Blockclique Multi-Staking Protection

March 23, 2022

1 Introduction

In Blockclique [1] as in other Proof-of-Stake blockchains, the computational cost of creating a block is negligible, which is different from Proof-of-Work blockchains. When a node is selected to produce a block in a slot, in practice it can thus create multiple blocks with different hashes for the same slot, while only one can be taken into account by other nodes in that slot. Double- or multi-staking is thus considered as a kind of flooding attack. Tezos [2], a Proof-of-Stake blockchain with a Nakamoto-like consensus seems to have no particular protection against this attack other than an incentive mechanism with a double-staking punishment.

In a blockchain, if the attacker sends many different and incompatible blocks of the same slot to each node, the next block producer will choose one of them as parent, then produce and broadcast its block. This block producer thus helps in choosing one of the alternative attacker blocks. However, if no particular protection is taken, the attacker can still bloat the network by creating thousands or millions of valid blocks in all of its slots. This attack could make it harder for honest nodes to get to a consensus on the executed blocks, even more if the attacker has a large proportion of the block production power, say 30%.

Intuitively, there are three possible strategies to cope with this flooding attack:

- `FIGHT`: Wait for the next block producers to choose one of the alternative chains and ask for the missing blocks. Repeat if the attacker reiterates. Ban nodes that send too many unsolicited blocks for the same slot.
- `IGNORE`: Once at least two attacker blocks have been received for the same slot, consider all of them invalid and consider that the attacker missed. Ban nodes that send too many unsolicited blocks for the same slot.
- `EQUIVALENT`: Consider all alternative blocks of a same slot as the same equivalent block. Ban nodes that send too many unsolicited blocks for the same slot.

While in blockchains, the `FIGHT` strategy might be sufficient to avoid consequences on liveness, it may not be for the parallel structure of Blockclique due to the numerous and parallel opportunities for the attacker to create incompatible blocks. The `IGNORE` strategy seems to be a bad idea: an attacker could trigger a finality fork by broadcasting a second block for a slot around the moment when the first one becomes final. Here we study the `EQUIVALENT` strategy for Blockclique.

2 The `EQUIVALENT` strategy in Blockclique

The goal of this strategy is to consider all blocks of the same slot as compatible and equivalent, such that each node never needs to download more than one full valid block per slot (see Fig. 1).

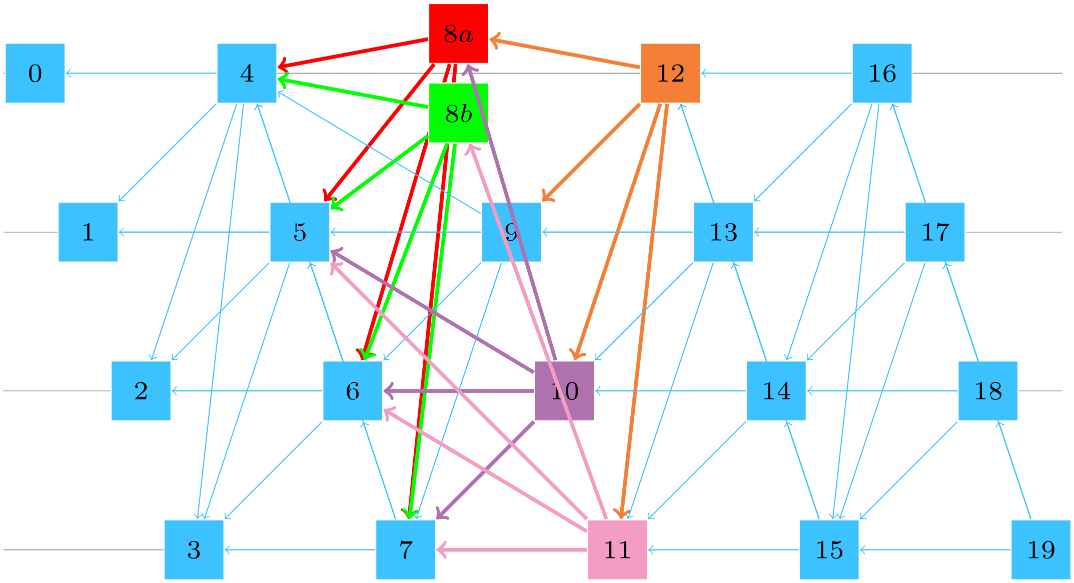

Figure 1: Node selected for slot 8 published 2 distinct blocks for this slot, 8a and 8b. Block 10 points to 8a and block 11 points to 8b. Node selected for slot 12 received all previous blocks, in the order (8a, 8b, 9, 10, 11). There are no incompatibilities so only one clique. Transactions of 8a and 8b won't be processed when they become final, they can be included again in block 12 and processed when 12 becomes final.

2.1 Graph Rules

The thread and grandpa rules are replaced by the following sequential validity rule and parallel incompatibility rule.

2.1.1 Sequential Validity Rule

B is valid if

$$\begin{cases} \text{All parents of } B \text{ are in slots before the slot of } B \\ \text{Ancestors of } B \text{ in a given thread are ancestors of the parent of } B \text{ in that thread} \\ \text{All parents of } B \text{ are mutually compatible} \end{cases} \tag{1}$$

2.1.2 Parallel Incompatibility Rule

Let B_1 and B_2 be two parallel blocks (each one is not the ancestor of the other one). B_1 and B_2 are incompatible if and only if:

$$\begin{cases} |t(B_1) - t(B_2)| \geq t_0 \\ \text{or } B_1 \text{ is incompatible with an ancestor of } B_2 \\ \text{or } B_2 \text{ is incompatible with an ancestor of } B_1 \end{cases} \tag{2}$$

where $t(B)$ is the timestamp of block B.

Notes:

- Inside a thread, the difference with the thread incompatibility rule is that here multistaked blocks in a same slot are compatible (unless they inherit incompatibilities, e.g. if they have a different parent).
- Across threads, this rule is stronger than the GPI rule: GPI-incompatible blocks are incompatible with this rule, but there are other cases where GPI-compatible blocks are incompatible here. In particular, it is not longer possible to publish valid blocks in a given thread that is lagging behind other threads due to block misses.

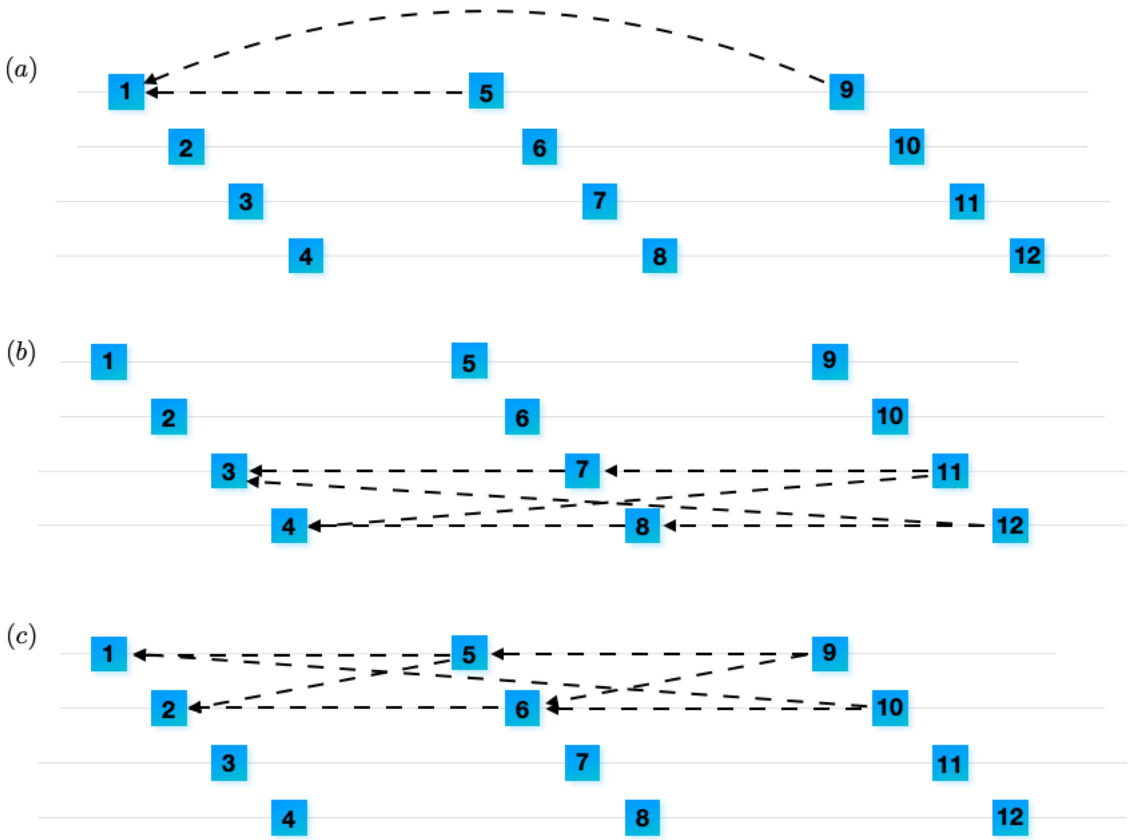

Figure 2: (a) Block 5 and 9 are incompatible. This correspond to the old "thread incompatibility rule". (b) Blocks 7 and 12 are incompatible. As a result, blocks 11 and 12 are incompatible. This corresponds to the old grand-pa incompatibility rule. (c) Examples of blocks that were compatible under the old grand-pa incompatibility rule but are now incompatible: blocks 5 and 10 are incompatible because block 10 does not reference block 5 as a parent in thread 0 even though block 5 was created more than t_0 before block 10.

Proposition. *The parallel incompatibility rule is stronger than the grand parent incompatibility rule.*

2.1.3 Incompatibilities of multistaked blocks

Multistaked blocks can be incompatible in the following cases:

- Two blocks in a same slot S are incompatible if they reference two different blocks in a same previous slot S' with $t(S') \leq t(S) - t_0$.

- Two blocks in a same slot S can be incompatible if one of them has an older parent in a given thread than the other one.

Each incompatible multistaked block leads to a new clique with maximal or close to maximal fitness. To avoid this, we add the following rule which avoid computation of parallel incompatibilities:

- Two blocks of a same slot are considered parents of one another if all their parents are mutually compatible.

Notes:

- With this rule, the processing of a multistaked block in a slot S can change the incompatibilities of other blocks in S, so we have to recompute the incompatibilities of all blocks when we process a new multistaked block (and when multistaked blocks become final ?, when we see the multistaked block is not denounced? see operation finality).

2.2 Particular Rules on Multistaked Blocks

2.2.1 Explicit Denunciations

The producer of block B can add proofs of double-staking (explicit denunciations) in the header of B. Each denunciation must concern a pair of blocks in a same slot S.

Block B is valid only if each explicit denunciation by B of a slot S is such that all blocks of slot S in the ancestry of B have a fitness in their descendants in the ancestry of B that is lower than $\Delta_f^0 + (1+E) * T/2$, or if there is no block in slot S in the ancestry of B, that $t(B) \leq t(S) + t_0$.

The double-staked blocks may or may not be compatible. Denunciations are not operations, they are included in block headers. Each block header can include a maximum number of $T/2$ denunciations. A denunciation in a block B concerning slot S is valid only if there are a maximum of $T/2 - 1$ other explicit denunciations for S in the ancestry of B (why not: valid if there is no other denunciation in the ancestry ?).

2.2.2 Implicit Denunciations

When a block B has in its ancestry a pair of blocks in the same slot S and this pair of blocks was not in the ancestry of any of its parents, B is implicitly denunciating slot S.

Each implicit denunciation in B for a slot S is valid only if $t(B) \leq t(S) + t_0$.

2.2.3 Block Finality

The final and stale rules for blocks stay the same.

2.2.4 Operation Finality

Operations in multistaked blocks must wait for possible denunciations before becoming final. The operation finality rules are the two following rules.

Operations of a final block B in slot S are considered **final** if all of the below:

- (**Waiting for denunciations**) for all threads τ: one of the last final blocks in τ has a fitness above Δ_f^0 in its ancestry in the descendants of B.
- (**No denunciations**) S has no denunciations in later final blocks, and there is no final or active block in S other than B.
- (**Sequentiality of final operations**) for all threads τ: one of the last blocks in τ strictly before S has final or stale operations and either this block B_τ is in the last slot in τ strictly before S or slots between B_τ and that slot are final missed slots.

Operations of a final block B in slot S are considered **stale** in the context of this block if all the following conditions are verified:

- (**Denunciations**) There are multiple final blocks in S or S is denounced in a final block.
- (**Sequentiality of final operations**) same as before

Notes:

- A slot in thread τ with no block is a final missed slot if there exists a final block in any thread at or after the next slot of τ.
- After **Waiting for denunciations**, B is necessarily final, so not in the incremental graphs anymore. Do we really need to recompute all incompatibilities of final blocks from B and blocks in the incremental graph because B was not denounced and the parent links should be removed ?
- We can probably do a faster **Waiting for denunciations** rule

2.2.5 Endorsements

Endorsements of multistaked blocks are not taken into account.

2.2.6 Rewards

The reward of multistaked blocks is not credited.

2.2.7 Fitness

The fitness added to a clique by a set of multi-staked blocks is 0.

2.2.8 Punishments

A double-staking denunciation (for the double production of a block or an endorsement) or a set of blocks becoming final in a same slot (implicit denunciation) trigger an implicit slash of exactly one roll. The $T/2$ first denunciations for a slot are rewarded with $2/T$ of the slashed roll value, the rest is burned.

2.2.9 Double-endorsements

Double-endorsing stays same as Tezos, as only one node has to take into account the endorsements of an endorsement slot, there is no risk of parallel divergence. Double-endorsements can be denounced.

2.3 Processes

2.3.1 Best Parents

We add this heuristic to help choose the best parents in the blockclique:

- The best parent in a thread where there are multiple blocks in the same tip slot is the one with the most fitness in its descendants in the blockclique, and if tie is the first processed.

2.3.2 Network

We assume that the multistaked blocks will have a denunciation in the blockclique in time. If in the end there was no denunciation and one multistaked block is included in the blockclique, we must have the full block. If we don't have the full block, it means the attacker is also able to succeed a finality fork attack (TODO proof).

We ban peers who sent:

- an unsolicited full block
- strictly more than 2 unsolicited valid headers of the same slot
- an invalid header or full block

When we receive:

- an unsolicited full block, we ignore it
- an unsolicited block header for a slot where we have no other block: look in storage or ask for the parents (and ancestry) if we don't have them, process the parents (for each parent we need the full block or the header and another header in the same slot), if parents are all processed, ask the full block, process it, process blocks that depended on it.
- an unsolicited block header for a slot where we already have one header: we process this block as a multistaked, we don't ask the full block.
- an unsolicited block header for a slot where we already have two block headers: we put it in storage for some time in case we need it later, we don't ask for the full blocks or the parents.

3 Analysis of Attacks in the `EQUIVALENT` Strategy

How many headers and full blocks can be received or processed in the worst case ?

Attacker actions: produce many blocks in a same slot, send some of them in any order to any node in the network, send any of them with delay.

3.1 Assuming only one attacker slot

Assuming a denunciation will be included.

Only one full block can be downloaded and processed: when we receive one block header, we ask for the full block. Otherwise we drop the unsolicited full block. If we received two block headers before the first full block, we won't need the full block, we drop it if we receive it.

Regarding block headers, each node of the attacker can directly send only 2 different block headers for the same slot to each peer before being banned, the 3rd block header won't be processed by the peer. In a network of N honest nodes with a maximum of P connections, there can be $2*P*N$ attacker block headers around. Each peer will receive a maximum of 2*P unsolicited block headers, but will process only 2 of those.

Each honest node creating a block after the multistaked slot will link one of those multistaked blocks if the attacker sent them in time. For a duration $t_0/2$, honest nodes creating blocks after the multistaked slot S may not have received any of the honest blocks building on S and therefore build on a different one each time, $T/2$ in total, for which we will ask and process the headers.

In total we may process $2 + T/2$ block headers.

3.2 Assuming multiple attacker slot

At each new attacker slot after the first multistaked block in slot S while S is not final, the attacker can create multistaked blocks that points to a new block in slot S, that we will ask and process (at least the headers). There can be a large number of block headers to propagate and process depending on the attacker's power.

To limit the number of headers to ask and process, one optimization could be to ask the parents of the new block B only if B is not multistaked or we have received the header of a non-multistaked descendent of B.

References

[1] Sébastien Forestier, Damir Vodenicarevic, and Adrien Laversanne-Finot. Blockclique: scaling blockchains through transaction sharding in a multithreaded block graph. *arXiv preprint arXiv:1803.09029*, 2018.

[2] LM Goodman. Tezos: A self-amending crypto-ledger position paper. *Aug*, 3:2014, 2014.