CBlock`-like structure. This is done by merging the transactions from all referenced ancestors into a single set, dropping duplicates. The merkle root in the delta block's header corresponds to this merged transaction set. The resulting Bitcoin block is hereby aptly named the *merged block*. The coinbase from the most recent (tip) delta block is taken as the merged block's coinbase, and all other coinbase transactions are dropped.

It is required that the delta set contains neither duplicates with any of the referenced ancestors nor double-spends. It is also required that different delta blocks are *compatible* with each other, meaning that they might refer to the same transaction multiple times, but do not contain any doublespends.

Fig. 1 shows this relationship between delta blocks, merged blocks and their `CBlock`-compatible representation in Bitcoin.

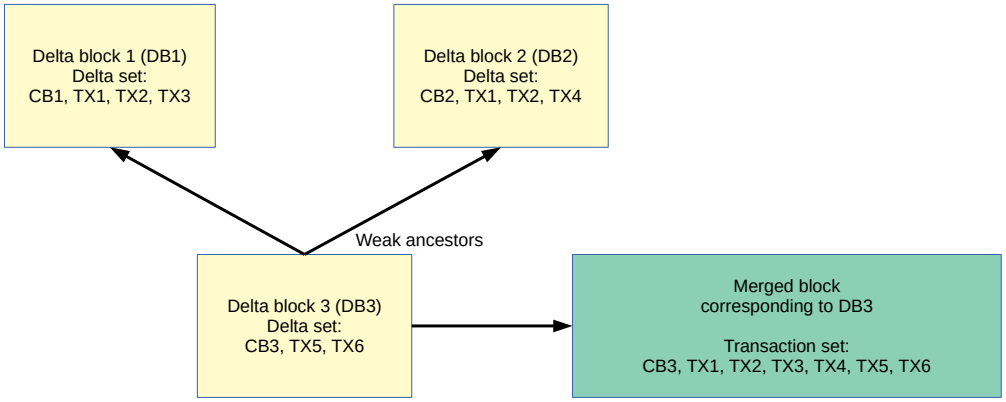

Figure 1: Simple exemplary scenario showing two concurrently mined delta blocks DB1, DB2 being referenced by delta block DB3 and the resulting merged block(s).

With CTOR transaction ordering being active on the BCH blockchain, the method of merging transaction is fixed and does not need to be further specified in this context,

however for further discussion on this topic, see below.

On set removal It should be noted that it does not make sense to implement any scheme that would allow operations that undo or remove transactions in weak block descendants. The Bitcoin miners operate a distributed time-stamping system, and *one cannot unwitness* the existence of a formerly confirmed and otherwise referenced transaction hash. The definition of delta blocks assumes that it makes sense to allow for independent duplicate confirmation of the same fact but not retracting former confirmations of a fact.

3.4 Longest weak chain

Weak blocks needed a notion of longest chain which is a straight forward extension of the strong blocks POW concept, with the caveat that there is no difficulty adjustment within a weak blocks chain (from strong to strong block) and thus each weak POW (WPOW) can be considered a unit of the same one POW for longest chain measurement purposes, equating the concepts of longest chain and greatest difficult in this context.

For any (upwards-pointing) tree of delta blocks in the delta blocks DAG, the total WPOW (of a merged block) is calculated as the size of the set of all nodes that are direct or indirect ancestors of the delta block in question, not counting repeat delta blocks and requring that all delta blocks cross a minimum WPOW threshold. In other words, it is the cardinality of the tree set referred to by the respective tip. Fig. 2 illustrates this in more detail.

3.5 Longest chain and miner selection

The above definition will naturally result in zero or more delta blocks that have a longest chain property in the sense of belonging to the set of delta blocks that have maximum WPOW. Those are the delta block chain tips (in short: *delta tips*), with the quite likely situation of there being competing delta tips of same WPOW. It is assumed henceforth that the delta blocks discussed here are built on the same strong block chain fork and therefore have the same `hashPrevBlock`. Note that delta tips for multiple strong block chain forks can be tracked.

A miner that wants to start mining using delta blocks can pick any of the delta tips known to him (or start from scratch from the last strong block), merge them into a new delta block template, set the delta set to the transactions he wants to additionally include and start looking for solutions to the POW puzzle.

3.6 POW measurement and potential consensus rule changes

To address the other main concern with regular weak blocks from above, which is the lack of incentives to follow and build them, it is hereby proposed to change the POW measurement rule in Bitcoin from the chain of most cumulative strong POW to one where weak POW is used to break the tie of a strong block race by selecting the chain which has more WPOW in that situation.

If delta blocks are preferred in this way, a miner has an interest to select and merge those delta tips that will make the resulting mined weak block the longest chain in the above WPOW chain length metric. A miner furthermore has an incentive to share his (or her) weak block as it will make sure that the network tends towards his (or her) reading of the situation in case there are conflicting transactions. As s-/he does not necessarily know in each instance and at each point in time whether a transaction will

be double-spent, s-/he has an incentive to preventatively argue for his variant of reality by publishing a found weak solution to the POW problem.

However, should this incentive turn out to not be strong enough, a more direct incentive could be - and this is rather a discussion than a proposal - reward sharing amongst WPOW blocks. The approach would be to add another soft fork that demands a miner share his block reward among all the delta blocks that make up the strong chain tip that s-/he found.

It should be without question that any consensus change should only be implemented with strong support from the userbase, developers and general members of the ecosystem and in case a more comprehensive instant confirmation solution is deemed really desirable.

As a side note, a potential further tweak to a delta block scheme could be to estimate the desired delta block difficulty and the WPOW threshold dynamically and adjust it independently from the strong blocks difficulty.

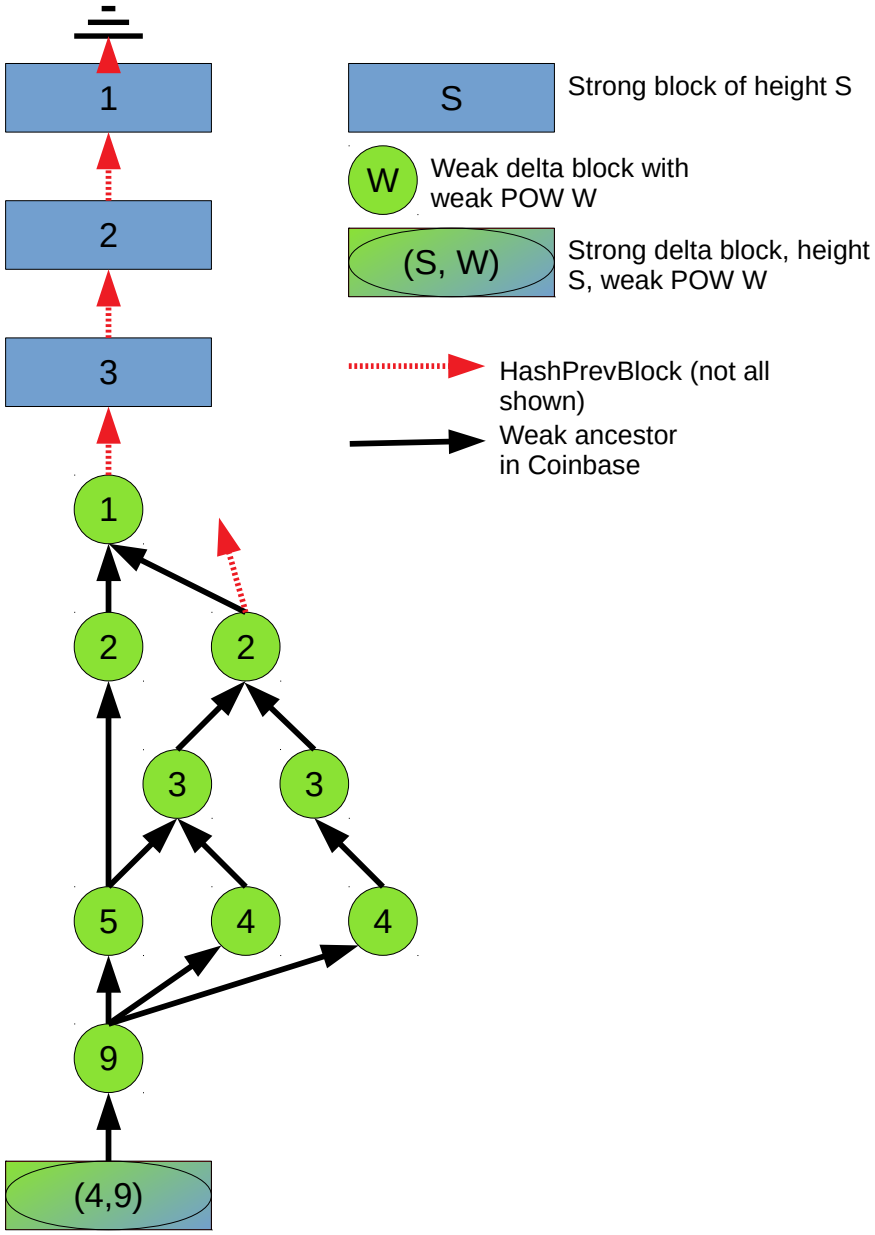

Figure 2: Relation of the weak and strong block chains / trees in Storm.

4 Implementation details

The author believes that the only way to get a complete picture on the potential pitfalls and roadblocks for a moderately complex change is to code a prototype and test and review it. What follows are the implementation details of the above high level scheme for a prototype implementation on the Bitcoin Unlimited full node software.

Unmanageable runtime or space complexity increases, as well as code complexity, would be a potential "show stopper" for this scheme and the expected complexity of the most important heuristics and algorithms in the Storm system is discussed below.

4.1 Persistent data structures

As weak blocks are strong blocks except for their proof-of-work measure, it follows that merged blocks have to still behave, in addition to the proposed stacking of smaller delta sets, like ordinary `CBlock` data in the node implementation. Most importantly, the full view of the set of constrained transactions, and not only the independent (net) delta block representation needs to be available in the full node software.

A naive way to construct merged blocks internally would be to keep and update the full transaction set referred to by each delta block. This, however, is prohibitively costly in terms of memory as well as processing time, as the time as well as space required for each merged block would be on the order of a full block, thus creating space and time requirements of the order of $O(N^2)$ (for N transactions).

A much better way is to create a persistent data structure[12], in this implementation a persistent binary tree to store a view of the full transaction set for a merged block, but share as much as possible with other merged blocks. In the current prototype/demo implementation, this is done by using the tree that is the biggest ancestor (in terms of numbers of transactions) as the base for building a given merged block transaction set and only adding the missing delta sets on top. This reduces space (and time) complexity of delta blocks to something managable, about $O(N \log N)$ in the big picture. Details on this approach are given below.

The persistent data structure has been implemented as a template container persistent binary tree in `persistent_map.h`, with a very loose adherence to the C++ `std::` container data types. Note that no balancing of this tree has been implemented and if it or a similar one is extended to store Merklix data as is, is also not desirable.

To have a better picture of how persistent data is used and reused in the prototype implementation, see Fig. 3.

4.2 Merklix trees and transaction order

For the Bitcoin ABC implementation, their lead developer Amaury Séchet has proposed a road map to generalize the current Merkle tree that is used to encode a given transaction set in a cryptographically secure way to "Merklix" trees[2], which can be formed in any order and can (and generally will) be unbalanced. This should allow $O(\log N)$ access operations on the tree and would make efficient merging of competing weak blocks possible.

Alternatively, without Merklix trees available, it might make sense to generalize the canonical transaction ordering (CTOR) to any order, as to allow to stack partial Merkle trees together in an efficient way.

It should be duly noted here that, in the current demonstration implementation, the whole merkle tree is recalculated for a delta block, which is $O(N^2)$ for N total

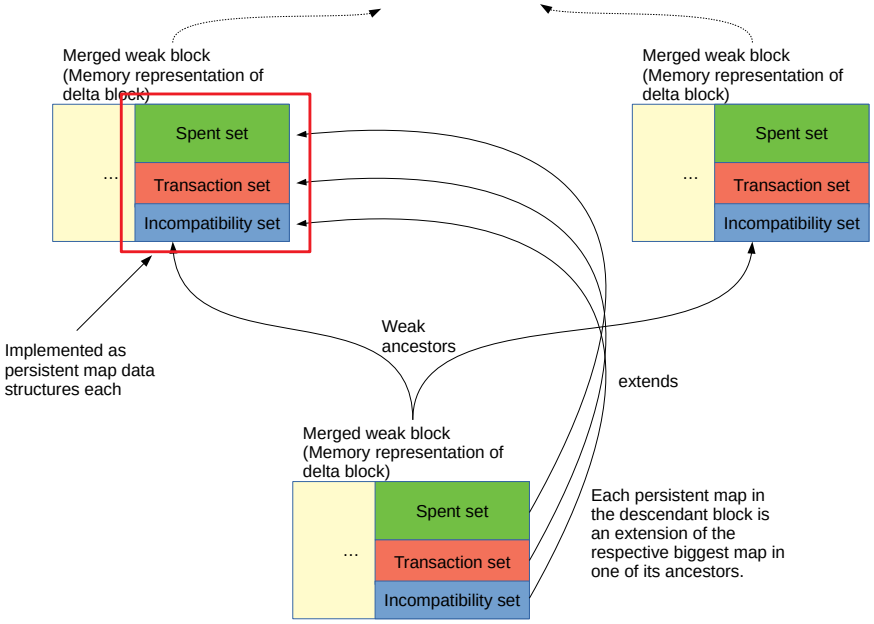

Figure 3: Current use of persistent maps in the prototype implementation. (The incompatibility set that is shown here, that would track mutually incompatible weak blocks, has not actually been implemented yet). An additional option might be to make the mempool a (persistently shared) part of a corresponding merged block.

transactions and thus not sustainable in the long term. With a better approach, it seems straight-forward, however, to reduce this to $O(N \log N)$.

4.3 Compatibility (merge) checking

As per the above definition of being able to merge two delta blocks, the individual transactions from their respective transaction sets have to be checked for compatibility. To save work, it seems best to test a delta set for compatibility against an already merged block. In case of delta blocks and net delta blocks, it is thus a good idea to test a delta set vs. the full transaction set of a block. With an efficient $\log N$ - way to check for spend outputs by other transactions available (see below), testing compatibility for a delta set of size n vs a bigger set of N transactions in the merged block then becomes $O(n \log N)$. To allow this efficient checking, a spend-index for each delta block is kept as well, which maps each spent transaction output to its respective spending transaction ID (TXID). This spend index is shared in the same way as the transaction set using the persistent binary tree container among multiple delta blocks. Only spends since the last strong block are tracked in this partial spend-index.

Now, potentially multiple delta tips have to be tested against each other and they might contain multiple independent delta sets, which is of square complexity still. For d delta sets, n transactions per delta set and N transactions per strong block, a total of $O(d^2 n \log N)$ tests need to be made.

A persistent map-based approached can be further used to efficiently keep track of the mutual compatibility of delta blocks with each other as well, though this has not been implemented yet.

Excluding the problem of incompatibility due to double-spends for a moment and with all miners following the expected incentives, they will greedily merge delta sets that are floating around (to achieve most WPOW for their mined block), so that each emitted block by a miner will collapse the available delta tips to just one by merging them all. The number of different delta sets d to check for a merging, can thus be expected to be on the order of the network diameter in time L, divided by emission time t of weak blocks. This means that the total complexity for merge testing above becomes $O\left(\left(\frac{L}{t}\right)^2 n \log N\right)$. This clearly shows that there is going to be a superlinear strain on the network that increases with higher weak emission rates. Fortunately, it seems that the emission rates for a good customer merchant experience (seconds) are similar to the network diameter (seconds), so this squaring behavior is not expected to be a large problem.

Miner adversaries Although simulations of well-behaved miners as well as basic intuition say that there is a probabilistic collapse of the delta tree to a single ancestor every so often, it is not at all a given that this is going to be the case under adversarial conditions. For the time being, it can be expected that attackers can overpower the BCH chain in the short term, and the implementations need to deal with that (however, the current demo implementation does not!). An attacker could, for example, create a lot of mutually incompatible delta blocks, creating a situation where mutual testing would be of square complexity. A relatively straight forward way to address this, however, should be to implement a heuristic at a miner that only looks at the top couple delta tips for potential merging and a node that receives delta blocks checks them with parallel validation and maybe even deprioritizes blocks that have a lot of ancestors. This way, blocks that are intentionally made like this will have a very hard time to win the beauty contest while not overly impacting full node operation. This would be similar to the way BU principally addressed squaring complexity in some aspects of block validation.

Double-spending adversaries The problem of a miner including double-spending transactions in a delta block is handled similarly to the above miner-adversary scenario. If a delta block cannot be merged with what the majority thinks is the otherwise valid delta DAG, it will have correspondingly high risk of being orphaned.

Also, miners could develop tools to assess the likelihood of doublespends over time. For example, miners could go and predict double-spending likelihood from past behavior of given outputs.

The best strategy for a miner to decide on whether and when to include a transaction is out of the scope of this document. It seems that this is might be an area where a good game theory approach could make sense, but like all potential areas for application of game theory, it could also turn out to be simply too complex in the real world to approach it this way. It is also an area where miners can and should compete to find good solutions.

That all said, a way for a miner to approach the problem would be to simply wait a certain while before including a transaction in a (delta) block. A constant wait is discussed in the simulation section.

Randomizing this wait would make it hard for a double spender to time his or her double spend. A simple example function that decides inclusion wait time and as a function of transaction fee would be to select a wait time from an exponential distribution

(plus potentially a per-miner constant) that has an expected value simply proportional to $\frac{1}{\text{TX fee}}$, chosen at the miner's discretion and in a competitive way, optimizing his or her income. It seems appropriate to use an exponential distribution corresponding to a memory-less process, giving any potential adversary the least amount of information here.

Notably, the detailed fee structure and complex dynamics in this regime also further strengthens the idea of a natural fee market.

4.4 Network messages

Currently, solved delta blocks are send in a fully expedited mode, which simply means that a full net delta block is sent to all connected nodes whenever it is found, and it is forwarded on all links except the receiving one when it is received the first time. This reduces the expected delta blocks latency by up to a factor of two compared to a `inv, getdata`-style approach of announce- and request rounds as is currently largely used in `bitcoind`. But this approach is likely prohibitive in terms of bandwidth used when used on the real network.

One area which is out of the scope of this document and which will likely considerably improve all (including delta blocks) of the various "instant" confirmation approaches is the implementation of expedited (see also section 4.6) transmission modes for delta blocks as well as transactions.

Two message types for delta block propagation are defined at the moment. One is `CNetDeltaBlock` which is the serialization of a delta block, meaning the set of transactions in the delta set plus coinbase with weak ancestors and block header. See Fig. 4.

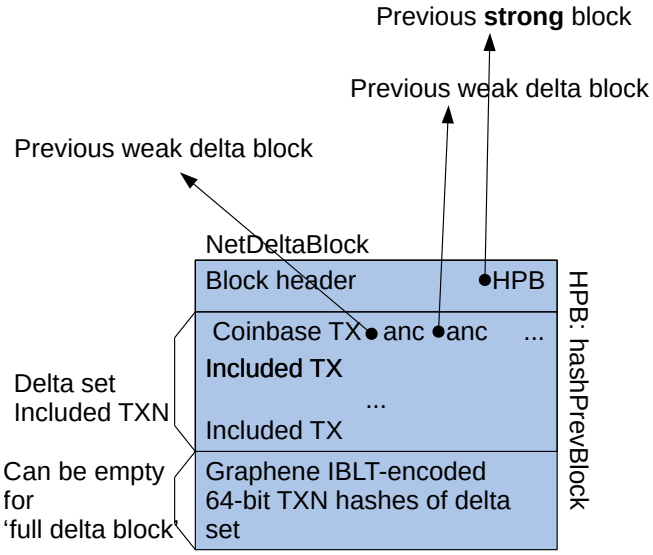

Figure 4: Format of a `NetDeltaBlock` network message. `anc` denotes pointers to weak ancestors and `HPB` is the `hashPrevBlock`. The included transactions are either included in full or as part of the IBLT-encoded short-ID graphene set.

The delta set of transactions is optionally (and by default) graphene encoded, meaning that a Bloom filter + IBLT combination is used to transmit short 64 bit hashes of the included transactions. Alternatively, transactions can be sent in full. The list of full transactions or the IBLT section of the message are allowed to be empty.

In case there are missing transactions in a net delta block or a net delta block is missing, a `CNetDeltaRequestMissing` message is sent (Fig. 5), which includes the corresponding hash of the delta block and any missing short transaction IDs. Alternatively, the requested list is empty to indicate requesting transmission of a non-graphene-encoded `CNetDeltaBlock`.

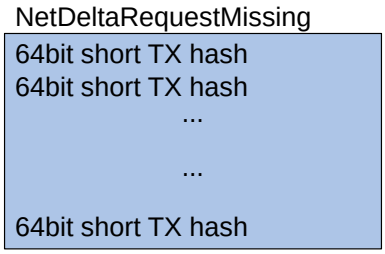

Figure 5: Format of a `NetDeltaRequestMissing` network message.

4.5 Transaction confirmation lookup

One of the operations potentially taking up a lot of time is the tracing of transactions and the lookup of the number of weak confirmations a transaction has on one or all delta block chains. The straight forward implementation is to check all delta blocks in the DAG to see if it contains the particular transaction, which is $O(d \log N)$ (with d delta blocks, N transactions per strong block). This is acceptable for a few look ups (single independent merchant node) but will quickly become an $O\left(n^2\right)$ problem if the number of transactions tracked become a significant fraction of the full set in the strong block. To avoid this, the confirmation calculation can be simplified by stopping when a given number of confirmations that are deemed safe is reached. More complex schemes that simplify the calculation of the point of inclusion for a transaction from $O(d)$ to about $O(\log d)$ and would allow an efficient exact calculation seem feasible though. They are also probably not needed, given the low transaction exclusion risk after a few confirmations. However, the relative complexity of this operation hints at it being a good idea to keep the block chain just a single chain overall, except for the short term delta blocks / weak blocks instant confirmation regime.

It therefore also seems to be a good idea to keep delta blocks ephemeral; in the current demo implementation, they are a memory-only data structure and a freshly started node will start without any delta blocks known to it. This is also the meant to be the real-world operation mode.

4.6 Expedited propagation

Bitcoin's network protocol right now works to a large extent by an exchange of this sort: "Do you have X? No, please give me X. Ok, here's X.". This is a handshake that adds double the roundtrip latency for each hop in the Bitcoin network, a bottleneck which can slow down any instant confirmation scheme by a factor of two. A simple heuristic that has been investigated loosely is to always propagate transactions to each connected node and receive rejects, and stop when a propagating node finds that there is a large amount of redundancy when forwarding transactions from one to a another node, tracking redundancy for pairs of (sending-node, receiving-node). Work is underway by Peter Tschipper and Andrew Stone to implement automatic expedited transaction propagation via a different scheme. Finding and implementing a good general expedited propagation heuristic, both for (delta-)blocks as well as transactions is out of the scope of this proposal. The simulation in the next section assumes such an efficient expedited propagation scheme has been implemented.

5 Simulation

An initial python implementation of a discrete event system to simulate the propagation of transactions and delta blocks has been ported to a self-contained C++ program with no external dependencies (< 1000 lines). In the author's hope, making this easy to understand, run, and modify simulation available will help establishing Storm's value.

The simulation is single-threaded since a typical run takes up to dozens of minutes so far and sufficient parallelism can be achieved by just running different scenarios in parallel.

This simulation is available at the time of publication of this document at `github.com/awemany/storm-sim`.

It can be compiled, assuming a recent standard C++ compiler, with just a single call to `make` after cloning the `git` repository and filling in the `deltablocks.h` header file with appropriate values using the `deltablocks.h.tmpl` template. Alternatively, tue `runsim.py` python file can be used to automate this.

The simulation is a very simplified view of the network, hopefully covering just enough complexity to answer the main questions regarding delta blocks while not dropping (too many) pernicious corner cases under the table. In the analysis of simulation results, no error bars are given (but statistical error bars can also be easily estimated as most results adhere to Poisson statistics or binomial statistics that can in turn be roughly simplified to Poissonian count statistics). The simulation is considered to be mostly qualitative and at best fit for arriving at order-of-magnitude estimates. To be relatively certain to catch the corner cases, more detailed simulations should be made with real Bitcoin implementations in a test network that is delta Blocks enabled. However, this simulation is a valuable base to implement further qualitative analyses of different implementation approaches. The next section 6 contains a discussion of a very simple such setup using a deltablocks-enabled `bitcoind` and a comparison to the simulation code discussed here.

5.1 Transactions

In the simulation, transactions are just pairs of (txnnumber, TXID). The txn number is a global, unique ID assigned to each transaction thanks to its omniscient point-of-view of the situation. Double-spends are modeled as transactions with same TXID but different txn numbers twice. The transactions in the simulation do not have inputs and outputs, and instead of an input double spend, the whole transaction is 'a doublespend' which should still sufficiently capture the transaction doublespending problem without getting bogged down in reimplementing input/output selection. Consequentially, all transactions are also of the same size.

5.2 Blocks

Blocks are just delta blocks, as above. Strong and weak blocks are not differentiated, everything is a delta block. The current strong-blocks-only without deltas situation can be simulated as a degenerate case of the delta blocks with just a single ancestor chain and a block time of 600 s. Similarly, weak blocks in the sense of Rizun's Subchain paper can be simulated with a single chain and block times $\ll 600$ s.

5.3 Nodes and links

Nodes are connected through network links, and at the moment it is assumed that each connection is bidirectional, with a link of the same capacity in each direction. It is also assumed that the link bandwidth is the bandwidth per node-to-node connection, which is akin to the bandwidth of one half of a TCP connection (one direction) and not the bandwidth of a whole network link. (Probable) link congestion is detected in the simulation by checking for runaway send buffer sizes.

5.4 Parameters

The simulation time granularity is set to 1 µs, which should be detailed enough to capture all behaviors yet makes the simulation time fit into a 64bit integer for any realistic simulation run time.

In Tab. 1 the default parameters for the simulation are given. They have generally been selected to be realistic and still give reasonable good results. If no other, more specific values are given, these are the parameters that are used throughout here. Of course, the reader is invited to try the simulation with different parameters. This list is in the same order as the parameters that can be set in the file `deltablocks.h` so it should be reasonably easy to adapt the C++ for different scenarios.

It should be noted that both the simulation here as well as the full node implementation assume a full relay of all doublespending transactions (or at least a doublespending pair for each input that is ever doublespend).

Parameter	Value	Comments
Weak block time t_{w} $\left(=: \frac{1}{r_{\mathrm{W}}}\right)$	600 ms	One thousandth the nominal strong block time. Have exponential time distribution. Generated at any network node with equal chance.
Transaction rate r_{TX}	$10\,\mathrm{s}^{-1}$	Exponential time distribution. Injected into any network node with equal probability. This is the transaction rate including all doublespends.
Node connection scenario	random links	For each simulation run, a random set of links is generated between the nodes. The resulting graph is checked to connect all nodes to each other and is regenerated if that is not the case.
Number of nodes	10	Number of nodes in the simulation. All nodes behave equally, except for selected debugging/tracing functionality (see below).
Number of links	30	Number of links between nodes. A link is a bidirectional connection between two nodes.
Mean link latency	250 ms	Mean latency of a link. Link latencies are picked from a Gaussian distribution (with clamped floor) upon simulation startup.
Latency σ	100 ms	Width of the link latency distribution
Latency floor	50 ms	The floor of the link latency distribution. No latency below this value is allowed.
Mean link bandwidth	$200\,\mathrm{kB\,s}^{-1}$	The mean bandwidth of a link. Link bandwidth is as well drawn from a Gaussian distribution upon simulation startup and clamped at the lower end.
Link bandwidth σ	$20\,\mathrm{kB\,s}^{-1}$	The width of the link bandwidth distribution.
Link bandwidth σ	$150\,\mathrm{kB\,s}^{-1}$	No link bandwidth is allowed to fall below this value.
Transaction size	223 B	Size of a transaction in the simulation. This value is used solely to calculate bandwidth and latency of transaction propagation.
Transaction size in a delta block	100 B	Size of a transaction in a delta block, used for block propagation estimates. This value is lower than the above to account for efficient block propagation schemes. The details of the various efficient block propagation schemes are subsumed in this simple ratio.
Ancestor size in a delta block	100 B	Amount of data (for network traffic calculations) an ancestor link takes up in a delta block.
Empty delta block size	200 B	Size of an empty delta block (for propagation calculations).
Block rerequest probability	0.15	Probability with which a node re-requests a delta block from a sending node, simulating decode failure. Multiple failures are possible and occur with corresponding multiplied probabilities.
Block request size	100 B	Size of a (re-)request message for re-requesting a delta block. For network calculations.

Table 1: Default simulation parameters (continued)

Doublespending fraction f_{DS}	10 %	Fraction of transactions that are doublespent. A doublespend is injected simultaneously at two random nodes in the network (it can happen that the same node is selected).
Drop doublespends for mining	`true`	If a transaction is detected as a doublespend locally at any node, it is excluded from block building.
Relay doublespends	`true`	Doublespends are relayed just like normal transactions through the network, ensuring that all nodes get a full picture as fast as possible.
Minimum transaction age for block inclusion	0	The minimum amount of time a transaction has to spend in a node's mempool before it is included in any mined block.
Maximum transaction age for block inclusion	72 s	Maximum age of a transaction in a node's mempool - after this it is evicted. This serves essentially as a garbage-collection threshold for a node's mempool. Note that the mempool state is also used to determine transaction forwarding in the simulation. It is thus necessary that this value is a lot larger than the end-to-end propagation time in the network so as to not form propagation loops!
Use deltablocks method	`true`	Should the deltablocks building method be used? Changing this value to `false` in the simulation causes just at most a single ancestor to be included in a deltablock, thus degrading the simulation back to the ordinary weak blocks case.
Transaction tracing probability	1.0	Probability of tracing a transaction's detailed fate at a selected node.
Transaction tracing node number	0	Node that the transactions are traced at. Just a single node is selected for efficiency. As all nodes are otherwise equal, this value should not matter in the default random network graph scenario.
Simulated time	600 s	Real time that is simulated. One nominal strong block period.

Table 1: Default simulation parameters

5.5 Results

5.5.1 Orphan rates

The key metric to look for when looking at the impact of delta blocks on the overall safety of the system is the orphan rate. Because an adversary should be assumed to be well-organized internally (low orphans for himself) and also in the worst case in control of all double spends that appear on the network, it is important to make sure that the honest miners do not put a large fraction of blocks and thus hash power to waste. As incompatible (doublespending) delta blocks cause a certain orphan rate, the orphan rate furthermore is a bound on the number of successful doublespends than can happen in the network.

For a total number of observed blocks N and n blocks in the most-WPOW chain tip, the (total, network-wide) orphan rate r_o as used here is defined as:

$$r_\mathrm{o} = \frac{N - n}{N}.$$

For a couple key parameters from the above table, the orphan rate for varying that parameter is shown in Figures 6 to 10.

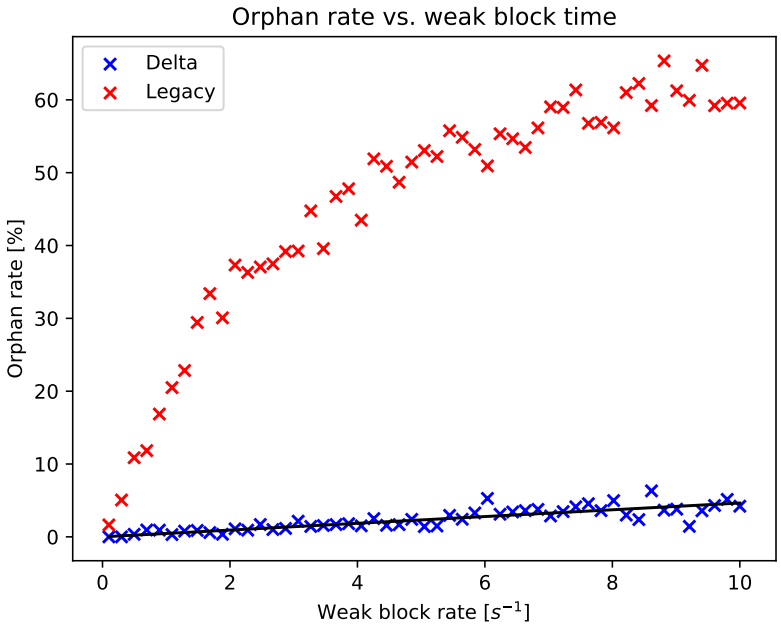

Figure 6: Total orphan rate vs. weak block rate. Merging of deltablocks are in blue whereas the simple stacking of weak blocks is in red.

Considering weak block time In Fig. 6, the orphan rate for varying weak block times is shown, both using delta blocks as well as ordinary weak blocks. This shows the key advantage of delta blocks. By merging compatible but competing blocks into a single confirmed transaction set, a lot less POW is wasted compared to the straight-forward weak blocks model. All blocks that do not mine conflicting doublespends can be merged into a single set, allowing to extend delta blocks to a faster emission regime than before. Each weak block has an otherwise fixed time window for an incompletely-propagated

doublespend to be included, which results in a larger rate of conflicting weak blocks for higher emission rate, which is the linear (least squares) fit in the above plot.

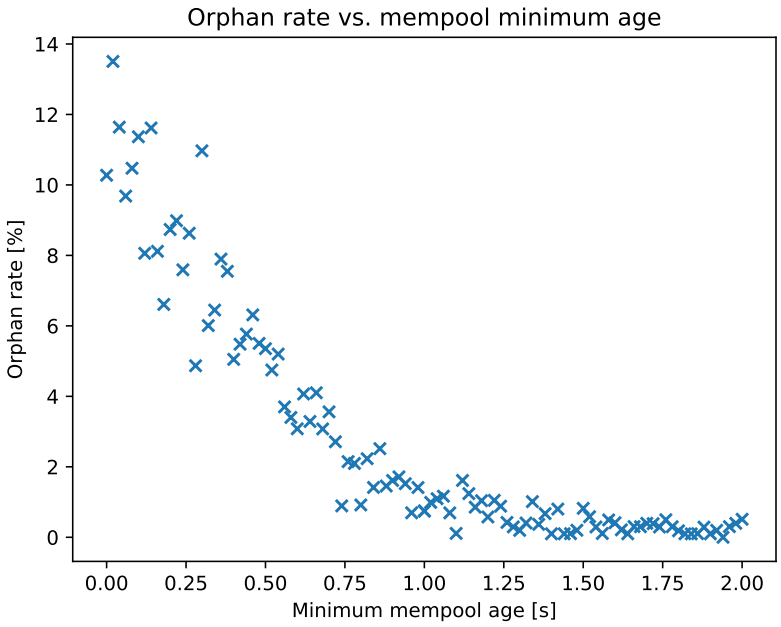

Figure 7: Orphan rate. vs. mandatory wait time before inclusion. *Note:* For this scenario, the mean link latency has also been increased to 1 s to actually be in a regime where a noticeable amount of orphans due to conflicting transaction sets occur. The default parameters from above are too optimistic to show the influence of this extra wait.

Considering minimum waits In the delta blocks model, orphan blocks are produced only when the blocks cannot be merged into the main delta tree. This is either due to an incompatible, doublespending transaction included in the block or as miners refrain from considering a competing chain for performance reasons (e.g. during an attack that tries to slow down the network with expensive compatibility cross-checks). A very simple way to make sure that no doublespends are included in a block is to simply throw out all known doublespending transactions before building blocks. This has been implemented in the full node code as well as the simulation. This however, still leaves a timing gap for doublespending transactions to be injected simultaneously into different parts of the network and a doublespending attacker that exploits the finite propagation time might still be able to get a doublespend to confirm. A very simple defense against this in the delta blocks model is to let transactions age in the mempool before even considering them for inclusion. Waiting for the end-to-end propagation time in the network and assuming that all doublespends are forwarded makes any doublespending attack infeasible.

For a transaction with a sufficiently high fee, a miner might be compelled to take the risk of including that transaction earlier, which might correspond to a natural fee market for (very) fast transactions.

In Fig. 7, the orphan rate is shown for different minimum inclusion waits for the given simulated network topology. In this example, a wait of just over one second makes succesful doublespends highly unlikely. As mentioned above, to avoid giving

doublespenders the chance to time their transaction injections just barely "right", it might make sense to either replace the fixed wait with a draw from an exponential distribution or add an exponentially distributed wait on top, potentially scaled by the fee rate that the transaction would pay. A further potential enhancement to fend off attacks where lots of doublespends are injected into the network would be to make this wait depend on the observed orphan rate in the network.

Overall, the delta blocks approach works best when doublespends are avoided as much as possible to be even considered by miners, which is an alignment of incentives with an overall well-operating system.

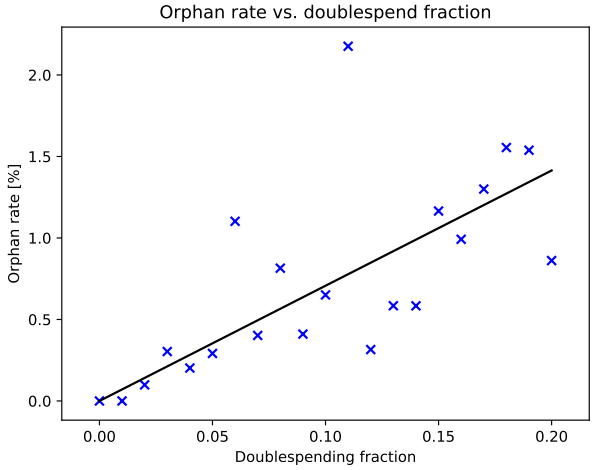

Figure 8: Total orphan rate vs. doublespend fraction.

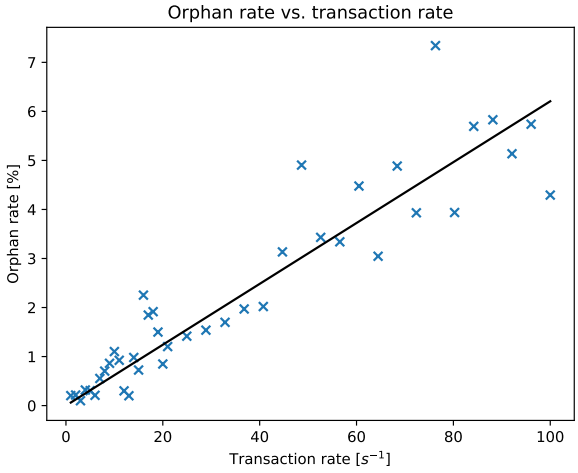

Figure 9: Total orphan rate vs. transaction rate.

Considering doublespending and transaction rate In Fig. 8 and Fig. 9, the effect of the doublespending fraction respectively the transaction rate (at constant doublespend fraction) on orphan rate is shown. Not surprisingly, both show the same (assumed to be

linear) behavior at low orphan rates. Any single included doublespending transaction will create an orphaned delta block and as it is expected the likelihood that this occurs during delta block building scales linearly with both these variables. As these two variables are most easily influenced by an attacker, it is most important to make sure that they have the least effect. The aforementioned minimum miner wait is a viable approach to tackle this as well.

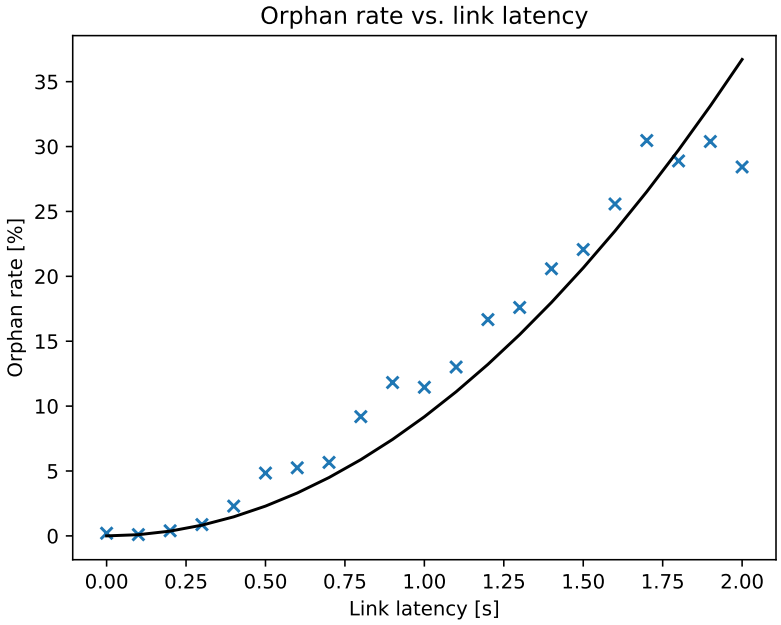

Figure 10: Total orphan rate vs. mean link latency l.

In Fig. 10, the effect of mean link latency and thus network diameter on the orphan rate is shown. This parameter is one of the most important to a viable delta blocks system. The line shown in the plot is a l^2 least squares fit to the simulation. The assumed explanation for the squaring behavior is that l scales both the time window for a doublespending opportunity due to transaction propagation as well as rate at which competing blocks will appear on the network (with a given risk of containing conflicting doublespends).

Overall behavior For small orphan rates, the simplified overall orphan rate is thus expected to vary roughly by the above parameters (except miner wait!) as:

$$r_{\mathrm{o}} \sim f_{\mathrm{DS}} \cdot l^2 \cdot r_{\mathrm{TX}} \cdot r_{\mathrm{w}}.$$

In the perfect case of always bounded and complete propagation of all transactions, the miner wait can be expected to always push orphan rate to *zero* if it is just long enough. The exact shape of the orphan rate vs. minimum wait curve depends on the network topology.

5.5.2 Merchant risk

The property that is key to answering the viability of delta blocks as an actual instant confirmation solution is how well and how fast it will allow a merchant to make risk-informed decisions on incoming customer transactions.

Notably, it is assumed that all miners follow the weak consensus and that the transaction in question has sufficient fees to entice the miners to mine it. The additional risk involved if any of those two conditions are not met is not further discussed here.

From a merchant's point of view, there are several aspects that s-/he can look at to estimate the likelihood of final transaction confirmation and thus a successful sale.

The merchant is always assumed to observe any potential doublespends of the transaction and react to them accordingly, independent of delta blocks. Should a merchant observe a doublespend for the transaction in question, he can almost certainly assume a fraud attempt. In case that no doublespend is initially observed, the deltablocks weak consensus scheme will eventually give him the certainty that the transaction paying him is not doublespend later on. This is the scenario that is addressed by deltablocks.

The first metric would be to observe the weak confirmations piling up on the transaction in question. As blocks can potentially be orphaned and replaced with conflicting (doublespending) ones, the likelihood for final inclusion, given a number of weak confirmations, is a quick and easy way to estimate a transaction's fate from the number of observed weak confirmations.

Then, it is also important for the miner to know how long it will take for the given number of confirmations to show up (since he received the transaction).

Combining those two measures will give a transaction inclusion certainty estimate over time.

To answer these questions, the simulation has been instrumented such that transaction confirmations can be traced at the selected node (#0). When a transaction first shows up at that node in a block, its inclusion (or not) in the longest chain as per the discussed maximum WPOW rule is tracked.

Scenario	**x = 1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
WB, 600 ms	1.3‰	0	0	0	0	0	0	0
DB, 600 ms	1.0‰	0	0	0	0	0	0	0
WB, 100 ms	9.2‰	1.6‰	0.5‰	0	0	0	0	0
DB, 100 ms	4.0‰	3.0‰	0.8‰	1.1‰	0.1‰	0	0.1‰	0

Table 2: Number of transactions that got confirmed x times in the longest chain but did *not* make it into the longest chain eventually. DB: Delta blocks, WB: Ordinary weak blocks. Approximately 6000 transactions total in each run, as per the default simulation parameters.

Non-final confirms For four scenarios, for 100 ms as well as 600 ms and both delta as well as ordinary weak blocks, the likelihood that a transaction confirm is observed while the transaction does not eventually end up in the most-WPOW tip is listed in Tab. 2.

It should be noted that these numbers are this surprisingly low as those of the approximately 600 doublespends made in total, most are discovered in time before the block is emitted and are not considered further at all. Any transaction that is not doublespend and not in the longest chain is included with very high likelihood in the longest chain (due to the -in this context- comparatively long mempool timeout of 72 s).

Furthermore, a doublespend has to be included in at least *two* competing weak blocks before showing up in the above statistics.

In general, these figures hint at $t_{\mathrm{w}} = 600\,\mathrm{ms}$ being a good compromise for the given other simulation parameters. It further shows that waiting for just three weak confirmations in either 600 ms scenario is enough to be certain with very high likelihood (99.9%) that a transaction is going to make it into the strong block. And this is for a comparatively high double-spending rate of 10%.

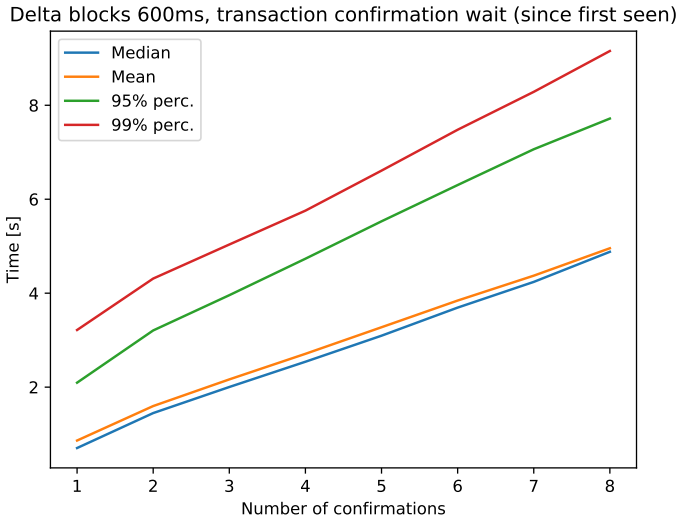

Figure 11: Confirmation times for $t_{\mathrm{w}} = 600\,\mathrm{ms}$.

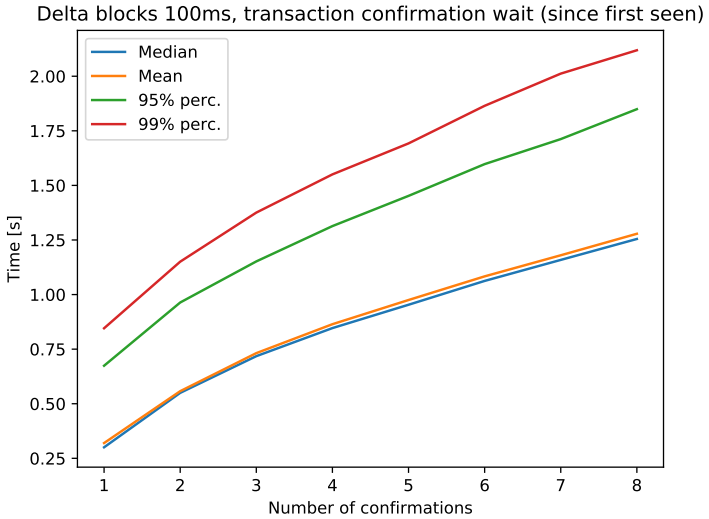

Figure 12: Confirmation times for $t_{\mathrm{w}} = 100\,\mathrm{ms}$.

Confirmation times In Fig. 11 and Fig. 12, the time to reach a given number of confirmations in the delta block scenario is shown for a delta block time of 600 ms and 100 ms, respectively. Even if the simulation parameters are considered overly optimistic, reaching sufficient confirmations within a few seconds still seems realistic.

With the above likelihood of a confirmation being non-final taken into account, this shows that the proposed scheme should allow merchants to accept transactions as final within seconds while accepting a doublespend risk in the ‰-range. And this does not include the other approaches a merchant can use to mitigate transaction risk, such as (0-conf) forfeits or knowledge about the customer.

6 Full node test

Early, preliminary tests have also been made with the C++ implementation for the BU full node that is now available as a work-in-progress pull request to Bitcoin Unlimited. Writing this preliminary implementation is first and foremost a check whether and which surprising complexity roadblocks might turn up during a production implementation.

It should be noted that the current implementation has a few drawbacks still. To simplify testing, it has also *not* been written with having all adversarial conditions properly in mind and before it is used in production, it needs to be heavily reworked and tested accordingly. *It contains various known bugs.*

Furthermore, and impacting performance for the tested parameters somewhat, the Merkle tree is still fully rebuilt for each and every delta block that is being produced, which means that there is a case of $O(n^2)$ complexity (with n transactions) still.

Delta block building likewise is not optimized yet and iterates over all transactions in the merged block to find those that are included already, creating another $O(n^2)$ behavior.

6.1 Setup

With the above caveats, a very simple test scenario, consisting of three nodes running on the same host has been set up:

$$\text{Node}_2 \leftrightarrow \text{Node}_1 \leftrightarrow \text{Node}_3$$

The node is a decent server with 256 GiB of RAM and a 20-thread Xeon CPU at ≈ 2.5 GHz clock frequency. Latencies between the nodes $1, 2$ and $1, 3$ have been simulated to be 500 ms each[2]. All three nodes are set up to generate delta blocks with equal rate (1 Core each).

The parameters for the chosen setup are shown in Tab. 3.

Simulation comparison. A run of the discrete event simulation from the last section has been set up as well, and set to be as close as possible to the `bitcoind` test. This is to have a comparison of the simulation with a more realistic test of real code. Differences in setup are noted below. In contrast to the full node setup, the simulated links have been set for a default delay of 1 s each, to account for the fact that the simulation simulates full expedited transaction propagation. Though both this full node test as well as the simulations from the last section are in a more general sense *simulations* of a real network, using the compiled `bitcoind` is called the full node test and only using the previous discrete event simulation is called "a simulation" herein. Comparison plots for the simulation can be found at the end of this section.

[2] Using the Python script that is available here: `https://superuser.com/questions/789420/how-do-i-delay-tcp-traffic-on-a-socket`.

Parameter	Value	Comments
Delta block time t_{w}	2.16 s	Set to 600 ms. Due to various reasons (`chain_nol` difficulty adjustment intervals, different number of CPUs during different tests, etc.) the actual value is different. Furthermore, due to the non-constant block building times, this average value also varies with a delta blocks merged size. The simulated delta block time has been set to 2.16 s as well.
Total transaction rate r_{total}	$5.44\,\mathrm{s}^{-1}$	Transactions come with approximately Poisson-distributed timing and arrive at any of the three nodes[3]. This is the measured transaction rate arriving at the nodes for the selected $[t_0, t_{\mathrm{max}}]$ as discussed below.
Double spend fraction	10%	Here, this is the chance to emit a double spend. For each doublespend, two transactions are injected into the network.
Mempool expiry time	72 s	This very fast mempool expiry time (seconds, not hours!) has been set as to minimize the non-linear effect a large mempool has on block building. Note that the mempool keeps double-spend transactions until they expire.
Deltablock retransmission likelihood	15 %	This value is only set for the simulation and corresponds to the retransmission rate that has been observed during the full node run.

Table 3: Most important parameters for simple full node test.

[3] Approximately Poisson-distributed as injection rate limit of the `python`-based transaction injector start to show at this rate.

6.2 Results

In Fig. 13 (Fig. 14 for the simulation), the WPOW of the generated delta blocks is plotted vs. time for parts of an example run with 5 arriving strong blocks in blue. If not otherwise noted, data from Node_1 is shown here and below. It can be seen that due to the $O(n)$ behavior in block building speed, delta block building gets progressively slower and slows down the node so much that eventually it cannot keep up (time span $t > 10\,000\,\text{s}$). This behavior seems furthermore not consistent, hinting at further performance bugs in the code. Therefore and for all preliminary views of transaction confirmation behavior below, only data up to $t_{\text{max}} = 5200\,\text{s}$ is further analyzed. The red curve shows the delta set size of each handled delta block.

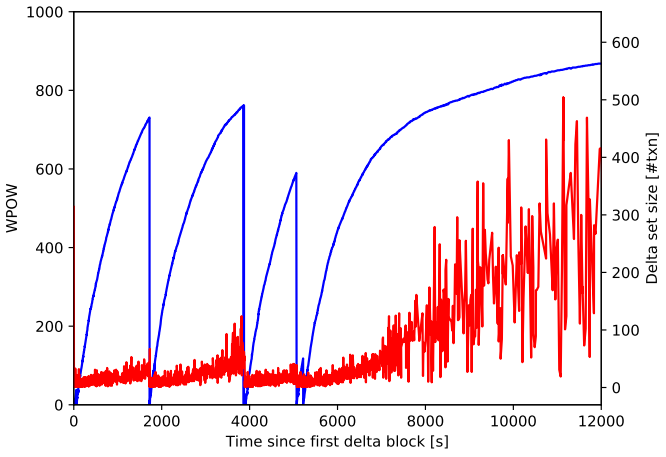

Figure 13: WPOW and delta set size over time for blocks arriving at Node_1.

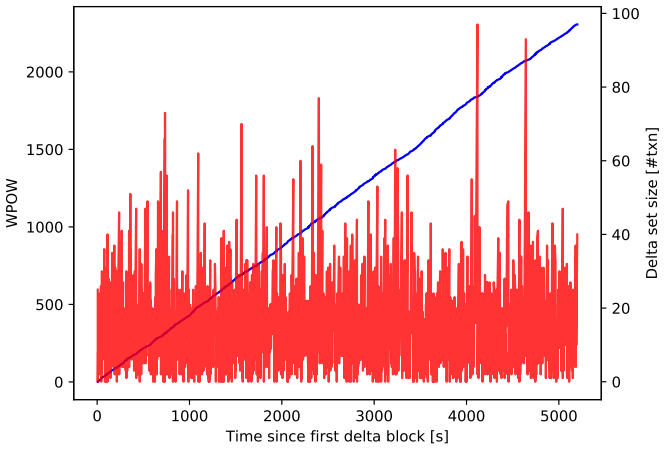

Figure 14: WPOW and delta set size over time for any blocks generated in the simulation. There are no returns to zero for the WPOW graph as strong blocks are not simulated. This is the desired, eventual (almost) constant-time behavior for the full node implementation. 5200 s were simulated to match the full node test.

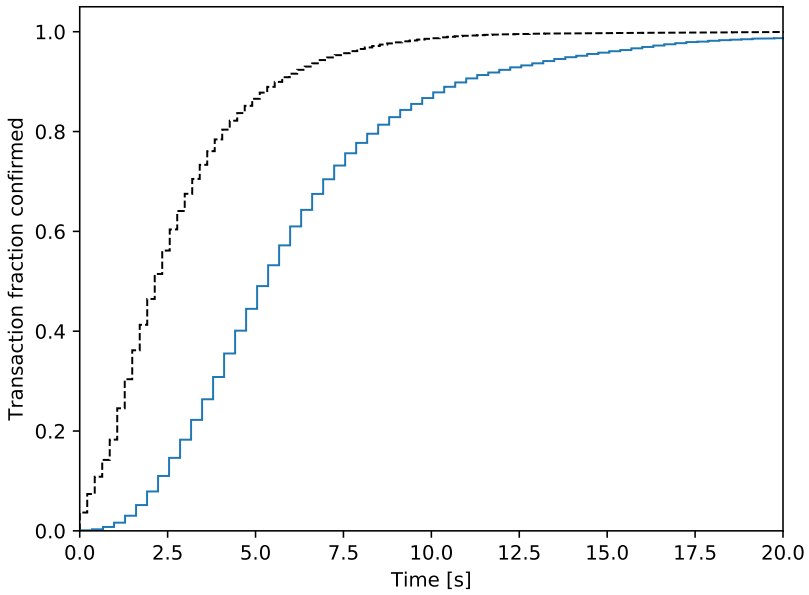

Figure 15: Transaction confirmation probability over time for those transactions that are included in delta blocks (unconfirming doublespends excluded) in blue. The dashed black line is the (better) simulation result. The discrepancy and worse performance of the full node implementation is likely explained by the non-optimal block building.

From the 2nd strong block as visible in Fig. 13, the POW efficiency can be estimated: Maximum WPOW is 731 for a total of 777 weak blocks, yielding an efficiency of $\eta_{\mathrm{FN}} = 94.1\,\%$ of all WPOW included in the final strong block. For the simulation, there are 2416 weak blocks mined total with the top block having 2306 WPOW, thus yielding a roughly comparable $\eta_{\mathrm{sim}} = 95.4\,\%$.

6.3 Transactions

In Fig. 15, the confirmation probability for any confirmed transaction in this example run is shown. Tab. 4 lists the fate of various selected transaction subsets in this delta block run. Most importantly, the last few lines lists the probability that a transaction can be deemed to be finally included in the strong block chain for a given number of weak confirmations, as also discussed in the preceding simulation section. In this context, a transaction is doublespend if a conflicting transaction has been made and thus exists anywhere in the network.

Transaction set	**Value**	**Sim. Value**	**Comments**
Total number	28 296	28 188	The total number of transactions in the selected time range.
Ever confirmed	24 064, 85 %	23 867, 85 %	Transactions that are ever confirmed in a delta or strong block.
Finally confirmed	23 894, 84.4 %	23 752, 84.3 %	Transactions that are finally confirmed in a strong block.
Doublespends	5171, 18.3 %	5098, 18.1 %	Doublespend transactions, as per the set rate.
Ever confirmed doublespends	939, 3.3 %	779, 2.8 %	Doublespend transactions that are included in at least one delta block.
Finally confirmed doublespends	769, 2.7 %	665, 2.4 %	Doublespend transactions that are included in a strong block.
Finally confirmed non-doublespends	23 125, 81.7 %	23 087, 81.9 %	Regular transactions that are included in a strong block.
Not finally confirmed non-doublespends	0, 0 %	3, 0 %	Regular transactions that are *not* included in a strong block. Essentially regular transactions that "fell through the cracks" and are dropped from the mempool too early.
Final inclusion fraction after **1** confirmation	28 217 99.72 %	28 073, 99.59 %	Fraction of transactions that are also finally included in a strong block after having seen a single confirmation.
Final inclusion fraction after **2** weak confirmations	28 276 99.93 %	28 162, 99.91 %	Like above.
Final inclusion fraction after **3** weak confirmations	28 293 99.99 %	28 184, 99.98 %	Like above.
Final inclusion fraction after **4** weak confirmations	28 294 99.99 %	28 186, 99.99 %	Like above.
Final inclusion fraction after **5** weak confirmations	28 294 99.99 %	28 188, 100 %	Like above.
Final inclusion fraction after **6** weak confirmations	28 296 100 %	28 188, 100 %	Like above.

Table 4: Fate of transaction subsets in the full node test.

The high likelihood of final inclusion demonstrates the overall workability of the delta block scheme for quickly reaching consensus on transaction inclusion, to give merchants the tools needed to estimate doublespending risk.

Furthermore, these values appear to be compatible in general to the results from the previous simulation section. Very obviously, however, work is still necessary to implement the larger structural changes necessary to `bitcoind` to support constant time delta block generation and to speed up delta block generation to be closer to the more optimistic simulation values.

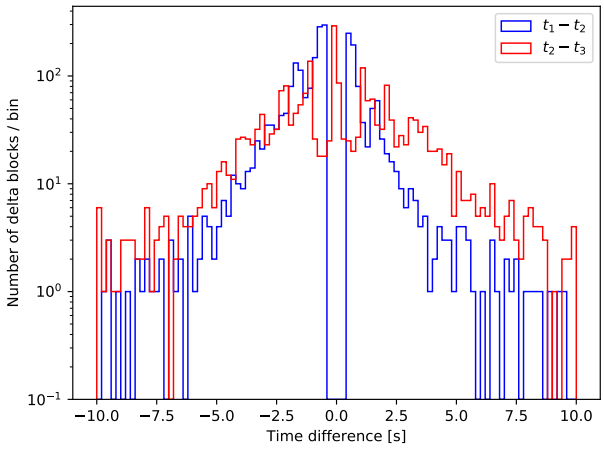

Figure 16: Delta block propagation times. Blue shows $t_1 - t_2$ difference for arrival time t_1 at Node_1 and t_2 arrival time at Node_2, red the difference $t_2 - t_3$ with t_3 arrival time at Node_3. 3 respectively 68 outliers not shown in histograms. All blocks propagated within 30 s.

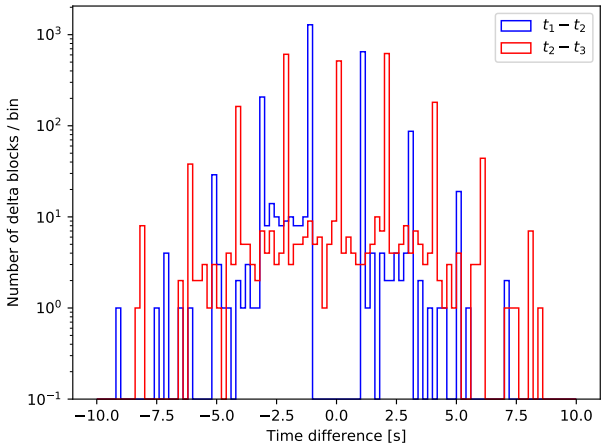

Figure 17: Simulated delta block propagation times, as in Fig. 16. The "real" full node setup has more variance in the effective block propagation times, smearing the peaks at multiple round trips for deltablock retransmission out.

6.4 Delta block propagation

In Fig. 16 (resp. Fig. 17 for the simulation), delta block propagation time histograms between Node_1 and Node_2 as well as between Node_2 and Node_3 (through Node_1 as the intermediate hop) are shown in blue and red, respectively. The peak at 0 s in the latter histogram is from delta blocks originating from Node_1 and reaching both outer nodes at the same time. It can be seen that the expedited mode of propagation that has been implemented works most of the time and in principle, though there is a considerable number of retransmissions and extreme outliers.

7 Conclusions

While Storm is far from being a ready-to-use and fully specified implementation, the demonstration code and simulation hints that this approach is likely viable in principle for instant confirmations and that it is in some ways superior to discussed alternatives. Further exploration and comparison of this method to competing approaches seems prudent. One area to emphasize here is further testing and analysis of delta blocks propagation when using probabilistic protocols such as Graphene to make sure that intermittent set reconciliation failures or other communication problems do not allow for a "long tail" runaway failure of weak blocks propagation in any realistic real world scenario. Furthermore, adversarial cases have to be analyzed in detail.

It should also be noted that there is some rough similarity to Ethereum's GHOST and uncle system[10, 11]. The approach discussed here, however, deviates in many ways such as implementing transaction sharing in addition to reward sharing as well as handling all blocks in a symmetrical way.

There have been repeated calls to shorten the block time on BCH. However, using a weak blocks solution like the one discussed here has several advantages to a shorter block time: It touches neither the emission schedule nor the other core parts of the system and lives (to a large degree) side-by-side to the current 10-min strong block propagation model.

Furthermore, the incentive to make, publish and follow delta blocks is due to the proposed small and narrow soft fork that can be undone relatively easily in case of an emergency (by just ignoring all weak blocks by the majority of hash power), such as an unforeseen bug or incentive problem on the live network. The effects of this would be to fall back to the current mode of operation.

Aside from necessary test-net testing, weak blocks can also be enabled and tested on the live network with only a fraction of the BCH miners particpating. This might also allow further tweaks to the parameter set before switching to "production mode".

In the view of the author, these properties fit best with the desired conservative, slow and steady evolution of the core protocol.

References

[1] Amaury Séchet, Comment on previous-POW on reddit, https://www.reddit.com/r/btc/comments/auz27d/question_for_avalanche_has_anyone_thought_of/ehcx7yu/ (https://archive.fo/Te02u)

[2] Amaury Séchet, Merklix Trees, https://www.deadalnix.me/2016/09/24/introducing-merklix-tree-as-an-unordered-merkle-tree-on-steroid/ (https://archive.fo/UQou5)

[3] Andrew Stone et al., Bitcoin Unlimited v0.12.1bu Release Notes, https://bitco.in/forum/threads/bitcoin-unlimited-v0-12-1bu-%E2%80%9Cxpedited%E2%80%9D-official-bu-release.1350/ (https://archive.is/niJRQ)

[4] Imaginary Username, Standard Priority and Doublespend Proof, https://gist.github.com/imaginaryusername/edcd611313abb5390872b7dc4911d170 (https://archive.is/K9tka)

[5] awemany, Zero Conf Forfeits, https://gist.github.com/awemany/619a5722d129dec25abf5de211d971bd (https://archive.is/Ikdmr)

[6] awemany, Implementation of weak blocks / subchains for Bitcoin Unlimited, https://github.com/BitcoinUnlimited/BitcoinUnlimited/pull/856 (https://archive.fo/l6YCV)

[7] Kevin Sekniqi and Emin Gün Sirer, Using Avalanche for Pre-Consensus on Nakamoto Consensus Protocols, Lake Garda, Italy, https://www.youtube.com/watch?v=MW4UW8fR_Y8

[8] Team Rocket, Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies, https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV

[9] Peter Rizun, Subchains, https://www.bitcoinunlimited.info/resources/subchains.pdf(https://archive.fo/u0bUZ)

[10] Tristan Winters, Uncle Mining in Ethereum, https://www.ethnews.com/uncle-mining-in-ethereum(https://archive.is/xK7rS)

[11] Yonatan Sompolinsky and Aviv Zohar, Secure High-Rate Transaction Processing in Bitcoin, https://eprint.iacr.org/2013/881.pdf (https://archive.is/g4xgM)

[12] Wikipedia, Persistent Data Structure, https://en.wikipedia.org/wiki/Persistent_data_structure(https://archive.fo/cUZUt)