Blockchain Proportional Governance Reconfiguration: Mitigating a Governance Oligarchy

Deepal Tennakoon
University of Sydney
Sydney, Australia
dten6395@uni.sydney.edu.au

Vincent Gramoli
University of Sydney
Sydney, Australia
vincent.gramoli@sydney.edu.au

***Abstract*—Blockchain governance is paramount to lead securely a large group of users towards the same decisions without disputes about the legitimacy of a blockchain instance over another. As of today, there is no efficient way of protecting this governance against an oligarchy. This paper aims to offer a new dimension to the security of blockchains by proposing a solution known as *proportional governance reconfiguration*. This solution mitigates the formation of an oligarchy by (1) electing governors proportionally using a proportional multi-winner election protocol (2) reconfiguring the governance automatically and periodically. The proportional governance reconfiguration relies on a Solidity based implementation making it compatible and usable in many smart contract supported blockchains. We prove our solution solves the proportional governance problem and we evaluate our solution on two smart contract supporting blockchains Ethereum-PoA and Smart Redbelly Blockchain. Our results indicate that our proportional governance can elect 200 governors within 6-12 minutes when 1000 voters from 5 continents vote for 500 candidates.**

***Index Terms*—Blockchain, Governance, Reconfiguration**

I. Introduction

The notion of *governance*, which is generally understood as the processes relied upon to make decisions and modify the protocol, has become an important topic in blockchain [1], [2], [3]. In the context of blockchains, governance can include decisions such as updating the blockchain protocol, varying blockchain parameters (e.g., changing the block period), and deciding upon a block to be executed (e.g., reaching consensus) [3], [4]. The absence of governance has led users to create dissident instances of the two largest blockchains: Bitcoin is now split into BTC and BCH while Ethereum is now split into ETH and ETC [5], [6].

A pernicious threat in blockchain governance is the risk of an attacker controlling an *oligarchy* amongst the governors. If this happens the oligarchy can dictate the decisions to modify the blockchain protocol making the blockchain governance centralized. There are two methods which can create an oligarchy in blockchain governance.

First, an oligarchy can be formed in the blockchain governance through the governance election process. More specifically, modern blockchains, which have mostly replaced Proof-of-Work (PoW) with Proof-of-Stake (PoS) to improve performance, elect governors based on their stake providing more opportunities to those that have a higher stake to become governors [7], [8], [3], [9]. Given the skewed distribution of wealth, this can inadvertently create an oligarchy.

Second, an adversary can form an oligarchy by corrupting governors in a committee with a bribe. Such *bribery* attacks on governors are a potent threat because governors in blockchains are usually limited in number [9], [3] and therefore forming an oligarchy among such a limited set is not as challenging as bribing an entire network of blockchain nodes.

To mitigate the two aforementioned methods of forming an oligarchy of governors, we propose *proportional governance reconfiguration* that is compatible with smart contract supported blockchains.

The first part of proportional governance reconfiguration is the proportional governance to tackle the formation of an oligarchy among governors through the election process. Proportional governance selects governance users or *governors* that proportionally represent the voters. This is to prevent an adversary from creating an oligarchy in the governance. Proportionality is a concept widely used in social choice theory to elect a set of candidates fairly to a legislative body [10]. In general terms, proportionality ensures that a diverse set of candidates are elected ensuring even the minority voters are represented in a legislative body.

As multiple governors need to be elected to a blockchain governance committee, we needed a multi-winner election protocol. Thus, we used the Single Transferable Vote (STV) protocol [11], used for example to elect the Australian senate [12]. STV outputs a set of candidates proportionally representative of the voted preferences. However, the STV protocol is synchronous: a voter simply has to cast a vote within a limited known period of time for its vote to be counted when tallying votes. Blockchains instead operate in a general network (e.g., the Internet) where the communication is not synchronous[1] and where Byzantine nodes can arbitrarily delay messages. Thus, the STV protocol executing on a blockchain with n nodes that waits for votes from all n nodes cannot progress if Byzantine voters do not cast votes. This is because one cannot distinguish a slow voter from a Byzantine voter due to the upper bound on the message delay being unpredictable.

[1]synchronous communication assumes the transmission delay of the messages/votes sent over the communication links is bounded by a known value [13]

Blockchain	Election	Proportional Governance
Tendermint [16]	None	no
Algorand [7]	Sortition	no
Hybrid consensus [17]	PoW puzzle	no
Zilliqa [18]	PoW puzzle	no
OmniLedger [19]	Sortition	no
RapidChain [15]	PoW puzzle	no
ComChain [20]	None	no
Libra [21]	None	no
SmartChain [22]	None	no
Polkadot [23]	Multi-winner approval voting	no
EOS [9]	Multi-winner approval voting	no
This work (compatible with any blockchain)	Multi-winner preferential voting	yes

TABLE I
BLOCKCHAINS WITH GOVERNANCE RECONFIGURATION DO NOT SOLVE THE PROPORTIONAL GOVERNANCE PROBLEM (DEF. 1)

To solve this problem, we develop a variant of STV known as BFT-STV that offers (1) the same proportionality guarantees as STV, (2) does not assume synchrony and (3) works in a Byzantine setting s.t. at most $t < n/3$ of n voters are Byzantine (we denote $f \leq t$ as the actual number of Byzantine voters). The ratio of f comes from (i) the need for voters to reach consensus on the new set of governors and (ii) the impossibility of solving consensus with $f \geq n/3$ Byzantine participants in blockchains under the general setting [14]. We implement BFT-STV in a smart contract to make our proportional governance pluggable and compatible with smart contract supported blockchains.

The second part of proportional governance reconfiguration is the governance reconfiguration to reduce the risk of forming an oligarchy through bribery. During reconfiguration, the current governors are replaced with new governors based on an election outcome. Such reconfigurations are used in notable prior work to mitigate bribery attacks [15], [7], [3]. By periodically selecting a diverse set of governors proportionally representative of a sample set, the formation of an oligarchy through bribery among governors can be mitigated. The problem is if the current governors do not reconfigure upon the selection of a set of new governors, the blockchain service would create a split, leading users to create dissident instances of the same blockchain [5], [6]. Instead, if all users initially joining agree that the blockchain self-reconfigures upon a special smart contract execution, then no split can occur. Thus, our work proposes an automatic governance reconfiguration protocol that rotates governors based on a smart contract output.

To the best of our knowledge, the proportional governance reconfiguration we propose is the first solution that (1) mitigates an oligarchy among governors and (2) is pluggable and compatible with smart contract supported blockchains due to its generality and smart contract based implementation.

In summary, this paper defines the proportional governance reconfiguration problem (§III), designs a solution for it known as the proportional governance reconfiguration that is compatible and pluggable with many smart contract supported blockchains (§IV and §V), proves the solution correct (§IV-E) and evaluates the solution (§VI). Our proposed solution offers the following practical contributions:

- We introduce the first Byzantine fault tolerant multi-winner election protocol, called *BFT-STV* to elect a set of governors proportionally in order to mitigate an oligarchy among governors (§IV). BFT-STV is a new primitive that augments the STV election protocol to work in a setting where at most $t < n/3$ Byzantine voters exist among n voters without assuming synchrony (we denote $f \leq t$ as the actual number of Byzantine voters). As it is impossible to distinguish a non-responsive Byzantine voter from a delayed message, we introduce a new election quota $q_B = \frac{n-t}{k+1}$ where k is the size of the committee. Interestingly, we show that our BFT-STV protocol preserves the proportionality of STV while ensuring termination (§IV-E).
- We implement this new protocol in a smart contract written in the Solidity programming language, making our protocol easily compatible and pluggable with smart contract supported blockchains [24], [25], [26], [27] (one can reimplement our protocol to make it work with a different smart contract programming language). Implementing the BFT-STV protocol on a smart contract comes with its own technical challenges. First, smart contracts are public, thus to preserve privacy of votes to avoid strategic voting, we employ a commit-reveal scheme (§IV-B). Second, the STV algorithm and our BFT-STV adaptation of it is NP-hard. Executing such an algorithm on a smart contract leads to poor performance. Due to this reason, we integrate an election sharding scheme to make our solution scalable (§IV-D).
- The BFT-STV smart contract alone is not sufficient to action the outcome of the election to reconfigure the blockchain. Thus, we introduce a novel and automatic reconfiguration protocol that rotates governor sets periodically based on the BFT-STV smart contract output. The periodic reconfiguration of governors helps mitigate bribery attacks launched by slowly-adaptive adversaries [19], [15]. This mitigates the formation of an oligarchy among governors. In particular, our protocol revokes permissions of existing governors to elect new governors periodically before a large portion of them could be bribed forming an oligarchy.
- We prove that our protocols are correct (§IV-E and §V). In particular, we also show that BFT-STV satisfies proportionality without assuming synchrony. Our world-scale evaluations of BFT-STV with 200 validators of Ethereum-PoA and Smart Redbelly Blockchain [27] spanning 5 continents can elect 200 governors from 500 candidates with 1000 voters casting ballots within 6-12 minutes (§VI).

In the remainder of this paper, we present the background and motivations (§II), and the proportional governance reconfiguration problem (§III). Next, we present our solution to this problem in §IV and §V along with proofs that our solution solves the proportional governance reconfiguration problem. In §VI, we evaluate our solution. Finally, we present the related work (§VII) and conclude (§VIII).

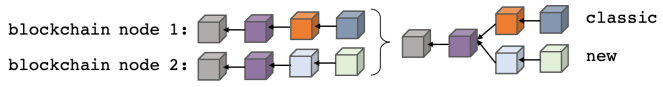

Fig. 1. If blockchain nodes disagree on a protocol update then they may start accepting distinct blocks, which results in a hard-fork with a classic version of the blockchain (e.g., ETC, BTC) and a new version of it (e.g., BCH, ETH).

II. Background and Motivations

A. Importance of blockchain governance

The notion of governance in blockchains, which encompasses the processes followed to make decisions impacting the blockchain protocol has become an important topic recently [1], [2], [3]. The governance structure includes the identity of parties capable of suggesting changes, the avenue through which such changes are proposed, the users capable of deciding the changes and the parties implementing these changes. Due to the large number of users of a blockchain, governance is especially relevant to lead this large cohort towards a common goal. With a lack of governance, the divergence of opinions may result in the split of the blockchain into multiple instances sharing a common transaction history but accepting distinct transactions.

As an example, consider Figure 1, where blockchain node 1 rejects a software upgrade and keeps accepting old-formatted blocks whereas blockchain node 2 accepts this upgrade and starts accepting blocks in a new format, leading to a hard fork. The two largest blockchains were victims of such splits: Bitcoin is now split into BTC and BCH [6] whereas Ethereum is now split into ETH and ETC [5]. The absence of governance can draw blockchain users into such clashes.

B. Preventing governance oligarchy

The biggest challenge is to prevent an attacker from obtaining the control of the governance potentially forming an oligarchy. This is usually tackled through making the governance decentralized across multiple governors. However, having a fixed set of governors can expose these governors to bribery attacks [7], [15], [19], [28].

As blockchains typically handle valuable assets, several works already noted the risk for a user to bribe other users to build an oligarchy capable of stealing these assets [29]. Most of these works explicitly assume a *slowly-adaptive adversary* [15], [30], [19] that can corrupt a limited number of nodes between consensus epochs but cannot corrupt participants during an epoch.

To reduce the chances that governance users, or *governors* being bribed, forming an oligarchy, the aforementioned works propose governance reconfigurations [7], [15], [19], [28]. None of these works solve the proportional governance reconfiguration problem (§III).

C. Social choice theory with Byzantine fault tolerance

To propose meaningful properties for blockchain governance, we draw inspiration from classic work on social choice theory. Given a set of n voters, each casting an *ordinal ballot* as a preference order over all m candidates, a *multi-winner election* protocol outputs a winning committee of size k.

Black [31] was the first to define the proportionality problem where elected members must represent "all shades of political opinion" of a society.

Dummett [32] introduced *fully proportional representation* to account for ordinal ballots, containing multiple preferences. Given a set of n voters aiming at electing a committee of k governors, if there exists $0 < \ell \leq k$ and a group of $\ell \cdot q_H$ who all rank the same ℓ candidates on top of their preference orders, then these ℓ candidates should all be elected. However, it builds upon Hare's quota q_H, which is vulnerable to strategic voting whereby a majority of voters can elect a minority of seats [33]. This problem was solved with the introduction of Droop's quota q_D as the smallest quota such that no more candidates can be elected than there are seats to fill [11].

Woodall [34] replaces Hare's quota with Droop's quota $q = \lfloor \frac{n}{k+1} \rfloor$ and defines the *Droop proportionality criterion* as a variant of the fully proportional representation property: if for some whole numbers j and s satisfying $0 < j \leq s$, more than $j \cdot q_D$ of voters put the same s candidates (not necessarily in the same order) as the top candidates in their preference list, then at least j of those s candidates should be elected. This is the property we target in this paper and we simply rename it *proportionality* (Def.1).

It is known that the First-Past-The-Post (FPTP) single-winner election and the Single Non-Transferrable Vote (SNTV) multi-winner election cannot ensure fully proportional representation [35]. The reason is that voters can only reveal their highest preference.

This property can however be achieved using the *Single Transferable Vote (STV)* algorithm. In STV, candidates are added one by one to the winning committee and removed from the ballots if they obtain a quota q of votes. STV is used to elect the Australian senate and is known to ensure fully proportional representation.

Unfortunately, this protocol is synchronous [36] in that its quotas generally rely on the number of votes n received within a maximum voting period. As one cannot predict the time it will take to deliver any message on the Internet, one cannot distinguish a slow voter from a Byzantine one. Considering n as the number of governors or potential voters among which up to t can be bribed or Byzantine, our protocol only waits for at most $n - t$ votes to progress without assuming synchrony. Waiting for $n-t$ prevents us from guaranteeing that the aforementioned quotas can be reached. We thus define a new quota called the *Byzantine quota* $q_B = \lfloor \frac{n-t}{k+1} \rfloor$ such that $t < n/3$ and reduce the number of needed votes to start the election to $n - t$. Based on q_B, we propose BFT-STV that extends STV for a Byzantine fault tolerance environment. We also show that BFT-STV satisfies proportionality without assuming synchrony (§IV-E). Of course, up to t of these $n - t$ ballots may be cast by Byzantine nodes, however, we show in Theorem 2 that no adversary controlling up to t Byzantine nodes can act as a dictator a property known as *non-dictatorship* proposed by Arrow [37].

III. The Proportional Governance Reconfiguration Problem

Our goal is to solve the proportional governance reconfiguration problem to mitigate a governance oligarchy in blockchains. The proportional governance reconfiguration problem encapsulates (1) the proportional governance problem (§III-B) and (2) the governance reconfiguration problem (§III-C). To put it simply, first we offer a blockchain governance that allows distributed users to elect a committee of governors proportionally representative of the voters and without dictatorship, which solves the proportional governance problem (§III-B). Second, we mitigate bribery attacks by periodically changing the governors, which solves the governance reconfiguration problem (§III-C).

In this section, we first present the computation model (§III-A) before defining the proportional governance problem (§III-B), the governance reconfiguration problem (§III-C) and the threat model (§III-D).

A. Byzantine fault tolerant distributed model

We consider a distributed system of n governor nodes also known as a governor committee, identified by public keys I and network identifiers (e.g., domain names or static IP addresses) A. We assume public key cryptography and that the adversary is computationally bounded. Hence, only the issuer of a transaction can *sign* it and any recipient can correctly verify the signature. Governor nodes (i) execute the consensus protocol in order to agree on a unique block to be appended to the chain and (ii) execute transactions and maintain a local copy of the state of the blockchain. Client[2] nodes simply send transaction requests to read from the blockchain (to check an account balance), transfer assets, upload a smart contracts or invoke a smart contract Candidate nodes m are nodes eligible to become governors and are voted upon by n current governors to be included in new governor sets periodically. We assume $m >> k$ s.t. k is the target next governor committee size. The number of Byzantine nodes in the candidate nodes set is assumed to be f_c s.t. $f_c \leq m/4$.

As we target a secure blockchain system running over an open network like the Internet, we consider the strongest fault model called the Byzantine model [39], where nodes can fail arbitrarily by, for example, sending erroneous messages or delaying messages. We assume that there is no known bound on the transmission delay of messages between nodes, a property called *partial synchrony* [36]. As governors execute consensus and consensus cannot be solved in our model with $n/3$ Byzantine nodes [39], we assume there are $f \leq n/3$ Byzantine nodes among the governor nodes. We assume a *slowly adaptive adversary* as many prior works [40], [15], [19] such that the adversary can only corrupt/bribe nodes between governor reconfigurations (i.e., between committees). The slowly-adaptive adversary cannot bribe nodes within a committee due to its slowly adaptive nature. Note that a node that is not Byzantine is called *correct*.

[2]The term "client" is often used in Ethereum to refer to a node regardless of whether it acts as a server. We use client in the traditional sense of the client-server distinction [38].

B. Proportional governance problem

We refer to the proportional governance problem as the problem of designing a BFT voting protocol in which n voters rank m candidates to elect a committee of k governors ($k < m$ and $m > n$) to ensure non-dictatorship as defined by Arrow [37] and proportionality as defined by Dummett [32], Woodland [34] and Elkind et al. [41] (cf. §II-C). The main distinction is that we adapt this problem from social choice theory to the context of distributed computing.

Definition 1 (The Proportional Governance Problem)**.** *The* secure governance problem *is for a distributed set of n voters, among which $f \leq t < n/3$ are Byzantine, to elect a winning committee of k governors among m candidates (i.e., $m > k$) such that the following two properties hold:*

- Proportionality: *if, for some whole numbers j, s, and k satisfying $0 < j \leq s \leq k$, more than $j(n-t)/(k+1)$ of voters put the same s candidates (not necessarily in the same order) as the top s candidates in their preference listings, then at least j of those s candidates should be elected.*
- Non-dictatorship: *a single adversary, controlling up to $f < n/3$ Byzantine voters, cannot always impose their individual preference as the election outcome.*

The need for these two properties stems from our goal of guaranteeing proportional representation (proportionality) but also disallowing a coalition of Byzantine nodes from imposing their decision on the rest of the system (non-dictatorship). Note that the non-dictatorship property differs slightly from the original definition [37] that did not consider a Byzantine coalition. In particular, our property considers coalitions and prevents them from imposing their preference in "all" cases.

C. Governance reconfiguration problem

We refer to the governance reconfiguration problem as the problem of ensuring blockchain safety despite governance reconfiguration. We adopt the safety property from Garay et al. [42] and restated more recently by Chan et al. [43] to governance reconfiguration.

Definition 2 (The Governance Reconfiguration Safety)**.** *The first block stored locally after governance reconfiguration by any two correct governor nodes in the governance committee should be equal.*

Any two correct governors in the same committee starting from the same block ensures that after the governance reconfiguration, governors start with the same state. Thus, when the governance committee executes the same set of totally ordered new transactions from clients, the state at any two correct governors remain identical since the start state is identical. This helps satisfy blockchain safety.

D. Threat model

As in previous blockchain work [30], [7], [15], [19], we assume a slowly adaptive adversary with a limited bribing power that cannot bribe governors within a committee but can only bribe/corrupt up to f_c nodes between reconfigurations such that $f_c < m/4$ where m is the candidates.

For the initial set of governors to be sufficiently diverse, we can simply select governors based on their detailed information. This can be done by requesting initial candidates to go through a *Know-Your-Customer (KYC)* identification process, similar to the personal information requested from the Ethereum proof-of-authority network users before they can run a validator node [44]. A set of governors could then be selected depending on the provided information while ensuring multiple governors are not from the same jurisdiction, they are not employed by the same company, they represent various ethnicities, they are of balanced genders, etc.

1) Bribery attack: Limiting the number of nodes responsible to offer the blockchain service as done in recent open blockchains [45] exposes the service to a bribery attack [29], which is an act of offering something to corrupt a participant. This is because it is typically easier to bribe fewer participants. In particular, as consensus cannot be solved with at least $\frac{n}{3}$ Byzantine processes among n when message delays are unknown [36], it is sufficient to bribe $\frac{n}{3}$ governors to lead correct governors to disagree on the next block appended to the blockchain and thus create a fork in the blockchain. The attacker can then exploit this fork to have its transaction discarded by the system and then re-spend the assets he supposedly transferred in what is called a *double spending*. Our reconfiguration protocol mitigates such a bribery attack in the presence of a slowly-adaptive adversary by re-electing n new governors from m candidates that execute the consensus protocol every x blocks. This is how we mitigate the risk of $\frac{n}{3}$ of the current governors getting bribed that can form an oligarchy. More specifically, due to the assumption of a slowly adaptive adversary that bribes/corrupts at most f_c candidate nodes s.t. $f_c < m/4$, a governance committee k periodically elected proportionally from a diverse set m will have $f < n/3$ with high probability (A reasonable assumption made in prior work given that $m >> k$). Within the governance committee period this f will remain static as the slowly-adaptive adversary can only corrupt nodes between reconfigurations.

2) Sybil attacks: A Sybil attack consists of impersonating multiple identities to overwhelm the system—in the context of votes, a Sybil attack could result in an adversary voting with multiple identities to alter the outcome of an election. Proof-of-stake based voting approaches weigh a ballot cast by a voter based on the coins they have staked. Thus, minimizing the impact on the election outcome if an adversary splits their stake among multiple identities and cast ballots. We adopt a solution that consists of providing authenticating information, in the form of know-your-customer (KYC) data, in exchange for the permission to vote for governors, or be a governor candidate. This authentication copes with Sybil attacks by preventing the same authenticated user from using distinct node identities to cast votes.

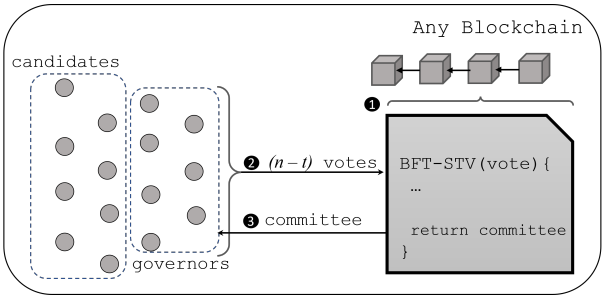

Fig. 2. The smart contract that implements the BFT-STV protocol is on-chain ❶, takes as an input a set of at least $(n-t)$ ballots (each ranking k candidates among m) cast by $(n-t)$ voters among the n governors ❷ and outputs a committee of k elected nodes ❸ to play the role of the new governors. Note that the last committee of governors elected will then vote for the next committee of governors ❷ and so on (one can fix $k = n$ so that the committee size never changes).

IV. Byzantine Fault Tolerant Proportional Governance

In this section, we present how to elect, despite $f \leq t < n/3$ Byzantine nodes, a diverse set of governors to mitigate the formation of an oligarchy. The idea is to allow a set of n blockchain nodes that are current governors to vote and elect the committee of next governors proportionally representing the current governor votes.

To this end, we propose the *Byzantine Fault Tolerant Single Transferrable Vote (BFT-STV)* smart contract that solves the proportional governance problem (Def. 1).

A. Overview

In order to guarantee that the election solves the proportional governance problem (Def. 1), we designed the BFT-STV algorithm and implemented it as a smart contract. In this section, we present the high level pseudo code of the BFT-STV algorithm. To bootstrap, the initial permissions to vote are obtained by n initial governors after identification using KYC to ensure diversity and prevent Sybil attacks (§III-D2). Recall that governors cannot use the classic STV algorithm to elect a new committee as the smart contract has to progress despite up to $t < n/3$ Byzantine voters not casting proper ballots and as the upper-bound on the message delay is unpredictable. As depicted in Figure 2, the BFT-STV smart contract takes, instead, as an input $n-t$ ballots cast by the voters that are governors. Each ballot consists of a rank of all the candidates, hence the name *ordinal ballot*. Once the threshold $n-t$ of cast ballots is reached, the BFT-STV contract selects the governors based on the preference order indicated in the $n-t$ ballots. Traditionally, the STV algorithm consists of counting which candidates received a number of votes that exceed the quota $q_D = \frac{n}{k+1}$ where k is the size of the governance committee to be elected. However, as there can be at most t Byzantine nodes among the voters, we introduce the Byzantine quota

$q_B = \frac{n-t}{k+1}$ (denoted q when clear from the context). We will explain how the blockchain replaces the current governors by the newly elected committee of governors in Section §V.

1: **Initial state:**
2: k, the size of the targeted committee.
3: n, the number of voters.
4: t, an upper bound on the number f of byzantine replicas, $f \leq t$.
5: m, the number of candidates per ballot.
6: v, a mapping from candidates to their number of votes.
7: $ballots$, the set of received ordinal ballots, initially empty
8: $C \subseteq I$, the set of candidates.
9: $E \subseteq C$ the set of eliminated candidates, initially empty.
10: $S \subseteq C$ the set of winning candidates, initially empty.
11: $pref[ballot] = index$ a map of ballot and its current preference index.
12: $voted[vAddr] = false$ a map of voter addresses and whether voted.
13: $privateVotes[vAddr] = hashVote$ a map of voter addr. and priv. vote (ballot).
14: $countprivate$ no. of private votes.
15: $Sender[vAddr] = b$ a map of voter addr. and ballot.

16: commitVote($hashVote$):
17: **if** $voted[vAddr] ==$ false **then** ▷ *prevents double voting*
18: $privateVotes[vAddr] \leftarrow hashVote$ ▷ *store priv. votes for voters*
19: $countprivate \leftarrow countprivate + 1$ ▷ *no. priv. votes received*
20: $voted[vAddr] \leftarrow$ true ▷ *vAddr has voted*
21: **if** $countprivate == n$-t **then**
22: $votingEnded \leftarrow$ true
23: **emit** "threshold of votes reached" ▷ *notify all priv. votes are received*

24: reveal(b, h):
25: **if** $votingEnded ==$ true **then** ▷ *voters reveal votes if election ended*
26: **if** hash(b) == h & $privateVotes[vAddr] == h$ **then** ▷ *hashes eq.*
27: **if** $well\text{-}formed(b)$ **then** $ballots \leftarrow ballots \cup \{b\}$
28: **if** ($ballots$ has $n - t$ ballots from distinct voters revealed) **then**
29: change-committee($ballots$) ▷ *replace committee*

30: change-committee($ballots$): ▷ *replace committee*
31: **for all** $b \in ballots$ **do** ▷ *for each received ballot*
32: **if** ($b[0] = c$ such that $c \in C$) **then**
33: $v[c] \leftarrow v[c] + 1$ ▷ *# 1st pref = c*
34: $pref[b] \leftarrow 0$ ▷ *assign pref. index of b to the first preference/index 0*
35: $round \leftarrow 0$ ▷ *first round*
36: **while** ($|S| < k$) **do** ▷ *until the new committee is full*
37: $S \leftarrow \mathsf{STV_B}(v, ballots, pref)$ ▷ *invoke classic STV*
38: $round \leftarrow round + 1$ ▷ *increment round number*
39: **if** ($|C| - |E| = k$) **then** break ▷ *stop eliminating*
40: **for all** $b \in ballots$ **do** ▷ *for each ballot*
41: **for** ($j = 0; j < m; j$++) **do** ▷ *each candidate in decreasing pref. order*
42: **if** ($|S| < k \wedge b[j] \in C \setminus S \setminus E$) **then** ▷ *if eligible*
43: $S \leftarrow S \cup \{c\}$ ▷ *select c*
44: **emit** S ▷ *explicitly emit committee*

Algorithm 1: Threshold Single Transferable Vote (Threshold-STV) - Part 1

45: **Initial state:**
46: k, the size of the targeted committee.
47: n, the number of voters.
48: t, an upper bound on the number f of byzantine replicas, $f \leq t$.
49: $q_B = \frac{n-t}{k+1}$, the quota of votes to elect a candidate.
50: $C \subseteq I$, the set of candidates.
51: $E \subseteq C$, the set of eliminated candidates, initially empty.
52: $S \subseteq C$, the set of winning candidates, initially empty.
53: $X \subseteq C$, the set of excess candidates, initially empty.

54: $\mathsf{STV}_B(v, ballots, pref)$:
55: **if** $\exists c \mid v[c] > q_B$ **then** ▷ *if the quota is exceeded*
56: $S \leftarrow S \cup \{c\}$ ▷ *elect candidate*
57: $X \leftarrow X \cup \{c\}$ ▷ *save candidates that exceed quota in X*
58: $x[c] \leftarrow v[c] - q_B$ ▷ *excess vote from candidate c*
59: **for all** $b \in ballots$ **do** ▷ *for each ballot*
60: **if** $b[pref[b]] = c$ and $c \in X$ **then** ▷ *if current ballot pref = one of X*
61: $count[c] \leftarrow count[c] + 1$ ▷ *the number of candidates c*
62: $pref\text{-}next[b] \leftarrow pref[b] + 1$ ▷ *point to next preferred candidate*
63: **while** $b[pref\text{-}next[b]] \in (S \vee E)$ **do** ▷ *while not uneligible*
64: $pref\text{-}next[b] \leftarrow pref\text{-}next[b] + 1$ ▷ *try next pref. pointer*
65: **if** $b[pref\text{-}next[b]] \notin (S \cup E)$ **then** ▷ *if eligible candidate found*
66: $pref[b] = pref\text{-}next[b]$ ▷ *move the preference pointer*
67: $z \leftarrow b[pref\text{-}next[b]]$ ▷ *next preferred candidate in ballot*
68: $cand\text{-}next \leftarrow cand\text{-}next \cup \{\langle c, z\rangle\}$ ▷ *current&next candidates*
69: $count[z] \leftarrow count[z] + 1$ ▷ *The number of candidates z*
70: **for all** unique $\langle c, z\rangle \in cand\text{-}next$ **do** ▷ *transfer excess votes*
71: $v[z] \leftarrow v[z] + x[c] \cdot (count[z]/count[c])$ ▷ *to next candidates*
72: **if** $\forall c : v[c] \leq q_B$ **then** ▷ *if no candidates exceed the quota in the round*
73: $E \leftarrow (E \cup t \mid t = \min_{\forall c}(v[c]))$ ▷ *eliminate candidate with least votes*
74: $transfer\text{-}vote \leftarrow v[t]$
75: $v[t] \leftarrow 0$ ▷ *reset votes of least candidate to 0*
76: **for all** $b \in ballots$ **do**
77: **while** $s < size$ **do**
78: **if** $b[s] = t$ **then** ▷ *store ballot and preference index...*
79: $elimpointer \leftarrow elimpointer \cup (b, s)$ ▷ *...of least voted cand.*
80: $s \leftarrow s + 1$ ▷ *Increment preference*
81: **for all** $(b, s) \in elimpointer$ **do**
82: **if** $b[s] = m \wedge m \in E$ **then** ▷ *If preference s of ballot b is eliminated*
83: $pref\text{-}next[b] \leftarrow s + 1$
84: $count[m] \leftarrow count[m] + 1$ ▷ *count of candidates m in all ballots*
85: **while** $b[pref\text{-}next[b]] \in (S \vee E)$ **do** ▷ *until candidate is found*
86: $pref\text{-}next[b] \leftarrow pref\text{-}next[b] + 1$ ▷ *...increment pref. pointer*
87: **if** $b[pref\text{-}next[b]] \notin S \cup E$ **then**
88: $pref[b] \leftarrow pref\text{-}next[b]$ ▷ *move the preference pointer*
89: $z \leftarrow b[pref\text{-}next[b]]$
90: $cand\text{-}next \leftarrow cand\text{-}next \cup (m, z)$ ▷ *least voted & next cand.*
91: $count[z] \leftarrow count[z] + 1$ ▷ *the number of candidates z*
92: **for all** unique $(m, z) \in cand\text{-}next$ **do** ▷ *transfer from least voted cand.*
93: $v[z] \leftarrow v[z] + transfer\text{-}vote \cdot (count[z]/count[m])$
94: $X \leftarrow null$
95: **return** S ▷ *return the set of winning candidates*

Algorithm 2: Threshold Single Transferable Vote (Threshold-STV) - Part 2

B. Byzantine Fault Tolerant Single Transferrable Vote

Alg. 1 presents the main functions of the BFT-STV smart contract that the governors can invoke whereas Alg. 2 is the classic STV algorithm adapted to progress in a partially synchronous [36] environment and despite the presence of up to t Byzantine voters, hence its name STV_B.

a) Commit votes: Initially, the governors cast their hashed ballots by invoking the function commitVote($\cdot$) at line 16 of Alg. 1. This prevents the ballot content of each voter from being known to other voters until the election counting begins. This is to mitigate strategic voting. Once governors cast $n-t$ private votes in the form of ballot hashes, the BFT-STV smart contract emits a broadcast notifying governors that their respective ballots can be revealed to commence counting votes (lines 21-23).

b) Reveal votes: Governors/voters upon receiving the broadcast in line 23, invoke the reveal($\cdot$) function parsing the plain ballot b and the hash of this ballot h (line 24). If (1) hash of b equals h and (2) h equals the hash of the ballot previously stored in the commit phase for the same voter, then the validity of the ballot b is checked. Upon successful validation, the ballot b is added to the list of ballot $ballots$ (lines 26-27). Note

that verifying the validity of a ballot involves checking that the governors have not voted for themselves on their ballots and there are no duplicated preferences. Once the smart contract receives $n - t$ well-formed ballots the change-committee$(\cdot)$ function is invoked (line 29).

c) Count votes: The change-committee function starts by computing the score of the valid candidates as the number of votes they receive at lines 31–33. Valid candidates are initially selected through KYC (§III-D2) before being periodically voted upon by governors to be elected as the next set of governors. A preference pointer is initialized to the first preference of each ballot at line 34. Then a new round of the STV election process starts (lines 35–38). This execution stops once the committee of new governors is elected (line 36). If before the targeted committee is elected, the number of eliminated candidates has reached a maximum and no more candidates can be eliminated to achieve the target committee size, then the STV election stops (line 39). The remaining non-eliminated candidates are elected by decreasing order of preferences at lines 40–43 until the target committee size is reached. Finally, the smart contract emits the committee of elected candidates (line 44), which notifies the replicas of the election outcome.

C. Classic STV with the Byzantine quota

Alg. 2 presents the classic STV algorithm but using the new Byzantine quota q_B by electing candidates whose number of votes exceed q_B (line 55). This algorithm executes two subsequent phases: in the first phase (lines 54–71) the algorithm elects the candidates whose number of votes exceeds the quota $q_B = \frac{n-t}{k+1}$; in the second phase (lines 72–94), the algorithm eliminates the least preferred candidate if no candidates received a number of votes that exceeds the quota. In each round of STV function call (line 37), when a candidate exceeds the quota (line 55), their excess votes are transferred to the next eligible preferences of the ballots that contain the candidate (line 71). In each round of ballot iteration, if no candidate has reached the quota, the candidate with the least vote(s) is eliminated (line 73). This candidates' excess votes are transferred to the next eligible preference of the ballots that contain the candidate that received the least votes (line 93). The elimination of candidates stops when no more candidates can be eliminated to achieve the committee size (line 39). At this point, even though the remaining candidates did not receive enough votes to reach the quota, they are elected as part of the committee (line 43).

D. Election sharding

As many modern blockchains cannot handle demanding smart contract workloads [46], executing the NP-hard BFT-STV algorithm on a smart contract with many voters and candidates is a challenging task. To tackle this challenge, when the number of voters n and candidates m are large, we shard the BFT-STV algorithm into smaller sub-elections. More specifically, our election sharding approach groups voters and candidates into N groups of equal size. Thus, a voter can only vote for candidates in their group. If there are N groups, each group can elect k/N governors to the committee such that the total elected governors is k in the end.

From a voting theory perspective, this sharding approach is exactly the same as dividing the seats in a main legislative body into electoral districts such that voters living in a specific electoral district only cast ballots for candidates from the same district. Thus, providing better representation in the legislative body for populations in each electoral district.

From a computer science perspective, sharding the BFT-STV elections as mentioned above makes the ordinal ballot sizes smaller as a voter only orders preferred candidates in their own group. This decrease in ballot sizes reduces CPU and memory usage of the blockchain node during BFT-STV contract execution helping us achieve election outcomes fast without losing ballots cast by voters. With this election sharding approach (§VI), we see that the BFT-STV smart contract can elect 200 governors from 500 candidates within 6-12 minutes when 1000 voters cast ballots. Thus, our approach is faster than many committee election methods [3], [47], [8].

E. Proofs of proportional governance

In this section, we show that BFT-STV (Alg. 1 and 2) solves the proportional governance problem (Def. 1). To this end, the first theorem shows that the BFT-STV protocol ensures *Proportionality*. As mentioned in §III-B, recall that n, m and k denote the number of voting governors, the number of candidates and the targeted committee size, respectively. Note that the proof holds even if Byzantine voters vote in the worst possible way.

Theorem 1. *The BFT-STV multi-winner election protocol satisfies Proportionality.*

Proof. By examination of the code of Alg. 1 and 2, the only difference between BFT-STV and STV is the number of votes needed to elect a candidate. STV typically starts with n received ballots whereas the BFT-STV starts the election as soon as $(n - t)$ ballots are received (line 28 of Alg. 1), where t is the upper bound on the number f of Byzantine nodes and n is the total number of governors eligible to vote. This number of BFT-STV ballots is distributed among a larger number of candidates m. This can result in less than k candidates receiving enough votes to reach the classic STV quota where k is the size of the committee. By the Proportionality definition (Def.§III-B), we need to show that if $j \cdot (n - t)/(k + 1)$ voters put the same s candidates as the top s candidates in their ballot preference, then j of those s candidates will still be elected. The proof follows from [48, p. 48–49]: line 73 of Alg. 2 indicates that by elimination, and lines 40-43 of Alg. 1 indicates by electing the remaining non eliminated candidates in decreasing preference order, we elect the required k seats if k candidates cannot reach the q_B quota. Thus, we still elect the top j candidates such that $j = s = k$, satisfying proportionality.

The next theorem shows that the BFT-STV protocol ensures *Non-dictatorship* as defined in Def. 1.

Theorem 2. *The BFT-STV multi-winner election protocol satisfies Non-dicatorship.*

Proof. The proof shows the existence of an input of correct nodes for which a single adversary controlling f Byzantine nodes cannot have its preference b_a be the winning committee. Let $b_a[-1]$ be the least preferred candidate of the adversary, we show that there exist preferences $b_1, ..., b_{n-f}$ from correct nodes such that the winning committee includes $b_a[-1]$. The result then follows from the assumption $k < m$.

By examination of the pseudocode, the winning committee is created only after receiving $n-t$ correctly formatted ballots (line 28 of Alg. 1). By assumption, there can only be at most $f \leq t < n/3$ ballots cast by Byzantine nodes. As a result, among all the $n-t$ received ballots, there are at least $n-2t > n/3$ ballots cast from correct nodes. In any execution, an adversary controlling all the Byzantine nodes could have at most f ballots as the adversary cannot control the ballot cast by correct nodes. Let $b_1, ..., b_{n-f}$ be the ballots input by correct nodes to the protocol such that their first preference is the least preferred candidate of the adversary, i.e., $\forall i \in \{1, n-t\} : b_i = b_a[-1]$. Because $f \leq t < n/3$, we know that $b_a[-1]$ will gain more votes than any of the other candidates, and will thus be the first to be elected (line 55 of Alg. 2). By assumption, we have $k < m$, which means that there is a candidate the adversary prefers over $b_a[-1]$ that will not be part of the winning committee. Hence, this shows the existence of an execution where despite having an adversary controlling f Byzantine nodes, the adversary preference is not in the winning governance committee.

V. Automatic Governance Reconfiguration

In this section we present our governance reconfiguration protocol to mitigate bribery attacks that could form an oligarchy among governors. Subsequently, we prove that our reconfiguration protocol solves the governance reconfiguration safety (Def. 2).

Offering proportionality and non-dictatorship is not sufficient to cope with an adaptive adversary. In order to mitigate bribery attacks, we now propose a governance reconfiguration that complements the BFT-STV algorithm. The subsequent governance reconfiguration protocol assumes that all blockchain nodes in a network has the BFT-STV smart contract deployed at bootstrap time.

Alg. 3 allows switching from the current governance committee S_0 to the new governance committee S elected with the BFT-STV smart contract (§IV) . Note once a governor $g : g \in S_0$ emits S (line 44 of Alg. 1), they immediately stop processing any further blocks.

Once a blockchain node (i.e., candidate, governor, client) receives from governor $g : g \in S_0$ newly elected governors S and a blockchain prefix (line 10 of Alg. 3), the reconfiguration protocol commences. Note that duplicate broadcasts received by the same governor are not considered. First, every received S from a governor $g : g \in S_0$ is added to $Elected$. Thus, $Elected$ stores all received S from current governors (line 12 of Alg. 3). Next, every blockchain prefix $Blockchain[start : end]$ received from a governor $g : g \in S_0$ is stored in BC (line 13 of Alg. 3). The blockchain prefix contains a chain of blocks where the start index $start$ is the first block decided in the blockchain of governor g when in S_0 while the end index end is the last block decided in the blockchain of g when S was emitted by g.

Once a governor broadcasted event S is received $n-t$ times (i.e., same S received $n-t$ times) from $n-t$ unique governors (line 17 of Alg. 3), that means at least $n-2t$ of the received S were from correct governors in the committee. Since $f \leq t < n/3$, S is the correct governor committee elected. When this condition is met, Alg. 3 executing on every blockchain node finds the block with the highest index decided by $n-t$ unique governors $g : g \in S_0$ using the blockchain prefixes received (lines 19-24 of Alg. 3).

Subsequently, the reconfiguration protocol closes the existing network connection with the previous governor committee (line 27 of Alg. 3). Then, every blockchain node connects with the new governor committee (line 28 of Alg. 3). Finally, if the blockchain node is also a governor elected in S, these governors initialize themselves with $B_{decided}$ which is the highest index block decided by $n-t$ governors from committee S_0.

For sake of simplicity, we consider that nodes connect to the IP addresses of the new governors. The implementation could be easily adjusted so that nodes connect to a specific

```
1:  initial:
2:      A is a set of IP addresses.
3:      BC is a set of blockchains s.t. Blockchain[start : end] ∈ BC.
4:      Elected: a set s.t. S ∈ Elected.
5:      count: a map between a governor sets received and its occurrences.
6:      Bcount: a map between a block and its occurrences in received prefixes.
7:      S: newly elected governor committee
8:      S_0: current governor committee
9:
10: upon receiving S, Blockchain[start : end] from a governor in S_0:
        ▷ recv. sc emits event from Alg.1, line 44 and bc prefix
11:     if g ∈ S_0 & received[g] == false then   ▷ prevents duplicate broadcast
12:         Elected ← Elected ∪ S
13:         BC ← BC ∪ Blockchain[start : end]
14:     for all S ∈ Elected do
15:         count ← count[S] + 1
16:         if count[S] = n − t then
17:             threshold ← S                    ▷ received same S from n − t
18:     if threshold == S then
19:         for all bc ∈ BC do
20:             for all B ∈ bc do
21:                 Bcount[B] ← Bcount[B] + 1    ▷ no. of block B in recv. prefix
22:                 if Bcount[B] == n − t & B.index > highestIndex then
        ▷ B is in n − t gov. chains
23:                     highestIndex ← B.index
24:                     B_decided ← B            ▷ decided block so far
25:         for all ip ∈ S do                    ▷ for IPs in committee
26:             A ← A ∪ {ip}                     ▷ add the IP address
27:         close-connect                        ▷ stop connections with current governors
28:         connect(A)                           ▷ connect with elected governors in A
29:         if my-ip ∈ S then                    ▷ if I'm an elected governor
30:             init(B_decided)                  ▷ init. governor with B_decided and its state.
31:         threshold ← NULL                     ▷ reset variables
32:         Bcount ← NULL
33:         count ← NULL
```

Algorithm 3: Governance reconfiguration at a blockchain node

node ID that uniquely identifies a node. Since every blockchain node connects with the newly elected governor committee, (1) clients can send requests to the new governor committee (2) governors can reach consensus on governance decisions and (3) governors can elect the next set of governors.

Theorem 3. *The governance reconfiguration (Alg. 3) satisfies the Governance Reconfiguration Safety property.*

Proof. By examination of Alg. 3 from the blockchain prefixes received from $n-t$ governors that sent S, each correct blockchain node finds the common block with the highest index $B_{decided}$ of all $n-t$ prefixes. This block is the highest confirmed/decided block by the governance committee S_0 (lines 19-24 of Alg. 3). If a correct local blockchain node is elected to the new governance committee S, then this node initializes with $B_{decided}$ (line 30 of Alg. 3). Every newly elected correct governor node in S initializes with the same $B_{decided}$. Thus, the first block locally stored after governance reconfiguration by any two correct governor nodes is equal satisfying our safety property.

VI. Evaluation of Byzantine Fault Tolerant Proportional Governance

A. World-scale evaluation

We evaluate our Byzantine Fault Tolerant Proportional Governance protocol on a world-scale to observe its feasibility. To this end, we integrated our solution to Ethereum PoA and Smart Redbelly Blockchain (SRBB) [27] which are two smart contract supporting blockchains on the slower and faster end of the blockchain spectrum. We used the Diablo blockchain benchmarking suite [46] that evaluates blockchains against pre-specified workloads. Our pre-specified workload consisted of 1000 voters (i.e., current governors) casting random ordinal ballots to 500 candidates to elect a committee of 200 governors using our BFT-STV smart contract. We employed 200 AWS c5.2xlarge EC2 instances of Ethereum PoA and SRBB [27], spanning 10 AWS regions and 5 continents. Each AWS instance represented 5 governors of the respective blockchain realising a total of 1000 governors (i.e., 200×5), a restriction we placed due to budgetary constraints. Finally, we used a transaction sending rate of 1000 TPS, and considered the number of Byzantine voters as t=333 ($t < n/3$).

Table II depicts the time taken in seconds for the BFT-STV smart contract to elect a committee of 200 governors when 1000 voters (i.e., current governors) cast random ordinal ballots to 500 candidates on Ethereum PoA and SRBB. Ethereum PoA takes 728 seconds (i.e., 12 minutes) to elect a committee of 200 governors while SRBB [27] which was recently found to yield better performance compared to modern blockchains like Algorand [7], Solana [25], and Avalanche [49], elected a committee of 200 governors within 358 seconds (i.e., 5.96 minutes).

Based on Table II, the BFT-STV algorithm executed on a smart contract was able to elect a committee of governors within 12 minutes in one of the slowest smart contract supported blockchains (i.e., Ethereum) and within half that time in a faster blockchain (i.e., SRBB [27]). In contrast, Polkadot [3] and Tron [47] elects a committee of governors in 24 hours and 6 hours respectively. EoS elects a committee of governors in 63 seconds [50] but elects only a small committee of 21 validators.

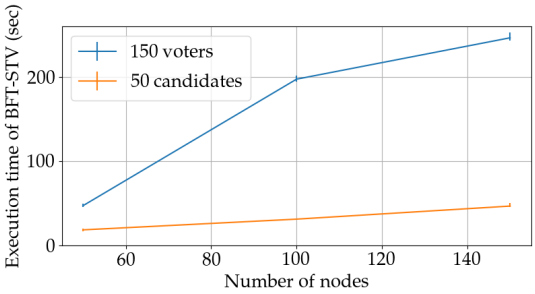

Fig. 3. The execution time of BFT-STV in SRBB as we vary the number of candidates (with 150 voters) and as we vary the number of voters (with 50 candidates)

Blockchain	#voters	#ballots	#candidates	#governors	time (seconds)
Ethereum PoA	1000	1000	500	200	728
SRBB	1000	1000	500	200	358

TABLE II
BFT-STV: The time in seconds for 200 geo-distributed nodes of Ethereum PoA and SRBB representing 1000 voters (current governors) to elect 200 new governors from 500 candidates.

B. Micro-benchmarking the BFT-STV smart contract

In Figure 3 we present the performance of the BFT-STV smart contract with varying numbers of voters/governors and candidates. More specifically, Figure 3 presents the average BFT-STV smart contract execution time in seconds on a single SRBB [27] node over 3 runs for different numbers n and m of voters and candidates, respectively.

In Figure 3, the top curve varies the number m of candidates whereas the bottom curve varies the number n of voters. More specifically, we varied the number n of voters and the number m of candidates from 50 to 150 to elect a committee of $k = m/2$ validators. Therefore, while we fixed $m = 50$ and varied n, we had to fix $n = 150$ to vary m up to 150. Varying m also allowed us to elect a varying committee of size $k = m/2$, showing the ability of BFT-STV to elect committees of dynamic size. We generated a random ordinal/ranked ballot for each voter in this benchmark. As our goal was to purely observe the BFT-STV smart contract execution, we only used a single node for this particular benchmark to avoid the execution times being distorted with the consensus times. We also considered $t < n/3$ (i.e., the number of Byzantine voters). The observations from Figure 3 are as follows:

1) First, we observe that the number of candidates m impacts the performance significantly with $n = 150$ voters, which confirms our expectation. However, we also observe that the raise decreases as m exceeds 100. We conjecture that this is due to the way the Ethereum Virtual Machine [24] in EVM-based blockchains garbage collects and alternates

between CPU resource usage for transaction execution and I/O usage to persist the information.

2) Second, we observe that when the number of voters increases with $m = 50$, the execution time increases sub-linearly: it doubles while the number of voters triples. This is because increasing the number n of voters helps candidates reach the quota q_B of votes rapidly without transferring the vote excess. Hence, the committee is elected faster than expected and raises the execution time only slightly.

VII. Related Work

In this section, we present the work related to blockchain governance. Table I, provides a summary of such blockchains. For the sake of brevity, we omit the discussion of blockchains that assume synchrony [51], [24], [40], [49], [19], [15], [52].

A. Proof-of-stake blockchain governance

Algorand [7] assumes a slowly adaptive adversary to mitigate bribery like we do (§III-D). Algorand offers governance through *sortition*, the act of electing governors randomly among a set of candidates. To mitigate bribery attacks, Algorand replaces governors at each step of the consensus protocol within a consensus round. The key advantage of the sortition is its non-interactive cryptographic technique that prevents adversaries from predicting future governors. However, Algorand does not aim at offering proportionality as it does not execute a proportional election to select governors. The PoS based sortition used in Algorand can lead to an oligarchy among governors as nodes with more stake have more probability of being elected as governors.

Polkadot [23] rotates its governors every *era*, a period that lasts about one day, with a multi-winner election. Unlike our solution that employs KYC, Polkadot exploits a nominated proof-of-stake (NPOS). In NPOS, the chances of being elected as a governor is proportional to the stake a candidate possesses. Thus, despite using a multi-winner election, Polkadot favors the wealthiest, leaving the potential for an oligarchy to be formed within the governance.

EOS [9] runs a delegated multi-winner approval voting system to elect 21 governors. As opposed to BFT-STV (§IV), EOS exploits delegated proof-of-stake (DPOS) where token holders elect governors by casting a vote with a weight proportional to the token holder's stake. Thus, EOS [9] also favors the wealthiest to be elected as governors leaving the potential for an oligarchy to be formed among the governors.

In summary, the aforementioned solutions do not offer proportionality as each vote is based on the wealth or assets the corresponding voter owns: the more they own the higher weight their vote gets. Given the Pareto Principle [53] stating that few users typically own most of the resources (as an example in 2021, the wealthiest 1% of US citizens owned about 1/3 of the total wealth [54]), these approaches have the risk of forming an oligarchy of governors.

B. Proof-of-work blockchain governance

Zilliqa [18] requires a candidate to solve a PoW puzzle and produce a reconfiguration block to join a committee of governors. Thus, the election of the governance committee favors powerful nodes capable of solving the PoW puzzle fast. Such PoW blockchain governance mechanisms can lead to an oligarchy in the governance where the oligarchy consists of governors with the highest computation power.

C. BFT blockchain governance

The vast majority of Byzantine fault tolerant (BFT) blockchains assume that the list of governors is selected by an external service. As a result, no proportionality is offered. ComChain [20] lists the public keys of governors in configuration blocks but assumes that the new lists of governors are proposed by an external service. Similarly, Tendermint/Cosmos [55] lists the public keys of governors in blocks but associates a voting power to each validator based on its stake, hence risking the same bribery attacks as other proof-of-stake blockchains (§VII-A). SmartChain [22] also stores the committee public keys in dedicated reconfiguration blocks but simply grants governor credentials to every requesting node, without requiring an election. Libra [21] mentions a similar reconfiguration service but no details are provided regarding the selection of governors or whether this selection offers proportionality. As far as we know other BFT blockchains have a static set of governors, which makes them more vulnerable to bribery attacks and the formation of an oligarchy, including Stellar [56], SBFT [57], Concord [58] and Quorum [26].

VIII. Conclusion

We presented proportional governance reconfiguration to mitigate the formation of an oligarchy of governors in blockchain governance committees. Proportional governance reconfiguration is the first solution that embeds the following two contributions: (1) preventing an oligarchy among governors using proportionality (Def. 1) and automatic governance reconfiguration, and (2) providing compatibility with a wide range of smart contract supported blockchains [26], [7], [59]. We proved the proportional governance reconfiguration ensures proportionality and non-dictatorship (Def. 1) and implemented proportional governance on Ethereum-PoA and Smart Redbelly Blockchain [27] which are two smart contract supporting blockchains. Our evaluation showed that our proportional governance solution implemented as BFT-STV on a smart contract (Alg. 1) can elect 200 governors within 6-12 minutes when 1000 voters cast ordinal ballots to 500 candidates.

Acknowledgements

This work is supported in part by the Australian Research Council Future Fellowship funding scheme (#180100496) entitled "The Red Belly Blockchain: A Scalable Blockchain for Internet of Things" and the Ethereum Foundation.

References

[1] F. Michelle, *Blockchain Governance*. Cambridge University Press, 2018, pp. 182–209.

[2] V. Zamfir, "Blockchain governance," in *Ethereum Community Conference*, 2019, accessed: 2021-05-28, https://www.youtube.com/watch?v=PKyk5DnmW50.

[3] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, and G. Wood, "Overview of polkadot and its design considerations," arXiv, Tech. Rep. 2005.13456, 2020.

[4] "Block producers ranking - real time statistics," accessed: 2020-11-14, https://eosauthority.com/producers_rank.

[5] L. Kiffer, D. Levin, and A. Mislove, "Stick a fork in it: Analyzing the ethereum network partition," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, pp. 94–100.

[6] N. Webb, "A fork in the blockchain: Income tax and the bitcoin/bitcoin cash hard fork," *North Carolina Journal of Law & Technology*, vol. 19, no. 4, 2018.

[7] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Operating Syst. Principles*, 2017, pp. 51–68.

[8] "The eth2 upgrades," accessed: 2020-11-14, https://ethereum.org/en/eth2/.

[9] "EOS.IO technical white paper v2," accessed: 2020-12-07, https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md#consensus-algorithm-bft-dpos.

[10] R. Dixon, "Fair criteria and procedures for establishing legislative districts," *Policy Studies Journal*, vol. 9, no. 6, p. 839, 1981.

[11] N. Tideman, "The single transferable vote," *Journal of Economic Perspectives*, vol. 9, no. 1, pp. 27–38, March 1995.

[12] "Proportional representation voting systems of australia's parliaments," 2021, accessed:2021/06/04 –https://www.ecanz.gov.au/electoral-systems/proportional.

[13] R. Guerraoui and A. Schiper, "Fault-tolerance by replication in distributed systems," in *International conference on reliable software technologies*. Springer, 1996, pp. 38–57.

[14] M. C. Pease, R. E. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[15] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 931–948. [Online]. Available: https://doi.org/10.1145/3243734.3243853

[16] K. J., "Tendermint: Consensus without mining," 2014.

[17] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[18] "The zilliqa technical whitepaper," https://docs.zilliqa.com/whitepaper.pdf. [Online]. Available: https://docs.zilliqa.com/whitepaper.pdf

[19] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *IEEE Symposium on Security and Privacy (S&P)*, 2018, pp. 583–598.

[20] G. Vizier and V. Gramoli, "Comchain: A blockchain with Byzantine fault tolerant reconfiguration," *Concurrency and Computation, Practice and Experience*, vol. 32, no. 12, Oct 2019.

[21] S. Bano, M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," 2019, accessed: 2019-10-01, https://developers.libra.org/docs/assets/papers/libra-consensus-state-machine-replication-in-the-libra-blockchain.pdf.

[22] A. Bessani, E. Alchieri, J. Sousa, A. Oliveira, and F. Pedone, "From byzantine replication to blockchain: Consensus is only the beginning," in *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2020, pp. 424–436.

[23] A. Cevallos and A. Stewart, "A verifiably secure and proportional committee election rule," *arXiv e-prints*, pp. arXiv–2004, 2020.

[24] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2015, yellow paper.

[25] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0. 8.13," *Whitepaper*, 2018.

[26] J. Chase, "Quorum whitepaper," accessed: 2020-12-04, https://github.com/ConsenSys/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf.

[27] D. Tennakoon, Y. Hua, and V. Gramoli, "Smart Redbelly Blockchain: Reducing congestion for Web3," in *Proceedings of the 37th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2023.

[28] F. T. Lui, "An equilibrium queuing model of bribery," *Journal of Political Economy*, 1985.

[29] J. Bonneau, "Why buy when you can rent? Bribery attacks on Bitcoin-style consensus," in *Financial Cryptography and Data Security Workshops*, 2016, pp. 19–26.

[30] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *CCS*, 2016.

[31] D. Black, *The Theory of Committees and Elections*. Cambridge University Press, 1958.

[32] M. Dummett, *Voting Procedures*. Oxford University Press, 1984.

[33] J. L. . I. D. Hill, "To advance the understanding of preferential voting system - notes on the droop quota," *Voting matters*, 2007.

[34] D. Woodall, "Properties of preferential election rules," in *Voting Matters*, 1994, accessed: 04/05/2021, https://www.votingmatters.org.uk/ISSUE3/P5.HTM.

[35] P. Faliszewski, P. Skowron, A. Slinko, and N. Talmon, *Multiwinner Voting: A NewChallenge for Social Choice Theory*. Lulu.com, 2017.

[36] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. pp.288–323, 1988.

[37] K. J. Arrow, "A difficulty in the concept of social welfare," *Journal of Political Economy*, vol. 58, no. 4, pp. 328–346, 1950.

[38] A. S. Tanenbaum and M. van Steen, *Distributed systems - principles and paradigms, 2nd Edition*. Pearson Education, 2007.

[39] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[40] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, "Solida: A blockchain protocol based on reconfigurable byzantine consensus," in *21st International Conference on Principles of Distributed Systems*, 2017, pp. 25:1–25:19.

[41] E. Elkind, P. Faliszewski, P. Skowron, and A. Slinko, "Properties of multiwinner voting rules," *Social Choice and Welfare*, vol. 48, no. 3, pp. 599–632, 2017.

[42] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *34th Annu. Int. Conf. the Theory and Applications of Crypto. Techniques*, 2015, pp. 281–310.

[43] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 1–11.

[44] poa.network, "Poa validator dapp," 2022, accessed: 2022-22-04 - https://validators.poa.network/poa-dapps-validators.

[45] T. Crain, C. Natoli, and V. Gramoli, "Red belly: a secure, fair and scalable open blockchain," in *IEEE Symposium on Security and Privacy (S&P)*, May 2021, pp. 1501–1518. [Online]. Available: https://www.computer.org/csdl/pds/api/csdl/proceedings/download-article/1t0x9nljvwI/pdf

[46] V. Gramoli, R. Guerraoui, A. Lebedev, C. Natoli, and G. Voron, "Diablo: A benchmark suite for blockchains," To appear in 18th European Conference on Computer Systems (EuroSys), 2023. [Online]. Available: https://gramoli.github.io/pubs/Eurosys23-Diablo.pdf

[47] D. Staff, "Tron governance: How to vote using trx," accessed:2022-10-14, https://decrypt.co/resources/tron-governance-how-to-vote-using-trx.

[48] S. Janson, "Thresholds quantifying proportionality criteria for election methods," *arXiv preprint arXiv:1810.06377*, 2018.

[49] T. Rocket, "Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies," Tech. Rep., 2018, accessed: 2021-12-01. [Online]. Available: https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV

[50] G. A. F. Rebello, G. F. Camilo, L. Guimaraes, L. A. C. de Souza, and O. Duarte, "Security and performance analysis of quorum-based blockchain consensus protocols," *Electrical Engineering Program, COPPE/UFRJ, Tech. Rep*, 2020.

[51] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2008.

[52] V. K. Bagaria, S. Kannan, D. Tse, G. C. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, 2019, pp. 585–602.

[53] V. Pareto, *Cours d'Économie Politique: Nouvelle édition par G.-H. Bousquet et G. Busino*. Librairie Droz, 1964.
[54] R. Frank, "The wealthiest 10% of americans own a record 89% of all u.s. stocks," https://www.cnbc.com/2021/10/18/the-wealthiest-10percent-of-americans-own-a-record-89percent-of-all-us-stocks.html. [Online]. Available: https://www.cnbc.com/2021/10/18/the-wealthiest-10percent-of-americans-own-a-record-89percent-of-all-us-stocks.html
[55] tendermint.com, "Tendermint," accessed: 2021-07-21 https://docs.tendermint.com/master/.
[56] M. Lokhava, G. Losa, D. Mazières, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowsky, and J. McCaleb, "Fast and secure global payments with stellar," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019, pp. 80–96.
[57] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: a scalable and decentralized trust infrastructure," in *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.
[58] VMware, "Concord," accessed: 2020-11-28, https://github.com/vmware/concord.
[59] "Ethereum Proof-of-Authority Consortium - Azure," https://docs.microsoft.com/en-us/azure/blockchain/templates/ethereum-poa-deployment. [Online]. Available: https://docs.microsoft.com/en-us/azure/blockchain/templates/ethereum-poa-deployment