RadiumBFT: Deterministic Byzantine Fault Tolerance with Instant Finality

Radium Labs

Abstract

RadiumBFT is a deterministic consensus protocol that provides *instant finality* without forks or rollbacks. Each block is committed in a single proposal–vote–proof pipeline, enforced by quorum intersection guarantees. RadiumBFT integrates a verifiable **slot pacemaker**, **adaptive timing**, and a **hint-based gossip layer**, achieving one-second block intervals with throughput above 16,000 objects per second. This white paper presents the protocol, algorithms, message flow, state machine, and correctness arguments (safety and liveness).

1 Introduction

Classical BFT protocols like PBFT provide safety but incur high message complexity. HotStuff improves responsiveness but requires multi-phase commits before finality. RadiumBFT advances the state of the art by requiring conflict resolution before commitment, resulting in deterministic (instant) finality. Applications include high-value settlement, satellite/LEO-based ledgers, and geographically topology-aware blockchains.

2 System Model

- **Nodes:** A fixed set of n delegates.
- **Faults:** Up to $f \leq \lfloor (n-1)/3 \rfloor$ Byzantine nodes.
- **Communication:** Authenticated, partially synchronous.
- **Quorum:** Decisions require weighted supermajority $\geq \tau$ where $\tau \in (1/2, 1]$; typically $\tau \geq 2/3$.

3 Core Concepts

- **Round:** Identified by $(\mathit{index}, \mathit{sequence}) = (i, s)$. The block proposed targets $i{+}1$.
- **Leader Schedule:** Deterministic, round-robin with window size w, mapping $(i, s) \mapsto \mathsf{LeaderFor}(i, s)$.
- **Proposal:** Leaders block candidate for $i{+}1$ (extends the committed block at i).
- **Proof:** A leader-produced object for round (i, s) that *contains the aggregated vote objects* collected for the same proposal until the supermajority threshold τ is reached. The proof acts as a certificate *by inclusion of votes* (not a separate quorum-certificate type).
- **Instant Finality:** A block is final once its proof is formed and validated; no rollback.

- **Slot Pacemaker:** A deterministic, verifiable mechanism replacing timers; slots regulate the pace of proposals and ensure all nodes advance consistently.

4 Algorithms

```
Algorithm 1: Leader Proposal
  Input: Current round $R = (i, s)$, parent block $B_i$
  Output: Broadcast proposal
  $B \leftarrow$ CreateBlock($B_i$);
  $P \leftarrow (B, R, \text{signer} = \mathsf{LeaderFor}(i, s))$;
  Broadcast(Proposal, $P$);
```

```
Algorithm 2: Replica Handling Proposal
  Input: Proposal $P = (B, R = (i, s), \text{signer})$
  if $\mathit{LeaderFor}(i, s) = \mathit{signer}$ and Valid($P$) and SafeToVote($R$) then
      $V \leftarrow (\text{type=proposal}, \text{hash} = B.\text{hash}, R, \text{signer=self}, \text{weight} = w_{\text{self}})$;
      SendVote($V$, $\mathsf{LeaderFor}(i, s)$);
```

```
Algorithm 3: Vote Aggregation and Proof Formation (at leader for $(i, s)$)
  Input: Vote $V$ for round $R = (i, s)$
  Add $V$ to $\mathsf{VoteSet}[R]$;   $\mathsf{Weight}[R] \leftarrow \mathsf{Weight}[R] + V.\text{weight}$;
  if $\mathit{Weight}[R] \geq \tau$ then
      $PF \leftarrow (\text{round} = R, \text{votes} = \mathsf{VoteSet}[R], \text{signer} = \mathsf{LeaderFor}(i, s))$;
      Broadcast(Proof, $PF$);
```

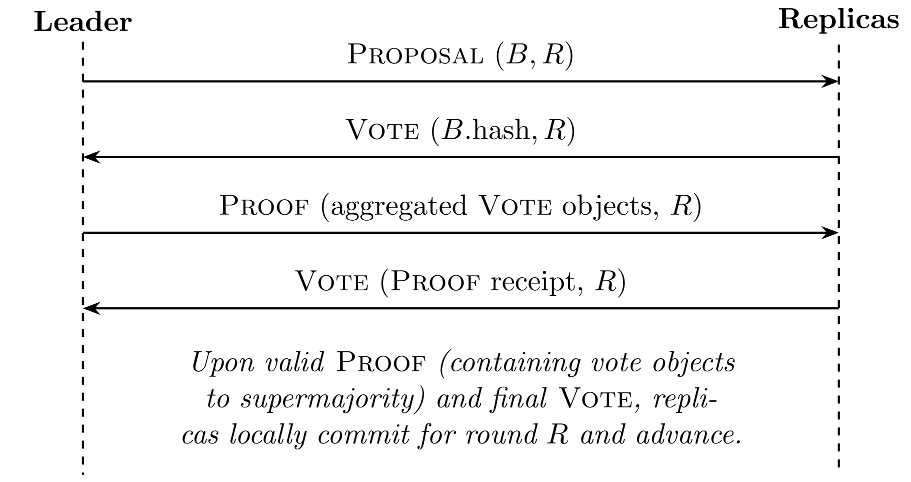
```
Algorithm 4: Replica Handling Proof
  Input: Proof $PF$ for round $R = (i, s)$ containing aggregated vote objects
  if $\mathit{LeaderFor}(i, s) = PF.\mathit{signer}$ and Valid($PF$) and SafeToVote($R$) then
      $V \leftarrow (\text{type=proof}, \text{hash} = PF.\text{digest}, R, \text{signer=self}, \text{weight} = w_{\text{self}})$;
      Broadcast($V$);
```

```
Algorithm 5: Replica Handling Proof and Commit
  Input: Proof $PF$ for round $R = (i, s)$
  if $\mathit{LeaderFor}(i, s) = PF.\mathit{signer}$ and HaveProposal($R$) then
      Commit($R$);   ResetState($i$+1);
```

```
Algorithm 6: Timeout and Sequence Advancement
  Input: Current round $R = (i, s)$
  OnTimeout:;
      $V_t \leftarrow (\text{type=timeout}, R, \text{signer=self}, \text{weight} = w_{\text{self}})$;   Broadcast($V_t$);
  OnSuperMajorityTimeout:;
      AdvanceSequence($s$+1);
```

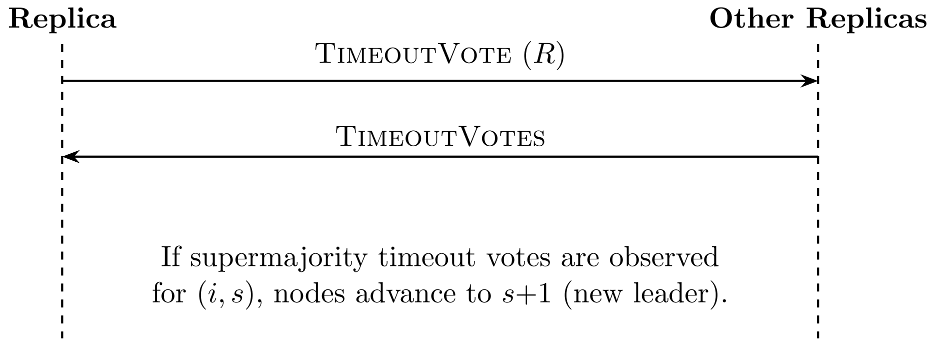
5 Message Flow

5.1 Normal Case


Leader
Replicas
PROPOSAL (B, R)
VOTE $(B.\text{hash}, R)$
PROOF (aggregated VOTE objects, R)
VOTE (PROOF receipt, R)

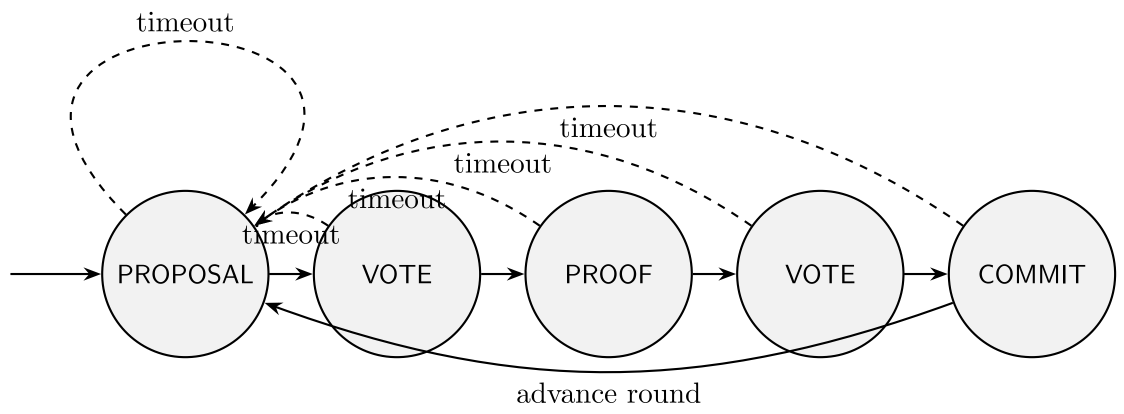
Upon valid PROOF *(containing vote objects to supermajority) and final* VOTE*, replicas locally commit for round R and advance.*

5.2 Timeout / Recovery



If supermajority timeout votes are observed for (i, s), nodes advance to $s+1$ (new leader).

6 State Machine



7 Slots as Pacemaker

Traditional BFT protocols rely on timers to regulate progress, but timers are vulnerable to drift and cannot be independently verified. RadiumBFT integrates **slots** as a verifiable pacemaker mechanism. Slots are core to the protocol, ensuring all nodes advance rounds in deterministic synchrony.

7.1 Slot Structure

A slot consists of a fixed sequence of $M = 101$ *slot entries.* Each entry contains:

- An **index** (contiguous counter across the slot),
- A **difficulty** value,
- A **digest prefix**, the first4(H) of a hash derived from the previous digest.

Entries form a chain: each digest is computed by hashing the previous digest with the configured difficulty. This makes the slot sequence deterministic and publicly verifiable.

7.2 Slot Algorithms

```
Algorithm 7: GenerateSlot
  Input: Previous digest $D_{prev}$, difficulty $d$, length $M = 101$
  Output: Slot entries $E$
  $cur \leftarrow D_{prev}$; $E \leftarrow [\,]$;
  for $k \leftarrow 1$ to $M$ do
      $H \leftarrow$ HashDigestToDigest($cur, d$);
      $E \leftarrow E \cup \{(k, d, \mathsf{first4}(H))\}$;
      $cur \leftarrow H$;
  return $E$
```

```
Algorithm 8: VerifySlot
  Input: Optional previous digest $D_{prev}$; slot entries $E$
  foreach checkpoint entry $e$ in $E$ do
      $H \leftarrow$ HashDigestToDigest($e.prev, e.difficulty$);
      if $\mathsf{first4}(H) \neq e.digest\_prefix$ then
          return false
  return true
```

8 Integrating Slots, Blocks, and Proposals

This section ties together the on-chain data path used by RadiumBFT leaders:

Generate Slot $\longrightarrow$ **Build Block (embed slot)** $\longrightarrow$ **Generate Proposal**.

8.1 Data Path and Responsibilities

- **Slot**: verifiable pacemaker proof for round $R = (i, s)$; leader computes M=101 entries at difficulty d, signs the slot.

- **Block**: targets height $i+1$, *embeds the full slot* (entries, signer, signature, previous-slot digest).
- **Leader Proposal**: Includes the block and the round metadata; proposal is signed by the leader.

The verification stack at replicas is strictly layered:

$$\textsc{VerifyProposalSig} \wedge \textsc{VerifyBlock} \wedge \textsc{VerifyEmbeddedSlot}$$

Only after all three pass does a replica consider the proposal *valid* and apply SafeToVote for R.

8.2 Encoding

The leader proposal encodes the `Round` (i, s), the `block` (which itself contains the serialized slot), then `signer` and `signature`.

8.3 Pseudocode

Algorithm 9: BuildBlockWithEmbeddedSlot

```
Input: Parent block $B_i$, previous-slot digest $D^{slot}_{i-1}$, difficulty $d$, length $M$=101
Output: Block $B$ for height $i$+1 containing slot $S$
$S \leftarrow \textsc{GenerateSlot}(D^{slot}_{i-1}, d, M)$;
$B \leftarrow \textsc{AssembleBlock}(B_i, S, \text{txs}, \text{metadata})$;
return $B$
```

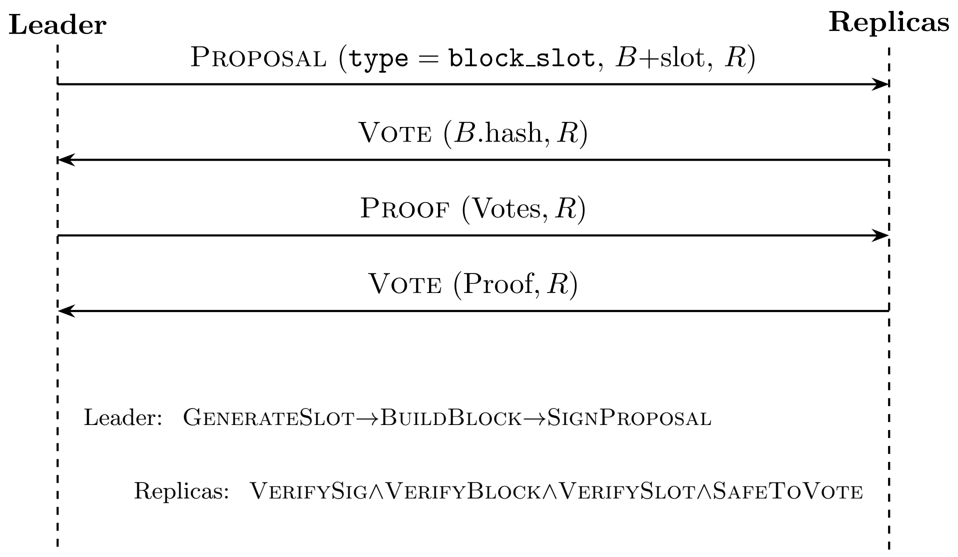
Algorithm 10: LeaderProposalWithSlot

```
Input: Round $R = (i, s)$, keypair $k$, parent $B_i$, $D^{slot}_{i-1}$, difficulty $d$
Output: Signed proposal $P$ of type block_slot
$B \leftarrow \textsc{BuildBlockWithEmbeddedSlot}(B_i, D^{slot}_{i-1}, d)$;
$P \leftarrow (\texttt{round}{=}R, \texttt{block}{=}B, \texttt{signer}{=}k.\text{pub})$;
Sign(P, k);
Broadcast(Proposal, P);
```

Algorithm 11: ReplicaValidateProposal

```
Input: Proposal $P = (\texttt{type}, R = (i, s), B)$
if $\textsc{VerifySignature}(P) = \textit{false}$ then
    return reject
if $\textsc{VerifyBlock}(B) = \textit{false}$ then
    return reject
if VerifySlot(B.slot)=false then
    return reject
if $\textsf{LeaderFor}(i, s) \neq P.\texttt{signer}$ then
    return reject
if $\textsc{SafeToVote}(R)$ then
    $\textsc{SendVote}(V, \textsf{LeaderFor}(R))$
```

8.4 Sequence View (clean rendering)


Leader
Replicas
PROPOSAL (type = block_slot, B+slot, R)
VOTE (B.hash, R)
PROOF (Votes, R)
VOTE (Proof, R)
Leader: GENERATESLOT→BUILDBLOCK→SIGNPROPOSAL
Replicas: VERIFYSIG∧VERIFYBLOCK∧VERIFYSLOT∧SAFETOVOTE

8.5 Correctness Linkage

Embedding the slot inside the block makes the pacemaker *first-class* in consensus: (i) leaders cannot issue a proposal without a verifiable slot, (ii) replicas can deterministically validate the delay before casting votes, and (iii) the commitment pipeline (Proposal → Vote → Proof → Vote) remains unchanged while being paced by a cryptographic, non-drift mechanism. This preserves instant finality and quorum-intersection safety while replacing fragile timing assumptions.

8.6 Generation and Verification

Leaders must generate a full slot before they are eligible to issue a proposal:

1. Start from the digest of the previous slot.
2. Iteratively compute M entries using the difficulty.
3. Produce the final digest and sign the slot.

Replicas verify only the *checkpoints* (first and last entries by default), recomputing the expected digest and comparing against the stored prefix. This reduces verification cost while preserving security.

8.7 Dynamic Difficulty

The system dynamically adjusts slot difficulty to target a desired interval (e.g., ∼500ms). Overhead such as block assembly and gossip delay is measured locally; difficulty is scaled so that the effective slot duration remains aligned. This ensures fairness and pacing across heterogeneous hardware and network conditions.

8.8 Role in Consensus

Slots serve as the *pacemaker* of RadiumBFT:

- Leaders cannot propose before completing the slot, preventing premature advancement.
- Replicas can independently verify that proposals are correctly delayed.
- All nodes consume the same amount of computational work per round, ensuring synchronized pace.

Thus slots replace fragile timers with a deterministic, cryptographically enforced mechanism that underpins the liveness and consistency of RadiumBFT.

Algorithm 12: UpdateDynamicDifficulty

```
Input: Fixed difficulty $d_{fix}$; overhead $over$; target interval $T$
if $over \geq T$ then
    return 1
$d_{dyn} \leftarrow d_{fix} \cdot \frac{T - over}{T}$;
return $\max(1, \lfloor d_{dyn} \rceil)$
```

9 Correctness

We establish safety (no two conflicting commits at the same height) and liveness (eventual commit) under the system model in Section 2 and the protocol rules in Sections 3 and 7.

9.1 Preliminaries and Validity Predicates

Let n be the number of delegates, each with voting weight $w_j > 0$ normalized so that $\sum_j w_j = 1$. A *quorum* is any set of votes whose total weight is at least τ with $\frac{1}{2} < \tau \leq 1$ (typically $\tau \geq 2/3$). We assume at most $f \leq \lfloor (n-1)/3 \rfloor$ Byzantine nodes; the remainder are honest.

A *round* is $R = (i, s)$, where the target commit height is $i{+}1$ and s is the leader sequence for height i. A *proposal* P for R contains a block B that extends height i, and embeds a *slot* that serves as a verifiable pacemaker proof for R.

Proof validity. A *proof* PF for round R is an object produced by the scheduled leader $\ell = \mathsf{LeaderFor}(i, s)$ that *contains* a multiset of *vote objects*

$$\text{PF.votes} = \{V_1, \ldots, V_m\},$$

such that:

1. (Same round) $\forall k : V_k.\text{round} = R$.
2. (Same value) $\forall k : V_k.\text{hash} = H_B$ where H_B is the hash of the same proposal block B.
3. (Distinct signers) Signers are unique across the included votes.
4. (Threshold) $\sum_k V_k.\text{weight} \geq \tau$.
5. (Leader signature) PF is signed by ℓ and $\ell = \mathsf{LeaderFor}(i, s)$.

We write $\textsc{ValidProof}(\text{PF}, R, B)$ when all conditions hold.

Local voting rule. Each honest replica maintains a variable *LastVoted*. A replica may cast at most one vote per round and only if $\textsc{SafeToVote}(R)$ holds, where $\textsc{SafeToVote}(R)$ requires: (i) the local round equals R, (ii) the received proposal for R is valid, including $\textsc{VerifyProposalSig} \wedge \textsc{VerifyBlock} \wedge \textsc{VerifySlot}$, and (iii) $R > \mathit{LastVoted}$ in lexicographic order on (i, s).

Commit rule. Upon receiving PF with $\textsc{ValidProof}(\mathrm{PF}, R, B)$ and having cached the matching proposal $P = (B, R)$, an honest replica commits B at height $i{+}1$ and advances its local state to height $i{+}1$.

Pacemaker neutrality. Slots gate proposal eligibility (leaders must embed a valid slot) and are independently verifiable by replicas. Slot difficulty and verification checkpoints affect *performance only*, not the validity predicates above.

9.2 Invariants

Lemma 1 (Quorum intersection)**.** *Let Q_1, Q_2 be two quorums each with weight at least $\tau > \frac{1}{2}$. Then $\mathrm{wt}(Q_1 \cap Q_2) \geq 2\tau - 1 > 0$.*

Proof sketch. $\mathrm{wt}(Q_1 \cap Q_2) = \mathrm{wt}(Q_1) + \mathrm{wt}(Q_2) - \mathrm{wt}(Q_1 \cup Q_2) \geq 2\tau - 1$. □

Lemma 2 (Vote monotonicity)**.** *An honest replica casts at most one vote per round R and never votes for two different blocks in the same round or height.*

Proof sketch. By the local rule, a vote is permitted only when $\textsc{SafeToVote}(R)$ holds and $R > \mathit{LastVoted}$; upon voting, $\mathit{LastVoted} \leftarrow R$. Thus no second vote in the same round is possible, and value checks tie the vote to one block. □

Lemma 3 (Proof soundness)**.** *If $\textsc{ValidProof}(\mathrm{PF}, R, B)$ holds, then at least $(\tau - (1 - \tau)) = 2\tau - 1$ of the weight in PF must be honest (under the worst case that all non-honest weight is $(1 - \tau)$).*

Proof sketch. Threshold is $\geq \tau$, at most $(1 - \tau)$ can be Byzantine, so honest weight in the proof is $\geq \tau - (1 - \tau) = 2\tau - 1 > 0$. □

9.3 Safety

Theorem 1 (At most one commit per height)**.** *No two conflicting blocks $B \neq B'$ at the same height $i{+}1$ can both be committed by honest replicas.*

Proof sketch. Suppose for contradiction that both B and B' are committed at height $i{+}1$ via valid proofs PF for round $R = (i, s)$ and PF' for round $R' = (i, s')$, respectively. Each proof contains a quorum of votes for its value. By Lemma 1, the two quorums intersect with positive weight. By Lemma 2, an honest voter cannot vote for two different blocks at the same height and compatible round order. Thus the intersection cannot contain honest weight for both B and B', contradicting Lemma 3. Hence only one block can be committed at height $i{+}1$. □

Corollary 1 (Instant finality)**.** *Once a block B at height $i{+}1$ is committed under the commit rule, it cannot be reverted; there is no alternate committed history at that height.*

9.4 Liveness

We assume partial synchrony: there exists a (possibly unknown) GST after which message delays are bounded by Δ, and the network remains connected among honest replicas. We also assume a fair leader schedule: for any fixed height i, an honest leader appears infinitely often in the sequence $s = 0, 1, 2, \ldots$.

Lemma 4 (Round alignment via slots)**.** *If all honest replicas verify the same embedded slot for round $R = (i, s)$, then their local view of R is aligned (up to message delay), independent of local wall-clock timers.*

Proof sketch. Slot verification is deterministic: each replica checks the same entry chain (with identical inputs: previous slot digest, length, and difficulty). Therefore acceptance or rejection of Rs slot is identical across honest replicas, producing the same eligibility barrier for proposals and the same logical round. $\square$

Lemma 5 (Progress across faulty leaders)**.** *If a round $R = (i, s)$ fails to produce a valid proof, honest replicas eventually move to some $R' = (i, s')$ with $s' > s$.*

Proof sketch. Post-GST, either (a) the scheduled leader is faulty or partitioned and no valid proof forms, in which case the timeout mechanism yields supermajority timeout votes to advance the sequence; or (b) the leader is honest but cannot gather a quorum due to transient delays, in which case the same mechanism triggers. Timeouts are independent of slots and do not affect slot validity; they only advance the leader sequence. $\square$

Theorem 2 (Liveness)**.** *Under partial synchrony and a fair leader schedule, RadiumBFT eventually commits a block at height $i{+}1$.*

Proof sketch. After GST, by Lemma 5 the system advances through sequences until an honest leader $\ell = \mathsf{LeaderFor}(i, s^\star)$ is selected. By Lemma 4, honest replicas are aligned on round $R^\star = (i, s^\star)$ via slot verification. The honest leader proposes a valid block with an embedded valid slot; honest replicas validate and vote once (Lemma 2). The leader aggregates votes to weight $\geq \tau$ and issues a valid proof (Lemma 3); replicas commit by the commit rule. Thus the protocol makes progress to height $i{+}1$. $\square$

9.5 Neutrality of Performance Parameters

Lemma 6 (Safety independence)**.** *Safety does not depend on slot difficulty, the number of slot checkpoints verified, or the adaptive difficulty heuristic.*

Proof sketch. These parameters influence *when* a leader becomes eligible to propose and the verification cost, but not the vote monotonicity, quorum threshold, or proof validity predicates used in the safety proof. $\square$

Summary. Safety follows from vote monotonicity and quorum intersection with proofs defined as *aggregations of vote objects.* Liveness follows from partial synchrony, fair leader rotation, deterministic round alignment via slots, and sequence advancement across faulty leaders. Instant finality is a direct consequence of the commit rule and the impossibility of two conflicting commits at the same height.

10 Performance

Throughput metric. We report throughput in *objects per second* (OPS), where an "object" is the atomic unit executed by Radiums runtime. A transaction may carry multiple objects; hence TPS can be misleading. OPS measures executed work directly and is the primary metric we use in this paper.

11 Related Work

PBFT [1] introduced the classical three-phase protocol (pre-prepare, prepare, commit) proving that safety and liveness are achievable in partially synchronous networks with $f \leq (n-1)/3$ Byzantine replicas, albeit at $O(n^2)$ per round.

Tendermint [2] adapted BFT consensus to blockchains with rotating leaders and height-based rounds (pre-vote, pre-commit). It achieves deterministic finality but often incurs extra phase changes under failures.

HotStuff [3] linearizes BFT with pipelined chained quorum certificates; it reaches finality after observing a three-deep chain of QCs. RadiumBFT pursues a single proposal–vote–proof pipeline while preserving quorum intersection safety and providing instant finality.

12 Conclusion

RadiumBFT achieves deterministic finality with minimal phases, enabling high-throughput, forkless blockchains. Its topology-aware design and IPv6 optimizations make it suitable for terrestrial and space-based deployments alike. Future work includes formalization of weighted-quorum thresholds in adversarial network partitions and mechanized verification.

References

[1] Castro, M., & Liskov, B. (1999). *Practical Byzantine Fault Tolerance*. In Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI).

[2] Buchman, E. (2016). *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. Masters thesis, University of Guelph.

[3] Yin, M., Malkhi, D., Reiter, M. K., Gueta, G. G., & Abraham, I. (2019). *HotStuff: BFT Consensus with Linearity and Responsiveness*. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC).