Afgjort: A Partially Synchronous Finality Layer for Blockchains

Bernardo Magri[1], Christian Matt[2], Jesper Buus Nielsen[1], and Daniel Tschudi[1,2]

[1]Concordium Blockchain Research Center, Aarhus University, Denmark
{magri, jbn}@cs.au.dk
[2]Concordium, Zurich, Switzerland
{cm, dt}@concordium.com

November 21, 2019

Abstract

Most existing blockchains either rely on a Nakamoto-style of consensus, where the chain can fork and produce rollbacks, or on a committee-based Byzantine fault tolerant (CBFT) consensus, where no rollbacks are possible. While the latter ones offer better consistency, the former can be more efficient, tolerate more corruptions, and offer better availability during bad network conditions. To achieve the best of both worlds, we initiate the formal study of finality layers. Such a finality layer can be combined with a Nakamoto-style blockchain and periodically declare blocks as final, preventing rollbacks beyond final blocks.

As conceptual contributions, we identify the following properties to be crucial for a finality layer: finalized blocks form a chain (*chain-forming*), all parties agree on the finalized blocks (*agreement*), the last finalized block does not fall too far behind the last block in the underlying blockchain (*updated*), and all finalized blocks at some point have been on the chain adopted by at least k honest parties (k-*support*).

As technical contributions, we propose two variants of a finality layer protocol. The first variant satisfies all of the aforementioned requirements (with $k = 1$) when combined with an arbitrary blockchain that satisfies the usual common-prefix, chain-growth, and chain-quality properties. The second one needs an additional, mild assumption on the underlying blockchain, but is more efficient and satisfies $k = n/3$-support. We prove both of them secure in the setting with $t < n/3$ Byzantine parties and a partially synchronous network. We finally show that $t < n/3$ is optimal for partially synchronous finality layers.

Contents

1 Introduction

In classical blockchains such as Bitcoin [25], parties that win a "lottery" are given the right to append blocks to a growing chain. Due to network delays, the chain can fork and become a tree since parties can append new blocks to the chain before even seeing other blocks already appended to the chain by other parties. Such forks can thus be created intentionally due to adversarial behavior. Therefore, parties need a chain-selection rule, e.g., the longest-chain rule, determining which chain in the tree is considered valid and where to append new blocks. Therefore, the chain selected by a given party can change over time, causing rollbacks and invalidating transactions on the previously selected chain. Since very long rollbacks are unlikely, the risk can be mitigated by waiting until "sufficiently many" blocks are below a certain block before considering it "final".

This is problematic for the practical adoption of blockchains, especially for applications such as cryptocurrencies, where the confirmation time for blocks (and transactions) needs to be almost immediate. One problem is that "sufficiently many" depends on unknown parameters such as the network condition. It is therefore unclear how long one really needs to wait. Another problem is that this waiting time will often be longer than what is desirable for applications: Even assuming perfect network conditions and $1/3$ corruption, the adversary can win k lotteries in a row with probability $1/3^k$ and thereby cause a rollback of length k. This means that to limit the rollback probability of a block to 2^{-80}, one needs to wait for at least 50 blocks appended to it. Taking Bitcoin as an example, where a new block is generated roughly every 10 minutes, this results in waiting for more than 8 hours for a block to be final. Considering more sophisticated attacks and unclear network conditions, an even longer waiting time would be necessary. The main reason for this slow finality is that the simplistic rule of looking far enough back in the chain needs to take a worst-case approach: it is extremely unlikely that the adversary wins 49 blocks in a row, but to obtain 2^{-80} security against $1/3$ corruption, you must assume all the time that it can happen.

We propose a partially synchronous finality layer that can be composed with any synchronous or partially synchronous Nakamoto-style blockchain (cf. [13, 27, 11]) that has the properties of common prefix, chain growth, and chain quality (as defined by [13]). Our finality layer allows to dynamically "checkpoint" the blockchain by using Byzantine agreement to identify and then mark common blocks in the honest user's chain as final. The goal of our finality layer is to guarantee that a block becomes final much faster than the 50 blocks waiting time mentioned above; once a block is agreed upon by the honest users it should quickly become final. In this sense, the finality layer should be responsive: if blocks quickly become agreed upon (even before they reach the worst-case common-prefix bound), they also quickly become final. Once a block is final, honest parties will never do a rollback behind this block.

Desirable properties of a finality layer. We identify the following properties to be crucial for a finality layer. First, no forks should be in the final part of the tree, i.e., all finalized blocks should be on a chain. We call this property *chain-forming*. *Agreement* further ensures that all parties agree on all finalized blocks. Moreover, finalized blocks should keep up with the chain growth. More concretely, for $\Delta \in \mathbb{N}$, the Δ-*updated* property guarantees that the chains held by honest parties are at most Δ blocks beyond the last finalized block. Finally, we want that all finalized blocks had been on the chain adopted by at least k honest parties (k-*support*) for k as large as possible. Without this property, many parties are forced to jump to another chain under the finalized block, causing rollbacks for them. A minimal requirement is 1-support because otherwise, finalized blocks are not on the path of any honest party, resulting in bad chain quality.

Pure BFT consensus blockchains. As an alternative to Nakamoto-style blockchains, committee-based Byzantine fault tolerant (CBFT) consensus designs such as the ones employed by Tendermint [22, 6] and Algorand [23, 16] have been proposed. Such blockchains provide immediate finality, i.e., every block that makes it into such a blockchain can be considered final. They have, however, also disadvantages: Compared to Nakamoto, producing new blocks takes more time, especially if the committees are large. Furthermore, responsive CBFT protocols cannot tolerate more than $t < n/3$ corruptions (cf. [28]), while Nakamoto-style blockchains typically tolerate up to $t < n/2$ corruptions. The two-layer approach of using a Nakamoto-style blockchains with a partially synchronous finality layer on top as we propose in this paper aims to provide the best of both worlds.

Advantages of our two-layer approach. Using a two-layer approach with a finality layer on top of a Nakamoto-style blockchain has several advantages over using a CBFT consensus design or using only a Nakamoto-style blockchain:

- A finality layer can be put on top of *any* Nakamoto-style blockchain, yielding a modular design. This allows to optimize the two aspects separately. In particular, our finality layer could be retrofitted on top of existing blockchains to get responsive finality.
- Compared to a Nakamoto-style blockchain, under good conditions, blocks can be declared final much faster than waiting for sufficiently many blocks. This is because the eventual agreement of a Nakamoto style blockchain is a probabilistic property, and therefore one needs to be very conservative on setting its parameters in order to reach meaningful guarantees. As mentioned earlier, for Bitcoin one would have to wait for more than 50 new blocks appended to a block before considering it final with a probability of approximately $1 - 2^{-80}$. It can be observed in Bitcoin, however, that appending 6 new blocks is typically already enough for all honest users to have a common prefix.
- When a Nakamoto-style blockchain is combined with a finality layer, much more aggressive designs can be allowed for the underlying blockchain. The blockchain should just ensure that on average there is a good common-prefix, and that there is always *some* upper bound on the common prefix; this bound does not have to be known. This can, e.g., allow for setting more aggressive block-times, leading to faster production of blocks.
- CBFT protocols tend to be very slow if the number of participants is high. Therefore most CBFT protocols like Algorand and Tendermint work by sampling the committee as relatively small subsets of parties. Algorand proposes a committee size of $1,500$ to get security in at the level normally consider by cryptography (2^{-60}). Tendermint in some versions propose using significantly smaller committees for efficiency, which results in sub-cryptographic security. By only using the CBFT for finality, we can tolerate to run with bigger committees. The underlying blockchain ensures throughput by having a block-time in the seconds, and the finality layer can be designed to work with that: If the finality layer takes 10 minutes to finalize a block and you have a 10 second block-time, simply finalize only every 60 blocks.
- Using a two-layer approach, as we propose in this paper, provides the best of both worlds: Transactions make it into the blockchain very quickly, but may still be subject to a rollback until they are finalized. For small transactions between mutually trusting parties, they can decide to accept the risk and proceed directly. If stakes are high, the parties can wait until the transaction appears on a finalized path where, for a good finality layer, this waiting time should roughly correspond to the time it takes a CBFT protocol to produce the next

block. Therefore, parties in our design can, for each transaction, adaptively decide whether they want to be fast or safe, giving the best possible guarantees in both cases.

- The well-known CAP-Theorem [5, 17] implies that in the presence of a catastrophic network partition, one cannot have both consistency and availability. This means that in case of a partition, say between different continents, one has to choose between consistency and availability. In Nakamoto-style blockchains, the chains will keep growing on each continent, i.e., there is availability. Once the network joins again, there will be long rollbacks on some continent, i.e., there is no consistency during partitions. On the other hand, CBFT blockchains choose consistency over availability by not producing any new blocks during the partition (assuming that no continent contains more than 2/3 of all parties/stake/computing power/...). Our design again provides the best of both worlds: The blockchains on each continent will keep growing, but finalization will pause during the partition. As in the previous point, users can now again choose whether they want to accept transactions on the chain (i.e., choose availability), or wait until finalization continues to work (i.e., choose consistency). This means that our design puts the choice between consistency and availability into the hands of the users of the system. Furthermore, this choice can be made for each transaction individually. In contrast, for existing designs like Bitcoin and Algorand, this choice is fixed by the protocol designer once and for all.

- Lastly, responsive CBFT protocols cannot tolerate more than $t < n/3$ corruptions (cf. [28]). This means that designs which use CBFT for producing blocks can catastrophically break down if it happens *once* in the lifetime of the system that more than $n/3$ of the parties are corrupted. On the other hand, some blockchain designs tolerate up to $t < n/2$ corruptions. Some designs in fact tolerate periodic corruption of $t > n/2$ if only over long enough periods the average corruption is below $n/2$. It therefore makes sense to have a two-layer design where the blockchain ensures the eventual agreement and the finality layer ensures only finality. If at some point the corruption reaches $n/3 < t < n/2$, the finality property might break, but eventual agreement still holds, as the blockchain keeps running. One can even mitigate such a catastrophic event by having a fallback finalization on the blockchain using a very conservative common-prefix bound. We propose it as interesting future work to further explore such a graceful degradation of security and recovery from catastrophic events. A modular two-layer design dividing the responsibility of eventual agreement and finality seems to facilitate this study.

We can summarize the advantages of our design as follows: When the network conditions are good and the corruption is below $n/3$, our finality layer can declare blocks as final as soon as they reach the common-prefix of the underlying blockchain (e.g., after at most 6 blocks instead of more than 50 blocks in the Bitcoin example above). On the other hand, when the corruption threshold is between $n/3 < t < n/2$, the finality layer can turn off, and the blockchain keeps running by relying on its eventual agreement through its common-prefix property.

When to accept a transaction. To take full advantage of our two-layer approach as outlined above and obtain the best of both worlds for security and efficiency, we suggest the following method for deciding when a transaction in a certain block can be accepted. Under normal conditions, accept a transaction if it is in a block that lies on the path to the last finalized block. This should normally happen faster than the Nakamoto-style rule of waiting for sufficiently many blocks and at a speed comparable to getting a new block in CBFT consensus designs.

If the finality layer breaks because of bad network conditions or too much corruption, fall back to Nakamoto-style finality. In more detail, if conflicting blocks are marked as final by the finality layer, ignore the finality layer and consider blocks final that are far enough back. Note that this can only happen if there is more than $n/3$ corruption and in this case, CBFT consensus designs fail completely. Furthermore, if the last finalized block is much further back than usual, the parties can decide based on the transaction value and on how much they trust the sender, whether they are willing to fall back to the Nakamoto-style rule, or keep waiting for the finality layer to continue working. This situation can again arise if there is more than $n/3$ corruption or if the network is partitioned. Again, in both cases, all bets are off with a pure BFT consensus design.

Communication model. We consider a network model that is *partially synchronous* (or semi-synchronous) [12, 11]. This means that there is an upper bound $\Delta_{\mathtt{net}}$ on the network delay, but in contrast to the synchronous model, this bound is not known to the protocol. In particular, $\Delta_{\mathtt{net}}$ cannot be used in a partially synchronous protocol. This can be modeled by letting the adversary pick $\Delta_{\mathtt{net}}$ after the protocol was designed.

We next argue that this is an appropriate model for blockchain finality layers. Assuming that it is possible for parties to exchange messages within some bounded time seems to be reasonable in practice (note that this bound can be very large, say one day). Typically, the network will run much faster than this bound, though. Using some large bound in a synchronous protocol is therefore hugely inefficient. On the other hand, a partially synchronous protocol runs fast if the network is fast as it cannot depend on the unknown upper bound, only on the actual network delay. As we discuss below, a finality layer is particularly useful in situations where the network is partitioned for some limited time (due to for instance a misconfiguration of the network, a distributed denial of service (DDoS) attack, or a huge earth quake crippling Internet connectivity). In such situations, messages cannot be delivered within the usual timeframe (so synchronous protocols assuming a too small bound fail), but they can be delivered again once the network rejoins. This therefore fits nicely into the partially synchronous model, where $\Delta_{\mathtt{net}}$ can just be set to be larger than the duration of such partitions.

An alternative model of partial synchrony is also proposed in [12]. In that model model, a protocol $\pi_{\Delta_{\mathtt{Guess}}}$ has a hard-coded guess $\Delta_{\mathtt{Guess}}$ of the network delay $\Delta_{\mathtt{net}}$. The network model guarantees that eventually $\Delta_{\mathtt{net}} < \Delta_{\mathtt{Guess}}$ for long enough to allow the protocol to progress and have liveness. During non-synchronous periods the protocol should not violate any safety properties. For the sake of comparison, we here refer to that model as *eventually synchronous*, and to the model we use as *unknown bound* model. Note that any protocol π that works in the unknown bound model also works in the eventually synchronous model: If in the eventually synchronous model, from time t_0 on all messages are delivered within time $\Delta_{\mathtt{Guess}}$, this means that messages are always delivered within time $\Delta_{\mathtt{net}} \coloneqq t_0 + \Delta_{\mathtt{Guess}}$, and π works for all $\Delta_{\mathtt{net}}$. The converse is not true, however: A protocol $\pi_{\Delta_{\mathtt{Guess}}}$ for the eventually synchronous model needs to know a bound $\Delta_{\mathtt{Guess}}$ and thus cannot be used directly in the unknown bound model. We therefore use the unknown bound model in this paper (and from now on simply refer to it as partially synchronous).

Combining a partially synchronous finality layer with a synchronous blockchain. It is clear that a finality layer should be partially synchronous (or asynchronous) if the underlying blockchain is partially synchronous (cf. [11, 3]). But even when combined with a blockchain that relies on synchrony, we argue why a finality layer should be partially synchronous: Consider a two-layer approach with a synchronous blockchain and a finality layer. While the synchrony assumption holds, the blockchain and the finality layer keep running smoothly. If the synchrony assumption at some point gets broken (and the underlying blockchain properties are void), e.g., due to a catastrophic event or a massive attack on the network, a synchronous finality layer can completely break and finalize conflicting blocks. A partially synchronous finality layer, on the other hand, can in the worst case pause (and loose liveness). Once the network recovers and the underlying blockchain re-starts making progress, the partially synchronous finality layer continues finalizing blocks. This means in particular that our method of dealing with the CAP theorem described above crucially relies on the finality layer being partially synchronous.

1.1 Our Techniques

We assume the existence of a *finalization committee* such that less than 1/3 of the committee can be corrupted. We emphasize that selecting such a committee is an independent problem, which has already been studied in the literature and is not in the scope of this work. For completeness, in Section 8, we give an overview and references to some known strategies on how to select committees. The finalization committee is responsible for finalizing the next block. The block they are to finalize is the one at depth d, where d is some depth in the tree that they all agree on and which is deeper than the currently last finalized block. To ensure all parties agree on the value of d, it is deterministically determined by the blocks from genesis up to the last finalized block.

Insufficiency of using arbitrary Byzantine agreement. At first, it may appear that there is an easy way to finalize a block at depth d: Simply let the committee run some existing Byzantine agreement protocol to agree on a block at depth d, which is then declared final. Typical Byzantine agreement protocols, however, do not provide the guarantees we need: the usual validity property only guarantees that if all honest parties have the same input value, they agree on this value; if honest parties start with different values, the agreement can be on an arbitrary value. This means if we use this approach and start finalization before depth d is in the common prefix of all honest parties, any block, even ones not on the chain of any honest party, can be declared as final. This is clearly undesirable and violates the support property we require from finality layers.

Better guarantees could be achieved by using Byzantine agreement with strong validity introduced by Neiger [26]. This requires the agreed value to be the input of an honest party. Even this strong notion, however, only gives 1-support, while we aim for higher support. Furthermore, strong impossibility results are known for this type of validity if the set of possible input values is large, which is the case in our setting since there could be many different blocks at depth d.

Our solution. The basic insight that allows us to overcome these limitations is that we can utilize the common-prefix property of the underlying blockchain: If we wait long enough, all honest parties will have the same block at depth d on their chain. In that case, they can decide on this block. If honest parties have different blocks at depth d, they can just agree to wait longer.

More concretely, our protocol proceeds as follows. When a committee member has a best chain which reached depth $d + 1$, it votes on the block it sees at depth d on its best chain using a committee-based Byzantine fault tolerance consensus protocol (CBFT). This protocol is designed such that it succeeds if all parties vote for the same block, otherwise it might fail. If the CBFT announces success, the block that it outputs is defined to be final. If the CBFT reports failure, the committee members will iteratively retry until it succeeds. In the i'th retry they wait until they are at depth $d + 2^i$ and vote for the block they see at depth d on their best chain. Eventually 2^i will be large enough that the block at depth d is in the common-prefix, and then the CBFT will succeed. The process then repeats with the next committee and the next depth $d' > d$.

This finality layer works under the assumption that there is some non-trivial common-prefix. It does not need to know how long it is, it only assumes that some unknown upper bound exists. Note that the length of the common prefix generally can, among other things, depend on the network delay, which is unknown in our partially synchronous model. This also gives responsive finality: when the number of blocks after the common-prefix value is low, we finalize quickly. Furthermore, it makes the finality layer work as a hedge against catastrophic events, during which there are more blocks than usual after the common prefix.

Common-prefix and unique justified votes. The procedure described above ensures that at some point, the block to be finalized at depth d is in the common-prefix. Then, the common-prefix property ensures that all *honest* parties vote for that block. It is still possible for dishonest parties to vote for another block. We propose two protocol variants that deal with this in different ways.

The first variant requires an additional property of the underlying blockchain, which we call bounded *dishonest chain growth*. It implies that a chain only adopted by dishonest parties grows slower than the chains of honest parties. This holds for many blockchains (assuming honest majority of the relevant resource), but it may not hold, e.g., if the blockchain allows parties to adaptively adjust the hardness of newly added blocks. In that case, dishonest parties can grow long chains with low hardness quickly without violating the common-prefix property, since honest parties will not adopt those chains with low hardness.

Given this additional property, we have that at some point, there will be only one block at depth d lying on a sufficiently long path. When a party votes for a block at depth d, we ask the party to *justify* the vote by sending along an extension of the path of length 2^i. So we can ask that our CBFT has success only when *all* parties vote the same, even the corrupted parties. In all other cases it is allowed to fail. Since any path can eventually grow to any length, the property that there is a unique justified vote is temporary. We therefore start our CBFT with a so-called Freeze protocol which in a constant number of rounds turns a temporarily uniquely justified vote into an eternally uniquely justified vote. After that, the CBFT can be finished by running a Boolean-valued Byzantine agreement on whether Freeze succeeded.

The second protocol variant does not rely on bounded dishonest chain growth and consequently works with any blockchain with the typical properties. We still get from the common-prefix property that at some point, all *honest* parties will vote for the same block. We exploit this by adding an additional step at the beginning of the first protocol variant, which tries to filter out votes that come only from dishonest parties.

Keeping up with chain growth. The updated property of the finality layer requires that the finalized blocks do not fall behind the underlying blockchain too much. To guarantee this, the depths for which finalization is attempted need to be chosen appropriately. Ideally, we would like the distance between two finalized blocks to correspond to the number of blocks the chain grows during the time needed for finalization. Since parties have to agree on the next depth to finalize beforehand, they can only use information that is in the chain up to the last finalized block to determine the next depth.

We use the following idea to ensure the chain eventually catches up with the chain growth: When parties add a new block, they include a pointer to the last block that has been finalized at that point. They also include a witness for that finalization, so that others can verify this. If the chain does not grow too fast, at the time a finalized block is added to the chain, the previously finalized block should already be known. If the chain grows too fast, however, we keep finalizing blocks that are too high in the tree. In the latter case, the pointer to the last finalized block in some block will be different from the actually last finalized block. If we detect this, we can adjust how far we need to jump ahead with the following finalization.

1.2 Related Work

To the best of our knowledge, there is no prior work on *provably secure* finality layers, let alone works formalizing the desirable properties of finality layers.

The most closely related work seems to be Casper [7], which was the first proposal of a modular finality layer that can be built on top of a Nakamoto-style blockchain. Casper presents a finality layer for PoW blockchains where a finalization committee is established by parties that are willing to "deposit" coins prior to joining the committee. The committee members can vote on blocks that they wish to make final and a CBFT protocol is used to achieve agreement; if more than 2/3 of the committee members (weighted by deposit value) vote on the same block, then the block becomes "final". Casper also employs a penalty mechanism known as slashing; if a committee member signs two conflicting votes, its previously deposited coins can be slashed from the system as a penalty. However, since the authors do not present a formal network model and a detailed protocol description and analysis, it is not clear if the Casper protocol guarantees liveness and/or safety in our partially synchronous model. In particular, the authors only consider what they call "plausible liveness", but there is no guarantee that liveness actually holds.

We stress that we are *not* presenting a blockchain, but rather a finality layer that runs on top of a blockchain. Yet, it is instructive to compare our finality layer to existing consensus protocols.

The consensus protocol closest to ours is Hybrid Consensus by Pass and Shi [28], and the closely related Thunderella [29]. They take an underlying synchronous blockchain and use it to elect a committee. Then the committee runs a CBFT protocol to get a responsive consensus protocol, i.e., the committee is producing the blocks. This does not add finality to the underlying blockchain, and blocks produced by the committee are not final either, unless the committee itself is already final. If the blockchain experiences a rollback behind the block where the committee was elected, all the blocks of the committee are lost. The way this is handled in Hybrid Consensus is to assume that the underlying blockchain has a known upper bound on how long rollbacks can be. They then look that far back in the currently best chain to elect the committee. One disadvantage is that the committee will tend to be old, so they implicitly assume that nodes stick around for a long time. Also, the design is not robust against a catastrophic partitioning. If some day the network partitions for a long time, separate committees will form on both sides of the partitioning. We could cast our work in terms of theirs as follows: we can elect the next committee in the same way as HC. But the committee would not produce blocks, instead it introspectively tries to agree on a recent block in the underlying blockchain. We then do a binary search to look far enough back to reach agreement. When we agree, that block is defined as final. Now we could use that final block to elect the next committee in the same way as HC. That way, we can typically elect the next committee from a much more recent block. Thus, we do not need to assume that recent block winners stay online for as long as HC. In principle after having added responsive finality like above to any blockchain, one could add HC on top: elect committees from recent final blocks and let the committee produce blocks using a CBFT. We, however, conjecture that eventually the fastest designs will be by fast blockchains with a finality layer on top and not from blockchains with a block producing CBFT committee, at least when run with comparable cryptographic security. This is an interesting question to explore in future theoretical and experimental work.

Another closely related consensus protocol is Algorand [23, 16]. It also uses a CBFT to produce blocks. In Algorand the committee is elected at random using a proof-of-stake approach. It introduces a very elegant protocol which is "player replaceable". For each move in the protocol, like "send player i's messages in round r", a party is elected using a lottery. The protocol is such that any party can take on the role and each winner speaks only once. The adversary cannot predict who is the winner until the message is sent. This gives an ultimate protection against DDoS attacks. Like other CBFT designs, if the network partitions, Algorand will deadlock: it picks consistency over availability by design. This is because in at least one side of the partition a too small committee will be elected. This is under the assumption that honest participation does not fluctuate too fast. In case of rapid fluctuations in the amount of online honest stake this could lead to a fork. To illustrate this, consider an extreme example: if the amount of stake quickly doubles and the network experiences a temporary partition at the same time, a new committee of sufficient size might be elected on both sides of the partition, leading to a fork. An adversary can try to provoke this by turning all his stake active at the time of a spike in active honest stake and mounting a DDoS attack on the network at the same moment. An interesting open problem is to get a player replaceable CBFT which can simultaneously tolerate rapid fluctuations in honest participation and network partitions.

Another recent example of a partially synchronous CBFT blockchain is PaLa [10], where the authors propose a blockchain with finality built-in. As other CBFT designs, it provides not liveness during partitions and lacks any guarantees under $t \geq n/3$ corruption.

1.3 Outline

In Section 2, we describe our assumptions on the network and the overall model, and recall some basic concepts from graph theory that we use later. In Section 3, we describe how we model the underlying blockchain and our assumptions on its properties. We formalize the goal of a finality layer in Section 4. In Section 5, we present the Afgjort protocol, which uses a weak multi-valued Byzantine agreement that we present in Section 6. In Section 7, we prove that the Afgjort protocol satisfies the properties of a finality layer we introduced in Section 4. In Section 8 we finally discuss how to select finalization committees.

2 Preliminaries

2.1 Model and Network Assumptions

We assume that there is a physical time $\tau \in \mathbb{N}$ that is monotonously increasing and that parties have access to local clocks. These clocks do not have to be synchronized; we only require the clocks to run at roughly the same speed. We need that they drift from τ by at most some known bound $\Delta_{\texttt{Time}}$. For the sake of simpler proofs we will pretend in proofs that $\Delta_{\texttt{Time}} = 0$. All proofs can easily be adapted to the case of a known $\Delta_{\texttt{Time}} > 0$.

For simplicity, we assume that there is a fixed set of parties $\mathcal{P}$ with $n \coloneqq |\mathcal{P}|$, where we denote the parties by $P_i \in \mathcal{P}$. There is an adversary which can corrupt up to $t \in \mathbb{N}$ parties. We call P_i honest if it was not corrupted by the adversary. We use `Honest` to denote the set of all honest parties. For simplicity, we here assume static corruptions, i.e., the adversary needs to corrupt all parties at the beginning. The set of parties $\mathcal{P}$ constitutes what we call a committee. In Section 8 we discuss how the set $\mathcal{P}$ can be sampled from a blockchain.

Network. We further assume parties have access to a gossip network which allows them to exchange messages. This models how the peer-to-peer layer distributes messages in typical blockchains. We work in a partially synchronous model, which means that there is a hidden bound $\Delta_{\texttt{net}}$ on message delays. In contrast to synchronous networks, $\Delta_{\texttt{net}}$ is not known, i.e., the protocols cannot use $\Delta_{\texttt{net}}$, they can only assume the existence of some bound. One can think of $\Delta_{\texttt{net}}$ as an unknown parameter of the assumed network functionality, or alternatively as being chosen by the adversary at the beginning of the protocol execution (after the protocol has been fixed). More concretely, we make the following assumptions on the network:

- When an honest party sends a message at time τ, all honest parties receive this message at some time in $[\tau, \tau + \Delta_{\texttt{net}}]$.
- When an honest party receives a message at time τ (possibly sent by a dishonest party), all honest parties receive this message until time $\tau + \Delta_{\texttt{net}}$.

Remark 1. The above assumptions on the network are not realistic for a basic gossip network. We have chosen them because they allow for a proof focusing on the important and novel aspects of our protocol. The assumptions can be weakened significantly. A weaker network model could for instance assume that if the network partitions, it is always at some future point connected again for long enough, and that there exist an unknown bound Δ such that the network will not drop a message sent between two connected parties if the same message is sent Δ times. In such a model we could for instance let all honest committee members save all messages they sent in the ongoing finalization attempt. They will keep occasionally resending these message until they see the finalization attempt terminate. That way we would only need that all honest committee members are eventually connected to all other members for long enough. Furthermore, parties would only have to store a finite number of messages, namely those belonging to the current finalization event.

Signatures. We finally assume that each party has a signing key for some cryptographic signature scheme where the verification key is publicly known (e.g., is on the blockchain). For our analysis, we assume signatures are perfect and cannot be forged. Formally, this can be understood as parties having access to some ideal signature functionality [9, 2]. We do not model this in detail here because the involved technicalities are not relevant for our protocols.

2.2 Graphs and Trees

We next recall some basic concepts from graph theory.

Definition 1. A *graph* $G = (V, E)$ consists of a set of *nodes* V and a set of *edges* E, where every edge $e \in E$ is a 2-element subset of V. A *path* (also called *chain*) of length $k-1$ in G is a sequence $(v_1, \ldots, v_k)$ of distinct nodes such that $\{v_i, v_{i+1}\} \in E$ for all $i \in \{1, \ldots, k-1\}$.

The graphs we are most interested in are trees. We only consider trees with a root (corresponding to the genesis block) in this work and always mean *rooted tree* when saying *tree*.

Definition 2. A *(rooted) tree* T is a graph (V, E) together with a node $r \in V$, called *root*, such that for every $v \in V$, there is a unique path from r to v. We denote this path by $\mathrm{PathTo}(T, v)$. A *leaf* is a node $v \in V \setminus \{r\}$ that occurs in only one edge. We further let $\mathrm{Depth}(T, v)$ be the length of $\mathrm{PathTo}(T, v)$ and $\mathrm{Height}(T, v)$ be the length of the longest path from v to a leaf. When the tree is clear from context, we may also write $\mathrm{PathTo}(v)$, $\mathrm{Depth}(v)$, and $\mathrm{Height}(v)$. The height of a tree equals the height of its root: $\mathrm{Height}(T) \coloneqq \mathrm{Height}(T, r)$ (equivalently, the height of a tree is the depth of its deepest node).

Remark 2. Some papers in the blockchain literature use the term height for what we call depth, e.g., [7]. We instead use the terms depth and height as common in computer science literature, which are derived from the understanding that tree data structures grow from top (root) to bottom (leaves).

Definition 3. Let $T = ((V, E), r)$ be a rooted tree and let $u, v \in V$ be two nodes. If u is on $\text{PathTo}(T, v)$, then u is an *ancestor* of v and v is a *descendant* of u.

One can define several operations on graphs. The ones we need are the union and intersection, which are simply defined as the union and intersection of the nodes and edges, respectively.

Definition 4. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_1, E_2)$, we define their union as $G_1 \cup G_2 := (V_1 \cup V_2, E_1 \cup E_2)$, and their intersection as $G_1 \cap G_2 := (V_1 \cap V_2, E_1 \cap E_2)$. For two rooted trees $T_1 = (G_1, r)$, $T_2 = (G_2, r)$ with common root r, we define $T_1 \cup T_2 := (G_1 \cup G_2, r)$ and $T_1 \cap T_2 := (G_1 \cap G_2, r)$.

Note that $T_1 \cup T_2$ and $T_1 \cap T_2$ are not necessarily trees.

3 Abstract Model of Blockchains

We want to describe our finality layer independently of the underlying blockchain protocol. Therefore, we use an abstract model that captures only the relevant properties needed for our finalization layer. The properties are modeled via an ideal functionality $\mathcal{F}_{\text{TREE}}$, to which all parties have access.

3.1 Description of Tree Functionality

At a high level, $\mathcal{F}_{\text{TREE}}$ provides each party access to their view of all existing blocks arranged in a tree with the genesis block at its root. The adversary can grow these trees under certain constraints. Formally we give the adversary access to commands which grow the individual trees $\texttt{Tree}_i$ of the parties P_i. We also give party P_i access to a GetTree command which returns the current $\texttt{Tree}_i$. There are additional commands covered below.

The functionality $\mathcal{F}_{\text{TREE}}$ maintains several variables that evolve over time. For a time τ and a variable X, we use X^τ to denote the value of the variable X at time τ.

Inside $\mathcal{F}_{\text{TREE}}$ each P_i has an associated tree $\texttt{Tree}_i$. The nodes in these trees correspond to blocks and can contain several pieces of information, which we do not further specify since this is not relevant here. We only assume that blocks contain a field for some metadata `data` used by our finalization protocols. The party P_i can read $\texttt{Tree}_i$ but is not allowed to modify it. All trees have a common root G, called genesis, and initially, all trees only consist of G. We let

$$\texttt{HonestTree} := \cup_{P_i \in \texttt{Honest}} \texttt{Tree}_i$$

be the graph that consists of all blocks in the view of any honest party. The adversary can add nodes to any tree at will, under the constraint that `HonestTree` remains a tree at all times.

All P_i also have a position $\texttt{Pos}_i \in \texttt{Tree}_i$. We require that $\texttt{Pos}_i$ is a leaf of $\texttt{Tree}_i$ and can be set at will by the adversary. If the adversary adds a node in $\texttt{Tree}_i$ that is a child of $\texttt{Pos}_i$, $\texttt{Pos}_i$ gets updated to be the new leaf.

Recall that for a node B in $\texttt{Tree}_i$, $\text{PathTo}(\texttt{Tree}_i, B)$ denotes the (unique) path from the root to B. We define $\texttt{Path}_i := \text{PathTo}(\texttt{Tree}_i, \texttt{Pos}_i)$. In a typical blockchain protocol, $\texttt{Path}_i$ corresponds to the best chain (e.g., the longest chain, or the chain with maximal total hardness) in the view of P_i.

Remark 3. New blocks are typically not added only by the adversary, but also by honest parties that are baking. Furthermore, the positions of honest parties are not set by the adversary, but by the parties themselves following some chain selection rule, e.g., by setting the position to the deepest leaf in the tree. We give the adversary full control over these two aspects for two reasons: First, it allows us to abstract away details about these mechanisms. Secondly, giving the adversary more power makes our results stronger.

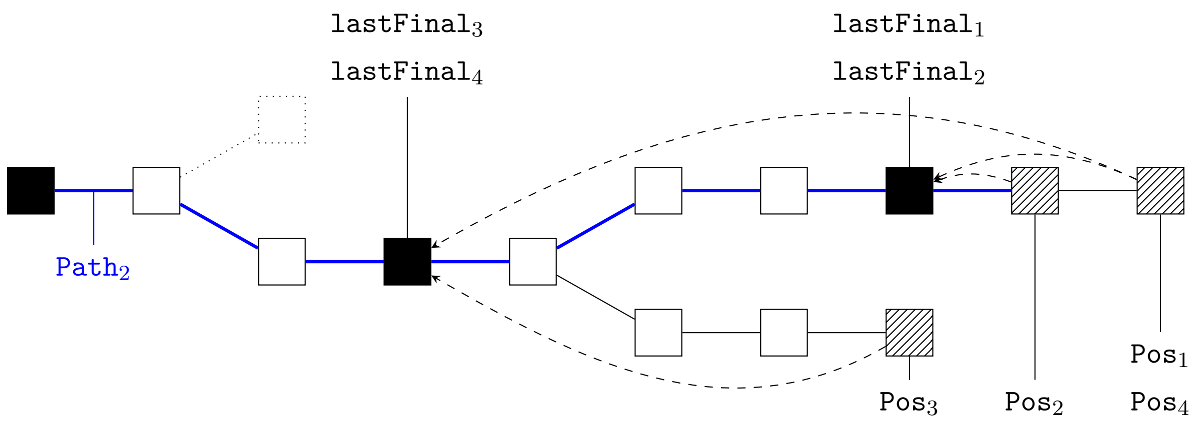

Figure 1: Example of a possible `HonestTree` with honest parties P_1, P_2, P_3, and P_4. The block at the very left is the genesis block. Finalized blocks are drawn in solid black, blocks corresponding to positions of honest parties with a dashed pattern. The dashed arrows point to `lastFinal` of the parties having their positions at the origins of the arrows. In this example, P_3 and P_4 have not yet learned about the third finalized block. The dotted node does not belong to `FinalTree` because it is not on a path through `lastFinal` of any honest party. Everything else is part of `FinalTree`. The thick blue line corresponds to $\texttt{Path}_2$.

Finalization friendliness. To be able to finalize, we need the blockchain to be finalization friendly. This basically means that it needs to provide an interface for our finalization protocols. Concretely, parties need to additionally have access to the two commands SETFINAL and PROPDATA. A party calls $(\textsc{setFinal}, R)$ once this party considers R to be final. More formally, each party has a variable $\texttt{lastFinal}_i \in \texttt{Tree}_i$, initially set to the genesis block G. The command $(\textsc{setFinal}, R)$ for $R \in \texttt{Tree}_i$ sets $\texttt{lastFinal}_i$ to R. Inputs $(\textsc{setFinal}, R)$ by P_i where R is not a descendant of $\texttt{lastFinal}_i$ are ignored. The intended effect on the blockchain is that parties will eventually set their position to be a descendant of R and maintain this indefinitely. In our formalization, this corresponds to a restriction on how the adversary sets the positions and is discussed in Section 3.2. In a real blockchain protocol, this can be achieved by modifying the chain selection rule to reject all chains not containing R. For honest P_i we use $\texttt{FinalTree}_i$ to be the tree consisting of all paths in $\texttt{Tree}_i$ going through $\texttt{lastFinal}_i$. Note that this consists of only a single path from G to $\texttt{lastFinal}_i$ and then possibly a proper tree below $\texttt{lastFinal}_i$. We let $\texttt{FinalTree} = \cup_{P_i \in \texttt{Honest}} \texttt{FinalTree}_i$.

The command $(\textsc{propData}, \texttt{data})$ allows parties to propose some $\texttt{data} \in \{0,1\}^*$ to be included in a future block. This is different from transactions being added to blocks in that we only have weak requirements on it: Roughly speaking, we want a constant fraction of all honest paths to contain `data` corresponding to the last proposal of some honest party at the time the block was first added. This requirement is discussed in more detail in Section 3.2; here we only assume the adversary can add arbitrary data to blocks, which is implicit in the model since blocks are chosen by the adversary.

Figure 1 shows an example of a tree with the relevant variables. We conclude with a formal specification of the functionality $\mathcal{F}_{\textsc{Tree}}$.

Functionality $\mathcal{F}_{\text{Tree}}$

Initialization

for $P_i \in \mathcal{P}$ **do**
  $\texttt{Tree}_i \coloneqq ((V_i \coloneqq \{G\}, E_i \coloneqq \varnothing), r_i \coloneqq G)$
  $\texttt{Pos}_i \coloneqq G$, $\texttt{lastFinal}_i \coloneqq G$, $\texttt{lastProp}_i \coloneqq \bot$
end for

Interface for party $P_i \in \mathcal{P}$

Input: getTree
  return copy of $(\texttt{Tree}_i, \texttt{Pos}_i, \texttt{lastFinal}_i)$ to P_i

Input: $(\text{setFinal}, R)$
  if $\texttt{lastFinal}_i \in \text{PathTo}(\texttt{Tree}_i, R)$ **then**
    $\texttt{lastFinal}_i \coloneqq R$
    send $(\text{setFinal}, P_i, R)$ to adversary
  end if

Input: $(\text{propData}, \texttt{data})$
  $\texttt{lastProp}_i \coloneqq \texttt{data}$
  send $(\text{propData}, P_i, \texttt{data})$ to adversary

Interface for adversary

Input: $(\text{addNode}, P_i, B, p)$ // add B as child of p in $\texttt{Tree}_i$
  if $B \notin V_i$ and $\texttt{HonestTree}$ remains a tree after adding B as child of p in $\texttt{Tree}_i$ **then**
    $V_i \coloneqq V_i \cup \{B\}$
    $E_i \coloneqq E_i \cup \{\{p, B\}\}$
    if $p = \texttt{Pos}_i$ **then**
      $\texttt{Pos}_i \coloneqq B$
    end if
  end if

Input: $(\text{setPosition}, P_i, B)$ // set position of P_i to B
  if B is a leaf of $\texttt{Tree}_i$ **then**
    $\texttt{Pos}_i \coloneqq B$
  end if

3.2 Desirable Properties and Bounds

We now state some important assumptions and properties of blockchain protocols in our model. All properties are essentially restrictions on how the adversary can grow the trees. The definitions below involve a number of so-called *hidden bounds*. These parameters are supposed to exist (possibly depending on the security parameter), but are *not* made public to the parties. In particular, they cannot be used in the protocols; one may only assume in proofs that these parameters exist. We require that the bounds are polynomial in the security parameter.

We first define two properties that are not directly related to the security of the blockchain, but rather follow from the assumptions on the network and how the protocols are supposed to work. Widely considered properties of blockchain protocols include common prefix, chain growth, and chain quality, introduced in [14, 18, 27]. We recast the two former in our model. Since our model does not have a notion of a party creating a block, chain quality is not directly applicable. We instead formalize two properties we need for our protocol that are implied by chain quality and chain growth.

Tree propagation. There is a hidden bound $\Delta_{\mathtt{tree}}$ such that for all honest parties $P_i \in \mathtt{Honest}$, $\mathtt{HonestTree}^{\tau-\Delta_{\mathtt{tree}}} \subseteq \mathtt{Tree}_i^{\tau}$. This models the case that when a honest party sees a chain, then eventually all honest parties will see that chain.

New root taking effect. There is a hidden parameter $\Delta_{\mathtt{final}}$, that intuitively is the time that it takes for a $\textsc{setFinal}$ command to take effect. We require that $R \in \mathtt{Path}_i$ after $\Delta_{\mathtt{final}}$ time units since P_i gave the command $(\textsc{setFinal}, R)$. This means that the adversary must in reasonable time put P_i under the finalized block R, and when this happens $\mathtt{Pos}_i$ will stay in a path under R forever.

Common prefix. The common-prefix property intuitively means that if any two honest parties look far enough back in their own tree, then they will be on the same path to the root. We formally define this property for $\xi \in \mathbb{N}$, which determines how far parties have to look back, via the predicate $\mathrm{Prefix}(\xi)$:

$$\mathrm{Prefix}(\xi) :\equiv \forall \tau_1, \tau_2 \in \mathbb{N}, \tau_1 \leq \tau_2,\ \forall P_1, P_2 \in \mathtt{Honest}\ \left(\mathtt{Path}_1^{\tau_1}\right)^{\lceil \xi} \preceq \mathtt{Path}_2^{\tau_2},$$

where $(\cdot)^{\lceil \xi}$ denotes the operation of removing the last ξ blocks and $\preceq$ is the prefix relation.

Chain growth. The chain-growth property guarantees that chains of honest parties grow within time $\Delta_{\mathtt{growth}}$ at least at rate $\rho_{\mathtt{growth}}$ and at most at rate $\rho'_{\mathtt{growth}}$, $0 < \rho_{\mathtt{growth}} \leq \rho'_{\mathtt{growth}}$. Note that the chain of party P_i in our model corresponds to $\mathtt{Path}_i$ and its length is equal to $\mathrm{Depth}(\mathtt{Pos}_i)$. We thus use the following formalization:

$$\begin{aligned}\mathrm{ChainGrowth}(\Delta_{\mathtt{growth}}, \rho_{\mathtt{growth}}, \rho'_{\mathtt{growth}}) :\equiv\ & \forall \tau \in \mathbb{N}\ \forall P_i \in \mathtt{Honest} \\ & \rho_{\mathtt{growth}} \cdot \Delta_{\mathtt{growth}} \leq \mathrm{Depth}\big(\mathtt{Pos}_i^{\tau+\Delta_{\mathtt{growth}}}\big) - \mathrm{Depth}\big(\mathtt{Pos}_i^{\tau}\big) \leq \rho'_{\mathtt{growth}} \cdot \Delta_{\mathtt{growth}}.\end{aligned}$$

Remark 4. Some papers, e.g., [27, 11] consider a stronger variant of chain growth by comparing the lengths of chains from two different honest parties at different times. For our purposes, the simple definition above that only considers a single party is sufficient.

Remark 5. Note that earlier formalizations of chain growth only considered a lower bound on growth. It turns out that for several non-trivial uses of blockchains, one also needs an upper bound as introduced in [27]. It is for instance impossible to create a finalization layer which keeps being updated if the underlying blockchain can grow by an unbounded length in one time unit. For any length L that you might want as a bound on how far finalization can fall behind, the blockchain could grow by $L + 1$ blocks faster than it takes one message in the finalization protocol to propagate. In such a model one would get trivial impossibility of designing updated finalization layers.

Bounded path growth. The chain growth property above only bounds the growth of the positions of honest parties. That is, it does not prevent purely dishonest chains to grow faster. To prove the updated property of our finality layer, we need the following slightly stronger property: Denote by $\tau(B)$ the first time a block B appeared in $\texttt{HonestTree}$. Bounded path growth with parameters $\Delta_{\texttt{pgrowth}}$ and $\rho_{\texttt{pgrowth}}$ says that

$$\forall \tau \in \mathbb{N}\ \forall P_i \in \texttt{Honest}\ \forall B_1, B_2 \in \texttt{Path}_i^\tau\ \big((\mathrm{Depth}(B_2) \geq \mathrm{Depth}(B_1) + \Delta_{\texttt{pgrowth}}) \\ \to \mathrm{Depth}(B_2) - \mathrm{Depth}(B_1) \leq \rho_{\texttt{pgrowth}} \cdot (\tau(B_2) - \tau(B_1))\big).$$

This means that for any two blocks with sufficient distance on the path of an honest party, the path between these blocks cannot have grown arbitrarily fast. If we assume chain quality, this property follows from bounded chain growth: If a certain fraction of the blocks on this path have been generated by honest parties, the growth of this path gets bounded since honest parties are subject to chain growth. Note that we still allow completely dishonest paths to grow arbitrarily as long as no honest party ever moves there.

Proposal quality. This property is formally unique to our finalization friendliness involving the PROPDATA command, but it is closely related to chain quality as discussed next. *Proposal quality* with parameter $\ell_{\texttt{PQ}} \in \mathbb{N}$ means that at any time τ, for all honest $P_i \in \texttt{Honest}$, and for all $\ell_{\texttt{PQ}}$ consecutive blocks $B_1, \ldots, B_{\ell_{\texttt{PQ}}}$ in $\texttt{Path}_i^\tau$, there exists a block $B' \in \{B_1, \ldots, B_{\ell_{\texttt{PQ}}}\}$ that was added to $\texttt{HonestTree}$ at time τ' and an honest party $P_j \in \texttt{Honest}$ such that $\texttt{lastProp}_j^{\tau'}$ is contained in (the $\texttt{data}$ field of) a block on $\mathrm{PathTo}(\texttt{HonestTree}^{\tau'}, B')$. In other words, at the time B' is added to $\texttt{HonestTree}$, if the last proposal of some honest party is not already contained in an ancestor of B', that proposal is included in B'.

Note that proposal quality can be achieved by any blockchain that has chain quality: Chain quality with parameters μ and ℓ' says that within any sequence of at least ℓ' consecutive blocks in an honest path, the ratio of blocks generated by honest parties is at least μ. This implies that for $\ell_{\texttt{PQ}} \geq \ell'$ with $\ell_{\texttt{PQ}} \cdot \mu \geq 1$, at least one block within $\ell_{\texttt{PQ}}$ consecutive blocks is generated by an honest party. Whenever honest parties add a block B', they can check whether their last proposed $\texttt{data}$ is already contained in a previous block, and if not, they include that $\texttt{data}$ in B'. This yields proposal quality with parameter $\ell_{\texttt{PQ}}$.

Dishonest chain growth. We here introduce a new property that is needed for the more efficient variant of our protocol. It is concerned with how fast dishonest parties can grow chains. The usual chain growth property bounds the growth of the positions of honest parties. We here consider a bound on the growth of chains no honest party is positioned on. In typical blockchain protocols, bakers extend the chains at their positions, i.e., we are here interested in how fast dishonest parties can grow their chains.

Definition 5. For $\tau \in \mathbb{N}$ and $B \in \texttt{FinalTree}^\tau$, let $\hat{B}$ to be the deepest ancestor of B in $\texttt{FinalTree}^\tau$ that has at some point been on an honest path,

$$\hat{B} := \operatorname*{argmax}_{B' \in \mathrm{PathTo}(\texttt{FinalTree}^\tau, B) \cap \left(\cup_{\tau' \leq \tau} \cup_{P_i \in \texttt{Honest}} \texttt{Path}_i^{\tau'}\right)} \{\mathrm{Depth}(B')\},$$

and let $\hat{\tau}_B$ be the first time $\hat{B}$ appeared in an honest path:

$$\hat{\tau}_B := \min\left\{\tau' \in \mathbb{N} \;\middle|\; \hat{B} \in \bigcup_{P_i \in \texttt{Honest}} \texttt{Path}_i^{\tau'}\right\}.$$

Let $\Delta_{\texttt{growth}} \in \mathbb{N}$, and $\rho_{\texttt{disgro}} \geq 0$. We define the *dishonest chain growth* with parameters $\Delta_{\texttt{growth}}, \rho_{\texttt{disgro}}$ to hold if for all B in $\texttt{FinalTree}^\tau$ such that $\tau - \hat{\tau}_B \geq \Delta_{\texttt{growth}}$, the length of the path from $\hat{B}$ to B is bounded by $\rho_{\texttt{disgro}} \cdot (\tau - \hat{\tau}_B)$, and by $\rho_{\texttt{disgro}} \cdot \Delta_{\texttt{growth}}$ if $\tau - \hat{\tau}_B < \Delta_{\texttt{growth}}$:

$$\begin{aligned}\mathrm{DCGrowth}(\Delta_{\texttt{growth}}, \rho_{\texttt{disgro}}) :\equiv \forall \tau \in \mathbb{N}\ \forall B \in \texttt{FinalTree}^{\tau} \\ \mathrm{Depth}(B) - \mathrm{Depth}(\hat{B}) \leq \rho_{\texttt{disgro}} \cdot \max\{\Delta_{\texttt{growth}}, \tau - \hat{\tau}_B\}.\end{aligned}$$

Intuitively, the path from $\hat{B}$ to B is grown only by dishonest parties since no honest party was ever positioned on it, and $\tau - \hat{\tau}_B$ is the time it took to grow this path. Taking the maximum over $\Delta_{\texttt{growth}}$ and $\tau - \hat{\tau}_B$ allows that for periods shorter than $\Delta_{\texttt{growth}}$, the growth can temporarily be faster. Note that it is possible that the adversary knows $\hat{B}$ before it appears on an honest path or even in `FinalTree`. In that case, there is actually more time to grow the chain. The definition thus implicitly excludes that dishonest parties know blocks honest parties will have on their path far in the future.

Remark 6. A more straightforward definition of dishonest chain growth might appear to be something like the following: The length of any path between two nodes that have never been on any honest path and appeared in `FinalTree` within a time interval of length $\Delta_{\texttt{growth}}$ is bounded. The problem with that definition is that dishonest parties can grow a path just "in their heads" and then publish the whole chain at once. Hence, dishonest chains in this sense can grow arbitrarily long within a very short time. To obtain a meaningful notion, we need to estimate at what point in time dishonest parties have started growing their chains. This estimate corresponds to $\hat{\tau}_B$ in the above definition.

4 The Finality Layer

4.1 Formalization

We now formalize the properties we want from a finality layer. The finality layer is a protocol that interacts with a blockchain as described above and uses the setFinal-command. The properties correspond to restrictions on how the setFinal-command is used.

Definition 6. Let $\Delta, k \in \mathbb{N}$. We say a protocol achieves (Δ, k)-*finality* if it satisfies the following properties.

Chain-forming: If an honest party $P_i \in \texttt{Honest}$ inputs $(\textsc{setFinal}, R)$ at time τ, we have $\texttt{lastFinal}_i \in \mathrm{PathTo}(\texttt{Tree}_i^{\tau}, R)$ and $R \neq \texttt{lastFinal}_i$.

Agreement: For all $l \in \mathbb{N}$ we have that if the l-th inputs $(\textsc{setFinal}, \cdot)$ of honest P_i and P_j are $(\textsc{setFinal}, R_i)$ and $(\textsc{setFinal}, R_j)$, respectively, then $R_i = R_j$.

Δ-Updated: At any time τ, we have

$$\max_{P_i \in \texttt{Honest}} \mathrm{Depth}(\texttt{Pos}_i^{\tau}) - \min_{P_i \in \texttt{Honest}} \mathrm{Depth}(\texttt{lastFinal}_i^{\tau}) \leq \Delta.$$

***k*-Support:** If honest $P_i \in \texttt{Honest}$ inputs $(\textsc{setFinal}, R)$ at time τ, there are at least k honest parties $P_j \in \texttt{Honest}$ and times $\tau_j \leq \tau$ such that $R \in \texttt{Path}_j^{\tau_j}$.

The chain-forming property guarantees that all finalized blocks are descendants of previously finalized blocks. That is, the finalized blocks form a chain and in particular, there are no forks. Agreement further guarantees that all honest parties agree on the same finalized blocks. This means that all ancestors of the last finalized block can be trusted to never disappear from the final chain of any honest party. The updated property ensures that the final chain grows roughly at the same speed as the underlying blockchain. This also implies liveness of the finalization protocol if the underlying blockchain keeps growing, in the sense that all honest parties will keep finalizing new blocks.

The property k-support finally ensures that whenever a block becomes finalized, at least k parties had this block on their path at some point. The smaller k is, the more honest parties need to "jump" to a new position under the next finalized block, which can cause rollbacks. We want to guarantee that *at least* $k \geq 1$ because otherwise we finalize blocks that are not supported by any honest party, what would inevitably lead to bad chain quality.

In Section 7 we show that our finalization protocol from Section 5 satisfies the above properties.

On long range attacks. In a long-range attack on a proof-of-stake blockchain, an attacker can in several plausible situations, given enough time, grow a deeper alternative chain from far back in time that overtakes the real one.[15] To prevent long-range attacks, many existing proof-of-stake protocols use some form of checkpointing, which prevents honest parties from adopting such alternative chains.[15] For example, Ouroboros [19] and Ouroboros Praos [11] use a chain-selection rule that selects the longest chain that does not fork from the current chain more than some parameter k blocks. The rule ensures that everything more than k blocks ago is final and prevents long-range attacks. The parameter k needs to be chosen such that $\mathrm{Prefix}(k)$ holds, which can be problematic in practice since a correct bound on the common prefix needs to be known. If a finality layer such as Afgjort is added to the blockchain, this finality provides checkpointing, which is then not needed anymore in the underlying blockchain. Therefore, one can use simpler chain selection rules, such as choosing the longest chain. For this to be secure, we need that the time required to finalize the next block is shorter than the time needed to mount a successful long-range attack. To put this into perspective, the analysis by Gaži et al. [15] of a hypothetical proof-of-stake blockchain suggests that, e.g., an attacker with 0.3 relative stake needs more than 5 years for the attack considered there.

4.2 On Proving UC Security

We here discuss briefly how to model security in the UC framework and how the proof that the finality layer has the desired properties translates into a UC proof. The reader not familiar with the UC model or not interested in how to translate the property based proof into a UC proof can safely skip this section.

Ideal functionality. To model UC security, we introduce the ideal functionality $\mathcal{F}^{\Delta,k}_{\textsc{FinTree}}$ with parameters $\Delta, k \in \mathbb{N}$. Roughly speaking, it is a variant of $\mathcal{F}_{\textsc{Tree}}$ making sure that finalizations respect all desired properties of a finality layer. Compared to the functionality $\mathcal{F}_{\textsc{Tree}}$, finalized blocks are not set by the parties, but by the adversary. The ideal functionality then makes sure the adversary grows the tree and finalizes blocks such that the properties corresponding to chain-forming, agreement, Δ-updated, and k-support hold. Note that since the adversary must respect all desired properties, giving it control over finalization is not a weakness of the functionality. More technical differences to $\mathcal{F}_{\textsc{Tree}}$ are that we drop the payload data of blocks, which was used only for implementation purposes of the finality layer, and to enforce the agreement property, $\mathcal{F}^{\Delta,k}_{\textsc{FinTree}}$ internally keeps track of all previously finalized blocks. We formally define the ideal functionality as follows.

Functionality $\mathcal{F}_{\text{FinTree}}^{\Delta,k}$

Initialization

for $P_i \in \mathcal{P}$ **do**
  $\texttt{Tree}_i \coloneqq ((V_i \coloneqq \{G\}, E_i \coloneqq \varnothing), r_i \coloneqq G)$
  $\texttt{Pos}_i \coloneqq G$, $\texttt{lastFinal}_i \coloneqq G$
  $\texttt{numFinal}_i \coloneqq 1$, $\texttt{final}_i^{\texttt{numFinal}_i} \coloneqq G$
end for

Interface for party $P_i \in \mathcal{P}$

Input: GETTREE
  return copy of $(\texttt{Tree}_i, \texttt{Pos}_i, \texttt{lastFinal}_i)$ to P_i

Interface for adversary

Input: $(\text{ADDNODE}, P_i, B, p)$ // add B as child of p in $\texttt{Tree}_i$
  if $B \notin V_i$ **and** $\texttt{HonestTree}$ remains tree after adding Bas child of p in $\texttt{Tree}_i$
    and $\forall P_j \in \texttt{Honest}$ $(\text{Depth}(p) + 1 \leq \text{Depth}(\texttt{lastFinal}_j) + \Delta)$ **then** // Δ-updated
    $V_i \coloneqq V_i \cup \{B\}$
    $E_i \coloneqq E_i \cup \{\{p, B\}\}$
    if $p = \texttt{Pos}_i$ **then**
      $\texttt{Pos}_i \coloneqq B$
    end if
  end if

Input: $(\text{SETPOSITION}, P_i, B)$ // set position of P_i to B
  if B is a leaf of $\texttt{FinalTree}_i$ **then**
    $\texttt{Pos}_i \coloneqq B$
  end if

Input: $(\text{DECLAREFINAL}, P_i, R)$ // declare block R as final for P_i
  if $\texttt{lastFinal}_i \in \text{PathTo}(\texttt{Tree}_i, R)$ **and** $R \neq \texttt{lastFinal}_i$ // chain-forming
    and $\forall P_j \in \texttt{Honest}(\texttt{numFinal}_j > \texttt{numFinal}_i \rightarrow R = \texttt{final}_j^{\texttt{numFinal}_i+1})$ // agreement
    and R has been on $\texttt{Path}_j$ for at least k honest parties P_j **then** // k-support
    $\texttt{numFinal}_i \coloneqq \texttt{numFinal}_i + 1$
    $\texttt{lastFinal}_i \coloneqq \texttt{final}_i^{\texttt{numFinal}_i} \coloneqq R$
  end if

Network model and assumed hybrids. We want to implement $\mathcal{F}_{\text{FinTree}}^{\Delta,k}$ on top of $\mathcal{F}_{\text{Tree}}$, i.e., we assume the protocol has access to $\mathcal{F}_{\text{Tree}}$, which means in UC terminology that $\mathcal{F}_{\text{Tree}}$ is assumed as a *hybrid*. To model our assumptions on the partially synchronous network, we further assume a clock functionality and a network functionality $\mathcal{F}_{\text{Net}}^{\Delta_{\texttt{net}}}$, which provides the guarantees discussed in Section 2.1, where $\Delta_{\texttt{net}}$ is a parameter polynomial in the security parameter. We then require from a UC protocol $\Pi_{\texttt{Fin}}$ given the required hybrids to UC securely realize $\mathcal{F}_{\text{FinTree}}^{\Delta,k}$ for all $\Delta_{\texttt{net}}$, where $\Delta_{\texttt{net}}$ is not given to the protocol and Δ and k can depend on $\Delta_{\texttt{net}}$.

The finality layer we present in this paper further makes use of signatures and a lottery that can be implemented by a VRF (cf. Section 6.3). It is therefore convenient to also model signatures and VRFs as hybrid functionalities. Modeling all these hybrids involves several subtleties and is beyond the scope of this paper. Hybrids with similar guarantees have been modeled, e.g., in [3] and we refer the reader to that paper for more details.

Constructing a UC protocol and simulator. Given a finality layer and the hybrids discussed above, it is straightforward to construct a protocol $\Pi_{\texttt{Fin}}$ implementing $\mathcal{F}^{\Delta,k}_{\text{FinTree}}$: Simply run the finality layer on top of $\mathcal{F}_{\text{Tree}}$ and forward GETTREE request and the respective answers.

Notice that in $\mathcal{F}^{\Delta,k}_{\text{FinTree}}$ there are no inputs to honest parties that are kept secret from the adversary/simulator. We can therefore construct a UC simulator by running the protocol $\Pi_{\texttt{Fin}}$ on the real inputs for all parties (including the honest ones). The simulator updates the variables $\texttt{Tree}_i, \texttt{lastFinal}_i, \texttt{Pos}_i$ in $\mathcal{F}^{\Delta,k}_{\text{FinTree}}$ to have exactly the values they have in the simulated execution of the protocol. This gives a *perfect* simulation as long as $\mathcal{F}^{\Delta,k}_{\text{FinTree}}$ allows the simulator to update $\texttt{Tree}_i, \texttt{lastFinal}_i, \texttt{Pos}_i$ as needed. It can be seen that $\mathcal{F}^{\Delta,k}_{\text{FinTree}}$ allows the simulator to to do so exactly as long as the finality layer has the properties chain-forming, agreement, Δ-updated, and k-support.

4.3 Impossibility of Better Bounds for the Number of Corruptions

We next show that our protocol is optimal in its corruption bound, and that the hope for a $t \geq n/3$ partially synchronous finality layer is void.

Theorem 1. *A partially synchronous finality layer for n parties with Agreement and Updated must have $t < n/3$.*

Proof. Assume for contradiction that we have a partially synchronous finality layer for n parties which tolerates that $t = n/3$. Assume that it has Agreement and Δ-Updared for some Δ.

We divide the set of parties into three set P_1, P_2, and P_3, each of size t.

Let $\Delta_{\texttt{Fast}}$ be some fixed bound on all eventual bounds in the model. In particular, the model will deliver all blocks and messages before time $\Delta_{\texttt{Fast}}$.

For $e = 1, 2$ consider the following experiment E_e: We run only $P_e \cup P_3$. We deliver all blocks and messages before time $\Delta_{\texttt{Fast}}$. We grow a tree which is just a long chain of empty blocks $B^e_0, B^e_1, B^e_2, B^e_3, \ldots$ where B^e_0 is the genesis block. We add a new block every $\Delta_{\texttt{Fast}}$ seconds. Since the finality layer is Δ-updated for some Δ, it will eventually finalize some block $B^e_{F_e}$ for $F_e > 0$. Let T^e be an upper bound such that the protocol finalizes a block before time T^e with probability at least $2/3$.

Let $\Delta_{\texttt{Slow}} = T^1 + T^2 + 1$. Consider the following experiment $\hat{E}^e$. We set all the eventuality bounds of the model to be $\Delta_{\texttt{Slow}}$. Yet, we still deliver all messages and blocks before time $\Delta_{\texttt{Fast}}$. In $\hat{E}^e$ the protocol finalizes a block before time T^e with probability at least $2/3$. This reason is that E^e and $\hat{E}^e$ are identical to the protocol as in $\hat{E}^e$ we still deliver all message and blocks before $\Delta_{\texttt{Slow}}$, and in the partially synchronous model the parties do not see the eventuality bound.

Consider the following experiment E. We set all the eventuality bounds of the model to be $\Delta_{\texttt{Slow}}$. We make two copies of the parties in P_3. Call them P^1_3 and P^2_3. We run $P_1 \cup P^1_3$ together and we run $P_2 \cup P^2_3$. We set the eventuality bound to $\Delta_{\texttt{Slow}}$. We grow a tree with two branches $B_0, B^1_1, B^1_2, B^1_3, \ldots$ and $B_0, B^2_1, B^2_2, B^2_3, \ldots$, where B_0 is genesis. We add a new block to each chain every $\Delta_{\texttt{Fast}}$ seconds. During the first $\Delta_{\texttt{Slow}}$ seconds we only show $B_0, B^1_1, B^1_2, B^1_3, \ldots$ to $P_1 \cup P^1_3$ and we only show $B_0, B^2_1, B^2_2, B^2_3, \ldots$ to $P_2 \cup P^1_3$. Furthermore, during the first $\Delta_{\texttt{Slow}}$ we propagate no messages between parties in $P_1 \cup P^1_3$ and parties in $P_2 \cup P^1_3$. But inside each group $P_e \cup P^1_3$ we still deliver all messages and blocks before time $\Delta_{\texttt{Fast}}$. Note that $P_e \cup P^1_3$ has exactly the same view as in $\hat{E}^e$. So by time $\Delta_{\texttt{Slow}}$ the parties finalizes a block $B^e_{F_e}$ with probability at least 2/3. So with probability at least 1/3 (by a union bound), by time $\Delta_{\texttt{Slow}}$ the protocol finalized two blocks $B^1_{F_1}$ and $B^2_{F_2}$ with $F_1 > 0$ and $F_2 > 0$. These block are different. Since the experiment is consistent with a run of the model wtih P_3 being Byzantine corrupted and the eventuality bound being $\Delta_{\texttt{Slow}}$, this violated Agreement. □

5 Afgjort Protocol

In this section we describe our finality protocol. The protocol consists of a collection of algorithms that interacts with each other making finalization possible. In the main routine FinalizationLoop, parties regularly try to finalize new blocks by invoking the Finalization algorithm.

The goal of Finalization is to make all the honest parties agree on a common node R at depth d of their own local trees. This finalization happens with a "delay" of γ blocks, i.e., honest parties will only start the agreement process once their $\texttt{Path}_i$ has length at least $d + \gamma$. If the honest parties successfully agree on a block R, they will finalize it by re-rooting their own local tree for the new root R. If no agreement is achieved the parties increase the finalization delay γ and re-run the agreement protocol with the new delay; this process repeats until an agreement is met. The idea is that once γ is large enough, there will be only one candidate for a final block at depth d, which will then successfully be agreed on.

Justifications. We introduce the concept of *justifications*. A justification J is a predicate which takes as input a value v and the local state of a party (in particular its tree). We say that the value v is J-justified for party P_i if the predicate evaluates to true for v and P_i's state.

Definition 7. For a value v that can be sent or received, a *justification* is a predicate J which can be applied to v and the local state of a party. Justifications are monotone with respect to time, i.e, if J is true for a value v at party P at time τ, then J is true (at that party) any time $\geq \tau$.

An example is the following justification $J^{d,\gamma}_{\textsc{InTree}}$ where the value v is a block.

Definition 8. A block B is $J^{d,\gamma}_{\textsc{InTree}}$-justified for party P_i if B is at depth d of a path of length at least $d+\gamma$ in $\texttt{FinalTree}_i$.

We call such justification *eventual*, in the sense that if a block is $J^{d,\gamma}_{\textsc{InTree}}$-justified for a honest party P_i, then it will be eventually $J^{d,\gamma}_{\textsc{InTree}}$-justified for any other honest party. This is a direct consequence of tree propagation.

Definition 9. A justification J is an *eventual justification* if for any value v and parties P_i and P_j the following holds. If v becomes justified for party P_i at time τ and both P_i and P_j from that point in time are live and honest, then eventually v becomes justified for party P_j.

Keeping up with the tree growth. After a block at some depth d has successfully been finalized, one needs to choose the next depth d' for finalization. For the updated property, this new depth should ideally be chosen such that $d' - d$ corresponds to how long the chain grows during one finalization round. In case this value was set too small before, we need to temporarily increase it to catch up with the chain growth. In the finalization protocol, parties use the subroutine NextFinalizationGap, which returns an estimate ℓ, and set the next depth to $d' = d + \ell$. We discuss this procedure in Section 5.1.

Finalization witnesses. After a successful finalization, parties use PROPDATA to add a *finalization witness* W to the blockchain. A finalization witness has the property that whenever a valid witness for some R exists, then R indeed has been finalized. In our protocols, such a witness consists of $t+1$ signatures on the outcome of the finalization. We put such witnesses on the blockchain for two reasons: First, it allows everyone (including parties not on the finalization committee) to verify which blocks have been finalized. Secondly, we use the witnesses for computing the next finalization gap (see Section 5.1).

Finalization. The finalization loop algorithm FinalizationLoop is used to periodically invoke the finalization procedure to finalize blocks at increasing depths.

Protocol FinalizationLoop(sid)

Party P_i does the following:

Set $\gamma := 1$, $d := 6$, and $\ell := 5$
for $\texttt{ctr} = 1, 2, 3, \ldots$ **do**
    Set $\texttt{faid} := (\texttt{sid}, \texttt{ctr})$
    Run $(R, \texttt{W}, \gamma') := \mathsf{Finalization}(\texttt{faid}, J^{d,\gamma}_{\text{InTree}}, d, \gamma)$
    Invoke (SETFINAL, R)
    Invoke (PROPDATA, W)
    Set $\ell := \mathsf{NextFinalizationGap}(\texttt{lastFinal}_i, \ell)$
    Set $d := d + \ell$
    Set $\gamma := \lceil 0.8 \cdot \gamma' \rceil$
end for

The basic building block of our finality protocol is the algorithm Finalization which is used to agree on a final block for depth d. The algorithm takes as inputs a unique id faid, a depth d, and an integer $\gamma \geq 1$ corresponding to number of blocks that need to occur under the block that is attempted to be finalized. If there is no agreement on a final block, γ is doubled and the parties try again. Once the parties have agreed on a block R, the algorithm outputs R and the value γ. The finalization loop then again reduces γ by multiplying it with 0.8 so that over time, a good value for γ is found. The factor 0.8 is not significant and only used for simplicity here. In practice, one can use some heuristics to optimize efficiency.

Protocol Finalization(faid, $J^{d,\gamma}_{\text{InTree}}, d, \gamma$)

Party P_i does the following:

repeat
    Set $\texttt{baid} := (\texttt{faid}, \gamma)$
    Wait until $\texttt{lastFinal}_i$ is on $\texttt{Path}_i$ and $\texttt{Path}_i$ has length $\geq d + \gamma$
    Let B_d be the block at depth d on $\texttt{Path}_i$
    Run $(R, \texttt{W}) := \mathsf{WMVBA}(\texttt{baid}, J^{d,\gamma}_{\text{InTree}})$ with input B_d
    if $R = \perp$ **then** set $\gamma := 2\gamma$ **end if**
until $R \neq \perp$
Output $(R, \texttt{W}, \gamma)$

The Finalization algorithm relies on a weak multi-valued Byzantine agreement protocol, that we call WMVBA. We discuss the general idea of the WMVBA protocol next, and we defer a more detailed treatment to Section 6.

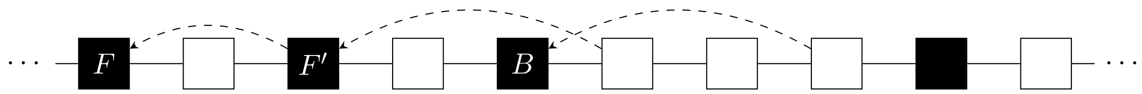

Figure 2: Computing the finalization gap. Finalized blocks are drawn in solid black, a dashed arrow from X to Y indicates that X contains a finalization witness for Y. Block F' contains a finalization witness for F, but no block up to block B contains a witness for F'. Since $F \neq F'$, the gap ℓ is increased from $\ell = 2$ to $\ell = 4$.

WMVBA. The input to the WMVBA protocol are proposals in the form of blocks; we require all proposals in WMVBA to be $J^{d,\gamma}_{\text{InTree}}$ justified, i.e., the block proposal must be in the tree of honest parties at depth d and height γ. This prevents the corrupted parties from proposing arbitrary blocks. By the design of the Finalization protocol, where γ is doubled between the calls to WMVBA it will quickly happen that all honest parties agree on the block B at the depth where we try to finalize. Furthermore, by the sustainable prefix property it will also happen that no other block is $J^{d,\gamma}_{\text{InTree}}$-justified. This moment where B is the only valid proposal is a sweet spot for agreement as we have pre-agreement. However, the sweet spot is temporary; if enough time passes, the corrupted parties could grow a long enough alternative chain which would make another proposal legitimate. We therefore want to quickly exploit the sweet spot.

For $n > 3t$ we construct in Section 6 a WMVBA protocol which consists of two subprotocols called Freeze and AƎBA. First the subprotocol Freeze is used to boil down the agreement problem to a choice between either at most one block B or the decision that there was no pre-agreement. The output of Freeze is a block or $\perp$ and is again justified by some justification. After Freeze terminates one of two will happen: If there was a pre-agreement (as is in the case of the sweet spot), then all parties decided on the same block B. However, if there was no pre-agreement, it might be the case that some parties have decided on a block B while others have decided on $\perp$. WMVBA therefore uses the binary Byzantine agreement protocol AƎBA which decides which of the two cases happened. Given the decision of AƎBA, parties can then either output the agreed block or output $\perp$ to signal disagreement.

5.1 Computing the Next Finalization Gap

To measure whether the finalization falls behind, we use the following approach: When a block B is finalized, let F be the deepest node for which a finalization witness exists in the path to B, and let F' be the deepest ancestor of B that has been finalized. If the chain does not grow too fast, we should get $F = F'$. However, if finalization is falling behind the chain a lot, B has been added to the tree before F' was finalized, in which case we have $F \neq F'$. We use this observation to adjust the gap between finalized blocks: If $F \neq F'$, we increase it, otherwise we slightly decrease it. See Figure 2 for a visualization. Below is a formal description of the procedure.

Protocol NextFinalizationGap(B, ℓ)

Let F be the deepest node in `HonestTree` for which a valid finalization witness exists on PathTo(`HonestTree`, B) (let $F := G$ if this does not exist)
if $\text{Depth}(B) - \text{Depth}(F) = \ell$ **then**
    Output $\lceil 0.8 \cdot \ell \rceil$
else
    Output $2 \cdot \ell$
end if

The values 0.8 and 2 are again somewhat arbitrary and can in practice be optimized for better results.

We next show that $\mathsf{NextFinalizationGap}$ increases ℓ if and only if the depth of the next block to be finalized is deeper than the deepest current block (plus a certain margin). This means that eventually $\mathsf{NextFinalizationGap}$ will have adjusted ℓ such that after finalizing a block, the depth of the next block to be finalized is set to a value close to the deepest position of an honest party. This will help us obtain the updated property. The proof requires that the underlying blockchain has bounded path growth and some proposal quality. Furthermore, we have to assume that the finality layer has at least 1-support, which we prove later for our protocol.

Lemma 1. *Assume the underlying blockchain satisfies bounded path growth with parameters $\Delta_{\texttt{pgrowth}}$ and $\rho_{\texttt{pgrowth}}$, and proposal quality with parameter $\ell_{\texttt{PQ}}$. Further assume the finality layer has k-support for $k \geq 1$. Let $B \in \texttt{HonestTree}$ be a block that gets finalized at time τ (i.e., τ is the first time when a party holds a finalization witness for B), let $d := \mathrm{Depth}(B)$, and let ℓ be the result of $\mathsf{NextFinalizationGap}$ for B. That is, the next finalized block B' will be at depth $d' := d + \ell$. Further let $\hat{B}$ be the first block in $\mathrm{PathTo}(\texttt{HonestTree}, B')$ that was added to $\texttt{HonestTree}$ after time τ, and let $\hat{d}$ be its depth. If $d' < \hat{d}$, then $\mathsf{NextFinalizationGap}(B', \ell)$ will output $\ell' = 2 \cdot \ell$. If $d' > \hat{d} + \rho_{\texttt{pgrowth}} \cdot \Delta_{\texttt{net}} + \Delta_{\texttt{pgrowth}} + \ell_{\texttt{PQ}}$, then $\mathsf{NextFinalizationGap}(B', \ell)$ will output $\ell' = \lceil 0.8 \cdot \ell \rceil$.*

Proof. First assume $d' < \hat{d}$. In this case, no party holds a finalization witness for B when B' or any of its ancestors are added to the tree. Thus, the deepest node F for which a valid finalization witness exists on $\mathrm{PathTo}(\texttt{HonestTree}, B')$ will not be B. Hence, $\mathrm{Depth}(B') - \mathrm{Depth}(F) > \mathrm{Depth}(B') - \mathrm{Depth}(B) = \ell$ and therefore $\mathsf{NextFinalizationGap}(B', \ell)$ outputs $2 \cdot \ell$ in that case.

Now assume $d' > \hat{d} + \rho_{\texttt{pgrowth}} \cdot \Delta_{\texttt{net}} + \Delta_{\texttt{pgrowth}} + \ell_{\texttt{PQ}}$. Let $\tilde{B}$ be the deepest block in $\mathrm{PathTo}(\texttt{HonestTree}, B')$ at time $\tau + \Delta_{\texttt{net}}$. Since we assume the finality layer has 1-support, B' was at some point on the path of an honest party. We can therefore apply the bounded-path-growth property: If the distance between $\hat{B}$ and $\tilde{B}$ is at least $\Delta_{\texttt{pgrowth}}$, then

$$\mathrm{Depth}(\tilde{B}) - \mathrm{Depth}(\hat{B}) \leq \rho_{\texttt{pgrowth}} \cdot (\tau(\tilde{B}) - \tau(\hat{B})) \leq \rho_{\texttt{pgrowth}} \cdot (\tau + \Delta_{\texttt{net}} - \tau).$$

Hence, $\mathrm{Depth}(\tilde{B}) \leq \hat{d} + \rho_{\texttt{pgrowth}} \cdot \Delta_{\texttt{net}} + \Delta_{\texttt{pgrowth}} =: d_0$. By definition of $\tilde{B}$, this means that all blocks in $\mathrm{PathTo}(\texttt{HonestTree}, B')$ deeper than depth d_0 are generated after time $\tau + \Delta_{\texttt{net}}$. Since $\Delta_{\texttt{net}}$ is an upper bound on the network delay, all honest parties hold a finalization witness for B from this time on. Proposal quality therefore implies that one of the next $\ell_{\texttt{PQ}}$ blocks will contain a finalization witness for B. Since B' has depth $d' > d_0 + \ell_{\texttt{PQ}}$, we can conclude that $\mathsf{NextFinalizationGap}(B', \ell)$ outputs $\lceil 0.8 \cdot \ell \rceil$ in that case. □

5.2 Existence of Unique Justified Proposals

For our more efficient finalization protocol to succeed, we need that there will be a unique justified proposal at some point such that all honest parties will agree on that. More precisely, we need for every depth d we want to finalize, for all time intervals δ_{freeze} required to run Freeze, for all times τ at which we start to finalize a block, and for all sufficiently large γ, there is a time $\tau_0 \geq \tau$ at which Freeze will succeed, i.e., in the time interval of length δ_{freeze} starting at τ_0, there is exactly one block at depth d that has height at least γ, and all honest parties will have that block on their path. We give a precise formalization below.

Definition 10. We say that UJP holds if there exists a polynomial $\gamma_0(d, \delta_{\text{freeze}}, \tau)$ such that the following conditions are satisfied for all $d, \tau, \delta_{\text{freeze}} \in \mathbb{N}$, and for all $\gamma \geq \gamma_0(d, \delta_{\text{freeze}}, \tau)$:

1. There exists a time $\tau_0 \geq \tau$ such that there is an honest party $P_i \in \mathtt{Honest}$ and $B \in \mathtt{Path}_i^{\tau_0}$ with $\mathrm{Depth}(B) = d$ and $\mathrm{Height}(B) \geq \gamma$.

2. For the smallest τ_0 satisfying the first condition and for all $\tau' \in [\tau_0, \tau_0 + \delta_{\text{freeze}}]$, there is only one $B' \in \mathtt{FinalTree}^{\tau'}$ with $\mathrm{Depth}(B') = d$ and $\mathrm{Height}(B') \geq \gamma$ (namely $B' = B$).

3. For all $\tau' \in [\tau_0, \tau_0 + \delta_{\text{freeze}}]$ and for all $P_j \in \mathtt{Honest}$, we have $B \in \mathtt{Path}_j^{\tau'}$.

Proving the existence of unique justified proposals. We finally show that the property from Definition 10 is implied by dishonest chain growth together with standard assumptions on the underlying blockchain.

Lemma 2. *Assume* $\mathrm{Prefix}(\xi)$ *holds for some* $\xi > 0$*, and* $\mathrm{ChainGrowth}(\Delta_{\mathtt{growth}}, \rho_{\mathtt{growth}}, \rho'_{\mathtt{growth}})$ *as well as* $\mathrm{DCGrowth}(\Delta_{\mathtt{growth}}, \rho_{\mathtt{disgro}})$ *hold for some* $\Delta_{\mathtt{growth}} \in \mathbb{N}$*,* $\rho_{\mathtt{growth}} > 0$*,* $\rho'_{\mathtt{growth}}$*, and* $\rho_{\mathtt{disgro}} < \rho_{\mathtt{growth}}$*. Then,* UJP *holds.*

Proof. Let d, δ_{freeze}, and $\tau \in \mathbb{N}$ be arbitrary. Let $\bar{d} := \max_{B \in \mathtt{FinalTree}^{\tau}} \mathrm{Depth}(B)$ the maximal depth of any block at time τ. We then define

$$\gamma_0 := \max\left\{\xi + \frac{\rho_{\mathtt{disgro}}\big(\rho_{\mathtt{growth}}(\delta_{\text{freeze}} + \Delta_{\mathtt{growth}}) + \rho'_{\mathtt{growth}} \cdot \Delta_{\mathtt{growth}}\big)}{\rho_{\mathtt{growth}} - \rho_{\mathtt{disgro}}}, \bar{d} + 1 - d\right\}.$$

Note that $\gamma_0 > \bar{d} - d$ and $\gamma_0 \geq \xi$ because $\rho_{\mathtt{disgro}} < \rho_{\mathtt{growth}}$. Now let $\gamma \geq \gamma_0$ and let τ_0 be the smallest time for which there exists some $P_i \in \mathtt{Honest}$ such that $\mathrm{Depth}(\mathtt{Pos}_i^{\tau_0}) \geq d + \gamma$. Note that this exists because we assume positive chain growth. Let B be the node on $\mathtt{Path}_i$ at depth d. By the choice of γ_0, we have $\mathrm{Depth}(\mathtt{Pos}_i^{\tau_0}) \geq d + \gamma > \bar{d}$, and thus, $\tau_0 > \tau$. Hence, condition 1 of UJP holds.

We first show that all honest parties have B on their path during the time interval $[\tau_0, \tau_0 + \delta_{\text{freeze}}]$. Let $\tau' \in [\tau_0, \tau_0 + \delta_{\text{freeze}}]$ and $P_j \in \mathtt{Honest}$. We have by $\mathrm{Prefix}(\xi)$ that $\big(\mathtt{Path}_i^{\tau_0}\big)^{\lceil \xi} \preceq \mathtt{Path}_j^{\tau'}$. Since $\mathrm{Depth}(\mathtt{Pos}_i^{\tau_0}) \geq d + \gamma \geq d + \xi$, we have that $B \in \big(\mathtt{Path}_i^{\tau_0}\big)^{\lceil \xi}$ and thus, $B \in \mathtt{Path}_j^{\tau'}$. This proves condition 3 of UJP.

Let $\tau' \in [\tau_0, \tau_0 + \delta_{\text{freeze}}]$ and let $B' \in \mathtt{FinalTree}^{\tau'}$ be an arbitrary block that is not a descendant of B (in particular, $B' \neq B$). Let $\hat{B}'$ be the deepest ancestor of B' that has at some point (until τ') been on an honest path, and let $\hat{\tau}_{B'}$ be the first time $\hat{B}'$ appeared on an honest path. Let $\hat{d}' := \mathrm{Depth}(\hat{B}')$. We claim that

$$\hat{d}' < d + \xi.$$

To prove this, note that at some time until τ', $\mathrm{PathTo}\big(\mathtt{FinalTree}^{\tau'}, \hat{B}'\big)$ was a prefix of $\mathtt{Path}_k$ for some honest $P_k \in \mathtt{Honest}$. Hence, $\mathrm{Prefix}(\xi)$ implies that $\mathrm{PathTo}\big(\mathtt{FinalTree}^{\tau'}, \hat{B}'\big)^{\lceil \xi}$ is a prefix of $\mathtt{Path}_j^{\tau'}$ for all $P_j \in \mathtt{Honest}$. As we have shown above, $B \in \mathtt{Path}_j^{\tau'}$. Thus, we either have $B \in \mathrm{PathTo}\big(\mathtt{FinalTree}^{\tau'}, \hat{B}'\big)^{\lceil \xi}$ or $\mathrm{PathTo}\big(\mathtt{FinalTree}^{\tau'}, \hat{B}'\big)^{\lceil \xi}$ is a prefix of $\mathrm{PathTo}\big(\mathtt{FinalTree}^{\tau'}, B\big)$. Because B' is a descendant of $\hat{B}'$ and we assume that B' is not a descendant of B, the former is impossible. In the latter case, we have $\hat{d}' < d + \xi$ as claimed.

We next want to bound $\tau_0 - \hat{\tau}_{B'}$. We assume this value is positive, otherwise we obtain the bound $\tau_0 - \hat{\tau}_{B'} \leq 0$. At time $\hat{\tau}_{B'}$, some honest party had a position with depth at least $\hat{d}'$. By definition of τ_0, all honest parties at time $\tau_0 - 1$ have positions with depth less than $d + \gamma$. Hence, $\mathrm{ChainGrowth}(\Delta_{\mathtt{growth}}, \rho_{\mathtt{growth}}, \rho'_{\mathtt{growth}})$ implies that all honest parties at time τ_0 have positions with depth less than $d + \gamma + \rho'_{\mathtt{growth}} \cdot \Delta_{\mathtt{growth}}$. This means that between times τ_0 and $\hat{\tau}_{B'}$, the depth of the position of some honest party has grown by at most $d + \gamma + \rho'_{\mathtt{growth}} \cdot \Delta_{\mathtt{growth}} - \hat{d}'$.

Note that this value is positive since $\gamma \geq \xi$ and $\hat{d}' < d + \xi$. The number of time intervals of length $\Delta_{\texttt{growth}}$ that fit into $[\hat{\tau}_{B'}, \tau_0]$ equals

$$\left\lfloor \frac{\tau_0 - \hat{\tau}_{B'}}{\Delta_{\texttt{growth}}} \right\rfloor \geq \frac{\tau_0 - \hat{\tau}_{B'}}{\Delta_{\texttt{growth}}} - 1.$$

Using the upper bound on chain growth, this implies

$$(\tau_0 - \hat{\tau}_{B'} - \Delta_{\texttt{growth}}) \cdot \rho_{\texttt{growth}} \leq \left\lfloor \frac{\tau_0 - \hat{\tau}_{B'}}{\Delta_{\texttt{growth}}} \right\rfloor \cdot \Delta_{\texttt{growth}} \cdot \rho_{\texttt{growth}} \leq d + \gamma + \rho'_{\texttt{growth}} \cdot \Delta_{\texttt{growth}} - \hat{d}'.$$

Hence, we obtain

$$\tau_0 - \hat{\tau}_{B'} \leq \Delta_{\texttt{growth}} + \frac{d + \gamma + \rho'_{\texttt{growth}} \cdot \Delta_{\texttt{growth}} - \hat{d}'}{\rho_{\texttt{growth}}}.$$

We finally want to bound $\mathrm{Depth}(B')$. Using $\mathrm{DCGrowth}(\Delta_{\texttt{growth}}, \rho_{\texttt{disgro}})$, we obtain

$$\begin{aligned}
\mathrm{Depth}(B') - \mathrm{Depth}(\hat{B}') &\leq \rho_{\texttt{disgro}} \cdot \max\{\Delta_{\texttt{growth}}, \tau' - \hat{\tau}_{B'}\} \\
&\leq \rho_{\texttt{disgro}} \cdot \max\{\Delta_{\texttt{growth}}, \tau_0 + \delta_{\text{freeze}} - \hat{\tau}_{B'}\} \\
&\leq \rho_{\texttt{disgro}} \cdot \left(\Delta_{\texttt{growth}} + \frac{d + \gamma + \rho'_{\texttt{growth}} \cdot \Delta_{\texttt{growth}} - \hat{d}'}{\rho_{\texttt{growth}}} + \delta_{\text{freeze}}\right).
\end{aligned}$$

Thus,

$$\mathrm{Depth}(B') \leq \hat{d}' \cdot \left(1 - \frac{\rho_{\texttt{disgro}}}{\rho_{\texttt{growth}}}\right) + \rho_{\texttt{disgro}} \cdot \left(\Delta_{\texttt{growth}} + \frac{d + \gamma + \rho'_{\texttt{growth}} \cdot \Delta_{\texttt{growth}}}{\rho_{\texttt{growth}}} + \delta_{\text{freeze}}\right).$$

Since $\rho_{\texttt{disgro}} < \rho_{\texttt{growth}}$, we have $0 < 1 - \frac{\rho_{\texttt{disgro}}}{\rho_{\texttt{growth}}} \leq 1$. Further using $\hat{d}' < d + \xi$, this implies

$$\begin{aligned}
\mathrm{Depth}(B') &< (d + \xi) \cdot \left(1 - \frac{\rho_{\texttt{disgro}}}{\rho_{\texttt{growth}}}\right) \\
&\quad + \rho_{\texttt{disgro}} \cdot \left(\Delta_{\texttt{growth}} + \frac{d + \gamma + \rho'_{\texttt{growth}} \cdot \Delta_{\texttt{growth}}}{\rho_{\texttt{growth}}} + \delta_{\text{freeze}}\right) \\
&\leq d + \xi + \rho_{\texttt{disgro}} \cdot \left(\Delta_{\texttt{growth}} + \frac{\gamma + \rho'_{\texttt{growth}} \cdot \Delta_{\texttt{growth}}}{\rho_{\texttt{growth}}} + \delta_{\text{freeze}}\right) \\
&= d + \xi + \rho_{\texttt{disgro}} \cdot \left(\Delta_{\texttt{growth}} + \frac{\rho'_{\texttt{growth}} \cdot \Delta_{\texttt{growth}}}{\rho_{\texttt{growth}}} + \delta_{\text{freeze}}\right) + \gamma \cdot \frac{\rho_{\texttt{disgro}}}{\rho_{\texttt{growth}}}.
\end{aligned}$$

By the choice of $\gamma_0 \leq \gamma$, we have

$$\rho_{\texttt{disgro}}(\rho_{\texttt{growth}}(\delta_{\text{freeze}} + \Delta_{\texttt{growth}}) + \rho'_{\texttt{growth}} \cdot \Delta_{\texttt{growth}}) \leq (\gamma - \xi) \cdot (\rho_{\texttt{growth}} - \rho_{\texttt{disgro}}),$$

which implies

$$\rho_{\texttt{disgro}} \cdot \left(\delta_{\text{freeze}} + \Delta_{\texttt{growth}} + \frac{\rho'_{\texttt{growth}} \cdot \Delta_{\texttt{growth}}}{\rho_{\texttt{growth}}}\right) \leq (\gamma - \xi) \cdot \left(1 - \frac{\rho_{\texttt{disgro}}}{\rho_{\texttt{growth}}}\right).$$

Therefore,

$$\mathrm{Depth}(B') < d + \xi + (\gamma - \xi) \cdot \left(1 - \frac{\rho_{\texttt{disgro}}}{\rho_{\texttt{growth}}}\right) + \gamma \cdot \frac{\rho_{\texttt{disgro}}}{\rho_{\texttt{growth}}} \leq d + \gamma.$$

Since B' was an arbitrary block that is not a descendant of B, we can conclude that all blocks with depth at least $d + \gamma$ are descendants of B. This concludes the proof of condition 2 of UJP. $\square$

6 Weak Multi-Valued Byzantine Agreement

At the core of the $\mathsf{Finalization}$ algorithm from Section 5, parties use a Byzantine agreement protocol relative to a justification J (here $J = J^{d,\gamma}_{\text{InTree}}$). Each party P_i inputs a justified proposal $\mathtt{p}_i$ (a block) and gets a decision $\mathtt{d}_i$ (a block or $\bot$) as output. The Byzantine agreement must satisfy consistency and termination which are defined as follows.

Consistency: If some honest parties P_i and P_j output decisions $\mathtt{d}_i$ and $\mathtt{d}_j$ respectively, then $\mathtt{d}_i = \mathtt{d}_j$.

Termination: If all honest parties input some justified proposal, then eventually all honest parties output a decision.

$\mathsf{Finalization}$ also requires that the agreement protocol satisfies some form of validity. The exact validity guarantee we require depends on whether the underlying blockchain satisfies DCGrowth.

We therefore propose two variants of such a protocol, namely WMVBA and $\mathsf{FilteredWMVBA}$ which are inspired by classic asynchronous BA protocol such as [8] and [4]. Both protocols satisfy consistency and termination as defined above. The first variant, WMVBA requires DCGrowth from the underlying blockchain, whereas $\mathsf{FilteredWMVBA}$ only relies on standard blockchain properties.

The WMVBA protocol satisfies weak validity and $n/3$-support:

Weak Validity: If during the protocol execution there exists a decision $\mathtt{d}$ such that no other decision $\mathtt{d}'$, where $\mathtt{d}' \neq \mathtt{d}$ is J-justified for any honest party, then no honest party P_i outputs a decision $\mathtt{d}'$ with $\mathtt{d}' \neq \mathtt{d}$.

$n/3$**-Support:** If some honest party P_i outputs decision $\mathtt{d}$ with $\mathtt{d} \neq \bot$, then at least $n/3$ of the honest parties had J-justified input $\mathtt{d}$.

Remark 7. The $n/3$-support property is a strengthening of *strong validity*, which has been introduced by Neiger [26]. Strong validity requires the output of honest parties to be the input of some honest party, i.e., it roughly corresponds to 1-support (ignoring $\bot$-outputs and justifications). As was shown by Neiger [26], strong validity is impossible (even in a synchronous network) if $n \leq mt$, where m is the number of possible inputs. We circumvent this impossibility by allowing parties to output $\bot$ when there are too many possible inputs (i.e., justified proposals).

As we have shown in Lemma 2, a blockchain satisfying Prefix, ChainGrowth, and DCGrowth has the property that at some point, there is a unique justified proposal in the tree. Weak validity guarantees that running WMVBA at that point leads to parties outputting that block.

If DCGrowth does not hold, there could always be more than one justified proposal, in which case WMVBA can always output $\bot$. To deal with this, $\mathsf{FilteredWMVBA}$ provides (non-weak) validity, at the expense of only having 1-support. More precisely, $\mathsf{FilteredWMVBA}$ satisfies consistency, termination, and the following two properties:

Validity: If all honest parties input the same J-justified $\mathtt{d}$, then no honest party P_i outputs a decision $\mathtt{d}'$ with $\mathtt{d}' \neq \mathtt{d}$.

1-Support: If some honest party P_i outputs decision $\mathtt{d}$ with $\mathtt{d} \neq \bot$, then at least 1 of the honest parties had J-justified input $\mathtt{d}$.

The usual common-prefix property implies that if honest parties have more than the prefix parameter number of blocks below the block they propose to finalize, then they all propose the same block. Hence, validity of $\mathsf{FilteredWMVBA}$ ensures that in this case, they agree on this block. Note that without DCGrowth, it is possible to have other chains of equal length in the tree (and thus no unique justified proposal), but Prefix implies that no honest party adopts these alternative chains, i.e., only dishonest parties can input them to FilteredWMVBA.

We first present WMVBA. FilteredWMVBA is essentially the same with an additional filtering step at the beginning, with the goal of filtering out proposals of dishonest parties. In Section 6.5, we describe how WMVBA needs to be modified to obtain FilteredWMVBA.

Protocol idea. At the beginning of the WMVBA protocol all parties first run the Freeze sub-protocol. In Freeze, parties send their proposals to all other parties and every party checks whether they received at least $n - t$ proposals for the same block. In that case, their output for Freeze is that block, otherwise it is $\perp$. Freeze thereby boils the decision for a finalized block down to the binary decision between $\perp$ and a unique block output by Freeze (if that exists). To this end, a binary Byzantine agreement protocol AꓭBA is run after Freeze. We provide details about the sub-protocols and WMVBA in the following sections.

Related work. Our protocols are inspired by classic asynchronous BA protocols such as [8] and [4]. In contrast to many classical protocols, such as the one in [8], we implement a coin-flip using a VRF-based approach instead of a distributed coin-flip protocol. Also note that our protocols are not asynchronous; we explicitly make use of the partially synchronous network assumption.

The idea of reducing a multivalued Byzantine agreement to a binary Byzantine agreement as used in WMVBA (via Freeze and AꓭBA) was first proposed by Turpin and Coan [30]. They propose a synchronous multivalued Byzantine agreement where the parties try to agree whether they reached an agreement on their multivalued inputs; if there is pre-agreement (all input values are equal) then the output is the same as the input value, otherwise there is no pre-agreement and all the nodes output a predefined fixed value.

The idea of core-set selection as used in AꓭBA has been presented, e.g., in [1].

The most important difference of our protocol compared to classical ones is that classical protocols provide validity (i.e., if all honest parties have the same input v, then no honest party decides on $v' \neq v$), which is stronger than our weak validity. They do not, however, provide any support. In our setting, weak validity is sufficient and support is an important property for a finality layer. Hence, while we use mostly known techniques, we need different guarantees and cannot directly rely on existing protocols.

6.1 Freeze Protocol

Each honest party P_i has a J-justified input $\mathtt{p}_i$, called proposal. In our use case these proposals are blocks. Each honest party P_i (eventually) outputs a decision $\mathtt{d}_i$ which is either from the space of proposals (e.g., a block) or $\perp$. The output decision $\mathtt{d}_i$ of P_i is justified by justification J_{dec} (see Definition 13). The Freeze protocol satisfies the following properties.

Weak Consistency: If honest parties P_i and P_j output decisions $\mathtt{d}_i \neq \perp$ and $\mathtt{d}_j \neq \perp$ respectively, then $\mathtt{d}_i = \mathtt{d}_j$.

Weak Validity: If during the protocol execution[1] there exists a J-justified proposal $\mathtt{p}$ such that no other proposal $\mathtt{p}' \neq \mathtt{p}$ is J-justified for any honest party, then no honest party P_j outputs $\mathtt{p}'$.

[1]That is until the first honest party gets an output.

$n-2t$-Support: If honest party P_i outputs decision $\mathtt{d}_i \neq \bot$, then at least $n-2t$ honest parties had $\mathtt{d}_i$ as input.

Termination: If all honest parties input some justified proposal, then eventually all honest parties output a decision.

Next, we define the following justifications relative to the input justification J.

Definition 11. A proposal message $m = (\mathtt{baid}, \text{PROPOSAL}, \mathtt{p})$ from P_i is considered $J_{\mathtt{prop}}$-justified for P_j if m is signed by P_i and $\mathtt{p}$ is J-justified for P_j.

Definition 12. A vote message $m = (\mathtt{baid}, \text{VOTE}, \mathtt{v})$ from P_i is considered $J_{\mathtt{vote}}$-justified for P_j if it is signed by P_i and either for $\mathtt{v} \neq \bot$ P_j has collected $J_{\mathtt{prop}}$-justified messages $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{v})$ from at least $n-2t$ parties or for $\mathtt{v} = \bot$ P_j has collected $J_{\mathtt{prop}}$-justified messages $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{p})$ and $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{p}')$ (from two different parties) where $\mathtt{p}' \neq \mathtt{p}$.

Definition 13 ($J_{\mathtt{dec}}$-justification)**.** A decision message $m = (\mathtt{baid}, \text{FROZEN}, \mathtt{d})$ is $J_{\mathtt{dec}}$-justified for P_j if P_j collected $J_{\mathtt{vote}}$-justified messages $(\mathtt{baid}, \text{VOTE}, \mathtt{d})$ from at least $t+1$ parties.

Observe that for example a proposal message $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{p})$ can become $J_{\mathtt{prop}}$-justified for P_j much *after* it was received from P_i. This due to J being an eventual justification. The proposal $\mathtt{p}$ thus can become J-justified after receiving a proposal message containing $\mathtt{p}$.

Protocol. We describe the Freeze protocol next.

Protocol Freeze($\mathtt{baid}$, J)

Each (honest) party P has a J-justified proposal $\mathtt{p}$ as input. Party P does the following:
Propose:

1. Broadcast proposal message $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{p})$.

Vote:

2. Collect proposal messages $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{p}_i)$. Once $J_{\mathtt{prop}}$-justified proposal messages from at least $n-t$ parties have been collected do the following (but keep collecting proposal messages).
   (a) If $J_{\mathtt{prop}}$-justified proposal messages from at least $n-t$ parties contain the same proposal $\mathtt{p}$, broadcast vote message $(\mathtt{baid}, \text{VOTE}, \mathtt{p})$.
   (b) Otherwise broadcast vote message $(\mathtt{baid}, \text{VOTE}, \bot)$.

Freeze:

3. Collect vote messages $(\mathtt{baid}, \text{VOTE}, \mathtt{p}_i)$. Once $J_{\mathtt{vote}}$-justified vote messages from at least $n-t$ parties have been collected and there is a value contained in at least $t+1$ vote messages do the following.
   (a) If $J_{\mathtt{vote}}$-justified vote messages from at least $t+1$ parties contain the same $\mathtt{p} \neq \bot$ then output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d})$, where $\mathtt{d} = \mathtt{p}$.
   (b) Otherwise if $\bot$ is contained in vote messages from at least $t+1$ parties output $(\mathtt{baid}, \text{FROZEN}, \bot)$.

4. Keep collecting vote messages until WMVBA is terminated (i.e., until P_i gets an output in WMVBA). Party P_i keeps track of all decisions $(\texttt{baid}, \text{FROZEN}, \texttt{d})$ which become $J_{\texttt{dec}}$-justified.

Lemma 3. *For $t < \frac{n}{3}$ the protocol* Freeze *satisfies weak agreement, weak validity, $n-2t$-support, and termination. The outputs of honest parties are $J_{\texttt{dec}}$-justified.*

Proof. We prove each individual property next.

Weak Consistency: To prove the weak agreement property, we have to show that no honest parties P_i and P_j will ever output different decisions $\texttt{d}_i$ and $\texttt{d}_j$ when $\texttt{d}_i \neq \perp$ and $\texttt{d}_j \neq \perp$.

If all honest parties output $\perp$ then we are done. So assume that honest party P_i outputs $\texttt{d}_i$. Then at least one honest party P_k broadcast $J_{\texttt{vote}}$-justified message $(\texttt{baid}, \text{VOTE}, \texttt{d}_i)$. So P_k must have collected $J_{\texttt{prop}}$-justified messages $(\texttt{baid}, \text{PROPOSAL}, \texttt{d}_i)$ from at least $n-t$ parties. This implies that any other honest party has received $(\texttt{baid}, \text{PROPOSAL}, \texttt{d}_j)$ from at most $2t$ parties where $\texttt{d}_i \neq \texttt{d}_j \neq \perp$. So all honest parties will vote either for $\texttt{d}_i$ or $\perp$. Thus all honest parties will output either $\texttt{d}_i$ or $\perp$. This implies the property.

Weak Validity: Assume that there exists a proposal $\texttt{p}$ such that during the protocol execution there exist no other $\texttt{p}' \neq \texttt{p}$ that is J-justified for any honest party. Thus, the only proposal message which could be $J_{\texttt{prop}}$-justified for honest parties is $(\texttt{baid}, \text{PROPOSAL}, \texttt{p})$. This implies that the only vote message which could be $J_{\texttt{vote}}$-justified for honest parties is also $(\texttt{baid}, \text{VOTE}, \texttt{p})$. Thus, $(\texttt{baid}, \text{FROZEN}, \texttt{d})$, where $\texttt{d} = \texttt{p}$ is the only decision that could become $J_{\texttt{dec}}$-justified for any honest party.

$n-2t$-Support: Assume P_i outputs decision $\texttt{d}_i \neq \perp$. That means that P_i received $J_{\texttt{vote}}$-justified vote message $(\texttt{baid}, \text{VOTE}, \texttt{p})$ from strictly more than t parties. Out of those parties at least one must be honest. That honest must have received $J_{\texttt{prop}}$-justified $(\texttt{baid}, \text{PROPOSAL}, \texttt{d}_i)$ from at least $n-t$ parties. Thus at least $n-2t$ honest parties have sent $J_{\texttt{prop}}$-justified $(\texttt{baid}, \text{PROPOSAL}, \texttt{d}_i)$ which they only do if $\texttt{d}_i$ is their input.

Termination: Note that all used justifications are eventual. So if there exists a proposal which is J-justified for some honest party it eventually becomes J-justified for all honest parties. Thus, all honest parties will eventually send out $J_{\texttt{prop}}$-justified proposal messages and all honest parties will eventually send out $J_{\texttt{vote}}$-justified vote messages. As honest parties vote for at most two different values, all will eventually receive vote messages from $n-t$ parties where one values is contained in at least $t+1$ votes. Therefore all honest parties will eventually output a decision.

Finally, we show that the output $\texttt{d}_i$ of honest party P_i is $J_{\texttt{dec}}$-justified for P_i. If $\texttt{d}_i \neq \perp$ then P_i collected $J_{\texttt{vote}}$-justified messages $(\texttt{baid}, \text{VOTE}, \texttt{d}_i)$ from at least $t+1$ parties. Thus the output is $J_{\texttt{dec}}$-justified. If $\texttt{d}_i = \perp$ and P_i collected $J_{\texttt{vote}}$-justified messages $(\texttt{baid}, \text{VOTE}, \perp)$ from at least $t+1$ parties, then the output is also $J_{\texttt{dec}}$-justified. □

Corollary 1. *At most one decision $\texttt{d} \neq \perp$ will ever be $J_{\texttt{dec}}$-justified for any honest party.*

Proof. This follows from the argument of weak agreement. □

Lemma 4. *Assume any message received by an honest party will eventually be received by all other honest parties. If an honest party P_i outputs ($J_{\texttt{dec}}$-justified) decision $\texttt{d}_i \neq \perp$ in* Freeze*, then eventually all honest parties will accept $\texttt{d}_i$ has $J_{\texttt{dec}}$-justified.*

Proof. Assume P_i outputs ($J_{\texttt{dec}}$-justified) decision $\texttt{d}_i \neq \perp$. That means that P_i received $J_{\texttt{vote}}$-justified vote message $(\texttt{baid}, \text{VOTE}, \texttt{p})$ from strictly more than t parties. This also means that at least one honest party received $J_{\texttt{prop}}$-justified $(\texttt{baid}, \text{PROPOSAL}, \texttt{d}_i)$ from at least $n-t$ parties. This implies that $\texttt{d}_i$ is J-justified for that party.

The decision $\texttt{d}_i$ will therefore be J-justified for any other honest party. Under the assumption that any message received by an honest party will eventually be received by all other honest parties we have that any honest party will have $J_{\texttt{vote}}$-justified $(\texttt{baid}, \text{VOTE}, \texttt{p})$ vote messages from strictly more than t parties. This makes all honest parties accept $\texttt{d}_i$ as $J_{\texttt{dec}}$-justified eventually. □

6.2 Core Set Selection

The *weak core-set selection* protocol CSS is used in our binary byzantine agreement protocol $\mathsf{A}\mathsf{B}\mathsf{BA}$ (see Section 6.3) to compute a common core-set of party-value tuples. The global inputs, i.e., the pre-agreed parameters, are input justification $J_{\texttt{cssin}}$ and a delay Δ_{CSS}. Each party inputs a $J_{\texttt{cssin}}$-justified bit where $J_{\texttt{cssin}}$ is some (eventual) justification which is later defined by ABBA. Each honest party P_i (eventually) outputs a set $\texttt{Core}_i$ which contains justified tuples $(P, \texttt{b})$.

The idea with the delay Δ_{CSS} is to give honest parties more time to submit their input to the core-set. This allows to counter the effect of de-synchronization. In particular, assume that honest parties start the protocol within Δ_{st} and that the network delay is at most $\Delta_{\texttt{net}}$. Then honest parties are at most $\Delta_{st} + \Delta_{\texttt{net}}$ de-synchronized. By waiting $\Delta_{\mathsf{CSS}} > \Delta_{st} + \Delta_{\texttt{net}}$ the inputs of all honest parties will be part of the core set. The protocol has the following properties.

Common Core: The output sets of honest parties have a common core $\texttt{Core} \subseteq \bigcap_i \texttt{Core}_i$ which contains tuples $(P, \texttt{b})$ from at least $n-t$ different[2] parties.

Weak Validity: If during the protocol execution of CSS for some $\texttt{baid}$ there exists a $J_{\texttt{cssin}}$-justified $\texttt{b}$ such that no other bit $\texttt{b}'$ is $J_{\texttt{cssin}}$-justified for any honest party, then all tuples in the output set $\texttt{Core}_i$ of honest party P_i are of the form $(\cdot, \texttt{b})$.

Unique Honest Tuple: The output set $\texttt{Core}_i$ of honest party P_i contains for each honest party P_j at the tuple $(P_j, \texttt{b}_j)$ where $\texttt{b}_j$ is the input of P_j.

Termination: If all honest parties have $J_{\texttt{cssin}}$-justified input, then all honest parties will eventually terminate.

Δ_{CSS}-Waiting: If Δ_{CSS} is larger than the de-synchronization of honest parties, then output set $\texttt{Core}_i$ of honest party P_i contains tuples from all honest parties. Moreover, all honest outputs are fixed before the first honest party gives an output.

We define the following justifications relative to justification $J_{\texttt{cssin}}$.

Definition 14. A tuple $(P_i, \texttt{b}_i)$ is $J_{\texttt{tpl}}$-justified for P_j if it is correctly signed by P_i and $\texttt{b}_i$ is $J_{\texttt{cssin}}$-justified for P_j.

Definition 15. A *seen message* $(\text{SEEN}, P_k, (P_i, \texttt{b}_i))$ is $J_{\texttt{seen}}$-justified for P_j if it is correctly signed by P_k and $(P_i, \texttt{b}_i)$ is $J_{\texttt{tpl}}$-justified for P_j.

Definition 16. A *done-reporting message* $(\text{DONEREPORTING}, P_k, \texttt{iSaw}_k)$ is $J_{\texttt{done}}$-justified for P_j if it is correctly signed by P_k and for each tuple $(P_i, \texttt{b}_i) \in \texttt{iSaw}_k$ P_j has a $J_{\texttt{seen}}$-justified $(\text{SEEN}, P_k, (P_i, \texttt{b}_i))$.

We give a formal description of the protocol next.

[2] Note that $\texttt{Core}$ or any $\texttt{Core}_i$ contain multiple tuples with the same (dishonest) party.

Protocol $\mathsf{CSS}(\mathtt{baid}, J_{\mathtt{cssin}}, \Delta_{\mathsf{CSS}})$

The protocol is described from the view point of a party P_i which has $J_{\mathtt{cssin}}$-justified input bit $\mathtt{b}_i$.

Start:

- Party P_i sets flag $\mathtt{report}_i$ to $\top$. It initializes sets $\mathtt{iSaw}_i$ and $\mathtt{manySaw}_i$ to $\varnothing$. Then P_i sends its $J_{\mathtt{cssin}}$-justified input $\mathtt{b}_i$ signed to all parties.

Reporting Phase:

- Once P_i receives signed $\mathtt{b}_j$ from P_j such that $(P_j, \mathtt{b}_j)$ is $J_{\mathtt{tpl}}$-justified, P_i adds $(P_j, \mathtt{b}_j)$ to $\mathtt{iSaw}_i$ and sends signed $(\textsc{seen}, P_i, (P_j, \mathtt{b}_j))$ to all parties. Party P_i does this for each party P_j at most once.
- Once P_i received $J_{\mathtt{seen}}$-justified $(\textsc{seen}, P_k, (P_j, \mathtt{b}_j))$ from at least $n-t$ parties, party P_i adds $(P_j, \mathtt{b}_j)$ to $\mathtt{manySaw}_i$.
- Once $\mathtt{manySaw}_i$ contains tuples $(P_j, \cdot)$ for at least $n-t$ parties, P_i waits for Δ_{CSS} (while still collecting tuples) and then sets $\mathtt{report}_i$ to $\perp$.

Closing Down:

- Once P_i sets $\mathtt{report}_i$ to $\perp$, P_i sends to all parties signed $(\textsc{doneReporting}, P_i, \mathtt{iSaw}_i)$.
- Once P_i received $J_{\mathtt{done}}$-justified $(\textsc{doneReporting}, P_j, \mathtt{iSaw}_j)$ from at least $n-t$ parties, P_i sets $\mathtt{Core}_i$ to be the set of all currently $J_{\mathtt{tpl}}$-justified $(P_j, \mathtt{b}_j)$. It then waits for Δ_{CSS} (and stops collecting messages), and afterwards outputs $\mathtt{Core}_i$.

Lemma 5. *For $t < \frac{n}{3}$ the protocol* CSS *satisfies common core, weak validity, unique honest tuples, termination, and Δ_{CSS}-waiting.*

Proof. We prove each individual property next.

Common Core: Let P_i be the first honest that sends out $(\textsc{doneReporting}, P_i, \mathtt{iSaw}_i)$. At this point P_i's $\mathtt{manySaw}_i$ contains $J_{\mathtt{tpl}}$-justified tuples $(P_j, \mathtt{b}_j)$ from at least $n-t$ parties. Additionally note that if $(P_j, \mathtt{b}_j) \in \mathtt{manySaw}_i$ then at least $n-2t > t$ honest parties must have added $(P_j, \mathtt{b}_j)$ to their $\mathtt{iSaw}$.

Let P_k be an honest party with output $\mathtt{Core}_k$. We now argue that any tuple $(P_j, \mathtt{b}_j)$ in $\mathtt{manySaw}_i$ must be part of $\mathtt{Core}_k$. At the point where P_k computed $\mathtt{Core}_k$ the party has seen at least $n-t$ $J_{\mathtt{done}}$-justified $(\textsc{doneReporting}, P, \mathtt{iSaw})$. So one of them must come from an honest party which has $(P_j, \mathtt{b}_j)$ added to their $\mathtt{iSaw}$ (as $n-2t > t$ have added it to their $\mathtt{iSaw}$). Thus P_k will consider $(P_j, \mathtt{b}_j)$ $J_{\mathtt{tpl}}$-justified at this point and add it to $\mathtt{Core}_k$.

Weak Validity: The output set $\mathtt{Core}_i$ contains only tuples $(P_k, \mathtt{b}_k)$ which are $J_{\mathtt{tpl}}$-justified for P_i. As $\mathtt{b}$ is the only $J_{\mathtt{cssin}}$-justified value, only tuples of the form $(\cdot, \mathtt{b})$ are $J_{\mathtt{tpl}}$-justified. Thus all tuples in $\mathtt{Core}_i$ are of the form $(\cdot, \mathtt{b})$.

Unique Honest Tuple: An honest party P_j will only send out its signed input bit $\mathtt{b}_j$. Thus if a tuple $(P_j, \mathtt{b})$ is considered $J_{\mathtt{tpl}}$-justified by P_i, we have that $\mathtt{b} = \mathtt{b}_j$.

Termination: Each honest party P_i will send out its signed input bit $\mathtt{b}$. Any other honest P_j will add $\mathtt{b}$ to its $\mathtt{iSaw}_j$ (as $J_{\mathtt{cssin}}$ is an eventual justification) and send out $(\textsc{seen}, P_j, (P_i, \mathtt{b}_i))$. As there are at least $n-t$ honest parties, all honest parties will add at least $n-t$ tuples to their $\mathtt{manySaw}$. This implies that they all will send out $(\textsc{doneReporting}, \cdot, \cdot)$ messages which are justified for all other honest parties. Thus every honest party P_i will eventually output a $\mathtt{Core}_i$.

Δ_{CSS}**-Waiting:** If Δ_{CSS} is large enough, then any honest party P_i will have enough time to broadcast their input bit, such that any other honest party P_j will receive it before they set $\mathtt{report}_i$ to $\perp$. Furthermore, waiting for Δ_{CSS} time after fixing $\mathtt{Core}_i$ guarantees that all honest outputs are fixed before any honest party gives an output.

□

Corollary 2. *The output set* $\mathtt{Core}_i$ *of honest party* P_i *contains tuples* $(P_j, \mathtt{b}_j)$ *from at least* $n-t$ *different parties.*

Corollary 3. *The output* $\mathtt{Core}_i$ *of party* P_i *contains tuples* $(P_j, \mathtt{b})$ *and* $(P_j, \mathtt{b}')$ *with* $\mathtt{b} \neq \mathtt{b}'$ *for at most* t *parties.*

Proof. The corollary is implied by the unique-honest-tuple property. □

6.3 Another Binary Byzantine Agreement

We now describe a Binary Byzantine Agreement protocol (AƎBA). Parties use AƎBA to decide whether they agreed on a non-$\perp$ decision in Freeze (resp. FilteredFreeze). The global inputs, i.e., the pre-agreed parameters, are input justification $J_{\mathtt{in}}$ and a delay $\Delta_{\mathtt{ABBA}}$. Each party has a $J_{\mathtt{in}}$-justified bit $\mathtt{b} \in \{\perp, \top\}$ as input. The output of honest parties in AƎBA are $J_{\mathtt{out}}$-justified bits (see Definition 19).

The AƎBA protocol is a type of randomized graded agreement. The protocol consists of multiples phases. In each phase parties propose their current bit. After a weak core-set agreement using CSS parties make a choice to update their current bit. They each grade their choice from 0 to 2. The randomization comes in the form of a leader election where the elected leader helps parties with grade 0 to select their current bit. The protocol AƎBA has the following properties.

Consistency: If some honest P_i and P_j output bits $\mathtt{b}_i$ respectively $\mathtt{b}_j$, then $\mathtt{b}_i = \mathtt{b}_j$.

Validity: If all honest parties input the same $J_{\mathtt{in}}$-justified bit $\mathtt{b}$, then no honest P_j outputs a decision $\mathtt{b}''$ with $\mathtt{b}'' \neq \mathtt{b}$.

Termination: If all honest parties input some $J_{\mathtt{in}}$-justified bit, then eventually all honest voters output some bit.

We use the following justifications in AƎBA.

Definition 17. A bit $\mathtt{b}$ is $J_{\mathsf{phase,1}}$-justified (*phase*-1 *justified*) for P_i if it is $J_{\mathtt{in}}$-justified.

Definition 18. For $k > 1$ a bit $\mathtt{b}$ is $J_{\mathsf{phase,k}}$-justified (*phase-k justified*) for P_i if P_i has $t+1$ signatures on $(\mathtt{baid}, \textsc{justified}, \mathtt{b}, k-1)$.

Definition 19. A bit $\mathtt{b}$ is $J_{\mathtt{out}}$-justified (output) for P_i if P_i has $t+1$ signatures on $(\mathtt{baid}, \textsc{WeAreDone}, \mathtt{b})$.

Leader election lottery. The A∃BA protocol requires a lottery which ranks parties. We need that every party gets a "lottery ticket" such that other parties can verify the ticket and every party has the same probability of having the highest ticket. Furthermore, we require that lottery tickets of honest parties cannot be predicted before they sent it. This can, e.g., be implemented using a *verifiable random function* (VRF) [24] with unpredictability under malicious key generation [11]. Such a VRF that can locally be evaluated by every party and verified by others using a public key. Depending on the underlying blockchain, one can also use some other mechanism offered by the blockchain.

Protocol. We next describe the protocol.

Protocol A∃BA($\mathtt{baid}, J_{\mathtt{in}}, \Delta_{\mathtt{ABBA}}$)

The protocol is described from the view point of a party P_i which has $J_{\mathtt{in}}$-justified input $\mathtt{b}_i$. The party starts both the "Graded Agreement" and the "Closing Down" part of the protocol.

Graded Agreement

In each phase $k = 1, 2, \ldots$ do the following:

1. The parties jointly run CSS($\mathtt{baid}, J_{\mathtt{phase,k}}, k \cdot \Delta_{\mathtt{ABBA}}$) where P_i inputs $\mathtt{b}_i$. Denote by $\mathtt{Core}_i$ the output of P_i.
2. P_i computes its lottery ticket $\mathtt{ticket}_i$ and broadcasts signed $(\mathtt{baid}, \text{JUSTIFIED}, \mathtt{b}_i, k)$ along with $\mathtt{ticket}_i$.
3. P_i waits for time $k \cdot \Delta_{\mathtt{ABBA}}$ and then does the following:
   - If all bits (of the tuples) in $\mathtt{Core}_i$ are $\top$ let $\mathtt{b}_i = \top$ and $\mathtt{grade}_i = 2$.
   - Else if at least $n - t$ bits in $\mathtt{Core}_i$ are $\top$ let $\mathtt{b}_i = \top$ and $\mathtt{grade}_i = 1$.
   - Else if all bits in $\mathtt{Core}_i$ are $\perp$ let $\mathtt{b}_i = \perp$ and $\mathtt{grade}_i = 2$.
   - Else if at least $n - t$ bits in $\mathtt{Core}_i$ are $\perp$ let $\mathtt{b}_i = \perp$ and $\mathtt{grade}_i = 1$.
   - Else, select a bit $\mathtt{b}$ which occurs $> t$ in $\mathtt{Core}_i$. If this bit is not unique, verify all lottery tickets and select the bit $\mathtt{b}$ where $(\mathtt{b}, P) \in \mathtt{Core}_i$ and P has the highest valid lottery ticket for all parties in $\mathtt{Core}_i$. Let $\mathtt{b}_i = \mathtt{b}$ and $\mathtt{grade}_i = 0$.
4. Go to the next phase.

Closing Down Each party sends at most one $(\mathtt{baid}, \text{WEAREDONE}, \cdot)$ message.

1. When P_i achieves grade 2 for the first time it sends signed $(\mathtt{baid}, \text{WEAREDONE}, \mathtt{b}_i)$ to all parties.
2. Once having receiving at least $t + 1$ signed $(\mathtt{baid}, \text{WEAREDONE}, \mathtt{b})$ terminate the protocol and output $J_{\mathtt{out}}$-justified $\mathtt{b}$.

Lemma 6. *For $n > 3t$ the protocol* A∃BA *satisfies agreement, validity and, termination.*

Proof. We first proceed to prove the following claims.

Claim 1. *At the start of any phase k the current bit $\mathtt{b}_i$ of an honest party P_i is $J_{\mathtt{phase,k}}$-justified for P_i and is eventually $J_{\mathtt{phase,k}}$-justified for any other party.*

Proof. In the first phase the bit $\mathtt{b}_i$ is the $J_{\mathtt{in}}$-justified input bit, thus the statement holds. So assume that the start of phase $k-1$ all honest parties have a $J_{\mathtt{phase,k-1}}$-justified bit. In Step 2 of phase $k-1$ they broadcast $n-t \geq 2t+1$ messages of the form $(\mathtt{baid}, \text{JUSTIFIED}, \cdot, k-1)$. These messages will eventually be received by all honest parties. Thus in phase k at least one bit must be eventually $J_{\mathtt{phase,k}}$-justifiable for all honest parties. By the design of AƎBA honest parties will select such a bit in Step 3 of phase $k-1$. So they all end up with a $J_{\mathtt{phase,k}}$-justified bit at the start of phase k. □

Claim 2. *Eventually all honest party will end up with the same bit* $\mathtt{b}$ *and grade* $\mathtt{grade} = 2$.

Proof. Consider the following cases:

Case 1: Assume that in some phase k in Step 1 there exists a $J_{\mathtt{phase,k}}$-justified bit $\mathtt{b}$ such that all honest parties have $\mathtt{b}_i = \mathtt{b}$.

All honest parties send out signed $(\mathtt{baid}, \text{JUSTIFIED}, \mathtt{b}, k)$ and start CSS with justified inputs. By the termination property of CSS every honest party will eventually have an output. By the common-core property the output sets have a common core of size at least $n-t$. By the unique-honest tuple property $\mathtt{b}$ will occur at least $t+1$ in P_i's output set $\mathtt{Core}_i$. On the other hand $1-\mathtt{b}$ will occur at most t times in $\mathtt{Core}_i$. Therefore, any honest party P_i will select again $\mathtt{b}$ in Step 3 (with a grade of 0 or more). In the next phase $k+1$ all honest parties have $\mathtt{b}_i = \mathtt{b}$ and no other bit is $J_{\mathtt{phase,k+1}}$-justified. In this phase by the weak validity property of CSS it follows that all honest parties will have $\mathtt{b}_i = \mathtt{b}$ and $\mathtt{grade}_i = 2$ after Step 3. Afterwards, the honest parties will no longer change their values nor their grades.

Case 2: Assume that in some phase k after Step 3 an honest party P_i has $\mathtt{b}_i = \mathtt{b}$ and $\mathtt{grade}_i = 2$. That means $\mathtt{Core}_i$ from CSS and thus the core-set only contains tuples with $\mathtt{b}$. So any other honest party P_j has $\mathtt{b}$ at least $n-t$ times in its $\mathtt{Core}_j$. At the same time P_j cannot have at least $n-t > 2t$ tuples with $1-\mathtt{b}$ in $\mathtt{Core}_j$. Hence after Step 3 P_j will set $\mathtt{b}_j = \mathtt{b}$ with $\mathtt{grade}_j \geq 1$. Thus in the next phase we are in Case 1.

Case 3: Assume that in some phase after Step 3 there is a bit $\mathtt{b}$ such that any honest party P_i either has $\mathtt{b}_i = \mathtt{b}$ with $\mathtt{grade}_i \geq 1$ or $\mathtt{b}_i$ arbitrary with $\mathtt{grade}_i = 0$.

We assume that $k \cdot \Delta_{\mathtt{ABBA}}$ is larger than the network delay, which will eventually happen. Otherwise the adversary can potentially delay messages from honest parties with high lottery tickets and we end up in one of the cases 1-3. In case $k \cdot \Delta_{\mathtt{ABBA}}$ is large enough, the lottery tickets of all honest parties have arrived after waiting in Step 3. Furthermore, the Δ_{CSS}-waiting property of CSS guarantees that all $\mathtt{Core}_i$ and the common $\mathtt{Core}$ contains the tuples of all honest parties.

sub-case a): Assume that there is some honest party P_i with $\mathtt{b}_i = \mathtt{b}$ and grade $\mathtt{grade}_i = 1$ after Step 3. Then, there are at least $n-t$ tuples of the form $(\cdot, \mathtt{b})$ in $\mathtt{Core}_j$. Let $x \geq n-t$ be the size of the core-set. Then, there are at least $x-t \geq n-2t > t$ tuples of the form $(\cdot, \mathtt{b})$ from honest parties in the core-set. This implies that $\mathtt{b}$ is a justified choice for all honest parties. If the other bit is not justified for any honest party, all honest parties choose $\mathtt{b}$ and we are in Case 1 in the next phase. Otherwise, some honest parties will use the highest lottery ticket to determine their output. We analyze this case below.

sub-case b): Assume that all honest parties have $\mathtt{grade}_i = 0$. Let $\mathtt{b}'$ be the bit input to CSS by more honest parties (and $\mathtt{b}' = \top$ if both bits are input equally often).

Since the tuples of all honest parties are in the core-set, $\mathtt{b}'$ is at least $t+1$ times in the core-set, and thus in every $\mathtt{Core}_i$. It is therefore a justified choice for all honest parties. Hence, either all parties will choose $\mathtt{b}'$ since it is the only justified bit, or some honest parties will choose their bit according to the highest lottery ticket.

We finally consider the case where some honest parties make their choice according to the highest lottery ticket. In both sub-cases, there is a bit $\mathtt{b}''$ that corresponds to at least $n/3$ honest lottery tickets such that if all honest parties choose $\mathtt{b}''$, then we are in Case 1 in the next phase (in sub-case a), $\mathtt{b}'' = \mathtt{b}$ and in sub-case b), $\mathtt{b}'' = \mathtt{b}'$). Note that the Δ_{CSS}-waiting property of CSS guarantees that all honest outputs of CSS are fixed before the tickets are generated. Thus, the lottery tickets are independent of $\mathtt{b}''$. Since all tickets have the same probability of being the largest one, and all honest tickets are considered by all honest parties, the probability that the winning ticket is an honest one with bit $\mathtt{b}''$ is at least $\frac{1}{3}$. Otherwise, we again end up in one of the cases 1-3.

Case 4: Assume that in some phase after Step 3 some honest party P_i has $\mathtt{b}_i = \mathtt{b}$ and $\mathtt{grade}_i \geq 1$ while an other honest party P_j has $\mathtt{b}_j = 1 - \mathtt{b}$ and $\mathtt{grade}_i \geq 1$. We now show that this case can not happen. This would imply that in $\mathtt{Core}_i$ there are at least $n-t$ tuples $(\cdot, \mathtt{b})$ and in $\mathtt{Core}_j$ there are at least $n-t$ tuples $(\cdot, 1-\mathtt{b})$. As the sets have a common core-set of size at least $n-t$ we have that in the core-set there are $n-2t$ parties with tuples for both $\mathtt{b}$ and $1-\mathtt{b}$. This is a contradiction to Corollary 3.

Clearly the network is always in one of the four above case. In each possible case and in each phase, we have that once $k \cdot \Delta_{\mathtt{ABBA}}$ exceeds the de-synchronization of the parties, they end up in Case 1 in the next phase with probability at least 1/3. This means that once $k \cdot \Delta_{\mathtt{ABBA}}$ is large enough, the expected number of phases needed to reach Case 1 is constant. Once in Case 1, they will stay there forever. □

Finally, we can show the properties.

Termination: If all hones parties have a $J_{\mathtt{in}}$-justified input, then we start in one of the Cases 1-3. Claim 2 implies that eventually we end up in Case 1 or Case 2. In particular, once the first honest party has grade 2 for $\mathtt{b}$ all honest parties will decide with grade 2 on that bit. Thus all honest parties will eventually send out signed $(\mathtt{baid}, \textsc{WeAreDone}, \mathtt{b})$. Therefore all honest parties eventually output $\mathtt{b}$ together with $n-t$ signatures on $(\mathtt{baid}, \textsc{WeAreDone}, \mathtt{b})$.

We can give a bound on the expected number of phases in ABBA can be expressed with respect to the actual network delay Δ. As long as $k \cdot \Delta_{\mathtt{ABBA}} < \Delta$, in the worst-case parties are stuck in Case 3. This is the case for at most $\frac{\Delta}{\Delta_{\mathtt{ABBA}}}$ phases. Afterwards we have a probability of at least $\frac{1}{3}$ to transition from Case 3 to Cases 1-2. Thus the expected number of phases is at most $\frac{\Delta}{\Delta_{\mathtt{ABBA}}} + c$ for some constant c.

Consistency: Once the first honest party sends $(\mathtt{baid}, \textsc{WeAreDone}, \mathtt{b})$ all honest parties will converge in Case 1 with $\mathtt{b}$. Thus they all will all send out $(\textsc{WeAreDone}, \mathtt{b})$ as well. If an honest party outputs a bit, it must be $\mathtt{b}$. Note that this also ensures that there will no other $J_{\mathtt{out}}$-justified bit.

Validity: Assume all honest parties input $J_{\mathtt{in}}$-justified bit $\mathtt{b}$. Then in the first phase of Freeze, the honest parties are Case 1 of Claim 2. Thus all honest parties will decide on $\mathtt{b}$ with grade 2 in the first phase. Thus they will not output any other bit $\mathtt{b}'$.

□

Remark 8. In A∃BA, there are three places (including two within CSS) where parties wait. These waiting times are effective once they exceed the de-synchronization of the parties (and are the reason our protocol is partially synchronous and not asynchronous): The first one in CSS ensures that all honest parties make it into the core-set, the second one in CSS ensures that all honest outputs of CSS are fixed before honest parties give their outputs (which in turn guarantees that these outputs do not depend on the lottery tickets in A∃BA), and the last one in A∃BA ensures that all honest lottery tickets arrive in time. By reordering the protocol and letting one instance of waiting take care of more than one property, it is possible to reduce the overall waiting time. Since this complicates the analysis, we do not discuss this further here.

6.4 WMVBA Protocol

We can now describe the actual WMVBA protocol. Each party inputs a J-justified proposal and gets a $J_{\mathtt{fin}}$-justified output which is either a proposal or $\bot$.

The idea of is WMVBA to first call Freeze to boil down the choice to a unique proposal or $\bot$. Parties then use A∃BA which one is the case. For A∃BA the parties use some globally known $\Delta_{\mathtt{ABBA}}$ as the initial waiting time. We conjecture it works well in practice to set $\Delta_{\mathtt{ABBA}}$ equal to the expected network delivery time. Any value works in principle since we increase the waiting time in each phase. In particular, a bad choice of $\Delta_{\mathtt{ABBA}}$ has no influence on the properties of A∃BA.

Note that it can happen that an honest party decides on $\bot$ at the end of Freeze, but A∃BA nevertheless outputs $\top$. In this case, at least one honest party had a justified non-$\bot$ decision as output in Freeze. This decision is unique. So we must ensure that honest parties with $\bot$ output in Freeze can somehow get their hands on that decision. For that, parties do not terminate Freeze once they get their output, but instead continue to collect decisions and vote-messages. By Lemma 4, this ensures that all honest parties will eventually receive the unique non-$\bot$ decision.

We define the following justification. First, we look at the justification for inputs of A∃BA. The idea is that parties input $\bot$ (resp. $\top$) to A∃BA if their $J_{\mathtt{dec}}$-justified output of Freeze was $(\mathtt{baid}, \text{FROZEN}, \bot)$ (resp. $(\mathtt{baid}, \text{FROZEN}, \mathtt{d})$ for $\mathtt{d} \neq \bot$).

Definition 20. A bit $\mathtt{b}$ is $J_{\mathtt{in}}$-justified (*input justified*) for party P_i if P_i has a $J_{\mathtt{dec}}$-justified tuple $(\mathtt{baid}, \text{FROZEN}, \mathtt{d})$ where $\mathtt{d} \neq \bot$ if and only if $\mathtt{b} \neq \bot$.

Finally, we define the justification for outputs of WMVBA.

Definition 21. A decision $\mathtt{d}$ is considered justified with respect to *final justification* $J_{\mathtt{fin}}$ for P_i if P_i has $t+1$ signatures on the message $(\mathtt{baid}, \text{WeAreDone}, \mathtt{d})$.

Note that a $\bot$ output from A∃BA is already $J_{\mathtt{fin}}$-justified. The protocol formally works as follows:

Protocol WMVBA($\mathtt{baid}, J$)

Let $\Delta_{\mathtt{ABBA}}$ be a globally know estimate of the network delay. The protocol is described from the view point of party P_i which has J-justified input $\mathtt{p}_i$.

1. Run Freeze($\mathtt{baid}, J$) with input $\mathtt{p}_i$. Denote by $\mathtt{d}_i$ the $J_{\mathtt{dec}}$-justified output for P_i from Freeze.
2. Run A∃BA($\mathtt{baid}, J_{\mathtt{in}}, \Delta_{\mathtt{ABBA}}$) with input $\mathtt{b}_i$ where $\mathtt{b}_i = \bot$ if $\mathtt{d}_i = \bot$ and $\mathtt{b}_i = \top$ otherwise. Denote by $\mathtt{b}'_i$ the output of A∃BA for P_i.
3. If $\mathtt{b}'_i = \bot$, then terminate and output $\mathtt{b}'_i$ (which is $J_{\mathtt{fin}}$-justified) together with $\mathtt{W} = \bot$,

otherwise (if $\mathtt{b}'_i = \top$) do:

- Once P_i has a $J_{\mathtt{dec}}$-justified decision message $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_i)$ with $\mathtt{d}_i \neq \bot$ (from Freeze) it sends signed $(\mathtt{baid}, \text{WeAreDone}, \mathtt{d}_i)$ to all other parties.
- Once $t+1$ signed $(\mathtt{baid}, \text{WeAreDone}, \mathtt{d})$, for some $\mathtt{d}$, have been received, terminate and output $(\mathtt{d}, \mathtt{W})$, where $\mathtt{W}$ contains $\mathtt{baid}$ and $t+1$ of these signatures.

Theorem 2. *For $t < \frac{n}{3}$ the protocol* WMVBA *satisfies consistency, weak validity, $n/3$-support, and termination.*

Proof. We begin by showing that for ABBA validity is equivalent to 1-support. Assume validity holds. If parties have different inputs 1-support follows directly. Otherwise if all honest parties have the same input, then validity implies that they output this value. Assume 1-support holds and all parties input the same bit. By 1-support they must output this bit.

Next, we prove each individual property.

Consistency: Consider two honest parties P_i, P_j with $J_{\mathtt{fin}}$-justified outputs outputs $\mathtt{d}_i$ and $\mathtt{d}_j$. Assume that they are different.

case i) Assume that without loss of generality $\mathtt{d}_i = \bot$ and thus $\mathtt{d}_j \neq \bot$. In this case P_i had output $\bot$ from ABBA and P_j had output $\top$ from ABBA. This contradicts the consistency property of ABBA.

case ii) Assume that both $\mathtt{d}_i$ and $\mathtt{d}_j$ are not equal to $\bot$. Then, both parties P_i and P_j got $\top$ as output from ABBA. This implies that at least one honest party P_k had input $\top$ to ABBA due to the 1-support of ABBA. This party P_k must have had $J_{\mathtt{dec}}$-justified output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_k)$ from Freeze (with $\mathtt{d}_k \neq \bot$).

Let without loss of generality $\mathtt{d}_i \neq \mathtt{d}_k$, then it must be that P_i collected at least $t+1$ signed messages $(\mathtt{baid}, \text{WeAreDone}, \mathtt{d}_i)$ of which *at least* one must have been broadcast by an honest party P_l. That party P_l must consider $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_i)$ to be $J_{\mathtt{dec}}$-justified. This is a contradiction to the weak consistency of Freeze.

Thus, the output of honest parties must be the same.

Weak Validity: Assume that there exists a $\mathtt{d}$ such that any $\mathtt{d}' \neq \mathtt{d}$ is not J-justified during the protocol run.

The weak validity of Freeze implies that no $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}')$ with $\mathtt{d}' \neq \mathtt{d}$ is output by an honest party in Freeze. In particular, $(\mathtt{baid}, \text{FROZEN}, \bot)$ cannot become $J_{\mathtt{dec}}$-justified. So $\bot$ is not a $J_{\mathtt{in}}$-justified input for ABBA.

So all honest parties will input $\top$ to ABBA. Hence by the validity of ABBA it follows that $\bot$ is not a $J_{\mathtt{out}}$-justified output for ABBA. Therefore neither $\bot$ nor a decision $\mathtt{d}' \neq \mathtt{d}$ can be an $J_{\mathtt{fin}}$-justified output of ABBA.

$n/3$-Support: Assume that honest party P_i outputs $\mathtt{d}_i \neq \bot$.

Therefore P_i had output $\top$ from ABBA. By 1-support of ABBA at least one honest party P_j had input $\top$ to ABBA. This party P_j must have had $J_{\mathtt{dec}}$-justified output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_j)$ from Freeze with $\mathtt{d}_j \neq \bot$.

We also know that P_i collected at least $t+1$ signed $(\mathtt{baid}, \text{WeAreDone}, \mathtt{d}_i)$. So, at least one honest party considers $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_i)$ to be $J_{\mathtt{dec}}$-justified. By Corollary 1 we must have $\mathtt{d}_i = \mathtt{d}_j$. The $n-2t$-support property of Freeze implies that at least $n-2t$ honest parties had input $\mathtt{d}_i$ for Freeze. This means that at least $\frac{n}{3}$ honest parties had input $\mathtt{d}_i$ for WMVBA.

Termination: Assume that all honest parties have a J-justified proposal as input. By the termination property of Freeze all honest parties will have a $J_{\mathtt{dec}}$-justified output. Thus they all have a $J_{\mathtt{in}}$-justified input for ABBA. By the termination property of ABBA they all have an $J_{\mathtt{out}}$-justified output from ABBA. Consider the following cases.

case i): Assume honest party P_i has $J_{\mathtt{out}}$-justified output $\perp$ from ABBA. Then P_i output $J_{\mathtt{fin}}$-justified $\perp$.

case ii): Assume honest party P_i has $J_{\mathtt{out}}$-justified output $\top$ from ABBA. The 1-support of ABBA implies that at least one honest party P_j had input $\top$ for ABBA. Thus party P_j had $J_{\mathtt{dec}}$-justified output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_j)$ from Freeze with $\mathtt{d}_j \neq \perp$. Lemma 4 implies that eventually all honest parties will accept $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_j)$ as $J_{\mathtt{dec}}$-justified. Thus P_i will eventually output a $J_{\mathtt{dec}}$-justified decision.

□

Message complexity. We observe that in WMVBA all messages are multi-cast, i.e., addressed to all parties. In the following we thus count the number of multi-cast messages sent by (honest) parties.

Lemma 7. *Let Δ be the actual network delay. Then* WMVBA *has an expected message complexity of $\mathcal{O}(\frac{\Delta}{\Delta_{\mathtt{ABBA}}} n^2)$.*

Proof. In Freeze honest each party sends 2 messages, thus we have a message complexity of $2n$. In each phase of ABBA the parties run CSS which has a message complexity of $2n+n^2$ and each send one message. Thus one phase consists of $3n+n^2$ messages. The expected number of phases is $\frac{\Delta}{\Delta_{\mathtt{ABBA}}} + c$ for some small constant c (see proof of Lemma 6). At the end of ABBA each party sends an additional message. Thus ABBA as an expected message complexity of $(3n+n^2)(\frac{\Delta}{\Delta_{\mathtt{ABBA}}} + c) + n$. Finally, at the end of WMVBA an additional n messages might be sent. So overall we have

$$4n + (3n+n^2)(\frac{\Delta}{\Delta_{\mathtt{ABBA}}} + c) = \frac{\Delta}{\Delta_{\mathtt{ABBA}}}(3n+n^2) + (4+3c)n + cn^2$$

So WMVBA has an expected message complexity of $\mathcal{O}(\frac{\Delta}{\Delta_{\mathtt{ABBA}}} n^2)$. □

6.5 Filtered WMVBA Protocol

As described before, FilteredWMVBA is a variant of the WMVBA protocol for blockchains without the DCGrowth property. It has a stronger validity guarantee such that we do not need a unique justified proposal to achieve finalization. Instead, it is enough if all honest parties agree on a proposal. However, this comes at the cost that FilteredWMVBA only offers 1-support instead of $n/3$-support. Technically, FilteredWMVBA is the same as WMVBA except we use a slightly altered Freeze subprotocol called FilteredFreeze.

Filtered Freeze. FilteredFreeze is a variant of the Freeze protocol. It is essentially Freeze with an additional step where proposals with low support are filtered out. It provides a stronger validity guarantee. This comes at the cost of a lower support guarantee. Each party honest P_i has a J-justified input $\mathtt{p}_i$ and the output of P_i is justified by justification $J_{\mathtt{dec}}$ (cf. Definition 13). The protocol has the following properties.

Weak Consistency: If some honest P_i and P_j output decisions $\mathtt{d}_i \neq \bot$ respectively $\mathtt{d}_j \neq \bot$, then $\mathtt{d}_i = \mathtt{d}_j$.

Validity: If all honest parties input the same J-justified proposal $\mathtt{p}$, then no honest P_j outputs a decision $\mathtt{p}''$ with $\mathtt{p}'' \neq \mathtt{p}$.

1-Support: If honest party P_i outputs decision $\mathtt{d}_i \neq \bot$, then at least one honest party had $\mathtt{d}_i$ as input.

Termination: If all honest parties input some justified proposal, then eventually all honest parties output some decision.

For the new filter step we need the following justification.

Definition 22. A filtered proposal message $m = (\mathtt{baid}, \text{FILTERED}, \mathtt{p}, \sigma)$ is considered $J_{\mathtt{filt}}$-justified for P_j if either σ contains $J_{\mathtt{prop}}$-justified proposal messages for $\mathtt{p}$ from $t+1$ different parties or σ contains $J_{\mathtt{prop}}$-justified proposal messages from $n-t$ different parties such that no proposal is contained in more than t of those messages.

Vote messages are now cast after the filter step and depend on filtered proposal messages. We thus redefine the justification for vote messages as follows. The definition of $J_{\mathtt{dec}}$ (relative to the redefined $J_{\mathtt{vote}}$) stays the same as for Freeze.

Definition 23. A vote message $m = (\mathtt{baid}, \text{VOTE}, \mathtt{v})$ from P_i is considered $J_{\mathtt{vote}}$-justified for P_j if is signed by P_i and either for $\mathtt{v} \neq \bot$ P_j has collected $J_{\mathtt{filt}}$-justified filtered proposal messages from at least $n-2t$ parties or for $\mathtt{v} = \bot$ P_j has collected $J_{\mathtt{filt}}$-justified filtered proposal messages $(\mathtt{baid}, \text{FILTERED}, \mathtt{p}, \sigma)$ and $(\mathtt{baid}, \text{FILTERED}, \mathtt{p}', \sigma')$ (from two different parties) where $\mathtt{p}' \neq \mathtt{p}$.

Protocol FilteredFreeze($\mathtt{baid}$, J)

Each (honest) party P has a J-justified proposal $\mathtt{p}$ as input. Party P does the following:
Propose:

1. Broadcast signed proposal message $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{p})$.

Filter:

1. Collect proposal messages $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{p}_i)$. Once $J_{\mathtt{prop}}$-justified proposal messages from at at least $n-t$ parties have been collected do the following (but keep collecting proposal messages).
   (a) If your input $\mathtt{p}$ is contained in at least $t+1$ $J_{\mathtt{prop}}$-justified proposal messages, broadcast filtered proposal message $(\mathtt{baid}, \text{FILTERED}, \mathtt{p}, \sigma)$ where σ is a set of $t+1$ signed proposal messages which all contain $\mathtt{p}$.
   (b) Else if there is any $\mathtt{p}'$ which is contained in at least $t+1$ $J_{\mathtt{prop}}$-justified proposal messages, broadcast filtered proposal message $(\mathtt{baid}, \text{FILTERED}, \mathtt{p}', \sigma)$ where σ is a set of $t+1$ signed proposal messages which all contain $\mathtt{p}$. Do this for at most one proposal.
   (c) Else broadcast $(\mathtt{baid}, \text{FILTERED}, \mathtt{p}, \sigma)$ where σ is a set of $n-t$ signed proposal messages such that no proposal is contained in more than t of those proposal messages.

Vote:

2. Collect filtered proposal messages $(\mathtt{baid}, \text{FILTERED}, \mathtt{p}_i)$. Once $J_{\mathtt{filt}}$-justified filtered proposal messages from at at least $n-t$ parties have been collected do the following (but keep collecting filtered proposal messages).

   (a) If $J_{\mathtt{filt}}$-justified filtered proposal messages from at at least $n-t$ parties contain the same proposal $\mathtt{p}$, broadcast vote message $(\mathtt{baid}, \text{VOTE}, \mathtt{p})$.

   (b) Otherwise broadcast vote message $(\mathtt{baid}, \text{VOTE}, \perp)$.

Freeze:

3. Collects vote messages $(\mathtt{baid}, \text{VOTE}, \mathtt{p}_i)$ messages. Once $J_{\mathtt{vote}}$-justified vote messages from at least $n-t$ parties have been collected and there is a value contained in at least $t+1$ vote messages do the following.

   (a) If $J_{\mathtt{vote}}$-justified vote messages from strictly more than t parties contain the same $\mathtt{p} \neq \perp$ output $(\mathtt{baid}, \text{FROZEN}, \mathtt{p})$.

   (b) Otherwise if $J_{\mathtt{vote}}$-justified vote messages from strictly more than t parties contain $\perp$ output $(\mathtt{baid}, \text{FROZEN}, \perp)$.

4. Keep collecting vote messages until WMVBA is terminated (i.e., until P_i gets an output in WMVBA). Party P_i keeps track of all decisions $(\mathtt{baid}, \text{FROZEN}, \mathtt{p})$ which become $J_{\mathtt{dec}}$-justified.

Lemma 8. *For $t < \frac{n}{3}$ the protocol* FilteredFreeze *satisfies weak consistency, validity, 1-support, and termination. The outputs of honest parties are $J_{\mathtt{dec}}$-justified.*

Proof. **Weak Consistency:** Assume that some honest party sends $J_{\mathtt{vote}}$-justified $(\mathtt{baid}, \text{VOTE},\mathtt{d}$ $)$ for $\mathtt{d} \neq \perp$. It must have received $J_{\mathtt{filt}}$-justified filtered proposal messages for $\mathtt{d}$ from at least $n-t$ different parties. Thus at most t honest parties sent $J_{\mathtt{filt}}$-justified filtered proposal messages for $\mathtt{d}'$ where $\mathtt{d}' \neq \mathtt{d}$. Therefore at most $2t < n-t$ parties sent $J_{\mathtt{filt}}$-justified filtered proposal messages for $\mathtt{d}'$ where $\mathtt{d}' \neq \mathtt{d}$. Therefore no honest party will send a $J_{\mathtt{vote}}$-justified vote message for $\mathtt{d}'$ for $\mathtt{d}' \neq \mathtt{d}$. Therefore, in **Freeze**, if two honest parties output $J_{\mathtt{dec}}$-justified $(\mathtt{baid}, \text{FROZEN}, \mathtt{d})$ and $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}')$, then $\mathtt{d} = \mathtt{d}'$.

Validity: Assume all honest parties have J-justified input $\mathtt{d}$.

So all honest parties will send out $J_{\mathtt{prop}}$-justified proposal messages for $\mathtt{d}$.

So all honest parties will receive at least $t+1$ $J_{\mathtt{prop}}$-justified proposal messages for $\mathtt{d}$ and thus all send out $J_{\mathtt{filt}}$-justified filtered proposal messages for $\mathtt{d}$.

On the other hand for any $\mathtt{d}' \neq \mathtt{d}$ there are no $t+1$ $J_{\mathtt{prop}}$-justified proposal messages for $\mathtt{d}'$. Also any set of $J_{\mathtt{prop}}$-justified proposal messages from $n-t$ different parties will contain at least $t+1$ proposal messages for $\mathtt{d}$. This means that no $J_{\mathtt{filt}}$-justified filtered proposal message for $\mathtt{d}'$ can exist.

Thus all honest parties will vote for $\mathtt{d}$ while no other $J_{\mathtt{vote}}$-justified can exist.

Thus all honest parties will output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d})$ which is $J_{\mathtt{dec}}$-justified while no other $J_{\mathtt{dec}}$-justified can exist.

1-Support: Assume honest party P_i outputs $\mathtt{d}_i \neq \perp$. Then it collected at least $t+1$ votes for $\mathtt{d}_i$. So at least one honest party sent out a vote for $\mathtt{d}_i$. This party must have collected at least $t+1$ filtered proposals for $\mathtt{d}_i$. So at least one honest party sent out a filtered proposal message for $\mathtt{d}_i$. This party must have collected at least $t+1$ proposal messages for $\mathtt{d}_i$. So at least one honest party had input $\mathtt{d}_i$.

Termination: Note that all used justifications are eventual justifications.

Every honest party will send out a justified proposal message. Thus all honest parties will eventually receive $J_{\mathtt{prop}}$-justified proposal message from $n-t$ different parties. They therefore send out all filtered proposal messages. Thus all honest parties will eventually receive $J_{\mathtt{filt}}$-justified filtered proposal messages from $n-t$ different parties and send out vote messages. So eventually they will all collect $J_{\mathtt{vote}}$-justified vote messages from $n-t$ and thus all output a decision.

□

Similar to Lemma 4 for Freeze we get.

Lemma 9. *Assume any message received by an honest party will eventually be received by all other honest parties. If an honest party P_i outputs ($J_{\mathtt{dec}}$-justified) decision $\mathtt{d}_i \neq \perp$ in* FilteredFreeze*, then eventually all honest parties will accept $\mathtt{d}_i$ has $J_{\mathtt{dec}}$-justified.*

The proof follows along the lines of the proof for Lemma 4.

Filtered WMVBA. The protocol FilteredWMVBA is identical to WMVBA where Freeze is replaced by FilteredFreeze.

Theorem 3. *For $t < \frac{n}{3}$ the protocol* FilteredWMVBA *satisfies consistency, validity, 1-support, and termination.*

Proof. We begin by showing that for AƎBA validity is equivalent to 1-support. Assume validity holds. If parties have different inputs 1-support follows directly. Otherwise if all honest parties have the same input, then validity implies that they output this value. Assume 1-support holds and all parties input the same bit. By 1-support they must output this bit.

Next, we prove each individual property.

Consistency: Consider two honest parties P_i, P_j with $J_{\mathtt{fin}}$-justified outputs outputs $\mathtt{d}_i$ and $\mathtt{d}_j$. Assume that they are different.

case i) Assume that without loss of generality $\mathtt{d}_i = \perp$ and thus $\mathtt{d}_j \neq \perp$. In this case P_i had output $\perp$ from AƎBA and P_j had output $\top$ from AƎBA. This contradicts the consistency property of AƎBA.

case ii) Assume that both $\mathtt{d}_i$ and $\mathtt{d}_j$ are not equal to $\perp$. Then, both parties P_i and P_j got $\top$ as output from AƎBA. This implies that at least one honest party P_k had input $\top$ to AƎBA due to the 1-support of AƎBA. This party P_k must have had $J_{\mathtt{dec}}$-justified output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_k)$ from FilteredFreeze (with $\mathtt{d}_k \neq \perp$).

Let without loss of generality $\mathtt{d}_i \neq \mathtt{d}_k$, then it must be that P_i collected at least $t+1$ signed messages $(\mathtt{baid}, \text{WeAreDone}, \mathtt{d}_i)$ of which *at least* one must have been broadcast by an honest party P_l. That party P_l must consider $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_i)$ to be $J_{\mathtt{dec}}$-justified. This is a contradiction to the weak consistency of FilteredFreeze.

Thus, the output of honest parties must be the same.

Validity: Assume that all honest parties input J-justified $\mathtt{d}$.

The validity of FilteredFreeze implies that no $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}')$ with $\mathtt{d}' \neq \mathtt{d}$ is output by an honest party in Freeze. In particular, $(\mathtt{baid}, \text{FROZEN}, \bot)$ cannot become $J_{\mathtt{dec}}$-justified. So $\bot$ is not a $J_{\mathtt{in}}$-justified input for A∃BA.

So all honest parties will input $\top$ to A∃BA. Hence by the validity of A∃BA it follows that $\bot$ is not a $J_{\mathtt{out}}$-justified output for A∃BA. Therefore neither $\bot$ nor a decision $\mathtt{d}' \neq \mathtt{d}$ can be an $J_{\mathtt{fin}}$-justified output of A∃BA.

1-Support: Assume that honest party P_i outputs $\mathtt{d}_i \neq \bot$.

Therefore P_i had output $\top$ from A∃BA. By 1-support of A∃BA at least one honest party P_j had input $\top$ to A∃BA. This party P_j must have had $J_{\mathtt{dec}}$-justified output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_j)$ from FilteredFreeze with $\mathtt{d}_j \neq \bot$. The 1-support of FilteredFreeze implies that at least on honest party had input $\mathtt{d}_j$. This party had input $\mathtt{d}_j$ to FilteredWMVBA.

Termination: Assume that all honest parties have a J-justified proposal as input. By the termination property of FilteredFreeze all honest parties will have a $J_{\mathtt{dec}}$-justified output. Thus they all have a $J_{\mathtt{in}}$-justified input for A∃BA. By the termination property of A∃BA they all have an $J_{\mathtt{out}}$-justified output from A∃BA. Consider the following cases.

case i): Assume honest party P_i has $J_{\mathtt{out}}$-justified output $\bot$ from A∃BA. Then P_i output $J_{\mathtt{fin}}$-justified $\bot$.

case ii): Assume honest party P_i has $J_{\mathtt{out}}$-justified output $\top$ from A∃BA. The 1-support of A∃BA implies that at least one honest party P_j had input $\top$ for A∃BA. Thus party P_j had $J_{\mathtt{dec}}$-justified output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_j)$ from FilteredFreeze with $\mathtt{d}_j \neq \bot$. Lemma 9 implies that eventually all honest parties will accept $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_j)$ as $J_{\mathtt{dec}}$-justified. Thus P_i will eventually output a $J_{\mathtt{dec}}$-justified decision.

□

Message Complexity The message complexity of FilteredWMVBA is similar to the message complexity of WMVBA.

Lemma 10. *Let Δ be the actual network delay. Then* FilteredWMVBA *has an expected message complexity of $\mathcal{O}(\frac{\Delta}{\Delta_{\mathtt{ABBA}}} n^2)$.*

Proof. In FilteredFreeze parties send n more messages than in Freeze. Thus the overall message complexity is still dominated by the n^2 from A∃BA (see proof of Lemma 7) We get that FilteredWMVBA has an expected message complexity of $\mathcal{O}(\frac{\Delta}{\Delta_{\mathtt{ABBA}}} n^2)$. □

7 Security Analysis of Finalization

In this section we show that the protocol described in Section 5 is a finality protocol as defined in Section 4.

Theorem 4. *For $t < \frac{n}{3}$ and a blockchain satisfying* DCGrowth *there exists a Δ such that the protocol described in Section 5 satisfies $(\Delta, n/3)$-finality.*

Proof. We argue each property individually next.

Agreement: We proof the property by induction.

The statement is true at the beginning of the protocol ($k = 0$).

So assume the statement holds for $k-1$. An honest party will call ($\textsc{setFinal}, \cdot$) for the k-th time after getting output R_i from $\mathsf{Finalization}$. The agreement property of WMVBA (cf. Theorem 2) guarantees that all honest parties output the same R_i. Thus they will all input ($\textsc{setFinal}, R_i$).

Chain-forming: Consider an honest party P_i which at time τ inputs ($\textsc{setFinal}, R$). Let $\mathtt{lastFinal}_i$ be the last finalized block of P_i. As P_i is honest R was the output of $\mathsf{Finalization}$. In $\mathsf{Finalization}$ the WMVBA protocol is used to agree on R. The support property of WMVBA (cf. Theorem 2) implies that R was input by at least one honest party P_j. By the design of $\mathsf{Finalization}$ party P_j selected R in the subtree of $\mathtt{lastFinal}_j$ (at that time). By the agreement property we have $\mathtt{lastFinal}_i = \mathtt{lastFinal}_j$ and it follows that $\mathtt{lastFinal}_i \in \mathrm{PathTo}(\mathtt{Tree}_i^\tau, R)$. As the output of $\mathsf{NextFinalizationGap}$ is ≥ 1 we have that R is at greater depth than $\mathtt{lastFinal}_i$. So $R \neq \mathtt{lastFinal}_i$.

$n/3$-Support: Consider honest party P_i at time τ inputting ($\textsc{setFinal}, R$). As P_i is honest R was the output of $\mathsf{Finalization}$. In $\mathsf{Finalization}$ the WMVBA protocol is used to agree on R. The $n/3$-support property of WMVBA (cf. Theorem 2) implies that R was input by at least $n/3$ honest parties. By the design of $\mathsf{Finalization}$ these parties selected R as on their $\mathtt{Path}$.

Δ-Updated: First, we show that any invocation of $\mathsf{Finalization}$ eventually terminates. As the blockchain satisfies chain growth we know that all honest parties will eventually start WMVBA in each execution of the repeat until loop of $\mathsf{Finalization}$. Note also that all honest parties have a justified input to WMVBA so it will terminate. The parties will exit the loop if WMVBA outputs a non-$\perp$ decision. Let δ_{freeze} be an upper bound on the duration of the Freeze sub-protocol. If the blockchain satisfies the UJP, then for the given depth d of the to-be-finalized block and any time τ there exists a γ_0 such that for all $\gamma \geq \gamma_0$ we have that there is a time period of length δ_{freeze} where there is a unique justified proposal and all honest parties will have that proposal on their path. This implies by the weak validity and termination property of WMVBA that eventually $\mathsf{Finalization}$ will terminate with a new finalized block $\mathtt{lastFinal}$ at depth d.

It remains to show that the protocol achieves Δ-updated. For this we give an upper bound on the gap between the last finalized block and the maximal depth of an honest position. Ideally, the depth d of the last finalized block is roughly the maximal depth of an honest position. We observe that there are two cases where the depth of the last finalized block can lag behind. In the first case, finalization itself fell behind the tree growth. In the second case, the depth d' of the next finalized block is set larger than the current maximal depth of an honest position. Until some honest party reaches depth d, no new finalized block is created. Thus, the gap to the last finalized block can grow up to $d' - d$. To achieve an upper bound on the gap, we proceed as follows. We first give a bound for the first case and then use this to bound the gap in the second case.

We assume an upper-bound on chain-growth. This implies that within one finalization attempt the chain-growth is bounded by some constant. Let d be the depth of the next finalized block. Then we assume that the positions of honest parties grow at most b in depths, between the time some party first reaches depth d (start of finalization) and the time the last honest party receives a witness for the finalization of a block at depth d (end of finalization).

For the first case consider the following situation. A block at depth d was just finalized and the maximal depth of an honest position is $d + x$. In other words, finalization lags behind and we have a gap of x blocks. As it lags behind, finalization will try to catch up by doubling ℓ (cf. Lemma 1). However, as long as $\ell < b$, the gap will increase. In the worst-case the initial ℓ is 1. After $\lceil \log_2(b) \rceil$ finalizations we have $\ell \geq b$, and during each finalization, the gap can increase by at most b. The gap in the catch-up phase can therefore be loosely bounded by $x + \lceil \log_2(b) \rceil \cdot b$.

To get a bound for the first case, it remains to show that x is bounded as well. It is enough to bound x for the case where ℓ after the finalization at depth d was decreased (or stayed at the minimum of $\ell = 1$). Otherwise, the finalization at depth d is part of a larger catch-up phase and we just consider the x' and d' at the beginning of this larger phase. If $d = 1$, i.e., at the beginning of the protocol, we have $x \leq 1$. Otherwise, we know that at the point when d was selected as the next finalization depth, the maximal depth of an honest position was less than d. So at the start of the finalization for depth d, the maximal depth of an honest position was d. Thus, at the end of the finalization this depth was at most $d + b$, and thus $x \leq b$. We conclude that in the first case the gap is bounded by $b + \lceil \log_2(b) \rceil \cdot b$.

Finally consider the second case. Again, let d be the depth of the block which was just finalized. Assume that the depth $d' = d + \ell$ of the next finalized block is set larger than the current maximal depth of an honest position. The gap in this case is bounded by $d' - d = \ell$. So we need to bound ℓ. Note that ℓ is maximal after it has been increased for the last time. According to Lemma 1, the value of ℓ gets decreased if the next finalized block is at least $c \coloneqq \rho_{\texttt{pgrowth}} \cdot \Delta_{\texttt{net}} + \Delta_{\texttt{pgrowth}} + \ell_{\texttt{PQ}}$ blocks deeper than the positions of all honest parties. By the analysis in the first case, we are never behind more than $b + \lceil \log_2(b) \rceil \cdot b$ blocks. This means that as soon as ℓ reaches the value $b + \lceil \log_2(b) \rceil \cdot b + c$, it can be double at most once more before it will be reduced. Hence, we obtain an overall gap bound of $\Delta \coloneqq 2(b + \lceil \log_2(b) \rceil \cdot b + c)$.

□

Finalization with Filtered Byzantine Agreement.

Theorem 5. *For $t < \frac{n}{3}$, and a blockchain satisfying common-prefix there exists a Δ such that the protocol described in Section 5 where calls to* WMVBA *replaced by calls to* FilteredWMVBA *satisfies $(\Delta, 1)$-finality.*

Proof. The agreement and chain-forming properties follow as in the proof of Theorem 4 as FilteredWMVBA as 1-support. Similarly, the 1-support property follows directly from the 1-support of FilteredWMVBA.

It remains to check the Δ-updated property. First, we show that any invocation of Finalization eventually terminates. As the blockchain satisfies chain growth we know that all honest parties will eventually start FilteredWMVBA in each execution of the repeat until loop of Finalization. Note also that all honest parties have a justified input to WMVBA so it will terminate. The parties will exit the loop if WMVBA outputs a non-$\perp$ decision. By the common-prefix property of the underlying blockchain and the increasing γ in Finalization, all honest parties will eventually input the same justified proposal. This implies by the validity and termination property of FilteredWMVBA that eventually Finalization will terminate with a new finalized block `lastFinal` at depth d. The actual Δ-updated property follows as in the proof of Theorem 4. □

8 Committee Selection

The protocol of Section 5 is (intentionally) described in a simplified setting that abstracts away many aspects of the underlying blockchain. We stress, however, that our goal is to present a finality layer, and *not* a full-fledged blockchain. Therefore, the reason for considering a simplified setting is that by abstracting the properties of the underlying blockchain, we end up with a protocol that is generic enough to be used in tandem with virtually *any* Nakamoto-style blockchain. Hence, if the underlying blockchain has properties such as permissionlessness and dynamic stakes then our protocol can preserve those properties.

Properly selecting a committee is a challenging task that has been extensively studied [28, 20, 21]. The appropriate strategy to select a finalization committee is highly tied to the specific type of blockchain one considers. Therefore, it is out of the scope of this paper to propose a definitive answer on how to select a committee for each particular setting. We do however discuss some possible approaches that can be used to select the finalization committee in a few settings. We can categorize committees into two main categories, namely *external committees* and *chain-based committees*. We discuss both next.

8.1 External Committees

An external committee is usually selected prior to the deployment of the system, and can be dynamic or static during the lifetime of the system. External committees are more common in *permissioned* blockchain applications, where there are restrictions to parties joining the system. As an example, consider a blockchain backed by a foundation (e.g., Ethereum); the selection of the committee to run the finality layer can be initially chosen by the foundation, perhaps among a few nodes that are previously registered with the foundation to perform the task. The committee can be later updated during the lifetime of the system; the only requirement is that the corrupted nodes compose less than 1/3 of the total number parties. We stress that allowing a *permissioned* committee for the finality layer does not make the protocol trivial or the result any weaker; in fact, it shows that our protocol is flexible enough to allow virtually all types of blockchains to take advantage of finality capabilities.

8.2 Chain-Based Committees

Chain-based committees are deterministically selected from the blockchain itself. To abstract the selection procedure from the underlying blockchain we define an interface $C = \texttt{FC}(\texttt{Tree}, B)$ for a function that takes as input the current blocktree `Tree` and a block B and selects a committee C. The actual function `FC` is implemented by the underlying blockchain and can select the committee in an arbitrary way; e.g., read all the state data in the previous epoch plus any auxiliary information that might have been added via a survey layer (e.g., live nodes information), and from this data select the committee C including each party's stake. It is important to note that the committee C is deterministically selected from $B \in \texttt{Tree}$ and the path from B to the genesis block. The committee C is selected by invoking the function `FC` and passing the current tree `Tree` and the last finalized block B as input.

Chain-based committees in PoW. In a PoW blockchain miners employ computational power to solve a hash puzzle to eventually get the right to append a new block to the chain. By inspecting the history of mined blocks, one can infer the proportion of computational power each party (or public key) possess in relation to the overall system within some time period. To select a committee in a PoW chain, one could employ similar techniques from [28, 20, 21] and consider a sliding time window (e.g., last 1000 blocks) that ends just before the last final block (initially one can consider a pre-selected committee in the genesis block), where the miners within the time window would constitute the committee. Note that the previously described committee selection strategy assumes synchronicity. This, however does *not* imply that Afgjort is synchronous, but rather that it supports many different strategies of committee selection.

Chain-based committees in PoS. To instantiate the function `FC` for PoS blockchains one could use the data and stake distribution from the chain and run the committee selection as a VRF lottery using the party's stake as the "lottery tickets", as is done in Algorand [23] and Ouroboros Praos [11]; the more stake one has the higher is the probability of being selected.[3] A similar approach would be to run the selection based on the party's stake by using randomness produced by a coin tossing protocol ran by all the online parties in the previous epoch, as is done in Ouroboros [19].

9 Acknowledgements

We would like to thank Mateusz Tilewski for countless discussions during the design of the finality layer, his deep insights into practical distributed systems were valuable in designing a system which is at the same time efficient in practice and provably secure. We would like to thank Matias Frank Jensen and Emil Morre Christensen. Their work on generalising the Finality layer gave valuable insights which were adapted into the protocol presented in this paper.

References

[1] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley Series on Parallel and Distributed Computing. Wiley, 2004.

[2] Michael Backes and Dennis Hofheinz. How to break and repair a universally composable signature functionality. In Kan Zhang and Yuliang Zheng, editors, *ISC 2004*, volume 3225 of *LNCS*, pages 61–72. Springer, Heidelberg, September 2004.

[3] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *ACM CCS 18*, pages 913–930. ACM Press, 2018.

[4] Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *3rd ACM PODC*, pages 154–162. ACM, August 1984.

[5] Eric Brewer. Towards robust distributed systems, 7 2000. Invited talk at Principles of Distributed Computing.

[6] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. Master's thesis, The University of Guelph, Guelph, Ontario, Canada, 6 2016.

[7] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.

[8] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541. Springer, Heidelberg, August 2001.

[3] Note that the voting "power" of selected committee members can also be based on their respective stakes.

[9] R. Canetti. Universally composable signature, certification, and authentication. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004*, pages 219–233, 6 2004.

[10] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. *IACR Cryptology ePrint Archive*, 2018:981, 2018.

[11] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018.

[12] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[13] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.

[14] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 291–323. Springer, Heidelberg, August 2017.

[15] Peter Gazi, Aggelos Kiayias, and Alexander Russell. Stake-bleeding attacks on proof-of-stake blockchains. In *Crypto Valley Conference on Blockchain Technology, CVCBT 2018, Zug, Switzerland, June 20-22, 2018*, pages 85–92, 2018.

[16] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68, 2017.

[17] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 6 2002.

[18] Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. Cryptology ePrint Archive, Report 2015/1019, 2015. `http://eprint.iacr.org/2015/1019`.

[19] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.

[20] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 279–296, 2016.

[21] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy*, pages 583–598. IEEE Computer Society Press, May 2018.

[22] Jae Kwon. Tendermint: Consensus without mining. Technical report, 2014.

[23] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[24] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.

[25] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[26] Gil Neiger. Distributed consensus revisited. *Information Processing Letters*, 49(4):195 – 201, 1994.

[27] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, April / May 2017.

[28] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 39:1–39:16, 2017.

[29] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 3–33. Springer, Heidelberg, April / May 2018.

[30] Russell Turpin and Brian A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Inf. Process. Lett.*, 18(2):73–76, 1984.