Afgjort – A Semi-Synchronous Finality Layer for Blockchains[*]

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

[1]Aarhus University, Aarhus, Denmark
[2]Concordium, Zurich, Switzerland

May 22, 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*). We also put forward an argument why finality layers should be asynchronous or semi-synchronous.

As technical contributions, we propose two variants of a finality layer protocol. We prove both of them secure in the setting with $t < n/3$ Byzantine parties and a semi-synchronous network. 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 other one needs an additional, mild assumption on the underlying blockchain, but is more efficient and satisfies $k = n/3$-support. We finally show that $t < n/3$ is optimal for semi-synchronous finality layers.

[*]This work was supported by Concordium Blockchain Research Center, Aarhus University, Denmark.

Contents

1 Introduction

In classical blockchains such as Bitcoin [22], the parties that win some sort of lottery get the right to append blocks to a chain. Due to network delays, what may cause parties to append new blocks without knowing about other blocks already appended to the chain by other parties or adversarial behavior, the chain can fork and become a tree. For that case, parties have some 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 that block can be considered "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 with probability $1/3^k$ win k times in a row and thereby cause a rollback of length k. This means that to limit the rollback probability to 2^{-80}, one needs to wait for at least 50 blocks. Taking Bitcoin as an example, where a new block is generated roughly every 10 minutes, this results in waiting for more than 8 hours. Considering more sophisticated attacks and unclear network conditions, an even longer waiting time would be necessary in practice.

A main reason for the 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 50 blocks in a row, but if you want 2^{-80} security against a 33% adversary, you constantly have to behave as if it just happened.

We propose a finality layer that can be composed with any synchronous or semi-synchronous blockchain (cf., [10, 23, 8]) that has the property of common prefix (as defined by [10]). 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, essentially turning the new final blocks into genesis blocks. 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 in the common-prefix it should quickly become final. In this sense, the finality layer should be responsive: if blocks quickly become agreed upon, they also quickly become final. Once a block is final, honest parties will never do a rollback behind this block.

Communication model. We assume a *semi-synchronous* network model [9, 8]. This means that there is an upper bound Δ_{NET} on the network delay, but in contrast to the synchronous model, this bound is not known to the protocol. In particular, Δ_{NET} cannot be used in a semi-synchronous protocol. This can be modeled by letting the adversary pick Δ_{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 semi-synchronous protocol runs fast if the network is fast as it cannot depend on the unknown upper bound, only the actual network delay. As we discuss in the following paragraph, 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 semi-synchronous model, where Δ_{NET} can just be set to be larger than the duration of such partitions.

Finality layers versus pure BFT consensus. Committee-based Byzantine fault tolerant (CBFT) consensus designs such as the ones employed by Tendermint [19, 3] and Algorand [20, 13] provide immediate finality, i.e., every block that makes it into such a blockchain can be considered final. Such a blockchain obviously does not need a finality layer. There are several reasons why it still makes sense to combine a Nakamoto-style blockchain with a finality layer, instead of directly using a CBFT 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.
- Nakamoto-style consensus, especially its proof-of-stake variant, can produce blocks much faster than CBFT consensus. This can be obtained by, for instance, setting the block-time aggressively. For such high-throughput blockchains it can be hard to give a reasonable good worst-case bound on common-prefix. But if you add to them a semi-synchronous finality layer, you turn them into fast blockchains with responsive finality. Another view on this is that if you know you are designing a blockchain that will be given a finality layer, you can allow yourself to make much more aggressive designs. Your 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. You need not even know this bound.
- 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 1500 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 for instance 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 [2, 14] 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 partition. 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 turn off 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. [24]). 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 34% of the parties are corrupted. On the other hand, some blockchain designs tolerate 49% corruption. Some designs in fact tolerate periodic corruption of $> 51\%$ if only over long enough periods the average corruption is below 50%. 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 49% 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 gradual degradation of security and recovery from such catastrophic events. Having a modular two-layer design dividing the responsibility of eventual agreement and finality seems to facilitate this study.

1.1 Our Techniques

We assume that the path from the genesis block up to any finalized block deterministically defines who makes up the next *finality committee*. This could for instance be by looking at recent block winners, or by the result of some lottery being reported on the chain. This finality committee will be 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. This d is also deterministically determined by the blocks from genesis up to the last finalized block. The initial committee is defined in the genesis block.

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. This is enforced by modifying the best chain rule to always prefer the chain with the most final blocks. 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: the protocol is semi-synchronous. The rational behind this is twofold. It gives responsive finality: when the common-prefix value is low, we finalize quickly. It also makes the finality layer work as a hedge against catastrophic events.

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. Note that when a party votes for a block at depth d, we can 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. We want that the finalized blocks do not fall behind the underlying blockchain too much. We call this the *updated* property of the finality layer. 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 security properties of finality layers.

The most closely related work seems to be Casper [4], 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 semi-synchronous model. In particular, the authors only consider what they call "plausible liveness", but there is no guarantee that liveness actually holds.

We do not present a blockchain. Our finality layer needs to run on top of a secure blockchain to achieve consensus. Similarly, existing consensus protocols are not finality layers. Yet, it is instructive to compare our finality layer to existing consensus protocols, in particular Algorand [20, 13], Thunderella [25], and Hybrid Consensus [24].

The consensus protocol closest to ours is Hybrid Consensus by Pass and Shi, and the closely related Thunderella. 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. 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. 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 an irreparable fork. An adversary can try to provoke this by turning all his stake active at the time of a spike in online honest stake and mounting a DDoS attack on the network in the same moment. Notice that in our design the previous committee "approves" the next committee, as we assume that the next committee can be determined deterministically from the previous final block, which was uniquely chosen by the previous committee. This Algorand cannot do, as the next committee by design is not known until they speak. We have chosen the design of "approving" the next committee to get finality independently of how fast honest participation fluctuates, as we did not want to make too many assumptions on the underlying blockchain. The price is to open up for DDoS attacks. It seems like an interesting open problem to get a player replaceable CBFT which can simultaneously tolerate rapid fluctuations in honest participation and network partitions.

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. 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 assume they drift from τ by at most some known bound Δ_{TIME}. For the sake of simpler proofs we will pretend in proofs that Δ_{TIME}. The proofs trivially adapt to the case of a known $\Delta_{\text{TIME}} > 0$.

For simplicity, we assume that there is a fixed set of parties $\mathcal{P}$ with $n := |\mathcal{P}|$, 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 semi-synchronous model, which means that there is a hidden bound Δ_{NET} on message delays. In contrast to synchronous networks, Δ_{NET} is not known, i.e., the protocols cannot use Δ_{NET}, they can only assume the existence of some bound. One can think of Δ_{NET} as being chosen by the adversary at the beginning of the protocol execution, after the protocol has been fixed. 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 $[\tau, \tau + \Delta_{\text{NET}}]$.
- When an honest party receives a message at time τ (possibly sent by a dishonest party), all honest parties receive this message until time $\tau + \Delta_{\text{NET}}$.

We model the network using an ideal functionality. The adversary inputs the delay to the ideal functionality in unary, i.e., as $1^{\Delta_{\text{NET}}}$. After that, whenever a party sends a message, it is given to the adversary, and the adversary can instruct the network to deliver messages as long as it maintains the above timing restrictions.

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 out 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 [6, 1]. 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 $(v_1, \dots, v_k)$ of distinct nodes such that $\{v_i, v_{i+1}\} \in E$ for all $i \in \{1, \dots, k-1\}$.

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 $\text{PathTo}(T, v)$. A *leaf* is a node $v \in V \setminus \{r\}$ that occurs in only one edge. We further let $\text{Depth}(T, v)$ be the length of $\text{PathTo}(T, v)$ and $\text{Height}(T, v)$ be the length of the longest path from v to a leaf. When the tree is clear from context, we may also write $\text{PathTo}(v)$, $\text{Depth}(v)$, and $\text{Height}(v)$. The height of a tree equals the height of its root: $\text{Height}(T) \coloneqq \text{Height}(T, r)$ (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., [4]. 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 $\mathrm{PathTo}(T, v)$, 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 $G_1 \cup G_2 \coloneqq (V_1 \cup V_2, E_1 \cup E_2)$, and their intersection as $G_1 \cap G_2 \coloneqq (V_1 \cap V_2, E_1 \cap E_2)$. For two rooted trees $T_1 = (G_1, r)$, $T_2 = (G_2, r)$ with common root r, we define $T_1 \cup T_2 \coloneqq (G_1 \cup G_2, r)$ and $T_1 \cap T_2 \coloneqq (G_1 \cap G_2, r)$.

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. We use the UC framework [7]. The properties are modelled via an ideal functionality $\mathcal{F}_{\text{TREE}}$, to which all parties have access.

3.1 Description of Tree Functionality

We will not describe an ideal functionality in full detail, as the exact details do not matter for the proof. 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 Tree_i of the parties P_i. We also give party P_i access to a GETTREE command which returns the current Tree_i. There are additional commands covered below. We also use $\mathcal{F}_{\text{TREE}}$ to model the network, but will not go into depth of the commands for the network here.

The ideal functionality also maintains a time τ. It is initially 0. It can be incremented by 1 be the adversary giving the command INCREMENT. The time cannot be seen by the parties. It is for internal accounting. Note that the adversary needs to use runningtime to increment the time by 1, so the time will be polynomial.

We here describe $\mathcal{F}_{\text{TREE}}$, which 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 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 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

$$\mathsf{HonestTree} \coloneqq \cup_{P_i \in \mathsf{Honest}} \mathsf{Tree}_i$$

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 $\mathsf{HonestTree}$ remains a tree at all times.

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

Recall that for a node B in Tree_i, $\mathrm{PathTo}(\mathsf{Tree}_i, B)$ denotes the (unique) path from the root to B. We define $\mathsf{Path}_i := \mathrm{PathTo}(\mathsf{Tree}_i, \mathsf{Pos}_i)$. In a typical blockchain protocol, 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.

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 $(\text{SETFINAL}, R)$ once this party considers R to be final. More formally, each party has a variable $\mathsf{lastFinal}_i \in \mathsf{Tree}_i$, initially set to the genesis block G. The command $(\text{SETFINAL}, R)$ for $R \in \mathsf{Tree}_i$ sets $\mathsf{lastFinal}_i$ to R. Inputs $(\text{SETFINAL}, R)$ by P_i where R is not a descendant of $\mathsf{lastFinal}_i$ 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 $\mathsf{FinalTree}_i$ to be the tree consisting of all paths in Tree_i going through $\mathsf{lastFinal}_i$. Note that this consists of only a single path from G to $\mathsf{lastFinal}_i$ and then possibly a proper tree below $\mathsf{lastFinal}_i$. We let $\mathsf{FinalTree} = \cup_{P_i \in \mathsf{Honest}} \mathsf{FinalTree}_i$.

The command $(\text{PROPDATA}, \mathsf{data})$ allows parties to propose some $\mathsf{data} \in \{0,1\}^*$ 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.

We conclude with a formal specification of the functionality $\mathcal{F}_{\text{TREE}}$.

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

Initialization

```
for $P_i \in \mathcal{P}$ do
    $\mathsf{Tree}_i := ((V_i := \{G\}, E_i := \varnothing), r_i := G)$
    $\mathsf{Pos}_i := G$, $\mathsf{lastFinal}_i := G$, $\mathsf{lastProp}_i := \bot$
end for
```

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

Input: GETTREE

```
    return copy of $(\mathsf{Tree}_i, \mathsf{Pos}_i)$ to $P_i$
Input: $(\textsc{setfinal}, R)$
    if $\mathsf{lastFinal}_i \in \mathrm{PathTo}(\mathsf{Tree}_i, R)$ then
        $\mathsf{lastFinal}_i := R$
        send $(\textsc{setfinal}, P_i, R)$ to adversary
    end if
Input: $(\textsc{propdata}, \mathsf{data})$
    $\mathsf{lastProp}_i := \mathsf{data}$
    send $(\textsc{propdata}, P_i, \mathsf{data})$ to adversary

Interface for adversary
Input: $(\textsc{addNode}, P_i, B, p)$ // add $B$ as child of $p$ in $\mathsf{Tree}_i = ((V_i, E_i), r_i)$
    if $B \notin V_i$ and $\mathsf{HonestTree}$ remains a tree after adding $B$ as child of $p$ in $\mathsf{Tree}_i$ then
        $V_i := V_i \cup \{B\}$
        $E_i := E_i \cup \{\{p, B\}\}$
        if $p = \mathsf{Pos}_i$ then
            $\mathsf{Pos}_i := B$
        end if
    end if
Input: $(\textsc{setPosition}, P_i, B)$ // set position of $P_i$ to $B$
    if $B$ is a leaf of $\mathsf{Tree}_i$ then
        $\mathsf{Pos}_i := B$
    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. Formally, we require that the adversary inputs the bounds in unary to the ideal functionality before it interacts with the ideal functionality in any other way.

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 [11, 15, 23]. 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 that proposals of honest parties make it into blocks, which is closely related to chain quality.

Tree propagation. There is a hidden bound Δ_{TREE} such that for all honest parties $P_i \in \mathsf{Honest}$, $\mathsf{HonestTree}^{\tau-\Delta_{\text{TREE}}} \subseteq \mathsf{Tree}_i^{\tau}$. 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 $\Delta_{\textsc{final}}$, that intuitively is the time that it takes for a SETFINAL command to take effect. We require that $R \in \mathsf{Path}_i$ after $\Delta_{\textsc{final}}$ 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 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 Prefix(ξ):

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

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 Δ_{growth} at least at rate ρ_{growth} and at most at rate ρ'_{growth}, $0 < \rho'_{\mathsf{growth}} \leq \rho_{\mathsf{growth}}$. Note that the chain of party P_i in our model corresponds to Path_i and its length is equal to $\mathrm{Depth}(\mathsf{Pos}_i)$. We thus use the following formalization:

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

Remark 4. Some papers, e.g., [23, 8] 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. 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 adversary could let the blockchain 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.

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 $\ell_{\mathrm{PQ}} \in \mathbb{N}$ means that at any time τ, for all honest $P_i \in \mathsf{Honest}$, and for all ℓ_{PQ} consecutive blocks $B_1, \ldots, B_{\ell_{\mathrm{PQ}}}$ in Path_i^{τ}, there exists a block $B' \in \{B_1, \ldots, B_{\ell_{\mathrm{PQ}}}\}$ that was added to $\mathsf{HonestTree}$ at time τ' and an honest party $P_j \in \mathsf{Honest}$ such that $\mathsf{lastProp}_j^{\tau'}$ is contained in (the data field of) a block on $\mathrm{PathTo}(\mathsf{HonestTree}^{\tau'}, B')$. In other words, at the time B' is added to $\mathsf{HonestTree}$, 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 $\ell_{\mathrm{PQ}} \geq \ell'$ with $\ell_{\mathrm{PQ}} \cdot \mu \geq 1$, at least one block within ℓ_{PQ} consecutive blocks is generated by an honest party. Whenever honest parties add a block B', they can check whether their last proposed data is already contained in a previous block, and if not, they include that data in B'. This yields proposal quality with parameter ℓ_{PQ}.

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

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

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

Δ-Updated: At any time τ, we have

$$\text{Height}(\mathsf{HonestTree}^\tau) - \min_{P_i \in \mathsf{Honest}} \text{Depth}(\mathsf{lastFinal}_i^\tau) \leq \Delta.$$

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

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.[12] 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.[12] For example, Ouroboros [16] and Ouroboros Praos [8] 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 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. [12] 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

Note that our requirements on the finality layer are given as some properties on how it uses $\mathcal{F}_{\text{TREE}}$. What we will prove later is that our finality layer has these properties except with negligible probability. We furthermore show that if the hidden bounds of $\mathcal{F}_{\text{TREE}}$ is a polynomial, so is for instance the hidden bound Δ in Δ-updated. This shows that if the underlying blockchain has "polynomial liveness", so does the blockchain resulting from adding our finality layer.

We here discuss briefly how to model the 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.

With a few modifications given below, we model UC security of a finality layer by requiring that it UC securely implements the following ideal functionality $\mathcal{F}_{\text{FINTREE}}$, where we drop the payload data of blocks for brevity. The payload was used only for implementation purposes of the finality layer.

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

```
Initialization
  for $P_i \in \mathcal{P}$ do
    $\mathsf{Tree}_i := ((V_i := \{G\}, E_i := \varnothing), r_i := G)$
    $\mathsf{Pos}_i := G$, $\mathsf{lastFinal}_i := G$
  end for

Interface for party $P_i \in \mathcal{P}$
Input: GETTREE
  return copy of $(\mathsf{Tree}_i, \mathsf{lastFinal}_i, \mathsf{Pos}_i)$ to $P_i$

Interface for adversary
Input: $(\text{ADDNODE}, P_i, B, p)$ // add $B$ as child of $p$ in $\mathsf{Tree}_i = ((V_i, E_i), r_i)$
  if $B \notin V_i$ and HonestTree remains a tree after adding $B$ as child of $p$ in $\mathsf{Tree}_i$ then
    $V_i := V_i \cup \{B\}$
    $E_i := E_i \cup \{\{p, B\}\}$
    if $p = \mathsf{Pos}_i$ then
      $\mathsf{Pos}_i := B$
    end if
  end if

Input: $(\text{SETPOSITION}, P_i, B)$ // set position of $P_i$ to $B$
  if $B$ is a leaf of $\mathsf{FinalTree}_i$ then
    $\mathsf{Pos}_i := B$
    end if
**Input:** $(\textsc{IsFinal}, P_i, R)$
    if $\mathsf{lastFinal}_i \in \mathrm{PathTo}(\mathsf{Tree}_i, R)$ and $R \in \mathrm{PathTo}(\mathsf{Tree}_i, \mathsf{Pos}_i)$ then
        $\mathsf{lastFinal}_i := R$
    end if
```

Note that $\mathcal{F}_{\textsc{FinTree}}$ has taken finality out of the hand of the user and put it into the hand of the adversary. The adversary is free to announce any block as final, as long as it promises to keep the party in a position under that final block for all future times. In addition the adversary needs to update the trees, positions, and final blocks according to the desirable properties of $\mathcal{F}_{\textsc{Tree}}$ and the following additional properties.

Definition 6. Let $\Delta, k \in \mathbb{N}$.

Chain-forming: If the adversary inputs $(\textsc{IsFinal}, P_i, R)$ at time τ, we have $\mathsf{lastFinal}_i \in \mathrm{PathTo}(\mathsf{Tree}_i^\tau, R)$ and $R \neq \mathsf{lastFinal}_i$.

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

Δ-Updated: At any time τ, we have

$$\mathrm{Height}(\mathsf{HonestTree}^\tau) - \min_{P_i \in \mathsf{Honest}} \mathrm{Depth}\big(\mathsf{lastFinal}_i^\tau\big) \leq \Delta.$$

Here Δ is a hidden parameter of $\mathcal{F}_{\textsc{FinTree}}$ set by the adversary.

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

Given a finality layer we can construct a protocol Π_{FIN} for the $\mathcal{F}_{\textsc{Tree}}$-hybrid model implementing $\mathcal{F}_{\textsc{FinTree}}$ as follows. It runs the finality layer on top of $\mathcal{F}_{\textsc{Tree}}$. Whenever the finality layer updates $\mathsf{lastFinal}_i$ party P_i saves the old value in $\mathsf{lastFinal}'_i$ and records the new value in $\mathsf{lastFinal}_i$. On $\textsc{getTree}$, call $\textsc{getTree}$ on $\mathcal{F}_{\textsc{Tree}}$ and get $(\mathsf{Tree}_i, \mathsf{Pos}_i)$. If $\mathsf{lastFinal}_i \in \mathrm{PathTo}(\mathsf{Tree}_i, \mathsf{Pos}_i)$, then return $(\mathsf{Tree}_i, \mathsf{lastFinal}_i, \mathsf{Pos}_i)$. Otherwise, return $(\mathsf{Tree}_i, \mathsf{lastFinal}'_i, \mathsf{Pos}_i)$. The reason for the slight complication here is that in $\mathcal{F}_{\textsc{Tree}}$, it might take a while for the new root to take effect. On $\mathcal{F}_{\textsc{FinTree}}$ we therefore only announce the new root as final once we find ourselves in a position under it.

We model UC security of a finality layer by requiring that $\Pi_{\text{FIN}}[\mathcal{F}_{\textsc{Tree}}]$ UC securely realizes $\mathcal{F}_{\textsc{FinTree}}$. Notice that in $\mathcal{F}_{\textsc{FinTree}}$ all inputs to all honest parties are given to the adversary/simulator. So we can construct a UC simulator simply by running $\Pi_{\text{FIN}}[\mathcal{F}_{\textsc{Tree}}]$ on the real inputs of $\mathcal{F}_{\textsc{FinTree}}$. The simulator updates the variables $\mathsf{Tree}_i, \mathsf{lastFinal}_i, \mathsf{Pos}_i$ in $\mathcal{F}_{\textsc{FinTree}}$ to have exactly the values they have in the execution of $\Pi_{\text{FIN}}[\mathcal{F}_{\textsc{Tree}}]$ in the simulation. This gives a *perfect* simulation as long as $\mathcal{F}_{\textsc{FinTree}}$ allows the simulator to update $\mathsf{Tree}_i, \mathsf{lastFinal}_i, \mathsf{Pos}_i$ as needed. It can be seen that $\mathcal{F}_{\textsc{FinTree}}$ allows the simulator to update $\mathsf{Tree}_i, \mathsf{lastFinal}_i, \mathsf{Pos}_i$ as needed exactly as long as the finality layer has the desired properties.

Note that the simulator also has to set the hidden parameters of $\mathcal{F}_{\textsc{FinTree}}$. It in particular has to set the hidden parameter Δ in the Δ-Updated property. This parameter has to be input as 1^Δ before the protocol starts executing. This puts two requirements on the simulator: 1) it ha be be able to compute Δ before the protocol starts executing, and 2) it needs to have running time Δ available to write the string 1^{Δ}. Note now that the simulator in the simulated execution of $\Pi_{\text{FIN}}[\mathcal{F}_{\text{TREE}}]$ sees how the hidden parameters Δ' of $\mathcal{F}_{\text{TREE}}$ are set. These parameters are input to the simulator as $1^{\Delta'}$. It can then be seen that it suffice to show for a given finality layer that there is some Δ-Updated property and that Δ can be computed as a fixed polynomial of the hidden parameters Δ' of $\mathcal{F}_{\text{TREE}}$. The reason this is enough is that it 1) allows the simulator to compute Δ before the protocol starts executing and that 2) in the UC model an input like $1^{\Delta'}$ gives the simulator an additional running time budget of Δ' and the total running time of the simulator is allowed to be a fixed polynomial of its budget. We can therefore set the running time polynomial of the simulator to be larger than the one needed to compute the parameters Δ of $\mathcal{F}_{\text{FINTREE}}$ from the parameters Δ' of $\mathcal{F}_{\text{TREE}}$.

Moving forward it is therefore sufficient to prove that the finality layer has the desired properties except with negligible probability and that if the hidden bounds of $\mathcal{F}_{\text{TREE}}$ are polynomials, so are the hidden bounds of $\mathcal{F}_{\text{FINTREE}}$, and they are related by some fixed polynomial. We will throughout the paper derive explicit bounds on protocols from the assumed bound on their sub-protocols. It will be immediately clear from these that they polynomially relate the produced bounds from the assumed bounds, so we will not explicitly note this moving forward. As an example, consider the bound γ_0 derived in (1) in the proof of Lemma 1.

A final remark on the bounds. In some case we prove only a bound on the expected value of a bound Δ, in particular for our randomized Byzantine agreement protocols. In all case a worst-case bound can be derived by multiplying by the security parameter. This gives a bound which will hold except with negligible probability, which is enough to prove UC security. If the actual running time exceeds the bound, the simulator give up the simulation, but this happens only with negligible probability.

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 $t \geq n/3$ semi-synchronous blockchain that satisfies the properties of a finality layer is void.

.

Theorem 1. *A semi-synchronous blockchain (satisfying the properties of liveness and persistence [10]) for n parties that satisfies the properties of finality (Section 4) must have $t < n/3$.*

Proof. We prove this by contradiction. We first define a simple distributed problem called unscheduled broadcast (UB) and show that it cannot be solved in semi-synchronous network for $n = 3$ parties and $t = 1$ corruptions. We then show that if we are given a semi-synchronous blockchain (satisfying the properties of liveness and persistence [10]) that satisfies the properties of finality (Section 4) then we can solve the UB problem. The proof generalizes to any n and $t = n/3$.

The UB problem for P_1, P_2, P_3 is defined as follows. At some point in time P_1 receives as input a bit b_1 that it is to broadcast. We assume that P_1 immediately outputs b_1, so we also call b_1 the output of P_1. At some point in time P_i, for $i = 2, 3$, might output a bit b_i. Agreement: If both P_i and P_j are honest and they output b_i and b_j, then $b_i = b_j$. Liveness: If P_1 and P_i, for $i \in \{1, 2\}$, is honest and P_1 receives input b_1, then eventually P_i outputs some bit b_i. The network is semi-synchronous, i.e., there is a hidden bound Δ on the network delivery time. The delay is picked adversarially after the protocol is designed.

It is clear that we can solve UB given a semi-synchronous blockchain. Party P_1 will send a signed b_1 on the blockchain. If a party P_i, for $i = 2, 3$, sees a b_1 on the blockchain behind a finalized block, it will output $b_i = b_1$.

We now show that one cannot solve UB in a semi-synchronous network. Let π be a protocol solving UB. Set $\Delta = 1$. Let T be an upper bound such that when running with network delay Δ and input b_1 to P_1 at the very beginning of the execution, the protocol π will terminate in time T except with negligible probability. This T exists by the assumption of liveness above. Now consider running π with network delay $\Delta' = T + 1$ and input b_1 to P_1 at the very beginning of the execution. But schedule the message delivery such that all messages are still delivered in time Δ. It follows that the protocol still terminates in time T on any such scheduling, as π does not know Δ', it can only depend on the actual network delivery times, which are consistent with Δ.

Now consider a protocol where P_1 has input 0 and where $P_1(0)$ and P_2 are honest and P_3 is crashed. Deliver all message in time Δ. Let $\langle P_1(0), P_2\rangle$ denote the output b_2 of P_2 in such an execution. By agreement we have that $\langle P_1(0), P_2\rangle = 0$. The output occurs before time T.

Now consider a protocol where P_1 has input 1 and where $P_1(1)$ and P_3 are honest and P_2 is crashed. Deliver all message in time Δ. Let $\langle P_1(1), P_3\rangle$ denote the output b_3 of P_3 in such an execution. By agreement we have that $\langle P_1(1), P_3\rangle = 1$. The output occurs before time T.

Now consider an execution where P_2 and P_3 are honest and where P_1 is corrupt. We let P_1 act as $P_1(0)$ towards P_2 , and we let P_1 act as $P_1(1)$ towards P_3 . We delay all messages between P_2 and P_2 for more than time T. This is allowed as $\Delta' = T + 1 > T$. The output of P_2 will be exactly the same as in $\langle P_1(0), P_2\rangle$ as it interacts with $P_1(0)$ and receives no messages from P_3 before it gives its output b_2: it gives output at time T and all messages are delayed for time $T + 1$. Similarly the output of P_3 will be exactly the same as in $\langle P_1(0), P_3\rangle$. Hence $b_2 = 0$ and $b_3 = 1$, violating 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 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 $J^{d,\gamma}_{\text{InTree}}$ where the value v is a block.

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

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 $n - t$ 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 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:
 1: Set $\gamma := 1$, $d := 6$, and $\ell := 5$
 2: for ctr $= 1, 2, 3, \dots$ do
 3:     Set faid := (sid, ctr)
 4:     Run $(R, \mathtt{W}, \gamma') :=$ Finalization(faid, $J^{d,\gamma}_{\textsc{InTree}}, d, \gamma$)
 5:     Invoke (SETFINAL, $R$)
 6:     Invoke (PROPDATA, W)
 7:     Set $\ell :=$ NextFinalizationGap($\mathsf{lastFinal}_i, \ell$)
 8:     Set $d := d + \ell$
 9:     Set $\gamma := \lceil 0.8 \cdot \gamma' \rceil$
10: 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 for which agreement was reached. 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($\texttt{faid}, J^{d,\gamma}_{\text{InTree}}, d, \gamma$)

Party P_i does the following:

```
1: repeat
2:     Set $\texttt{baid} := (\texttt{faid}, \gamma)$
3:     Wait until $\mathsf{lastFinal}_i$ is on $\mathsf{Path}_i$ and $\mathsf{Path}_i$ has length at least $d + \gamma$
4:     Let $B_d$ be the block at depth $d$ on $\mathsf{Path}_i$
5:     Run $(R, \texttt{W}) := \mathsf{WMVBA}(\texttt{baid}, J^{d,\gamma}_{\text{InTree}})$ with input $B_d$
6:     if $R = \perp$ then set $\gamma := 2\gamma$ end if
7: until $R \neq \perp$
8: 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.

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 AᗺBA. 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 AᗺBA which decides which of the two cases happened. Given the decision of AᗺBA, 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 the tree growth, 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 long before F' has been 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. Below, we give a formal description of the procedure.

Protocol NextFinalizationGap(B, ℓ)

```
1: Let $F$ be the deepest node for which a valid finalization witness exists on
   PathTo(HonestTree, $B$) (let $F := G$ if this does not exist)
2: if Depth($B$) − Depth($F$) $= \ell$ then
3:     Output $\lceil 0.8 \cdot \ell \rceil$
4: else
5:     Output $2 \cdot \ell$
6: end if
```

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

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 $P_i \in \mathsf{Honest}$ and $B \in \mathsf{Path}_i^{\tau_0}$ with $\text{Depth}(B) = d$ and $\text{Height}(B) \geq \gamma$.
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 $B' \in \mathsf{FinalTree}^{\tau'}$ with $\text{Depth}(B') = d$ and $\text{Height}(B') \geq \gamma$ (namely $B' = B$).
3. For all $\tau' \in [\tau_0, \tau_0 + \delta_{\text{freeze}}]$ and for all $P_j \in \mathsf{Honest}$, we have $B \in \mathsf{Path}_j^{\tau'}$.

Dishonest chain growth. To prove that unique justified proposals exist, we need an additional property of the underlying blockchain, which is concerned with how fast dishonest parties can grow chains. The usual chain growth property defined in Section 3.2 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 11. For $\tau \in \mathbb{N}$ and $B \in \mathsf{FinalTree}^{\tau}$, let $\hat{B}$ to be the deepest ancestor of B in $\mathsf{FinalTree}^{\tau}$ that has at some point been on an honest path,

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

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

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

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

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

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 Δ_{growth} and $\tau - \hat{\tau}_B$ allows that for periods shorter than Δ_{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 $\mathsf{FinalTree}$. 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 $\mathsf{FinalTree}$ within a time interval of length Δ_{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.

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

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

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

Note that $\gamma_0 > \bar{d} - d$ and $\gamma_0 \geq \xi$ because $\rho_{\mathsf{disgro}} < \rho_{\mathsf{growth}}$. Now let $\gamma \geq \gamma_0$ and let τ_0 be the smallest time for which there exists some $P_i \in \mathsf{Honest}$ such that $\mathrm{Depth}(\mathsf{Pos}_i^{\tau_0}) \geq d + \gamma$. Note that this exists because we assume positive chain growth. Let B be the node on Path_i at depth d. By the choice of γ_0, we have $\mathrm{Depth}(\mathsf{Pos}_i^{\tau_0}) \geq d + \gamma > \bar{d}$, 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 $[\tau_0, \tau_0 + \delta_{\mathrm{freeze}}]$. Let $\tau' \in [\tau_0, \tau_0 + \delta_{\mathrm{freeze}}]$ and $P_j \in \mathsf{Honest}$. We have by $\mathrm{Prefix}(\xi)$ that $\big(\mathsf{Path}_i^{\tau_0}\big)^{\lceil \xi} \preceq \mathsf{Path}_j^{\tau'}$. Since $\mathrm{Depth}(\mathsf{Pos}_i^{\tau_0}) \geq d + \gamma \geq d + \xi$, we have that $B \in \big(\mathsf{Path}_i^{\tau_0}\big)^{\lceil \xi}$ and thus, $B \in \mathsf{Path}_j^{\tau'}$. This proves condition 3 of `UJP`.

Let $\tau' \in [\tau_0, \tau_0 + \delta_{\mathrm{freeze}}]$ and let $B' \in \mathsf{FinalTree}^{\tau'}$ 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', $\mathrm{PathTo}(\mathsf{FinalTree}^{\tau'}, \hat{B}')$ was a prefix of Path_k for some honest $P_k \in \mathsf{Honest}$. Hence, $\mathrm{Prefix}(\xi)$ implies that $\mathrm{PathTo}(\mathsf{FinalTree}^{\tau'}, \hat{B}')^{\lceil \xi}$ is a prefix of $\mathsf{Path}_j^{\tau'}$ for all $P_j \in \mathsf{Honest}$. As we have shown above, $B \in \mathsf{Path}_j^{\tau'}$. Thus, we either have $B \in \mathrm{PathTo}(\mathsf{FinalTree}^{\tau'}, \hat{B}')^{\lceil \xi}$ or $\mathrm{PathTo}(\mathsf{FinalTree}^{\tau'}, \hat{B}')^{\lceil \xi}$ is a prefix of $\mathrm{PathTo}(\mathsf{FinalTree}^{\tau'}, B)$. 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, $\mathsf{ChainGrowth}(\Delta_{\mathsf{growth}}, \rho_{\mathsf{growth}}, \rho'_{\mathsf{growth}})$ implies that all honest parties at time τ_0 have positions with depth less than $d + \gamma + \rho'_{\mathsf{growth}} \cdot \Delta_{\mathsf{growth}}$. 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 $d + \gamma + \rho'_{\mathsf{growth}} \cdot \Delta_{\mathsf{growth}} - \hat{d}'$. Note that this value is positive since $\gamma \geq \xi$ and $\hat{d}' < d + \xi$. The number of time intervals of length Δ_{growth} that fit into $[\hat{\tau}_{B'}, \tau_0]$ equals

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

Using the upper bound on chain growth, this implies

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

Hence, we obtain

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

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

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

Thus,

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

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

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

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

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

which implies

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

Therefore,

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

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`. □

6 Weak Multi-Valued Byzantine Agreement

At the core of the Finalization algorithm from Section 5, parties use a Byzantine agreement protocol relative to a justification J (here $J = J^{d,\gamma}_{\mathrm{INTREE}}$). Each party P_i inputs a proposal $\mathtt{p}_i$ (a block) and gets a decision $\mathtt{d}_i$ (a block or $\perp$) as output. We propose two variants of such a protocol, namely WMVBA and FilteredWMVBA which are inspired by classic asynchronous BA protocol such as [5]. WMVBA requires DCGrowth from the underlying blockchain, whereas FilteredWMVBA only relies on standard blockchain properties. Both protocols satisfy consistency and termination, 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.

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

Weak Persistency: 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 $\mathtt{d} \neq \perp$, then strictly more than $n/3$ of the honest parties had J-justified input $\mathtt{d}$.

As we have shown in Lemma 1, a blockchain satisfying Prefix, ChainGrowth, and DCGrowth have the property that at some point, there is a unique justified proposal in the tree. Weak persistency 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 $\perp$. To deal with this, FilteredWMVBA provides (non-weak) persistency, at the expense of only having 1-support. More precisely, FilteredWMVBA satisfies consistency, termination, and the following two properties:

Persistency: 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 $\mathtt{d} \neq \perp$, then strictly more than 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, persistency of 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 of the supplementary material, 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 AᗺBA is run after Freeze. We provide details about the sub-protocols and WMVBA in the following sections.

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 14). 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 Persistency: 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 12. A proposal message $m = (\mathtt{baid}, \textsc{proposal}, \mathtt{p})$ 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 13. A vote message $m = (\mathtt{baid}, \textsc{vote}, \mathtt{v})$ 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 $(\mathtt{baid}, \textsc{proposal}, \mathtt{v})$ from at least $n-2t$ parties or for $\mathtt{v} = \bot$ P_j has collected $J_{\mathtt{prop}}$-justified messages $(\mathtt{baid}, \textsc{proposal}, \mathtt{p})$ and $(\mathtt{baid}, \textsc{proposal}, \mathtt{p}')$ (from two different parties) where $\mathtt{p}' \neq \mathtt{p}$.

Definition 14 ($J_{\mathtt{dec}}$-justification)**.** A decision message $m = (\mathtt{baid}, \textsc{frozen}, \mathtt{d})$ is $J_{\mathtt{dec}}$-justified for P_j if P_j collected $J_{\mathtt{vote}}$-justified messages $(\mathtt{baid}, \textsc{vote}, \mathtt{d})$ from at least $t+1$ parties.

Observe that for example a proposal message $(\mathtt{baid}, \textsc{proposal}, \mathtt{p})$ 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 $(\mathtt{baid}, \textsc{proposal}, \mathtt{p})$.

Vote:

2. Collect proposal messages $(\mathtt{baid}, \textsc{proposal}, \mathtt{p}_i)$. 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 $J_{\mathtt{prop}}$-justified proposal messages from at at least $n-t$ parties contain the same proposal $\mathtt{p}$, broadcast vote message $(\mathtt{baid}, \textsc{vote}, \mathtt{p})$.
   (b) Otherwise broadcast vote message $(\mathtt{baid}, \textsc{vote}, \bot)$.

Freeze:

3. Collect vote messages $(\mathtt{baid}, \textsc{vote}, \mathtt{p}_i)$. 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 $(\mathtt{baid}, \textsc{frozen}, \mathtt{d})$, where $\mathtt{d} = \mathtt{p}$.
   (b) Otherwise if $\bot$ is contained in vote messages from at least $t+1$ parties output $(\mathtt{baid}, \textsc{frozen}, \bot)$.

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 $(\mathtt{baid}, \text{FROZEN}, \mathtt{d})$ which become $J_{\mathtt{dec}}$-justified.

Lemma 2. *For $t < \frac{n}{3}$ the protocol* Freeze *satisfies weak agreement, weak persistency, $n-2t$-support, and termination. The outputs of honest parties are $J_{\mathtt{dec}}$-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 $\mathtt{d}_i$ and $\mathtt{d}_j$ when $\mathtt{d}_i \neq \bot$ and $\mathtt{d}_j \neq \bot$.

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

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

$n-2t$-Support: Assume P_i outputs decision $\mathtt{d}_i \neq \bot$. That means that P_i received $J_{\mathtt{vote}}$-justified vote message $(\mathtt{baid}, \text{VOTE}, \mathtt{p})$ from strictly more than t parties. Out of those parties at least one must be honest. That honest must have received $J_{\mathtt{prop}}$-justified $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{d}_i)$ from at least $n-t$ parties. Thus at least $n-2t$ honest parties have sent $J_{\mathtt{prop}}$-justified $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{d}_i)$ which they only do if $\mathtt{d}_i$ 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 $J_{\mathtt{prop}}$-justified proposal messages and all honest parties will eventually send out $J_{\mathtt{vote}}$-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 $\mathtt{d}_i$ of honest party P_i is $J_{\mathtt{dec}}$-justified for P_i. If $\mathtt{d}_i \neq \bot$ then P_i collected $J_{\mathtt{vote}}$-justified messages $(\mathtt{baid}, \text{VOTE}, \mathtt{d}_i)$ from at least $t+1$ parties. Thus the output is $J_{\mathtt{dec}}$-justified. If $\mathtt{d}_i = \bot$ and P_i collected $J_{\mathtt{vote}}$-justified messages $(\mathtt{baid}, \text{VOTE}, \bot)$ from at least $t+1$ parties, then the output is also $J_{\mathtt{dec}}$-justified. □

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

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

Lemma 3. *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 $\mathtt{d}_i \neq \bot$ in* Freeze*, then eventually all honest parties will accept $\mathtt{d}_i$ has $J_{\mathtt{dec}}$-justified.*

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

The decision $\mathtt{d}_i$ 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 $J_{\mathtt{vote}}$-justified $(\mathtt{baid}, \textsc{vote}, \mathtt{p})$ vote messages from strictly more than t parties. This makes all honest parties accept $\mathtt{d}_i$ as $J_{\mathtt{dec}}$-justified eventually. □

6.2 Core Set Selection

The *weak core-set selection* protocol CSS is used in our binary byzantine agreement protocol ABBA (see Section 6.3) to compute a common core-set of party-value tuples. The global inputs, i.e., the pre-agreed parameters, are $J_{\mathtt{cssin}}$ and a delay Δ_{CSS}. Each party inputs a $J_{\mathtt{cssin}}$-justified bit where $J_{\mathtt{cssin}}$ is some (eventual) justification which is later defined by ABBA. Each honest party P_i (eventually) outputs a set $\mathtt{Core}_i$ which contains $J_{\mathtt{tpl}}$-justified tuples $(P, \mathtt{b})$ (see Definition 15).

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 $\Delta_{\textsc{net}}$. Then honest parties are at most $\Delta_{st} + \Delta_{\textsc{net}}$ de-synchronized. By waiting $\Delta_{\mathsf{CSS}} > \Delta_{st} + \Delta_{\textsc{net}}$ 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 $\mathtt{Core} \subseteq \bigcap_i \mathtt{Core}_i$ which contains tuples $(P, \mathtt{b})$ from at least $n-t$ different[2] parties.

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

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

Termination: If all honest parties have $J_{\mathtt{cssin}}$-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 $\mathtt{Core}_i$ 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 $J_{\mathtt{cssin}}$.

Definition 15. A tuple $(P_i, \mathtt{b}_i)$ is $J_{\mathtt{tpl}}$-justified for P_j if it is correctly signed by P_i and $\mathtt{b}_i$ is $J_{\mathtt{cssin}}$-justified for P_j.

Definition 16. A *seen message* $(\textsc{seen}, P_k, (P_i, \mathtt{b}_i))$ is $J_{\mathtt{seen}}$-justified for P_j if it is correctly signed by P_k and $(P_i, \mathtt{b}_i)$ is $J_{\mathtt{tpl}}$-justified for P_j.

Definition 17. A *done-reporting message* $(\textsc{doneReporting}, P_k, \mathtt{iSaw}_k)$ is $J_{\mathtt{done}}$-justified for P_j if it is correctly signed by P_k and for each tuple $(P_i, \mathtt{b}_i) \in \mathtt{iSaw}_k$ P_j has a $J_{\mathtt{tpl}}$-justified $(\textsc{seen}, P_k, (P_i, \mathtt{b}_i))$.

We give a formal description of the protocol next.

[2]Note that the $\mathtt{Core}$ or any $\mathtt{Core}_i$ 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 $\bot$.

Closing Down:

- Once party P_i had set $\mathtt{report}_i$ to $\bot$ 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 4. *For $t < \frac{n}{3}$ the protocol* CSS *satisfies common core, weak persistency, 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 Persistency: 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.

□

From the common-core property we directly get the following corollary.

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.*

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

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.*

6.3 A Binary Byzantine Agreement

We now describe a Binary Byzantine Agreement protocol (A∃BA). Parties use A∃BA to decide whether they agreed on a non-$\perp$ decision in Freeze (resp. FilteredFreeze).

Each party has a $J_{\mathtt{in}}$-justified bit $\mathtt{b} \in \{\perp, \top\}$ as input (see Definition 18). The idea is that parties input $\perp$ (resp. $\top$) if their $J_{\mathtt{dec}}$-justified output of Freeze was $(\mathtt{baid}, \textsc{frozen}, \perp)$ (resp. $(\mathtt{baid}, \textsc{frozen}, \mathtt{d})$ for $\mathtt{d} \neq \perp$). The output of honest parties in A∃BA are $J_{\mathtt{out}}$-justified bits (see Definition 21).

The A∃BA 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 A∃BA has the following properties.

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

Weak Persistency: If during the protocol execution there exists a $J_{\mathtt{in}}$-justified $\mathtt{b}$ such that no other bit $\mathtt{b}'$ is $J_{\mathtt{in}}$-justified for any honest party, then no honest party outputs $\mathtt{b}' \neq \mathtt{b}$.

1-Support: If some honest P_i output bit $\mathtt{b}_i$, then some other honest party had $\mathtt{b}_i$ as input.

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 A∃BA.

Definition 18. A bit $\mathtt{b}$ is $J_{\mathtt{in}}$-justified (input) for P_i if P_i has a $J_{\mathtt{dec}}$-justified tuple $(\mathtt{baid}, \textsc{frozen}, \mathtt{d})$ where $\mathtt{d} \neq \perp$ if and only if $\mathtt{b} \neq \perp$.

Definition 19. A bit $\mathtt{b}$ is $J_{\mathtt{phase,1}}$-justified (*phase*-1 *justified*) for P_i if it is $J_{\mathtt{in}}$-justified.

Definition 20. For $k > 1$ a bit $\texttt{b}$ is $J_{\texttt{phase,k}}$-justified (*phase-k justified*) for P_i if P_i has $> t$ signatures on $(\texttt{baid}, \textsc{justified}, \texttt{b}, k-1)$.

Definition 21. A bit $\texttt{b}$ is $J_{\texttt{out}}$-justified (output) for P_i if P_i has $n-t$ signatures on $(\texttt{baid}, \textsc{WeAreDone}, \texttt{b})$.

Leader election lottery. The ABBA 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) [21] with unpredictability under malicious key generation [8]. 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 to run the lottery.

Protocol. We next describe the protocol. It uses some globally known Δ as the initial waiting time. We conjecture it works well in practice to set Δ equal to the expected network delivery time. Any value works in principle since we increase the waiting time in each phase.

Protocol ABBA$(\texttt{baid}, J_{\texttt{in}})$

The protocol is described from the view point of a party P_i which has $J_{\texttt{in}}$-justified input $\texttt{b}_i$. 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$(\texttt{baid}, J_{\texttt{phase,k}}, k \cdot \Delta)$ where P_i inputs $\texttt{b}_i$. Denote by $\texttt{Core}_i$ the output of P_i.
2. P_i computes its lottery ticket $\texttt{ticket}_i$ and broadcasts signed $(\texttt{baid}, \textsc{justified}, \texttt{b}_i, k)$ along with $\texttt{ticket}_i$.
3. P_i waits for time $k \cdot \Delta$ and then does the following:
   - If all bits (of the tuples) in $\texttt{Core}_i$ are $\top$ let $\texttt{b}_i = \top$ and $\texttt{grade}_i = 2$.
   - Else if at least $n-t$ bits in $\texttt{Core}_i$ are $\top$ let $\texttt{b}_i = \top$ and $\texttt{grade}_i = 1$.
   - Else if all bits in $\texttt{Core}_i$ are $\bot$ let $\texttt{b}_i = \bot$ and $\texttt{grade}_i = 2$.
   - Else if at least $n-t$ bits in $\texttt{Core}_i$ are $\bot$ let $\texttt{b}_i = \bot$ and $\texttt{grade}_i = 1$.
   - Else, select a bit $\texttt{b}$ which occurs $> t$ in $\texttt{Core}_i$. If this bit is not unique, verify all lottery tickets using verifyABBALotteryTicket and select the bit $\texttt{b}$ where $(\texttt{b}, P) \in \texttt{Core}_i$ and P has the highest valid lottery ticket for all parties in $\texttt{Core}_i$. Let $\texttt{b}_i = \texttt{b}$ and $\texttt{grade}_i = 0$.
4. Go to the next phase.

Closing Down Each party sends at most one $(\texttt{baid}, \textsc{WeAreDone}, \cdot)$ message.

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

Lemma 5. *For $n > 3t$ the protocol* AaBA *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 AaBA 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 persistency 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 $k \cdot \Delta$ 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 $k \cdot \Delta$ 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 $k \cdot \Delta$ 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 $k \cdot \Delta$ 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. Thus 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 $(\mathtt{baid}, \textsc{WeAreDone}, \mathtt{b})$.

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 $\mathtt{b}$.

1-Support If there are two honest parties with different inputs then this property follows directly. So assume that all honest parties have the same input $\mathtt{b}$. Then similar to Case 1 from the proof of the previous Claim, any phase > 1 $\mathtt{b}$ is the only justified bit. Thus it will be output by all honest parties.

Weak Persistency: If all hones parties have a $J_{\texttt{in}}$-justified input $\texttt{b}$ and no other $J_{\texttt{in}}$-justified input exists, then honest parties will decide on $\texttt{b}$ with grade 2 in the first phase. Thus they will not output any other bit $\texttt{b}'$.

□

Remark 7. In AᗺBA, 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 semi-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 AᗺBA), and the last one in AᗺBA 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 $J_{\texttt{fin}}$-justified output which is either a proposal or $\perp$.

The idea of is WMVBA to first call Freeze to boil down the choice to a unique proposal or $\perp$. Parties then use AᗺBA which one is the case. Note that it can happen that an honest party decides on $\perp$ at the end of Freeze, but AᗺBA nevertheless outputs $\top$. In this case, at least one honest party had a justified non-$\perp$ decision as output in Freeze. This decision is unique. So we must ensure that honest parties with $\perp$ 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 3, this ensures that all honest parties will eventually receive the unique non-$\perp$ decision.

We define the following justification.

Definition 22. A decision $\texttt{d}$ is considered justified with respect to *final justification* $J_{\texttt{fin}}$ for P_i if P_i has $t+1$ signatures on the message $(\texttt{baid}, \text{WEAREDONE}, \texttt{d})$.

Note that a $\perp$ output from AᗺBA is already $J_{\texttt{fin}}$-justified. The protocol formally works as follows:

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

The protocol is described from the view point of party P_i which has J-justified input $\texttt{p}_i$.

1. Run Freeze($\texttt{baid}, J$) with input $\texttt{p}_i$. Denote by $\texttt{d}_i$ the $J_{\texttt{dec}}$-justified output for P_i from Freeze.
2. Run AᗺBA($\texttt{baid}, J_{\texttt{in}}$) with input $\texttt{b}_i$ where $\texttt{b}_i = \perp$ if $\texttt{d}_i = \perp$ and $\texttt{b}_i = \top$ otherwise. Denote by $\texttt{b}'_i$ the output of AᗺBA for P_i.
3. If $\texttt{b}'_i = \perp$, then terminate and output $\texttt{b}'_i$ (which is $J_{\texttt{fin}}$-justified) together with $\texttt{W} = \perp$, otherwise (if $\texttt{b}'_i = \top$) do:
   - Once P_i has a $J_{\texttt{dec}}$-justified decision message $(\texttt{baid}, \text{FROZEN}, \texttt{d}_i)$ with $\texttt{d}_i \neq \perp$ (from Freeze) it sends signed $(\texttt{baid}, \text{WEAREDONE}, \texttt{d}_i)$ to all other parties.
   - Once $n-t$ signed $(\texttt{baid}, \text{WEAREDONE}, \texttt{d})$, for some $\texttt{d}$, have been received, terminate and output $(\texttt{d}, \texttt{W})$, where $\texttt{W}$ contains $n-t$ of these signatures.

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

Proof. We prove each individual property next.

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 $\mathsf{A}\exists\mathsf{BA}$ and P_j had output $\top$ from $\mathsf{A}\exists\mathsf{BA}$. This contradicts the consistency property of $\mathsf{A}\exists\mathsf{BA}$.

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 $\mathsf{A}\exists\mathsf{BA}$. This implies that at least one honest party P_k had input $\top$ to $\mathsf{A}\exists\mathsf{BA}$ due to the 1-support of $\mathsf{A}\exists\mathsf{BA}$. 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 Persistency: 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 persistency 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 $\mathsf{A}\exists\mathsf{BA}$.

By the weak persistency of $\mathsf{A}\exists\mathsf{BA}$ it follows that $\bot$ is not a $J_{\mathtt{out}}$-justified output for $\mathsf{A}\exists\mathsf{BA}$. Therefore neither $\bot$ nor a decision $\mathtt{d}' \neq \mathtt{d}$ can be an $J_{\mathtt{fin}}$-justified output of $\mathsf{A}\exists\mathsf{BA}$.

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

Therefore P_i had output $\top$ from $\mathsf{A}\exists\mathsf{BA}$. By 1-support of $\mathsf{A}\exists\mathsf{BA}$ at least one honest party P_j had input $\top$ to $\mathsf{A}\exists\mathsf{BA}$. 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 $\mathsf{A}\exists\mathsf{BA}$. By the termination property of $\mathsf{A}\exists\mathsf{BA}$ they all have an $J_{\mathtt{out}}$-justified output from $\mathsf{A}\exists\mathsf{BA}$. Consider the following cases.

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

case ii): Assume honest party P_i has $J_{\mathtt{out}}$-justified output $\top$ from $\mathsf{A}\exists\mathsf{BA}$. The 1-support of $\mathsf{A}\exists\mathsf{BA}$ implies that at least one honest party P_j had input $\top$ for $\mathsf{A}\exists\mathsf{BA}$. Thus party P_j had $J_{\mathtt{dec}}$-justified output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_j)$ from Freeze with $\mathtt{d}_j \neq \bot$.

Lemma 3 implies that eventually all honest parties will accept $(\texttt{baid}, \text{FROZEN}, \mathtt{d}_j)$ as $J_{\texttt{dec}}$-justified. Thus P_i will eventually output a $J_{\texttt{dec}}$-justified decision.

□

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 persistency 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 persistency 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 $J_{\texttt{dec}}$ (cf. Definition 14). The protocol has the following properties.

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

Persistency: 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 $\mathtt{d}_i \neq \perp$, 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 23. A filtered proposal message $m = (\texttt{baid}, \text{FILTERED}, \mathtt{p}, \sigma)$ is considered $J_{\texttt{filt}}$-justified for P_j if either σ contains $J_{\texttt{prop}}$-justified proposal messages for $\mathtt{p}$ from $t+1$ different parties or σ contains $J_{\texttt{prop}}$-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 $J_{\texttt{dec}}$ (relative to the redefined $J_{\texttt{vote}}$) stays the same as for Freeze.

Definition 24. A vote message $m = (\texttt{baid}, \text{VOTE}, \mathtt{v})$ from P_i is considered $J_{\texttt{vote}}$-justified for P_j if is signed by P_i and either for $\mathtt{v} \neq \perp$ P_j has collected $J_{\texttt{filt}}$-justified filtered proposal messages from at least $n-2t$ parties or for $\mathtt{v} = \perp$ P_j has collected $J_{\texttt{filt}}$-justified filtered proposal messages $(\texttt{baid}, \text{FILTERED}, \mathtt{p}, \sigma)$ and $(\texttt{baid}, \text{FILTERED}, \mathtt{p}', \sigma')$ (from two different parties) where $\mathtt{p}' \neq \mathtt{p}$.

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

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 $(\mathtt{baid}, \text{VOTE}, \bot)$.

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

Lemma 6. *For $t < \frac{n}{3}$ the protocol* FilteredFreeze *satisfies weak consistency, persistency,* 1-*support, and termination. The outputs of honest parties are $J_{\mathtt{dec}}$-justified.*

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

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

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

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

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

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

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

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

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

□

Similar to Lemma 3 for Freeze we get.

Lemma 7. *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* FilteredFreeze, *then eventually all honest parties will accept $\texttt{d}_i$ has $J_{\texttt{dec}}$-justified.*

The proof follows along the lines of the proof for Lemma 3.

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, persistency, 1-support, and termination.*

Proof. We prove each individual property next.

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 A∃BA and P_j had output $\top$ from A∃BA. This contradicts the consistency property of A∃BA.

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 A∃BA. This implies that at least one honest party P_k had input $\top$ to A∃BA due to the 1-support of A∃BA. This party P_k must have had $J_{\mathtt{dec}}$-justified output $(\mathtt{baid}, \text{FROZEN}, \mathtt{d}_k)$ from FilteredFreeze (with $\mathtt{d}_k \neq \bot$).

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 FilteredFreeze.

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

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

The persistency 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.

By the weak persistency 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 7 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.

□

Finalization with Filtered Byzantine Agreement.

Theorem 4. *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 5 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 persistency 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 5. □

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 5. *For $t < \frac{n}{3}$ and a blockchain satisfying* UJP *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 (SETFINAL, $\cdot$) for the k-th time after getting output R_i from Finalization. The agreement property of WMVBA (cf. Theorem 2) guarantees that all honest parties output the same R_i. Thus they will all input (SETFINAL, R_i).

Chain-forming: Consider honest party P_i which at time τ inputs (SETFINAL, R). Let $\mathsf{lastFinal}_i$ be the last finalized block of P_i. As P_i is honest R was the output of Finalization. In 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 Finalization party P_j selected R in the subtree of $\mathsf{lastFinal}_j$ (at that time). By the agreement property we have $\mathsf{lastFinal}_i = \mathsf{lastFinal}_j$ and it follows that $\mathsf{lastFinal}_i \in \mathrm{PathTo}(\mathsf{Tree}_i^{\tau}, R)$. As the output of NextFinalizationGap is ≥ 1 we have that R is at greater depth than $\mathsf{lastFinal}_i$. So $R \neq \mathsf{lastFinal}_i$.

$n/3$-Support: Consider honest party P_i at time τ inputting (SETFINAL, R). As P_i is honest R was the output of Finalization. In 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 Finalization these parties selected R as on their Path.

Δ-Updated: 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 WMVBA 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. 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 persistency and termination property of WMVBA that eventually Finalization will terminate with a new finalized block lastFinal at depth d.

It remains to show that protocol achieves Δ-updated. According to Section 5.1 after some point in time we have that the following holds right after finalizing a block. First, the ℓ output of NextFinalizationGap is roughly the same as the number of blocks added to the chain during finalization. Second, the depth at which the next finalized block should be roughly at the same depth as the deepest honest block, i.e., its height (which can be negative if it is deeper than the tree) $\mathtt{h}$ is roughly zero. In particular, $\mathtt{h} \leq \ell$

So consider a finalization run at this point in time. Let d be the depth of the last finalized block B and let d' be the depth where the new finalized block has to be determined. We have to upper bound the distance between the depth d of the last finalized block and the depth of honest positions during the run of Finalization for d'.

This difference Δ is the largest right before a new finalized block is output by Finalization. The difference consists of the following parts. First, we have the height $\mathtt{h}_1$ of the finalized block B at the beginning of its own finalization. Then, we have the number of blocks added during the finalization of B, i.e. around ℓ. Next, if the height for d' was negative, there are $\mathtt{h}_2$ more blocks. Finally, there are ℓ more blocks which have been added during finalization for d'. We get $\Delta = \mathtt{h}_1 + \mathtt{h}_2 + 2\ell \leq 4\ell$.

□

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 [24, 17, 18]. The appropriate strategy to select a finality 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 finality 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 $C = \mathsf{FC}(\mathsf{Tree}, B)$ 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 $B \in \mathsf{Tree}$ 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 [24, 17, 18] 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 [20] and Ouroboros Praos [8]; 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 [16].

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.

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

References

[1] 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.

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

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

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

[5] 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.

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

[7] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

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

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

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

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

[23] 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.

[24] 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.

[25] 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.