Forward Blocks

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

Mark Friedenbach

October 6, 2018

No organizational affiliation

Introduction

Goals

Forward blocks arose out of considering how a proof-of-work change could be accomplished as a soft-fork, combined with mechanisms for soft-fork deployment of privacy-enhancing alternative ledgers. It was later discovered that it **also** provides scaling benefits including:

- Improved censorship resistance through sharding.
- Direct on-chain scaling up to 3584x (for bitcoin specifically).

It also provides a few miscellaneous other benefits such as a linearized block subsidy and the underlying ledger support for future chain enhancements such as confidential transactions and sidechains.

Note: This talk is a summary of the paper, which has too many moving parts to be fully described in a 30 min talk.

Definitions

A *soft fork* is a tightening of the consensus rules such that some blocks which were previously valid are now invalid, but no previously invalid blocks become valid. Simply: old nodes still see the chain advance.

A *forwards compatible soft fork* is a soft-fork for which un-upgraded nodes still receive and process **all** transactions.

We are specifically interested in forwards compatibility because it fits our prior model for the safety of soft forks:

- Non-mandatory upgrades paths.
- No flag days beyond which chain access is limited.
- An ability for un-upgraded infrastructure to continue working during and after the transition period.

A note on centralization risks

Centralization risks broadly fall into two categories:

1. Increasing the *cost of validation*, or 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.

Cost of validation is proportional to the number of transactions in full nodes and the number of blocks for SPV nodes.

2. Reducing *censorship resistance*, which is that property which results from any user being able 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 large or small.

Censorship resistance has a non-linear relationship to the ratio of block propagation time to the average block interval.

Dual Proof-of-Work

A block subject to 2 proofs-of-work?

No.

Forward blocks involves separate chains with separate proof-of-work functions, rather than a single block chain with each block subject to multiple work requirements.

But just for the moment, let's use a single block subject to two proofs-of-work as an example.

Achieving a transition in difficulty

With a block subject to two work functions, the difficulty needs to non-disruptively transition from the old proof-of-work to the new.

The easiest way to do this is have a sliding block reward that transitions block reward from going primarily to the solver of the old proof-of-work challenge to the new, over of period of time long enough as to prevent mining disruption.

This is expressed by a function $P(t) \in [0, 1]$ which represents the proportion of the block reward that goes to the new proof-of-work miners vs. the old:

$$P(t) = \min(\frac{t}{3.5\,\mathrm{yr}}, 1)$$

At the end of the transition period, in this case 3.5 yr, the new proof-of-work will represent nearly all of the security / network hash power, and the old proof-of-work will be at minimum difficulty.

New proof-of-work, or merge mining?

We are given an opportunity with the deployment of forward blocks to change proof-of-work functions. But we not required to do so—the "new proof of work" could be double-SHA256 merged mining.

I do not wish that "proof-of-work upgrade" be interpreted as an adversarial move against the current set of bitcoin/double-SHA256 miners. Rather it is a direct consequence of the design that the forward block chain **requires** a new proof-of-work function. A compatible form of salted merge mining is a sufficiently different function to work for this purpose.

Or it could be something entirely new rendering most existing hardware useless once the 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.

Raising the Block-Weight Limit

Forced hard-forks

The commonly discussed “safe” mechanism[1] for raising the block weight limit is the *forced hard fork*: move transactions into a committed extension block with higher aggregate limits—and/or any other consensus changes—and then force the old blocks to be empty.

Confirming the validation of transactions is only possible by upgrading to a client which understands the extension block. Un-upgraded nodes are protected from seeing divergent spend histories[2] by the fact that the old blocks are kept empty. To restore service, they are required to upgrade.

---

[1] Forced hard forks are described as safe, but safety is relative. I object to describing a purposeful denial of service attack against un-upgraded nodes as “safe.”

[2] Again, I’d contend that empty blocks should be considered divergent with material consequences. A lightning node needs to see its channel closure, for example.

Forward blocks

Forced hard forks break forward compatibility, but note:

- If we ensure that the extension block only violates aggregate, block-level consensus rules, then all transactions in the new blocks would hypothetically be valid on the old chain.[3]
- Instead of forced empty blocks we can have the old *compatibility block chain* repeat the same transactions in the same order.
- By switching from extension blocks to a separate chain with loosely coupled state, the *time warp bug* can be exploited to lower compatibility block intervals to keep pace with higher limits.

This new chain which determines transaction ordering we call the *forward block chain*.[4]

[3] Except for coinbases and their children. We have a way of dealing with this.

[4] A reference to *forward observers* of mobile infantry units that scout the path ahead.

Two chains, two separate ways to scale

The forward block chain achieves on-chain scaling by increasing its per-block aggregate weight limit while maintaining a fixed, long duration target inter-block interval. This achieves the smallest possible impact on centralization risks for both full validators and SPV nodes.

The compatibility block chain can only scale by exploiting the time warp bug to lower its expected inter-block interval. Lowering the block interval is strongly centralizing on the compatibility chain, but this has **no effect** on censorship resistance because transaction ordering is already determined solely by the forward block chain.[5]

[5] During the transition between proof-of-works, we wouldn't want the forward chain to scale so much that the compatibility chain becomes centralized before the forward chain has enough mining security. The flexible cap will prevent this.

Some annoying loose ends...

An (incomplete!) list of issues left open by this explanation:

- Many compatibility blocks could be needed to process a single forward block, so having the same miner produce these blocks unacceptably introduces the notion of work progress.
- A coinbase transaction of the forward block is not seen by un-upgraded nodes, so it cannot enter the UTXO set. Both miners need to be paid by the compatibility block's coinbase.
- Since different miners[6] generate forward and compatibility blocks, there needs to be some form of state synchronization in order for:
  - Compatibility block miners to learn forward block transactions; and
  - Forward block miners to learn the coinbase transactions of the compatibility chain, so as to process them into the UTXO set.

[6] The same set of miners if merged mining is used, but miner of a specific forward block would not mine the corresponding compatibility blocks except by random chance.

Loosely Coupled Chain State

Cross-chain header commitments

Both the forward chain and the compatibility chain commit to the headers of the other. When a header reaches 100 confirmations, it becomes *locked-in*:

- Any *locked-in* header must reference a valid block.
- When the compatibility chain locks in a forward block header, the transactions of that block are added to the *transaction processing queue* and its coinbase outputs to the *coinbase payout queue*.
- When the forward chain locks in a compatibility block header, the coinbase of that block enters the UTXO set.[7]

[7] Subject to the usual 100-block maturity requirement.

One coinbase shared by two chains

The coinbase of the forward block has outputs that cannot be spent, so they are repeated in the compatibility block's coinbase after lock-in.

The forward block miner claims a portion P of the coinbase reward of their block. The remaining $1 - P$ value goes into a fund to pay compatibility block miners. The compatibility block miner gets $1/k$ the current size of the fund.

Using the compatibility coinbase to synchronize payments based on the state of multiple chains is a common pattern we will reuse.

A Flexible Weight Limit

Initial parameters of the forward block chain

Target block interval: 15 min

Increasing the block interval from 10 min to 15 min achieves a one-shot boost in censorship resistance, at the cost of lengthening confirmation times. For a chain tailored to handling settlement transactions, that's a good trade-off.

Initial max block weight: 6 MWe

With 15 min blocks, this represents the same transaction processing rate as the original chain's limit of 4 MWe blocks every 10 min.

Growing the forward block chain with a flexcap

The forward block miner is allowed to *bias* their work target by up to $\pm 25\%$, making their blocks easier or harder to solve. In return, the aggregate weight limit of their block is adjusted:

$$x = \frac{T - T_0}{T_0}$$
$$x \in [-0.25, 0.25]$$
$$w(x) = w_0(2x - 4x^2)$$

Every $L = 2016$ forward blocks, two things happen:

- The base limit w_0 adjusts to the (possibly gain limited) average of the past L declared weights.
- The new-PoW difficulty adjusts up or down as necessary to maintain the 15 min inter-block interval.

Offset reward as a function of difficulty bias

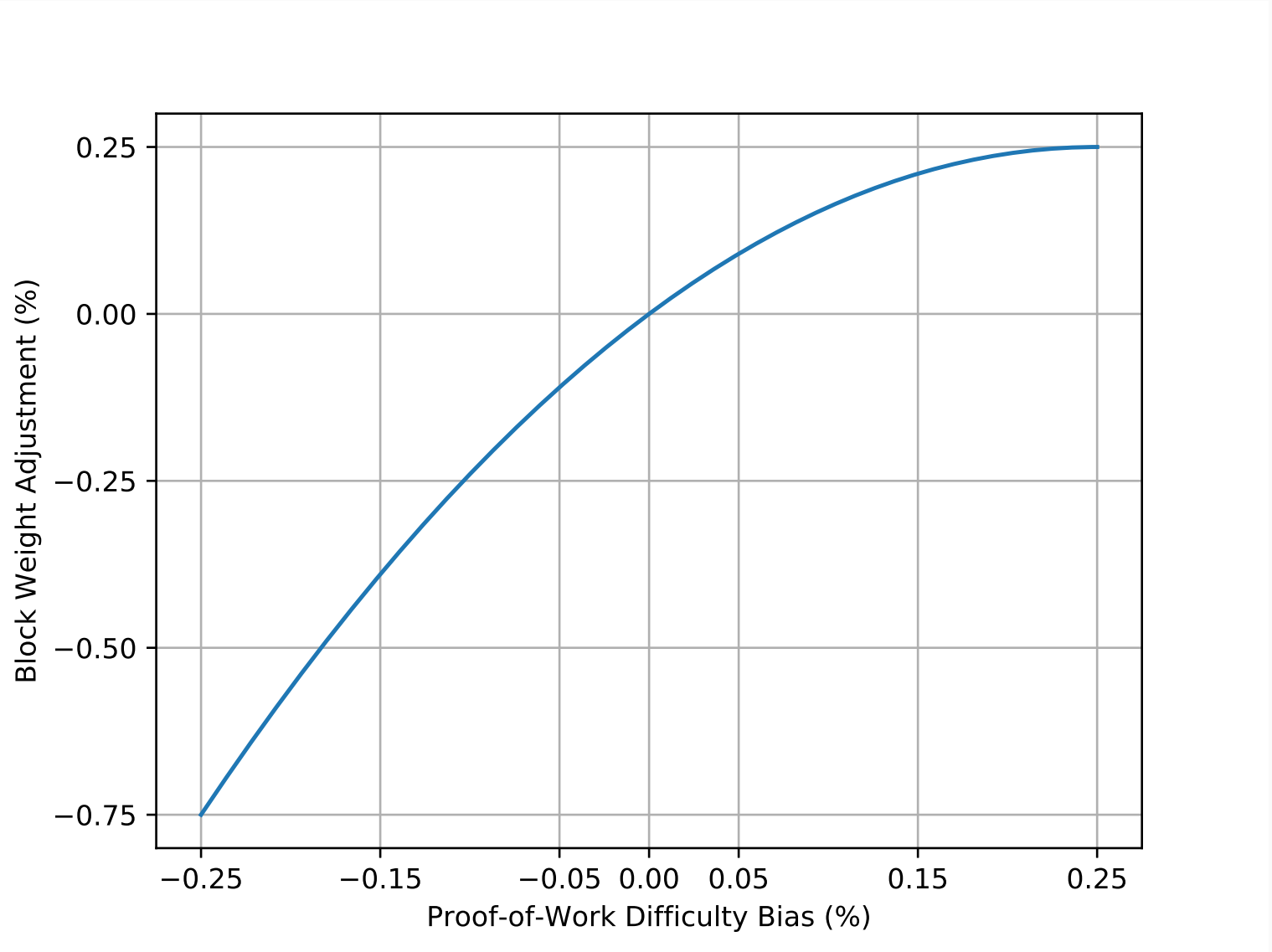

Figure 1: Relationship between max block weight and difficulty bias.

Time-warping the compatibility chain

For compatibility blocks that are **not** the last block in a difficulty adjustment window:

1. If after constructing the block the transaction queue is empty, no additional rules apply.
2. 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.
3. 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.

Time-warping the compatibility chain (2)

For compatibility blocks that **are** the last block in a difficulty adjustment window:

The *warp factor* is a ratio Q equal to the forward block utilization over the original chain's settlement capacity:

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

The Satoshi difficulty adjustment formula is applied in reverse to calculate the block timestamp that would cause the adjustment needed to handle Q blocks in expectation every 600 s.

A Smooth Subsidy Schedule

Eliminate the “halvening” with a continuous subsidy curve

We have the freedom to set a different subsidy schedule in 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.

Most notably, we can use this opportunity to smooth out the step function used to calculate subsidy, making subsidy a continuous function.

Eliminate the “halvening” with a continuous subsidy curve

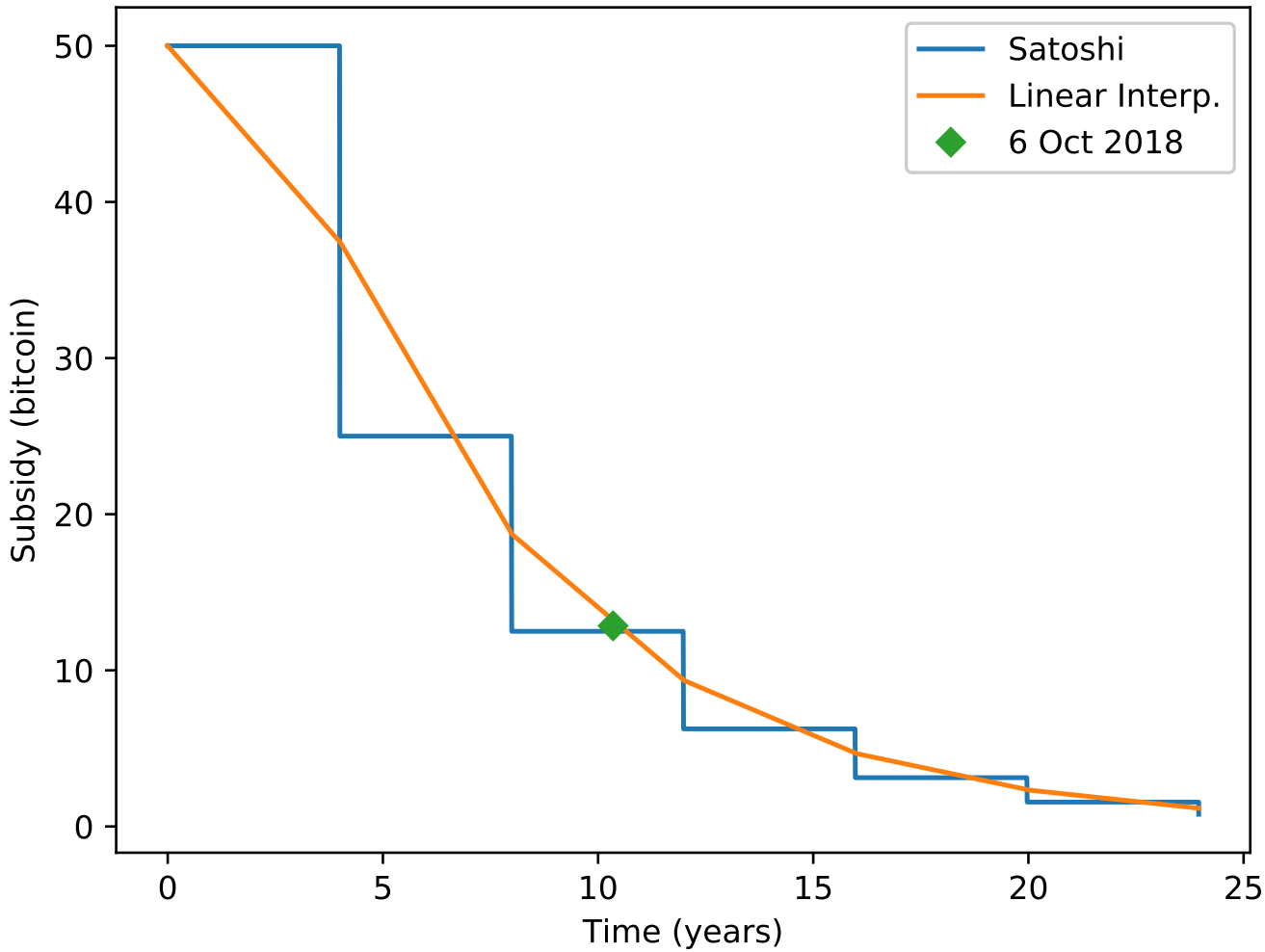

Figure 2: Alternative Subsidy Curve

Multiple Forward Block Chains

Sharding: multiple forward block chains

A very significant gain in censorship resistance can be had by sharding[8]the forward block chain into multiple chains with disjoint UTXO sets.

By splitting the chain into M shards, requiring transactions source their funds from a single shard only, and using a separately-salted work function for each, a shard block will appear in expectation every $15\,\text{min}/M$, but blocks from a given shard chain will be separated by a full 15 min. This achieves an M-fold increase in censorship resistance for a given aggregate weight across all shards, if activity is evenly distributed amongst the shards.

[8]Sharding is a term of art borrowed from the database field. The “sharding” described here is largely **not** the sharding talked about in crypto currency projects, but it is nevertheless the correct term to use.

Transferring value between shards

Table 1: Shard Identifier Prefixes

Shard	Prefix	Opcode
1	0x00	FALSE
2	0x4f	1NEGATE
3	0x51	TRUE
4	0x52	2
	...	
18	0x60	16
19	0x61	NOP
20	0x74	DEPTH
21	0xb0	NOP1
22	0xb3	NOP4
	...	
28	0xb9	NOP10

A native segwit output can be prefixed with a 1 B value indicating the destination shard identifier. This makes it unspendable on the forward block chain. Note that `NOP2` and `NOP3` are not usable as such on bitcoin.

These *prefixed outputs* are added to the coinbase payout queue verbatim, and the value is claimed by the compatibility block miner and added to the carry-forward balance.

The high end of scaling limits

The compatibility chain's block timestamp must tick forward once every 6 blocks, which gives a maximum of 3600 compatibility blocks per 600 s legacy block interval.

This is closely matched by a 768 MWe maximum weight on each of 28 separate shards with a 900 s block interval.

Together this would allow for 14.336 GWe of transactions to be processed every 10 min, which is about 1tx/pp/day for everyone currently alive.

To prevent denial of service from premature growth, it is recommended that a gain limiter of $\pm 0.78125\%$ per adjustment period be applied, which results in a maximum growth of $\pm 14.5\,\%/\mathrm{yr}$.

Extension Outputs

Generalized ledger transfer mechanism

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.

Generalized coinbase maturity 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.

Questions?