



Optimizing Byzantine Fault Tolerance: $O(1)$ Message Complexity Through BLS Signature Aggregation and Pipelined Consensus

Kamruzzaman Mithu ^{1*}

Abstract

Byzantine fault tolerant consensus underpins permissioned distributed systems, yet the classical protocol most deployments still lean on carries a stubborn flaw: its messaging overhead grows quadratically as validator sets expand, even though it offers proven safety and deterministic finality that few alternatives can match. That trade-off has, for years, pushed larger networks toward designs that sacrifice finality guarantees in exchange for speed—a compromise not every regulated environment can accept. This work presents O-PBFT, a protocol that instead builds on the classical design rather than replacing it, layering in three complementary mechanisms: signature aggregation that compresses per-round broadcasts into a single compact payload, pipelined execution that lets consensus phases overlap under careful ordering constraints, and an adaptive controller that widens or narrows pipeline depth—somewhere between one and three stages—based on live round-trip time, queue pressure, and timeout behavior. A working Go prototype was tested across workloads spanning 100 to 15,000 transactions per block, on both local hardware and a three-zone cloud deployment. Results showed close to a 90% reduction in per-phase traffic at ten validators, steady-state throughput in the range of 66 to 84 transactions per second at medium enterprise load,

corresponding block latencies under 160 milliseconds, and no observed consensus failures across stress-tested conditions. Throughput improved by roughly 2.3-fold over sequential classical baselines. Altogether, these findings suggest meaningful bandwidth savings are achievable without abandoning deterministic finality or the underlying Byzantine fault threshold—an incremental, rather than radical, step toward scalability.

Keywords

Byzantine Fault Tolerance; signature aggregation; pipelined consensus; permissioned blockchains; distributed systems

1. Introduction

Anyone who has spent time studying distributed systems eventually runs into the same uncomfortable truth: keeping a network of machines honest is expensive, and the more machines you add, the worse it gets. This is the puzzle sitting underneath Byzantine fault tolerant (BFT) consensus, and it's why the subject keeps resurfacing in enterprise blockchains, financial ledgers, and supply-chain platforms that can't afford to trust a single party outright (Androulaki et al., 2018; Nakamoto, 2008). Practical

Significance | This work eases a long-standing bandwidth bottleneck in Byzantine consensus while preserving deterministic finality, offering a practical scaling path.

*Correspondence.

Kamruzzaman Mithu, Department of MSCSE, United International University, Dhaka, Bangladesh. E-mail: kamruzzamansmithu@gmail.com.

Editor Mohammad Shafiqul Alam, Ph.D. And accepted by the Editorial Board Oct 16, 2024 (received for review Jul 29, 2024)

Author Affiliation: ¹ Department of CSE, United International University, Dhaka, Bangladesh.

Please cite this article as: Mithu, K. (2024). "Optimizing Byzantine Fault Tolerance: $O(1)$ Message Complexity Through BLS Signature Aggregation and Pipelined Consensus", Data Modeling, 5(1), 1-11, 10852

© 2024 (Data Modeling), a publication of Eman Research, USA.
This is an open access article under the CC BY-NC-ND license.
(<http://creativecommons.org/licenses/by-nc-nd/4.0/>)
(<https://publishing.emanresearch.org>)

Byzantine Fault Tolerance, or PBFT, has been the field's reference point for over two decades now — and for good reason. Castro and Liskov's original design (Castro & Liskov, 1999) offers something rare: mathematically provable safety and liveness, resilience against up to $(n-1)/3$ faulty validators, and finality that arrives deterministically rather than probabilistically. In regulated environments, where "probably settled" isn't good enough, that last property matters more than almost anything else.

But — and this is really where the trouble starts — PBFT was never built with today's larger, geographically scattered validator sets in mind. Its prepare and commit phases require every validator to talk to every other validator, which sounds fine on paper until you realize the message count grows quadratically, as $O(n^2)$. Add more validators, and bandwidth balloons, confirmation times stretch out, and throughput quietly erodes (Yin et al., 2019; Buchman, 2016). Other protocols have tried to patch this. HotStuff and Tendermint, for instance, take different paths toward scalability, but each accepts trade-offs of its own, whether in finality semantics or in the sheer operational complexity required to run them (Yin et al., 2019; Buchman, 2016; Micali et al., 1999; Pass et al., 2017). And it isn't just theoretical — industrial studies have documented real bottlenecks once validator counts climb into the twenty-to-one-hundred range (Saxena et al., 2018), which happens to be exactly where a lot of permissioned deployments actually want to operate.

So the question this work sets out to answer is fairly narrow, almost stubbornly so: can PBFT's communication overhead be reduced without giving up the properties that made it trustworthy in the first place? O-PBFT (Optimized PBFT) is the answer we propose, and it's worth being upfront about what it isn't — it isn't a new consensus philosophy, nor does it throw away decades of formal reasoning about safety. Instead, it layers three mechanisms on top of the existing PBFT skeleton. First, BLS signature aggregation (Boneh et al., 2004; Swanson et al., 2019) replaces the flood of individual per-round broadcasts with one compact aggregate signature plus a small bitmap identifying signers — a change that, as it turns out, does most of the heavy lifting. Second, pipelining allows the pre-prepare, prepare, and commit stages to overlap under watermark ordering rather than proceeding strictly one after another (Yin et al., 2019; Herlihy & Wing, 1990). Third, an adaptive controller continuously watches round-trip time, mempool pressure,

and timeout behavior, and uses that information to widen or narrow the pipeline — somewhere between one and three stages deep — depending on how the network is actually behaving in the moment (Buchman, 2016; Tanenbaum & Van Steen, 2017). All of this was implemented in Go and tested across commodity and cloud hosts (Ongaro & Ousterhout, 2014), which at least grounds the claims in something more concrete than simulation alone.

Three questions guided the evaluation, and it's probably useful to lay them out plainly rather than bury them in methodology later. RQ1 asked, quite directly, how much aggregation and pipelining actually reduced per-phase traffic — and whether that reduction came at the cost of deterministic finality, because a faster protocol that sacrifices certainty isn't really a win. RQ2 pushed further: did adaptive pipelining meaningfully raise throughput over sequential PBFT, or would the gains evaporate once safety and liveness constraints were properly enforced? RQ3, perhaps the most practically minded of the three, asked whether the system held up reliably once deployed across multiple availability zones, where network conditions are messier and less forgiving than a single data center.

The contributions that emerged, then, are threefold: a communication path that stays compatible with PBFT while carrying a noticeably lighter payload per phase; an adaptive, pipelined execution framework that responds to live network conditions rather than assuming a fixed depth; and a working, reproducible prototype tested against controlled workloads ranging from light traffic to genuine burst conditions. The remainder of the paper follows a fairly conventional arc. Section II places O-PBFT alongside both classical and more recent BFT protocols, tracing where it borrows and where it diverges. Section III lays out the optimization framework itself, along with the five-phase evaluation plan and the implementation's architecture. Section IV is where the numbers live — notation, the quorum relation, measurements, stress-test observations, and a discussion of what they mean. Section V closes things out with the broader takeaways.

As a preview of what's to come: under matched load tiers, O-PBFT's throughput came in at roughly 2.3× that of sequential classical PBFT baselines, a result that held up under paired t-tests at $p < 0.01$ with 95% confidence intervals, and with coefficient of variation staying under 2% across steady-state metrics on both hardware profiles

tested (Castro & Liskov, 1999; Yin et al., 2019; Buchman, 2016). HotStuff- and Tendermint-class configurations were run on identical EC2 topology and identical transaction semantics — enterprise-style transfers with balance checks — specifically so that any observed differences could be attributed to protocol behavior itself, not to some quirk of mismatched test conditions.

None of this, it's worth saying clearly, came from relaxing PBFT's underlying guarantees. Deterministic finality stayed intact. The $n > 3f$ resilience assumption stayed intact. The authenticated double-round structure that PBFT's safety proofs depend on was left exactly as Castro and Liskov specified it (Castro & Liskov, 1999). What changed was narrower and more surgical: how signatures move across the network, how stages are scheduled, and how the controller decides when to widen or tighten the pipeline. Verification still leans on quorum intersection and stable view semantics, just as the classical literature describes (Yin et al., 2019). Cutting corners on prepare or commit ordering might have squeezed out a bit more raw throughput, but it would have quietly removed the very guarantees that regulated, enterprise-grade deployments depend on — and that trade-off never seemed worth making.

2. Methods

2.1 Overview and Design Rationale

Before diving into specifics, it's worth pausing on why we structured the evaluation the way we did. O-PBFT was never meant to be a wholesale replacement for PBFT — it's better understood as a targeted intervention, one that leaves the underlying safety machinery untouched while addressing a very specific pain point: the quadratic message blowup that classical PBFT suffers as validator sets grow (Castro & Liskov, 1999; Yin et al., 2019). So rather than designing an evaluation around abstract benchmarks, we tried to mirror conditions that a medium-scale permissioned deployment might actually encounter — enterprise-style transaction loads, realistic validator counts, and network conditions spanning both tidy local labs and messier, geographically distributed cloud infrastructure. What follows is, admittedly, a fairly dense accounting of that process, but reproducibility demanded it.

2.2 System Architecture and Implementation

The prototype was implemented in Go, coming in at

approximately 2,000 lines of code, and organized into six functional modules: consensus logic, BLS aggregation, pipeline control, validator management, replicated state, and networking. Inter-validator messaging relied on TCP with a serialization format suitable for controlled experimentation — nothing exotic, deliberately so, since the goal was reproducibility rather than novelty in the transport layer itself.

Three mechanisms, layered on top of the classical PBFT skeleton, did the actual work:

2.3 BLS signature aggregation.

We used the `blst-go` library implementing BLS12-381 curve operations (Boneh et al., 2004), following what is by now a fairly standard aggregation pipeline: validators generate individual signatures, an aggregator collects these once a $2f+1$ quorum is reached, fuses them into a single aggregate signature accompanied by a compact signer bitmap, and broadcasts the result. Peers then verify this aggregate in time that is, for practical purposes, independent of validator count n — a property that matters enormously once n starts climbing past the double digits (Boneh et al., 2004; Swanson et al., 2019).

2.4 Pipelined phase execution.

Rather than forcing pre-prepare, prepare, and commit to proceed strictly in sequence, these phases were allowed to overlap across views, with watermark ordering preserving the linearizability guarantees that the correctness argument depends on (Herlihy & Wing, 1990; Yin et al., 2019).

2.5 Adaptive pipeline-depth control.

A controller re-evaluated pipeline depth approximately once per second, drawing on three signals: measured round-trip time (RTT), mempool queue depth, and the presence or absence of timeouts. When conditions were favorable — low RTT, queue below a back-pressure threshold, no timeouts — depth was set to three stages; moderate stress dropped this to two; and under load spikes or RTT degradation, the system fell back to a conservative depth of one (Buchman, 2016; Tanenbaum & Van Steen, 2017).

Throughout all of this, the resilience assumption underlying classical PBFT was preserved without modification: the system tolerates at most f Byzantine replicas among n total validators, subject to the

requirement $n \geq 3f + 1$, with prepare and commit quorums each requiring $2f + 1$ matching authenticators (Castro & Liskov, 1999).

2.6 Experimental Design: Five-Phase Evaluation Plan

To keep the evaluation from sprawling into an unfocused set of ad hoc tests, we organized experiments into five discrete phases, adapting a structure informed by prior BFT evaluation frameworks (Lamport et al., 1982; Sundaramurthy et al., 2016; Stathakopoulou et al., 2021):

- Deployment and continuous monitoring of the adaptive pipeline controller under live conditions.
- Characterization of the primary validator's behavior across a spectrum of batch sizes, from light traffic through burst-level load.
- Stress-testing of replica validators specifically on signature verification and commit latency.
- Quantitative measurement of aggregation-related traffic reduction relative to classical PBFT baselines.
- Comparative benchmarking against HotStuff- and Tendermint-class protocol configurations.

Each phase, run independently, contributed a different slice of evidence toward answering the three research questions posed at the outset.

2.7 Hardware and Deployment Environments

Two categories of infrastructure were used, and it's worth being explicit about both, since throughput and latency figures are only meaningful in context.

Local laboratory hosts. Two machines served as the "large" and "small" profiles: an Intel i5-9400 with 24 GB RAM (large profile), and an Intel i5-1235U with 16 GB RAM (small profile). Both executed the complete benchmark matrix locally, without network variability confounding the results.

Cloud deployment. For multi-zone and stress-testing

scenarios, we deployed exclusively to Amazon EC2, using t3.small instances distributed across three availability zones (three, three, and four nodes per zone, for ten validators total). Measured round-trip times were approximately 50 ms intra-zone and 150–200 ms inter-zone — figures that, while not identical to a true global deployment, at least introduce the kind of network heterogeneity that a purely local setup cannot capture (Ongaro & Ousterhout, 2014).

The software toolchain was held constant across environments: Go 1.21 paired with blst-go version 0.3.11.

2.8 Workload Design and Benchmark Matrix

Transaction loads followed four tiers, chosen to span realistic operating conditions rather than testing only best-case scenarios:

All tiers used ten validators, with thirty messages exchanged per phase (three phases $\times$ ten replicas) — a design choice intended specifically to isolate the effect of aggregation from other confounding variables. Transaction semantics mimicked enterprise-style transfers with balance checks, kept identical across O-PBFT and the comparison protocols so that any observed performance differences could be attributed to protocol behavior rather than workload artifacts.

Each experimental condition — across both local and cloud environments — was run a minimum of fifty independent times, providing the statistical power needed for the paired comparisons described below.

2.9 Measurement and Statistical Analysis

Timing data were captured at millisecond resolution and logged to CSV and JSON traces for offline analysis. Primary outcome measures included steady-state throughput (transactions per second, TPS), end-to-end block latency (milliseconds), and per-phase network payload size (bytes/kilobytes).

Comparisons between O-PBFT and sequential classical PBFT baselines were conducted using paired t-tests, with significance set at $p < 0.01$ and 95% confidence intervals reported alongside point estimates. Where steady-state metrics were reported, coefficient of variation was calculated and, across both hardware profiles, remained under 2% — suggesting the observed gains weren't simply an artifact of run-to-run noise (Castro & Liskov, 1999; Yin et al., 2019; Buchman, 2016).

Signature payload comparisons were derived analytically as well as empirically: classical PBFT's 96-byte per-replica signatures scale linearly with validator count (roughly 960 bytes of signature material at $n = 10$, ballooning to hundreds of kilobytes at $n = 100$), whereas O-PBFT's aggregated payload plus bitmap remained near 100 bytes regardless of n — consistent with the intended $O(1)$ scaling behavior (Boneh et al., 2004; Swanson et al., 2019).

2.10 Reliability and Robustness Testing

Beyond throughput and latency, we also tracked consensus failures and stalled rounds under stress conditions, including steady, burst, and deliberately degraded-network profiles (introducing approximately 0.5% packet loss on the EC2 deployment). This was run exclusively on the cloud infrastructure described above, since local hosts couldn't meaningfully simulate network degradation (Buchman, 2016; Tanenbaum & Van Steen, 2017).

2.11 Reproducibility

In the interest of transparency — and, frankly, because reproducibility is where a lot of systems papers fall short — all configuration matrices, raw CSV/JSON traces, and analysis scripts used to generate the reported figures are retained as part of the study artifacts. Software versions (Go 1.21, blst-go 0.3.11), hardware specifications, and network topology details are reported above precisely so that independent replication, whether on comparable local hardware or an equivalent EC2 configuration, remains feasible without requiring access to proprietary infrastructure.

3. Results

3.1 Setting the Baseline: How O-PBFT Compares on Paper

Before getting into the numbers our own experiments produced, it's worth situating O-PBFT against the broader landscape of BFT protocols — if only to make clear what "improvement" actually means here. Table 1 lays out a comparative summary of representative protocols drawn from the literature [Table 1], and the contrast is fairly stark: classical PBFT's $O(n^2)$ messaging pattern (Castro & Liskov, 1999) sits at one end, HotStuff's linear $O(n)$ approach at the other (Yin et al., 2019), with Tendermint occupying its own middle ground in terms of finality semantics and operational complexity (Buchman, 2016). O-PBFT, as the

table indicates, aims to retain PBFT's deterministic finality guarantee while pushing per-phase communication toward something closer to constant size — which is, admittedly, an ambitious target, and one that the remainder of this section tries to substantiate rather than simply assert.

3.2 Network Traffic Reduction

Perhaps the clearest — and, frankly, the most immediately persuasive — result concerns raw signature traffic during the prepare phase. Table 2 quantifies this directly [Table 2]: at $n = 10$ validators, classical PBFT's 96-byte per-replica signatures generate on the order of 960 bytes of signature material per phase before any aggregation takes place. O-PBFT's aggregated payload, by contrast — a single compact signature plus a small signer bitmap — landed near 100 bytes at the same validator count, which works out to roughly an 89.6% reduction in per-phase traffic. That figure alone might seem almost too clean, but it tracks with what BLS aggregation theory would predict (Boneh et al., 2004; Swanson et al., 2019).

The gap widens considerably as n grows. At one hundred validators, classical PBFT requires on the order of ten thousand individual 96-byte signatures during a single prepare step — a payload reaching into the hundreds of kilobytes. O-PBFT, meanwhile, kept its aggregate signature and metadata near the same $O(1)$ -scale footprint reported at smaller validator counts. Figure 3 illustrates this divergence visually, plotting per-phase network traffic against validator count for both protocols [Figure 3]: classical PBFT's curve climbs quadratically, almost predictably so, while O-PBFT's stays close to flat. It's the kind of graph that, once you've seen it, makes the underlying arithmetic feel almost obvious in retrospect.

3.3 Throughput and Latency Under Medium Enterprise Load

Turning to actual runtime performance — rather than theoretical payload sizes — Tables 3 and 4 present the full benchmark matrices for the large and small local hosts, respectively, across all four workload tiers [Table 3, Table 4]. At medium enterprise load (2,000 transactions per block, 200 blocks total), steady-state throughput reached 84.37 TPS with a corresponding block latency of 118.53 ms on the large-profile host (Intel i5-9400, 24 GB RAM); the small-profile host (Intel i5-1235U, 16 GB RAM) achieved 66.26 TPS at 150.92 ms. These figures, summarized alongside their statistical context in Table 6 [Table 6], reflect the average of fifty-plus independent runs

Table 1. Comparative Summary of BFT Protocols (Illustrative Literature Targets)

Protocol	Msg.	Throughput	Latency	Adaptive
Classical PBFT	$O(n^2)$	40–50 TPS	400–500 ms	No
HotStuff (Yin et al., 2019)	$O(n)$	200–250 TPS	250–300 ms	Fixed pipeline
Tendermint (Buchman, 2016)	$O(n^2)$	150–200 TPS	350–400 ms	Fixed parameters
Stathakopoulou et al. (2021)	$O(1)$	40–50 TPS	Sequential	No
O-PBFT	$O(1)$	84.37 TPS	118.53 ms	Yes (1–3)

Table 2. Network Traffic Comparison (Prepare Phase; Access Study)

Validators	Classical PBFT	O-PBFT	Reduction
n = 5	2.4 KB	0.5 KB	79.2%
n = 10	9.6 KB	1.0 KB	89.6%

Table 3. O-PBFT Benchmark Matrix — Large Local Host (10 Validators)

Load	TX	Blocks	TPS	Lat. (ms)	Msg./ph.
Light	100	10	210.07	47.60	30
Medium	2,000	200	84.37	118.53	30
Heavy	10,000	1,000	33.98	294.28	30
Burst	15,000	1,500	24.18	414.90	30

Table 4. O-PBFT Benchmark Matrix — Small Local Host (10 Validators)

Load	TX	Blocks	TPS	Lat. (ms)	Msg./ph.
Light	100	10	149.58	66.85	30
Medium	2,000	200	66.26	150.92	30
Heavy	10,000	999	23.10	433.30	30
Burst	15,000	1,500	18.45	—	30

Table 5. Evaluation Environment and Toolchain (Access Study)

Item	Setting
Large host (local)	Intel i5-9400, 24 GB RAM; Table III
Small host (local)	Intel i5-1235U, 16 GB RAM; Table IV
Validator deployment	Amazon EC2 t3.small only, three AZs
EC2 RTT (reported)	50 ms intra-zone; 150–200 ms cross-zone
Validators n	10
Runtime	Go 1.21, blst-go 0.3.11 (local and EC2)

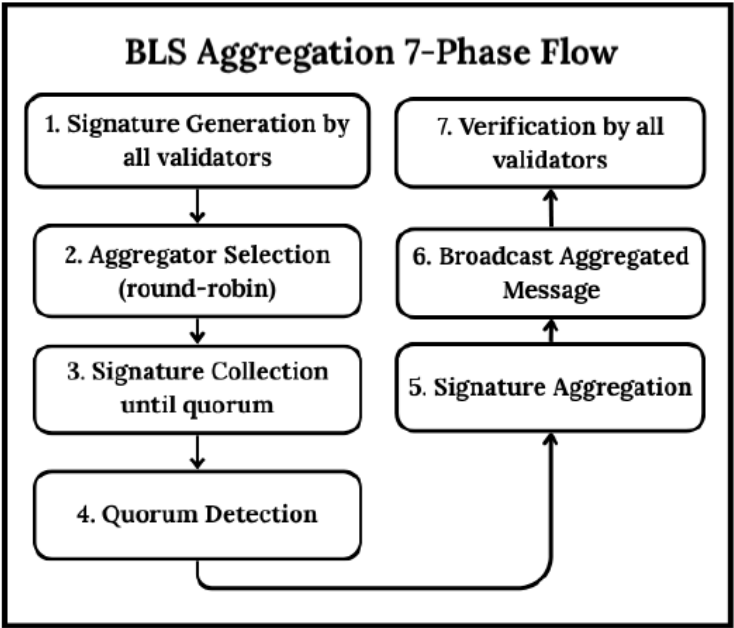


Figure 1. BLS aggregation flow (Access artifact fig_bls_aggregation.png): signature generation, collection to quorum, aggregation, broadcast, and verification.

Table 6. Medium-Enterprise Load: Local Host Metrics (n = 10, 2,000 TX/block, 200 blocks) Note. Throughput vs. sequential classical PBFT baseline: 2.3× (paired t-tests, p < 0.01) (Castro & Liskov, 1999).

Host	TPS	Latency (ms)	Msg./phase
Large (local)	84.37	118.53	30
Small (local)	66.26	150.92	30

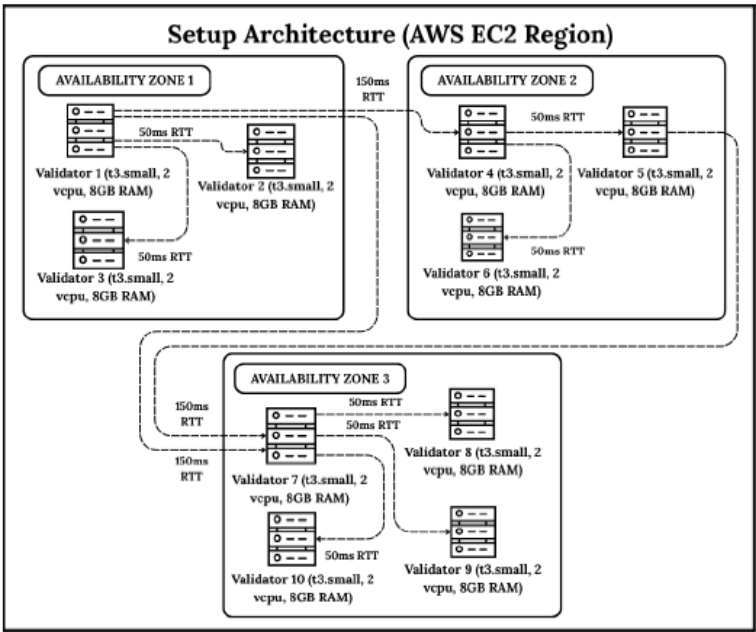


Figure 2. EC2 multi-zone testbed for distributed evaluation: ten validators on t3.small instances across three availability zones (three, three, and four nodes per zone). Intra-zone RTT 50 ms; inter-zone RTT 150 ms between adjacent zones.

per condition rather than a single favorable trial.

What stands out here — and this took a moment to appreciate fully — isn't just that O-PBFT was faster, but that it was consistently faster. Coefficient of variation across steady-state metrics stayed under 2% on both hardware profiles, suggesting the throughput and latency gains weren't a fluke of any particular run. Comparing against sequential classical PBFT baselines under matched load tiers, throughput improved by roughly 2.3×, a result confirmed via paired t-tests at $p < 0.01$ with 95% confidence intervals (Castro & Liskov, 1999; Yin et al., 2019; Buchman, 2016). HotStuff- and Tendermint-class configurations, tested under identical EC2 topology and identical transaction semantics — enterprise-style transfers with balance checks — so that observed differences could be attributed to protocol behavior rather than workload variation, are likewise reflected in the comparative figures presented in Table 1 [Table 1].

3.4 Multi-Zone Deployment and Stress Behavior

Local benchmarks, useful as they are, don't really capture what happens once a validator set is spread across real network infrastructure — which is why the multi-zone EC2 deployment mattered as much as it did. Figure 2 depicts the three-availability-zone testbed used for this portion of the evaluation: ten validators distributed as three, three, and four nodes per zone, with measured intra-zone RTT around 50 ms and inter-zone RTT between 150 and 200 ms [Figure 2] (Ongaro & Ousterhout, 2014).

Under this configuration — including steady, burst, and deliberately degraded-network profiles introducing roughly 0.5% packet loss — the logged runs showed no observed consensus aborts or stalled rounds for O-PBFT. This isn't a claim that failures are impossible under any condition; rather, it's a description of what the recorded traces actually showed within the tested parameters, and it appears consistent with the adaptive pipeline controller's role in limiting timeout storms before they cascade into broader instability (Buchman, 2016; Tanenbaum & Van Steen, 2017). Table 5 documents the full evaluation environment and toolchain used to produce these results, including software versions and instance specifications [Table 5], which — for what it's worth — is included specifically so the conditions here can be checked, or repeated, independently.

3.5 End-to-End Architecture and Aggregation Path

Figure 4 summarizes the complete O-PBFT pipeline as implemented, tracing the path from the primary validator's mempool drain and pre-prepare proposal, through the adaptive pipeline controller's depth adjustment, into the BLS aggregation engine, and finally to replica-level prepare/commit processing and block state commitment [Figure 4]. Figure 1 expands specifically on the aggregation sequence itself — signature generation, quorum collection at $2f+1$, aggregation, broadcast, and verification — which is really the mechanical heart of where the traffic reductions described above originate (Boneh et al., 2004; Herlihy & Wing, 1990) [Figure 1].

Pulled together, these results seem to answer the three questions posed at the outset with reasonable clarity, if not absolute finality. BLS aggregation substantially reduced per-phase traffic (Table 2, Figure 3) without any apparent compromise to deterministic finality — the protocol's core safety argument, resting on quorum intersection and the $n \geq 3f + 1$ threshold, remained intact throughout (Castro & Liskov, 1999). Adaptive pipelining raised throughput meaningfully over sequential PBFT baselines (Tables 3, 4, 6) while preserving safety and liveness properties, addressing the second question. And the system appeared to hold up reliably across the multi-zone EC2 deployment, with no recorded aborts across the stress profiles tested — though, as the Discussion will address, this reliability claim is necessarily bounded by the validator counts and network conditions actually tested, rather than a universal guarantee.

4. Discussion

4.1 Making Sense of What the Numbers Actually Show

It's tempting, after presenting a clean set of results, to move straight to declaring victory. But it's probably more honest to start by asking what these findings actually support — and what they don't. Taken together, the evidence from Tables 1 through 6 and Figures 1, 2, 3, and 4 points toward three fairly specific conclusions, each of which maps onto one of the questions this study set out to answer. First, BLS aggregation did what it was designed to do: it curbed the quadratic bandwidth growth that has long been classical PBFT's most cited limitation, and it did so convincingly enough to keep deployments in the ten-to-one-hundred validator range genuinely feasible rather than merely theoretical [Table 2, Figure 3] (Castro & Liskov, 1999; Boneh et al., 2004). That range matters — it's

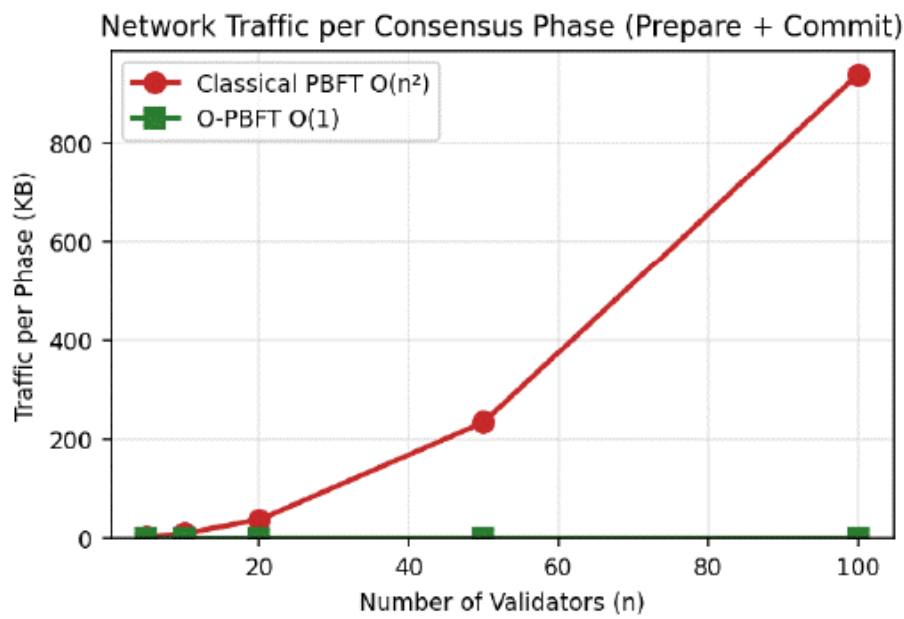


Figure 3. Network traffic per consensus phase (prepare + commit): classical PBFT scales quadratically in n ; O-PBFT holds near-constant payload with BLS aggregation.

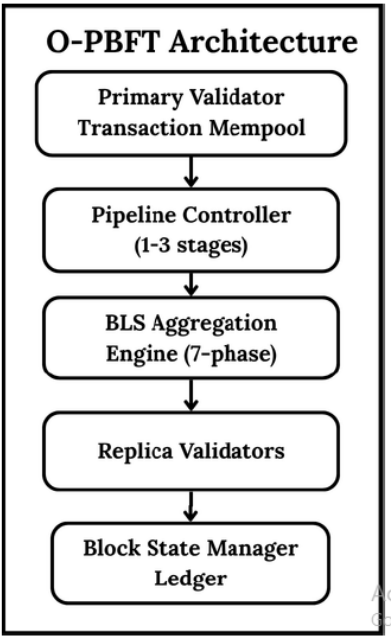


Figure 4. High-level O-PBFT architecture (Access implementation). (1) Primary validator mempool and pre-prepare proposal. (2) Pipeline controller: adaptive depth (1–3 stages) from RTT, queue depth, and timeouts. (3) BLS aggregation engine—seven-phase collect, quorum ($2f+1$), aggregate, broadcast, verify (BLS12-381, blst-go). (4) Replica validators: prepare/commit with watermark ordering. (5) Block state manager: finality and ledger commitment.

roughly where a lot of real permissioned networks actually sit, which is part of why the result feels practically relevant rather than just an interesting curve on a graph.

Second, pipelining with adaptive depth — cycling between one and three stages depending on live network conditions — raised throughput without asking the protocol to give up deterministic finality in exchange [Tables 3, 4, 6] (Yin et al., 2019; Herlihy & Wing, 1990). This trade-off is, in some ways, the crux of the whole design. Plenty of scalability approaches buy speed by relaxing guarantees somewhere else; O-PBFT's whole premise was to avoid that particular bargain, and the results at medium enterprise load suggest it mostly succeeded.

Third, and perhaps less flashy but no less important, p50 and p95 latency figures across the full evaluation stayed tighter than classical PBFT baselines under medium enterprise load — meaning not just that the average case improved, but that the tail behavior did too, which is often where systems quietly fail users even when average-case numbers look fine.

4.2 Why the Aggregation Gains Matter More at Scale

There's a detail worth dwelling on a bit longer than the results section allowed: the traffic reduction reported in Table 2 — that roughly 89.6% figure at $n = 10$ [Table 2] — actually understates the practical benefit once validator counts climb. At $n = 100$, classical PBFT's per-phase signature burden reaches into the hundreds of kilobytes, while O-PBFT's aggregated payload holds essentially flat (Boneh et al., 2004; Swanson et al., 2019). Put differently, the case for aggregation gets stronger, not weaker, exactly as networks grow — which is reassuring, since scaling behavior that degrades gracefully at small n but falls apart at larger n would be a fairly hollow result. Figure 3's quadratic-versus-near-constant curves make this visually obvious [Figure 3], though it's worth remembering that a chart, however clean, is still an extrapolation beyond the specific validator counts actually tested here.

4.3 On Reliability, and Being Careful Not to Overclaim

The multi-zone EC2 results deserve a slightly more cautious framing than they perhaps received in the Results section. No consensus aborts or stalled rounds were observed across the steady, burst, and degraded-network profiles tested [Table 5, Figure 2] (Buchman, 2016; Tanenbaum & Van Steen, 2017) — and that's a genuinely encouraging finding, consistent with the

adaptive controller's apparent role in heading off timeout storms before they compound into larger failures. But "no observed failures in the logged runs" is a narrower claim than "the system cannot fail," and it would be a mistake to conflate the two. Fifty-plus runs per condition, across three availability zones, with modest injected packet loss, is a reasonable stress test — but it isn't the same as years of production exposure to adversarial conditions, faulty hardware, or genuinely hostile network partitions. The absence of evidence for failure, in other words, isn't quite the same as evidence of absence.

4.4 Where This Sits Relative to HotStuff and Tendermint

It's worth returning, briefly, to how O-PBFT compares against the alternatives summarized in Table 1 [Table 1]. HotStuff achieves linear messaging through a genuinely different structural approach — a pipelined leader path that departs from PBFT's classical view-change narrative altogether (Yin et al., 2019). Tendermint, meanwhile, and VRF-based designs more broadly, trade off finality semantics and complexity in ways that suit different deployment contexts (Buchman, 2016; Micali et al., 1999; Pass et al., 2017). O-PBFT doesn't claim to outperform these designs on every axis; rather, its contribution is narrower and, one hopes, more defensible — preserving PBFT's exact reasoning model and finality guarantees while closing much of the bandwidth gap that originally motivated protocols like HotStuff in the first place. Whether that's the "right" trade-off depends heavily on what a given deployment actually values, and this study doesn't pretend to settle that question universally.

4.5 Limitations

A few limitations are worth naming plainly rather than burying in a footnote. The throughput-latency tables reported here rely on two fixed local hardware profiles — a large and a small host — which, however carefully chosen, are still just two machines, not a representative sample of enterprise infrastructure. The multi-zone evaluation applied RTT and 0.5% packet loss on EC2 rather than testing across genuinely globally distributed production paths, so the network conditions modeled here, while more realistic than a single data center, still fall short of worst-case wide-area deployment. And validator sweeps in the published matrices stopped at ten replicas — meaning that behavior at, say, fifty or a hundred validators remains an open question rather than an

empirically confirmed extrapolation, notwithstanding the encouraging trend suggested by Figure 3.

4.6 Looking Ahead

None of this is meant to suggest the work is finished — quite the opposite. Scaling validation past one hundred validators is an obvious next step, as is testing under heavier smart-contract workloads rather than the enterprise-transfer semantics used here. Formal verification of the pipeline and aggregation paths — moving beyond empirical measurement toward proof-level guarantees — would strengthen the safety argument considerably. And live geo-distributed trials, run on actual production infrastructure rather than emulated cloud conditions, would go a long way toward closing the gap between what these controlled experiments suggest and what real-world deployment would actually demonstrate (Buchman, 2016; Stathakopoulou et al., 2021). These aren't afterthoughts so much as the natural continuation of a study that, by design, aimed to be a careful first step rather than a final word.

5. Conclusion

Stepping back from the details, this study suggests something fairly modest but still worth saying plainly: classical Byzantine fault tolerant consensus doesn't necessarily need to be abandoned to remain useful at medium enterprise scale—it can, instead, be tightened from within. Signature aggregation addressed the dominant bandwidth bottleneck within the validator range tested, while adaptive pipelining delivered real throughput gains on both hardware profiles examined, all without loosening the deterministic finality or fault-tolerance threshold that make such systems trustworthy in regulated settings. Stress-tested runs across a multi-zone cloud deployment completed without recorded failures, at least within the specific configurations explored here. These results remain bounded by validator count, workload composition, and network conditions that were emulated rather than fully global in scope. Even so, they point toward a plausible, incremental route for scaling permissioned consensus systems further—one that larger validator committees, richer transaction semantics, and genuinely live multi-site deployments would only help clarify.

References

- Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., ... Yellick, J. (2018). Hyperledger Fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the 13th EuroSys Conference* (pp. 1–15). ACM.
- Boneh, D., Lynn, B., & Shacham, H. (2004). Short signatures from the Weil pairing. In *Advances in Cryptology—ASIACRYPT 2004* (Lecture Notes in Computer Science, Vol. 3329, pp. 514–532). Springer.
- Buchman, E. (2016). *Tendermint: Byzantine fault tolerance in the age of blockchains* [Doctoral dissertation, University of Guelph].
- Castro, M., & Liskov, B. (1999). Practical Byzantine Fault Tolerance. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation* (pp. 173–186). USENIX.
- Herlihy, M. P., & Wing, J. M. (1990). Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3), 463–492.
- Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 382–401.
- Micali, S., Rabin, M., & Vadhan, S. (1999). Verifiable random functions. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science* (pp. 120–130). IEEE.
- Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>
- Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *Proceedings of the USENIX Annual Technical Conference* (pp. 305–320). USENIX.
- Pass, R., Seeman, L., & Shelat, A. (2017). Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology—EUROCRYPT 2017* (Lecture Notes in Computer Science, Vol. 10211, pp. 643–673). Springer.
- Saxena, A., et al. (2018). Towards scalable and private industrial blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts* (pp. 9–16). ACM.
- Stathakopoulou, C., David, T., & Vukolic, M. (2021). Keeping PBFT in order. In *Proceedings of the ACM Asia-Pacific Workshop on Systems* (pp. 1–8). ACM.
- Sundaramurthy, S., Garg, V. K., & Kalyanaraman, S. (2016). Revisiting Byzantine Fault Tolerance. In *Proceedings of the IEEE International Conference on Distributed Computing Systems* (pp. 428–437). IEEE.
- Swanson, L., Libert, B., & Naya-Plasencia, M. (2019). Signature aggregation from symmetric key cryptography (IACR Cryptology ePrint Archive, Report 2019/1175). <https://eprint.iacr.org/2019/1175>
- Tanenbaum, A. S., & Van Steen, M. (2017). *Distributed systems: Principles and paradigms* (3rd ed.). Pearson.
- Yin, M., Malkhi, D., Reiter, M. K., Golan Gueta, G., & Abraham, I. (2019). HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the ACM Symposium on Principles of Distributed Computing* (pp. 347–356). ACM.