BANCA D'ITALIA
EUROSISTEMA

Mercati, infrastrutture, sistemi di pagamento

(Markets, Infrastructures, Payment Systems)

Byzantine Fault Tolerant consensus with confidential quorum certificate for a Central Bank DLT

by Marco Benedetti, Francesco De Sclavis, Marco Favorito, Giuseppe Galano, Sara Giammusso, Antonio Muci and Matteo Nardelli

July 2024

Number 50

BANCA D'ITALIA
EUROSISTEMA

Mercati, infrastrutture, sistemi di pagamento
(Markets, Infrastructures, Payment Systems)

Byzantine Fault Tolerant consensus with confidential quorum certificate for a Central Bank DLT

by Marco Benedetti, Francesco De Sclavis, Marco Favorito, Giuseppe Galano, Sara Giammusso, Antonio Muci and Matteo Nardelli

Number 50 – July 2024

ISSN 2724-6418 (online)
ISSN 2724-640X (print)

Banca d'Italia
Via Nazionale, 91 - 00184 Rome - Italy
+39 06 47921

Designed and printing by the Printing and Publishing Division of the Bank of Italy

Byzantine Fault Tolerant consensus with confidential quorum certificate for a Central Bank DLT

Marco Benedetti,* Francesco De Sclavis,* Marco Favorito,* Giuseppe Galano,* Sara Giammusso,* Antonio Muci* and Matteo Nardelli*

Abstract

Some essential characteristics of Distributed Ledger Technologies (DLTs), such as programmability and the use of advanced cryptographic techniques, can also be effectively utilized in controlled environments, overseen by a central authority or a group of delegated entities. This is especially relevant in the formal financial sector, and in other settings where compliance with regulation is of the essence. Technically, this requires DLTs to be deployed in *permissioned* or private versions, where only a set of authorized participants, called validators, are allowed to approve or reject transactions on the shared ledger. This is in contrast to *permissionless* or public versions, where no authorization is required.

All DLTs, whether *permissionless* or permissioned, function based on a cooperative decision process designed to reach an agreement among validators about the next state of the ledger. This process, known as a *consensus protocol*, is a critical component of DLTs because it enables validators to maintain uninterrupted operation of the system without human intervention, even if some validators are compromised or become disconnected from the network. In *permissionless* environments, achieving consensus is resource intensive: Since validators' identities are not known, they must prove their honesty by either committing significant computational power (Proof-of-Work, PoW) or substantial financial capital (Proof-of-Stake, PoS). In permissioned contexts, consensus can be achieved without the need for extensive resource commitments. In particular, Proof-of-Authority (PoA) consensus protocols rely on a predetermined group of validators, who are entrusted with the power to accept or reject transactions proposed by participants. Typically, these validators achieve consensus through *qualified majority voting*.

In this paper, we present the FBFT (FROSTed Byzantine Fault Tolerance) protocol, a novel approach to PoA meant to strengthen the security of the ledger, its tolerance to faults or attacks, and the confidentiality of validators. It combines the Practical Byzantine Fault Tolerance (PBFT) algorithm, a well-known contribution from distributed systems literature, with the Flexible Round-Optimized Schnorr Threshold (FROST) signature scheme, a recent finding in cryptographic research.

Leveraging state-of-the-art privacy-enhancing techniques, FBFT builds a collective agreement certificate (or *"joint cryptographic signature"*), which represents the endorsement of a given set of transactions by a quorum of validators. In addition, it provides strong guarantees of tolerance to *Byzantine faults* – situations where some validators may stop functioning or behave dishonestly, possibly due to software bugs or cyber-attacks. Finally, it preserves the confidentiality of validators: Their number and identities is only known to the central authority and its delegates, and not leaked to DLT participants. The advantage is a reduction in the risk of attacks targeted at specific validators.

We integrate our FBFT protocol into the code of a Bitcoin-like blockchain, effectively adapting its consensus component to a permissioned context, and we evaluate its performance across a variety of geographically distributed, realistic scenarios. To demonstrate its practicality and encourage further

* Directorate General for Information Technology, Banca d'Italia.

research, we provide an *open-source implementation of our DLT*. To the best of our knowledge, this is the first time that a Central Bank releases in open source a distributed consensus algorithm developed entirely in-house.

The resulting system, although experimental and lacking features expected of production-ready solutions, can be seen as an alternative platform for a distributed, resilient transactional system: Operated by a set of trusted actors, distributed at geographic scale, it holds potential for mission-critical applications, such as wholesale and retail Central Bank Digital Currencies, and – in perspective – asset tokenization schemes.

Keywords: Distributed ledger; blockchain; Byzantine consensus; confidential quorum; threshold signatures.

Sintesi

Alcune funzionalità delle tecnologie a registro distribuito (DLT), come la programmabilità e l'uso di crittografia avanzata, possono essere riutilizzate anche in ambienti controllati da un'autorità centrale, o da un insieme di autorità, come ad esempio nel settore finanziario. In tali contesti, è necessario implementare una DLT in versione *permissioned*, in cui solo un sottoinsieme dei partecipanti, anche detti validatori, sono autorizzati ad approvare o rifiutare transazioni. Al contrario, nei contesti *permissionless*, tale autorizzazione non è necessaria.

Tutte le DLT, siano esse *permissionless* o *permissioned*, funzionano grazie ad un processo decisionale cooperativo finalizzato a raggiungere un accordo sul prossimo stato del registro. Tale processo è chiamato protocollo di consenso e rappresenta un componente fondamentale di ogni DLT, in quanto consente ai validatori di operare senza interruzioni anche nel caso in cui alcuni di essi vengono compromessi o risultano disconnessi dalla rete. Nei contesti *permissionless*, raggiungere il consenso può richiedere un ingente consumo di risorse computazionali (*Proof-of-Work, PoW*) oppure finanziarie (*Proof-of-Stake, PoS*), ma nei contesti *permissioned* tale consumo non è necessario. In particolare, gli algoritmi basati su *Proof-of-Authority (PoA)* funzionano grazie al fatto che esiste un gruppo di validatori fidati, a cui viene data la capacità di accettare o rifiutare le transazioni proposte dai partecipanti. In genere, tali validatori raggiungono il consenso tramite votazione a maggioranza.

In questo lavoro, si presenta il protocollo FBFT (*FROSTed Byzantine Fault Tolerance*), che rappresenta un nuovo approccio al consenso *PoA*, in grado di garantire la confidenzialità dei validatori. Esso combina l'algoritmo *Practical Byzantine Fault Tolerance* (PBFT), un noto risultato in ambito sistemi distribuiti, con lo schema di firma a soglia chiamato *Flexible Round-Optimized Schnorr Threshold* (FROST), un recente contributo di crittografia.

Grazie a FROST, il protocollo FBFT certifica il raggiungimento del consenso utilizzando una firma congiunta, che rappresenta l'approvazione di un insieme di transazioni da parte di un quorum di validatori. Inoltre, grazie a PBFT, fornisce tolleranza a fallimenti cosiddetti "bizantini", in cui alcuni validatori smettono di funzionare o addirittura mostrano comportamenti malevoli, magari a causa di un guasto software o di un attacco cyber. Infine, FBFT preserva la confidenzialità dei validatori, in quanto il loro numero e le loro identità sono note solo all'autorità centrale o ai loro delegati, riducendo il rischio di attacchi mirati.

Il lavoro mostra come il protocollo FBFT possa essere integrato nel codice di una *blockchain* pubblica, adattandola ad un contesto *permissioned*, e valuta le performance del sistema in

scenari realistici e geograficamente distribuiti. Per incoraggiare ulteriori ricerche, tutto il codice sorgente viene pubblicato in *open-source*; è la prima volta che una Banca Centrale rende liberamente disponibile un algoritmo di consenso sviluppato interamente *in-house*.

Il sistema risultante, sebbene sperimentale e non sviluppato per essere *production-ready*, può essere visto come una piattaforma alternativa per una infrastruttura transazionale distribuita e resiliente: gestito da un insieme di attori fidati e distribuiti su scala geografica, è potenzialmente utilizzabile in applicazioni *mission-critical*, come le *Central Bank Digital Currencies* all'ingrosso e al dettaglio, oppure, in prospettiva, per la tokenizzazione di *asset* finanziari.

CONTENTS

1 Introduction[1]

The announcement of cryptoasset-inspired "stablecoins" by private companies and the prospective issuance of Central Bank Digital Currencies for retail use—i.e., CBDCs— (Cai *et al.*, 2021; Chaum *et al.*, 2021; European Central Bank, 2020, 2023; Federal Reserve, 2022), coupled with the unabated diffusion of blockchain-based digital assets, have reignited the interest in consensus protocols amenable to permissioned blockchains.

In this paper, we envision a distributed service provider that operates a modern, blockchain-based, programmable, and transactional engine, exhibiting high availability and strong fault tolerance. Each node (or small group of nodes) may be managed by independent actors, far removed from each other, either geographically or legally. These actors, which do not necessarily trust each other, share a common interest that would be perfectly served, technically, by a distributed ledger with no centralization point: Everyone enjoys equal rights, duties, and capabilities, and contributes to the system resilience. Nodes may even reside in different jurisdictions and conform to different laws, albeit under some shared regulatory framework.

These motivations hold for most permissioned Distributed Ledger Technology (DLT) platforms, and for both payment and non-payment domains. In this work, we specifically focus on *Bitcoin* [2] and on *digital payments*. Indeed, our target use case would be a speculative DLT-based payment system whose high availability and fault/attack tolerance rests upon a distributed platform operated *cooperatively* by several Central Banks in a given monetary area, as envisioned and described in Urbinati *et al.*, 2021, (Section 2.4 - The itCoin platform).

Instead of developing a new DLT platform from scratch (as done, for example, by Meta with the Libra, now Diem, blockchain–see Amsden *et al.*, 2020), we focus on an existing, largely deployed, open-source DLT platform, i.e., Bitcoin. In our permissioned setting, we can inherit the huge ecosystem of knowledge and applications that has been developed for the Bitcoin core infrastructure during the past decade. For example, the Lightning Network protocol (Poon *et al.*, 2016) is particularly promising in the digital payment domain (for its strong privacy and scalability properties), while other technologies are potentially interesting for relevant use cases (see Section 7). Nevertheless, it is widely known that Bitcoin has not been designed for a permissioned setting: It requires quite a few adaptations in order to properly work with a small set of validators, the most substantial of which is the replacement of its consensus algorithm.

Bitcoin (Nakamoto, 2008) is a peer-to-peer payment network launched in 2009: It implements a digital asset that does not rely on trusted third parties to guarantee its *scarcity* or to prevent *double spending*. To update the shared ledger, Bitcoin employs a decentralized *consensus protocol* among anonymous participants, based on Proof-of-Work (PoW). In PoW, validator votes (on what the next state of the system is) can be cast by just anyone, but each vote implies a substantial consumption of real-world resources (e.g., time, hardware, energy) to solve a computationally hard problem, whose solution is required to make the vote valid. This "costly postage stamp" of sort is key to preventing *sybil attacks* in open, anonymous settings[3]. For sure, in a permissioned setting with few validators, PoW would be not only very inefficient in terms of resource consumption, but also not safe: The resources sufficient to outcompete a

1 We would like to thank Sara Corbo and Claudia Biancotti for their feedback that helped improve this publication. All errors are the authors' sole responsibility.

2 A lexical note: This research paper references Bitcoin (with capital 'B'), i.e., a set of open source blockchain technologies, which are adapted for use in a permissioned, regulated setting, overseen by central authorities or delegated entities. It does not refer to bitcoin (with lowercase 'b'), i.e., a highly risky speculative crypto-asset. For a similar distinction in a related domain, see, e.g., (Banque de France, 2021), where successful experimentations with the Ethereum technology are presented.

3 To subjugate a PoW system, an attacker would have to outcompete the rest of the network in terms of available resources and willingness to sacrifice them. This so called 51% attack has been widely studied in the literature Lee *et al.*, 2020; Saad *et al.*, 2020; Ye *et al.*, 2018.

small network are likely within reach for any motivated and sponsored attacker. As we will show in the paper, if it is possible to identify a small set of actors that end-users trust to cooperatively guarantee scarcity and to prevent double spending, then a Bitcoin-like blockchain can be grown via, e.g., a consensus based on Proof-of-Authority (PoA).

To design our PoA consensus algorithm, we draw results from the scientific literature on distributed systems. By borrowing and modifying an existing Byzantine Fault Tolerant (BFT) consensus algorithms, we achieve high availability and tolerance to the so-called "Byzantine" faults, i.e., failure scenarios in which some nodes in the federation stop functioning (e.g., due to a software error) or start behaving incorrectly or maliciously (due to, e.g., a cyber-attack).

To summarize, we ask ourselves: Is the unorthodox notion of "*Precisely Bitcoin, minus its traditional consensus algorithm (PoW), plus identifiable third parties, in a permissioned setting*" a technically consistent one? Our goal is precisely to inherit *verbatim* all the algorithms, data structures, cryptography, and software from Bitcoin, getting rid of merely the ingredients (chiefly PoW) that are unnecessary/undesirable in a *permissioned setting*. In the paper, we show how to inject new algorithmic ingredients into Bitcoin while maximizing its codebase reuse, in order to inherit its technical virtues and strengths even after PoW is excised.

1.1 Scope and Contribution

We target settings where there are some (from 4 to $\approx$ 20) privileged and trusted nodes in charge of accepting and validating all transactions. All other participants can submit transactions and receive ledger updates. Before accepting a new ledger update, participants verify that it contains a sort of *quorum certificate*, testifying that it has been issued by a quorum of trusted validators. Digital signatures can be used to create such certificate and, in particular, *aggregated threshold signatures* can introduce the additional property of so-called *quorum confidentiality*: An aggregated signature appears as a single signature on behalf of a quorum of nodes that conceals the identify of the individual nodes that actively participated into the signing.

We focus only on the foundational issue of the consensus and signing protocol at the "on-ledger" layer, i.e., on designing and developing a working PoA-based BFT algorithm, which also achieves confidentiality of the validators network and quorum certificate, and is meant to sustain the growth of a permissioned but otherwise Bitcoin-like blockchain.

Our major contribution is to show how three fairly sophisticated protocols, coming from different communities—namely PBFT (*Practical Byzantine Fault Tolerance*, Castro *et al.*, 2002) from the distributed system research, FROST (*Flexible Round-Optimized Schnorr Threshold Signatures*, Komlo *et al.*, 2021), a result of recent cryptography studies, and the Bitcoin technological stack from the crypto-assets communities—can be combined to make them interlock neatly with one another. From such pooling, a permissioned Bitcoin-derived DLT emerges, with strong fault tolerance and a confidential quorum certificate. To the best of our knowledge, this is the first time algorithms such as PBFT and FROST are combined and adapted to a PoA setting that retains the wealth of technical tools accrued by Bitcoin. The major technical challenge to overcome is that a simple juxtaposition of PBFT and FROST does not work: Issues arise during distributed signature because the possible reluctance of (faulty or malicious) nodes to sign blocks is something PBFT is unaware of and FROST is unable to deal with.

To address these issues, we detail a novel Certified Byzantine Consensus protocol, named FBFT, that combines PBFT and FROST to finalize a block with a quorum of signatures aggregated into a single one (improving confidentiality and block space efficiency). We also evaluate our solution using a prototype implementation of the full system, which is made available in

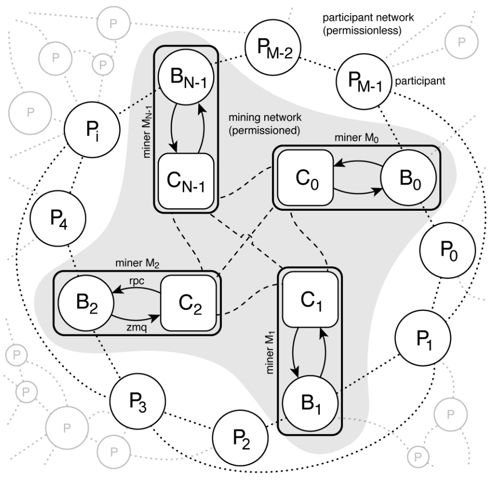


Figure 1. A permissioned mining network ($N = 4$) and a permissionless participant network.

open-source[4]. Further details can be found in Benedetti *et al.*, 2022.

The rest of this paper is organized as follows: Sect. 2 offers a high-level view of our architecture; Sect. 3 recalls some preliminary concepts; Sect. 4 presents FBFT; Sect. 5 evaluates the proposed solution; Sect. 6 reviews the related literature; Sect. 7 outlines future research directions, and Sect. 8 concludes the paper.

2 System Model and Requirements

2.1 High-level architecture

Our architecture is composed of a *participant network* and a *mining network*, each with different properties (see Fig. 1).

Participant and mining network. The *participant network* is composed of *participant nodes*, noted $P_0, \ldots, P_{M-1}$, which run a modified Bitcoin protocol (Sect. 4.3). Each participant node receives, validates, and stores a copy of the blockchain. The participants form a *permission-less* network, without a predefined topology or size. The bidirectional communication channels among them (dotted lines in Fig. 1) are used to propagate blocks and messages via gossiping, as in Bitcoin. The rounded rectangles inside the gray area are *mining*[5] *nodes*, or "miners" (there are 4 of them in Fig. 1). Each miner $M_i = (B_i, C_i)$ is composed of a *bridging node* B_i and a *consensus node* C_i, running on the same host and connected by synchronous *bridging channels* (see next). Each miner is operated by one member of a federation of N trusted actors[6], called *validators*. While the bridging node of each miner runs the same protocol as any other participant node (in particular, it collects transactions to be validated from participants and propagates new valid blocks to others as soon as it gets aware of them), the consensus node runs the certified Byzantine consensus protocol described in Sect. 4. Miners are connected to each other in a full mesh topology; the resulting *permissioned* network is called the *mining network* (everything within the gray area in Fig. 1). This is a peer-to-peer network too: Mining nodes are equivalent to each other, with no one playing any special role. The communication links among mining nodes (dashed lines) are bidirectional channels used to exchange authenticated messages required

4 https://bancaditalia.github.io/itcoin
5 The term "mining" is etymologically incongruous in our context where trusted nodes do not operate to *mining* any reward; however, we stick to them for historic reasons and for their close association with Bitcoin.
6 We target settings in which N is expected to be between 4 and ≈ 20.

by the consensus protocol (see Sect. 4).

Bridging channels. In between the bridging node and the consensus node of each miner, there are 2 host-local, synchronous channels (solid, oriented arcs in Fig. 1), acting as *bridging channels*: One uses an RPC protocol, whereby the consensus node takes the initiative to interact with the corresponding bridging component to, e.g., obtain a candidate template block, sign a block, ask to broadcast a block (see Sect. 4.1 and the self-loop messages in Fig. 2). The other bridging channel adopts a publish/subscribe model over the ZMQ protocol: The consensus node subscribes to the bridging node in order to get the mining federation notified of occurrences of new signed blocks. These interaction models and protocols allow maximum Bitcoin reuse because they leverage the standard Bitcoin core APIs exposed by B_i.

Roles and coupling. The mining network is a service provider: Its goal is to collect transactions from participants, reach a consensus on which ones to include in new blocks, and then deliver signed blocks back to participants, who will add them to their local blockchains. Thanks to the properties of the consensus and signing protocols, this network appears to the participants as a single mining entity. Dually, the participant network acts as a single virtual client submitting transactions to the blockchain managed by the mining nodes, and expecting such transactions to be timely validated. The participant network is reliably connected to the miner network via a few standard Bitcoin-like (P_j, B_k) channels freely established by at least some participant P_j towards one or more of the bridging nodes B_k; these channels are indistinguishable from regular channels within the permissionless network.

Failures. We assume a Byzantine failure model whereby F_B nodes can fail arbitrarily, with $F_B = \lfloor (N-1)/3 \rfloor$. The consensus we employ relies on synchrony to provide liveness, but not to provide safety. To avoid the FLP impossibility result[7] (Fischer *et al.*, 1985), we assume that (dashed) communication channels are weakly synchronous: Message delays among correct miners do not grow too fast and indefinitely, because a Global Stabilization Time (GST) event (Dwork *et al.*, 1988) eventually happens, after which the mining network behaves synchronously. Moreover, as in Garay *et al.*, 2015, we assume that all participants are able to synchronize in the course of a "round", and that each round includes a GST event. As long as the network is in a failed state, it may fail to deliver messages, delay/duplicate them, or deliver them out of order. We assume an adversary that can coordinate faulty nodes but cannot subvert cryptographic primitives.

2.2 Requirements

We call for our PoA consensus to exhibit the following properties.

R1 **Correctness** (or, validity, consistency). Each block needs to have content that is valid according to the rules of the blockchain, and must transition the blockchain from one valid state to another.

R2 **Safety** (or, agreement, deterministic finality). In a permissioned blockchain, safety forbids chain forks, i.e., different but valid versions of the most recent blocks of the same blockchain. This requires the Common Prefix Property (Garay *et al.*, 2015) to be deterministic instead of probabilistic. Specifically, at the end of a round, if a honest participant "prunes" $k \geq 0$ blocks from the tip of its chain, the probability that the resulting pruned chain is not a prefix of another honest participant chain is exactly 0 (instead of exponentially decreasing with k, as in PoW blockchains).

R3 **Liveness**. Within each round, new blocks must be produced every *block time* and prop-

7 The Fischer, Lynch, and Paterson (FLP) impossibility theorem states that, in an asynchronous system, it is impossible to design a *deterministic* consensus algorithm that satisfies agreement, termination, and fault tolerance.

agated to the participants network every *round time*. We use the Chain Growth property (Garay *et al.*, 2015), with parameters $\tau = \frac{\textit{round time}}{\textit{block time}} \in \mathbb{R}$ and $s \in \mathbb{N}$: For any honest participant, it holds that after any s consecutive rounds it adopts a chain that is at least $\lfloor \tau \cdot s \rfloor$ blocks longer.

R4 **Calmness**. The pace of block production is upper-bounded, which helps participants to form expectations on their resource requirements. If a Byzantine miner creates blocks at a rate significantly higher than $\frac{1}{\textit{block time}}$, it can cause participants to run out of resources, effectively carrying out a denial-of-service attack. We require that after any s consecutive rounds it adopts a chain that is at most $\lfloor \tau \cdot s \rfloor$ blocks longer.

R5 **Confidentiality**. At each round carried on *with no faulty miners*, [8] the mining network does not reveal information other than new valid blocks to the participants. Other information, e.g., the mining network configuration and the consensus quorum for block validation, should be kept hidden from the participants. This property can be used in addition to other anonymization mechanisms (e.g., at network level) to make targeted attacks against the miners harder.

R1-R3 have been already defined and studied in the context of blockchains (Garay *et al.*, 2020), whereas R4-R5 are peculiar to ours.

3 The FROST Signature Scheme

A (k, n)-*threshold signature* scheme, with $k \leq n$, requires that at least k participants over n cooperate to create a valid signature, i.e., it is not possible to create a valid signature with less than k participants. FROST is a threshold signature scheme that leverages the additive property of Schnorr signatures to quickly combine signatures into an aggregated one (Komlo *et al.*, 2021). The FROST signature scheme defines three main protocols: (i) a *key generation protocol* that creates secret shares for participants as well as public keys for signature verification; (ii) a *commitment protocol* that creates nonce/commitment share pairs for all participants; these commitments allow to prevent known forgery and replay attacks; (iii) a *signature protocol* coordinates the generation of the aggregated signature by signers. We briefly introduce these protocols, whose complete definition can be found in the original work (Komlo *et al.*, 2021).

Each participant M_i has a unique identifier $m_i \in \{1, \ldots, n\}$. Let $\mathbb{G}$ be a group of prime order q in which the Decisional Diffie-Hellman problem is hard, g be a generator of $\mathbb{G}$, and let H_1 and H_2 be cryptographic hash functions mapping to $\mathbb{Z}_q^*$, i.e., the set of non-zero integer numbers module q. We denote by $x \leftarrow A$ that x is selected uniformly randomly from set A.

Key Generation. Before signing any block, participants define secret and public keys. They share the same cipher suite that specifies the underlying prime-order group details and cryptographic hash function. The KeyGen protocol consists of two rounds. Afterwards, each participant M_i, with $i \in \{1, \ldots, n\}$, owns a secret share s_i, a public verification share $Y_i = g^{s_i}$, and the group's public key Y. The public verification share Y_i allows others to verify the participant signature shares; the group's public key Y enables the aggregate threshold signature verification, which depends on the set of participants n and the configured threshold k.

Commitment. In the commitment protocol, participants generate (secret) nonces for signatures and exchange their public commitments, which allow verifying the correct use of nonces. Each participant $M_i, i \in \{1, \ldots, n\}$, generates a pair of nonces $(d_i, e_i) \leftarrow \mathbb{Z}_q^* \times \mathbb{Z}_q^*$ and derives the public commitment shares $(D_i, E_i) = (g^{d_i}, g^{e_i})$.

Aggregated Signature. The aggregate signature protocol works in two rounds. First, each

8 This property is impossible to guarantee in rounds where Byzantine failures happen since the network configuration is known to each participant, and a Byzantine node can choose to reveal extra information to the outside world.

participant generates his signature share. Then, all participants' shares are combined to obtain the final signature. Let S be the set of participants in the signing process; the cardinality of S is α, with $k \leq \alpha \leq n$. Let $L = \langle(l, D_l, E_l)\rangle_{l=1}^{\alpha}$ be the list of α participants' commitments. When M_i receives the message to sign m, he can use his secret share s_i and L to compute his signature share z_i, which can then be sent to all other participants. Formally, M_i computes the set of binding values $\rho_l = H_1(l, m, L)$, $l \in \{1, \ldots, \alpha\}$, and derives the group commitment $R = \prod_{l=1}^{\alpha} D_l \cdot (E_l)^{\rho_l}$ and the challenge $c = H_2(R, Y, m)$. Then, M_i computes his signature share on m as $z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$, using (d_i, e_i) corresponding to $(i, D_i, E_i) \in L$, and S to determine the i-th Lagrange coefficient λ_i. Since nonces cannot be used multiple times, M_i deletes the $((d_i, D_i), (e_i, E_i))$ pair from his local storage. Then, M_i sends z_i to every other participant in S.

The second round starts when M_i receives all other signature shares z_l. For verification, M_i checks if the equality $g^{z_l} = R_l \cdot Y_l^{c \cdot \lambda_l}$ holds for each received z_l. If the verification is successful, M_i aggregates the signature shares locally by computing $z = \sum_{i \in S} z_i$. The resulting aggregated signature of m is $\sigma = (R, z)$, that can be verified as single-party signature.

4 Frosted Byzantine Fault Tolerance

In our permissioned setting, blocks need to be authenticated in front of the participants network. Therefore, we employ a *Certified Byzantine Consensus Algorithm*, which allows reaching consensus on a sequence of blocks even in face of Byzantine faults and produces a proof of validity for each block, which enables the participants to verify the whole blockchain validity. To reach a consensus on blocks, we leverage a modified version of PBFT, while to produce the proof, we leverage a Schnorr threshold signature scheme called FROST, which also provides private quorum accountability (Boneh *et al.*, 2022). It is worth noting that, in general, the set of block signers can differ from the set of nodes reaching the consensus.

4.1 Ordering blocks with PBFT

In a nutshell, PBFT is a state machine replication algorithm. It relies on a set of *replicas* to maintain a service state and to implement a set of *operations* onto it. The replicas move through a succession of configurations called *views*, which are numbered consecutively. In a view, one replica is the *primary* and the others are *backups*. *View changes* are carried out when it appears that the primary has failed. Service operations are invoked by *clients*, which send *requests* to the primary. Then a three-phase protocol begins: (i) in the *pre-prepare* phase, the primary assigns a sequence number to the request and multicasts it to the backups; (ii) in the *prepare* phase, the backups gather a *Byzantine quorum* of $2F_B + 1$ prepare messages in order agree on the sequence number proposed by the primary; (iii) in the *commit* phase, the replicas confirm that an agreement on the request and its sequence number has been reached by a *Byzantine quorum* of replicas. Then, each replica executes the operation and replies to the client. The client waits for a *reply quorum* of $F_B + 1$ replies from different replicas with the same result.

The client. In our setting, the PBFT client is a single virtual entity, i.e., the participants network, and the state machine has a single operation, i.e., append a new block. The participants expect a new valid block to be mined every *target block time*, which in our examples is set to one minute, starting from the *genesis block timestamp*. In light of these considerations, all replicas know in advance all valid request timestamps, that are obtained as *genesis block timestamp* plus multiples of the *target block time*. Requests having such valid timestamps are self-generated locally by each replica. Moreover, given that valid blocks are broadcast to the participants

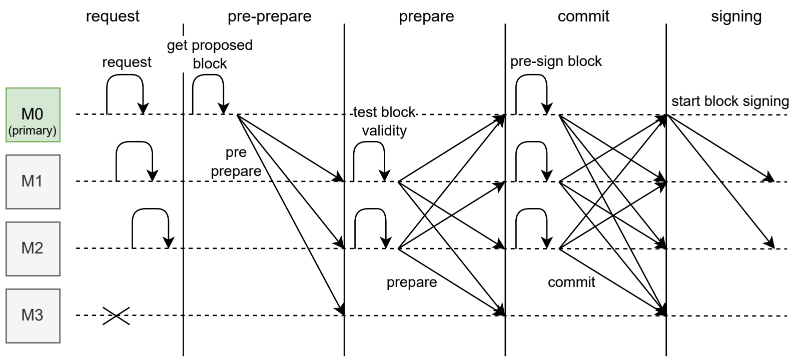


Figure 2. Normal operation (no faulty primary) with 4 nodes, M_0 is primary, M_3 is faulty.

network once a reply quorum of signatures by replicas is achieved, we omit the reply messages.

Normal operations (no faulty primary). The PBFT normal case protocol starts with a request to append a new block. The operation is non-deterministic, as its result depends on the actual content of the block to append (e.g., the set of transactions). We let the primary select and backups verify such content independently (Castro *et al.*, 1999). In the pre-prepare phase, when a block is expected at height n, the primary M_0 gathers a set of transactions from its mempool and forms a proposed block to be appended at height n, which corresponds to the sequence number of the operation. Then, the primary includes the block in the pre-prepare message and broadcasts it to backups. A backup (i.e., M_1, M_2, or M_3 in the figure) accepts a pre-prepare message *if and only if* it is valid according to the PBFT rules, its request has been already generated locally by the replica, its timestamp is not in the future according to the local clock of the replica, and the proposed block (checked by the participant node co-located with the replica) is also valid. If a backup accepts the pre-prepare message, then it enters the prepare phase and broadcasts the prepare message to all other replicas. A replica (primary or backup) accepts a prepare message *if and only if* all the PBFT conditions are met; no additional checks are present at this stage. When replicas reach an agreement on a block and its height, they proceed to the commit phase. In the commit phase, a replica begins the signing process of the prepared block, by including a so-called *FROST commitments* in the commit message, and broadcasting the commit to other replicas. A replica accepts a commit message *if and only if* the PBFT conditions are met. A *Byzantine quorum* of commit messages contains a valid *reply quorum* of FROST commitments, which allow the primary to start the FROST signing sessions (described in Sect. 4.2).

Checkpoints. The checkpoint mechanism is used in PBFT to discard old messages and to advance in the processing of requests. In our protocol, we rely on the Bitcoin block propagation mechanism of the underlying participants network for propagating checkpoints among replicas: Each block appended to the blockchain represents a PBFT checkpoint and causes the replica to move on to mining the next block.

4.2 FROSTing PBFT

Defining upfront the set of participants S that will collaborate to compute the aggregated signature z is cumbersome in presence of Byzantine nodes, which may arbitrarily refuse to sign blocks. We need rules for exchanging commitments (D_i, E_i) and identifying the set S of signers, two critical information for reconstructing the secret used to sign messages.

Therefore, we design *5-Phase Frosted-BFT* (FBFT, for short), which introduces two main changes to the PBFT protocol. First, it blends the commitment protocol and the signature

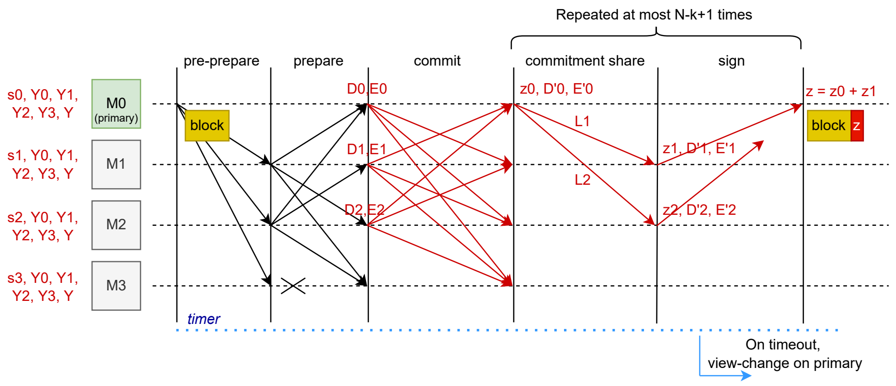


Figure 3. Normal operation of FBFT. Replicas exchange their first commitments in the commit phase. In the commitment share phase, the primary defines two sets of signers, $S_1 = \{0, 1\}$ and $S_2 = \{0, 2\}$, and related L_1 and L_2 parameters. In the sign phase, involved signers send the signature share to the primary who can determine the aggregated signature.

protocol into the normal case PBFT. Second, it extends PBFT with additional rounds to guarantee liveness in case Byzantine nodes play the role of signers. We assume that each replica is a participant in the signing process, which collaborate to apply the threshold signature on the block agreed upon consensus. As per Fig. 3, the new rounds introduced in FBFT are called commitment-share and sign. When a message is prepared by replica i, it runs the *commitment protocol* to randomly determine the nonce/commitment share pairs $((d_i, e_i), (D_i, E_i))$. Public commitments (D_i, E_i) are piggybacked to the commit message and exchanged with other replicas leveraging the PBFT protocol. The primary holds a list of responsive signers, among which the set of candidate signers will be defined. Replicas that send the public commitments in their commit message are considered as part of the initial set of responsive signers and will be candidates for the signing session in the commitment-share phase.

After replicas exchange the commit messages, a set of *commitment-share* phases of FBFT takes place. The primary defines a set of signers S among active replicas; the selection policies for defining the set S follow the rules of ROAST, a wrapper of the FROST protocol that guarantees liveness (Ruffing *et al.*, 2022). We choose S with cardinality $k = F_B + 1$ (i.e., a *reply quorum* of signatures), including the primary itself: In such a configuration, at least one honest signer is in, thus preventing forgery of aggregate signatures over blocks that are invalid or not agreed upon. Even though the primary might exclude nodes suspected to be unresponsive, malicious nodes may be unknown and could be included in S. For this reason, the primary can initiate multiple and concurrent commitment-share sessions, maintaining a set of responsive signers. As soon as there are at least k responsive signers in the set, the primary will initiate a new commitment-share session. When the primary determines S, he creates and sends the list of signers' public commitment $L = \langle (l, D_l, E_l) \rangle_{l \in S}$ to other replicas. Knowing L (and, consequently, S), other replicas $j \in S$ can compute the signature share z_j on the block.

The *sign* phase of FBFT allows replicas to run the aggregate signature protocol presented in Sect. 3. They create and exchange with the primary the signature shares z_i, with $i \in \{1, \ldots, k\}$, together with a new public commitment to be possibly used in another commitment-share session. If any signature share z_i is not valid, the primary marks the replica as malicious, so that it will not be included in subsequent commitment-share phases. When the primary receives all other signature shares z_i, with $i \in \{1, \ldots, k\}$, it can derive the aggregate signature $\sigma = (R, z)$, with $z = \sum_i z_i$ and R the group commitment. If σ is a valid Schnorr signature, the primary appends the signature to the previously proposed block, broadcasts it, and completes FBFT. As demonstrated in ROAST (Ruffing *et al.*, 2022), a non-faulty primary will receive all the signatures in at most $N - k + 1$ commitment-share sessions, under the hypothesis that the number of possible backup failures F_B is at most $N - k$. The PBFT view-change protocol described in (Castro *et al.*, 1999) allows to provide liveness also in presence of a faulty primary, which delays (but does not compromise) the ROAST protocol. When a view change is triggered by the block timeout, the (possibly new) primary replica will act as a new semi-trusted coordinator, that will run again the aggregate signature protocol. Note that the view-change cannot change values the quorum has agreed upon, so the block content cannot be updated.

We now informally argue that the FBFT algorithm satisfies the safety and liveness properties. The safety property relies on the usual cryptographic assumptions and a threshold adversary model with threshold $N > 3f$, while the liveness additionally relies on the partial synchrony of the network (as in PBFT).

Theorem 1 (Safety). *If a set of replicas produced a valid signature for block b_1 with sequence number n in view v, then no valid signature will be produced for block b_2 with n and v by another set of replicas.*

Proof. By contradiction, assume that for view v and sequence number n, there are two blocks b_1 and b_2, with $b_1 \neq b_2$, for which $f + 1$ signature shares have been collected. Consider non-faulty replicas r_1 and r_2 that signed for b_1 and b_2, respectively. If $r_1 = r_2$, we already get a contradiction: a correct replica signed two different blocks for n and v. If $r_1 \neq r_2$, by the safety property of PBFT, there cannot be two correct replicas that commit to two different blocks for the same sequence number. Similarly, the claim also holds for $v' > v$, since PBFT guarantees that if at least a correct replica locally committed the block b in v, which is the precondition to sign b, then no other request will be considered for the same sequence number n in later views $v' > v$. □

Theorem 2 (Liveness). *All valid block proposal are eventually committed and signed by all correct replicas.*

Proof. The claim follows from the proof of termination of ROAST (Theorem 4.3 of Ruffing *et al.*, 2022), and by the liveness property of PBFT, by taking care of triggering view-changes if a replica detects a Byzantine aggregator of signature shares. □

The requirements of Correctness and Calmness are satisfied by construction, as a pre-prepare message is accepted only if its block is valid according to the protocol rules (Correctness), its request has been generated locally by the replica, and its timestamp is not in the future (Calmness). Finally, the Confidentiality requirement is guaranteed by the confidentiality of the signature created using FROST.

4.3 Amending the Bitcoin protocol

This section describes the main changes we introduce to the Bitcoin codebase.

Block validity. Our blocks are valid *iff* they include the solution to a specific *block challenge*, as in the Bitcoin Signet (Alm *et al.*, 2019), that can be expressed either as a script or, since the introduction of Taproot (Wuille *et al.*, 2020), as a public key used to validate a Schnorr signature. For each block, the *block solution* to the challenge is stored in a special `OP_RETURN` output of the coinbase transaction, so it is automatically propagated to the participant network using the standard mechanisms for blocks and transactions. In our case, the solution is an aggregated Schnorr signature, representing a valid but opaque quorum of trusted miners who agreed to append a given block at a specific height. Since different quorums of signers may produce different valid signatures, in order to accommodate for our safety requirement (R2) in the context of a certified Byzantine consensus, it is necessary to exclude the block solution from the computation of the coinbase transaction hash[9]. In addition, it is necessary to include the PoW fields `nBits` and `nNonce` in the block signature, in order to prevent a (malicious) miner to cause a fork by tweaking them: If the PoW fields were not signed, then a miner could change `nNonce` to imply more work, and its block would replace the legitimate one by the Bitcoin rules.

Block mining. The steps for creating blocks become as follows: (1) Upon request by the consensus node of a miner, the corresponding bridging node assembles a block template, i.e., it selects a set of transactions from the mempool, and adds a coinbase transaction with an empty block solution; *the block Merkle root is now finalized*. The miner *grinds* the block, i.e., it finds a nonce that fulfills a trivial PoW-like challenge, which is purposely included for backward compatibility with the original Bitcoin protocol; *the block hash is now finalized*. (2) A quorum of miners signs the block and appends a valid block solution; *the transactions are now finalized*: the Merkle root and block hashes are unaffected.

Block interval. The interval between blocks is fixed to one minute, instead of the 10-minute interval of the public Bitcoin network. We could further increase the rate or the block size to improve the throughput, but this would limit the valuable ability of all network participants to stay in sync, especially those with low bandwidth. At any rate, transaction scalability is meant to be achieved off-chain.

Block subsidy. It plays an incentive role in the public Bitcoin, which is *non-existent* in our setting. We remove the block subsidy checks from the code base of participant nodes.

Coinbase maturity. In Bitcoin, coinbase transaction outputs can only be spent after a certain number of new blocks. In our settings, no forks occur as per our safety requirement, therefore the coinbase maturity is safely set to 0.

5 Evaluation

We evaluate the performance of FBFT in a geographically distributed environment, involving up to 22 mining nodes, placed in 8 different European regions of Amazon Web Services (AWS).[10] According to data collected by CloudPing in 2022[11], the median latency between these regions ranges between 20 ms and 50 ms, while the intra-region median latency stays below 4 ms.

The main measure of performance we consider is *consensus latency* (or just latency, for short), which represents the time needed by the mining network to reach an agreement and sign a new block. Measured at the primary node, it is the difference between the time at which a new signed block is submitted to the participant network (end of the consensus) and the time at which a new block is proposed to the mining network with a *pre-prepare* message (beginning of the consensus algorithm).

In Fig. 4, we compare FBFT with two baseline variants, named PBFT and 3FBFT. The former produces a naïve block solution as the concatenation of signatures by a threshold of the mining nodes. This block solution can be then verified by the participant network using the `OP_CHECKMULTISIG` opcode. The latter is a trivial solution for creating a quorum certificate using FROST: miners run multiple FROST sessions during the consensus, by exchanging, in

9 Different sets of signers for the same block could lead to valid but different block solutions. If the different solutions were included in the computation of the coinbase transaction hash, they would also be included in the Merkle trees, and would result in different block hashes, which would lead to a chain reorganization.

10 Namely: eu-west-1 (Ireland), eu-central-1 (Frankfurt), eu-south-1 (Milan), eu-north-1 (Stockholm), eu-west-2 (London), eu-central-2 (Zurich), eu-south-2 (Spain), eu-west-3 (Paris). We assign a sequential identifier to each node and determine the AWS region where to deploy it using the modulo function.

11 https://www.cloudping.co/grid/p_50/timeframe/1Y

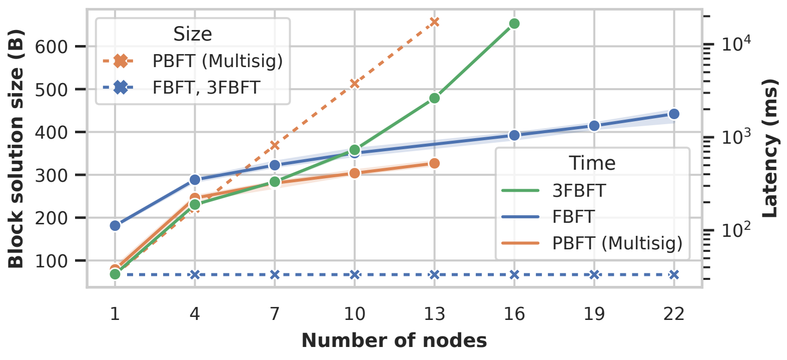
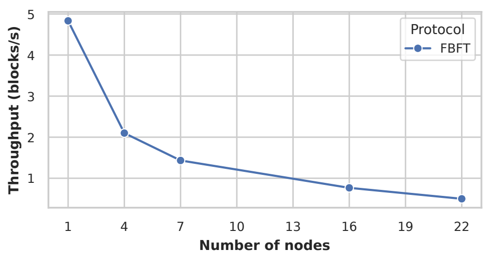


Figure 4. Solution size and average consensus latency achievable with naïve PBFT, FBFT, and 3FBFT.

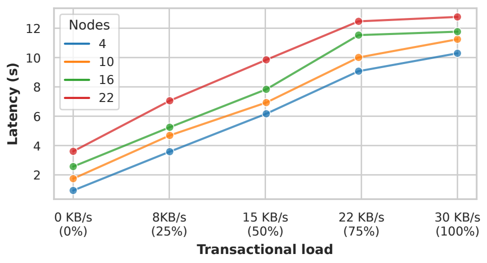
the commit message, a pair of commitments (D_i, E_i) for all the $\binom{N}{k}$ possible combinations of $k = 2F_B + 1$ signers. As a result, a replica that receives a *Byzantine quorum* of commit messages, can immediately execute the aggregation of the signature shares. 3FBFT optimizes communication because it minimizes the number of rounds required to finalize a block. However, since the number of signature shares grows exponentially, this protocol is practical only with small mining networks. For the three variants, Fig. 4 shows the size of the block solution (i.e., the block signature), which represents a witness of the mining network agreement and is broadcast to the participant network together with the block itself. PBFT produces a block solution whose size increases with the mining network size (from 222 bytes with 4 replicas to 657 bytes with 13 replicas). Notably, the `OP_CHECKMULTISIG` opcode, used by participants to verify the block solution, allows checking at most 15 public keys. Both 3FBFT and FBFT represent an improvement over the PBFT baseline because they use FROST for creating a quorum certificate and produce a single Schnorr signature, which can be verified with an ad-hoc Taproot output. Therefore, the block solution size is 67 bytes, no matter the number of miners. Fig. 4 also compares the three algorithms in terms of latency for different sizes of the mining network, here in absence of load. The experiments confirm that FBFT shows higher latency than PBFT, which is motivated by the presence of additional rounds needed for the FROST signature aggregation. Moreover, 3FBFT shows lower latency than FBFT in small mining networks, but its performances degrade very quickly when the number of nodes is greater than 10. This is motivated by the calculation of all possible combinations of a *Byzantine quorum* of signatures out of all the possible signers, leading to a prohibitively high latency of 19.2 s with 16 nodes. With 22 mining nodes, FBFT registers an average latency of 1.7 s in a setting spread across 8 AWS regions. This value of consensus latency is largely below our requirement of a new block every 60 s.

Fig. 5a reports the maximum throughput achievable with FBFT across the AWS European regions. The throughput is the number of blocks that could be produced by the mining network per unit of time. It is slightly lower than the inverse of the consensus latency, because it also takes into account the time for block propagation in the mining network. We configure the experiments so that the network produces blocks as fast as possible (i.e., we “disable” the calmness, forcing miners to recover a blockchain with a genesis block time in the past). As expected, the throughput decreases as the mining network size increases. This is mainly due to the quadratic communication complexity of FBFT, which builds on PBFT.

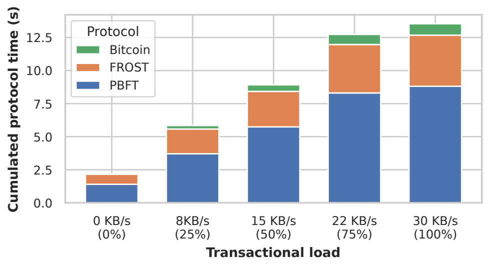
In Fig. 5b we investigate the impact of incoming transactions on consensus latency. We configure the mining network to generate a new block every minute in the steady state, during which we evaluate the performances. However, since we set the genesis block timestamp 30 minutes in the past, there is a warm-up period in which the miners will try to mine the first 30 blocks at the highest rate achievable with the given network conditions. In order to generate



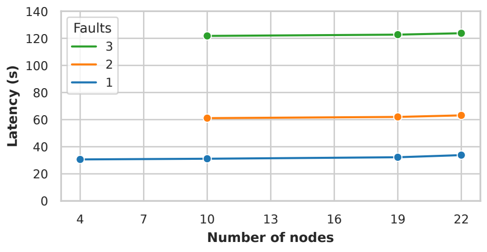
(a) Average throughput without load



(b) Average FBFT latency under different load conditions



(c) Avg time spent by FBFT in its protocols, 16 nodes



(d) Avg latency of the block following primary failures

Figure 5. Evaluation of FBFT: throughput, time spent in its protocols, and consensus latency.

load for the mining network, we set up additional 8 participant nodes that submit transactions to the mining network. The warm-up period described above allows the clients to fill their wallets with a number of coins sufficient to generate a transaction load at the desired rate. The client rate is set to target a given block size that we describe as 0%, 25%, 50%, 75%, 100% of maximum block size (1.8 MB). As Fig. 5b shows, when the load increases, the consensus latency increases as well. Indeed, the increase of the block size slows down the exchange of the *pre-prepare* message by the primary, and the block validity check by backups. We experience that the impact of transaction load is linear for all the mining network configurations. Nevertheless, even with blocks at full load and 22 mining nodes, the maximum latency we experienced is around 13 s, below our requirement to mine a block every 60 s.

To agree and sign the next block, FBFT uses different protocols: PBFT for consensus, FROST for signature aggregation, and Bitcoin for block validation. Fig. 5c details the time spent in these different stages, under different load conditions, with 16 mining nodes. When the load is 0 KB/s, FBFT closes empty blocks in 2170 ms: it spends 64% of time to complete the PBFT phases; 35.1% of time to exchange and aggregate the signature shares, whereas the Bitcoin block validity and submission checks take less than 1% of the time. If the load is at its maximum, then PBFT takes 65% of time, the signature aggregation takes 29%, and the block validity check takes 6% of time.

Finally, we evaluate the block latency in presence of failures. When the primary appears as faulty, FBFT uses the view change protocol to elect a new primary and recovers from failure. Fig. 5d shows the impact of failures in the worst-case scenario, where the primaries of subsequent views fail consecutively, and multiple view changes are triggered before finding the agreement on the next block. After 60 s, we forcefully terminate the primary of the mining network in the initial view, and possibly up to two other primaries expected for the next views. We set the initial view change timeout to 30 s for an expected block time of 60 s. Almost for every mining network size, the consensus latency is strongly delayed by the view-change protocol, which doubles subsequent timeouts with the number of subsequent views. It increases from less than 2 s to $\approx$ 30 s, with 1 failure, to $\approx$ 60 s, with 2 concurrent failures, to $\approx$ 120 s, with 3 concurrent failures (123.7 s with 22 nodes). When the view change is completed, the consensus protocol recovers the delayed blocks at the maximum throughput, and continues to mine with calmness at a consensus latency that is less than 2 s.

Overall, it appears that FBFT can provide Byzantine fault tolerance, network confidentiality, and efficient usage of block solution space, for just a reasonable increment in consensus latency.

6 Related work

There exist previous examples of Bitcoin-derived ledgers meant for private networks. In particular, Elements[12], whose production deployment (the "Liquid" sidechain—Nick *et al.*, 2020) uses a consensus algorithm within a permissioned mining network consisting of cryptocurrency businesses. Elements is the closest work to ours in terms of technologies and Bitcoin reuse goals. The implementation of its functionaries (the actors responsible of signing blocks in Liquid) has been open sourced[13] in August 2023. The production-ready software interfaces with Hardware Security Modules (HSM) for block signing and verification. However, Elements uses ECDSA in place of Schnorr, and it does not rely on a threshold signature scheme that creates a confidential quorum certificate.

The second largest DLT born public and then adapted to permissioned settings is Ethereum (Wood *et al.*, 2014). An example of an Ethereum-like ledger designed for private networks is Hyperledger Besu[14], which supports a PoA consensus based on Istanbul BFT (Moniz, 2020). Another example is Concord[15]. This is possibly the closest work to ours, in spirit, but (i) it implements SBFT (Golan Gueta *et al.*, 2019) instead of PBFT as consensus algorithm; (ii) it works with BLS signatures instead of Schnorr signatures; and (iii) it has Ethereum instead of Bitcoin as a foundation. Point (iii) is a profound differentiator: Ethereum exhibits a Turing-complete language as a key feature, and focuses on the development of complex decentralized applications via smart contracts, while for our objective we elected a blockchain technology mainly focused on digital payments that could easily scale off-ledger with a payment channel network. While in principle BLS signatures could also serve our purpose of aggregating multiple signature into one, their use is ruled out by the choice of the blockchain technology: namely Bitcoin only supports the *secp256k1* elliptic curve, which is not suited for BLS signatures (being not pairing-friendly).

There is a plethora of other relevant permissioned DLTs, whose main difference with respect to our approach is the absence by design of any attempt to profit from existing code bases from major public blockchains. E.g.: Hyperledger Fabric[16] is a general-purpose DLT that enables the development of enterprise applications, not necessarily financial. Among its components, there is a BFT consensus module, but its development appears to have ceased[17]. Corda[18] is a DLT designed for the financial industry; it has a token-based data model, like Bitcoin, and a Turing-complete programming language, like Ethereum. Its *notaries* can run either a crash fault-tolerant (CFT) consensus or a BFT consensus. Neither the BFT specification nor its implementation is available in the open-source repository and, apparently, no aggregated signature scheme is included. Hyperledger Sawtooth[19] allows deploying private DLT networks with a variety of consensus algorithms, including PBFT. Sawtooth has an open source implementation[20], with

12 https://blockstream.com/elements
13 https://github.com/Blockstream/liquid-functionary
14 https://www.hyperledger.org/use/besu
15 https://blogs.vmware.com/opensource/2018/08/28/meet-project-concord
16 https://www.hyperledger.org/use/fabric
17 https://github.com/bft-smart/fabric-orderingservice
18 https://www.corda.net and https://github.com/corda
19 https://www.hyperledger.org/use/sawtooth
20 https://github.com/hyperledger/sawtooth-pbft

an incomplete PBFT implementation and no Schnorr signature aggregation. Diem[21] (formerly Libra) implements a Turing-complete programming language designed for safe and verifiable transaction-oriented computation. It employs a custom BFT algorithm called DiemBFT (Baudet *et al.*, 2019), based on HotStuff (Yin *et al.*, 2019). Hamilton (Lovejoy *et al.*, 2022) is a DLT designed to support payments in a permissioned network (a use case similar to ours). It inherits certain elements from Bitcoin (e.g., UTXO and cryptographic primitives). Relevant differences: its ledger is not a blockchain and is meant to stay private; its consensus protocol is not BFT but CFT; its main focus is on obtaining transactional scalability on-ledger. Table 1 summarizes the main differences between the major permissioned ledgers.

We analyzed several BFT consensus algorithms (e.g., Baudet *et al.*, 2019; Buchnik *et al.*, 2020; Distler, 2021; Golan Gueta *et al.*, 2019; Yin *et al.*, 2019), and we decided to design our block creation process around PBFT (Castro *et al.*, 1999). PBFT sacrifices linear communication (the number of exchanged messages is quadratic in the cluster size) in return for a simpler implementation; however, our requirements call for the mining network to produce no more than a block per minute, and our cluster is (by design) small enough to make the superlinear communication complexity a minor drawback, whereas simplicity in the implementation helps a lot the cohabitation with the Bitcoin platform. We implemented the BFT consensus algorithm from scratch instead of relying on already existing blockchain implementations, such as Tendermint (Buchman, 2016), in order to maximize the reuse of and compatibility with the Bitcoin protocol, including its consensus engine. The interested reader can found an extensive analysis of blockchain consensus protocols in Xu *et al.*, 2023.

Finally, note how most BFT protocols use threshold BLS signatures (e.g., Golan Gueta *et al.*, 2019; Yin *et al.*, 2019), which rely on pairing-based cryptography. However, this may be challenging to implement in practice in our platform, since BLS signatures are not supported in Bitcoin. Conversely, Schnorr signatures received increased interest recently, and they have already been included in the Bitcoin protocol. In 2021, Komlo and Goldberg (Komlo *et al.*, 2021) proposed FROST (see Sect. 3), and it is currently considered the most efficient scheme for generating threshold Schnorr signatures. Recently, Ruffing *et al.*, 2022 proposed ROAST, a wrapper protocol around FROST that provides liveness guarantees in presence of malicious nodes and asynchronous networks. Differently from ROAST, our FBFT protocol guarantees liveness in a peer to peer network that aggregates signature shares even in the case of Byzantine coordinator.

7 Future directions

As future work, we plan to evolve our architecture towards different research directions, which we summarize as follows.

Improving the fairness of FBFT. The term *fairness* in the context of blockchain refers to the property that transactions submitted by participants are selected by miners for inclusion in a block, according to a well-defined criterion, that is transparent to the participants' network. This allows participants to form realistic expectations about the time that will be needed for the confirmation of a transaction. The most common criterion in public blockchains provides for selecting the transactions according to the paid transaction fees. In the current version of FBFT, fairness is not achieved, because a single Byzantine miner could hijack the block creation process and, among other things, delaying/censoring some transactions in favour of others, which would represent unfair behaviour. Fairness can be achieved *iff* the primary can generate

21 https://developers.diem.com/docs/welcome-to-diem

Table 1. Comparison of major permissioned distributed ledgers

	Byzantine Fault Tolerance	**Confidential Quorum Certificate**	**Distributed Ledger Technology**	**Schnorr Quorum Certificate**	**Open Source**
Elements	Strong Federations (Dilley *et al.*, 2016)	No	Bitcoin	No	Yes
Hyperledger Besu	IstanbulBFT (Moniz, 2020)	No	Ethereum	No	Yes
Concord	SBFT (Golan Gueta *et al.*, 2019)	Yes, BLS (Boneh *et al.*, 2004)	Ethereum	No	Yes
Hyperledger Fabric	No, Raft (Ongaro *et al.*, 2014)	No	Fabric	No	Yes
Corda	Unclear	Unclear	Corda	Unclear	Partially
Hyperledger Sawtooth	PBFT (Castro *et al.*, 1999)	No	Sawtooth	No	Yes
Diem	DiemBFT (Baudet *et al.*, 2019)	No	Diem	No	Yes
Project Hamilton	No, Raft (Ongaro *et al.*, 2014)	No	Hamilton	No	Yes
Our solution	FBFT	Yes	Bitcoin	Yes, FROST (Komlo *et al.*, 2021)	Yes

at most a single block, and the primary rotates at each block, e.g., using an election technique based on cryptographic sortition—such as the Verifiable Random Functions used in Algorand (Gilad *et al.*, 2017). An always-changing primary would prevent a Byzantine primary from censoring transactions. We plan to explore how to introduce fairness in the FBFT protocol while keeping the same deterministic finality guarantees that are currently provided.

Support for a dynamic federation of validators. The permissioned DLT platform described in this work has a static block challenge, that is the public key representing the group of validators (while the corresponding solution is a Schnorr threshold signature produced by a subset of the validators). Nevertheless, the capability of changing the secret key shares controlled by the validators is a must-have for any real-world deployment of the platform. Indeed, a variety of use cases require such a feature: validators may need to periodically rotate their secret key shares as a security best practice; the mining network may be reconfigured to include a new validator or to exclude a validator controlling a compromised key; finally, some federations may want to enforce policies in which one or more members of the federation rotate periodically. In theory, there are two ways to support dynamic key shares: updating the block challenge or finding new key shares which can be used to produce a threshold signature for the same group public key. Changing the block challenge is not trivial in practice. It requires participants to upgrade their node software to a new version that includes the new challenge; and participants that will not carry out such update will not be able to stay in sync with the chain. For this reason, this kind of updates must be planned well in advance, or require that the update procedures are very streamlined. The second option relies on the possibility to change the number of signers and/or the threshold without updating the group public key. More in general, this requires to explore dynamic secret sharing in FROST, and their implications on safety. This is unexplored territory for aggregated signature schemes, which we are currently investigating.

Analysis of a layer-2 payment channel network. As described in the vision paper (Urbinati *et al.*, 2021) and highlighted in Sect. 4.3, the throughput of the blockchain layer is limited by design to a few dozen transactions per second, and transaction scalability is meant to be achieved off-chain, using layer-2 protocols available for Bitcoin, such as a payment channel network (Poon *et al.*, 2016). Payment channel networks (PCNs) promise to overcome the scalability issues of blockchains by enabling instant, safe, and privacy-preserving transactions. These transactions are not settled one by one on the blockchain, but are routed by a network of intermediary nodes, which are connected by payment channels. Although the literature analyzes different aspects of PCNs, thus far it is not clear whether they can successfully handle real-world payment volumes, and if they can even go beyond that, e.g., in supporting additional use cases such as micro-payments. We will work towards a systematic investigation of some specific PCN topologies, which fit our permissioned scenarios. We aim to analyze and optimize the trade-offs between the liquidity locked in channels (representing a cost) and rates of successfully routed payments. Payment privacy guarantees offered by specific PCN topologies also deserve further investigation. A realistic payment load can be simulated using a statistical model calibrated on real-world payment data, available in aggregated form from official studies of major financial institutions (e.g., ECB Surveys, 2022), or derived from public datasets (e.g., Venmo, 2019).

Exploring asset tokenization for DvP and cross-border payments. The on-chain layer of our platform can be used by the participants to exchange large value transactions, which can be made dependent on programmable conditions, expressed using the scripting language of Bitcoin. For this reason, our permissioned blockchain is very similar to an advanced wholesale payment system. Moreover, the off-chain PCN extension or other layer-2 protocol, which can be built by participants on top of the ledger infrastructure, can be used to exchange a large volume of low-value payments, with privacy-preserving guarantees. For this reason, it may properly serve the needs of a retail payment system. There is a third and very relevant feature of traditional market infrastructures, that is the capability to use the ledger to represent assets other than the native currency, in order to facilitate cross-border payments and Delivery-versus-Payment (DvP), settlement methods that guarantee the atomicity of a transaction involving either multiple payments or both a payment and an asset transfer (e.g., see La Rocca *et al.*, 2022). In the last year, the industry and the open source communities have been developing open protocols, such as RGB[22] and Taproot Assets (formerly Taro)[23], that can be used for the tokenization of assets on top of Bitcoin-like blockchains. All these recent developments can be swiftly inherited by our permissioned network. Moreover, these protocols also have support for layer-2 smart contracts involving both the native ledger currency and the asset.

8 Conclusion

We presented and evaluated a Bitcoin-like, permissioned, distributed ledger in which valid blocks are signed by a federation of independent actors and transactions enjoy deterministic finality. Block signatures are aggregated via a threshold scheme based on FROST, that preserves the confidentiality of the mining network configuration and its quorum. We showed how such a federation could operate correctly also under Byzantine failures of a subset of the nodes.

Our design embodies one way of inheriting all the algorithms, data structures, and software of Bitcoin—but its PoW-based consensus protocol—in order to make its full technological stack openly available to permissioned settings managed by trusted actors.

What for? Bitcoin has inspired innumerable other blockchains and is perhaps the closest thing we have to an open standard for payments in the "crypto domain". It is possible that its technological stack—constantly scrutinized, improved, evolved—will one day percolate into the blockchain-friendly portion of the financial ecosystem for old and new use cases. This perspective has famed historical precedents. It is not unlike reusing the exact same technological

22 RGB, Private & scalable smart contracts for Bitcoin and Lightning Network: https://rgb-org.github.io/
23 A Taproot Asset Representation Overlay: https://docs.lightning.engineering/the-lightning-network/taproot-assets/

stack from a decentralized, public network (the Internet) into private, “permissioned” networks (intranets): TCP/IP. At the dawn of the networking era, the idea of a convergent public/private stack was unheard of, and a host of custom, proprietary networking suites were deployed for “permissioned” use cases. But eventually, the good-enough and widely adopted TCP/IP won over most specialized (and mutually incompatible) protocols. It became a *de facto* standard, the one we now take for granted and use every day, at work and at home (permissioned settings).

We are a very long way from a similar turn of events in the realm of digital payment systems. And, it may well be argued that a shared technological ground is unlikely to ever materialize. Still, we proved the idea makes technical sense, and we would better be ready. Solutions such as the one we present here pave the way to be prepared for such a possibility.

References

Alm, K.-J., & Towns, A. (2019). Signet. *BIP*, *325*.

Amsden, Z., Arora, R., Bano, S., Baudet, M., Blackshear, S., Bothra, A., Cabrera, G., Catalini, C., Chalkias, K., Cheng, E., Ching, A., Chursin, A., Danezis, G., Giacomo, G. D., Dill, D. L., Ding, H., Doudchenko, N., Gao, V., Gao, Z., … Zhou, R. (2020). *The libra blockchain* (tech. rep.). Novi. https://diem-developers-components.netlify.app/papers/the-diem-blockchain/2020-05-26.pdf

Banque de France. (2021). *Wholesale central bank digital currency experiments with the banque de france (2021)* (tech. rep.). https://diem-developers-components.netlify.app/papers/the-diem-blockchain/2020-05-26.pdf

Baudet, M., Ching, A., Chursin, A., Danezis, G., Garillot, F., Li, Z., Malkhi, D., Naor, O., Perelman, D., & Sonnino, A. (2019). *State machine replication in the Libra blockchain* (tech. rep.). The Libra Association.

Benedetti, M., De Sclavis, F., Favorito, M., Galano, G., Giammusso, S., Muci, A., & Nardelli, M. (2022). A PoW-less Bitcoin with Certified Byzantine Consensus. *arXiv:2207.0687*.

Boneh, D., & Komlo, C. (2022). Threshold signatures with private accountability. *Cryptology ePrint Archive*.

Boneh, D., Lynn, B., & Shacham, H. (2004). Short Signatures from the Weil Pairing. *J. Cryptol.*, *17*(4), 297–319.

Buchman, E. (2016). *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains* (Doctoral dissertation). University of Guelph.

Buchnik, Y., & Friedman, R. (2020). Fireledger: A high throughput blockchain consensus protocol. *Proc. VLDB Endow.*, *13*(9), 1525–1539.

Cai, L., Sun, Y., Zheng, Z., Xiao, J., & Qiu, W. (2021). Blockchain in china. *Communications of the ACM*, *64*(11), 88–93.

Castro, M., & Liskov, B. (1999). Practical Byzantine Fault Tolerance. *Proc. of OSDI'99*, 173–186.

Castro, M., & Liskov, B. (2002). Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.*, *20*(4), 398–461.

Chaum, D., Grothoff, C., & Moser, T. (2021). How to issue a central bank digital currency. *arXiv:2103.00254*.

Dilley, J., Poelstra, A., Wilkins, J., Piekarska, M., Gorlick, B., & Friedenbach, M. (2016). Strong federations: An interoperable blockchain solution to centralized third-party risks. *arXiv:1612.05491*.

Distler, T. (2021). Byzantine fault-tolerant state-machine replication from a systems perspective. *ACM Comput. Surv.*, *54*(1).

Dwork, C., Lynch, N., & Stockmeyer, L. (1988). Consensus in the Presence of Partial Synchrony. *Journal of the ACM*, *35*(2), 288–323. https://doi.org/10.1145/42282.42283

ECB Surveys. (2022). Study on the payment attitudes of consumers in the euro area (SPACE). https://www.ecb.europa.eu/stats/ecb_surveys/space/html/ecb.spacereport202212~783ffdf46e.en.html

European Central Bank. (2020). Report on a digital euro. *ECB publications*.

European Central Bank. (2023). Official digital euro project page [Online; accessed Sept 15th, 2023]. https://www.ecb.europa.eu/paym/digital_euro/html/index.en.html

Federal Reserve. (2022). Money and payments: The US Dollar in the age of digital transformation. *Federal Reserve publications*.

Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, *32*(2), 374–382.

Garay, J. A., Kiayias, A., & Leonardos, N. (2015). The Bitcoin backbone protocol: Analysis and applications. *EUROCRYPT 2015*, 281–310.

Garay, J. A., Kiayias, A., & Leonardos, N. (2020). Full analysis of Nakamoto consensus in bounded-delay networks. *IACR Cryptol. ePrint Arch.*, (277).

Gilad, Y., Hemo, R., Micali, S., Vlachos, G., & Zeldovich, N. (2017). Algorand: Scaling Byzantine agreements for cryptocurrencies. *Proc. of SOSP '17*, 51–68.

Golan Gueta, G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M., Seredinschi, D.-A., Tamir, O., & Tomescu, A. (2019). SBFT: A Scalable and Decentralized Trust Infrastructure. *Proc. of IEEE/IFIP DSN'19*, 568–580.

Komlo, C., & Goldberg, I. (2021). FROST: Flexible Round-optimized Schnorr Threshold Signatures. *SAC 2020*, *12804*, 34–65.

La Rocca, R., Mancini, R., Benedetti, M., Caruso, M., Cossu, S., Galano, G., Mancini, S., Marcelli, G., Martella, P., Nardelli, M., & Oliviero, C. (2022). Integrating DLTs with market infrastructures: analysis and proof-of-concept for secure DvP between TIPS and DLT platforms. *Markets, Infrastructures, Payment Systems. Bank of Italy*, (26).

Lee, S., & Kim, S. (2020). Short selling attack: A self-destructive but profitable 51% attack on pos blockchains [https://ia.cr/2020/019].

Lovejoy, J., Fields, C., Virza, M., Frederick, T., Urness, D., Karwaski, K., Brownworth, A., & Narula, N. (2022). A high performance payment processing system designed for central bank digital currencies. *Cryptology ePrint Archive*, (163).

Moniz, H. (2020). The Istanbul BFT Consensus Algorithm. *arXiv:2002.03613*.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*.

Nick, J., Poelstra, A., & Sanders, G. (2020). *Liquid: A Bitcoin sidechain* (tech. rep.). Liquid.

Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. *Proc. of USENIX ATC'14*, 305–320.

Poon, J., & Dryja, T. (2016). The Bitcoin lightning network: Scalable off-chain instant payments.

Ruffing, T., Ronge, V., Jin, E., Schneider-Bensch, J., & Schröder, D. (2022). ROAST: Robust asynchronous Schnorr threshold signatures. *Cryptology ePrint Archive*, (550).

Saad, M., Spaulding, J., Njilla, L., Kamhoua, C., Shetty, S., Nyang, D., & Mohaisen, D. (2020). Exploring the attack surface of blockchain: A comprehensive survey. *IEEE Commun. Surv. Tutor.*, *22*(3), 1977–2008.

Urbinati, E., Belsito, A., Cani, D., Caporrini, A., Capotosto, M., Folino, S., Galano, G., Goretti, G., Marcelli, G., Tiberi, P., & Vita, A. (2021). A digital euro: A contribution to the discussion on technical design choices. *Markets, Infrastructures, Payment Systems. Bank of Italy*, (10).

Venmo. (2019). Venmo Transaction Dataset [[Online; accessed July 2023]].

Wood, G., et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, *151*, 1–32.

Wuille, P., Nick, J., & Towns, A. (2020). Taproot: SegWit version 1 spending rules. *BIP*, *341*.

Xu, J., Wang, C., & Jia, X. (2023). A survey of blockchain consensus protocols. *ACM Comput. Surv.*, *55*(13s). https://doi.org/10.1145/3579845

Ye, C., Li, G., Cai, H., Gu, Y., & Fukuda, A. (2018). Analysis of security in blockchain: Case study in 51%-attack detecting. *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*, 15–24. https://doi.org/10.1109/DSA.2018.00015

Yin, M., Malkhi, D., Reiter, M. K., Gueta, G. G., & Abraham, I. (2019). HotStuff: BFT Consensus with Linearity and Responsiveness. *Proc. ACM PODC '19*, 347–356.