Forward Blocks

On-chain/settlement capacity increases without the hard-fork

Mark Friedenbach
mark@friedenbach.org

***Abstract*— Segregated witness achieved a 2 to 3.6x capacity increase for on-chain settlement by moving script and signature data out of the area constrained by the original 1 MB block size limit. Unfortunately this approach could only be used to achieve a one-time increase as there is no remaining unsigned transaction data of significant size that could be moved out of the transaction structure.**

The previously proposed concept of soft-fork extension blocks involves entire transactions being moved into segregated data structures, which allows for limitless scaling at the cost of breaking the transaction graph. Alternatively simply removing or replacing block-size and other aggregate limits preserves the transaction graph but at the cost of a hard-fork cutoff event, and the significant transition risks and centralizing pressure that comes with any scheduled hard-fork event.

We introduce the concept of *forward blocks* which combine the best of both approaches enabling aggregate block limits on size and number of signature operations to be circumvented as a soft-fork, while preserving the property that the spend graph, including all transactions and their witnesses, is seen by legacy clients. Effective block sizes of up to 14.336 GWe can be achieved in this way, permitting scaling up on-chain/settlement transaction volume to 3584x current levels, if permitted by the new consensus rules.

The construction of forward blocks also makes possible changing the proof-of-work algorithm in a soft-fork compatible way, either simultaneously with dropping the aggregate limits or as a later re-application of the concept in nested forward blocks. Other benefits of the implementation approach taken include the introduction of explicit sharding for better scaling properties, rebateable fee market for consensus fee detection, and smoothing out drops in miner subsidy. It also provides the necessary prerequisite protocol pieces for confidential transactions, mimblewimble, unlinkable anonymous spends, and sidechains.

I. Introduction

The bitcoin block chain has per-block aggregate limits that prevent the on-chain settlement capacity from increasing beyond fixed levels of 4 MWe per 10 min in expectation. While segregated witness achieved a 2 to 3.6x improvement in settlement capacity by removing witness data from the area of the block subject to classical limits, this was a one-time gain: there are no other large areas of the block that can be safely segregated into their own data structures. Conventional wisdom is that on-chain settlement capacity of bitcoin and bitcoin-derived systems can only be increased by one of four different mechanisms. Taken individually, each approach comes with significant tradeoffs:

1) *Hard fork* change the consensus rules in a way that is not forwards compatible with old clients, e.g. simply increasing the weight limit, thereby requiring **all** nodes to upgrade by some flag day. While conceptually simple, the flag day requirement violates many of the underlying principles of decentralization from which bitcoin-like systems derive their value.
2) *Soft fork* validation rules for segregating entire transactions from the classically visible block structure, in the form of a so-called *extension block* which contains extra transactions beyond the classical limit. While being a soft-fork, this category of solutions has the disadvantage of opaquely hiding the spend graph from un-upgraded nodes: in contrast to segregated witness which merely hid authorization information, an extension block approach will necessarily obscure entire transactions, including the inputs they spend and the outputs they generate from any un-upgraded client. Some proposals even involve forcing empty non-extension blocks for the purpose of coercively shutting down the legacy network, forcing an upgrade via denial of service. Again, this violates the principles of decentralization underlying bitcoin-like systems.
3) *Soft fork* validation rules for transferring value between sidechains enables scaling to happen on a separate network using tokens ultimately backed by original currency from the source chain, e.g. bitcoin. While sidechains are interesting as a way of locking value for use in new and novel off-chain ledgers providing features which are either immature or will never make it to the mainchain, as a mechanism for scaling it doesn't compare well. Either SPV trust assumptions are required for the value transfer authorization, or full validation of the other block is required and the sidechain mechanism becomes essentially an overly complicated implementation of extension blocks without the empty-block coercion, and inherits all the same negative tradeoffs regarding ledger opacity to un-upgraded nodes.
4) Manipulation of timestamps affecting the difficulty-adjustment algorithm to make blocks more frequent and thereby indirectly achieve similar results to a block size increase. Known as the *time-warp attack*, this exploit has been used to play out subsidy in various altcoins going back as far as at least 2011. It has been noticed by various people that coordinating an exploitation of this flaw to lower inter-block intervals would effectively achieve similar results as a block size increase in terms of on-chain settlement throughput.

*This work was not supported by any organization.

However due to fundamental limitations in latency-induced message propagation times, and the resulting consequences for centralization risk, comparatively very little gain can be had from using a time-warp exploit before the risks become unacceptable.

While each of these approaches individually have unacceptable trade-offs, it turns out, remarkably, that combining them all together "cancels out" most of the bad tradeoffs wile retaining the combined benefits. The resulting scheme, held together by a novel new mechanism we call *forward blocks*, is actually less complicated than one might think of a "everything and the kitchen sink" proposal.

Additionally, it was previously considered that upgrading the proof-of-work function was not possible in an incentive compatible way, and therefore necessarily a hard-fork alteration of the consensus rules. However implementation of forward blocks enables, and in fact requires a soft-fork proof-of-work change[1] as natural consequence of its construction. In fact this insight into accomplishing soft-fork upgrades of the proof-of-work function is the driving idea behind the forward blocks construction, and where we begin the exposition in Section III, after a brief discussion of centralization risks in Section II.

Taking the concept of a soft-fork proof-of-work change to its logical conclusion, we develop in Sections IV, V, and VI a two-chain structure where the *forward block chain* establishes a consistent transaction ordering reflected in the *compatibility block chain* after a loosely coupled process of state synchronization.

To actually increase on-chain settlement throughput, Section VII describes the approach of directly increasing aggregate limits used on the forward block chain, synchronized with time-warps to lower the inter-block interval on the compatibility chain, all constrained by a demand-responsive and growth-constrained flexible weight cap.

To safely reach scaling limits, Section IX describes how elements of the sidechain value peg mechanism are used to enable *sharding* across multiple forward block chains. Once fully utilized, an approx. 30x reduction in latency-induced centralization risk can be achieved.

We then present and justify the set of initial parameters and long-term growth constraints in Section X.

Finally, it turns out the mechanism developed for transferring value between shards is generally applicable as a mechanism for handling coinbase maturation requirements and transferring value between segregated ledgers. We briefly examine in Section XI how it can be used to provide a rebatable fee market for consensus fee detection as well as handle the transfer of explicit value between confidential transactions / mimblewimble, unlinkable anonymous spends, and sidechain ledgers.

[1]The "new" proof-of-work could be a variation of merged mining, however, should retiring existing mining hardware be deemed an anti-goal.

II. Centralization Risks of Larger Settlement Throughput

Centralization risks in bitcoin are generally concerning for two reasons: increasing *cost of validation* and decreasing *censorship resistance*.

The *cost of validation* is the amount of resources (computation, memory, bandwidth) required to initialize and maintain a full-node validator so as to be able to transact on the network without trusted third parties. Any increase of the on-chain settlement throughput necessarily incurs an increase in the cost of validation that is at least linear. In terms of magnitude, opinions differ but we contend that it would be largely noncontroversial to require that full node validation should require no more than the computational resources of a recent consumer laptop or desktop workstation, and have bandwidth requirements met by the lesser of a tail-weighted average of bandwidth available to home internet users or across onion routing overlay networks with large anonymity sets.

Censorship resistance is that property which results from the ability of any user to make a fair attempt at mining a block, with the chance of success proportional to their share of the hash rate, no matter how small; censorship being when another entity is able to unfairly control one's access to transact on the block chain, or unilaterally determine the order in which transactions are allowed to confirm. Resistance to censorship is achieved by ensuring anyone and everyone who desires has the ability to mine blocks if they so choose, and that the probability of finding a block, and of having that block accepted by the network and not later reorg'd out, is precisely proportional to one's share of the hashpower, no matter how large or small that share is. If Alice has twice as much hashpower as Bob, then she has twice as much say into transaction ordering as him, but no more and no less. And since mining is by lottery, even a small percentage hashrate miner still has a chance to mine a block in due time. As the miner of a block determines the order of transactions contained within, so it is that censorship resistance is having control over ordering of transactions shared by all users, fairly weighted by hashpower.

Generally on-chain settlement throughput can be increased in two ways: by allowing blocks to be mined more frequently, or by increasing the size of blocks. Both would adversely affect censorship resistance by increasing the proportion of time it takes to validate and propagate a block relative to the time it takes find it, as during propagation and validation other miners must either generate stale work while they wait for validation to complete, or open themselves to the risk of mining on top of an invalid block. Miners don't have to make this tradeoff when building on their own blocks (as they know them to be valid already), which provides an advantage to larger hashrate miners who find themselves in that lucky circumstance more often than a less powerful miner would. Whereas the cost of validation is linear with the on-chain settlement throughput, the relationship of throughput with censorship resistance is non-linear and dependent on many

factors.

There is a natural floor to block propagation times due to fundamental speed-of-light induced message latency, which limits the effectiveness of lowering the inter-block interval compared with directly raising weight limits. Therefore lowering the inter-block interval is not an effective mechanism for achieving higher on-chain settlement capacity as the costs grow rather quickly, and in excess of the benefits received. We will explore however in Section IV through Section VII a mechanism by which the benefits of increasing the per-block weight limit directly can be achieved, while also lowering the inter-block interval from the perspective of un-upgraded nodes, without the superlinear effect on centralization risk.

III. Changing Proof-of-Work as a Soft-Fork via a Dual-PoW Difficulty Transition

Changing the proof of work of an existing chain is normally thought of as something that can only be accomplished by means of a hard-fork. However it turns out it is possible to soft-fork a proof-of-work change while still delivering valid blocks to un-upgraded clients.

The essential observation is that it is a soft-fork amendment of the consensus rules to make a block subject to two proof-of-work checks, and that transitioning difficulty from one to the other is naturally accomplished by a sliding allocation of block reward. At the beginning of the transition period the hash rate would remain as before, as nearly all of the block reward is allocated to the original proof-of-work miners, but would slowly decrease as dropping rewards drive miners to wind down operations when profitability limits are crossed. Simultaneously miners of the new proof-of-work algorithm would deploy new mining hardware to capture the increasing amount of block reward available to them. At the end of the transition the block is still subject to two proof-of-work requirements, but the original proof-of-work is at minimum difficulty. During this transition un-upgraded clients will continue to see blocks produced with ever smaller difficulty, until the difficulty reaches its minimum value.

In this scenario total-work security would be progressively reduced for old clients who only see and validate the original proof-of-work algorithm, but this degradation of security will occur over a period of years, be visible in reduction of hashrate, and full security could be restored for any node at any time by upgrading at their convenience. Centralizing mining incentives would emerge rather rapidly as the reward is reduced for the set of old-PoW miners, but we will show that this has no adverse effects on censorship resistance in our scheme. For the new-PoW mining network a decentralization-inducing growth of subsidy provides some measure of protection for the duration of the transition period.

As stated so far, there remains the problem of providing the mechanism by which the sliding block reward is shared between the two sets of miners in a dual-PoW setup where a single block is subject to both proof-of-work requirements. This requires some level of coordination between both sets of miners, and the more coordination that is required the less decentralized mining can be. The solution we adopt instead is to have separate block chains with loose synchronization of state, which sidesteps most of the coordination complications entirely.

A. Merge Mining Is Also a No-Op Proof-of-Work Upgrade

We do not wish to imply that that this proposal is in any way an adversarial move against the current set of bitcoin/double-SHA256 miners. The scheme we develop for performing a proof-of-work upgrade has other desirable benefits, and the fact that the scheme requires changing the work function is merely an artifact of how the mechanism works. Whether switching to a new proof-of-work algorithm that invalidates existing ASIC investments is desirable is outside the scope of what is discussed here.

Requiring a proof-of-work "change" a necessary side effect of the scheme we are developing here. However, that new proof-of-work algorithm could be double-SHA256 merge mining, which would be a no-op proof-of-work "upgrade" as all the same double-SHA256 ASIC hardware in use today would be able to simultaneously mine both, or at least could be made to do so with a software or firmware upgrade.

Or the new proof-of-work function could be something completely different, rendering most existing hardware useless once the difficulty transition is complete. Either approach is permitted by this proposal, and we will make no further comment on what this choice should be, which is entirely orthogonal to the adoption of forward blocks as a scaling solution.

IV. Loosely-Coupled Chain Dependency via "Forward Blocks"

Rather than have a single block subject to multiple proof-of-work checks, and all the complications that would arise from that, we instead propose using a divergent fork of the original chain where the old-PoW miners look to the new-PoW blocks for ordering of transactions, and thus a consistent ledger is maintained. This new chain which forces the ordering of transactions is called the *forward block chain* and we will now build up the core concept in parts.

A. Using a "Forced Hard-Fork" To Change Work Functions

To begin with, consider a hard-fork proof-of-work upgrade, with no replay protections: at a specified activation height the new set of miners fork off from the historical chain, creating blocks conforming to the new proof-of-work requirement, and carry over the same set of transactions in their mempool for inclusion in future blocks. The rules for transaction validation remain the identical to the original chain, but any block-level consensus rules—such as aggregate limits—are free to be changed without regard for compatibility with existing clients, as forward compatibility is necessarily broken already.

Now combine this with what has been variously called "forced hard-forks," "soft hard-forks," or "evil forks" [*sic*]: a majority of miners on the original chain at the time of the split refuse to build upon any transaction-including blocks.

This is how the transition begins: a hard-fork chain that is continuing to mine compatible transactions but using a new-PoW function, and the original chain mining empty blocks with valid old-PoW headers but devoid of any user transactions.

To make sure that blocks continue to be mined on both chains, the block reward must be made to slowly transition from the chain continuing the old-PoW to the new at a rate not much greater than normal variance in mining income. Old-PoW miners would witness their income slowly dropping, while new-PoW miners start receiving equivalent increases in their block rewards. Over time less efficient old-PoW miners will wind down their mining operations causing a ramping down of difficulty in proportion to the reduction in block reward. At the same time, investments in new-PoW mining hardware to capture the growing share of block rewards on that chain will drive its mining difficulty (and security) up.

As the name implies, the *forced hard-fork* scenario was designed to force old nodes to upgrade to newer software versions which understand the new proof-of-work/extension blocks where transactions actually reside, as otherwise they are unable to witness any transaction confirm. It is considered by some to be "safe" because old nodes cannot be tricked into seeing a differing spend history on the old chain compared to the new. But sometimes not being aware of transactions can be just as dangerous as seeing the wrong transactions—e.g. not seeing a channel closure from old state—and it is coercive in requiring an upgrade. In the next two subsections we will explore how we can restore transaction processing on the old-PoW chain without risking a divergent history.

B. Preventing Double Spends with a Transaction Queue

To restore transaction processing for un-upgraded clients we alter this "forced hard-fork" setup in the following ways:

1) The coinbase outputs of the new-PoW chain are **not** entered into the UTXO set. They serve a purpose that is explained in Section V, but they are not themselves spendable in either chain.
2) The coinbase outputs of the old-PoW chain enter the UTXO set of **both** chains, after a maturation process explained in Section IV-C. To be clear, this means a post-fork coinbase output of the old-PoW chain will **also** be spendable on the new-PoW chain after the maturation process following synchronization of state between them.

With these changes a transaction valid on one chain is also valid on the other, and this remains true indefinitely into the future, so long as a consistent ordering of transactions is used in both chains.

At this point in the explanation a double spend is trivially not possible because the old-PoW chain lacks any transactions at all, other than its coinbase, so UTXO set compatibility may seem an academic concern. But we now relax this restriction without enabling transaction history to diverge:

3) The coinbase of the old-PoW chain is allowed to commit to block headers of the new chain, if doing so extends the chain of known headers to a more-work tip of equal or greater height than was previously known from prior commitments.
4) If at any time a header is buried by N or more further block headers in the same chain, then the buried header is *locked-in* and cannot be subject to a later header commitment reorg, except by reorg'ing the old-PoW chain to remove the Nth commitment.
5) Should an attempt be made to commit to a chain of block headers that would reorg an already locked-in header or even just remove its locked-in / Nth confirmation status, the block containing that commitment is invalid.
6) Adding an Nth confirmation of a block requires that buried and now locked-in header reference a valid block.

Thus full-block validation of the other chain is allowed to happen asynchronously during the maturation period, while any block referenced by a locked-in header **must** be valid. It also means that a block header commitment could be to an invalid block, but only so long as that block header never receives a Nth confirmation—or else the block containing that Nth confirmation commitment is invalid.

7) Once the old-PoW chain has accumulated N confirmations of a new chain block header, resulting in that new-chain block acquiring "locked-in" status, the transactions in that block are added to the end of an first in, first out transaction queue. This queue is initially empty at the point of activation of the fork, and transactions are added to the queue after creation/validation of the block containing the Nth confirmation commitment, so the earliest a transaction can appear is in the following block.
8) When the transaction queue is non-empty, and so long as the queue remains not-empty, new blocks in the original chain are required to include transactions from that queue, in order, for as long as the next transaction in the queue is valid and including it would not violate aggregate limits for the block.

A transaction will only ever be invalid for reasons of finality (time-locks or sequence locks) or coinbase maturity, both being infrequent edge cases[2]that resolve with time, after which blocks contain transactions again, pulling from the queue where the transaction processing stopped.

Observe that we have now restored transaction processing on the old chain without permitting divergent history, since the selection and ordering of transactions on the old chain is

[2]It is possible for a transaction to be temporarily invalid because finality and coinbase maturity requirements are checked differently on both chains. As a concrete example, a transaction with a 144-block relative lock-time must be separated from its input by that many blocks on *both* both the original chain and the post-fork chain. If the transaction were to first confirm exactly 144 blocks after its input on the new-PoW chain, but one of the interleaving blocks is empty, then the old-PoW chain could process the interleaving transactions in only 143 blocks. Doing so however would violate the 144-block relative lock-time.

fully determined by the lock-in of block header commitments to the new chain, itself an irreversible process within the context of a single chain history. A large reorg of the new chain can still be handled, but at the cost of requiring a reorg of the original chain to a point before the Nth confirmation of the point of divergence.

Simply put, the miners of the new chain provide a deterministic order of transactions which **must** be followed by in the continuation of the original chain after the point of activation. The old-PoW miners individually lose their control over transaction selection, as that is now fully determined by the most-confirmed-work of the new-PoW chain.[3] To prevent divergent UTXO sets the coinbase outputs of the old chain are used to collect fees, pay out miners of both chains, and other purposes requiring coinbase maturation, by a process explained in Section IV-C and Section V.

The purpose of the new-PoW chain is to provide a transaction ordering which **must** be used by the old-PoW miners in the construction of their blocks. For this reason a new-PoW block called a *forward block* and the new chain the *forward block chain* in analogy to the concept in finance and real estate of a "forward transaction"—the purchase of a good, service, or property **now** at a certain price for delivery on a fixed future date—or the "forward observer" of an infantry battalion which scouts in advance the route taken by a unit on the move. The miners of the new chain are choosing which transactions and in what order the old-PoW miners are required to include in their chain, after N confirmations of the block headers has been achieved in the block header commitments.

The old-PoW block is called a *compatibility block* and the original chain the *compatibility chain* because it relays, after some delay, transactions in the already agreed upon order to non-upgraded clients, and processes coinbase payouts in a way that would be seen and identically processed by old and new clients alike.

C. Coinbase Confirmation on the Forward Block Chain

We are now ready to explain how compatibility chain coinbases enter the UTXO set of the forward block chain, without which there would be no way for miners of either chain or recipients of coinbase payouts in general to collect their due. The mechanism for making coinbases spendable is a similar set of block header commitments and maturity rules as was used in Section IV-B to construct the transaction queue on the compatibility chain, this time on the forward block chain and with regard to coinbase payouts:

1) The coinbase of forward blocks are allowed to commit to block headers of the compatibility chain, if doing so extends the chain of known headers to a more-work tip than was previously known from prior commitments. A reorg of no more than N block headers is allowed in the process.[4] Only block headers are validated until the Nth confirmation, at which point block validity is also required.
2) Once the forward block chain has accumulated N confirmations of a compatibility block header, that block's coinbase outputs are entered into the UTXO set with an M-block maturity (allowing them to be spent in M additional blocks after the inclusion of the N-th block header commitment).

N is a free protocol choice known as the *forward-block settlement period*, and M is the *coinbase maturity period*, both of which combine to define the total length of time it takes for a coinbase payout to mature. We use $N = 96$ and $M = 100$ in our implementation.[5] Having M greater than N doesn't provide any higher level of security, but has the advantage of not requiring alteration of any existing 100-block coinbase maturity handling code, at the cost of waiting only an extra 4 blocks.

We can now construct a complete timeline for what happens when the new rules are activated:

1) At the time of activation of the fork the forward block chain splits from the compatibility chain, and the forward block miners begin minting blocks using the new-PoW and transactions selected from their mempools, using possibly different aggregate limits and other block-level consensus rule changes. The non-coinbase transactions included in the forward block chain would be valid on the original chain, in the counter-factual scenario that the new rules had not activated (assuming the equivalent block height and block time for time-locks, the equivalent ordering and separation for sequence locks, etc.).
2) Simultaneously the upgraded original-PoW miners, constituting a supermajority of hash power at the time of activation, begin mining blocks for the compatibility chain, initially consisting of only a coinbase transaction which may have zero, one, or more commitments to new-PoW forward block header(s).
3) As blocks progress on the compatibility chain, forward block miners include reciprocal commitments in the coinbases of their own forward blocks to the advancing tip of the old-PoW compatibility chain.
4) When in either chain knowledge of the other exceeds N blocks past the point of fork, asymmetrical effects occur:
    a) Once the compatibility chain has added knowledge of a forward chain tip N or more blocks beyond the point of fork, transactions other than the coinbase in forward blocks with N or more known confirmations are added to the transaction queue in the same order as they were seen in the

[3] Although when merged mining is used, the old-PoW miners are also the new-PoW miners.

[4] Technically the two separate N parameters used in the compatibility-block and forward-block settlement periods could be different, but there is no security benefit for that to be so, and doing so would only add delays to the maturation process.

[5] In Section X we recommend 15 min forward block intervals, which means $N = 96$ blocks is approximately 1 day.

forward block chain. Starting with the very next block, compatibility chain miners now reject any compatibility block which does not draw from the transaction queue filled by prior blocks, in the order added, allowing for less-than-full blocks only when the next transaction in the queue breaks a finality or maturation rule.

b) Once the forward chain has added knowledge of a compatibility chain tip N or more blocks beyond the point of fork, the coinbase outputs of compatibility blocks with N or more known confirmations are added to the UTXO set and become spendable after an M block maturity period.

For an upgraded wallet, confirmations on the forward block chain are the only confirmations that matter, as later confirmation on the compatibility chain is predetermined. For a non-upgraded wallet there is a perceptible and persistent delay in confirmation on the original/compatibility chain that they see, but transactions do eventually confirm and progress is made. So long as applications are written to gracefully handle confirmation delays, applications running against pre-fork versions of the client software will continue to work during the transition.

V. Managing the Shared Coinbase Payout Queue

A necessary condition for transaction compatibility across the forward block and compatibility chains is that the forward block coinbase outputs **not** enter the UTXO set, as they are not visible to un-upgraded nodes following the compatibility chain, and any transactions spending such outputs would be considered invalid by those nodes. However it is necessary that forward block miners be able to collect subsidy and fees, and generally be paid for the work that they do, which is a hard thing to accomplish when the outputs they have control over are unspendable.

To solve this problem we provide a mechanism for delayed cross-chain state synchronization, and use the block reward on the compatibility chain to pay out both sets of miners. In doing so we develop a mechanism for synchronizing state across chains or separate ledgers generally which we will reuse in our discussions of sharding in Section IX and potential future privacy-enhancing extensions in Section XI. However the best way to introduce this mechanism is through the one other solution it is designed to solve: providing a smooth difficulty transition between the old an new proof-of-work functions which we will cover in Section VI. But first we cover in this section the basic underlying mechanisms involved.

Forward compatibility demands that all block reward (subsidy and fees) be processed in the compatibility block chain. However payouts for both groups of miners are unconnected to whatever block reward happens to be available to a specific compatibility block. We resolve this issue by aggregating block rewards into a common fund that is used to payout miners on a first-in, first-out basis. Specifically:

1) Forward block miners are allowed a portion of the subsidy and fees of the blocks they mine. They specify the desired destination for these funds in the dummy outputs of their coinbase transactions.
2) Compatibility block miners are allotted a portion of the remaining block reward from already locked-in forward blocks. They too specify the desired destination for their share of the coinbase reward in a special commitment in their blocks.

The mechanism for dividing block reward between the forward and compatibility miners is the topic of Section VI. Suffice to say there is a fair mechanism for determining how much of the subsidy and fees are allocated to each set of miners, and the reward made available to compatibility chain miners—which depends on the stochastic forward block finding rate—is smoothed out.

When a compatibility block is processed, the desired outputs of the compatibility block miner are added to a first-in, first-out *coinbase payout queue*. The coinbase outputs of any forward block that achieves its Nth confirmation and locked-in status is also added to the end of this queue.

The coinbase of the compatibility block is required to pay out the collected subsidy and fees to outputs pulled from the payout queue in order. If there is insufficient reward remaining to process the next coinbase payout, or if the payout queue is empty, the remaining block reward is carried forward to cover future payouts. Procedurally:

3) The compatibility block miner commits to their own outputs, of sum total equal to or less than the output of the compatibility block reward function R specified in Section VI. The committed outputs are added to a payout queue maintained by each validator, before processing any other miner commitments.
4) The compatibility block miner also commits to a possibly zero-length chain of headers extending the known tip of the forward block chain. When a forward block receives its Nth confirmation reported to the compatibility chain, the forward block's coinbase outputs are also added to the same payout queue.
5) The normal block reward that would have been available for the old-PoW miners if (counter-factually) the new rules hadn't activated, plus any carry-over from the prior block, is used to fund outputs from the queue in the order added. If at any point there is insufficient funds to pay out the next output or if the payout queue is empty, the remaining block reward is put into a special *carry-forward output*.
6) Once the compatibility block receives N confirmations in block headers reported to the forward block chain, all the outputs of its coinbase except for the carry-forward output are entered into the UTXO set of the forward block chain, with an M block maturity period.
7) Once the carry-forward output matures according to the compatibility block rules, it is spent by a special miner-generated transaction, the last transaction of the block in which it became spendable. If there are

still payouts to be made from the payout queue, the necessary amount of funds are released as "fee" to be collected in the coinbase and paid out, otherwise the remaining funds are placed in a carry-forward output of this transaction to process coinbase payouts of the next block, or as needed in the future.

Compatibility chain miners are required to process outputs (meaning, include the payout in the block's coinbase transaction) taken from the payout queue, in order, until one of three stopping conditions are reached: (1) the payout queue is exhausted; (2) the block reward and carry-forward balance combined is less than the next item in the payout queue; or (3) adding additional coinbase outputs would exceed aggregate block limits. Remaining funds from the block reward and carry-forward balance are placed in the coinbase and last-transaction carry-forward outputs, respectfully.

One final rule settles contention between use of block space for making coinbase payouts and processing transactions:

8) The coinbase payout queue is processed before miner commitments or the transaction queue, so the contribution of the coinbase towards aggregate block limits is known.

The above rules are consensus enforced—blocks are considered invalid if they do not adhere to all of the above rules.

Now we have the right pieces in place to explain how mining rewards are claimed by both sets of miners:

1) A forward block is mined, containing a fake coinbase transaction for the purpose of specifying where the forward block miner wants their share of the block reward to go, as well as various other required or optional commitments.
2) Since the compatibility block reward function R is a function of the locked-in forward blocks over a specific period and there is very little cost to including a forward block header commitment, compatibility miners have incentive to include any valid forward block headers they know about in their block templates.
3) When a compatibility block is found, its coinbase contains payouts pulled from the front of the queue, and its own payouts are added to the end of the queue—both the compatibility miner's payout preferences and the outputs of any newly locked-in forward blocks.
4) Desiring to be paid for their prior work, the forward block miners have incentive to include commitments to the longest compatibility block chain they know about in their own block templates.
5) When a forward block is found, the coinbase transactions of any newly locked-in compatibility blocks enter the UTXO set, subject to the usual 100 block maturation delay.

By this mechanism both forward and compatibility block miners receive their pay by adding commitments to each other's blocks, maintaining the loosely coupled connection between the two chains.

VI. A Smooth Difficulty Transition Between Proof-of-Work Functions

Forward blocks and the compatibility chain work together to provide a smooth transition in difficulty from the old-PoW algorithm to the new. Achieving this difficulty transition requires a proportional allocation of reward between old-PoW and new-PoW miners that adjusts continuously and slowly enough over time as to prevent network disruption:

1) Forward-block miners are allowed a proportion $P \in [0,1]$ of the subsidy plus fees of the their own blocks. The remaining $1-P$ portion is set aside for the compatibility block miners.
   For example, if $P = 25\%$ and the subsidy is $6.25\,\text{btc}$ and fees are $1.75\,\text{btc}$, then the forward-block miner is allowed to put up to $0.25 * (6.25\,\text{btc} + 1.75\,\text{btc}) = 2\,\text{btc}$ in the forward block's non-spendable coinbase outputs.
2) At the point of activation P is assigned a value that is quite small (almost zero), and the new proof of work is starts with a minimal difficulty requirement. Over a sufficiently long period, the value P is increased slowly and without discontinuity until it is has the approximate value $P = 1$.
3) A smoothing filter controller $R(\text{tip})$ takes as input the compatibility block headers, including locked-in forward block headers and their coinbase transactions, and outputs an allowed block reward for each compatibility block. The content of the compatibility block, including its own forward block commitments, only affects the reward of future blocks.

The naive solution of having the compatibility block miners simply fill out the remaining subsidy of the forward blocks they confirm runs into difficulty with the loosely coupled connection between the two chains. The inherently random process of mining blocks means that some compatibility blocks will be responsible for confirming multiple forward block headers whereas others will include none at all, to say nothing of the perverse incentives this creates for reorgs. The solution of applying a smoothing filter transforms the naturally fluctuating supply into a smooth, continuous reward function to prevent adverse mining incentives, and has an added advantage of making all portions of the block except for miner commitments and payout destinations fully deterministic from prior-block data.

A simple controller that would work is "$R(\text{tip})$ is sum of compatibility block portions of the block reward of locked-in forward blocks minus payments already made, scaled by $1/k$" for some suitably large value of k, fixed in the consensus rules, that puts added variance within the range of already existing natural mining variance, and with some initial conditions to prevent block reward discontinuity at the point of activation.[6]

[6]The actual scale factor used should incorporate the current warp factor Q, as defined in Section VII: $\frac{1}{k\lfloor Q \rfloor}$. Incorporating Q as a component of the scaling factor bounds the increase in variance that would result from a steadily decreasing inter-block interval caused by an increasing aggregate weight limit on the forward block chain.

While this serves as an adequate existence proof of a solution, design of optimal controllers for this application is a topic deserving its own dedicated treatment, and we will not do so here. We remain open to the possibility that better controllers exist and invite more research in this area. However it is worth noting some design requirements:

1) The sum of the controller outputs over time must equal the total amount made available to compatibility block miners as inputs to the controller. The compatibility block miners must not earn more (or less) than their fair share.
2) This fairness condition must hold true even if forward block miners coordinate to control the schedule of compatibility block income being registered in the forward blocks (e.g. by including or delaying high-fee transactions).
3) This fairness condition must hold true even if compatibility block miners coordinate to control the booking of forward block income by choosing to commit to forward blocks at certain times.
4) This fairness condition must not be sensitive to other factors under miner control (e.g. block timestamps, or block stuffing with miner-generated transactions).

By providing a smooth transition from the regime of $P \approx 0$ to $P \approx 1$ over a period of time comparable to hardware replacement cycles, the loss of reward by old-PoW miners occurs slowly enough to be planned for and without risk of sharp discontinuity in hashrate and long block delays, while still achieving the goal of having the old-PoW be trivial to maintain (e.g. difficulty 1) at the end of the transition period.

A. *Smoothing Out Bitcoin's Subsidy Schedule*

The coinbase payout queue dissociates the block reward paid out to miners of both chains from the schedule of subsidy collected on the compatibility chain and used to process the coinbase payout queue. The reward received by the forward block miner depends on the block that they build, and the reward received by compatibility block miners depends on the history of forward blocks locked-in on that chain. Neither is necessarily tied to the subsidy-dropping schedule originally set by Satoshi—the block reward for both sets of miners is ultimately determined by the forward block chain, which has **no** requirement to match bitcoin's subsidy schedule:

- If there is greater subsidy on the forward block chain than the compatibility chain—due to a faster subsidy schedule or less than expected number of compatibility blocks—then the coinbase payout queue grows in length and miners have to wait correspondingly longer to receive payment.
- If there is less subsidy on the forward block chain than the compatibility chain—due to a slower subsidy schedule or greater than expected number of compatibility blocks—then the coinbase payout queue may be processed completely and excess coinage accumulates in the carry-forward balance.

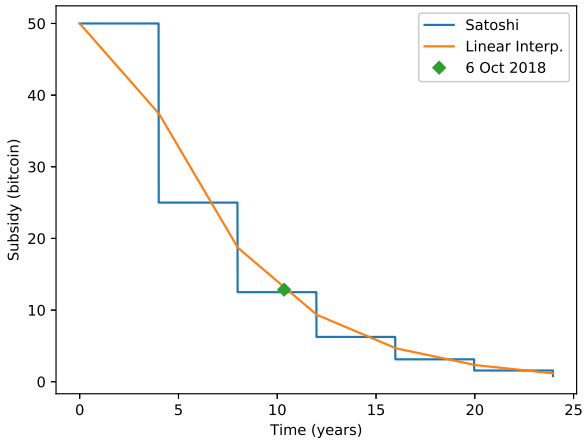

Fig. 1. Alternative Subsidy Curve

It is certainly not a problem for a carry-forward balance to exist, even a large one. And while a coinbase payout queue causes delays in coin maturation, it does not otherwise cause any other issues for miners or validators. Because of this we have the freedom to change the subsidy schedule for forward blocks, so long as:

1) The cumulative subsidy never exceeds the 21×10^6 btc total monetary base; and
2) The cumulative subsidy at any given time does not exceed the bitcoin subsidy schedule by too large a margin as to cause unreasonable delays in the coinbase payment queue.

So long as these conditions are met the coinbase payout queue may vary in size, but will not grow to excess, beyond what can be handled with a reasonable delay to coinbase maturation. To prevent unreasonable delays the subsidy curve for the forward block chain needs to approximately match the pre-activation subsidy curve, but it need not follow exactly the same curve as some variance or local deviation is allowed.[7] In particular, it is possible to "even out" the subsidy curve, and thereby remove the sharp discontinuities which presently exist, the so-called "halvenings."

The simplest correction would be to transform the 4 yr constant subsidy per halving period into a linear reduction over 8 yr instead, resulting in a smooth subsidy curve without any discontinuities, as shown in Figure 1. Regions of the graph for which the original/Satoshi subsidy curve exceed the linear-interpolation curve are periods of time where the excess subsidy on the compatibility chain would be banked in the carry-forward balance. That banked value would be used to pay out miners in those regions where the original subsidy value is less than the smoothed approximation.

In this way the dangerously incentive-incompatible "halvening" phenomenon can be removed from bitcoin, resulting in a continuous and predictable subsidy drop over time.

[7] Indeed, with the 15 min block times recommended in Section X, the per-block subsidy schedule would need to be increased 50% if nothing else.

VII. Increasing the Block Weight Limit with a Flexible Cap and a Flux Capacitor

The aggregate block limits of forward blocks are not restricted to be the same as the aggregate limits enforced in pre-activation consensus rules (and carried over to the compatibility chain); both the maximum block weight and signature operation limits[8] can have different values enforced in forward and compatibility blocks, so long as no single transaction exceeds the per-block limit of the compatibility chain. If the forward block consistently has more transaction-weight than can fit in a compatibility block, transactions will accumulate for as long as this remains the case; with larger forward blocks comes increasing pressure on the compatibility block miners to process the transaction and coinbase payout queues.

But if the difficulty of the compatibility block chain were to be lowered, then the same level of network hashrate would generate more compatibility blocks within a comparable interval of time. Under normal circumstances this would cause the difficulty to increase at the next adjustment up to the requisite level to maintain 599.7 s expected block times. However a flaw in bitcoin's difficulty adjustment algorithm can be used to trigger a difficulty lowering and used again to prevent any follow-up re-adjustments, thereby increasing the frequency at which compatibility blocks are found for as long as the flaw is exploited. This flaw is known as the time-warp bug: the 2016-block difficulty adjustment interval is larger than the 2015-block window used to make the calculation,[9]allowing for the careful use of bogus timestamps at the endpoints of this window to arbitrarily adjust difficulty in either direction up to the maximum adjustment limit of $\pm$4x.

We can combine both of these techniques of (1) increasing aggregate limits for forward blocks, and (2) decreasing the inter-block interval for compatibility blocks to achieve a higher transaction processing throughput, achieving effectively the same results as direct block weight limit increase on the original chain, but in a soft-fork compatible way. We will later show that while such an aggregate limit increase raises validation costs for all nodes, large gains in transaction processing throughput can be had without increasing other centralization risks.

A. Initial Parameters of the Forward Block Chain

On activation of forward blocks we are given a one-time opportunity to alter the block-level consensus rules of the forward block chain in ways that on the compatibility chain would require a hard-fork. We specifically recommend two changes that permit larger transaction processing throughput without decreasing censorship resistance:

1) Increase the inter-block interval to a larger value than the original chain's 10 min / 599.7 s target to reduce the ratio of network latency to the inter-block interval. We recommend a value of 15 min / 900 s as a compromise between transaction processing throughput and utility (confirmation time).
2) Raise the (initial) forward block maximum weight from 4 MWe to a recommended value of 6 MWe, permitting approximately the same transaction processing rate per unit time as the original chain.

Increasing the aggregate block weight limit while simultaneously increasing the inter-block interval is a net gain for centralization resistance even though the on-chain settlement throughput remains the same. This is because the fundamental network latency is unchanged, and the network latency portion of block propagation now constitutes a smaller percentage of the total inter-block time. This permits a slightly higher forward block weight limit for the same level of censorship resistance.

The full initial parameters of the forward block chain and their justification are given in Section X.

B. A Flexible Cap on Forward Block Weight

In the new consensus rules on the forward block chain we now permit under certain circumstances a larger or smaller block weight limit, at the discretion of the new-PoW miner according to the following rules:

1) Forward block miners are allowed to optionally *bias* their proof-of-work target by up to $\pm$25%. This essentially makes their block between 25% easier and 25% harder to solve, which implicitly increases or reduces the miner's block reward expectation.
2) Biasing the work target has an inverse-quadratic relationship with the adjustment made to the aggregate weight limit for the block relative to some baseline. Specifically, for a bias value $x = \Delta T/T_0$ (permitted values of $x \in [-0.25, 0.25]$), the permitted block weight w is given by:

$$w(x) = w_0(2x - 4x^2) \tag{1}$$

Equation (1) is depicted in Figure 2.

Note that lowering the proof-of-work target, which permits slightly larger blocks that are harder to find, is asymmetric with respect to raising the target, which permits much smaller blocks that are easier to find. A non-linear equation is required for there to be a fixed optimal value for a given transaction fee distribution.

The largest temporary block-weight increase that can be obtained is +25%, with a +25% bias to the proof-of-work target, whereas the smallest temporary block-weight is -75% the default value, obtained with a -25% bias.

Critically, this bias adjustment does not affect the representative work of the block for the purpose of determining the

[8]Throughout this text we will refer to the block weight limit as the defining aggregate limit, but we mean both. An increase or decrease of one limit in a forward block chain would presumably be accompanied by a proportional increase or decrease of the other limit, or the forward block chain may forego having a separate sigop limit entirely.

[9]This is, incidentally, why bitcoin targets approx. 599.7 s block intervals rather than exactly 10 min = 600 s:

$$600\,\text{s} * \frac{2015}{2016} \approx 599.7\,\text{s}$$

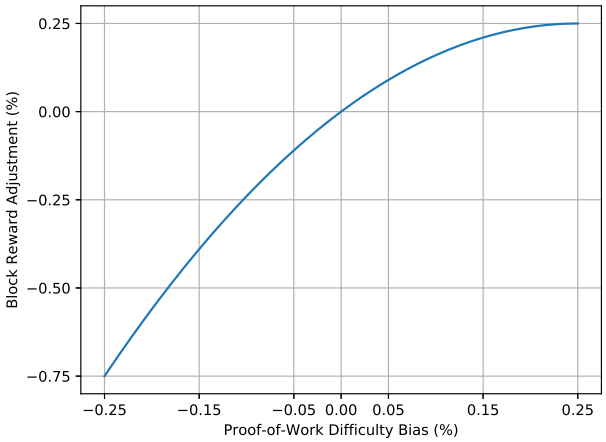

Fig. 2. Offset reward as a function of difficulty bias.

most-work chain. A miner cannot "win out" over another's block by choosing a higher work bias.

3) Miners provide in their blocks a declared weight, which must be greater than or equal to the actual weight of the block under current consensus rules, and less than or equal to the maximum weight permitted given their choice of work bias.[10]
4) Every L forward blocks (we propose $L = 2016$, approx. 3 weeks), two things happen:
   a) The base limit w_0 adjusts to the (possibly gain limited) average of the past L declared weights.
   b) The new-PoW difficulty adjusts up or down as necessary to maintain the 15 min inter-block interval.[11]

With the above rules, the forward block chain is able to provide burst transaction processing capacity as needed, and automatically adjusts its base limits in response to real demand. Miners are prevented from unilaterally increasing aggregate limits due to the prohibitive costs they would suffer directly as a result.

Should the forward block chain be subject to an aggregate sigop limit, it too could be raised or lowered by a proportional amount. However it is the opinion of this author that advances in validation speed and the elimination of quadratic costs have rendered the sigop limit mostly redundant, and as such the forward block chain should not be subject to an aggregate sigop limit at all.

C. Processing Larger Blocks on the Compatibility Chain via a Coordinated Time-Warp

Now that we've established how the forward block chain manages its aggregate weight limit, we can look at how the compatibility chain adjusts its inter-block interval to achieve a matching transaction processing rate, through coordinated exploitation of the time-warp bug.

Upon activation, the we grant the compatibility chain new consensus rules governing its block timestamps:

1) If a compatibility block is **not** the last block in a difficulty adjustment period (the block height is non-zero modulo 2016):
   a) If after constructing the block the transaction queue is empty, no additional rules apply.
   b) Otherwise, if the next transaction in the queue is non-final for reasons of a time-based lock-time or sequence-lock, the block timestamp is set to the minimum value necessary to satisfy that lock condition.
   c) Otherwise, the block's timestamp **must** be set to its minimum allowed value, which is one more than the median of the 11 blocks prior.

The implication of this rule is that so long as compatibility blocks are full, the timestamps of compatibility block headers advance no more quickly than one second every six blocks, until such time as the queue is emptied or the next item in the queue requires a higher timestamp value. As the flexible weight limit of the forward block chain is expanded, the transaction queue will fill up and the difficulty of the comptibility chain will need to be lowered to keep up. Making sure timestamps advance as slowly as possible ensures this can happen.

The mechanism for causing the difficulty adjustment is a bit more complicated:

2) The compatibility block miner makes a commitment alongside forward block headers to the aggregate declared weight of the forward block chain up to that point.[12] For the header to transition to locked-in status, the commitment must match the same values reported in the forward block chain, or else the compatibility block containing the Nth forward block commitment is invalid.
3) Since the `block.nTime` field now has its value subject to additional rules when blocks are full, the compatibility block miner commits to the wall-clock time in the `nLockTime` field of its coinbase transaction.[13] This commitment is subject to roughly the same rules that have always applied to the block timestamp:
   a) The coinbase `nLockTime` must be greater than the median of the past 23 coinbase `nLockTime`'s.[14]

[10]The declared weight is specified and constrained to a range rather than being a calculated value for reasons of forward compatibility with future soft-fork extensions.

[11]If bitcoin's difficulty adjustment algorithm is used, the time-warp bug should be fixed for forward blocks.

[12]If sharding is used (see Section IX), there is one aggregate value per shard chain and the number considered here is the sum of the aggregate locked-in weight across all shards.

[13]Implementation detail: the actual commitment would be to `LOCKTIME_THRESHOLD` plus the zero-clamped difference between the wall-clock time and the median of the past 11 block timestamps, or some other calculation which ensures the value is always larger than `LOCKTIME_THRESHOLD` but less than the median past 11 block timestamps. The above rules should be interpreted accordingly. Later references to this field refer to the actual UNIX timestamp, not the `LOCKTIME_THRESHOLD`-relative encoded value.

b) The coinbase `nLockTime` commitment must be no more than two hours ahead of the current wall-clock time.

4) If a compatibility block **is** the last block in a difficulty adjustment period (the block height is zero modulo 2016), then the block's timestamp is required to be set according to the following procedure:
   a) A ratio Q, the *warp factor*, is calculated as the ratio of forward block utilization[15] to the original chain's settlement capacity:

$$Q = \frac{w}{6\,\mathrm{MWe}} \tag{2}$$

   The integer $\lceil Q \rceil$ is the same value, rounded up.
   b) Using the last $2016 \times \lceil Q \rceil$ compatibility blocks' wall-clock timestamp information (the coinbase `nLockTime` fields, not the block header timestamp), a calculation is made of which proof-of-work target over that approximately two week period would have yielded Q blocks in expectation every 600 s, the necessary rate in order to process forward blocks at their present size.
   c) The original/Satoshi difficulty adjustment formula is applied in reverse to calculate the block timestamp which would cause in an adjustment to the proof-of-work target calculated above.
   d) The block timestamp is constrained by clamping to be no smaller than its minimum allowed value of one more than the 11 block median time past from block headers, and no greater than 14 400 s (4 h) beyond the last block's coinbase committed wall-clock value.

Together these rules cause the compatibility chain's block rate to adapt to whatever the forward block chain's transaction processing throughput is, while still remaining responsive to changes in its own network hash rate, while also preventing compatibility block timestamps from being set so high as to cause a network split due to a future-dated block.

We've now covered how (1) the forward block chain can grow (or shrink) in response to real demand; and (2) how the time-warp bug can be exploited to adjust the inter-block interval of the compatibility chain to make sure every forward-block transaction reaches every node. We look in Section VIII at what the hard limits are to scaling transaction processing capacity via this approach.

D. The Centralization Risks of a Flexible Weight Limit

The compatibility chain is restricted to increasing its transaction processing throughput by decreasing its inter-block interval only, though exploiting the time-warp bug. This increases validation costs for everyone, and adds centralizing pressure for compatibility miners. Furthermore, decreasing the compatibility miner's share of the block reward as part of the proof-of-work transition provides additional centralizing pressure so long as the distribution of cheap electricity and efficient mining hardware remains unequal: as lower block reward reduces mining profits for everyone, those with higher costs are driven out of business first, and what remains is an oligarchy of miners with access to the best hardware, the cheapest electricity, and/or the largest subsidies.

With the forward block chain we have a choice of changing the block weight limit, adjusting the inter-block interval, or both. Increasing forward block weight limit is preferable to decreasing the inter-block rate because of the non-linear relationship between inter-block time and centralization risk: some risks factors relate to block propagation delays caused by the speed of light, and these risks are independent of the size of a block. We even go so far as to recommend lengthening the inter-block time on the forward block chain in order to achieve a small but important boost in censorship resistance.

Considered by itself, indirectly increasing transaction processing throughput on the compatibility chain by shortening the inter-block interval proportionally increases the cost of validation, but does **not** affect censorship resistance. This is because the ordering of transactions is fully determined by the forward blocks, and so with regard to censorship the distribution of miners on the compatibility chain doesn't matter so long as the forward block chain makes progress independent of the compatibility chain.

More frequent compatibility blocks would be filled by larger forward blocks, however, and increasing the forward block size both increases the cost of validation for full nodes, and harms censorship resistance by disproportionately favoring large hashrate new-PoW miners. In Section IX we examine a solution that allows us to make more smaller, more frequent forward blocks while still maintaining the benefits of a long inter-block interval. But first we look at just how much scaling advantage we can extract out of the time-warp bug on the compatibility chain.

VIII. The High End of Scaling Limits on the Compatibility Chain

While the ability to increase the block rate must be limited so as to constrain resource utilization in validators and retain censorship resistance, which both together provide the properties we care about and call *decentralization*, let's set this aside for just a moment and consider just what the limits are of using time-warp to increase settlement capacity on the compatibility chain. After establishing what the protocol-allowed upper limits are, we will then look at whether these limits would ever be reasonable, and ways in which the forward block size and compatibility block interval can be limited so as to preserve decentralization properties that we care about in the interim.

Because a block's timestamp is constrained to be larger than the median timestamp of the 11 blocks prior, a sequence

[14] The median-time-past window is doubled to prevent ratcheting forward of the timestamp as a result of normal variance once compatibility chain scaling limits are reached.

[15] Aggregate across all shards, if sharding is used:

$$Q = \frac{\sum_{i=1}^{28} w_i}{6\,\mathrm{MWe}}$$

of up to 6 blocks may share the same timestamp before consensus rules require block-time to tick forward. Thus there cannot be more than 6 blocks per second in expectation before block time advances at a rate more quickly than actual (wall clock) time, which becomes problematic as block time is not allowed to exceed actual/wall clock time by more than 2 h at the tip.

At this maximum sustainable rate 14.4 GWe of transactions could be processed in 3600 full compatibility blocks every 10 min,[16] with one compatibility block every 167 ms in expectation. This is an effective 3600x transaction processing rate increase and represents the actual hard limit allowed by the bitcoin protocol, before this scaling approach would break forward compatibility with older clients. Should this limit be reached, the block timestamps of the compatibility chain would be at least two weeks old and advancing by one tick every six blocks, with each block arriving every ⅙th of a second in expectation. Every 2016 blocks, which would occur every five and half minutes, normal difficulty adjustment would be prevented by setting the timestamp to be approximately the present wall-clock time, two weeks later than the 2015th prior block timestamp. The next block that follows returns to the pattern of block timestamps increasing as slowly as possible.

In a Section X we will evaluate how ridiculous—or not—this real upper limit is, whether it would ever be reasonable to allow such upper limits to be reached, and by what mechanism lower limits could be enforced in the interim.

IX. Sharding across Multiple Forward Block Chains

There's a one-time win for decentralization that can be had simultaneous with the deployment of forward blocks, and that comes from *sharding*. By supporting multiple forward block chains, or *shards*, higher total transaction processing rates can be achieved for the same level of latency-induced centralization risk.

The technique is to explicitly segment outputs into disjoint UTXO sets for each *shard*, and to require that all inputs to a transaction draw from a single shard. Outputs may explicitly name a different shard that they belong to, but at the cost of a delayed transfer and maturation process achieving loose coupling of state between shards. The compatibility block chain tracks the block headers of all shards, and add transactions from each shard to the processing queue as the shard's forward blocks achieve N confirmations. When a transaction in one shard has an output destined for a different shard, the compatibility block chain adds that destination and amount to the coinbase payout queue and claims the output as part of the carry-forward balance. Once paid out and mature, the coinbase payout enters the UTXO set of the destination shard. The following rules define how outputs enter the UTXO set of the forward block shards, becoming spendable:

1) Each shard constitutes a disjoint ledger of value, with a UTXO set that does not intersect with the UTXO set of any other shard. As a consequence, transactions must source all their inputs from the same shard.
2) All legacy outputs and non-prefixed segwit outputs of a transaction belong to the shard that transaction was confirmed in (the same shard it sourced its inputs from), and are immediately spendable by later transactions of the same shard.
3) A prefixed native segwit output in a user transaction differs from a native segwit output in having a prefix byte indicating its destination shard. Prefixed native segwit outputs do **not** enter the UTXO set of any forward block shard.
4) As transactions are added to the transaction processing queue of the compatibility chain, any prefixed native segwit outputs in the transactions of that forward block are added to the coinbase payout queue.
5) When a transaction is processed from the queue and included in a compatibility block, the prefixed segwit output is spent by a miner-generated transaction within the same block, and its value is added to the carry-forward balance used to process items from the coinbase payout queue.
6) A prefixed native segwit output in the coinbase transaction of a compatibility block becomes spendable in the shard specified by its prefix, and that shard only, through the normal process of maturation. Any legacy or non-prefixed native segwit coinbase outputs mature into the default, first shard.

As a prefix we recommend using a single-byte script opcode, for efficiency. There are a total of 28 single-byte script opcodes which may be safely executed at the beginning of a scriptPubKey context: `1NEGATE`, `FALSE`, `TRUE`, the remaining 15 small number pushes (2 to 16), `NOP`, `NOP1`, `NOP4` through `NOP10`, and `DEPTH`. `CODESEPARATOR` could be used but is excluded from this list due to potential infrastructure screwups relating to its other, complicated semantics. Table I enumerates these shard prefixes in order.

Any user transaction output consisting of one of these prefixes followed by normal native segwit output script (a 0 to 16 small integer push followed by a small data push) is called a prefixed native segwit output, and follows the process of coinbase maturation detailed above. A coinbase payout of this form enters the UTXO set of the specified shard upon maturation, and can be spent within that shard exactly as if it were a native segwit script without the prefix.

Without complicating the prefix scheme further, the number of shards is limited by the number of prefix bytes available, which is a historical artifact of bitcoin script. As it happens however, the number of one-byte prefixes which can be safely executed at the beginning of a push-only script does align rather closely with the number of shards which can be safely allowed given other design constraints.

[16] $600 \times 6 = 3600$ blocks $\times \frac{4\text{MWe}}{\text{block}} = 14.4\,\text{GWe}$

[17] In Tradecraft, `NOP2` and `NOP3` are also used for a total of 30 shards.

TABLE I
SHARD PREFIXES

Shard	Prefix	Opcode
1	0x00	FALSE
2	0x4f	1NEGATE
3	0x51	TRUE
4	0x52	2
5	0x53	3
6	0x54	4
7	0x55	5
8	0x56	6
9	0x57	7
10	0x58	8
11	0x59	9
12	0x5a	10
13	0x5b	11
14	0x5c	12
15	0x5d	13
16	0x5e	14
17	0x5f	15
18	0x60	16
19	0x61	NOP
20	0x74	DEPTH
21	0xb0	NOP1
22	0xb3	NOP4[17]
23	0xb4	NOP5
24	0xb5	NOP6
25	0xb6	NOP7
26	0xb7	NOP8
27	0xb8	NOP9
28	0xb9	NOP10

A. The Impact of Sharding on Centralization Risks

The maximum forward block size limits discussed previously of 14.4 GWe / 10 min are far in excess of what could be supported by any presently imaginable network architecture: the centralization pressures would destroy the utility of the network, even with generous assumptions about future bandwidth and latency numbers.

But by allowing some complexity to be pushed onto the user in the form of sharding, we can make full utilization of the available capacity of a time-warped compatibility block chain without growing individual forward block shard chains to be in excess of the upper limits of previous conservative block size increase proposals, such as BIP-103.

Increasing the effective transaction processing rate through sharding proportionally increases the cost of validation for full nodes. For lightweight clients, resources are only increased in proportion to the number of shards they have outputs on or are watching for payments.

However increasing the effective transaction processing rate through sharding does not increase latency-dependent censorship risks, at least while the number of shards remains within reasonable limits. Each shard is its own long-interval forward block chain operating independently of the other shards, with a separately salted proof of work and a loosely-coupled value transfer process across shards. The separate and salted proof-of-work requirement ensures that there is no correlation between when blocks are found on different shards. The coinbase maturation period ensures that the state of one shard has no low-latency dependencies on the state of any other shard. Together these prevent the introduction of centralizing mining incentives from the transaction weight of other shards, preventing a decrease in censorship resistance, but only so long as the time it takes to validate a shard remains significantly less than the expected time between any shard block being found. 28 shards, with each shard having a 15 min inter-block time would have shards arriving every ~ 32 s in expectation on bitcoin, every 30 s on Tradecraft. On the assumption that individual blocks should not take more than a handful of seconds per block to validate in the worst case, this provides ample room even in the face of mining variance, the flex cap, and adversarial blocks to ensure that multiple blocks arriving simultaneously (and therefore impacting each others' validation time) is not the norm.

X. RECOMMENDED FORWARD SHARD WEIGHT LIMITS AND FLEXCAP PARAMETERS

In this section we lay out the suggested initial settings and growth limits for sharding and the per-shard flexible weight limit in bitcoin:

1) The forward block chain is split into 28 shards with separate UTXO sets, for a 28x increase in censorship resistance if fully utilized.
2) All forward shard chains have a fixed target inter-block interval of 15 min, for an additional few percentage points of censorship resistance.
3) At the point of activation, the initial maximum weight per block w_0 of the first/default forward shard chain is 6 MWe, and 25 kWe each for the other 27 remaining shards.
4) Simultaneously with activation, a previously implicit maximum transaction weight is imposed with a limit value very close to the original chain's maximum weight limit: 3.975×10^6 We. The presently lower policy limit still applies, but transactions larger than 3.975 MWe are now invalid, not just non-standard.

The per-shard target inter-block interval of 15 min implies a new block from a randomly selected shard chain arriving every ~ 32 s or so, which provides a total aggregate transaction processing rate of 4.45 MWe every 10 min, only 11.25% more than the original maximum block weight.

This configuration of the flexible block weight limit sets an initial limit on the first shard chain that provides a transaction processing throughput equivalent to the original chain, and therefore starts with exactly the same validation costs as all UTXO exist in that shard initially, and with approximately the same censorship resistance. The other shard chains provide an additional 11.25% of block space, but user adoption of this space is likely to take time as wallets and other infrastructure are adapted, just as was the case with segwit.

Once sharding is fully utilized, a censorship resistance will be improved by a factor equivalent to having reduced the original max block weight by ~ 30x.

5) Using a flexible cap, the miner is allowed to grow or shrink by up to $\pm 25\%$ the maximum weight of

their own forward shard blocks, receiving less (when growing) or more (when shrinking) block reward as a result, a tradeoff they would only make in response to abnormally high or low clearing fee rates.

6) Every 2016 blocks (a **three** week period) each shard chain will add to its baseline maximum weight 3.125% (1/32) of the difference from the previous baseline limit to the trailing average of the actual last 2016 block's weight limits, as were adjusted by the flex cap, constrained to be no smaller than 25 kWe and no larger than 768 MWe. Thus an adjustment of no more than 1.00781250x in either direction is possible per adjustment period, resulting in a maximum adjustment of around $\pm 14.5\,\%/\mathrm{yr}$.

Should every shard chain adjust in response to real usage to its maximum allowed size of 768 MWe, the total transaction processing throughput on the forward block shard chains taken together would be $28 \times 768\,\mathrm{MWe} = 21.504\,\mathrm{GWe}$ every 15 min, or expressed as a more conventional rate 14.336 GWe every 10 min which very nearly equals the maximum allowed throughput on the compatibility chain when fully exploiting the time-warp feature. However it would take consistent maximum upwards adjustments every period for half a century to reach this limit, and given the quadratic costs to raising the flexcap, probably much longer.

A. How Reasonable Is This Upper Bound, Really?

It is not inconceivable that ongoing network and validator improvements will permit a growth in the decades and centuries to come of the order of magnitude required, 7.5 to 12 doublings of compute resources, storage, and bandwidth to safely reach the maximum throughput of 14.336 GWe every 10 min, assuming today's 4 MWe limit as the present-day "safe" baseline. It is worth noting that even in this extreme the corresponding per-shard weight limits would be less than the maximum limits allowed by conservative block size increase proposals such as BIP-103 over the same timeframes.

768 MWe per 15 min shard-block is "only" 192x the post-segwit weight limit of 4 MWe per 10 min block—about 7.5 doublings. Hitting this limit would require new 4MWe compatibility blocks generated every 167 ms, fed from a queue of transactions generated by 28 loosely coupled shards, for a total of 21.504 GWe every 900 s / 15 min. Staying up to date with such a block chain would require 240 Mbps downlink, and participating as a relay node would require an approximate 4x multiple thereof, or around 1 Gbps.[18] Costs to validators would be 3584x present-day costs, or just under 12 doublings.

While a lot, and certainly outside of current capabilities, this is not so crazily outside the realm of possibility as it might seem when we consider what generations of technologies might exist into the foreseeable future:

- Mature graphene transistors, among many possible technological pathways, could provide consumer electronics with many orders of magnitude more performance for the same power and heat dissipation, allowing continued increase in the computational capacity available within consumer form factors (e.g. laptops & desktop workstations), and make possible large storage density non-volatile memory to remove I/O bottlenecks.
- Better electro-optical interconnects would allow existing deployed fiber optic cables to carry orders of magnitude more information, allowing for higher last-mile bandwidth and transfer caps from consumer internet services, allowing both wired connections and WiFi be linked with 1 Gbps or faster WAN connections.
- Mobile data is unlikely to advance as far (due to spectrum scarcity), but the broadcast nature of block chain data means that the partially-trusted solution of satellite or terrestrial radio broadcast could be used for block distribution to fully validating block-only mobile clients. 240 Mbps for absolutely full blocks is only about 12 HDTV channels, meaning there is sufficient spectrum available today for multiple operators to redundantly broadcast block data of that size. Mobile clients would only require a lower bandwidth bidirectional connection (3G or better) to broadcast their transactions.

Looking at the flip side of demand, $28 \times 768\,\mathrm{MWe} = 21.504\,\mathrm{GWe}/15\,\mathrm{min}$ is 2.064 TWe per day in transaction processing capability, or about 1 to $2\mathrm{TB}/\mathrm{d}$ of block chain data. Although a lot of data by any reasonable measurement, it's still only about 1 transaction per day for every human being presently alive, to say nothing of future populations in the cis-lunar light sphere. For payments this would seem to be more than enough as only rebalancing transactions of payment channels are needed at a frequency of a month or so. But:

- The napkin-math calculation of $\sim 1\mathrm{tx}/\mathrm{day}/\mathrm{pp}$ is an average, whereas the need to rebalance less frequently is a statement of the median. We should expect the need for block chain access to follow a power law distribution, which very well could push the necessary average much higher than the median value, as the median is more reflective of the long tail of casual use than the peak of businesses and services that make much higher use of the block chain as a settlement platform.
- Focusing on payments is myopic as what bitcoin provides is the first, and so far only platform for fully automated trustless dispute resolution. Payments are only the simplest form of what we might call *bitcoin-compatible smart contracts* that use bitcoin or other crypto tokens as collateral and UTXOs as semaphore primitives in the coordination of the actions of multiple real-world parties with respect to some pre-arranged contract. While it is possible that multiple contracts can be handled with a single pot of funds managed by a multi-party off-chain protocol—as is the case with lightning—there is an efficiency tradeoff to be made and access to the chain is eventually required, always.

In this light, it is entirely possible that $1\mathrm{tx}/\mathrm{day}/\mathrm{pp}$ is below

[18] 1 download, and 3 uploads per transaction.

the needs of what true global adoption would require for a block chain that handles all forms of automated commercial dispute resolution, not just consumer payments. However it is the limit of what can be achieved as a fully forwards compatible soft-fork in which all nodes see all transactions.

We contend then that these maximum limits are within the realm of reasonable demand in the circumstance of total world adoption, while also within the limits of what foreseeable technology might enable within the decades it would take, even optimistically, to get there. In the mean time an appropriately configured flexible cap ensures that a lower limit is maintained until such time in the future as there is real, paid-for, non-transient demand for that much block space, ensuring in the process that block size increases cannot be used as a way of avoiding transaction fees. Furthermore, a reasonable flex cap proposal, as we have laid out here, would make sure that increases happen slowly enough that collective user action could impose a lower limit if it were seen as being being prematurely raised.

XI. Extension Outputs: Generalizing the Coinbase Payout Queue for Future Protocol Additions

So far we've considered a unified coinbase payout queue for the purpose of coordinating the distribution of block rewards among the old-PoW and new-PoW miners, and for state synchronization between shards. However the mechanism serves as a general solution to the problem of transferring value between various ledger extensions, almost without modification.

The coinbase payout queue is useful anytime discrete accounting systems are used for maintaining a ledger of value within the same block chain. As examples:

- Splitting the block chain into multiple shards, with transfers between shards requiring coordination via explicit transfers, as already seen;
- Obscuring transaction value via confidential transactions (with or without mimblewimble kernel support);
- Obscuring the transaction graph via support of ring signature or zero-knowledge spends; or
- Transferring value between multiple sidechains via a two-way peg mechanism.

Coinbase payout queues are also useful for any circumstance where the value or other detail of an output depends on the circumstances of how the enclosing transaction is mined, and therefore a maturation process is required to prevent the fungibility risk that comes with allowing transactions that can be invalidated with a reorg. Examples from this problem domain include:

- Block reward for forward and compatibility block miners, as already seen;
- A rebatable fee market where excess fee beyond the clearing fee rate is returned to the transaction author; or
- Transaction expiry, or other mechanisms by which a transaction may become permanently invalid for some reason other than a reorg and double-spend.

Neither of these are meant as an exhaustive list! There are so many applications of coinbase payout queues as a maturation process that we cannot include them all, and the above should be treated as merely a list of interesting and/or relevant contemporary proposals, some of which will be elaborated on in the remainder of this talk.

We briefly lay out the mechanism of generalization, first for segregated ledgers:

1) Permit the locking up of funds by sending coins to "anyone can spend" script identifying the destination ledger and endpoint. The value is added to a running total identifying the total coinage tracked by the ledger, and the funds are immediately available at the endpoint, within the segregated environment.
2) The "anyone can spend" output paying into the ledger is claimed by the miner who creates the block containing the transaction, with the coinage added to the carry-forward balance used for coinbase payouts.
3) At a later point in time, any owner of funds on the segregated ledger can authorize a withdrawal, and in doing so specify the destination/recipient. The amount is subtracted from the running balance of funds on the segregated ledger, and an output of the specified size and intended recipient is added to the coinbase payout queue.

For transaction outputs subject to maturation, the process is even simpler:

1) The funds are sent to an "anyone can spend" script identifying the type of output and intended recipient/destination script.
2) The miner who includes the transaction in their block spends the output, adding the funds to the carry-forward balance, and adds an equal valued output[19] to the end of the coinbase payout queue.

The output arrives in the hands of its intended recipient, in the intended ledger, via the usual process of coinbase maturation.

We will make this abstract process more concrete with a number of examples drawn from protocol extensions that could be deployed in the very near future.

A. Rebatable Fees and a Consensus-Visible Fee Market

In this proposal any user transaction may contain one or more explicit rebatable fee outputs. A rebatable fee output has a non-zero explicit value and a `scriptPubKey` containing a single data-push. The data push consists of a weight value serialized as a variable-length 32-bit integer, followed by the fee rebate script verbatim.

The last transaction of a compatibility block is already granted special semantics: it is the location of the carry-forward balance which holds aggregated funds for future coinbase payouts. We now grant it further special semantics:

[19]The output has the same value, but with a `scriptPubKey` stripped of its extra-protocol components.

1) Rebatable fee outputs do **not** enter the UTXO set of the forward block (or shard) chain. They are not spendable by other user transactions.
2) The last transaction of a compatibility block **must** spend every rebatable fee output in that block, with the amount added to the carry-forward balance.
3) The committed total weight of a forward block must be greater than or equal to both (1) the actual weight of the block under known consensus rules; and (2) the sum of the serialized weights of all rebatable fee outputs in that block.
4) The *clearing fee rate* of a forward block is defined to be equal to the sum of all fees (implicit or rebatable) divided by the declared weight of the block.
5) For each rebatable fee output an excess value is calculated, equal to the value of the output minus its serialized weight times the clearing rate. A consensus-critical dust heuristic is performed to see if the excess fee would be considered spendable in the same block without lowering the clearing fee rate, and depending on the result:
    a) If the excess value is insufficient to be profitably spent the excess is implicitly added to the transaction fees of the block, and split between the forward block and compatibility block miners as per the current value of *P*.
    b) Otherwise the forward block coinbase is required to pay out the excess to the serialized `scriptPubKey` encoded in the rebatable fee output.

This achieves the soft-fork compatible rebatable fee & incentive-safe fee market described to the bitcoin-dev mailing list [1] and based off an earlier hard-fork fee market proposal [2].

It also shares a lot of the necessary ground work infrastructure for forward blocks, even if it may seem unrelated, making a rebatable fee market a very small addition on top of forward blocks.

B. Confidential Transactions and Mimblewimble

Confidential transactions is a scheme where the explicit `nValue` amount of an output is replaced by a Pedersen commitment, thereby obscuring the value to anyone who does not possess the blinding factor. This allows for selective disclosure of transaction amounts while maintaining bitcoin's non-inflation guarantees.

Mimblewimble is an application of confidential transactions in which the `scriptPubKey` is not used and therefore knowledge of the blinding factor of an output serves as authorization of a spend. Removing explicit authorization scripts prevents selective disclosure, but gained instead are the efficiency and privacy improvements of transaction cut-through and aggregation.

Bringing confidential transactions and mimblewimble to bitcoin requires (1) adding the necessary fields—Pedersen commitments, ECDH half-protocol nonces, and rangeproofs—to the transaction output data structure; (2) a mechanism for locking up a pool of funds backing the total value of all confidential outputs; and (3) the validation rules to enforce the above.

The transaction output data structure is most easily extended by adding a hash of the extra fields to the end of the `scriptPubKey`. For compatibility with future soft-forks, a general mechanism developed such that a native segwit output is allowed up to eight data additional data pushes (although typically only 1 to 3 are used), together called the *suffix*, with the last being a required-minimal serialized integer value specifying the extended output version.

The version corresponding with a confidential transaction output consists specifies an intervening data value consisting of a single 32 B SHA256 hash of the Pedersen commitment and ECDH nonce. The rangeproof is placed within the output's witness data structure.[20] The `nValue` of the output is 0 btc.

To move coins into a confidential output, the user makes an "anyone can spend" output containing the transferred value and, in an extended output field, a code indicating this is a transfer into the confidential transactions / mimblewimble ledger and a Pedersen commitment of its value (with the necessary signature of the nonce to prove it in the witness structure). The output is spent by the compatibility miner and added to the carry-forward balance, while within the transaction accounting a confidential "input" of the same amount is added. The transaction author, being the only one who knows the nonce of the implicit input, creates confidential outputs claiming it.

Within the confidential ledger, extended outputs can be spent as inputs and new outputs created freely so long as the Pedersen commitments sum correctly in each transaction.

Three mechanisms can be provided by which confidential coins can be converted back into explicit coins:

1) An explicit fee can be provided in the form of a "provably unspendable" output with an extended field specifying how much coinage should be added to the output side of the Pedersen commitment sum. The value is treated the same as an implicit fee would be.
2) A confidential to explicit conversion can be performed by including a "provably unspendable" output with extended data fields specifying the amount and destination. Upon achieving locked-in status, an explicit output of that amount and to the specified destination is added to the coinbase payout queue.
3) A rebatable confidential fee combines both of the above categories by providing an explicit amount that is counted against the confidential outputs side of the Pedersen commitment sum, and a *return script* to which any excess funds should be sent. It is treated

[20] Outputs need to be given witnesses in most of these schemes. It would be a simple matter to add an *output witness structure* whose root hash was also committed somewhere in the block, e.g. the "witness nonce" of segwit.

as a rebatable fee described in Section XI-A.[21]

Moving from confidential to explicit value requires use of the coinbase payout queue because the carry-forward balance that is where the locked up coinage backing the confidential value resides. If the user cannot wait for coinbase maturation, they can find another user willing to trustlessly front the money by signing an explicit input that covers the amount, minus their service fee, and a confidential to explicit payout to themselves.

C. Unlinkable Anonymous Spend Ledgers

Zcash demonstrates how a *zk-SNARK* can be used to prove a spend is from an ever growing set of historical anonymous outputs and that the output has never been spent before, but without revealing which output it was. *Bulletproofs* enable similar constructs with only elliptic curve discrete log assumptions, albeit with higher validation cost. *Monero* achieves a weaker property using *ring signatures* to prove that an input was one of *N* explicitly enumerated previous inputs, thereby gaining efficiency at the cost of a reduced anonymity set.

Any of the above approaches could be deployed in a similar manner to confidential transactions discussed in Section XI-B:

1) An anonymous output is a zero-`nValue` "provably unspendable" output containing some sort of commitment to its actual value.
2) Anonymous outputs are tracked with their own ledger:
   a) Creating an anonymous output requires a *transfer in* which sends explicit value to an "anyone can spend" output claimed in a compatibility block and added to the carry-forward balance.
   b) A "transfer out" requires some input from the same ledger and specification of the destination, which is added to the coinbase payout queue when the transfer out is locked in.
   c) Some mechanism exists for paying explicit or rebatable fees from the ledger, reducing coins available to outputs of the same ledger.
3) Spending an anonymous output requires keeping some amount of information available forever, to prevent double-spends. Data storage on validators can be avoided by using a Merkle tree updated on each spend.
4) The actual spend is not an input, as that would require specifying an input which defeats the purpose. Instead, it is a zero-`nValue` "output" whose witness provides the spend authorization, and the committed value pulled from the output witness is added on the input side of the anonymous spend ledger.

Which anonymous spend mechanism to use and the format of its spend authorization witness we leave up to debate; our concern here is with the block chain ledger accounting and ledger transfer mechanisms.

[21] It would be possible to support returning funds as a "confidential" Pedersen commitment, by providing the EC point corresponding to the blinding factor, but (1) such an output would not be confidential in any meaningful sense as its value would be determined by consensus; and (2) this wouldn't save any block chain space over the alternative of having the user aggregate return fee excess into confidential outputs themselves.

D. Sidechains and the two-way peg

For this topic we avoid the issue of which validator-enforced sidechain architecture should be used—whether the SPV peg, Drivechains, or something else. Regardless of the transfer authorization scheme the accounting mechanism is the same:

1) A *sidechain transfer* is an "anyone can spend" output which is claimed by the compatibility block miner and added to the carry-forward balance, and which commits to the sidechain identifier and destination script to receive the funds on the other chain.
2) A *return-peg transfer* is an "provably unspendable" output which commits to the amount and destination of the funds, and whose witness provides the necessary sidechain information to validate the return peg. Once locked-in, the return peg is added to the coinbase payout queue.

By now it should be clear that all of these extended-ledger mechanisms share the same common approach, using the carry-forward balance and coinbase payout queue to manage transfers in and out, and extra data pushes in the `scriptPubKey` to store extended transaction output fields.

XII. Conclusion

We demonstrated that the idea of *forward blocks* provides a unifying mechanism that:

- Provides on-chain settlement scaling of up to 3584x current limits as a soft-fork;
- Provides for an (optional) proof-of-work upgrade as a soft fork;
- Limits growth of validation costs with a *flexible weight limit*;
- Decreases centralization risks through the adoption of *sharding*; and
- Provides a framework for ledger accounting in future protocol extensions including but not limited to:
  - A rebatable fee market with consensus-determined transaction clearing fee rates;
  - Confidential transactions for obscuring transaction amounts;
  - Mimblewimble, ring signatures, or anonymous spends for obscuring the spend graph; and
  - Sidechain value-transfer mechanisms.

While there are many moving parts to this proposal, it is not beyond the level of complexity of prior extensions adopted by bitcoin (e.g. segregated witness), and achieves a variety of benefits comparable in magnitude.

References

[1] Mark Friedenbach. Rebatable fees & incentive-safe fee markets. Bitcoin Protocol Discussion `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2017-September/015093.html`. 29 September 2017.

[2] Ron Lavi, Or Sattath, and Aviv Zohar. Redesigning bitcoin's fee market. *CoRR*, abs/1709.08881, 2017.