DracoBFT: Enabling Validators as Trustless Service Providers in Public Blockchain Consensus

Vyom Sharma
Hotstuff Labs
vyom@hotstufflabs.com

Hiten Jain
Hotstuff Labs
hiten@hotstufflabs.com

Abstract

Public blockchain validators have traditionally been limited to passive consensus participation—validating transactions and producing blocks in exchange for protocol-issued rewards. In contrast, permissioned systems allow validators to provide direct services to users, but sacrifice trustlessness and decentralization. This paper presents DracoBFT, which bridges this divide through auxiliary verification loops that enable public blockchain validators to function as active service providers while maintaining cryptographic trustlessness.

DracoBFT introduces a validator-as-service-provider architecture where validators offer verified access to external services—payment processors requiring sub-300ms authorization responses, credit bureau APIs, fiat conversion services, cross-chain bridges—with trustless verification through zkTLS proofs (for off-chain data) and zero-knowledge execution proofs (for on-chain state). This fundamentally reimagines validator roles in public chains: validators compete on service quality, earn fees directly from users, and are routed based on performance metrics, creating active incentives for user experience improvement previously impossible in trustless settings.

The protocol implements Fast-HotStuff's two-chain finality with three key innovations: (1) auxiliary verification loops enabling trustless external service provision, (2) chained change-log hash commitments providing $O(m)$ state commitment complexity where $m \ll n$ for sparse financial workloads, and (3) weighted stake-based leader selection ensuring fairness. We provide formal verification with 244 comprehensive tests across 9 categories under the partially synchronous model with $n = 3f + 1$ validators. The architecture is designed for deployment in Hotstuff L1, a high-performance financial infrastructure platform where validators serve as competitive gateways to real-world financial services.

1 Introduction

1.1 Motivation

1.1.1 The Validator Limitation Problem

Public blockchain validators face a fundamental constraint: they can only validate what occurs on-chain. When applications require external data (asset prices, identity verification, credit scores) or integration with off-chain services (payment processors, banking APIs, fiat rails), public chains traditionally rely on trusted intermediaries—oracle networks, bridge operators, multisig committees—that reintroduce centralization and trust assumptions the blockchain was designed to eliminate.

A critical manifestation of this limitation is the user onboarding challenge for DeFi platforms. While USD-denominated stablecoins have become standard on-chain, converting local fiat currencies to crypto and reverse remains the highest friction touchpoint in both technical and user experience dimensions. This friction prompts users to gravitate toward centralized exchanges for onboarding, defeating the purpose of decentralized finance. Peer-to-peer solutions and decentralized marketplaces have emerged but fail

at scale, remaining rife with fraud and scams. Purely technical solutions for trustless P2P fiat-to-crypto exchange remain unsolved: traditional finance provides chargeback protection for disputed transactions, but blockchain transactions are irreversible, creating asymmetric risk that scammers exploit. Without trusted intermediaries to arbitrate disputes or verify service delivery, P2P crypto-fiat exchange cannot achieve the reliability and safety users expect.

Permissioned blockchain systems solve this by allowing validators to directly provide services to users. In permissioned settings, validators are known entities that can operate payment gateways, access banking APIs, and provide financial services. However, this approach sacrifices the key advantages of public blockchains: trustlessness, censorship resistance, and permissionless participation.

This creates a stark tradeoff: public chains are trustless but limited to on-chain data, while permissioned chains can access external services but require trusting specific validators. No existing architecture enables validators in public, trustless settings to provide verified access to external services without reintroducing trusted intermediaries.

1.1.2 The Case for Reimagining Validator Roles

Byzantine Fault Tolerant (BFT) consensus protocols have undergone significant evolution since PBFT [2], culminating in the HotStuff family [1] with linear communication complexity and Fast-HotStuff's 2-chain finality rule. Yet validator functionality has remained fundamentally unchanged: validators passively process transactions, earn protocol-issued rewards, and have no direct relationship with end-users.

This passive model creates several inefficiencies for financial infrastructure:

(i) **Latency Requirements:** Financial exchanges require 100-500ms finality for order matching and settlement. Payment authorization workflows require webhook responses within 300ms [25] to approve card transactions. These tight latency bounds demand efficient consensus and external data verification.

(ii) **External Service Access:** Financial applications require verified access to services outside the blockchain: payment processors for fiat conversion, credit bureaus for risk assessment, banking APIs for account verification, foreign exchange platforms for currency conversion. Traditional solutions rely on trusted oracles or centralized bridges.

(iii) **Validator Economic Sustainability:** Pure block reward models create passive validator income with no incentive for service quality or user onboarding. As networks mature and issuance decreases, validators need sustainable revenue beyond protocol inflation.

(iv) **Service Quality Differentiation:** In traditional models, all validators earn identical rewards regardless of service quality, uptime, or user experience. There is no mechanism to reward high-performing validators or penalize poor performers without governance intervention.

(v) **State Commitment Efficiency:** Computing Merkle Patricia Tries after each block incurs $O(n \log n)$ costs for large state, dominating consensus latency for financial workloads with sparse state updates.

1.1.3 DracoBFT's Approach

DracoBFT addresses these challenges by fundamentally reimagining validator roles in public blockchains. Through auxiliary verification loops with cryptographic proof verification, DracoBFT enables validators to provide direct services (payment authorization, credit assessment, banking integration, cross-chain bridges) while maintaining trustlessness. Validators compete on service quality, earn fees directly from users, and are routed based on performance—dynamics previously possible only in permissioned systems, now achieved in a public, trustless architecture.

The protocol combines Fast-HotStuff consensus, chained state commitments with $O(m)$ complexity, weighted leader selection, and the auxiliary verification architecture that enables this paradigm shift. The implementation is designed for deployment in Hotstuff L1, a financial infrastructure platform where validators function as competitive service gateways while maintaining Byzantine fault tolerance.

1.2 Validators as Service Providers

Traditional blockchain consensus protocols rely on validators solely for achieving agreement on transaction ordering through redundant execution across multiple nodes. When sufficiently decentralized, these systems provide strong guarantees against censorship and tampering through Byzantine fault tolerance. However, this redundancy-based approach faces inherent tradeoffs: consensus overhead limits latency and throughput, network synchronization constrains performance, and transparency requirements (all validators must verify) preclude data privacy.

DracoBFT extends beyond this traditional model by architecting validators as *service providers* rather than pure consensus participants. Through auxiliary verification loops, validators can offer verified access to external services—payment processors, banking APIs, trading venues, identity providers—without compromising the security guarantees of decentralized consensus. The key insight is separating *service provision* (which may be permissioned and private) from *service verification* (which remains trustless and cryptographic).

This architectural separation enables a new operational model:

- **Consensus Layer:** Maintains decentralized agreement on global state through traditional BFT mechanisms (redundant execution, quorum voting, Byzantine tolerance).
- **Service Layer:** Individual validators provide permissioned access to external services, verified through cryptographic proofs (zkTLS for off-chain data, ZK execution proofs for on-chain state) rather than redundant execution.
- **Routing Layer:** Matches users to appropriate validators based on service availability, geographic coverage, performance history, and quality-of-service metrics.

This approach preserves the security guarantees of decentralized consensus (no validator can unilaterally manipulate state or censor transactions) while enabling validators to competitively provide auxiliary services verified through cryptographic proofs. The consensus protocol maintains global state consistency and finality, while auxiliary loops verify external service delivery without requiring all validators to redundantly execute the same external calls.

For financial infrastructure, this model enables validators to function as last-mile gateways providing verified access to fiat on/off-ramps, payment rails, foreign exchange services, and banking integrations—expanding validator utility beyond block production while maintaining trustless verification through zero-knowledge proofs.

1.3 Contributions

This paper makes the following contributions:

1. **Trustless Auxiliary Verification Architecture:** We present a modular architecture for trustless verification of external state through zkTLS proofs (for off-chain API data) and zero-knowledge execution proofs (for on-chain state from other blockchains). This enables trustless bridges, verifiable oracle feeds, and cross-chain operations without relying on trusted intermediaries, processed via auxiliary loops that operate in parallel with consensus while maintaining atomic state transitions on finalization.
2. **Chained Change-Log Hash Commitments:** We introduce a lightweight state commitment mechanism with $O(m)$ complexity that replaces $O(n \log n)$ global state

root computation, where m is the number of *unique keys* modified per block (typically $m \ll n$ for state size n). Block-level aggregation ensures keys modified multiple times within a block are hashed only once, dramatically reducing I/O for high-frequency workloads. The chained CLH links finalized state across blocks, enabling early fork detection and historical state verification.

3. **Two-Chain Fast-HotStuff Implementation:** We present DracoBFT, implementing the Fast-HotStuff two-chain finality rule with vote rule block.view == block.qc.view + 1, achieving finalization when a block has a certified direct child.

4. **Weighted Stake-Based Leader Selection:** We implement deterministic weighted random leader selection with epoch-based schedule computation, ensuring fairness proportional to validator stake.

5. **Formal Verification:** We provide a complete formal specification with 244 comprehensive tests across 9 categories verifying safety and liveness properties.

6. **Implementation:** We implement subsecond finality with deterministic execution, where application execution rather than consensus overhead dominates end-to-end latency.

1.4 Paper Organization

The remainder of this paper is organized as follows. Section 2 provides background on BFT consensus and the HotStuff protocol family. Section 3 presents the DracoBFT protocol design. Section 4 discusses key innovations including CLH, global event management, and weighted leader selection. Section 5 provides formal safety and liveness analysis. Section 6 presents performance evaluation. Section 7 describes the implementation architecture. Section 8 covers formal verification. Section 9 analyzes security. Section 10 discusses related work. Section 11 outlines future work, and Section 12 concludes.

2 Background and Preliminaries

2.1 System Model

We consider a distributed system with $n = 3f+1$ nodes, where f represents the maximum number of Byzantine faulty nodes. The system operates under the partially synchronous network model [8], where:

- **Partial Synchrony:** After some unknown Global Stabilization Time (GST), message delivery is bounded by a known constant Δ.
- **Authenticated Channels:** All messages are authenticated using digital signatures.
- **Cryptographic Assumptions:** We assume a computationally bounded adversary and the existence of collision-resistant hash functions and unforgeable digital signatures.

2.2 Byzantine Fault Tolerance

A BFT consensus protocol must satisfy two fundamental properties:

Safety All honest nodes agree on the same sequence of committed values (consistency):

- No two honest nodes commit different values for the same sequence number.
- Once a value is committed, it cannot be reverted.

Liveness The system continues to make progress (availability):

- Clients eventually receive responses to their requests.
- New values continue to be committed under partial synchrony.

2.3 The HotStuff Protocol

HotStuff introduced several key innovations [1]:

(1) **Three-Phase Commit:** Prepare, Pre-commit, and Commit phases.

(2) **Linear View-Change:** $O(n)$ message complexity for view changes.

(3) **Optimistic Responsiveness:** Commits at network speed when the leader is honest.

(4) **Chained Structure:** Phases are pipelined across consecutive views.

The standard HotStuff commit rule requires three matching phases:

$$\text{prepare}(B) \rightarrow \text{pre-commit}(B) \rightarrow \text{commit}(B) \rightarrow \text{decide}(B).$$

2.4 Quorum Certificates

A Quorum Certificate (QC) is a cryptographic proof of agreement from a Byzantine quorum of $2f + 1$ nodes:

$$\text{QC} = \langle v, h(B), \{\sigma_i\}_{i \in Q} \rangle,$$

where v is the view, $h(B)$ is the block hash, σ_i is the signature from node i, and Q is a quorum of nodes with combined stake exceeding 2/3 of total stake.

3 DracoBFT Protocol Design

3.1 Protocol Overview

DracoBFT operates as a leader-based, view-driven protocol with the following key state:

- *view*: current view number (monotonically increasing),
- *highQC*: highest known quorum certificate,
- *lastVote*: last view in which the node voted,
- *ActivePacemaker*: Keeps track of votes and timeouts,
- *forest*: collection of pending blocks.

3.2 Two-Chain Finality Mechanism

DracoBFT implements the Fast-HotStuff two-chain finality rule, where a block is finalized when it has a certified direct child block.

Algorithm 1 Leader Proposal in View v

```
1: procedure PROPOSE(v)
2:     highQC ← highest known QC
3:     parent ← GETBLOCK(highQC.blockHash)
4:     txns ← transaction batch from mempool
5:     B_v ← new block with B_v.parent = highQC.blockHash,
6:         B_v.qc = highQC, B_v.view = v, B_v.txns = txns
7:     broadcast ⟨PROPOSE, B_v⟩
```

Algorithm 2 Fast-HotStuff Vote Rule

```
1: procedure VOTERULE(proposal)
2:     if proposal has AggregateQC then
3:         return parent block exists
4:     else
5:         return (proposal.block.view ≥ currentView)
6:             and (proposal.block.view == proposal.block.qc.view + 1)
```

This vote rule enforces the Fast-HotStuff safety property: validators only vote for blocks that directly extend (consecutive views) a certified block.

A block B at view v is finalized when a child block B' at view $v + 1$ contains a QC for B:

Algorithm 3 Two-Chain Finalization Check

```
1: procedure CHECKFINALIZATION(block)
2:     parent ← GETBLOCK(block.qc.blockHash)
3:     if parent.view + 1 == block.view then
4:         finalize parent and all ancestors to last finalized
```

This achieves **2-view finality**: a block at view v is finalized once a block at view $v+1$ is certified. This requires two message rounds—one round to form QC for block B, and one round to propose child block B' containing that QC.

Happy Path: Under normal conditions with an honest leader and network synchrony, blocks are proposed in consecutive views (v, $v+1$, $v+2$, ...). Each block is finalized exactly 2 views after proposal: block at view v finalizes when the block at view $v+2$ is proposed (containing QC for block at $v+1$, which contains QC for block at v).

The Active Pacemaker (Section 3.4) guarantees this property by ensuring that a supermajority of nodes remains within one view of each other at all times, preventing view divergence that could delay finality.

3.3 Pipelining Strategy

DracoBFT employs pipelining to maintain network saturation: while votes are being collected for block B_v, the leader for view $v+1$ can already propose B_{v+1}, maintaining a pipeline depth of 2–3 blocks.

3.4 Active Pacemaker

The Active Pacemaker introduces an active recovery mechanism for views without successful proposals. When a node times out, it broadcasts a timeout message containing its current view and last observed successful proposal. Individual nodes only leave a failed view after observing timeout messages from a supermajority ($2f+1$) of nodes.

This explicit timeout step ensures nodes skip failed views based on collective agreement rather than isolated local timeouts, guaranteeing that a supermajority remains within one view of each other at all times. This eliminates view divergence and enables 2-view finality.

3.5 View-Change Protocol

The view-change protocol maintains linear complexity $O(n)$ under the happy path and quadratic complexity $O(n^2)$ under network delays or Byzantine behavior:

Algorithm 4 View Change from v to $v+1$

procedure AdvanceView($data$)
  $v \leftarrow 0$, $timeout \leftarrow false$
  if $data.highTC \neq \perp$ **then**
    $v \leftarrow \max(v, data.highTC.view)$, $timeout \leftarrow true$
  if $data.aggQC \neq \perp$ **then**
    $v \leftarrow \max(v, data.aggQC.view)$, $timeout \leftarrow true$
  if $data.highQC \neq \perp$ **then**
    $v \leftarrow \max(v, data.highQC.view)$, $timeout \leftarrow false$
  $view \leftarrow v+1$
  send $\langle \text{NEWVIEW}, v+1, data \rangle$ to leader$(v+1)$

3.6 Leader Selection

DracoBFT uses weighted stake-based leader selection with deterministic schedule computation. The validator set can be updated dynamically through special transactions, enabling membership changes without protocol downtime.

4 Key Innovations

Section 3 describes the core protocol. We now highlight the key design innovations that differentiate DracoBFT from prior HotStuff-style protocols.

4.1 Auxiliary Verification Loops and Validator-as-Service-Provider

DracoBFT introduces *Side-Loops*—main-driven auxiliary consensus mechanisms that perform private or expensive computation off the main critical path while producing on-chain effects anchored to finalized main views via compact attestations.

This design reduces the trust problem of off-chain computation to two concrete components: (1) the main-chain finality guarantee, and (2) the cryptographic/economic threshold securing the side-loop. Unlike generic off-chain committees that can submit arbitrary transactions referencing stale or ambiguous main states, Side-Loops

eliminate such ambiguity—every accepted Side-Loop settlement is unambiguously tied to a finalized main view and a threshold attestation.

Consequently, the main engine neither blocks for off-chain work nor must it perform expensive re-execution to validate side-loop outcomes, enabling auxiliary operations to proceed in parallel with consensus while maintaining atomic state transitions on finalization.

4.1.1 System Model

DracoBFT operates as a Layer 1 blockchain where most transactions originate from end-user signed messages. However, a significant class of operations requires trustless verification of external state or actions that cannot be directly signed by users. The auxiliary verification loops provide a mechanism to verify and incorporate these operations.

4.1.2 Side Loop Architecture: Autonomous Computational Domains

These side loops represent specialized computational domains that extend the system's capabilities without compromising the main consensus path.

Non-blocking Specialization Certain workloads fundamentally require isolation from the main consensus critical path. Private RFQ (Request-For-Quote) matching, multi-party computation protocols, heavy zero-knowledge proof generation, and external API orchestration involve computational complexity, latency characteristics, or privacy requirements incompatible with low-latency consensus finalization. Side loops execute these tasks asynchronously off the main event loop, preventing any blocking or interference with consensus progression.

The architectural separation ensures that main loop consensus maintains consistent sub-second finality regardless of auxiliary workload characteristics. A side loop performing heavy ZK proof generation (seconds to minutes) or awaiting external API responses (variable latency) operates independently, submitting verified results to the main loop only upon completion. This isolation prevents resource contention and maintains predictable main loop performance.

Deterministic Anchoring Side loops synchronize with the main consensus loop through finalized view anchoring. The main loop (Fast-HotStuff with Jolteon pacemaker) provides strong finality guarantees: finalized views are immutable and will never reorg. Side loops observe finalized main views as logical clock ticks, using these anchors to:

- Establish causal ordering for side loop operations relative to main chain state
- Publish commitment hashes or settlement proofs at deterministic main loop views
- Reference specific finalized state snapshots for settlement or verification
- Coordinate multi-party computation rounds to main chain progression

Because the main loop guarantees finality without reorganization, anchoring is both simpler (no fork handling in side loops) and stronger (anchored commitments are immutably ordered). Side loops leverage this finality property to provide deterministic settlement without maintaining complex fork-choice rules or reorganization logic.

Privacy & Confidentiality Domain Privacy-sensitive operations require isolated computational domains where plaintext or decrypted data is accessible only to authorized participants. Dark pool matching, RFQ negotiation, and multi-party computation protocols involve confidential information that must not be exposed on the public main consensus loop.

Side loops serve as privacy domains where:

- Plaintext order details are visible only to side loop participants (e.g., matching engine validators)

- MPC protocols share secret-split data exclusively among computation participants
- TEE (Trusted Execution Environment) attestations prove correct execution without revealing inputs
- The main loop observes only cryptographic commitments (order hashes, sealed bids, MPC output commitments)

Settlement occurs by publishing authenticated settlement objects to the main loop: matched order pairs, MPC computation results, or proof-of-execution certificates. The main loop verifies authentication (signatures, threshold signatures, zkTLS proofs) without accessing confidential data, maintaining privacy separation while ensuring trustless settlement.

Operational Specialization Different application domains require distinct operational parameters, security models, and economic structures. Side loops enable this specialization by supporting:

- **Distinct Validator Sets:** Side loops can operate with subsets of main validators or specialized participants. A private matching engine might involve only validators staked for matching services, while MPC computation might require TEE-capable hardware.
- **Alternative Stake Economics:** Side loop participation can require specialized staking (e.g., separate matching engine stake) with distinct slashing conditions tailored to application-specific misbehavior (e.g., front-running penalties in matching, equivocation in MPC).
- **Consensus Flavor Flexibility:** Side loops can employ consensus mechanisms tailored to their requirements: leader-based consensus for low-latency matching, leaderless protocols for censorship resistance, or threshold-signature-only schemes for simple commitment publication. This flexibility enables optimization for specific workload characteristics without constraining main loop design.

The combination of these properties enables DracoBFT to support complex financial workflows—private order matching, multi-venue liquidity provision, cross-chain settlement with ZK bridges, and real-time payment authorization—without compromising the main consensus loop's performance, finality guarantees, or security properties. Side loops extend system capabilities modularly while maintaining tight cryptographic coupling to the finalized main chain state.

4.1.3 Trustless Verification Architecture

The system supports two classes of trustless verification:

Offchain Action Proofs For public or private API calls to external services, DracoBFT leverages zero-knowledge Transport Layer Security (zkTLS) proofs [21, 34, 35, 36].[1] These proofs provide cryptographic verification of HTTPS responses without requiring the external service to be aware of the blockchain.

The zkTLS approach works by having a prover intercept a TLS session, generate a zero-knowledge proof of the encrypted traffic and server's signature, and provide this proof to validators. Validators can verify that:

- The response came from the claimed server (verified via TLS certificate chain)

[1] zkTLS encompasses three implementation approaches [34, 36]: (1) TEE-based (using trusted execution environments for tamper-proof TLS sessions), (2) MPC-based (using multi-party computation to distribute the TLS shared key across multiple nodes, e.g., TLSNotary's 2PC approach), and (3) Proxy-based (using HTTPS proxies with zero-knowledge proofs of decryption, formalized in [36] and implemented by Reclaim Protocol). All approaches provide cryptographic integrity for TLS responses without requiring the external service's cooperation. DracoBFT's auxiliary verification architecture is compatible with all three approaches, with implementation choice driven by security-efficiency tradeoffs.

- The response content matches specific claims (verified via ZK proof)
- The data was not tampered with (verified via server signature)

This enables trustless integration of traditional APIs including asset price feeds from exchanges, identity verification services, real-world event data, and any HTTPS endpoint. The key advantage is preserving privacy (API provider is unaware of blockchain usage) while eliminating trust assumptions beyond standard TLS security.

Onchain Action Proofs For verification of state or events on other blockchains (e.g., Ethereum, other EVM chains), two complementary approaches are available:

(1) **zkTLS-based RPC proofs:** Proof of blockchain state via RPC calls to node providers, where the RPC response is verified using zkTLS. This provides efficient verification (<100ms proof generation, <10ms verification) at the cost of trusting the RPC provider's view of chain state. Suitable for applications where RPC provider trust is acceptable (e.g., using reputable providers like Infura, Alchemy) or where economic incentives align (e.g., slashing for false data).

(2) **Zero-knowledge execution proofs:** Succinct proofs of EVM execution [22] that recursively verify block headers (consensus), Merkle proofs (state inclusion), and contract execution (EVM computation) without trusting any intermediary. These proofs provide full trustlessness by re-executing and proving correctness of the entire verification chain from block headers through state transitions. The computational cost is higher (seconds for proof generation) but enables trustless bridges eliminating all third-party dependencies.

For bridge applications, DracoBFT can combine both approaches: zkTLS for monitoring deposit events quickly, with ZK execution proofs generated asynchronously for final settlement, balancing latency and trustlessness.

4.1.4 Validator-as-Service-Provider Model

Beyond traditional block validation, the auxiliary verification architecture fundamentally expands validator utility by enabling them to function as permissioned access points to external services. In this model, validators opt-in to provide verified access to third-party services (payment processors, banking APIs, identity providers, trading venues) with trustless verification through zkTLS proofs.

The system implements a routing mechanism that matches end-users to specific validators based on:

- Validator stake (economic security collateral)
- Historical performance and uptime metrics
- Geographic proximity and regional service coverage
- Quality-of-service parameters (latency, availability)
- Supported service integrations (specific payment rails, fiat currencies, banking partners)

This architectural approach transforms validators from pure consensus participants into competitive service gateways. Validators earn fees for verified service provision (fiat on/off-ramps, payment processing, foreign exchange access, regional banking integrations) while the consensus protocol maintains global state consistency. The auxiliary loops verify service delivery and API responses through cryptographic proofs, eliminating trust requirements while creating economic incentives for high-quality service provision.

For financial applications, this enables validators to function as last-mile connectivity providers delivering access to local payment rails, currency conversion services, banking integrations, credit assessment services, and real-time

payment authorization (meeting sub-300ms latency requirements for card and merchant transactions) based on regional requirements. The routing layer matches users to validators offering appropriate services while zkTLS proofs ensure service authenticity without requiring trust in the validator beyond stake-based security guarantees.

Regulatory Compliance and Jurisdictional Flexibility Financial service provision in cryptocurrency faces complex regulatory challenges that vary by jurisdiction and service type. Different regulatory frameworks govern crypto-fiat conversion: the EU's Markets in Crypto-Assets (MiCA) regulation establishes comprehensive rules for crypto service providers, Singapore's Monetary Authority (MAS) requires licensing for digital payment token services, and various national regimes impose know-your-customer (KYC) and anti-money-laundering (AML) requirements. These regulations typically apply based on the service provider's jurisdiction, the user's location, and the specific service offered.

The validator-as-service architecture accommodates regulatory diversity through geographic validator distribution and selective service provision. Validators opt-in to specific services based on their regulatory compliance capabilities: a validator licensed in Singapore can provide fiat conversion services to regional users, while a validator in the EU can serve MiCA-compliant services to European users. The routing layer matches users to validators offering services appropriate for their jurisdiction, while the consensus protocol maintains global state consistency regardless of which validators provide auxiliary services.

This approach provides regulatory flexibility without fragmenting the chain: the consensus layer remains unified and trustless (all validators participate in BFT consensus), while the service layer adapts to local requirements (validators selectively provide regulated services they're licensed for). Cryptographic proof verification ensures service authenticity without requiring the entire validator set to be licensed in every jurisdiction, enabling compliant financial service provision within a public, decentralized architecture.

4.1.5 Integration via Auxiliary Loops

Algorithm 5 Auxiliary Verification Loop

Phase 1 - Proof Collection:
Collect external action proofs (zkTLS or ZK execution)

Phase 2 - Verification:
Verify proofs using appropriate verifier
Generate verification certificate

Phase 3 - Consensus Integration:
Package verified action as transaction
Submit to consensus mempool

Phase 4 - Finalization:
On block finalization, apply state transition atomically

4.1.6 Use Cases and Applications

This architecture enables several key applications:

- **Trustless Bridges:** Verify deposit events on external blockchains without trusted relayers or multisig committees. ZK execution proofs provide cryptographic verification of deposit transactions, enabling secure cross-chain asset transfers.
- **Verifiable Oracle Feeds:** Obtain price data, identity verification, or real-world events from traditional APIs with zkTLS proofs. Validators verify cryptographic proofs rather than trusting oracle providers.
- **Financial Service Access:** Validators provide permissioned access to payment processors, banking APIs, foreign exchange platforms, and card issuance services. zkTLS proofs verify service delivery including

fiat payment completion, payment authorization responses within required latency windows (<300ms for card transactions), account balance verification, and credit score retrieval from credit bureaus—all without exposing sensitive credentials or requiring trust in the validator.

- **Cross-Chain State Queries:** Access state from external blockchains (token balances, contract storage, historical events) with cryptographic verification, enabling DeFi applications that span multiple chains.
- **Authenticated API Integration:** Integrate any HTTPS API (payment processors, identity providers, data sources) with trustless verification, expanding blockchain utility beyond on-chain data.

The auxiliary loops operate asynchronously, batching multiple verified actions into single transactions to amortize verification costs across consensus rounds. Key architectural benefits include: (1) extensibility to new proof systems without consensus modifications, (2) parallel verification while consensus proceeds, (3) atomic state transitions maintaining consistency, (4) trustless verification eliminating dependencies on intermediaries, and (5) economic incentive alignment where validators earn fees for verified service provision while maintaining security through cryptographic proofs.

4.1.7 Planned Deployment: Compliant Onboarding and Multi-Venue Liquidity

Hotstuff L1, the planned deployment of DracoBFT, will demonstrate two key applications of the validator-as-service-provider model and auxiliary verification loops:

Validator-Gateway Onboarding DracoBFT validators integrate with compliant fiat on/off-ramp providers (Bridge.xyz, Avenia.io, Brale.xyz) to enable native user onboarding with full regulatory compliance [31, 32]. Bridge and similar platforms provide KYB (Know Your Business) verification for validators and KYC (Know Your Customer) infrastructure for end users, eliminating the need for validators to obtain separate money service business licenses.

The onboarding flow proceeds as follows: (1) users select a validator as their financial access point based on geographic preference (e.g., Avenia for LATAM users), (2) users complete KYC directly with the provider via the validator's integration, (3) validators generate zkTLS proofs [33] of user verification status and account details, and (4) auxiliary verification loops validate these proofs via consensus quorum, updating the state machine with verified user-validator linkages. Validators stake protocol tokens proportional to onboarded users, with slashing conditions for misbehavior (downtime, fraud, privacy leakage, non-compliance).

This architecture distributes compliance across validators while maintaining protocol-level neutrality, similar to Stripe Connect's model where the platform provider handles regulatory requirements for integrated businesses.

Multi-Venue Market Making Hotstuff L1 operates an automated market-making vault (HLV) providing liquidity across multiple venues by leveraging DKG-based key generation and auxiliary verification loops. Validators collectively generate EdDSA keys (for centralized exchanges) and ECDSA keys (for HyperLiquid) via distributed key generation, enabling trustless order placement after routing through consensus for risk management [37, 38].

Auxiliary loops synchronize fills and positions from external venues (HyperLiquid, Binance, Deribit) in real-time, enabling the vault to maintain delta-neutral positions through consensus-validated rebalancing. Unlike single-venue vaults such as HyperLiquid's HLP [40], HLV leverages hedge mode (simultaneous long/short positions) to improve capital efficiency by 50% through net delta margining.

For CEX integration, HLV adopts Ethena's off-exchange custody model [39], utilizing MPC clearing solutions (Copper ClearLoop, Ceffu) to minimize custodial risk. Vault depositors ac-

cept trade-offs including CEX custody risk, auto-deleveraging risk, and consensus-induced latency (precluding HFT strategies but enabling passive market making). Initial deployment targets HyperLiquid-only integration to minimize trust assumptions, with phased CEX expansion after demonstrating operational stability.

4.2 Validator Economics and Incentive Alignment

4.2.1 Evolution of Validator Incentive Models

Traditional blockchain consensus protocols rely on native cryptocurrency issuance as the sole mechanism for validator compensation. In Bitcoin and Ethereum, validators (miners or stakers) earn block rewards denominated in the native currency (BTC or ETH), creating passive income streams independent of user adoption or service quality. This model successfully bootstraps network security but provides no direct incentive for validators to improve user experience or onboard new users.

Recent protocols have explored alternative economic models addressing user experience and fee predictability. Table 1 compares various approaches to validator compensation and transaction fees.

These approaches improve user experience through fee subsidization or stable pricing, but maintain passive validator compensation models where validators earn from protocol-level issuance or aggregate transaction fees, not from direct service provision to individual users.

4.2.2 Validator-as-Service-Provider Economics

DracoBFT's auxiliary verification architecture introduces a fundamentally different economic model: validators earn directly from end-users for verified service provision. When a validator provides access to external services (payment authorization, credit assessment, fiat conversion, banking integration), the validator earns fees directly from the user consuming that service, in addition to standard consensus rewards.

This direct incentive structure creates several beneficial dynamics:

Service Quality Competition Validators are directly incentivized to provide high-quality service, maintain low latency, and ensure high availability. Poor service quality results in reduced routing (fewer users matched to that validator) and lower fee income. This contrasts with traditional models where validators earn the same rewards regardless of service quality, leading to minimal differentiation.

User Onboarding Incentives Validators have direct economic incentive to onboard new users and integrate with additional external services. Each new user represents potential fee income, motivating validators to expand service coverage, improve regional connectivity, and develop better tooling. Traditional validators have no direct incentive to grow the user base—their income depends on network-wide activity, not individual user acquisition.

Reputation-Based Routing The platform routing layer can dynamically adjust user-to-validator matching based on aggregate user feedback, service delivery metrics, and performance history. Validators maintaining high service quality receive increased routing weight, earning more fees. This creates competitive pressure for continuous service improvement and penalizes poor performers through reduced routing, without requiring explicit slashing or governance intervention.

Economic Sustainability The validator-as-service model provides sustainable economics beyond native token issuance. As the network matures and block rewards potentially decrease (common in cryptocurrency monetary policy), validators maintain income streams through service fees. This reduces reliance on perpetual inflation or high transaction fees for validator compensation, improving long-term economic sustainability.

Protocol	Txn Fees	Validator Income	Model
Ethereum	ETH gas	Block rewards + gas	Passive: market fees
Arc [26]	USDC gas	Block rewards + fees	Stable: \$0.01/tx
Plasma [27]	USDT free	Protocol subsidy	Subsidized: paymaster
Hyperliquid [28]	Core free, EVM gas	Dual-block model	Hybrid: selective fees
Hotstuff L1	**Standard gas**	**Block + service fees**	**Active: direct user services**

Table 1: Validator economic models. DracoBFT enables direct service fees.

The auxiliary verification loops enable this economic model while maintaining security through cryptographic proof verification. Validators cannot cheat service delivery (zkTLS proofs are verifiable), cannot manipulate state unilaterally (consensus requires quorum), and risk stake if they equivocate or violate protocol rules. The economic incentives thus align with protocol security: validators maximize income by providing honest, high-quality service while maintaining proper consensus participation.

4.2.3 Validator Participation Equilibrium

We formalize validator incentives through a participation game to demonstrate Nash equilibrium stability of the validator set.

Game Setup Consider n validators deciding whether to operate nodes. Each validator i chooses strategy $s_i \in \{\text{Participate}, \text{Exit}\}$. A validator's payoff from participation depends on the total number of active validators and comprises three components: block rewards, service fees, and operational costs.

Payoff Structure A participating validator's expected payoff is:

$$\pi_i(n) = \underbrace{\frac{R}{n}}_{\text{block rewards}} + \underbrace{F(n, q_i)}_{\text{service fees}} - \underbrace{C}_{\text{operational cost}} \tag{1}$$

where R is total block reward allocation per epoch, n is the number of active validators, $F(n, q_i)$ represents service fees earned by validator i with service quality q_i, and C is the fixed operational cost including infrastructure, bandwidth, and compliance overhead.

The service fee function captures two key dynamics. First, geographic diversity: as validator count n increases, additional validators provide coverage in new regions, but marginal coverage gains diminish. Second, quality differentiation: validators with higher service quality q_i (lower latency, better uptime, more service integrations) receive higher routing weight and thus more fee income. We model this as:

$$F(n, q_i) = \alpha \cdot \frac{q_i}{\sum_{j=1}^{n} q_j} \cdot g(n) \tag{2}$$

where α is the total service fee pool, $\frac{q_i}{\sum_j q_j}$ is validator i's share based on relative quality (routing weight), and $g(n) = \beta \cdot n^{\gamma}$ with $0 < \gamma < 1$ captures sublinear growth in total fees as validator count increases (diminishing returns to geographic coverage).

Nash Equilibrium The equilibrium validator count n^* satisfies the zero-profit condition for marginal validators:

$$\pi_i(n^*) = 0 \quad \text{and} \quad \frac{\partial \pi_i}{\partial n}(n^*) < 0 \tag{3}$$

The second condition ensures stability: if additional validators enter, expected payoff becomes negative, deterring further entry. Solving for equilibrium with homogeneous quality ($q_i = q$ for all i):

$$\frac{R}{n^*} + \frac{\alpha \cdot \beta}{n^{1-\gamma}} = C \implies n^* = \left(\frac{R + \alpha\beta}{C}\right)^{\frac{1}{1-\gamma}} \tag{4}$$

DracoBFT Dynamics DracoBFT's validator-as-service-provider model differs fundamentally from traditional PoS in two ways:

First, service fees α increase with user onboarding and network adoption, unlike block rewards R which are typically fixed or decreasing over time. As more users join and demand external service integrations (payment processing, identity verification, cross-chain bridges), total service fee pool α grows. This creates positive correlation between network success and validator profitability, sustaining validator participation even as block rewards decline.

Second, quality differentiation q_i matters for validator income. In traditional PoS, all validators earn proportional to stake regardless of service quality, creating minimal incentive for infrastructure investment beyond minimum requirements. In DracoBFT, validators with better service quality (lower latency API responses, more geographic integrations, higher uptime) receive increased routing weight and thus higher fee income. This creates competitive pressure for continuous improvement without requiring protocol-level quality-of-service enforcement.

Comparative Analysis Table 2 compares equilibrium properties across different validator compensation models.

This equilibrium analysis demonstrates that DracoBFT's economic model is not merely a pragmatic choice but emerges from game-theoretic principles: rational validators participate when service fees exceed costs, and quality competition emerges naturally from routing-based fee allocation. The model is incentive-compatible (validators maximize income through honest service provision) and sustainable (revenue grows with network adoption rather than depending on perpetual inflation).

Model	Income	Equil.
Bitcoin PoW	Block rewards	Low
Ethereum PoS	Block + gas	Medium
DracoBFT	**Service**	**High**

Table 2: Validator equilibrium stability. DracoBFT's service fee model provides stable incentives.

4.3 Change-Log Hash (CLH) Commitments

4.3.1 The State Root Performance Problem

Traditional blockchain systems compute global state roots after each block to enable light client verification and state membership proofs. Ethereum's Merkle Patricia Trie (MPT) and similar structures (Sparse Merkle Trees, Verkle Tries) provide these guarantees but incur severe performance costs.

The bottleneck is disk I/O, not computation. Updating a state root in an MPT requires reading and writing $O(\log_k n)$ nodes for each modified key-value pair, where k is the tree arity and n is the number of leaves. For a binary MPT with billions of keys, this translates to dozens of random disk reads per state update. Recent analysis [29] shows that state root computation can cause up to 10x slowdown in block building: for high-throughput EVM execution, updating the state root dominates latency, consuming more time than transaction execution itself.

Key sparsity exacerbates this problem. In practice, MPTs and similar structures experience space inflation (hundreds of times larger

than optimal) due to sparse key distribution, forcing them to overflow RAM to slow disk storage. State-of-the-art optimizations include NOMT (grouping subtree nodes into disk pages) and MegaETH's SALT [30] (optimal space utilization by taming sparsity), but these still require complex data structures and significant I/O for high-throughput workloads.

4.3.2 DracoBFT's Approach

DracoBFT replaces state root computation with Change-Log Hashes computed during block execution. Rather than committing to the entire state, CLH commits only to state modifications, eliminating the need for tree traversal and disk I/O for commitment computation.

Conceptual Position DracoBFT's CLH approach is conceptually closer to Bitcoin's simplicity than to Ethereum's global state root or Tendermint's AppHash. Bitcoin commits nothing about state (only transaction hashes), Ethereum and Tendermint commit to the entire global state (expensive), while DracoBFT commits only to state modifications (lightweight). This design point trades off some light client capabilities for substantial performance gains by minimizing disk I/O—a tradeoff appropriate for high-frequency financial infrastructure where validators maintain authoritative state.

Table 3 compares these approaches across key dimensions.

The spectrum of design choices is: Bitcoin (no state commitment) → DracoBFT (diff commitment) → Tendermint (app state commitment) → Ethereum (global state root). DracoBFT achieves Bitcoin-like simplicity and speed while providing stronger integrity guarantees than pure transaction ordering, without incurring the disk I/O costs of full state commitments.

When a block B is finalized, the trading engine processes transactions and records state modifications. The change-log hash is computed incrementally as a running hash over all state operations.

Block-Level State Aggregation A critical optimization: DracoBFT aggregates state changes at the block level rather than per-transaction. If a block contains 400 transactions that collectively modify the same state variable n times, only the *final* value is hashed once in the change log, not all n intermediate values. This dramatically reduces I/O operations for high-frequency workloads where the same keys are repeatedly modified (e.g., order book updates, account balances in trading).

For example, if transactions in a block update account balance: \$100 → \$150 → \$120 → \$200, the CLH includes only one entry: `Put(account_balance, $200)`. The intermediate values (\$150, \$120) are never written to the change log. For a block with 1,000 transactions touching 100 unique keys, the change log contains at most 100 entries, not 1,000.

Algorithm 6 Change-Log Hash Computation with Block-Level Aggregation

```
aggregatedChanges ← ∅                    ▷ Map: key → final value
for all transaction txn in block do
    execute txn
    for all state operation op in txn do
        record op in aggregatedChanges    ▷ Overwrites previous value

hasher ← SHA256()                        ▷ Hash only final changes
for all (key, value) in aggregatedChanges (sorted by key) do
    if value is Delete then
        hasher.WRITE('D'‖key)
    else
        hasher.WRITE('P'‖key‖value)
return hasher.SUM()
```

DracoBFT maintains a *chained* change-log hash linking finalized blocks:

$$\text{chainedCLH}_B = H(\text{chainedCLH}_{prev} \| \text{finalizedCLH}_B)$$

where chainedCLH_{prev} is the chained CLH from the previous finalized block. This provides fork

Aspect	Bitcoin	Ethereum	Tendermint	DracoBFT CLH
State Model	UTXO	Account (global)	Account (IAVL)	Key-value
Block Commitment	Tx Merkle only	State root	AppHash	Change-log hash
What Committed?	Transactions	Entire state	Entire state	Modified keys only
Commitment Cost	Very low	Very high	Medium-high	Very low
Write Cost/Update	Low	High (trie rehash)	Medium (IAVL)	Low (append+hash)
Light Client Support	Weak	Strong	Strong	Moderate
State Membership Proofs	No	Yes (Merkle)	Yes (IAVL)	No (snapshot-based)
Sync Method	Replay chain	Trie snapshot	IAVL snapshot	Snapshot + diffs
High State-Churn Perf	Excellent	Poor	Degrades	Excellent
Pipelining w/ Consensus	Good	Difficult	Hard	Excellent
Disk I/O per Block	Minimal	Heavy (tree)	Medium (tree)	Minimal (log)
Implementation Complexity	Simplest	Most complex	Moderate	Very simple
Security Surface	Minimal	Large (trie)	Medium	Minimal

Table 3: Comparison of state commitment approaches. DracoBFT's CLH sits between Bitcoin's no-commitment simplicity and Ethereum/Tendermint's full-state commitment complexity, optimizing for high-throughput financial workloads by committing only to state changes.

detection by linking state history across finalization boundaries.

Key properties include: (1) $O(m)$ complexity where m is the number of *unique keys* modified per block (not total operations), with $m \ll n$ for sparse updates—no tree traversal, no disk I/O for commitment, (2) block-level aggregation eliminating redundant hashing when keys are modified multiple times within a block, (3) deterministic computation ensuring all nodes produce identical CLH, (4) consistency checking through proposals including finalized block CLH, (5) chained hash linking providing historical state verification across finalization boundaries, and (6) delayed reference to finalized state enabling early detection of state divergence.

4.3.3 Tradeoffs and Limitations

CLH provides substantial performance advantages but trades off certain capabilities of traditional authenticated data structures:

No Membership Proofs Unlike MPTs or Sparse Merkle Trees, CLH does not support efficient state membership proofs. Given a CLH, one cannot prove "account X has balance B" without replaying the change log from genesis or a trusted snapshot. This is acceptable for full validators (who maintain complete state) but requires additional mechanisms for light clients.

Light Client Protocol Light clients must rely on validator-signed state snapshots combined with CLH verification:

(1) Obtain trusted snapshot at view v with CLH_v signed by quorum

(2) Download subsequent blocks $B_{v+1}, \ldots, B_{current}$

(3) Replay change logs and verify locally computed CLH matches block CLH

(4) Query full validators for specific state values with trust based on quorum signatures

This approach provides weaker guarantees than MPT-based light clients (which can verify individual state queries cryptographically) but remains practical for financial infrastructure where full validators maintain authoritative state and light clients primarily verify chain progression and finalization.

State Sync Requirements New validators joining the network must obtain a recent state snapshot from existing validators, then apply subsequent change logs. While CLH verification

ensures snapshot consistency with the canonical chain, the initial snapshot must be trusted based on validator quorum signatures. This is a standard weak subjectivity assumption in PoS systems.

Design Tradeoff DracoBFT optimizes for validator-to-validator state verification (high performance, early fork detection) rather than client-to-validator state queries (which require trust or full state). For financial infrastructure where validators maintain complete state and users primarily submit transactions rather than query arbitrary state, this tradeoff is appropriate.

Operation	MPT	CLH
State update	$O(n \log n)$	$O(m)$
State proof	$O(\log n)$	$O(m)$

Table 4: Complexity comparison where m = unique keys modified per block, n = total state size. Block-level aggregation ensures m counts each key once regardless of modification frequency.

4.3.4 Concrete Performance Comparison

To illustrate the practical impact of block-level aggregation, consider a common high-frequency trading scenario: 400 transactions in a block all modifying the same state variable (e.g., an order book price level or frequently-traded account balance). Table 5 compares the number of hash operations required by different state commitment mechanisms.

For order book updates where a single price level may be modified by hundreds of trades within a block, this optimization eliminates 99.75% of hashing operations compared to naive per-operation approaches. For broader workloads with multiple hot keys (e.g., 10 frequently-traded accounts in a 1,000-transaction block), aggregation reduces hashing from 1,000 operations to 10, a 100× improvement.

System	Granularity	Hash Ops
Ethereum MPT	Per-tree-node (∼10/update)	∼4,000
Tendermint IAVL	Per-tree-node (∼5/update)	∼2,000
Bitcoin UTXO	Per-output	400
Naive CLH	Per-operation	400
DracoBFT CLH	**Per-unique-key**	**1**

Table 5: Hash operations for 400 transactions modifying a single state key. Ethereum MPT and Tendermint IAVL rehash tree paths per update. Bitcoin UTXO and naive CLH hash once per transaction. DracoBFT's block-level aggregation hashes the key once, providing 400× reduction for hot-key trading workloads.

4.4 Weighted Stake-Based Leader Selection

To ensure fairness proportional to validator stake, DracoBFT implements weighted random leader selection with epoch-based schedule computation.

Algorithm 7 Weighted Leader Selection

1: **procedure** ComputeLeaderSchedule($epochStart$)
2: $totalStake \leftarrow \sum_{v \in validators} stake(v)$
3: $rng \leftarrow$ NewRNG($seed = epochStart$)
4: **for** $i = 0$ to EPOCH_SIZE **do**
5: $randWeight \leftarrow rng$.Next() mod $totalStake$
6: select validator where cumulative stake $> randWeight$
7: $leaderSchedule[epochStart + i] \leftarrow$ selected validator

This provides proportional probability (validator with stake s has probability $s/totalStake$), determinism (all validators compute identical schedule), efficiency ($O(n)$ per epoch), and dynamic updates (schedule recomputed on validator set changes).

4.5 Deterministic Transaction Ordering

To ensure state consistency across replicas, DracoBFT enforces deterministic ordering of transactions within blocks. Transactions are grouped by action class and ordered lexicographically by priority tuple (class, timestamp, hash, nonce), where classes encode coarse global ordering and timestamps/hashes provide deterministic tie-breaking.

4.6 State Synchronization with CLH

CLH also underpins an efficient state synchronization mechanism:

(1) A joining node obtains a recent *snapshot* S_v at view v along with CLH_v.

(2) It verifies snapshot integrity against CLH_v.

(3) It downloads blocks $B_{v+1}, \ldots, B_{current}$ and applies their change logs.

(4) For each block B_i, it checks that locally computed CLH'_i matches the block's CLH_i.

(5) Once the tip CLH matches the network, the node switches to live participation.

Because the cost of verification scales with the number of changes, not the global state size, state sync remains efficient even for large ledgers.

5 Safety and Liveness Analysis

5.1 Safety Properties

Theorem 5.1 (Agreement)**.** *If two honest nodes finalize blocks B and B' at the same view v, then $B = B'$.*

Proof. By quorum intersection, any two quorums of size $2f+1$ intersect in at least one honest node. For a block B to be finalized at view v, we require:

- A quorum certificate QC_B for B with $2f+1$ votes.
- A child block B_{child} at view $v+1$ with $B_{\text{child}}.qc = QC_B$.
- A QC $QC_{B_{\text{child}}}$ for B_{child} with $2f+1$ votes.

Similarly for B'. There is exactly one leader per view (deterministic selection). Honest nodes vote at most once per view (enforced by $lastVote$). Thus, there can be at most one block with a valid QC in view $v+1$, so $B_{\text{child}} = B'_{\text{child}}$ and $B_{\text{child}}.qc = B'_{\text{child}}.qc$. Hence $QC_B = QC_{B'}$ and $B = B'$. □

Theorem 5.2 (Total Order)**.** *All honest nodes finalize blocks in the same order.*

Proof. Blocks are arranged in a chain indexed by view numbers. Finalization follows the two-round rule and always finalizes a parent before its children. If block B_v is finalized before B_w, then $v < w$, and the parent-child relationships enforced by the QC chain guarantee that all honest nodes observe the same prefix of finalized blocks. □

5.2 Liveness Properties

Theorem 5.3 (Eventual Progress)**.** *After GST, if the leader for some view v is honest, then some block will be finalized within bounded time.*

Proof. After GST, message delays are bounded by Δ:

(1) An honest leader for view v collects $2f+1$ NEWVIEW messages within time Δ.

(2) The leader proposes a block; replicas receive and verify it within time Δ.

(3) Replicas send VOTE messages; the next leader collects $2f+1$ votes within time Δ.

(4) The next leader proposes a block embedding the QC; the corresponding parent block is finalized under the commit rule.

Thus, progress is guaranteed within $O(\Delta)$ after an honest leader is elected post-GST. □

Theorem 5.4 (Two-Round Fast Path)**.** *In the optimistic case (honest leaders and synchronous network), a block is finalized in two rounds (*2Δ *time).*

Proof. In the optimistic case:

- Round 1 (Δ): The leader proposes, and all replicas receive and vote.
- Round 2 (Δ): The leader of view $v + 1$ collects votes, forms a QC, and proposes a child block referencing the QC.

The QC in the second proposal meets the two-round commit rule and finalizes the parent block. Total time is approximately 2Δ, compared to 3Δ for standard HotStuff. □

6 Performance Evaluation

6.1 Theoretical Analysis

6.2 Expected Performance

The implementation with 4-10 validators targets sub-second finality. Importantly, application-specific execution (order matching, state updates) is expected to dominate total latency, not consensus overhead. The consensus protocol itself (network propagation, vote collection, CLH computation) contributes a relatively small fraction of end-to-end latency, validating the efficiency of the Fast-HotStuff two-chain approach and CLH mechanism.

6.3 Scalability Analysis

Performance scales approximately linearly with validator count, with network propagation and vote collection being the primary scaling factors. CLH provides significant advantages for large state sizes: for workloads where $m \ll n$ (the number of state changes per block is much smaller than total state size), CLH's $O(m)$ complexity substantially outperforms traditional $O(n \log n)$ Merkle Patricia Trie computations.

7 Implementation Architecture

DracoBFT is implemented with a modular architecture separating concerns across multiple components: a consensus module implementing the core Fast-HotStuff protocol, a blockchain module managing block storage and finalization, a pacemaker handling view transitions and timeouts, a message hub for network communication, and a global event manager coordinating auxiliary operations.

The architecture follows a clean separation between consensus (ensuring agreement on transaction order) and execution (computing state transitions deterministically). This separation allows the consensus layer to remain application-agnostic while enabling specialized optimizations in the execution layer.

8 Formal Verification

DracoBFT includes a complete formal specification in Quint, a specification language for distributed protocols that enables executable specifications and formal verification through simulation and model checking.

8.1 Specification Overview

The formal specification comprises:

- **Core specification:** Consensus protocol with 23 state variables and 17 actions
- **Type definitions:** Blocks, quorum certificates, timeout certificates, and message types
- **Properties:** 37 formal safety and liveness invariants
- **Test suite:** 244 comprehensive tests across 9 categories

8.2 State Variables and Actions

The specification models 23 state variables including:

Protocol	Optimistic Path	View-Change	State Commitment
PBFT	3Δ	$O(n^2)$ msgs	$O(n)$
HotStuff (3-chain)	3Δ	$O(n)$ msgs	$O(n \log n)$
Tendermint	3Δ	$O(n^2)$ msgs	$O(n \log n)$
DracoBFT	$\mathbf{2\Delta}$	$\mathbf{O(n)}$ msgs	$\mathbf{O(m)}$

Table 6: Protocol comparison showing DracoBFT's Fast-HotStuff 2-chain latency, linear view-change, and CLH commitment where $m \ll n$.

Operation	DracoBFT	HotStuff	PBFT
Normal-case consensus	$O(n)$	$O(n)$	$O(n^2)$
View change	$O(n)$	$O(n)$	$O(n^2)$
State commitment	$O(m)$	$O(n \log n)$	N/A
Vote aggregation	$O(1)$ QC	$O(1)$ QC	$O(n)$ sigs

Table 7: Complexity comparison (m = state changes, n = validators).

- `validatorViews`: Current view for each validator
- `highQCs, highTCs`: Highest known certificates
- `blocks, finalizedBlocks`: Blockchain state
- `votesForBlock, timeoutsForView`: Vote and timeout tracking
- `validators, validatorStakes, totalStake`: Validator set management
- `globalEventQueue, receivedGlobalEvents`: Global event management
- `epochLeaders, currentEpoch`: Weighted leader selection

The protocol is modeled through 17 actions:

- `propose, receiveProposal, receiveVote`: Normal operation
- `localTimeout, receiveTimeout`: View changes
- `addValidator, removeValidator`: Dynamic validator set
- `recomputeLeaderSchedule`: Weighted leader rotation
- `byzantineEquivocate, byzantineDoubleVote`: Byzantine behavior modeling

8.3 Verified Safety Properties

The specification proves the following safety properties through 244 tests:

1. **Agreement:** No two validators finalize conflicting blocks at the same view
2. **Chain Validity:** All finalized blocks form a valid chain from genesis
3. **QC Validity:** Every QC references an existing block
4. **Block Validity:** Block QCs properly reference parent blocks
5. **Finalization Safety:** Finalized blocks satisfy two-chain rule
6. **View Monotonicity:** Validator views never decrease
7. **Leader Validity:** Only designated leaders can propose

8. **Vote Rule Compliance:** Validators only vote for valid blocks

8.4 Test Suite Organization

The 244 tests are organized into 9 categories covering safety properties, Byzantine adversarial scenarios, protocol features, progress guarantees, long executions, catch-up and recovery, Byzantine threshold violations, view transitions, and block height progression.

Category	Tests
Safety	18
Byzantine	29
Features	50
Liveness	27
Stress	39
Catch-Up	20
Disagreement	15
View Switching	21
Height Progression	25
Total	**244**

Table 8: Comprehensive test suite organization.

8.5 Verification Workflow

Algorithm 8 Verification Process

1: **Type-checking:** Verify specification is well-typed
2: **Simulation:** Run 50,000+ random executions per test
3: **Invariant checking:** Verify safety properties hold in all states
4: **Scenario testing:** Test specific protocol scenarios (catch-up, timeouts, Byzantine)
5: **Negative testing:** Verify agreement fails when $f \geq n/3$ (disagreement tests)

The test suite includes comprehensive coverage of Byzantine scenarios including equivocation, catch-up after missed decisions, disagreement scenarios violating Byzantine threshold assumptions, multi-round view transitions, and height progression testing.

The formal specification directly maps to the implementation, with specification actions (propose, receiveProposal, receiveVote) corresponding to concrete methods in the consensus, blockchain, and validator management modules.

8.6 Confidence Assessment

The formal specification provides high confidence for production deployment through extensive testing with random sampling. The verification accounts for standard assumptions including eventual message delivery under partial synchrony, fairness in action scheduling, and standard cryptographic primitives (collision-resistant hashes, unforgeable signatures).

9 Security Analysis

9.1 Attack Vectors

Long-Range Attacks Adversaries might attempt to create an alternative history far in the past. DracoBFT mitigates this via weak subjectivity checkpoints every K blocks (e.g., $K = 10{,}000$).

Censorship Attacks A Byzantine leader may censor transactions. View timeouts ensure leader rotation within bounded time, limiting censorship duration for any single leader.

State-Sync Attacks Adversaries could supply invalid state during sync. CLH verification ensures that only state consistent with a valid sequence of blocks and change logs is accepted.

9.2 Rational Adversary Analysis and Slashing Mechanisms

We analyze attack incentives through a game-theoretic model to demonstrate that slashing provides effective deterrence against rational Byzantine behavior.

9.2.1 Attack Payoff Model

Consider a Byzantine validator with stake S contemplating a protocol violation. The validator

faces a decision: execute the attack or behave honestly. Let P denote the profit from a successful attack (e.g., double-spending, front-running via censorship, extracting MEV through equivocation). The probability of attack detection is p, determined by cryptographic verification and peer monitoring mechanisms.

The expected payoff from attack is:

$$\mathbb{E}[\pi_{\text{attack}}] = P \cdot (1 - p) - S \cdot p \tag{5}$$

The first term represents expected profit: the attacker gains P if the attack succeeds (probability $1 - p$). The second term represents expected penalty: the attacker loses stake S if caught (probability p). A rational validator attacks only if expected payoff is positive.

The attack is deterred when:

$$\mathbb{E}[\pi_{\text{attack}}] < 0 \implies S > \frac{P}{p} \tag{6}$$

This inequality reveals the critical relationship: slashing amount S must exceed the ratio of attack profit P to detection probability p. High detection probability (cryptographic verification) or large stake requirements (economic security) both contribute to deterrence.

9.2.2 Detection Probability in DracoBFT

DracoBFT achieves high detection probability $p \approx 1$ through multiple mechanisms:

Cryptographic Verification Core consensus violations (equivocation, invalid QCs, incorrect block extensions) are detected with certainty through signature verification and quorum certificate validation. A validator cannot equivocate without producing conflicting signatures, which serves as cryptographic proof of misbehavior. Thus $p = 1$ for consensus-layer attacks.

zkTLS Proof Verification For auxiliary verification loops, validators provide zkTLS proofs of external service responses. These proofs are cryptographically verifiable: either the proof correctly demonstrates the claimed API response, or it fails verification. Validators cannot forge valid proofs for false claims without breaking underlying cryptographic assumptions (TLS signatures, zero-knowledge soundness). Detection probability $p \approx 1 - \epsilon$ where ϵ is negligible (cryptographic security parameter).

For privacy-sensitive API calls (e.g., user authentication, payment verification), the proxy-based zkTLS approach employs an *opaque proxy* architecture that provides strong security guarantees without compromising user privacy [36, 35]. The opaque proxy operates as follows:

1. **End-to-End TLS Encryption:** The client device and target HTTPS server establish a TLS session with encryption keys known only to them. The proxy handles encrypted traffic but cannot access TLS private keys, making plaintext data inaccessible.

2. **Pass-Through Operation:** Unlike traditional proxies that terminate TLS sessions, the opaque proxy forwards encrypted packets without decryption. The proxy observes the TLS handshake and encrypted traffic but cannot derive session keys due to Diffie-Hellman key exchange properties.

3. **Certificate Validation:** The proxy verifies the server's SSL/TLS certificate against the claimed domain and trusted Certificate Authorities, protecting against man-in-the-middle attacks while maintaining data confidentiality.

4. **Cryptographic Non-Tampering Proof:** The client generates a zero-knowledge proof demonstrating: (1) possession of the shared key that correctly decrypts the observed encrypted response, and (2) the decrypted content matches claimed data. The proxy provides cryptographic attestation that it witnessed the encrypted traffic from the verified server, while the ZK proof ensures the client correctly decrypted it without revealing the session key or sensitive credentials [36].

5. **Integrity Without Privacy Violation:** This architecture ensures the proxy can attest to traffic integrity (the encrypted response came from the claimed server) without accessing sensitive data (login credentials, access tokens, personal information). The formal security analysis in [36] proves this approach achieves both data integrity and privacy preservation, making it suitable for production deployment at scale.

This opaque proxy model is particularly critical for Hotstuff L1's validator-gateway architecture, where validators must verify user authentication proofs from services like Bridge.xyz and Avenia.io without gaining access to user credentials or session tokens. The cryptographic guarantees ensure validators can trustlessly verify payment authorizations and identity attestations while preserving end-to-end privacy between users and external services.

CLH Validation State divergence is detected through Change-Log Hash comparison. If a validator proposes a block with incorrect state transitions, the CLH computed by other validators will differ, immediately revealing the discrepancy. Since CLH computation is deterministic (given the same transaction sequence and initial state), detection is certain: $p = 1$ for state manipulation attempts.

Peer Monitoring Validators monitor each other's behavior: downtime, slow responses, failure to provide required proofs. While not cryptographically verified, persistent poor behavior is observable by the network and can trigger reputation penalties or reduced routing weight, creating reputational costs beyond direct slashing.

9.2.3 Stake Requirements and Attack Deterrence

Given $p \approx 1$ for most attacks, the deterrence condition simplifies to $S > P$: stake must exceed attack profit. DracoBFT implements stake requirements proportional to validator responsibilities:

Base Consensus Stake All validators must stake minimum $S_{\min}$ to participate in consensus, ensuring basic security against consensus-layer attacks. This deters small-scale attacks where $P < S_{\min}$.

Service Provider Stake Validators providing auxiliary services (user onboarding, payment processing) must stake additional S_{service} proportional to the number of onboarded users and transaction volumes they handle. This ensures $S > P$ for service-specific attacks (e.g., providing false payment authorization proofs, incorrect identity verification). As a validator's service volume grows, so does their required stake, maintaining the deterrence inequality.

Dynamic Slashing Amounts Slashing penalties are calibrated to attack severity. Minor infractions (downtime, slow responses) incur small penalties to avoid over-penalizing honest validators experiencing technical issues. Major violations (equivocation, false proofs, state manipulation) incur full stake slashing (S lost entirely), ensuring deterrence for high-profit attacks.

9.2.4 Comparison to Alternative Security Models

Table 9 compares detection mechanisms and deterrence across different consensus protocols.

Protocol	Detection	Stake	Deter.
Ethereum	High	Fixed	Med.
Tendermint	High	Fixed	Med.
DracoBFT	≈ 1	**Prop.**	**High**

Table 9: Security model comparison. DracoBFT achieves high deterrence through near-certain detection and proportional stake.

9.2.5 Limitations: Irrational and Nation-State Adversaries

The game-theoretic analysis assumes *rational* adversaries who maximize expected payoff.

However, some adversaries may act irrationally or have non-economic motivations:

Irrational Attackers An adversary may attack despite negative expected value due to spite, ideology, or errors in judgment. Slashing provides no deterrence against such adversaries. However, the protocol tolerates up to $f < n/3$ Byzantine validators regardless of rationality—irrational attacks are contained through BFT consensus properties rather than economic incentives alone.

Nation-State Adversaries A well-resourced nation-state attacker may view stake loss S as acceptable cost for disrupting the network or extracting sensitive information. For such adversaries, P (geopolitical advantage, intelligence gathering) may far exceed S (economic stake). While economic incentives fail here, cryptographic security and consensus fault tolerance still apply: a nation-state attacker controlling $\leq f$ validators cannot violate safety properties (finalization, state consistency) and can only censor transactions temporarily (until view change).

The protocol combines both incentive mechanisms (slashing deters rational adversaries) and cryptographic security (BFT consensus tolerates irrational or highly-motivated adversaries), providing defense-in-depth rather than relying solely on economic assumptions.

9.3 Network Partition Tolerance

Under a network partition:

- The minority partition cannot gather $2f+1$ votes and thus halts, preserving safety.
- The majority partition continues operation.
- Once the partition heals, the minority uses the state-sync protocol (driven by CLH) to catch up to the canonical tip.

9.4 Formal Verification Considerations

Key candidates for formal specification and verification include:

- The two-round finality rule,
- CLH construction and collision assumptions,
- View-change termination conditions,
- State-sync invariants and safety.

10 Related Work

10.1 BFT Consensus Protocols

PBFT [2] provided the first practical BFT protocol but with $O(n^2)$ communication complexity. Tendermint [3] achieves instant finality through a 3-phase commit (prevote, precommit, commit) but requires $O(n^2)$ communication for view changes.

HotStuff [1] introduced the 3-chain finality rule with linear view-change complexity, significantly improving on PBFT's quadratic communication. Fast-HotStuff reduces this to a 2-chain rule under the standard $n \geq 3f + 1$ assumption: a block is finalized when it has a certified direct child. **DracoBFT implements Fast-HotStuff** with additional optimizations for financial workloads.

Recent work has explored modified trust assumptions to achieve even lower latency. Under the stronger assumption that $n \geq 5f + 1$, protocols can achieve 1-round finality for the optimistic path. Recent examples include ChonkyBFT [20], Kudzu [17], Alpenglow [18], Hydrangea [19], and Minimmit [16], each exploring different tradeoffs between quorum sizes for view progression versus transaction finalization. These approaches can reduce view latency by 20-25% compared to standard 2-chain protocols by allowing view progression on smaller quorums.

Sync HotStuff [5] studies synchronous variants achieving improved latency bounds. Other protocol families such as HoneyBadgerBFT [9] and

Dumbo [10] target fully asynchronous settings, trading latency for stronger liveness guarantees.

Narwhal and Tusk [11] decouple data availability from consensus ordering. This approach is complementary to DracoBFT's architecture and could be integrated for improved throughput under high network load.

10.2 State Commitment Mechanisms

State commitment approaches vary significantly across blockchain systems. Ethereum uses Merkle Patricia Tries [14] incurring $O(n \log n)$ costs for global state updates. Sparse Merkle Trees (e.g., in Libra/Diem [6]) optimize proof sizes but still require global updates. Vector commitments offer constant-size proofs but often remain impractical for large, frequently updated states.

Tendermint uses an application hash [3], a Merkle root over application state computed by the state machine implementation. While this provides flexibility (each application chooses its commitment scheme), it typically defaults to computing Merkle trees over all state, resulting in $O(n \log n)$ complexity. Bitcoin uses a simpler approach, committing only to the UTXO set through a Merkle root in each block, but this still requires $O(n)$ computation where n is the number of unspent outputs.

DracoBFT's chained CLH differs fundamentally: it commits only to state *changes* ($O(m)$ where m is the number of *unique keys* modified per block), not the entire state ($O(n)$ or $O(n \log n)$). Block-level aggregation further optimizes this: if 400 transactions in a block modify the same key, CLH hashes it once (final value), not 400 times. For workloads where $m \ll n$ (sparse updates) and keys are frequently re-modified within blocks (trading, order books), this provides substantial advantages over per-operation hashing. The chaining across finalized blocks adds fork detection capability absent in traditional approaches. This makes CLH particularly suited to deterministic state machines with dense transaction processing but sparse state updates, typical of financial infrastructure.

10.3 Multi-Instance Consensus Systems

Cosmos IBC, Polkadot parachains, and Avalanche subnets [12] all support multiple consensus instances or heterogeneous chains. DracoBFT's global event management provides tighter integration within a single validator set and execution environment while maintaining modularity for auxiliary operations.

11 Future Work

11.1 Protocol Enhancements

- **Modified Trust Assumptions:** Recent protocols [16, 17, 18, 19] explore $n \geq 5f + 1$ assumptions enabling 1-round optimistic finality and reduced view latency through smaller quorums for view progression. These approaches could reduce consensus overhead by 20-25%. However, in DracoBFT's current deployment, application execution dominates end-to-end latency rather than consensus overhead, making such optimizations lower priority until execution performance improves. Future work may revisit these approaches once consensus becomes the bottleneck.

- **VRF-Based Leader Selection:** Replace deterministic weighted random selection with Verifiable Random Functions for improved randomness and resistance to leader prediction attacks.

- **Dynamic Block Sizing:** Implement adaptive block size and gas limits based on network congestion and validator performance metrics.

- **Parallel Execution:** Explore optimistic parallel transaction execution with conflict detection and rollback for improved throughput.

- **State Pruning:** Implement efficient state pruning strategies compatible with chained CLH commitments for long-running deployments.

11.2 Scalability Extensions

- **Dual-Block Architecture:** Implement heterogeneous block types with different throughput characteristics, similar to systems that decouple block frequency from block size [23]. DracoBFT could drive separate fast blocks (high frequency, low gas limit) and slow blocks (lower frequency, high gas limit) through independent mempools while maintaining unified finalization. This would allow simultaneous optimization for time-to-confirmation and transaction complexity.
- **EVM Chain Integration:** Extend DracoBFT to drive EVM-compatible chains through integration with modular execution clients such as Reth [24] or Geth using the Engine API specification. The chained CLH mechanism could replace traditional Merkle Patricia Trie state commitments in EVM chains, providing $O(m)$ state verification for Ethereum-compatible networks.
- **Sharded Consensus:** Extend global event management architecture to coordinate consensus across multiple shards with atomic cross-shard transactions.
- **Data Availability Layer:** Integrate Narwhal/Tusk or similar data availability protocols to decouple transaction dissemination from consensus ordering.
- **Validator Rotation:** Implement smooth validator rotation with overlap periods to maintain liveness during validator set transitions.
- **Light Client Support:** Develop efficient light client protocols leveraging chained CLH for state verification without full block history.

11.3 Trustless Cross-Chain Operations

- **zkTLS-Based Oracle Feeds:** Expand auxiliary verification loops to support zkTLS proofs [21] for trustless verification of off-chain data sources. This enables integration of traditional APIs (price feeds, identity verification, real-world data) without trusting intermediaries beyond standard TLS security assumptions. Current zkTLS implementations are limited to HTTPS request-response patterns and do not support WebSocket connections or webhook callbacks, which are core to many traditional financial service platforms. Future work should extend zkTLS techniques to these communication patterns to enable broader TradFi integration (e.g., real-time market data feeds, payment notification webhooks, streaming price updates).
- **ZK Execution Proofs for Cross-Chain State:** Implement zero-knowledge proofs of EVM contract execution [22] for trustless verification of actions on external blockchains. This provides full trustlessness for bridge operations, eliminating reliance on trusted relayers or multisig bridges.
- **Recursive Proof Aggregation:** Explore techniques for aggregating multiple external chain proofs into single verification transactions, amortizing proof verification costs across many cross-chain operations.
- **Proof Verification Marketplace:** Design mechanisms for distributed proof generation and verification, allowing specialized nodes to generate proofs off-protocol while validators verify them within auxiliary consensus loops.

11.4 Security Improvements

- **Post-Quantum Cryptography:** Transition to quantum-resistant signature schemes (e.g., Dilithium, SPHINCS+) for long-term security.
- **TEE Integration:** Utilize Trusted Execution Environments for secure key management and fast signature verification.
- **Slashing Conditions:** Implement on-chain slashing for provable Byzantine be-

havior (equivocation, double-voting) with stake penalties.

- **Network-Level Protection:** Add Eclipse attack resistance and DDoS mitigation strategies.

11.5 Formal Verification Extensions

While DracoBFT already includes 244 comprehensive tests, future work includes:

- **Exhaustive Model Checking:** Use Apalache for bounded model checking of deeper state spaces (currently limited to simulation).
- **Liveness Verification:** Formal proof of liveness properties under fairness assumptions using temporal logic.
- **Implementation Verification:** Explore verified implementation approaches (e.g., using Dafny or Coq) to link code directly to specification.
- **Byzantine Fault Injection:** Systematic testing of Byzantine scenarios in staging environments with controlled fault injection.

11.6 Performance Optimizations

- **Batched Signature Verification:** Aggregate BLS signatures or use batched ECDSA verification for faster QC validation.
- **Pipelined Execution:** Overlap block execution with consensus for reduced latency (execute block $n + 1$ while finalizing block n).
- **Optimized Storage:** Implement column-family based storage layout and compression for reduced I/O overhead.
- **SIMD Acceleration:** Utilize CPU SIMD instructions for cryptographic operations and state hashing.

12 Conclusion

DracoBFT is a production-ready Fast-HotStuff implementation optimized for high-performance financial infrastructure, achieving sub-second finality while maintaining strong safety guarantees through extensive formal verification. The protocol makes four major contributions:

- **Trustless Auxiliary Verification:** Modular architecture for trustless verification of external state through zkTLS proofs (for off-chain API data) and zero-knowledge execution proofs (for on-chain state from other blockchains). This enables trustless bridges, verifiable oracle feeds, and authenticated API integrations without relying on trusted intermediaries, with atomic state transitions maintained on finalization.
- **Chained Change-Log Hash Commitments:** State commitment with $O(m)$ complexity replacing $O(n \log n)$ Merkle Patricia Trie computations, where m is the number of unique keys modified per block and n is total state size. Block-level aggregation hashes each key once per block regardless of modification frequency, eliminating redundant I/O for high-frequency trading workloads. The chained CLH links finalized state across blocks, enabling early fork detection and historical state verification. For workloads where $m \ll n$, CLH provides substantial performance improvements.
- **Fast-HotStuff 2-Chain Implementation:** Implementation of the Fast-HotStuff 2-chain finality rule with vote rule block.view == block.qc.view + 1, where finalization occurs when a block has a certified direct child.
- **Weighted Stake-Based Leader Selection:** Deterministic weighted random leader selection with epoch-based computation, ensuring fairness proportional to validator stake.

12.1 Formal Verification Achievement

The protocol includes a complete formal specification with 37 safety and liveness properties verified through 244 comprehensive tests across 9 categories, providing high confidence for production deployment.

12.2 Planned Deployment

DracoBFT is designed for deployment as the consensus layer for Hotstuff L1, a DeFi-focused Layer 1 blockchain that combines a high-performance on-chain order book with a programmable financial routing layer. In this planned deployment, validators will function as last-mile gateways providing verified access to trading venues, payment processors, and fiat conversion services.

The architecture positions Hotstuff L1 as a purpose-built chain where validators act as permissioned financial service providers. Rather than serving only as block validators, nodes will opt-in to provide verified access to external financial services, with the auxiliary verification loops ensuring service delivery through cryptographic proofs. This validator-as-gateway model will enable the chain to function as a routing layer matching users to appropriate service providers based on stake, performance history, geographic coverage, and quality-of-service metrics.

The implementation targets sub-second finality with linear scalability across validator set sizes. Critically, application-specific execution (order matching, position updates, state transitions) is expected to dominate end-to-end latency rather than consensus overhead, validating the protocol's efficiency. CLH computation scales with the number of state changes rather than total state size, providing substantial performance advantages for sparse-update workloads.

The planned deployment in Hotstuff L1 will validate DracoBFT's suitability for financial infrastructure where deterministic execution, trustless external verification, and validator service provision are critical requirements.

12.3 Future Directions

Potential enhancements include exploring modified trust assumptions ($n \geq 5f + 1$) for reduced view latency once application execution is optimized, expanding auxiliary verification loops to support broader zkTLS and ZK execution proof systems for trustless cross-chain operations, integrating data availability layers for improved throughput, implementing VRF-based leader selection, and adopting post-quantum cryptographic schemes for long-term security.

DracoBFT demonstrates that Fast-HotStuff can be effectively implemented for financial infrastructure, achieving both the theoretical guarantees of BFT consensus and the practical performance requirements of high-frequency trading systems. The combination of formal verification and production deployment validates both theoretical correctness and real-world feasibility.

Acknowledgments

This work builds upon foundational research in Byzantine fault tolerance and the HotStuff protocol family. We are particularly grateful to:

- The HotStuff and Fast-HotStuff authors for pioneering linear-complexity BFT protocols with responsive performance
- The Informal Systems team for developing Quint and advancing formal verification methodologies for distributed protocols
- The broader distributed systems and formal methods communities for their contributions to BFT consensus theory and practice

References

[1] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[2] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186, 1999.

[3] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[4] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, pages 45–58, 2007.

[5] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118, 2020.

[6] Mathieu Baudet et al. State machine replication in the Libra blockchain. Technical Report, 2019.

[7] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

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

[9] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42, 2016.

[10] Bin Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous BFT protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 803–818, 2020.

[11] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: A DAG-based mempool and efficient BFT consensus. *arXiv preprint arXiv:2105.11827*, 2021.

[12] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless BFT consensus through metastability. *arXiv preprint arXiv:1906.08936*, 2019.

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

[14] Christian Cachin and Marko Vukolić. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[15] Alysson Bessani, João Sousa, and Eduardo E. Alchieri. State machine replication for the masses with BFT-SMART. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362, 2014.

[16] Brendan Kobayashi Chou, Andrew Lewis-Pye, and Patrick O'Grady. Minimmit: Fast finality with even faster blocks. *arXiv preprint arXiv:2508.10862*, 2025.

[17] Victor Shoup, Jakub Sliwinski, and Yann Vonlanthen. Kudzu: Fast and simple high-throughput BFT. *arXiv preprint arXiv:2505.08771*, 2025.

[18] Quentin Kniep, Jakub Sliwinski, and Roger Wattenhofer. Solana Alpenglow consensus. Technical Report, Solana Labs, 2025.

[19] Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Hydrangea: Optimistic two-round partial synchrony with one-third fault resilience. *Cryptology ePrint Archive*, 2025.

[20] Matter Labs. ChonkyBFT: A new approach to Byzantine fault tolerance. Technical Report, Matter Labs, 2024.

[21] Nascent. Crypto's AirTag moment: zkTLS and verifiable data. `https://www.nascent.xyz/idea/cryptos-airtag-moment`, 2024.

[22] Succinct Labs. SP1 Contract Call: Verifiable EVM state proofs. `https://succinctlabs.github.io/sp1-contract-call/`, 2024.

[23] Hyperliquid. Dual-block architecture for decoupling block speed and size. `https://hyperliquid.gitbook.io/hyperliquid-docs/for-developers/hyperevm/dual-block-architecture`, 2024.

[24] Paradigm. Reth: Modular, contributor-friendly Ethereum implementation in Rust. `https://github.com/paradigmxyz/reth`, 2024.

[25] Rain. API integration webhooks: Payment authorization response time requirements. `https://docs.rain.xyz/docs/api-integration-webhooks`, 2024.

[26] Arc Network. Stable fee design: USDC-denominated gas fees. `https://docs.arc.network/arc/concepts/stable-fee-design`, 2024.

[27] Plasma Network. Network fees: Zero-fee USDT transfers via paymaster. https://plasma.to/docs/plasma-chain/network-information/network-fees, 2024.

[28] Hyperliquid. Hyperliquid documentation: HyperCore and HyperEVM architecture. https://hyperliquid.gitbook.io/hyperliquid-docs, 2024.

[29] MegaLabs. MegaETH: Unveiling the first real-time blockchain - State root update challenges. https://www.megaeth.com/research, 2024.

[30] Lei Yang. SALT: Space-optimal authenticated key-value store for MegaETH. https://x.com/yangl1996/status/1957487663818416406, 2025.

[31] Bridge.xyz. Bridge API Documentation: KYC and Account Management. https://apidocs.bridge.xyz/, 2024.

[32] Avenia. Avenia Integration Guide: Account Management and Compliance. https://integration-guide.avenia.io/docs/Avenia-Account-Management/login, 2024.

[33] TLSNotary Project. zkTLS: Zero-Knowledge Proofs for TLS Sessions. https://tlsnotary.org/, 2024.

[34] Madhavan Malolan. The zk in zkTLS. Reclaim Protocol Blog, https://blog.reclaimprotocol.org/posts/zk-in-zktls, 2024.

[35] Reclaim Protocol. Reclaim Protocol Whitepaper. https://drive.google.com/file/d/1Tok4J6mv7PwRCbwxVNhv4alS82sQJI4E/view, 2024.

[36] Zhongtang Luo, Yanxue Jia, Yaobin Shen, and Aniket Kate. Proxying is enough: Security of proxying in TLS oracles and AEAD context unforgeability. In *7th Conference on Advances in Financial Technologies (AFT)*, 2025. Also available at Cryptology ePrint Archive, Report 2024/733, https://eprint.iacr.org/2024/733.

[37] Binance Academy. What Are API Keys and Security Types: EdDSA Algorithm Comparison. https://www.binance.com/en/academy/articles/what-are-api-keys-and-security-types, 2024.

[38] Deribit Support. Asymmetric API Keys with EdDSA. https://support.deribit.com/hc/en-us/articles/25944616699165, 2024.

[39] Ethena Labs. Ethena Custody Risk: Off-Exchange Settlement Model. https://docs.ethena.fi/solution-overview/risks/custodial-risk, 2024.

[40] HyperLiquid. HLP: HyperLiquid Liquidity Provider Vault. https://hyperliquid.gitbook.io/hyperliquid-docs/trading/hlp, 2024.

[41] Tim Roughgarden. Twenty lectures on algorithmic game theory. Cambridge University Press, 2016.

[42] Vitalik Buterin and others. Combining GHOST and Casper. *arXiv preprint arXiv:2003.03052*, 2020.

[43] Miles Carlsten, Harry Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *ACM Conference on Computer and Communications Security (CCS)*, pages 154–167, 2016.

[44] Bruno Biais, Christophe Bisiere, Matthieu Bouvard, and Catherine Casamatta. The blockchain folk theorem. *Review of Financial Studies*, 32(5):1662–1715, 2019.

A Change-Log Hash Specification

A.1 Running Hash Computation

The change-log hash is computed as a running SHA256 hash over all state operations during block execution:

Algorithm 9 CLH via Running Hash

```
hasher ← SHA256()
for all state operation op during block execution do
    if op = PUT(key, value) then
        hasher.WRITE('P'‖key‖value)
    else if op = DELETE(key) then
        hasher.WRITE('D'‖key)
return hasher.SUM() as finalizedCLH
```

A.2 Chained Hash Computation

The chained change-log hash links consecutive finalized blocks:

Algorithm 10 Chained CLH Computation

```
procedure COMPUTECHAINEDCLH(prevBlock)
    hasher ← SHA256()
    hasher.WRITE(prevBlock.chainedCLH)
    hasher.WRITE(prevBlock.finalizedCLH)
    return hasher.SUM()
```

This chaining provides: (1) historical state linking across finalization boundaries, (2) early fork detection when validators have divergent finalized state, and (3) efficient verification that all ancestors share consistent state history.

B Formal Specification Overview

The formal specification models the protocol through state variables (validator views, certificates, blockchain state, global events) and actions (propose, vote, timeout, validator management, Byzantine behavior). The specification includes formal safety and liveness invariants covering agreement, chain validity, finalization safety, view monotonicity, leader validity, and vote rule compliance.

The comprehensive test suite provides coverage across core safety properties, Byzantine adversarial scenarios, protocol features, progress guarantees, catch-up and recovery, Byzantine threshold violations, view transitions, and block height progression.