DracoBFT: Two-Chain Consensus with Validator Service Loops for Financial Infrastructure

Madhur Kumar Sharma* Hiten Jain† Vyom Sharma‡

Hotstuff Labs

August 2026

Abstract

Financial ledgers must agree on an ordered history while responding to information that originates outside the replicated state machine. We present DracoBFT, the consensus and validator-service stack operated by Hotstuff Labs since January 2026. The consensus protocol combines stake-weighted quorum and timeout certificates, an adaptive pacemaker, deterministic weighted leader selection, and two-certified-link finality. Separate validator service loops collect market, payment, and venue data, then submit results through the ordinary deterministic transaction path after satisfying a service-specific attestation, proof, or signing rule. The production system runs multi-source price feeds, Ethereum deposit admission, venue-equity synchronization, Reclaim zkTLS/zkFetch verification for Bridge.xyz neobanking actions, and threshold-ECDSA signing for venue operations.

We specify the vote, view-change, and finalization rules; prove safety and liveness under partial synchrony; derive message and latency bounds; and state the replay, freshness, and external-trust assumptions of each service. We also specify the chained change-log hash (CLH), which records the ordered mutation transcript of finalized execution, and a canonical domain-separated format for CLH-v2. In a controlled four-validator experiment, DracoBFT committed 48.703 and 97.404 transactions/s at offered loads of 50 and 100 transactions/s. Median client-observed latency was approximately 100 ms and p99 latency approximately 149 ms, with no observed safety or liveness violations. We relate these measurements to operational observations, an executable Quint specification, and implementation tests. The paper concludes with authenticated checkpoints, light clients, recursive proofs, provider markets, post-quantum signatures, and mechanized conformance.

*madhur@hotstufflabs.com; University of Waterloo alumnus, Computer Engineering.

†hiten@hotstufflabs.com

‡Correspondence: vyom@hotstufflabs.com

1 Introduction

Byzantine fault-tolerant replication gives a blockchain a deterministic history despite a bounded adversary. Financial applications need more than transaction ordering. Prices, deposits on other chains, balances at trading venues, payment status, and identity or compliance decisions all originate outside the state machine. Fetching them during block validation would tie consensus progress to external latency and availability. Accepting them from a single operator would place that operator inside the trust boundary.

DracoBFT separates these concerns. The consensus core orders deterministic transactions, while independent *validator service loops* collect or verify external facts in parallel. Each loop applies its own evidence rule and submits an ordinary transaction after satisfying the required stake threshold, proof policy, or threshold-signing authorization. An unavailable exchange, API, proof generator, or service leader may delay one service, but it does not enter the critical path of unrelated consensus views. Validators can therefore use the same identity and stake for replicated ordering and accountable service provision, with a distinct trust model for each role.

Hotstuff Labs has operated DracoBFT in production since January 2026. The deployed stack includes two-chain consensus, price-feed aggregation, Ethereum deposit admission, venue-equity synchronization, Reclaim zkTLS verification, threshold-ECDSA signing, and chained change-log hashes. We describe the protocol, its implementation, and the operating characteristics of these services.

1.1 Why implementation choices matter

Small implementation choices determine the guarantees that the system can support. DracoBFT accepts a QC at exactly two thirds of stake, elects leaders from a deterministic weighted schedule, and finalizes only after observing two consecutive certified links. CLH hashes the database mutations in execution order. The Reclaim verifier reconstructs the zkFetch claim identifier, recovers the EIP-191 signer, enforces Bridge.xyz origin and route policy, checks action-specific fields, and applies a

three-minute freshness window. Sections 3–5 give the corresponding protocol rules.

1.2 Contributions

The paper makes seven contributions.

1. A complete specification of the production two-chain protocol, including QC, timeout-certificate (TC), and aggregate-QC validation; proposal admission; adaptive view change; weighted leader selection; finalization; and persistent safety state.
2. Safety and liveness results tied to the implemented rules, with quorum-intersection and unique-certification lemmas, post-GST latency bounds, message complexity, and explicit reconfiguration assumptions.
3. A common architecture for five production services: multi-source price feeds, Ethereum deposits, venue-equity snapshots, Reclaim zkTLS neobanking, and threshold-ECDSA signing. Each service retains its own evidence and trust model.
4. A specification and analysis of CLH-v1 as an ordered mutation commitment, together with a backward-compatible CLH-v2 format that adds canonical framing and domain separation.
5. An implementation account that maps the protocol to its major components and gives pseudocode for vote admission, CLH, and Reclaim verification.
6. Controlled four-validator measurements, production observations, executable Quint scenarios, implementation tests, and mathematical analysis, reported as distinct sources of evidence.
7. A research agenda for validator service markets, light-client and cross-chain proofs, recursive zkTLS aggregation, confidential execution, authenticated checkpoints, post-quantum migration, fault injection, and implementation conformance.

1.3 System scope

Table 1 separates the production system from the specification and planned extensions.

1.4 Design scope and trust boundaries

The consensus protocol belongs to the two-chain HotStuff/Fast-HotStuff family [20, 11, 7]. DracoBFT's contribution is the way this protocol is combined with validator service loops, proof-carrying admission, threshold signing, and CLH in a production financial system. A Reclaim claim establishes the provenance of selected API content under Reclaim's attestor and TLS construction; the originating service remains authoritative for the account or payment state it reports. CLH commits the ordered execution transcript for replica comparison. Authenticated checkpoints, discussed in Section 12, add state membership proofs and trust-minimized bootstrap. Section 9 reports controlled measurements separately from production observations.

1.5 Organization

Section 2 defines the system and trust model. Sections 3–5 present consensus, validator service loops, zkTLS admission, and CLH. Sections 6 and 7 analyze incentives and correctness. Sections 8–10 cover the implementation, evaluation, test suite, and Quint specification. Sections 11 and 12 place the work in context and outline the remaining research problems.

2 System, Network, and Trust Model

2.1 Replicas, stake, and epochs

Let $\mathcal{V}_e$ denote the active validator set in epoch e, with stake function $w_e(v)$ and total weight $W_e = \sum_{v \in \mathcal{V}_e} w_e(v)$. A quorum has weight at least $2W_e/3$; the implementation evaluates this condition as $3w(Q) \geq 2W_e$. Byzantine stake is strictly less than $W_e/3$. Certificates bind the validator-set state, signed message, and distinct signers. Repeated signatures from one validator contribute weight once.

Each epoch seed covers 10,000 views, and the implementation precomputes 11,000 leader positions to span the epoch boundary. Validator-set changes take effect through finalized state transitions. We assume *epoch consistency*: certificate verification uses the validator set active for the certificate's view, and no quorum combines stake from different configurations.

2.2 Network and cryptography

We assume partial synchrony [6]. Before an unknown global stabilization time (GST), messages may be delayed, reordered, duplicated, or dropped. After GST, messages between honest validators arrive within Δ. The adversary may crash or equivocate, withhold votes, submit malformed transactions or evidence, control external endpoints, and schedule messages within this model.

We assume existentially unforgeable signatures and collision resistance for SHA-256 and Keccak-256. Proposals, votes, new-view messages, and timeout evidence are authenticated. Memory safety, key custody, deployment security, and dependency correctness remain engineering assumptions outside the consensus proof.

Table 1: DracoBFT system scope.

Component	Status	Scope
Two-chain consensus	Production	Stake-weighted QC/TC/AggQC verification, adaptive pacemaker, weighted leaders, two-certified-link finality, persistence, catch-up, and validator-set updates.
Price-feed service	Production	Multi-source collection, source filtering, weighted aggregation, validator attestations, and deterministic price admission.
Deposit service	Production	External receipt and event validation, quorum approval, replay protection, and deterministic account credit.
Venue-equity service	Production	Independent venue observations, tolerance-based comparison, quorum approval, and deterministic equity updates.
Reclaim zkTLS/zkFetch	Production	Attestor-signed Bridge.xyz claims with signer, origin, route, response-field, freshness, and replay checks.
Threshold-ECDSA signing	Production	Distributed key shares, authorized request lookup, interactive signing, session control, and signature publication.
Proof-carrying admission	Production	Service-specific request identifiers, evidence, freshness rules, policy checks, and durable replay state.
Chained change-log hash	Production	Ordered SHA-256 mutation recording and chaining across finalized execution.
Quint specification	Protocol analysis	Executable consensus and service-loop scenarios with safety and liveness properties.
Light clients, recursive proofs, provider markets, and post-quantum signatures	Future work	Extensions to the production protocol described in Section 12.

2.3 Blocks and certificates

A block B contains its view $v(B)$, parent hash $p(B)$, transaction Merkle root, proposer, timestamp, QC, optional aggregate QC, chained CLH, and the proposer's latest finalized view. We write $B \leftarrow A$ when $p(B) = H(A)$ and B carries a QC for A. A timeout message signs a view and includes the sender's high QC. A TC/AggQC combines timeout evidence from a quorum and identifies the highest valid QC from which the protocol resumes.

Proposal admission requires the QC target to exist and the parent hash to name that same block. This parent–QC invariant binds every accepted proposal to one certified branch.

2.4 Deterministic execution

Honest validators execute a finalized block from the same parent state, with the same transaction order and engine version. All writes pass through a recorded database batch. The execution path excludes external calls, wall-clock reads, nondeterministic iteration, floating-point ambiguity, and node-local policy. Proposer timestamps must increase relative to the parent and are treated as protocol metadata, not wall-clock truth.

Assumption 2.1 (Deterministic engine)**.** *For a common parent state and valid ordered transaction list, honest replicas emit identical return values, events, validator-set updates, and ordered storage-mutation byte transcripts.*

2.5 External-service trust

Validator service loops run outside the consensus event loop. Their results enter state through deterministic actions whose evidence is checked during execution. DracoBFT uses four evidence classes: (i) stake-weighted attestations over independently observed inputs; (ii) external-chain receipts or events checked through RPC and approved by a quorum; (iii) Reclaim zkFetch claims signed by registered attestors and checked against application predicates; and (iv) threshold-ECDSA results over previously authorized request hashes. Trust remains distributed across the relevant source chain, market venues, Bridge.xyz, RPC endpoints, PKI, Reclaim attestors, MPC library, and key-share custody.

An adversary may replay old evidence or attach a valid proof to different request parameters. Service actions therefore bind the request identity, nonce or processed-event identifier, route, response fields, and a block-relative freshness condition. Redaction limits disclosure, although public action fields and metadata may still reveal sensitive relationships. Confidentiality is treated separately from consensus safety.

2.6 Target properties

Agreement.
No two honest validators finalize conflicting blocks at one view or on divergent histories.

Chain growth.
After GST, recurring honest leaders and valid transactions eventually produce new finalized blocks.

Deterministic admission.
A service result either passes or fails identically at every honest replica with the same parent state.

At-most-once external effect.
A unique deposit, synchronization event, or request cannot update application state twice.

Service non-interference.
Failure of an auxiliary source or proof path leaves unrelated consensus progress available.

Execution-divergence detection.
Honest replicas sharing a finalized CLH anchor compute the same next chained CLH; different mutation bytes are detected except with a hash collision or encoding alias.

3 Two-Chain Consensus Protocol

3.1 State and invariants

Validator i persists its current view, highest verified QC highQC_i, greatest signed view $\mathsf{lastVote}_i$, finalized tip, validator-set state, and block forest. The pacemaker can be reconstructed after restart, whereas $\mathsf{lastVote}_i$ must remain durable to prevent a second vote in the same view. Leaders broadcast proposals, validators send votes to the next-view leader, timeout messages are broadcast, and new-view evidence is sent to the elected leader.

Before voting, a validator checks the QC or AggQC, proposer signature, elected leader, vote rule, transaction Merkle root, CLH, parent availability, timestamp monotonicity, and block-forest consistency. Listing 1 isolates the compact vote-rule component of this sequence.

3.2 Quorum and timeout certificates

A QC for block B consists of quorum stake signing $\langle \mathsf{VOTE}, v(B), H(B) \rangle$. Verification checks signer uniqueness, validator membership, the signed message, aggregate signature, and total stake. A timeout message for view v carries the sender's high QC. Quorum timeout evidence forms $TC(v)$; the aggregate representation batches the signatures and carries their highest valid QC. The next leader can then resume from a safe branch.

On the normal path, the proposal must extend a QC from the immediately preceding view. After a timeout, the AggQC establishes the highest safe QC and the block forest checks that the proposal extends its block. The same parent–QC invariant therefore applies on both paths.

3.3 Leader selection

For view v, the epoch starts at $s = 10{,}000\lfloor v/10{,}000 \rfloor$. Validators canonically sort the active set, seed the deterministic pseudo-random generator with s, and draw 11,000 positions from the cumulative stake distribution. Algorithm 2 gives the procedure.

Algorithm 1 Proposal admission and voting at validator i

procedure OnProposal(B)
Require: $\mathsf{VerifyLeaderSig}(B)$
Require: $\mathsf{proposer}(B) = \mathsf{Leader}(v(B))$
  if $B.\mathsf{AggQC} \neq \perp$ **then**
Require: $\mathsf{VerifyAggQC}(B.\mathsf{AggQC})$
    update highQC_i from its highest valid QC
  else
Require: $\mathsf{VerifyQC}(B.QC)$
Require: $v(B) \geq \mathsf{view}_i$ and $v(B) = v(B.QC) + 1$
Require: $p(B) = B.QC.\mathsf{blockHash}$ and the parent is available
Require: $\mathsf{MerkleRoot}(B.\mathsf{txIDs}) = B.\mathsf{root}$
Require: parent timestamp $\leq B.\mathsf{timestamp}$
Require: the advertised CLH matches when finalized views agree
Require: $v(B) > \mathsf{lastVote}_i$
  persist $\mathsf{lastVote}_i \leftarrow v(B)$ and the block
  send $\mathsf{Vote}(H(B), v(B))$ to $\mathsf{Leader}(v(B)+1)$

Algorithm 2 Deterministic weighted leader schedule

procedure ComputeLeaders($s, \mathcal{V}, w$)
  $\mathsf{rng} \leftarrow \mathsf{PRNG}(s)$
  $C[j] \leftarrow \sum_{k \leq j} w(\mathcal{V}[k])$
  for $r = 0$ to 10,999 **do**
    $x \leftarrow \mathsf{rng.Int63n}(C[|\mathcal{V}| - 1])$
    $L[r] \leftarrow \min\{j : C[j] > x\}$
  return L

For a fixed validator set, the selection probability of validator i is w_i/W, making its expected leader share proportional to stake. Because the full epoch schedule is deterministic, future leaders are predictable. Section 12 considers VRF and threshold-randomness alternatives.

3.4 Fast path and finalization

Consider blocks A, B, and C with $B \leftarrow A$, $C \leftarrow B$, and $v(B) = v(A) + 1$. When a validator processes C, it has both B and $QC(B)$. The QC carried by B must certify A in the preceding view. The validator can then finalize A together with any unfinalized ancestors between A and the previous finalized tip. One QC certifies a block; two consecutive certified links finalize it.

Definition 3.1 (Two-certified-link condition)**.** *A block A is committable when an honest validator observes certified block B with $v(B) = v(A) + 1$ and a valid later proposal carrying $QC(B)$, with parent/QC links $C \leftarrow B \leftarrow A$.*

Algorithm 3 Finalization advancement

procedure AdvanceFinalization(C)
  $B \leftarrow \mathsf{Get}(C.QC.\mathsf{blockHash})$
  $A \leftarrow \mathsf{Get}(B.QC.\mathsf{blockHash})$
  if $v(A) + 1 \neq v(B)$ **then**
    return
  walk $B.QC$ ancestors to the latest finalized block
Require: the walk terminates at its exact hash and view
  for all unfinalized ancestors in ascending order **do**
    execute transactions through the recorded batch
    set the block's finalized CLH and atomically commit the batch
    apply finalized validator-set updates and emit events
  persist the new finalized tip and prune obsolete forks

Algorithm 4 Timeout-driven view advance

procedure OnLocalTimeout(v)
  persist and broadcast $\langle v, \mathsf{highQC}_i, \sigma_i \rangle$
procedure OnTimeoutSet(T)
Require: distinct valid signers in T have weight $\geq 2W/3$
  $q \leftarrow$ highest verified QC carried in T
  $a \leftarrow \mathsf{Aggregate}(T, q)$
  $\mathsf{view}_i \leftarrow 1 + \max(v(T), v(q))$
  persist view and high QC
  if $i = \mathsf{Leader}(\mathsf{view}_i)$ **then**
    propose extending q with a
  else
    send new-view evidence to the leader

3.5 Pacemaker and view change

When view v times out, a validator broadcasts a signed timeout carrying highQC_i. Evidence from at least $2W/3$ stake forms a TC/AggQC. Validators advance to one view beyond the highest view justified by the QC or timeout evidence, and the new leader proposes from the highest safe QC.

The pacemaker estimates view duration as the recent mean plus 1.96 standard deviations and increases the estimate after a timeout. Production bounds the resulting duration to $[1\text{ s}, 2\text{ s}]$. The liveness result in Section 7 assumes that the chosen timeout exceeds post-GST communication and processing delay.

3.6 Communication and latency

The stable path uses one leader broadcast and one vote from each validator to the next leader, for $O(n)$ transmissions and deliveries apart from payload dissemination. During a timeout, every validator broadcasts its timeout message, yielding up to $O(n^2)$ deliveries. Aggregate signatures keep the resulting certificate compact but do not change the number of timeout messages delivered.

Let τ_p denote proposal construction and validation, τ_v vote verification and QC construction, τ_e deterministic execution and storage commit, and Δ the post-GST one-way delay bound. A QC-forming view completes within $2\Delta + \tau_p + \tau_v$. Two later QC-forming rounds expose the certified links needed to finalize a newly proposed block, giving

$$L_{\text{final}} \leq 4\Delta + 2(\tau_p + \tau_v) + \tau_e \tag{1}$$

from receipt of the initial proposal, before queueing and finalization pacing. A failed leader adds the local timeout and timeout-certificate exchange. Production also uses a 150 ms finalization pacing floor and records a slow block after 600 ms once the system has warmed up. These operating thresholds are separate from the analytical delay bound.

Algorithm 5 Stake-attested validator service loop

procedure ServiceRound(s, r, a)
  $x_i \leftarrow \mathsf{Observe}_s(r, a)$ outside consensus
  broadcast $\langle s, r, a, x_i, \sigma_i \rangle$
  collect distinct valid reports R
  $y \leftarrow \mathsf{Aggregate}_s(R)$
Require: supporting stake for y is at least $2W/3$
  submit action $A = (s, r, a, y, R, \mathsf{ReplayKey}_s(y))$
procedure ExecuteAction(A, S)
Require: $\mathsf{ReplayKey}(A) \notin S.\mathsf{processed}$
Require: $\mathsf{VerifyQuorum}(A)$ and $\mathsf{Accept}_s(A, S)$
  apply the deterministic transition
  record replay key atomically with the transition

4 Validator Service Loops

4.1 Isolation pattern

A service loop reads finalized state, performs variable-latency I/O through a separate worker and bounded queue, constructs evidence, and submits an ordinary deterministic action. State changes still pass through BFT finalization. Algorithm 5 gives the stake-attested form used by the price-feed, deposit, and venue-equity services. Reclaim admission replaces the report quorum with an attestor proof and application predicates; threshold ECDSA replaces it with an authorized-request lookup and a two-share signing session. Every service defines its own acceptance function and durable replay key.

This separation contains slow or faulty external services without weakening the consensus event loop. In a stake-attested loop, the resulting certificate records validator agreement over the submitted evidence; Reclaim and threshold signing carry their native cryptographic

evidence instead. Accuracy still depends on the relevant external trust boundary: correlated venues, a dishonest API, shared RPC infrastructure, or a compromised zkTLS attestor can cause several validators to accept the same false observation.

4.2 Price aggregation

The price-feed service maintains cached streams for centralized exchanges, on-chain sources, and Pyth feeds. The implementation includes adapters for Binance, Hyperliquid, OKX, Bybit, Ondo, Kraken, MEXC, Gate.io, KuCoin, Pyth, and specialized on-chain/ORRN sources. Each market enables the subset appropriate to its index.

Each validator computes a local price every two seconds, except on the configured USDC path, which uses a sixty-second interval. When at least three sources are available, observations pass through a median-absolute-deviation filter with threshold 3.0 and a minimum 0.5% deviation guard. The validator then computes a weighted median. Validators exchange these local results, and the round value is the median supported by the required stake. Source operations are timed; a read longer than 10 ms is recorded as slow because the normal path reads an already populated stream.

Definition 4.1 (Weighted median)**.** *For observations (p_j, w_j) sorted by price, a weighted median is the first p_k such that $\sum_{j \le k} w_j \ge \frac{1}{2} \sum_j w_j$.*

Under a source-diversity assumption, the median/-MAD construction rejects isolated outliers. The finalized result is a stake-certified aggregate of the observations available in that round. Its market accuracy therefore depends on the independence and quality of the enabled sources.

4.3 Ethereum deposit admission

For each deposit, the service verifies the source-chain receipt, event fields, and block number and rejects any event already present in the replay set. The round leader proposes a batch, which the remaining validators verify independently before signing. After the stake threshold is reached, the certified batch enters the mempool and executes deterministically. The account credit and the unique chain/log or deposit identifier are written atomically.

Deposit admission spans two finality domains. The configured Ethereum/RPC rule determines when a receipt becomes eligible, and DracoBFT finality determines when the resulting credit becomes irreversible on this ledger. Correlated RPC endpoints remain part of the source-chain trust assumption. Section 12 describes a light-client adapter that can replace this assumption with direct or succinct inclusion verification.

4.4 External-venue equity

Every five seconds, the elected service leader obtains venue-equity snapshots. Other validators fetch the corresponding account state independently and accept a proposal when its total equity lies within ±0.5% of their local observation. Fetches use a five-second timeout with retries, so an unavailable venue cannot block consensus processing. A stake quorum certifies the resulting venue-equity action, which is checked again at mempool admission and execution.

The tolerance absorbs small differences in observation time and mark price, while fixing the maximum disagreement accepted by the protocol. A venue that equivocates consistently, or validators that share credentials and infrastructure, can still create correlated observations. A fuller accounting record can retain source timestamps and asset-level components together with a commitment to the raw statements, in addition to aggregate equity.

4.5 Reclaim zkTLS/zkFetch admission

The neobank service admits selected Bridge.xyz API responses through Reclaim proofs. A Reclaim proof carries the provider, public parameters, canonicalizable context, owner, timestamp, epoch, claim identifier, and one or more attestor signatures. API credentials remain in Reclaim's secret options and do not appear in the public claim. Admission applies the following deterministic checks:

1. recompute the signed claim identifier

$$id = \mathsf{K256}(\mathsf{provider} \parallel \texttt{"\textbackslash n"} \parallel \mathsf{parameters} \parallel \texttt{"\textbackslash n"} \parallel \mathsf{canonicalContext})$$

   and require equality with the identifier carried by the proof;

2. rebuild the Reclaim signing string from identifier, lower-cased owner, timestamp, and epoch; recover every EIP-191 signer and require a signer in the configured Reclaim attestor set;

3. parse the parameters as structured JSON and restrict the URL to the production or sandbox Bridge.xyz origin and an action-specific route;

4. require the proof timestamp to be no earlier than the executing block timestamp minus three minutes;

5. require action-specific response matches and exact identifiers, enums, amounts, currencies, addresses, or transaction hashes; and

6. reject processed deposit identifiers and action nonces so that a valid proof cannot be credited twice.

The service covers customer registration and synchronization, virtual accounts, liquidation addresses, external accounts, transfers, and deposit synchronization. For

Algorithm 6 Reclaim-backed neobank admission

procedure VERIFYRECLAIMACTION(P, A, t_B)
Require: $P \neq \bot$ and $|P.\mathsf{signatures}| > 0$
Require: $P.\mathsf{identifier} = \mathsf{ComputeIdentifier}(P)$
Require: recovered EIP-191 signer is a registered attestor
    $q \leftarrow \mathsf{ParseHTTPParameters}(P.\mathsf{parameters})$
Require: $q.\mathsf{method}$ and exact URL satisfy policy for $A.\mathsf{type}$
Require: $1000P.\mathsf{timestampS} \geq t_B - 180{,}000$
Require: $\mathsf{ResponsePredicates}(q.\mathsf{matches}, A)$
Require: $\mathsf{ReplayKey}(A) \notin \mathsf{processed}$
    return valid

a deposit, the credited amount is derived from the verified receipt rather than supplied independently by the requester.

The adapter accepts only a Reclaim attestor-signed claim that also satisfies the predicates of the requested action. Its trust boundary includes Reclaim's proof and attestor construction, the configured attestor keys, HTTPS/PKI, Bridge.xyz, and DracoBFT consensus. The deployed freshness rule rejects timestamps older than three minutes relative to the executing block. A symmetric upper bound is included in the next verifier-policy version to reject claims dated too far into the future.

4.6 Threshold-ECDSA signing

DracoBFT also runs an interactive threshold-ECDSA service based on `tss-lib`. Each participating validator loads one local key share and the same canonically sorted party metadata. The deployed configuration has three parties and TSS threshold parameter one, so a signature requires two shares. Before joining a session for digest h, a party looks up h in the validated multi-venue request store and rejects any digest not authorized there. The parties exchange broadcast and addressed MPC messages over authenticated channels and publish the ECDSA signature after the local protocol instance completes.

The signed digest identifies the session and prevents duplicate concurrent instances. A session has a 30 s deadline; while the local party is initializing, an early peer message may be retried up to six times at 1 s intervals. Key shares are recovered from protected storage. This service authorizes confidential multi-venue operations without reconstructing the signing key at a single validator.

Consensus and threshold signing govern different decisions. DracoBFT determines which multi-venue request is admissible and durable. The TSS protocol requires two key-share holders to produce the external ECDSA signature. Its security also relies on share generation, backup and rotation, authenticated party-to-peer mapping, the correctness of the TSS library, and the acceptance rule of the external contract. Section 12 discusses proactive refresh, ceremony verification, and compromise recovery.

4.7 Common proof-carrying envelope

The five services retain native evidence formats. The following normalized envelope gives those formats a common set of request, policy, provenance, and replay fields:

$$\begin{aligned} E = (&\mathsf{chainID}, \mathsf{requestID}, \mathsf{serviceID},\\ &\mathsf{policyVersion}, \mathsf{provider}, \mathsf{anchor},\\ &\mathsf{observedAt}, \mathsf{inputCom}, \mathsf{outputCom},\\ &\mathsf{proofType}, \mathsf{proofHash}, \mathsf{nonce}). \end{aligned} \tag{2}$$

Each verifier binds its native evidence to a round or request identity and to durable replay state. Equation (2) adds uniform chain, service, policy, proof-size, and activation-view fields without changing the underlying verifier. A deposit identifier, price round, venue request, Reclaim identifier, or MPC request digest occupies `requestID`, as appropriate.

4.8 Non-interference limits

Bounded queues, proof-size and verification-cost limits, per-service worker pools, and circuit breakers prevent external work from exhausting consensus resources. Vote and new-view traffic receives priority. A service may stop without stopping the others, and every activation or disablement takes effect through a versioned deterministic policy at a finalized view.

5 Chained Change-Log Hash

5.1 Construction

The chained change-log hash (CLH) is a compact fingerprint of finalized execution. A recorder wraps the database batch and observes every finalized state write. Let $M_B = (m_1, \ldots, m_k)$ denote the exact mutation sequence emitted while executing block B. CLH-v1 encodes the operations as

$$\mathsf{enc}_1(\mathsf{Put}(K, V)) = \mathtt{P} \parallel K \parallel V, \tag{3}$$
$$\mathsf{enc}_1(\mathsf{Delete}(K)) = \mathtt{D} \parallel K, \tag{4}$$
$$\mathsf{enc}_1(\mathsf{DeleteRange}(K_0, K_1)) = \mathtt{R} \parallel K_0 \parallel K_1. \tag{5}$$

The per-block finalized fingerprint and chain value are

$$D_B = \mathsf{SHA256}(\mathsf{enc}_1(m_1) \parallel \cdots \parallel \mathsf{enc}_1(m_k)), \tag{6}$$
$$C_B = \mathsf{SHA256}(C_{\mathrm{prev}} \parallel D_B). \tag{7}$$

Algorithm 7 CLH-v1 finalization and comparison

procedure FINALIZEWITHCLH(B, C_{prev})
  reset SHA-256 recorder
  for all ordered engine mutations m from B **do**
    apply m to batch and append $\mathsf{enc}_1(m)$ to recorder
  $D_B \leftarrow \mathsf{RecorderHashAndReset}()$
  atomically persist block metadata and commit batch
  $C_B \leftarrow \mathsf{SHA256}(C_{\text{prev}} \parallel D_B)$
  return (D_B, C_B)
procedure CHECKPROPOSALCLH(P)
  if $P.\mathsf{finalizedView} = \mathsf{localFinalizedView}$ **then**
Require: $P.C = \mathsf{SHA256}(C_{\text{local}} \parallel D_{\text{local}})$

At the end of the block, the recorder returns D_B and resets its running state. The implementation assigns D_B to the block, persists finalized metadata, and commits the storage batch in finalization order. The next proposal carries C_B together with the latest finalized view. A validator that has finalized the same view recomputes Equation (7). On a mismatch it raises an alert and stops before casting another vote.

5.2 Security properties

Lemma 5.1 (Deterministic CLH)**.** *Under Assumption 2.1, honest validators starting with the same C_{prev} and finalizing the same block compute identical D_B and C_B.*

Proof. Deterministic execution yields the same ordered mutation byte transcript. SHA-256 is deterministic, so Equation (6) yields the same D_B; equal inputs to Equation (7) yield the same C_B. □

Theorem 5.2 (Byte-transcript divergence detection)**.** *Let two replicas share C_{prev} and hash distinct CLH-v1 byte transcripts $T \neq T'$. If they produce the same C_B, then they have found a collision in SHA-256 at the block or chain step.*

Proof. If $\mathsf{SHA256}(T) \neq \mathsf{SHA256}(T')$, equality of the two C_B values is a SHA-256 collision on distinct $C_{\text{prev}} \parallel D$ inputs. Otherwise, $T \neq T'$ and equal D already form a collision. □

The theorem concerns encoded byte transcripts. CLH-v1 does not place explicit length prefixes between K, V, K_0, and K_1. If a storage schema admits more than one parsing, different semantic operation sequences can therefore yield the same transcript without breaking SHA-256. Fixed-width or self-delimiting key schemas avoid many such cases; CLH-v2 removes the ambiguity at the encoding layer.

Corollary 5.3 (History sensitivity)**.** *If one finalized block produces a different D_B and no SHA-256 collision occurs, every later chain value differs until a checkpoint explicitly resets the anchor.*

History sensitivity supports replica alarms, replay comparison, binary search for the first divergent block, and incident analysis. Consequently, replicas with the same current state can have different CLH values if they reached that state through different mutation histories.

5.3 Cost

Let k be the number of storage mutations and $b = \sum_j |\mathsf{enc}_1(m_j)|$ the number of encoded bytes. The recorder performs $O(k + b)$ work, maintains constant-sized hash state, and produces one 32-byte value per block. Its cost is independent of the n untouched state keys. Repeated writes, delete/reinsert sequences, and range deletions remain in the transcript, so the work follows executed mutations rather than distinct final keys.

Unlike a Merkle or Verkle state tree, CLH requires no path rehashing for each mutation. In exchange, it supplies neither membership proofs nor an authenticated snapshot for a joining node. The two commitments serve different purposes: CLH continuously compares execution histories, while a periodic authenticated state root supports bootstrap and light-client verification.

5.4 Atomicity and crash recovery

The desired invariant is

$$\mathsf{PersistedTip} = B \iff \mathsf{State} = S_B, \tag{8}$$

$$\mathsf{PersistedTip} = B \iff \mathsf{RecordedD} = D_B. \tag{9}$$

Finalized-block metadata is prepared before the state batch commits, so durable consistency depends on the batching and write-order guarantees of the storage engine. Crash-injection tests cover the boundaries between transaction execution, finalized-block persistence, batch commit, tip update, event publication, and pruning. During recovery, the node can either recompute the last block or consult a small write-ahead record to distinguish an interrupted commit from a genuine CLH mismatch.

5.5 CLH-v2 format and migration

CLH-v2 uses an injective encoding and binds the digest to its consensus context:

$$\begin{aligned} D_B^{(2)} = H(&\texttt{DRACO/CLH/BLOCK/V2} \\ &\parallel \mathsf{u64}(\mathsf{chainID}) \parallel \mathsf{u64}(v(B)) \\ &\parallel H(B) \parallel \mathsf{u32}(k) \\ &\parallel \mathsf{LP}(m_1) \parallel \cdots \parallel \mathsf{LP}(m_k)), \end{aligned} \tag{10}$$

$$\begin{aligned} C_B^{(2)} = H(&\texttt{DRACO/CLH/CHAIN/V2} \\ &\parallel C_{\text{prev}}^{(2)} \parallel D_B^{(2)}), \end{aligned} \tag{11}$$

Every variable field is length prefixed, and integer widths and byte order are fixed. At a versioned activation view, the chain seeds $C^{(2)}$ from the last CLH-v1 value and an authenticated checkpoint root. Blocks carry both versions during a bounded compatibility window. Deterministic test vectors include empty blocks, repeated writes, zero-length values, deletions, range boundaries, maximum lengths, and malformed inputs.

An optional *semantic-difference* commitment can collapse repeated writes to the final value of each key and then sort the keys canonically. It remains a separately versioned object because final-difference semantics discard information that the ordered transcript preserves.

6 Applications, Routing, and Incentives

6.1 Validators as accountable providers

Validator service loops support market data, payment connectivity, compliance evidence, cross-chain observation, and confidential computation alongside transaction processing. Consensus stake provides identity and a slashable bond. Access rights, licensing, and evidence quality remain specific to the service, so provider eligibility is defined separately for each service:

$$\begin{aligned}\mathsf{Eligible}_s(p) = \mathsf{Active}(p) &\wedge b_p \geq b_s^{\min} \\ &\wedge \mathsf{Capabilities}_p \supseteq C_s \\ &\wedge \mathsf{Policy}_s(p). \end{aligned} \tag{12}$$

Capabilities include the relevant jurisdictions, currencies, venues, proof systems, hardware, source-chain clients, and legal agreements. A deterministic registry records public capabilities and bonds. Routers may rank eligible providers off chain, while the execution rule enforces the same eligibility predicate at every replica.

6.2 Provider scoring and selection

For request r, a router may compute

$$S(p, r) = \alpha A_p + \beta Q_p - \gamma \widehat{L}_p - \eta F_p - \zeta R_p, \tag{13}$$

where A denotes availability, Q evidence quality, $\widehat{L}$ estimated latency, F the quoted fee, and R correlated-source or concentration risk. The score is advisory and, where possible, computed from finalized observations. A requester may select the highest-ranked provider, sample from the top k, or divide work across independent providers. On-chain routing makes selection deterministic but also makes the score an object of strategic manipulation and requires every replica to possess the same routing data.

A request escrows a maximum fee f_r and binds the provider, policy, deadline, anchor, nonce, and evidence type. A valid result submitted before the deadline settles the agreed fee. Expiry follows the service refund policy, while cancellation after irreversible external work may compensate the provider. Separating proof-generation and endpoint costs from a success bonus allocates upstream failures explicitly rather than hiding them in a single fee.

6.3 Deterrence condition

Let G_p be a provider's maximum gain from a provably invalid or replayed result, q_p the probability that admissible evidence exposes the fault, B_p slashable service bond, L_p lost future service profit, and C_p attack cost. A risk-neutral one-shot deviation is deterred if

$$q_p(B_p + L_p) + C_p > G_p. \tag{14}$$

Equation (14) gives a one-shot deterrence condition. The gain G_p may include derivatives, bribes, regulatory avoidance, or harm to a competitor. The detection probability q_p is small when a failure reflects poor service rather than attributable equivocation. Slashing is therefore reserved for objective evidence, such as conflicting signed reports for one round, invalid proof signatures, reuse of a processed event, or violation of an action bound. Disputes over source truthfulness or discretionary customer decisions require insurance, arbitration, or legal recourse.

Proposition 6.1 (Truthful service under enforceable evidence)**.** *If every profitable deviation produces objective evidence with probability q_p, all such evidence is finalized and slashed, participants are risk neutral, and Equation* (14) *holds for every eligible provider, then honest service is a strict best response in the one-shot service game.*

Proof. Honest service yields the baseline payoff. Any deviation adds at most G_p and incurs expected loss $q_p(B_p + L_p) + C_p > G_p$, so its expected payoff is strictly lower. □

The proposition relies on attributable evidence and reliable enforcement. Collusion, censorship, incomplete evidence, wealth constraints, and retaliation in a repeated game can change the result.

6.4 Running and near-term applications

Markets. The price-feed service supplies index and collateral prices, while venue-equity actions reconcile external positions for risk and margin. The same admission path accommodates reference rates, funding curves, proof-of-reserves statements, and liquidation acknowledgments.

Payments and neobanking. Ethereum receipts and Reclaim-backed Bridge.xyz synchronization provide distinct evidence formats for value movement and account state. The service model also covers bank-account verification, payment authorization, chargeback-aware settlement, fiat ramps, invoice status, and treasury reconciliation. Each application defines its own economic-finality rule; an authenticated authorization and an irrevocable cash transfer have different settlement semantics.

Identity and compliance. Selective zkTLS claims can attest customer status or an approved endorsement while redacting unrelated response fields. Jurisdiction, consent, retention, sanctions screening, and rights of correction remain obligations of the surrounding legal system.

Cross-chain and real-world assets. Light-client proofs strengthen receipt admission, and custodians or transfer agents can publish proof-carrying asset and captable updates. The legal enforceability of the issuer's records remains part of the asset model.

Confidential computation. Validators or bonded providers can supply TEE/MPC attestations, model-inference receipts, or private matching outputs. Their evidence binds the program or model version, inputs, hardware or proof root, and output commitment. Application policy remains responsible for judging model quality and the meaning of the output.

6.5 Concentration and governance

Routing solely by stake or historical latency concentrates work and reduces observational independence. The registry therefore exposes source and provider correlation, supports per-provider share caps and requester overrides, and keeps service bonds distinct from consensus stake. Provider admission, verifier-key rotation, emergency disablement, and fee changes activate at future finalized views. A pending request retains the policy under which it was accepted unless an explicit emergency rule refunds it.

7 Correctness and Security Analysis

7.1 Quorum intersection

Lemma 7.1 (Stake intersection)**.** *For any two quorums Q_1, Q_2 with weight at least $2W/3$, $w(Q_1 \cap Q_2) \geq W/3$.*

Proof. $w(Q_1 \cap Q_2) = w(Q_1) + w(Q_2) - w(Q_1 \cup Q_2) \geq 4W/3 - W = W/3$. □

Because Byzantine weight is strictly below $W/3$, the intersection contains positive honest weight.

Lemma 7.2 (Unique honest certification per view)**.** *If every honest validator signs at most one block in a view and Byzantine weight is below $W/3$, two conflicting blocks cannot both obtain valid QCs in the same view.*

Proof. Two QCs intersect in at least $W/3$ weight. Since Byzantine weight is smaller, an honest validator lies in the intersection and would have signed both conflicting blocks in one view, contradicting the persistent last-vote rule. □

7.2 Consensus safety

Theorem 7.3 (Conditional finality safety)**.** *Assume signature unforgeability, epoch-consistent stake accounting, parent–QC consistency, persistent one-vote-per-view behavior, correct QC/AggQC validation, and the two-chain safe-extension rule. Then two honest validators cannot finalize conflicting blocks.*

Proof sketch. Suppose certified consecutive links commit block A and a conflicting block A'. Choose the earliest certified block on one branch whose view exceeds the committing QC on the other. Standard two-chain HotStuff safety [11, 7] requires that a quorum crossing this boundary contains honest stake whose safe state is at least the first committed branch. Such honest stake cannot vote for a conflicting proposal unless it carries valid higher aggregate timeout evidence that preserves the highest QC; AggQC validation and parent–QC consistency force that evidence to extend the certified safe branch. Otherwise the two conflicting certifications at the boundary require an honest double vote by Lemma 7.2. Both cases contradict the vote and aggregate-certificate rules. Ancestor finalization therefore preserves a single prefix. □

The proof depends on the implementation preserving proposal admission, certificate verification, parent–QC consistency, the durable last-vote rule, correct block-forest traversal, and epoch-consistent validator-set transitions. Section 10 traces these obligations through the executable model and the implementation tests.

7.3 Liveness

Theorem 7.4 (Post-GST progress)**.** *Assume that after GST honest messages and required block data arrive within Δ, deterministic validation completes within τ, the view timeout eventually exceeds $2\Delta + \tau$, validator-set state is consistent, and the weighted schedule elects honest leaders infinitely often. Then valid pending transactions are eventually included and the finalized chain grows.*

Proof sketch. Quorum timeout evidence brings honest validators to monotonically increasing views and propagates the highest QC. In a sufficiently long view with an honest leader, its proposal extends the highest valid QC,

reaches a quorum, and forms the next QC. Repeating for the later honest QC-forming views exposes two consecutive certified links, causing finalization. Retransmission eventually presents the transaction to such a leader. □

The 2 s pacemaker cap defines the deployed timing range. A partition that lasts longer can stop progress, although the quorum and voting assumptions continue to preserve safety.

7.4 Service admission

Theorem 7.5 (Deterministic service convergence)**.** *If an action verifier is deterministic, all referenced evidence bytes are available, and honest replicas execute the same action from the same parent state, then every honest replica returns the same admission result and state transition.*

Proof. All verifier inputs and state reads are equal. Determinism yields the same boolean and output; deterministic execution and atomic state updates yield the same resulting state. □

Theorem 7.6 (At-most-once effect)**.** *Let each service action carry a globally unique replay key, and let successful execution atomically insert that key before any later transaction can observe state. Then no finalized history applies the corresponding external effect more than once.*

Proof. The first successful transition records the key. Every later replay is ordered after it and fails the absence predicate. Consensus safety prevents an alternative finalized ordering that omits the first record. □

A deposit replay key contains the source chain and the unique event or log identity. Reclaim synchronization uses the action nonce or deposit identifier. When several applications may legitimately refer to the same external fact, the proof identifier by itself is not a sufficient application-level replay key.

7.5 Side-loop threshold integrity

Any report signed by at least $2W/3$ stake contains honest stake whose local acceptance predicate succeeded, provided Byzantine stake remains below $W/3$. That statement has a different external premise for each service. Price robustness relies on independent sources and robust aggregation; deposit integrity on source-chain and RPC finality; venue equity on the venue and the accepted tolerance; and zkTLS admission on Reclaim attestorship, TLS provenance, and the external origin.

Threshold ECDSA and consensus use different threshold domains. In the deployed three-party, threshold-one configuration, two key-share holders can complete a signature even when they control less than $2W/3$ consensus stake. Each honest MPC party must therefore sign only a request digest authorized by deterministic multi-venue state, and the external contract must verify the expected threshold public key. The BFT fault bound does not substitute for a separate share-compromise analysis.

7.6 Replay, freshness, and domain separation

A complete admission message binds the chain ID, service and action type, policy version, request or round, external origin, anchor, output, nonce, and expiration. These fields prevent a valid signature from being reused across chains, services, policy versions, or stale contexts. Existing services carry the bindings in their native structures; the common envelope in Section 4 makes them uniform and pairs them with explicit lower and upper timestamp checks.

7.7 Denial of service and equivocation

Admission orders checks by cost: malformed signatures and structural errors are rejected before JSON parsing, proof verification, or external lookups. Evidence values and collections have explicit size bounds. A Byzantine service leader may withhold its round until the service elects another leader, but cannot modify consensus state without a valid action. A Byzantine consensus leader may censor service actions during its view; recurring honest leaders restore inclusion under the liveness assumptions.

Slashing for conflicting service reports requires unambiguous round and domain identifiers. Two price reports from different observation times, or two venue-equity reports within the accepted tolerance, do not necessarily constitute equivocation. The evidence rule encodes this distinction so that ordinary variation in external observations is not treated as a slashable fault.

7.8 CLH security boundary

CLH values are comparable only when they are anchored at the same finalized view; a lagging replica may correctly report an older value. The comparison identifies divergent histories but does not select the correct replica, authenticate a downloaded snapshot, or supply a proof for an individual key. Detailed mutation logs help locate the divergence and are kept behind access controls or redacted when they contain customer data.

8 Implementation Architecture

8.1 Component organization

The Go implementation separates consensus, deterministic execution, storage, networking, telemetry, and the validator service loops. Table 2 maps each protocol mechanism to the component that enforces it.

Table 2: Protocol-to-implementation mapping.

Mechanism	Components	Implementation role
Vote and proposal admission	Consensus and certificate validation	Vote rule, QC/AggQC checks, proposer validation, transaction-root verification, CLH comparison, and durable last-vote state.
Pacemaker and timeouts	Pacemaker and message handling	View state, adaptive duration, TC/AggQC formation, new-view routing, and high-QC advancement.
Finalization and execution	Block forest, execution engine, and storage	Parent/QC linkage, two-link finalization, ordered execution, atomic storage commit, and pruning.
Stake and leaders	Validator-set management	Exact two-thirds predicate, signer membership, validator updates, and weighted deterministic epochs.
CLH-v1	Storage recorder and finalization	Ordered `P/D/R` mutation hashing, per-block fingerprints, chained hashes, and divergence alerts.
Price-feed service	Source adapters and validator service loop	Source collection, filtering, weighted aggregation, validator reports, and quorum-approved price actions.
Deposit and venue equity	External-state service loops	Receipt/equity validation, replay protection, tolerance checks, quorum formation, and deterministic actions.
Reclaim zkTLS	Neobank verifier and deterministic engine	Claim reconstruction, EIP-191 attestor recovery, origin and route policy, freshness, response predicates, and replay checks.
Threshold ECDSA	MPC service and peer transport	Authorized request lookup, interactive signing, session management, retry/deadline handling, and signature publication.
Telemetry	Consensus and service metrics	Per-stage timing, view/QC/TC state, timeout activity, block processing, service latency, and error counters.

8.2 Consensus implementation logic

Listing 1 summarizes the Fast-HotStuff vote rule. Proposal admission verifies aggregate timeout evidence and the high QC before the block forest checks that the proposal parent is the certified block.

Listing 1: Implementation-oriented vote-rule pseudocode.

```
VoteRule(proposal):
    if proposal.aggregateQC is present:
        return BlockStore.Contains(proposal.QC.blockHash)
    return proposal.view >= LocalView
       and proposal.view == proposal.QC.view + 1
```

The active validator set uses the exact quorum predicate `weight*3 >= totalStake*2`. Proposal processing also checks the leader identity, proposer signature, certificate, transaction root, durable last vote, timestamp monotonicity, and parent $= QC$.blockHash. A vote is sent to the leader of the next view, which may propose as soon as it assembles the QC.

8.3 CLH implementation logic

Listing 2 gives the CLH-v1 mutation recorder and chain update in implementation-oriented pseudocode.

Listing 2: Implementation-oriented CLH-v1 pseudocode.

```
Put(key, value):
    Recorder.Write('P' || key || value)
    Batch.Put(key, value)

Delete(key):
    Recorder.Write('D' || key)
    Batch.Delete(key)

ChainedCLH(previous, finalized):
    return SHA256(previous || finalized)
```

8.4 zkTLS implementation logic

Listing 3 gives the common Reclaim admission checks. The action verifier additionally binds the allowed Bridge.xyz route and the required response fields for customer, account, liquidation, transfer, and deposit operations.

Listing 3: Implementation-oriented Reclaim verification pseudocode.

```
VerifyReclaim(proof, action, blockTime):
    require proof.id == ComputeIdentifier(proof)
    signers = RecoverEIP191Signers(proof)
    require signers intersect RegisteredAttestors
    require AllowedBridgeOrigin(proof.requestURL)
    require AllowedRoute(proof.requestURL, action.type)
    require proof.timestamp >= blockTime - 3 minutes
    require ResponsePredicates(proof, action)
    require ReplayKey(action) is unused
```

8.5 Production observability

Consensus telemetry records proposals, votes, timeouts, QCs, TCs, and the current and finalized views. Timers cover cryptographic verification, block validation, deterministic execution, storage commit, and transaction processing. Each finalization records the block age, transaction count, view, and CLH. Service metrics add source latency, round completion, proof rejection, and replay re-

jection. Section 9 uses these measurements to separate consensus latency from external-service latency.

8.6 Production architecture

Consensus and the validator services share authenticated networking, validator identity, deterministic transaction admission, and telemetry. Their queues and worker pools remain separate, and consensus messages take priority over source polling, proof processing, and threshold-signing sessions. Durable protocol state includes the current view, last voted view, high QC, finalized tip, validator set, replay identifiers, and authorized service requests. An external service can therefore pause and recover while block production continues.

Protocol upgrades take effect at predetermined finalized views. Certificate rules, validator membership, CLH encoding, proof policy, and replay domains carry explicit version numbers and activation points. Nodes verify the active version before voting, while the compatibility window permits a rolling software upgrade before activation.

9 Performance Evaluation

9.1 Methodology

We evaluate the system using four forms of evidence:

1. controlled load tests with four validators;
2. configured protocol and service timers;
3. operational measurements collected since January 2026; and
4. the latency bound in Equation (1).

The controlled experiments establish a reproducible throughput and latency baseline. Operational measurements cover the wider production workload. The configured timers and analytical bound explain how network delay, validation, execution, and service work enter the observed latency.

9.2 Four-node controlled load

The controlled baseline used four validators and ran for approximately 123 seconds at offered loads of 50 and 100 transactions/s. Client latency was measured from submission to the committed response. Table 3 reports throughput, completion rate, and latency percentiles.

Neither run recorded a safety violation, liveness violation, or view change. At experiment termination, eight transactions remained uncommitted in the 50 transactions/s run and fourteen in the 100 transactions/s run. The two runs transferred 5.9 MB and 11.7 MB of network traffic, respectively. Recorded memory ranged from approximately 3.8–6.6 MB in the first run and 4.1–7.1 MB in the second.

The observed 50–150 ms range reflects the local test environment and the 150 ms finalization pacing policy. Multi-region delay, failed leaders, storage saturation, larger batches, and proof-heavy service actions exercise different portions of the latency budget and require separate controlled measurements.

9.3 Three-validator protocol microbenchmark

A separate 71-second loopback microbenchmark used three validators, 14-byte commands, one transaction per block, a 20 ms test timeout, and no batching. It processed 328,312 transactions at 4,624 transactions/s. The estimated p50 was 0.1 ms, the estimated p99 was 18 ms, and the average view duration was 0.225 ms.

This microbenchmark isolates the protocol loop from the production execution workload. With three validators it does not provide one-fault Byzantine tolerance under $n \geq 3f + 1$, and its percentiles are derived estimates. We therefore use the four-validator experiment as the controlled BFT baseline.

9.4 Configured and analytical latency

Table 4 records protocol and service timing parameters that affect user-visible behavior.

After a proposal is received, the consensus fast path comprises two QC-forming network rounds together with validation and execution. A service request has the broader latency decomposition

$$L_s = L_{\text{observe}} + L_{\text{proof}} + L_{\text{sideQC}} + L_{\text{queue}} + L_{\text{BFT}} + L_{\text{notify}}. \tag{15}$$

For a streamed price, L_{observe} may be only a cache read. A fresh zkTLS session or venue request can instead dominate this term. Because the work runs in a service loop, it does not extend L_{BFT} for unrelated transactions, but it remains part of the requesting client's end-to-end latency.

9.5 Production operation

DracoBFT has processed production traffic since January 2026. Normal-path blocks finalize in less than one second. The telemetry described in Section 8 measures proposal handling, QC formation, finalization, deterministic execution, storage, and the service loops. Table 3 fixes the workload and environment for comparison, whereas production measurements span a broader workload and network setting.

Table 3: Controlled four-validator baseline. Percentiles report client-observed transaction latency.

Offered (tx/s)	Sent	Committed	Success (%)	Mean TPS	Peak TPS	p50 (ms)	p95 (ms)	p99 (ms)	p99.9 (ms)
50	6,000	5,992	99.8667	48.703	58.444	100.017	145.009	149.054	149.955
100	12,000	11,986	99.8833	97.404	116.885	100.029	145.053	149.042	149.964

Table 4: Production protocol and service timing parameters.

Path	Timing
Finalization pacing floor	150 ms
Post-warm-up slow-block warning	600 ms
Adaptive consensus view interval	1–2 s
Price production interval	2 s
USDC price interval	60 s
Venue-equity interval	5 s
Venue fetch timeout	5 s
Threshold-ECDSA signing deadline	30 s
Reclaim proof maximum past age	3 min

9.6 Evaluation extensions

The next controlled campaign covers 4, 7, 10, and 16 validators distributed across several WAN regions. It varies batch size, transaction mix, state hotness, and evidence type, and injects failed leaders, equivocation, delay, partitions, restarts, storage faults, malformed proofs, source disagreement, and CLH mismatches. Independent runs will isolate the costs of CLH and the service loops and compare CLH with authenticated-state commitments. For zkTLS, the measurements will separate proof construction, claim size, attestor time, verification time, rejection cost, and finalization latency.

10 Specification, Tests, and Assurance

10.1 Assurance methodology

The assurance case draws on four distinct sources:

Protocol proofs.
Sections 3 and 7 establish conditional properties of the stake-weighted protocol.

Executable specification.
Quint modules define the transition system, named properties, and adversarial consensus and service scenarios.

Implementation tests.
Unit and integration tests exercise consensus objects, deterministic execution, storage, service behavior, and wire and storage codecs.

Runtime assertions.
Certificate, parent, root, timestamp, replay, proof, and CLH checks reject an invalid transition or halt on detected divergence and emit an operational signal.

Each source covers a different failure class. Protocol proofs depend on stated assumptions, the executable model searches a bounded state space, tests examine concrete code paths, and runtime assertions guard the live transition boundary.

10.2 Quint model

The Quint specification models views, blocks, certificates, finalization, timeouts, pacemaker behavior, validator changes, service events, and Byzantine actions. Nine scenario modules contain 190 direct `run` declarations. Across the full specification there are 217 run declarations and 67 property or helper definitions. The scenarios exercise agreement, chain validity, finalization, view monotonicity, leader validity, timeout recovery, validator changes, catch-up, and Byzantine behavior.

One central invariant requires every proposal to name the block certified by its QC as its parent. The implementation checks this relation before voting and again during block-forest traversal, connecting the abstract safe-extension rule to both proposal admission and finalization.

Quint simulation explores randomized traces; Apalache or TLC provides bounded verification for a stated configuration [9, 10]. The verification matrix in Section 12 covers weighted validators, aggregate timeout certificates, skipped views, reconfiguration, and two-link finality.

10.3 Implementation validation

The Go test suite combines hand-written unit and integration tests with generated codec tests. The hand-written cases cover deterministic execution, validator and certificate types, storage, price-feed aggregation, deposit and venue actions, Reclaim verification, replay protection, and error handling. Generated cases exercise serialization and persistent codecs across protocol message types.

Quint exercises the abstract state machine. The Go tests add cryptography, persistence, concurrency, networking, and external-service validation. The adversar-

Table 5: Implementation validation areas.

Area	Coverage
Consensus objects	Blocks, votes, QCs, timeout evidence, stake thresholds, and validator membership.
Deterministic execution	Transaction validation, state transitions, replay identifiers, and event output.
Validator services	Price feeds, deposits, venue equity, threshold actions, and leader failover.
Reclaim verification	Claim reconstruction, attestors, origins, routes, fields, freshness, and replay.
Storage and CLH	Ordered mutations, batch behavior, persistence, recovery, and chain comparison.
Serialization	Generated round-trip and compatibility tests for wire and storage formats.

ial matrix targets the boundaries where these concerns meet.

10.4 Certificate and adversarial validation

The validation matrix constructs certificates from deterministic test keys and checks both accepting and rejecting boundaries:

1. exactly $2W/3$ stake succeeds; one weight unit below fails; duplicate signers, nonmembers, wrong messages, and invalid aggregate signatures fail;

2. two conflicting QCs in one view cannot be assembled without an honest double-sign; persistent `lastVote` survives restart and torn-write injection;

3. AggQC rejects mixed views, a forged highest QC, missing timeout signers, wrong validator epoch, and a batch signature valid for different timeout bytes;

4. proposals reject parent/QC mismatch, missing ancestor, wrong Merkle root, nonleader proposer, regressing timestamp, stale view, and CLH mismatch;

5. finalization occurs only after the exact two certified links and finalizes ancestors in order across skipped views and forks;

6. reconfiguration tests straddle activation views and prohibit a certificate assembled from old-plus-new stake;

7. side loops test threshold edges, report equivocation, stale rounds, duplicate deposit/log IDs, $\pm 0.5\%$ venue boundaries, price-source correlation, and leader failover;

8. Reclaim tests mutate provider, parameters, context, signature recovery ID, attestor, host, route, method, response fields, amount, nonce, past/future time, and processed deposit ID;

9. MPC tests cover invalid/absent request hashes, duplicate sessions, wrong party mapping, late messages, retries, timeout, malformed shares, partial participation, signature verification, share rotation, and restart;

10. CLH tests cover operation order, repeated writes, empty values, prefix aliases, delete ranges, crash points, recovery, comparison at unequal finalized views, and v1–v2 activation.

Every negative test checks the rejection reason and verifies that state is unchanged. Fuzz targets include certificate decoding, JSON and canonical context, transaction codecs, mutation framing, and block-forest operations. Race-enabled builds, deterministic scheduling, packet duplication and reordering, and long-running randomized clusters exercise behavior outside the pure state model.

10.5 Specification-to-implementation conformance

Quint and the implementation share the following trace vocabulary:

$$\langle \mathsf{Propose}, \mathsf{Vote}, \mathsf{QC}, \mathsf{Timeout}, \\ \mathsf{AggQC}, \mathsf{AdvanceView}, \mathsf{Finalize}, \mathsf{Reconfigure} \rangle. \tag{16}$$

The implementation emits a deterministic trace projection containing the validator, view, block and QC digests, signer weight, high-QC view, and finalized tip. A model-based test generates a Quint trace, drives a cluster with the corresponding messages, and compares projected state after each action. The same projection checks operational traces for monotonic last-vote and finalized-view values, existing QC targets, exact quorum thresholds, and a single finalized prefix.

10.6 Evaluation manifest

The evaluation manifest records the compiler, Quint and dependency versions, build parameters, genesis stake, validator topology, cryptographic scheme, workload, metric definitions, and plotting procedure. Synthetic CLH vectors, certificate fixtures, and Reclaim fixtures provide deterministic inputs without customer data or service credentials.

11 Related Work

11.1 BFT consensus

PBFT established practical Byzantine state-machine replication with quadratic normal-case communication [4]. HotStuff introduced a leader-based chained protocol with linear communication and optimistic responsiveness [20]. Fast-HotStuff reduces optimistic latency and uses aggregate-QC view change [11]; Jolteon develops a two-chain HotStuff variant, and Ditto adds an asynchronous fallback [7]. Tendermint uses locked rounds under partial synchrony [2]. HoneyBadger and Dumbo study asynchronous BFT [14, 8], while Narwhal/Tusk separates data availability from ordering [5]. DracoBFT adopts the Fast-HotStuff/two-chain commit structure and develops its production implementation, validator service-loop composition, CLH integration, and operational analysis.

11.2 Oracles and authenticated external data

Oracle networks aggregate signed provider reports, reducing reliance on one source while retaining assumptions about the committee and its upstream data. Town Crier used trusted hardware to authenticate HTTPS data [21]. DECO proves statements about TLS data without cooperation from the web server or disclosure of the complete transcript [22]. TLSNotary, zkTLS, proxy, MPC, and TEE systems make different privacy, trust, and availability tradeoffs [12]. Reclaim's zkFetch construction packages selectively disclosed HTTP request/response claims for a verifier [16].

DracoBFT integrates Reclaim at the application and consensus boundary. The adapter performs deterministic claim reconstruction, attestor recovery, origin, route, and field checks, freshness and replay validation, and an atomic state transition. Multi-source price aggregation addresses the accuracy of a market observation, whereas zkTLS addresses the provenance of selected API content. Pyth and exchange adapters supply the observations [15]; a TLS proof can authenticate their origin but does not make correlated sources independent.

11.3 Cross-chain verification

Cross-chain systems use multisignature committees, optimistic fraud proofs, embedded light clients, and succinct validity proofs. zkBridge demonstrates efficient proof-based verification of cross-chain state [19]. DracoBFT's deposit service certifies validator checks of a receipt under the configured RPC and source-finality policy. An embedded or succinct light client can strengthen that evidence while preserving the same service-loop interface.

11.4 Authenticated state

Merkle trees provide succinct authenticated membership proofs [13], and Ethereum commits state with a Merkle Patricia Trie [18]. Sparse Merkle trees, IAVL, Verkle trees, and vector commitments make different update and proof tradeoffs. CLH commits an ordered mutation history instead of the current key space. Its $O(k + b)$ recorder cost and 32-byte block value support continuous replica comparison; authenticated checkpoints serve membership queries and trust-minimized bootstrap.

11.5 Incentives and service markets

Consensus security depends on rewards, fees, and incentives outside the protocol. Carlsten et al. analyze instability as block subsidies diminish [3]; Biais et al. study blockchain equilibria [1]; and Roughgarden develops foundations for transaction-fee mechanism design [17]. DracoBFT keeps consensus stake distinct from service bonds and reputation. A conflicting signed report can carry objective slashing evidence, whereas an upstream commercial dispute usually cannot.

11.6 Formal methods for distributed systems

TLA-style executable specifications support transition-level reasoning and counterexample search. Quint provides randomized simulation and bounded checking through Apalache or TLC [9, 10]. In DracoBFT, the model exposed the parent–QC obligation as an explicit invariant. Trace conformance and refinement are then needed to connect the abstract action to every relevant Go execution.

12 Future Work

The production system provides a base for further work on consensus performance, authenticated state, proof verification, service accountability, and mechanized conformance.

12.1 Consensus latency and throughput

One-round fast paths. Protocols with $n \geq 5f + 1$ can tolerate f Byzantine validators while committing optimistically in one round. For DracoBFT, the additional replicas must be weighed against the latency of the current two-certified-link path, including weighted stake and reconfiguration. A dual-mode protocol also requires a common safety proof for fast-path and fallback certificates.

Pipelining and batching. Proposal construction, signature verification, transaction prevalidation, finalized

execution, storage flush, and client notification can proceed through a deeper pipeline. Adaptive blocks can target an execution-time budget per view instead of a fixed transaction count. Batched or parallel signature verification, cache-aware decoding, SIMD hashing, and parallel execution of conflict-free access lists offer additional throughput while retaining a deterministic output order.

Leader unpredictability. A VRF, threshold beacon, or commit–reveal seed bound to finalized state can replace the predictable epoch-seeded PRNG. The design must remain deterministic across stake changes and make progress when a participant withholds randomness.

Pacemaker control. The current 1–2 s clamp is tuned for the deployed network. A multi-region controller could estimate proposal, vote, and execution quantiles separately, use hysteresis, and extend beyond 2 s after severe degradation. Its stability analysis must bound the influence of Byzantine timing samples.

Data availability. Narwhal-style dissemination or erasure-coded blocks can move payload transfer off the leader's QC path. The resulting certificate must bind the exact availability object and guarantee that finalized data remains retrievable for deterministic execution.

12.2 State, CLH, and storage

CLH-v2. The next CLH version applies the domain-separated, length-prefixed format from Section 5 and writes v1 and v2 values through a fixed activation window. Cross-language test vectors and semantic-aliasing fuzz cases will establish a stable encoding. Evaluation will compare the forensic value of the ordered transcript with a second, sorted final-difference commitment.

Authenticated checkpoints. A periodic sparse-Merkle, Verkle, or vector-commitment root can accompany CLH. A joining node would verify a snapshot against the checkpoint and replay only the later CLH-linked blocks. Building the tree incrementally in the background keeps checkpoint work within a fixed consensus resource budget.

Pruning and recovery. Retention policy must cover blocks, transaction payloads, mutation transcripts, proof bytes, and external observations. State synchronization can verify chunk commitments, the checkpoint root, validator-set history, and the CLH suffix. Recovery testing will include torn batches, full disks, corrupted write-ahead logs, and pruning concurrent with finalization.

Cross-implementation execution. A second deterministic engine or replay verifier in another language can consume finalized blocks and compare transaction roots, events, checkpoints, and CLH values. Requiring agreement before release would reduce common-mode implementation and runtime risk.

12.3 zkTLS and proof systems

End-to-end proof verification. The current adapter validates Reclaim attestor-signed claims. A direct mode could instead verify a succinct proof of the TLS/zkFetch statement, or an aggregate proof rooted in an on-chain commitment to the attestor set. In either mode, transcript provenance and the authority of the source remain separate properties.

Recursive aggregation. Many HTTP claims can be folded into one recursive proof whose public accumulator commits request identifiers, origins, policy versions, timestamps, and outputs. This reduces replicated verification work at the cost of proof availability and possible batch censorship. Each requester would therefore receive an inclusion proof.

Streaming sources. WebSocket frames and authenticated webhooks require a session proof rather than an isolated request/response proof. Such a session needs sequence numbers, gap proofs, reconnect semantics, origin and key rotation, and a rule for mutable server-side state. Price streams and payment-status updates are immediate applications.

Privacy. Typed predicates over canonical JSON can replace substring matches and disclose only the fields used by the action. Larger auxiliary values can remain encrypted, with metadata leakage measured separately. A complete selective-disclosure policy also covers revocation, consent, retention, and correction of a false upstream record without rewriting finalized history.

Verifier governance. A versioned registry can bind the provider schema, verification key or attestor set, allowed origins, public/private parameter split, freshness interval, cost bound, and activation and retirement views. Emergency disablement must refund pending work through a deterministic rule. The next policy version also adds an upper timestamp bound to the deployed lower freshness check.

12.4 Cross-chain and payment evidence

Ethereum sync-committee/finality proofs or succinct light-client updates can replace correlated RPC receipt checks. The admission proof would bind receipt inclusion, log index, emitting contract, chain ID, and finalized

header. A versioned client registry extends the same interface to other chains. Bridge accounting then needs a conservation invariant across mint, burn, disputed deposit, refund, and routing actions.

For fiat payments, authorization, capture, settlement, reversal, and chargeback are distinct events rather than one Boolean status. Provider bonds or insurance can cover the interval between authenticated API evidence and economic finality. A corresponding asset-liability invariant must include external balances and pending transfers.

12.5 Service markets and accountability

Sealed-bid and posted-price selection can incorporate requester constraints on jurisdiction, source independence, proof type, deadline, and maximum fee. Reputation should be derived from finalized objective outcomes with confidence intervals and decay. The design must resist Sybil identities without making the largest-stake validator the permanent winner. Correlation-aware routing further avoids choosing several providers backed by the same cloud, RPC service, venue, or attestor set.

Service-specific slashing requires a compact evidence format for signed equivocation, an invalid origin claim, or replay. Subjective commercial disputes remain in arbitration or insurance. Extending Proposition 6.1 to repeated games introduces collusion, bribery, malicious requesters, service refusal, and gains external to the protocol.

12.6 Confidential execution

The threshold-ECDSA service can extend beyond venue signing after proactive share refresh, audited DKG ceremonies, hardware-backed custody, dynamic party rotation, session transcripts, request-domain separation, and recovery procedures are in place. Broader MPC and TEE services could then support private matching, risk computation, and credentialed API access. A TEE policy binds the measurement, firmware and trusted computing base, data schema, output, freshness, and revocation status; diversity across TEE vendors reduces common-mode risk. ZK execution removes hardware trust for suitable deterministic circuits, while general MPC protects secrets held by several parties. Each result enters the same deterministic action path used by the existing services.

12.7 Formal verification and testing

Bounded model checking. The verification matrix covers weighted validators, timeout and AggQC behavior, skipped views, reconfiguration, and two-link finality. Each report records validators, views, blocks, Byzantine weight, solver, seed, time and memory limits, and whether exploration completed within the stated bound.

Refinement and conformance. The bidirectional trace harness in Section 10 can continuously compare Go proposal admission, certificate formation, view advancement, finalization, and reconfiguration with the model. Separate models for service replay state and CLH-v2 activation keep their state spaces tractable.

Fault injection. The fault matrix includes Byzantine twins, equivocation, forged or mixed certificates, clock skew, dropped votes, delayed blocks, failed leaders, partition and recovery, restart, disk corruption, slow proof verification, malformed JSON, compromised sources, and CLH mismatches. Minimized counterexample traces become permanent regression cases.

Concurrency and software supply chain. Continued assurance work includes race detection, deterministic scheduling, fuzzing, static analysis, dependency review, reproducible builds, software bills of materials and provenance, secret scanning, key-rotation exercises, and independent review of consensus and cryptography.

12.8 Post-quantum and long-horizon security

A post-quantum migration can begin with versioned hybrid certificates carrying both the current signature and a post-quantum signature. Evaluation must account for key and signature size, aggregate-verification alternatives, block and QC bandwidth, hardware support, and validator key rotation. Historical verification also needs an algorithm registry and deprecation policy so that old validator-set transitions remain verifiable after the original format is retired.

12.9 Evaluation roadmap

Longitudinal evaluation will combine sanitized operational summaries with reproducible benchmark snapshots. Each snapshot records the validator count and regions, stake skew, paper artifact version, block and transaction mix, payload bytes, normal and timeout views, proposal-to-QC and proposal-to-finality percentiles, execution and storage breakdown, service-loop latency, proof size and cost, availability, and incident counts. Signed aggregate records and analysis scripts make comparisons across releases reproducible without exposing customer data or private deployment details.

13 Conclusion

DracoBFT is a stake-weighted, two-chain consensus protocol built around validator services for financial infrastructure. QCs, TCs, and AggQCs support a deterministic weighted leader schedule, adaptive view changes, and finality after two certified links. The price-feed, deposit,

venue-equity, Reclaim zkTLS, and threshold-ECDSA services perform external work outside the consensus event loop and return deterministic, replay-protected actions. CLH commits the ordered mutations of finalized execution, giving replicas a continuous history check at low cost.

Hotstuff Labs has operated DracoBFT in production since January 2026. In the controlled four-validator experiment, an offered load of 100 transactions/s produced 97.404 committed transactions/s, with approximately 100 ms median and 149 ms p99 client-observed latency. The safety and liveness theorems make the network, quorum, persistence, and deterministic-execution assumptions explicit. Quint scenarios, implementation tests, and runtime checks examine the corresponding transitions and adversarial boundaries.

CLH-v2, authenticated checkpoints, light clients, direct and recursive proof verification, privacy-preserving service markets, broader confidential computation, post-quantum certificates, multi-region evaluation, and mechanized conformance form the next stage of the work. The present system shows that external financial services can be integrated at the validator layer while the replicated state machine retains a deterministic critical path.

Competing Interests

The authors are affiliated with Hotstuff Labs, which operates DracoBFT and may benefit from its adoption. Section 9 distinguishes controlled experiments from operational observations.

References

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

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

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

[4] 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.

[5] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: A DAG-based mempool and efficient BFT consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

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

[7] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. *arXiv preprint arXiv:2106.10362*, 2021.

[8] Bingyong 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.

[9] Informal Systems. Quint: An executable specification language. Project documentation and source code, 2026.

[10] Informal Systems. What does quint do? model checker and simulator. Quint documentation, 2026.

[11] Mohammad M. Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-hotstuff: A fast and resilient hotstuff protocol. *arXiv preprint arXiv:2010.11454*, 2020.

[12] Zhongtang Luo, Yanxue Jia, Yaobin Shen, and Aniket Kate. Proxying is enough: Security of proxying in TLS oracles and AEAD context unforgeability. *Cryptology ePrint Archive, Paper 2024/733*, 2024.

[13] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO '87*, pages 369–378, 1988.

[14] 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.

[15] Pyth Data Association. Pyth core price feeds. Official developer documentation, 2026.

[16] Reclaim Protocol. zkfetch: Generate and verify proofs for authenticated http data. Official developer documentation, 2026.

[17] Tim Roughgarden. *Twenty Lectures on Algorithmic Game Theory*. Cambridge University Press, 2016.

[18] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Yellow Paper, 2014.

[19] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. *Cryptology ePrint Archive, Paper 2022/073*, 2022.

[20] 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.

[21] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 270–282, 2016.

[22] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: Liberating web data using decentralized oracles for TLS. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1919–1938, 2020.

A Evaluation and Test Details

A.1 Evaluation configuration

The controlled BFT baseline uses four validators, offered loads of 50 and 100 transactions/s, and runs of approximately 123 seconds. The protocol microbenchmark uses three loopback validators, 14-byte commands, one transaction per block, a 20 ms test timeout, and a 71-second run. The accompanying manifest records the paper artifact version, validator weights, workload parameters, cryptographic configuration, host resources, and metric definitions.

A.2 CLH-v1 test-vector procedure

Given an initial 32-byte value C_0, a test vector lists every operation and the exact hexadecimal bytes of each key and value. The procedure concatenates the operation tag and fields according to CLH-v1, computes $D_1 = \mathsf{SHA256}(T_1)$, and then computes $C_1 = \mathsf{SHA256}(C_0 \,\|\, D_1)$. The vector set contains an empty block, one put, an empty value, one deletion, one range deletion, two operations in reversed order, repeated writes to one key, deletion followed by restoration, keys and values containing tag bytes, and two semantic transcripts with ambiguous v1 field boundaries. CLH-v2 adds chain, view, block digest, field lengths, domain tags, dual-write activation, and malformed-length cases.

A.3 Certificate fixtures

QC fixtures use both four equal-weight validators and a weight vector such as $(40, 30, 20, 10)$ to exercise the exact threshold. Every valid QC, TC, and AggQC has variants with one missing signer, a duplicate entry after decoding, a nonmember, the wrong view or block, an invalid signature, a signer-set/aggregate mismatch, and mixed epochs. Two-link fixtures cover straight chains, skipped views, forks below the finalized tip, parent–QC mismatch, a missing ancestor, and restart before the last vote becomes durable.

A.4 Reclaim fixtures

Reclaim fixtures use synthetic Bridge.xyz-shaped responses and test keys. Cases include valid customer, account, and deposit statements; reordered canonical-context keys; parameters changed after signing; an incorrect provider, owner, or epoch; EIP-191 recovery identifiers 0/1/27/28 and invalid values; untrusted attestors; origin and route-prefix confusion; URL encoding; stale and future timestamps; response-predicate failures; amount, currency, and transaction-digest mismatches; and replayed deposit identifiers. Logging tests confirm that secret options and response bodies are redacted.

A.5 Benchmark records

Each benchmark record contains:

- offered, submitted, and committed transaction counts;
- elapsed time, mean and peak throughput, and success rate;
- p50, p95, p99, and p99.9 transaction latency;
- view changes and detected safety or liveness violations; and
- network traffic, memory samples, and workload configuration.

B Model-Checking Report Template

A bounded verification report states the validator count and weight vector; Byzantine set; maximum views, blocks, timeouts, pending service events, and reconfigurations; symmetry reduction; solver and version; invocation; seed; wall-clock and memory bounds; invariant set; explored-state count; and completion status. Counterexample traces become regression scenarios. Separate reports cover consensus safety, liveness under fairness, service replay, CLH activation, and epoch transitions.

C Operational Release Checklist

Before activating a protocol upgrade, operators reproduce the controlled suite; run certificate, restart, and

fault tests; compare a full replay with the previous version; and confirm identical CLH and checkpoint roots. A nonvoting canary exercises the new version before activation. The release procedure also verifies rollback, key and attestor rotation, and telemetry for QC formation, timeouts, finality, execution, storage, service-loop lag, proof rejection, and CLH mismatch. Consensus-format changes include an explicit compatibility interval and rollback point.

D Ethics and Disclosure

The authors are employees or affiliates of the operator and may benefit from adoption of the system. Operational telemetry is aggregated and excludes customer, trading, identity, credential, and payment data. Tests involving Bridge.xyz, Reclaim, exchanges, and external chains follow authorization and rate-limit requirements. Cryptographic provenance establishes the origin of specified data; legal and regulatory conclusions depend on the relevant jurisdiction and application.