Differential Load Modeling and Multiobjective Optimization of BFT Consensus Parameters in Hyperledger Besu

Oleksandr Khoshaba[0000−0001−5375−6280]

Vinnytsia National Technical University, Vinnytsia 21021, Ukraine
Oleksandr.Khoshaba@gmail.com
http://www.vntu.edu.ua

Abstract. The article proposes a mathematical model for analysing the performance of a Hyperledger Besu-based permissioned blockchain network under a Byzantine Fault Tolerant (BFT) consensus protocol. Using a system of differential equations characterised by a time-varying transaction arrival rate $\lambda(t)$, we derive conditions for stable network operation, showing in particular that the system remains stable (i.e., avoids overload) if $\lambda < \mu$, where μ denotes the processing rate. Building on these theoretical foundations and formal results (lemmas and theorems), we formulate a multiobjective optimisation problem for tuning key consensus parameters (such as block interval and consensus timeout). Three performance criteria are considered: block finalisation latency (F_1), throughput (F_2), and the deviation of validator participation from a target level (F_3). The formulation seeks to minimise latency and participation deviation while maximising throughput, under a quorum constraint requiring that the fraction of participating validators $p(x)$ meets a threshold $q(n)$ (e.g., at least two-thirds of the n validators). Solving this problem yields concrete recommendations for optimal configuration, indicating that an ideal participation target p^* lies between 0.85 and 0.95 and that a moderate number of validators provides the best performance balance. The results offer practical guidance for network configuration and represent a novel integration of stability analysis with multiobjective optimisation in the context of BFT consensus protocols.

Keywords: Hyperledger Besu, IBFT consensus, QBFT, permissioned blockchain performance, differential load modelling, multiobjective optimisation, Pareto frontier, quorum-aware control

1 Introduction

Permissioned blockchain platforms based on Byzantine Fault Tolerant (BFT) consensus, such as Hyperledger Besu, are increasingly used for mission-critical, low-latency applications in finance, supply chains, and public services. Their performance hinges on protocol and system parameters that must be tuned under realistic, time-varying workloads. In practice, operators face coupled trade-offs

among *latency* (time to finality), *throughput* (transactions per second), and *validator participation* (security margin above quorum). Ad-hoc tuning is brittle: pushing for higher throughput can trigger backlog growth and unpredictable delays, while enforcing maximal participation often degrades responsiveness without proportional security gains. A principled, model-based approach is needed to (i) explain stability limits under dynamic load and (ii) expose Pareto-optimal operating points subject to quorum constraints.

Problem and approach. We study the control of Hyperledger Besu (IBFT 2/ QBFT) parameters under dynamic transaction generation. First, we develop a differential-equation model for the mempool/processing pipeline driven by a time-varying arrival rate $\lambda(t)$, and we derive verifiable stability conditions in terms of the effective service rate μ. Second, we formalize the configuration problem as a multiobjective program over the parameter vector $x = (x_1, x_2)$ (e.g., block interval and consensus timeout) with three criteria: block-finalization latency F_1, throughput cost $F_2 = -\text{TPS}$, and participation deviation $F_3 = |p(x) - p^*|$, under the BFT quorum constraint $p(x) \geq q(n) = (\lfloor 2n/3 \rfloor + 1)/n$. We analyse Pareto frontiers across validator-set sizes n and use Pareto hypervolume to aggregate performance over n.

Contributions.

(i) A load model with *dynamic* inputs (step/exponential/gaussian/pulses) and provable stability: the system is stable with bounded backlog precisely when $\lambda < \mu$; sustained $\lambda > \mu$ causes backlog divergence with throughput saturation at μ.

(ii) A quorum-aware, multiobjective *control* formulation that couples performance goals with the safety requirement $p(x) \geq q(n)$.

(iii) A data-driven criterion (Pareto hypervolume) to *select the validator-set size* and to compare configurations across n.

(iv) Design guidelines indicating that mid-range n and high-but-not-maximal participation targets $p^* \in [0.85, 0.95]$ yield robust latency–throughput–participation trade-offs.

Relation to prior work. Empirical benchmarking and steady-state queueing models explain saturation phenomena but typically lack a dynamic view and an integrated treatment of quorum-aware control. Our framework unifies (a) dynamic stability analysis under $\lambda(t)$ with (b) multiobjective optimisation constrained by BFT quorum, thereby connecting theory to deployable tuning rules for Besu-based consortia.

Notation (key symbols).
$\lambda(t)$ – arrival rate (TPS);
μ – effective service rate;
$x = (x_1, x_2)$ – tunable parameters (block interval, timeout);
n – number of validators;
$q(n)$ – quorum fraction;
$p(x)$ – realized participation;
p^* – target participation;

TPS – achieved throughput;
F_1, F_2, F_3 – objective functions.

Organization. We first review related work. We then present the differential load model and derive stability results. Next, we formulate the quorum-aware multiobjective optimisation, analyse Pareto trade-offs across validator-set sizes, and discuss implications for configuration. Finally, we outline future research directions and conclude.

2 Related Work

2.1 Blockchain Load Modeling

Blockchain performance under varying load has been extensively studied through both empirical benchmarking and analytical modeling. Fan *et al.* [6] provide a systematic survey of performance evaluation methods, highlighting how benchmarking tools and simulations are used alongside stochastic models to assess throughput and latency. For example, Dinh *et al.* introduced *BlockBench*, a widely-used framework for empirically benchmarking private blockchain platforms.

Using such tools, researchers have quantified how system throughput scales with increasing transaction rates or network size. Pongnumkul *et al.* [14] and Nasir *et al.* [13] measured that permissioned blockchains (e.g., Ethereum Clique and Hyperledger Fabric) reach a saturation point where throughput plateaus and transaction latencies grow rapidly under high workload, due to consensus and I/O bottlenecks. In particular, recent work on Hyperledger Besu by Fan *et al.* [7] showed that fundamental blockchain parameters like block interval and block size strongly dictate throughput and latency, and that suboptimal settings can drastically underutilize network. These empirical insights underline the need for accurate load modeling.

In parallel, analytical models have been developed to capture the performance of blockchain transaction processing pipelines. Queueing network models and Petri nets allow researchers to estimate how varying arrival rates and configuration parameters impact system metrics. Sukhwani *et al.* [18] proposed a hierarchical queueing model of Hyperledger Fabric, breaking down transaction flow into ordering, validation, and commit stages to predict throughput and latency. Similarly, Yuan *et al.* [22] applied generalized stochastic Petri nets (GSPN) to model a Fabric network, enabling calculation of latency distributions under different workload intensities.

These models have confirmed, for instance, that increasing block size yields higher throughput but also longer queuing delays, whereas shortening block time can lower latency at the cost of higher protocol overhead. Xu *et al.* [20] and Jiang *et al.* [9] further refined performance models to account for resource constraints (CPU, I/O) and application-level factors. Overall, both empirical and modeling studies concur that careful load characterization is critical: it reveals the existence of performance ceilings and helps identify bottlenecks (e.g., transaction execution or state update costs) that inform system optimization.

2.2 BFT Consensus Performance

Byzantine Fault Tolerant (BFT) consensus protocols are central to permissioned blockchain networks, and their performance characteristics have been a focus of much research. Classical BFT algorithms like PBFT (Practical Byzantine Fault Tolerance) achieve consensus in the face of faulty or malicious nodes but incur high communication costs that scale quadratically with the number of replicas [4]. Castro and Liskov's PBFT protocol requires multiple rounds of all-to-all message exchanges for each block, leading to significant latency as the validator count grows. Vukolic [19] contrasted this with proof-of-work, noting that while BFT avoids PoW's probabilistic delays, its throughput advantage diminishes beyond dozens of nodes due to coordination overhead.

Quantitative modeling by Sukhwani *et al.* [17] confirmed that PBFT's transaction latency rises sharply with each additional replica, reflecting the $O(n^2)$ message complexity in a network of n validators. This inherent scalability limit means that standard BFT consensus is practical only for relatively small consortium sizes.

To mitigate these issues, numerous improvements to BFT consensus have been proposed, especially in the blockchain context. One prominent example is the HotStuff protocol by Yin *et al.* [21], which achieves linear communication complexity. HotStuff introduces a pipelined, leader-based approach that reduces the costly broadcast steps per decision, allowing the system to reach consensus with significantly lower messaging overhead. This innovation enables networks of tens or even a hundred nodes to agree with latencies comparable to earlier protocols operating at much smaller scale. Another line of work has focused on tailoring BFT to permissioned blockchains: Hyperledger Fabric's design, for instance, decouples transaction ordering into a modular service.

Sousa *et al.* [16] implemented a BFT ordering service in Fabric using the BFT-SMaRt library, demonstrating that a network of 4-7 validating peers can sustain thousands of transactions per second by optimizing the consensus pipeline (batching, speculative execution, etc.) while maintaining Byzantine fault tolerance. In Ethereum-based consortia, the Istanbul BFT (IBFT) family of protocols has been adopted.

However, early versions of IBFT had liveness and safety issues under certain network conditions. Saltini and Hyland-Wood's IBFT 2.0 [15] addressed these by refining the view-change mechanism to ensure the protocol remains live and safe in eventually synchronous networks. The performance of IBFT-style consensus (as used in Hyperledger Besu and Quorum) is generally similar to PBFT for moderate node counts, with recent experiments indicating it can handle up to about 10-15 validators without significant throughput loss. For example, Fan *et al.* [7] report that Besu's QBFT (a variant with quadratic voting) maintained stable transaction throughput up to 14 validators, after which latency began increasing notably.

In summary, the state of the art in BFT consensus performance has advanced via protocol optimizations (e.g., HotStuff's responsiveness) and system architecture (modularizing consensus), yet the fundamental trade-off between

fault tolerance and scalability remains: BFT networks achieve low-latency finality in small to medium clusters, but careful design is required to extend this performance to larger scales.

2.3 Multiobjective Optimization in Networked Systems

Designing and tuning distributed ledgers and networks often requires balancing multiple conflicting performance objectives. In networked systems (including blockchain networks), one must simultaneously consider metrics such as throughput, latency, fault tolerance, security, and resource usage. This naturally leads to a multiobjective optimization problem, where improving one aspect of the system may degrade another. For instance, in blockchain settings, increasing the block size or block rate can raise throughput but also elevates propagation delays and the risk of forks or stale blocks, thereby worsening confirmation latency or consistency.

Similarly, adding more validator nodes strengthens fault tolerance but introduces extra communication overhead, harming transaction latency. A recent performance study of Hyperledger Besu explicitly highlights such tensions: shortening the block interval beyond a certain point yielded marginal throughput gains but disproportionately higher latencies as the network struggled to propagate blocks in time [7]. To navigate these trade-offs, researchers employ Pareto analysis to identify "sweet spots" that offer a good compromise between objectives, rather than optimizing a single metric in isolation.

A variety of optimization techniques have been applied to find Pareto-optimal configurations in networked systems. Evolutionary algorithms are a popular choice for multiobjective problems, as they can evolve a diverse set of solutions approximating the Pareto frontier. The NSGA-II algorithm by Deb *et al.* [5] is a seminal example that introduces elitism and fast non-dominated sorting to effectively explore trade-off solutions. Such algorithms have been used in communication networks to jointly optimize, for example, latency and reliability, or throughput and energy consumption.

In the blockchain domain, Hang and Kim [8] proposed a methodology to tune Hyperledger Fabric parameters (e.g., endorsement policy, batch size) to maximize performance while satisfying constraints, effectively treating it as a multi-criteria optimization task. Machine learning approaches are also emerging: Liu *et al.* [12] employ deep reinforcement learning to automatically adjust blockchain system parameters in an Industrial IoT setting, with a reward function capturing multiple objectives (transaction latency, energy cost, etc.). Their approach learns policies that significantly improve the throughput-versus-delay balance compared to static configurations.

Likewise, Zhang et al. [23] propose a genetic-algorithm-based node-selection scheme to accelerate PBFT in consortium blockchains, improving consensus efficiency under realistic deployments. These works illustrate that no single setting is universally optimal – instead, the goal is to provide a set of Pareto-efficient options and then select an operating point based on the application's priorities (for example, favoring lower latency for financial exchanges or higher throughput

for high-volume data logging). Multiobjective optimization frameworks and tools are thus increasingly important for architects of networked and blockchain systems, enabling systematic exploration of the configuration space under multiple constraints and objectives.

2.4 Pareto Analysis and Quorum-Aware Control

Beyond offline optimization, recent research has turned to runtime adaptation strategies that are *quorum-aware*, meaning they account for the consensus quorum requirements when adjusting system behavior. In Byzantine consensus, a quorum (typically a super-majority of nodes) must agree on each decision; thus any control mechanism must maintain this fundamental threshold to preserve safety. Pareto analysis is used to understand the trade-off surface between performance and fault tolerance. One key trade-off is the size of the validator set: a larger quorum increases reliability (able to tolerate more faulty nodes) but generally decreases throughput and increases latency due to extra communication.

Leduc *et al.* [10] address this by proposing a self-adaptive BFT consensus framework (DRAFTEE) that dynamically regulates the number of active validators in response to workload fluctuations. The system can temporarily operate with a smaller set of validators to boost throughput under high load, and later scale out to a larger quorum to reinforce security, all while ensuring the quorum never drops below the minimum required for Byzantine safety. This quorum-aware control approach essentially performs a continuous Pareto optimization: it finds a near-optimal point between maximum performance and required fault tolerance at any given time, rather than statically choosing a one-size-fits-all configuration. By monitoring network conditions and transaction demand, the controller in DRAFTEE can, for example, deactivate some validators (with their consent) to streamline consensus when throughput is paramount, then seamlessly reincorporate them when the system needs to harden against potential faults or when the load subsides.

Another line of work focuses on adaptive adjustment of consensus protocol parameters (such as leader timeout, quorum thresholds, or message batching) to approach Pareto-optimal performance under varying conditions. Berger *et al.* [2] introduce the Mercury protocol, which leverages a dual-threshold quorum mechanism to minimize consensus latency on a geo-distributed network. Mercury can form "compact quorums" with a lower resilience threshold (fewer nodes) to order transactions faster when it detects that only a small fraction of nodes are faulty, but it retains the ability to revert to the full resilient quorum (the classic $2f + 1$ of $3f + 1$ model) if faults increase.

This adaptive mode enables significantly lower commit latencies (sub-0.5 s globally) while still upholding standard BFT safety and liveness guarantees. The result is a family of operating points on the latency-fault-tolerance Pareto frontier: in normal cases with all nodes honest and responsive, the system opts for the extreme low-latency point (sacrificing some excess fault tolerance temporarily), whereas under adversarial conditions it shifts toward the high-resilience point.

One authoritative source on dynamic quorum-aware control in Byzantine Fault Tolerant (BFT) blockchain networks is the work by Li et al. on DynaNet [11]. This paper introduces a dynamic membership (quorum adaptation) framework for BFT consensus, allowing nodes to join or leave the network within a single consensus round. It extends existing protocols (demonstrated with a HotStuff variant called D-HotStuff) to support adaptive quorum selection and membership changes, and it evaluates the performance impact on throughput and latency. This makes it highly relevant for enhancing HotStuff/IBFT/PBFT-style protocols with dynamic quorum control.

Overall, Pareto analysis in these contexts guides the design of controllers that can safely navigate the trade-offs inherent in distributed consensus. By being quorum-aware, such control strategies ensure that any performance gain (e.g., higher throughput or lower latency) is achieved without violating the core requirement that a sufficient quorum of correct nodes agrees on every decision. This fusion of control theory and consensus algorithm design marks an important evolution in blockchain network management, promising systems that are not only performant and robust, but also adaptive to the demands of diverse operating environments.

3 Mathematical Model

Consider a permissioned blockchain network (such as one running Hyperledger Besu with the IBFT 2.0 consensus protocol [15]), where performance and operational stability depend on the choice of specific block generation parameters. We denote the decision vector of controllable protocol parameters as:

$$x \;=\; (x_1, x_2) \;\in\; X \subset \mathbb{R}^2 \;, \tag{1}$$

where:

- x_1 is the block proposal interval (in seconds, often called `blockperiodseconds`), which defines the nominal time between consecutive block proposals in the network;
- x_2 is the consensus round timeout (in seconds, `requesttimeoutseconds`), specifying the maximum duration allowed for the validators to achieve consensus on a proposed block in a given round.

We assume a network of n validators using a Byzantine fault-tolerant consensus. For simplicity, we focus on normal operating conditions without faults or prolonged network delays (we discuss the role of x_2 later). The following axioms summarise the basic assumptions of our model:

Axiom 1 *Transactions arrive in the network according to a random process with a constant average rate λ (transactions per second). For tractability, we assume the arrival process is Poisson with rate $\lambda > 0$.*

Axiom 2 *The consensus protocol operates in a stable regime with no Byzantine faults causing view changes. In particular, each block is successfully finalised within a single consensus round (i.e., no timeout event requiring a new round). Thus, under normal conditions, the time to finalise a block is effectively x_1 seconds.*

Let $N(t)$ denote the number of pending transactions (transactions waiting in the transaction pool or mempool) at time t. This value $N(t)$ represents the *current load* on the network. When new transactions arrive, they increase $N(t)$, and when a block is created, some transactions are removed from the pool (committed to the blockchain), decreasing $N(t)$. We denote by B the maximum number of transactions that can be included in a single block (the block capacity, determined by block gas limit or similar constraints). We define the *service rate* μ of the network as the maximum rate at which transactions are processed (included in blocks) under continuous heavy load:

$$\mu \;=\; \frac{B}{x_1} \quad \text{(transactions per second)} \;, \tag{2}$$

assuming that every block is filled B when there are enough pending transactions.

Using the above definitions, we can model the time evolution of $N(t)$ by a differential equation. When $N(t)$ is large (the system is busy processing transactions continuously), the rate of decrease in $N(t)$ due to block confirmations is μ. The rate of increase in $N(t)$ due to arrivals is λ.

Thus, while $N(t) > 0$, we can write:

$$\frac{dN(t)}{dt} \;=\; \lambda \,-\, \mu \;, \tag{3}$$

with initial condition $N(0) = N_0 \geq 0$.

Equation (3) follows directly from Axiom 1 and the definition of μ in (2). It represents a fluid approximation to the discrete queuing process of transactions. We note that if $N(t)$ reaches zero (i.e., the transaction pool is empty at some time), the network will produce either an empty block or pause block production until a new transaction arrives. In such idle periods, $N(t)$ stays at zero (since no negative backlog is possible). For the analysis of stability, we will primarily consider the behaviour when the system is under load (and $N(t)$ is nonzero).

Before stating the main results, we formalise what it means for the network to be stable in terms of the load $N(t)$ as Definition (1):

Definition 1. *The network is* stable *(concerning the transaction load) if the number of pending transactions remains bounded over time. Formally, the system is stable if there exists some finite K such that $N(t) < K$ for all $t \geq 0$ (given some initial $N(0) = N_0$). If $N(t)$ can grow without bound as $t \to \infty$, the system is* unstable *or overloaded.*

4 Stability Analysis

Using the model described above, we now derive conditions for stability. First, we solve the differential equation (3) under the assumption of continuous load (i.e., $N(t) > 0$).

Lemma 1. *Consider the differential equation* (3) *with* $N(0) = N_0 \geq 0$. *While* $N(t) > 0$, *the solution is given by*

$$N(t) \;=\; N_0 + (\lambda - \mu)\, t \;.$$

In particular, if $\lambda \neq \mu$, $N(t)$ *evolves linearly.*

Proof. Equation (3) is a first-order linear ordinary differential equation with constant coefficients. Its general solution is $N(t) = C+(\lambda-\mu)t$ for some constant C. Applying the initial condition $N(0) = N_0$, we get $C = N_0$. Thus $N(t) = N_0 + (\lambda - \mu)t$. This linear solution is valid as long as $N(t)$ remains positive.

Using this result, we can characterise the stability of the system by examining the sign of $\lambda - \mu$.

Theorem 1. ***(Stability Condition)*** *Under Axioms 1 and 2, the blockchain network is stable in the sense of Definition 1 if and only if* $\lambda < \mu$. *Equivalently, stability requires that the average transaction arrival rate is lower than the processing rate (throughput) of the network. Conversely, if* $\lambda \geq \mu$, *the system is unstable, meaning the pending transaction count* $N(t)$ *will grow without bound (or, in the borderline case* $\lambda = \mu$, *remain at a constant positive level).*

Proof. If $\lambda < \mu$, then $\lambda - \mu$ is negative. By Lemma 1, the solution for $N(t)$ while $N(t) > 0$ is $N(t) = N_0 + (\lambda - \mu)t$. Since $\lambda - \mu < 0$, $N(t)$ is a decreasing linear function. Starting from N_0, $N(t)$ will eventually decrease to 0 at time $t^* = N_0/(\mu - \lambda)$. After that time, $N(t)$ cannot become negative; in reality, once the queue empties ($N = 0$), it will remain at zero until a new transaction arrives, and thereafter $N(t)$ might fluctuate but will never exceed N_0 again if $\lambda < \mu$. Therefore, $N(t)$ stays bounded (in fact, it drops to 0 and stays $\leq N_0$), satisfying the stability condition.

If $\lambda = \mu$, then $N(t) = N_0$ for all t (from Lemma 1). In this case, the backlog neither grows nor decays; it remains constant (assuming $N_0 > 0$). This situation is a knife-edge: the system can be considered marginally stable (the backlog is bounded by N_0), but does not clear. Any infinitesimal increase in arrival rate would cause growth. In practice, $\lambda = \mu$ is not sustainable due to random fluctuations, so it is treated as the threshold of instability.

If $\lambda > \mu$, then $\lambda - \mu > 0$. By Lemma 1, while $N(t) > 0$, we have $N(t) = N_0 + (\lambda - \mu)t$, which grows without bound as t increases. Once $N(t)$ becomes very large, the model continues to predict further increase (the network continuously has a backlog, and each interval adds $(\lambda - \mu)x_1$ new transactions net of processing). Thus, $N(t)$ will diverge to infinity, and the system is unstable. (In practice, an ever-growing $N(t)$ means transaction queues lengthen without bound and transaction latency becomes unbounded.) □

Theorem 1 implies that to ensure stability, the network's block processing capacity must exceed the incoming transaction rate. In terms of the utilisation factor (often denoted $\rho = \lambda/\mu$ in queueing theory), the condition is $\rho < 1$. If this condition is satisfied, the pending transactions will not accumulate indefinitely; instead, the system will be able to catch up with incoming transactions, eventually emptying the queue during periods of lower activity. On the other hand, if $\rho \geq 1$, the backlog will grow and the transaction confirmation delays will become arbitrarily large, which is undesirable for any practical system.

It is worth noting that the consensus round timeout x_2 does not explicitly appear in the stability condition. This is because under Axiom 2 we assumed that x_2 is sufficiently large to avoid triggering in regular operation. If network delays are such that consensus frequently requires multiple rounds (i.e., the actual block time becomes larger than x_1), then μ would effectively be lower than B/x_1 and the stability condition would become more stringent. In other words, an inappropriately short x_2 or insufficient network communication can reduce throughput, requiring an even lower arrival rate for stability. A more detailed model could incorporate the probability of needing multiple consensus rounds as a function of x_2 and network conditions, but that is beyond the scope of this theoretical analysis.

5 Multiobjective Optimisation under Quorum Constraints

5.1 Problem Statement

We formalise the configuration tuning problem as a multiobjective optimisation under a quorum constraint. Let $x = (x_1, x_2) \in X \subset \mathbb{R}^2$ be the vector of protocol parameters to be optimized. In our case, x_1 represents the block proposal interval (in seconds) and x_2 the consensus round timeout (in seconds), each bounded within operationally feasible ranges (with $x_2 \leq x_1$ to ensure the timeout does not exceed the block interval). We define three objective functions to be *minimized*: $F_1(x)$, $F_2(x)$, and $F_3(x)$, corresponding to *latency*, *throughput cost*, and *participation deviation*, respectively. Specifically:

- $F_1(x)$ (Latency) – the average block finalisation time (or related latency metric) under parameters x. Lower values of F_1 are preferable.
- $F_2(x)$ (Throughput Cost) – a cost associated with transaction throughput. We define $F_2(x)$ such that it decreases as throughput (TPS) increases; for example, one can take $F_2(x) = -\mathrm{TPS}(x)$ so that minimizing F_2 equates to maximizing throughput.
- $F_3(x)$ (Participation Deviation) – a measure of deviation from the desired validator participation level. Let $p(x)$ denote the fraction of validators actively participating in consensus under configuration x. We fix a target participation level $p^* \in (0, 1]$ (e.g. p^* may be 0.90, representing a design goal of 90% participation). Then $F_3(x)$ is defined as the deviation from this target,

$F_3(x) = |\, p(x) - p^* \,|$, which attains its minimum value 0 when the observed participation equals the target p^*.

The optimization problem can then be stated as finding configurations x that simultaneously minimize the objective vector $F(x) = (F_1(x),\, F_2(x),\, F_3(x))$, subject to maintaining consensus safety. In particular, the "quorum constraint" requires that the participation $p(x)$ never falls below the minimum quorum ratio required by the consensus protocol:

$$\min_{x \in X} \ \big(F_1(x),\, F_2(x),\, F_3(x)\big) \qquad \text{s.t.} \qquad p(x) \ \geq \ q(n) \ = \ \frac{\lfloor 2n/3 \rfloor + 1}{n}\,. \tag{4}$$

Here n denotes the total number of validating nodes, and $q(n)$ is the quorum threshold (the fraction of validators that must participate to safely finalise a block, given by the standard 2/3 rule for Byzantine fault tolerance). The feasible set X is assumed to be constrained such that X is closed and bounded (e.g. by protocol limits and practical considerations on x_1, x_2), and configurations not satisfying $p(x) \geq q(n)$ are considered infeasible. The goal is not to find a single "optimal" x (since the objectives conflict) but rather to characterise the set of *efficient* or *Pareto optimal* solutions that best trade off latency, throughput, and participation within these constraints.

5.2 Axioms and Definitions

To enable a rigorous analysis, we first lay out formal assumptions (axioms) about the optimisation model and define key concepts from multiobjective optimisation theory:

Axiom 3 (Feasibility) *The decision space $X \subset \mathbb{R}^2$ (set of all allowable parameter configurations) is nonempty, compact, and closed, as delimited by the system's operational constraints. In particular, X includes only those (x_1, x_2) satisfying required bounds (such as timing and logical constraints) and the consensus stability condition $p(x) \geq q(n)$.*

Axiom 4 (Continuity) *Each objective function $F_i(x)$ for $i = 1, 2, 3$ is a continuous (preferably differentiable) function on the feasible region X. Moreover, $p(x)$ is continuous in x so that small changes in parameters do not cause abrupt drops in participation.*

These axioms ensure the optimisation problem is well-posed: a compact feasible set and continuous objectives guarantee the existence of solutions and allow the use of standard analytical tools. We now recall the formal definitions of Pareto-optimality:

Definition 2 (Pareto Dominance). *For two candidate configurations $x^a, x^b \in X$, we say x^a* Pareto-dominates *x^b (denoted $x^a \prec x^b$ in objective space) if and only if $F_i(x^a) \leq F_i(x^b)$ for all objectives $i \in \{1, 2, 3\}$,* and *$F_j(x^a) < F_j(x^b)$ for*

at least one objective j. In other words, x^a is at least as good as x^b in every criterion and strictly better in at least one.

$$F(x^a) \prec F(x^b) \iff \begin{cases} \forall\, i \in \{1,2,3\} :\ F_i(x^a) \le F_i(x^b), \\ \exists\, j \in \{1,2,3\} :\ F_j(x^a) < F_j(x^b)\,. \end{cases} \tag{5}$$

Definition 3 (Pareto Optimality). *A configuration $x^* \in X$ is* Pareto optimal *(or* Pareto-efficient*) if no other feasible solution dominates it. Formally, x^* is Pareto optimal if and only if there does* not *exist any $x \in X$ such that $F(x) \prec F(x^*)$. Equivalently:*

$$\nexists\, x \in X :\ F(x) \prec F(x^*)\,. \tag{6}$$

In other words, any attempt to improve one objective of x^* would necessarily degrade at least one of the other objectives.

Definition 4 (Pareto Frontier). *The set of all objective vectors corresponding to Pareto-optimal solutions is called the* Pareto frontier *(or Pareto front). In our context, the Pareto frontier P can be characterized as*

$$P \;=\; \Big\{\, F(x^*) = \big(F_1(x^*), F_2(x^*), F_3(x^*)\big) \;\mid\; x^* \text{ is Pareto-optimal in } X \,\Big\}\,. \tag{7}$$

This frontier represents the boundary of achievable trade-offs among latency, throughput cost, and participation deviation. Decision-makers must select one solution from P by considering higher-level preferences or external criteria, since all Pareto-optimal points are non-dominated (i.e. equally valid trade-offs).

5.3 Properties and Theorems of the Pareto Set

Given the above formulation, we derive several fundamental properties of the Pareto-optimal set (efficient set) for our multi-criteria problem.

First, under our continuity and compactness assumptions, we are assured that at least one Pareto-optimal configuration exists:

Theorem 2 (Existence of Pareto-Optimal Solutions). *Under Axioms 3 and 4, the set of Pareto-optimal solutions is non-empty. Moreover, the Pareto-optimal set (and the corresponding Pareto frontier P in objective space) is compact.*

Proof (Sketch of Proof.). Because X is compact and each $F_i(x)$ is continuous on X, any continuous scalarization of the vector objective $F(x)$ attains a minimum on X. For example, one may consider a weighted sum or weighted max of the objectives; by the Pareto efficiency of scalarization minima, each such minimiser is Pareto-optimal for the multiobjective problem. This guarantees that at least one Pareto-optimal point exists. Furthermore, the set of Pareto-optimal solutions can be shown to be closed (hence compact, since it is a closed subset of the compact set X) under mild regularity conditions, due to the lower semicontinuity of the dominance relation.

Next, we characterise Pareto-optimal solutions via first-order optimality conditions. Intuitively, at a Pareto-optimal point x^*, there should be no feasible direction in which all objectives improve simultaneously. If the objectives are differentiable, this intuition can be formalised by conditions on the gradients of F_1, F_2, F_3 at x^*. We adopt a multiobjective version of the Karush–Kuhn–Tucker (KKT, [1,3]) conditions:

Theorem 3 (First-Order Pareto Optimality). *Suppose that each $F_i(x)$ is continuously differentiable in a neighbourhood of $x^* \in X$, and let X be regular enough to admit a well-defined tangent (and normal) space at x^* (for example, X is convex, or x^* is not on a sharp boundary corner). If x^* is Pareto-optimal, then there exist nonnegative scalars $(\alpha_1, \alpha_2, \alpha_3)$, not all zero, such that the following condition holds:*

$$\alpha_1 \nabla F_1(x^*) + \alpha_2 \nabla F_2(x^*) + \alpha_3 \nabla F_3(x^*) \ \in \ N_X(x^*)\,, \tag{8}$$

where $N_X(x^*)$ denotes the normal cone of the feasible set X at x^*. In particular, if x^* lies in the interior of X (away from boundaries), then $\sum_{i=1}^{3} \alpha_i \nabla F_i(x^*) = \mathbf{0}$.

Proof (Sketch of Proof.). This result is a multiobjective analogue of the KKT optimality conditions. The necessity of the condition comes from the fact that if no convex combination of the gradients can yield a zero (or a normal-to-boundary) vector, then one could find a descent direction in the tangent space that simultaneously decreases all objectives, contradicting the Pareto optimality of x^*. Conversely, if X is convex and the objectives are quasi-convex, a solution satisfying condition (8) (together with complementary slackness for any active constraints in $N_X(x^*)$) is guaranteed to be Pareto-optimal. We omit the full proof for brevity.

Corollary 1 (Sufficiency under Convexity). *If the objectives F_1, F_2, F_3 are such that F_1 and F_2 are convex functions and F_3 is convex in x (as is the case if $F_3(x) = |p(x) - p^*|$ with linear $p(x)$, or if $p(x)$ is concave and we consider $-p(x)$), then any solution $x^* \in X$ satisfying the first-order condition* (8) *is* globally *Pareto-optimal. In other words, under convexity assumptions, the first-order necessary condition becomes sufficient.*

Proof (Remark.). The convexity of all objective functions ensures that the multiobjective problem does not have local Pareto optima that are not global. In such cases, any x^* that satisfies the gradient condition (and feasibility) cannot be dominated by any other point in X, guaranteeing its Pareto optimality. This aligns with the well-known result that for convex multiobjective problems, any Pareto-critical point is globally Pareto-optimal.

The above theorems establish that the Pareto-optimal set is well-defined and can be characterised by analytical conditions. In practice, one may solve for Pareto-optimal configurations using techniques like weighted-sum optimisation or evolutionary multiobjective algorithms, and verify optimality via these conditions. The Pareto frontier P provides insight into the trade-offs: for instance, it quantifies how much latency must increase to gain a certain throughput improvement while maintaining a given participation level, and vice versa.

5.4 Conditions on p^* and Solution Stability

The choice of the target participation level p^* in the objective $F_3(x)$ plays a crucial role in determining which Pareto-optimal solution will be preferred or achieved in operation. We discuss how p^* should be set and how it affects the stability and robustness of the resulting solutions.

First, it is essential that p^* not be lower than the minimum quorum requirement. We impose

$$p^* \;\geq\; q(n) \;=\; \frac{\lfloor 2n/3 \rfloor + 1}{n}\,, \tag{9}$$

so that the target participation level is at least the protocol's safety threshold. If one were to set p^* equal to $q(n)$ exactly, the optimisation would treat any participation above the bare minimum quorum as unnecessary (since $F_3 = |p(x) - q(n)|$ would achieve 0 whenever $p(x) = q(n)$). In such a case, Pareto-optimal solutions might drift toward configurations that barely satisfy quorum, leaving no safety margin for unexpected validator delays or failures. Such solutions are *fragile*: any slight drop in participation (due to, say, a slow or crashed validator) could violate the quorum constraint and halt consensus.

On the other hand, setting p^* overly high (close to 1) forces the system to aim for nearly 100% participation at all times. While this maximises fault tolerance, it can unduly penalise otherwise efficient configurations. Requiring full participation means that the block production interval x_1 and timeout x_2 might need to be increased to allow even the slowest validators always to contribute, which in turn can degrade latency and throughput. In extreme cases, an excessively large p^* could lead to solutions that sacrifice significant throughput or latency gains for only marginal increases in participation (especially in large networks where communication delays might make 100% participation impractical).

Guideline 1 (Quorum Safety Margin). For a stable and robust configuration, choose a target participation p^* that provides a *safety margin* above the bare quorum threshold. That is, $p^* = q(n) + \delta$ for some $\delta > 0$ reflecting tolerable fault levels. This ensures that a Pareto-optimal solution achieving $p(x) \approx p^*$ will maintain consensus even if a fraction δ of validators (approximately $\delta \cdot n$ nodes) are slow or fail to respond. In practice, network operators might select p^* in the range of 85–95% (i.e., 0.85–0.95 fraction of validators), which provides a buffer above $q(n) \approx 66.7\%$ for large n. Such a choice was observed to yield robust trade-offs in our experiments.

Guideline 2 (Performance–Stability Balance). The target p^* should be balanced against performance objectives. A moderate p^* (significantly above $q(n)$ but not equal to 1) often produces the best compromise: it keeps $F_3(x)$ low (participation close to target) without overly constraining F_1 and F_2. In contrast, if p^* is too low, the resulting optimal x may run the system "hot," with minimal slack for errors (high throughput and low latency but risky consensus stability). If p^* is too high, the system runs "conservative," with excellent fault tolerance but possibly suboptimal throughput/latency.

In summary, the parameter p^* can be viewed as a design knob that shifts the Pareto frontier by prioritising validator participation to a desired level. By ad-

justing p^*, one effectively moves along the trade-off surface between performance and fault tolerance: - Lowering p^* expands the feasible region for performance (allowing more aggressive x_1, x_2 settings) but increases the risk that the system operates with minimal quorum (reduced stability margin). - Raising p^* contracts the feasible region to more conservative configurations (ensuring higher participation at all times) but may incur higher latency or lower throughput.

For the Hyperledger Besu network under study, we identify that choosing p^* in an intermediate range (around 0.85–0.95 as noted) yields *stable Pareto-optimal solutions*: these configurations tolerate typical network delays and node failures without violating quorum, and simultaneously achieve favourable latency and throughput. In effect, such p^* values align the network's operating point with a regime of diminishing returns: beyond this range, forcing even higher participation yields little additional resilience yet comes at a steep performance cost (due to communication overhead and synchronization delays); below this range, small performance gains come at the expense of drastically reduced safety margins.

Overall, by respecting the quorum constraint and judiciously selecting p^*, we ensure that the multi-criteria optimisation results in configurations that are not only Pareto-efficient in theory but also practical and stable for real-world blockchain deployment. This completes the formal problem formulation and theoretical underpinning for multiobjective tuning of Hyperledger Besu parameters under Byzantine quorum requirements.

Nomenclature

$\lambda(t)$ – Instantaneous transaction arrival rate (TPS).
μ – Effective service (processing) rate (TPS).
B – Max number of transactions per block (capacity).
$x = (x_1, x_2)$ – Tunable consensus parameters (block interval x_1, timeout x_2).
n – Validator set size.
$q(n)$ – Quorum fraction $(\lfloor 2n/3 \rfloor + 1)/n$.
$p(x)$ – Realized validator participation under configuration x.
p^* – Target participation level.
TPS – Achieved throughput (transactions per second).
HV – Pareto hypervolume (aggregate quality indicator).
F_1, F_2, F_3 – Objectives: latency, throughput cost ($-$TPS), participation deviation $|p(x) - p^*|$.

6 Novelty and Contribution

This work advances the state of the art in performance analysis and control of permissioned blockchains (Hyperledger Besu, IBFT/QBFT) by unifying *dynamic* load modelling, *quorum-aware* multiobjective optimisation, and *deployable* design rules. In contrast to empirical-only benchmarking and steady-state queueing approximations, our framework provides provable stability guarantees

under time-varying inputs. It turns them into actionable tuning guidelines under Byzantine quorum constraints.

1. **Dynamic differential load model with axiomatic input classes.**
   We introduce a continuous-time model driven by a time-varying arrival rate $\lambda(t)$ (TPS) and formalise admissible load profiles via axioms (piecewise-smooth steps, exponential ramps, Gaussian bursts).
   The queue (mempool) dynamics are captured by $\dot{N}(t) = \lambda(t) - \mu$ for $N(t) > 0$, where μ is the effective service rate determined by protocol/implementation parameters.
   This bridges synthetic benchmark design with mathematically tractable dynamics.
2. **Sharp stability boundary and transient behaviour.**
   We prove a necessary and sufficient stability condition:
   the system is stable (bounded backlog/latency) if $\lambda < \mu$;
   sustained $\lambda > \mu$ yields unbounded backlog growth with throughput saturating at μ.
   For prototypical inputs (step/exponential/Gaussian), we analytically characterise transient regimes (backlog rise/peak/drain), identifying when worst-case latency occurs and how quickly the system returns to steady operation.
3. **Quorum-aware multiobjective control of consensus parameters.**
   We pose the tuning of $x = (x_1, x_2)$ (block interval, consensus timeout) as a three-criterion program.

   $$\min_{x \in X} \big(F_1(x),\, F_2(x),\, F_3(x)\big), \quad F_1{=}\text{latency}, \;\; F_2{=}-\,\text{TPS}, \;\; F_3{=}|p(x) - p^*|,$$

   subject to the hard BFT safety constraint $p(x) \geq q(n) = (\lfloor 2n/3 \rfloor + 1)/n$.
   We establish the existence of Pareto-optimal solutions under compactness/continuity and derive first-order (multiobjective KKT-type) [1,3] necessary conditions that certify Pareto optimality of candidate configurations.
4. **Cross-n aggregation via Pareto hypervolume and validator-count selection.**
   We treat the validator set size n as a design variable and aggregate performance across n using the Pareto hypervolume. This provides a principled, data-driven criterion to select n and to compare trade-offs across heterogeneous validator committees.
5. **Robust operating regions and design rules for Besu consortia.**
   From the derived frontiers and hypervolume analysis, we identify:
   (i) *mid-range* validator committees dominate extremes (they balance coordination overhead and resilience);
   (ii) targeting high-but-not-maximal participation $p^* \in [0.85, 0.95]$ preserves a safety margin above quorum while enabling favourable latency–throughput trade-offs;
   (iii) tuning (x_1, x_2) to operate near (but below) the stability boundary $\lambda < \mu$ maximises utilisation without inducing overload.

6. **Methodological integration for synthetic benchmarking and control.**
   The framework unifies control-theoretic stability analysis with multiobjective optimization under quorum constraints, yielding a reusable blueprint to:
   (a) generate stress scenarios from admissible $\lambda(t)$ families,
   (b) certify stability/transients analytically, and
   (c) select Pareto-efficient configurations (x_1, x_2, n, p^*) for deployment.

Collectively, these contributions provide both *theoretical guarantees* (stability proofs, Pareto optimality conditions) and *practical guidance* (validator-count selection, quorum targets, parameter tuning) for configuring Hyperledger Besu networks that must meet stringent real-time performance under realistic, time-varying loads.

7 Discussion

Our results indicate that the system remains *stable* as long as the average input rate λ stays below the effective processing service rate μ of the network. In this regime ($\lambda < \mu$), every incoming request can be handled without persistent accumulation, so the backlog of pending transactions remains bounded. Consequently, the network operates in a steady state with finite queues, and transaction latencies stay manageable. This finding is intuitive in light of queueing theory. When capacity exceeds demand on average, the system can keep up with the workload, preventing uncontrolled growth of waiting transactions and ensuring consistent throughput. When the input rate exceeds processing capacity ($\lambda > \mu$), the model predicts an onset of instability characterised by unbounded backlog growth. In this overloaded regime, incoming transactions arrive faster than they can be completed, causing the queue of unprocessed requests to lengthen over time. The immediate consequence is a sharp increase in waiting times (latency) for transactions – as the backlog grows, each transaction spends longer in the queue before completion. Throughout this regime, the throughput saturates at its maximum possible value (approximately μ), but any excess load beyond μ is effectively buffered, leading to ever-increasing delays. In practical terms, if a blockchain node or network operates above its capacity, users will observe degrading responsiveness (e.g., transactions taking progressively longer to be confirmed). These results highlight the critical importance of keeping the system load under control or provisioning sufficient capacity so that λ does not persistently exceed μ.

They also imply that beyond the stability threshold, simply adding more load yields no increase in actual throughput – only a larger backlog – underscoring a hard limit on sustainable transaction processing rate. Another key insight from our study is the inherent multi-criteria trade-off between *latency*, *throughput*, and *participation* in the consensus process. It is not possible to optimise all three metrics simultaneously without compromise. For instance, achieving lower latency (faster transaction finalisation) often requires processing smaller

batches of transactions or shortening consensus rounds, which can reduce overall throughput.

Conversely, maximising throughput – by aggregating more transactions per block or increasing the block production rate – typically causes higher latencies, as the system needs more time to process larger blocks or to coordinate more frequent consensus decisions. Participation (here referring to the number of validators actively involved in making each consensus decision and the inclusivity of the process) also intertwines with these two metrics.

Broad participation (a large validator set) can increase the security and fairness of the system, but it generally incurs greater communication and coordination overhead, tending to increase latency and sometimes limit throughput. In contrast, reducing the number of participants or the frequency of their involvement can improve performance (lower latency and higher throughput) but at the cost of decentralisation and collective trust.

Our results quantify these trade-offs, reinforcing the notion that system designers must balance latency, throughput, and participation according to the priorities and requirements of a given application or use case. The role of specific consensus parameters – notably the quorum threshold and the validator count n – is evident in shaping system performance. Quorum requirements: A higher quorum (closer to n, e.g. needing almost all validators to sign off on a block) increases the confidence in each decision but also adds to latency, since the block must wait for more acknowledgements before finalisation.

In our model (absent Byzantine faults), this manifests as a longer decision time as quorum increases, which in turn can throttle throughput because fewer consensus decisions (blocks) are completed per unit time. Conversely, a lower quorum threshold (for example, just over a simple majority) allows consensus to reach decisions faster – improving latency and throughput – but with fewer participants confirming each block, which could reduce the fault tolerance margin. Validator count: Increasing the total number of validators n tends to slow down the protocol's performance due to communication complexity.

Many consensus algorithms incur message complexity that grows with n (often quadratic or worse in n for Byzantine agreement), meaning that larger networks require more time and bandwidth to coordinate each block. Thus, as n grows, we observe an increase in consensus latency and a potential drop in throughput, all else being equal. On the positive side, a larger n improves the decentralisation and diversity of the consensus process (more organisations or nodes participating), which may be desired for security and trust.

The interplay between n and quorum is also essential: for a fixed n, raising the quorum size increases latency (as noted), and for a fixed quorum percentage, increasing n also effectively raises the absolute number of signatures needed, again slowing performance. The findings suggest that there is an optimal range for these parameters depending on goals – for instance, a moderate validator committee size with a quorum that is strict enough for safety but not maximal can achieve a balanced outcome, preserving good performance while involving a broad set of validators.

Our analysis also examined the influence of the parameter p (in particular, the choice of an optimal p^*) on system stability and performance. The parameter p represents a configurable probability or fraction in the protocol (as defined in our model) that affects how aggressively the system operates – for example, it might control the rate of producing blocks or the likelihood of a validator initiating some action in each round. We find that selecting p judiciously is crucial.

An optimal value p^* emerges from the analysis, at which the system achieves maximal throughput without sacrificing stability. Setting $p = p^*$ essentially drives the system to operate at the edge of its capacity (λ approaching μ from below) but not beyond it. If one chooses a p significantly larger than p^*, the system is pushed into an unstable regime where λ effectively exceeds μ; this leads to the overload effects discussed earlier (growing backlog and latency).

On the other hand, a p much smaller than p^* keeps the system well within stable limits but under-utilises resources – throughput will remain stable but below what the system could handle, and latency, while low, could have been slightly higher in exchange for processing more transactions per unit time.

In summary, p^* can be seen as a sweet-spot operating point: it balances the desire for high throughput with the need to maintain $\lambda < \mu$. The sensitivity analysis around p^* shows that deviating from this optimum in either direction (too high or too low) incurs a cost, either in the form of stability risk or in forgone performance.

Therefore, tuning this parameter to p^* (or as close as practical) is recommended for maintaining a healthy, efficient network. Beyond theoretical analysis, these insights have practical implications for real-world blockchain deployments. Consider, for example, a permissioned Ethereum network using the IBFT consensus protocol (such as a Hyperledger Besu cluster in a consortium setting).

Our results provide guidance for configuring such a network. First, ensuring $\lambda < \mu$ translates to making sure that the transaction arrival rate into the network does not routinely exceed what the network can process in each block interval. In practice, this could mean implementing back-pressure or admission control on transaction submission when the system is nearing capacity, or scaling up node performance (or the number of nodes) to increase μ.

The latency-throughput-participation trade-offs inform choices like block interval time, block size (or gas limit), and the size of the validator pool. For instance, if a Besu-based consortium prioritises low latency for rapid transaction finality (as might be the case in a financial setting where quick settlements are crucial), the network operator might opt for shorter block intervals and a relatively small validator committee. This configuration reduces the communication overhead per block. It speeds up consensus, at the expense of processing fewer transactions per block (lower per-block throughput) and involving fewer entities in each decision (which is acceptable if the consortium is small and trusted). In contrast, if the application demands high throughput (e.g., a supply chain network processing large volumes of events), one might increase the block size or allow a slightly longer block time to accumulate more transactions, accepting a moderate increase in latency.

The number of validators n and the quorum threshold should also be chosen with an eye to these trade-offs: a large n with a high quorum gives stronger assurance that many nodes endorse each block, but might only be feasible if the expected transaction rate is modest (so that added latency is tolerable). If using our identified optimal parameter p^* in such a cluster (for example, if p governs how frequently validators propose new blocks or some other consensus aggressiveness setting), practitioners should set it to the recommended value to maximize throughput while avoiding the onset of instability (which in a real network could manifest as congested mempools and delayed confirmations).

In summary, the analytical results act as design guidelines for tuning a Besu or similar blockchain network: they help determine safe operating throughput, appropriate consensus committee size, and other protocol parameters to meet targeted performance goals without compromising network stability. Finally, it is important to recognise the limitations of our model and analysis. To obtain analytical tractability, we adopted several idealised assumptions. No Byzantine faults: We assumed that all validators follow the protocol correctly (no malicious or faulty nodes). This allowed us to focus purely on performance characteristics without considering the additional delays and protocol complexity that arise from tolerating Byzantine behaviour (e.g., extra communication steps for validation or view-change procedures when a leader fails). In a real deployment, Byzantine faults could increase latency and reduce throughput, so our performance results represent a best-case scenario.

Stable network conditions: Our model assumes a stable network with reliable and relatively uniform message delivery times (no significant network partitions or highly variable delays). In practice, network conditions can fluctuate – messages can be delayed or dropped, and temporary network slowdowns can occur, which would affect consensus timing and could cause performance to deviate from our predictions. Linear processing and fixed service rate: We treated the processing capacity μ as a constant linear rate, essentially assuming each node's throughput scales linearly with the load and that there are no internal resource contention issues.

Real systems might have non-linear performance characteristics (for example, due to multi-threading overhead, I/O bottlenecks, or caching effects), and clever engineering (such as parallel transaction processing or hardware acceleration) could improve throughput beyond the simple linear model. Our analysis does not capture these intricacies, so it may underestimate or overestimate performance in specific regimes. Lack of traffic variability: We assumed a Poisson-like steady arrival process with a well-defined average rate λ.

Actual transaction arrival patterns in deployed networks are often bursty or time-varying (for example, periods of sudden high load followed by lulls). Such variability means that even if the long-term average load is below μ, short-term bursts could temporarily exceed capacity and create backlogs before subsiding.

Our steady-state analysis does not fully reflect these transient dynamics. Due to these assumptions, the quantitative results of our model should be interpreted

with caution. They likely represent optimistic outcomes; real-world performance could be lower or require safety margins.

Nonetheless, the qualitative insights (stability conditions, trends under changing parameters, and trade-offs) remain valuable. Future work could relax some of these assumptions – for instance, introducing Byzantine behaviour in the model, or considering stochastic bursty traffic and variable network delays – to study how the system performance degrades or what additional safety buffers are needed in those more complex scenarios.

8 Scope and Limitations

We adopted several idealised assumptions in our model to ensure analytical tractability, which define the scope of this study.

No Byzantine faults: All validators are assumed to follow the protocol correctly (i.e., no malicious or faulty nodes), allowing us to focus purely on performance characteristics without accounting for additional consensus overhead (such as extra communication steps or view-change procedures to handle faulty behaviour).

Stable network conditions: The network is assumed to be stable and homogeneous, with reliable and relatively uniform message delivery times (no significant network partitions or prolonged delays). In practice, transient network slowdowns or message loss could occur, which would affect consensus timing and performance; those effects are not captured in our model.

Linear processing rate: Each node's transaction processing capacity μ is treated as a constant, linear rate. We do not model internal resource contention or non-linear performance factors (e.g., multi-threading overhead, I/O bottlenecks), so the results reflect a simplified "best-case" throughput scenario.

Steady traffic load: Transaction arrivals are assumed to follow a Poisson-like process with a well-defined average rate λ, without extreme short-term bursts. Real-world traffic can be bursty or time-varying (e.g., sudden surges of transactions followed by lulls), which our steady-state analysis does not fully capture.

Due to these deliberate scope limitations, the quantitative results and optimal configurations derived in this work should be interpreted as applying to an idealised environment. Factors beyond our assumed scope – such as malicious participants, unstable network conditions, non-linear scaling effects, or highly variable workloads – are not addressed here and could lead to different performance outcomes if present. Acknowledging these limitations helps to clarify the context of our findings and suggests avenues for future work where the model could be extended to more realistic settings.

9 Future Work

The present study establishes a theoretical framework for analysing and optimising the performance of a Hyperledger Besu blockchain network under BFT

consensus. Several promising directions exist for extending and deepening this work.

First, the analytical model can be generalised to account for *Byzantine faults* explicitly. Incorporating fault scenarios would require modelling additional message exchanges for view-change or leader replacement, and capturing the resulting effects on both throughput and latency. This extension would bridge the gap between the best-case (fault-free) performance analysed here and the more variable conditions of real-world deployments.

Second, future work should relax the assumption of *stable and homogeneous network conditions*. Investigating the model's behaviour under variable message delays, network partitions, and heterogeneous validator performance could reveal new stability boundaries and parameter trade-offs. Stochastic models for bursty or time-varying transaction arrival processes would allow for evaluating transient behaviour, including short-term overloads and recovery dynamics.

Third, the linear processing assumption (μ constant) could be replaced with a more detailed *nonlinear service model*, capturing multi-threaded execution, I/O contention, and state update costs. This would make predictions more accurate for networks operating near hardware or software bottlenecks, and could inform hardware provisioning strategies.

Fourth, the optimisation framework can be extended beyond static tuning to *adaptive control* of consensus parameters. Techniques from control theory and machine learning (e.g., reinforcement learning) could be employed to adjust block interval, timeout, or quorum thresholds dynamically in response to observed workload and network conditions. Such self-optimising mechanisms could keep the system close to Pareto-optimal operation even under fluctuating demand.

Finally, empirical validation remains a key step. Deploying the model in controlled testbeds or production-like Hyperledger Besu environments, subject to synthetic and real transaction loads, would enable calibration of model parameters and verification of predictions. The resulting data could also guide refinements to both the stability analysis and the multiobjective optimisation criteria, ultimately producing a robust and deployable decision-support tool for blockchain network operators.

10 Conclusion

We presented a rigorous, differential-equation model of transaction load for Hyperledger Besu under BFT consensus and used it to derive operational guarantees and tuning rules. Modelling the arrival process as $\lambda(t)$ and the effective service rate as μ, we proved that the system is stable (bounded backlog and latency) precisely when $\lambda < \mu$, while sustained $\lambda > \mu$ yields backlog divergence with throughput saturating at μ. By probing the model with standard dynamic inputs (step, exponential, Gaussian burst), we characterised transient behaviours that matter in practice (backlog peaks, settling, and saturation effects).

On this foundation, we formulated a quorum-aware, multiobjective optimization of protocol parameters $x = (x_1, x_2)$ with three criteria: block-finalization la-

tency F_1, throughput cost $F_2 = -\text{TPS}$, and participation deviation $F_3 = |p(x) - p^*|$, subject to the BFT quorum constraint $p(x) \geq q(n)$. Pareto-front analysis across validator-set sizes showed that *mid-range* committees dominate extremes, and that targeting high—but not maximal—participation ($p^* \in [0.85, 0.95]$) enables robust latency–throughput trade-offs.

The resulting guidelines are practical:

(i) choose n in the mid range compatible with network conditions;

(ii) set p^* to maintain a safety margin above the quorum;

(iii) tune (x_1, x_2) to operate near, but below, the stability boundary $\lambda < \mu$.

This work contributes: a formal load model with provable stability bounds; a quorum-constrained, multiobjective control formulation; and decision criteria (via Pareto frontiers and hypervolume) for selecting n, p^*, and (x_1, x_2). Limitations (fault-free assumption, idealised networking, linear service) and concrete paths to relax them are discussed in *Discussion* and *Future Work*.

Overall, the framework provides a principled, deployment-oriented methodology for configuring Besu-based consortium blockchains.

References

1. Bazaraa, M.S., Sherali, H.D., Shetty, C.M.: Nonlinear Programming: Theory and Algorithms. John Wiley & Sons, Hoboken, NJ, 3rd edn. (2006)
2. Berger, C., Rodrigues, L., Reiser, H.P., Cogo, V., Bessani, A.: Chasing the Speed of Light: Low-Latency Planetary-Scale Adaptive Byzantine Consensus. In: Proc. ACM/IFIP Middleware Conference (2024), arXiv:2305.15000 [cs.DC]
3. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge, UK (2004)
4. Castro, M., Liskov, B.: Practical Byzantine Fault Tolerance. In: Proc. 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI). pp. 173–186 (1999)
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
6. Fan, C., Ghaemi, S., Khazaei, H., Musilek, P.: Performance Evaluation of Blockchain Systems: A Systematic Survey. IEEE Access **8**, 126927–126950 (2020). https://doi.org/10.1109/ACCESS.2020.3006078
7. Fan, C., Lin, C., Khazaei, H., Musilek, P.: Performance Analysis of Hyperledger Besu in Private Blockchain. In: Proc. IEEE Int'l Conf. on Decentralized Applications and Infrastructures (DAPPS). pp. 64–73 (2022). https://doi.org/10.1109/DAPPS55786.2022.00016
8. Hang, L., Kim, D.H.: Optimal Blockchain Network Construction Methodology Based on Analysis of Configurable Components for Enhancing Hyperledger Fabric Performance. Blockchain: Research and Applications **2**(1), 100009 (2021). https://doi.org/10.1016/j.bcra.2021.100009
9. Jiang, L., Chang, X., Liu, Y., Mišić, J., Mišić, V.B.: Performance Analysis of Hyperledger Fabric Platform: A Hierarchical Model Approach. Peer-to-Peer Networking and Applications **13**(2), 1014–1025 (2020). https://doi.org/10.1007/s12083-019-00850-z

10. Leduc, G., Kubler, S., Georges, J.P.: DRAFTEE: A Self-Adaptive Framework for BFT-Based Consensus to Meet Fluctuating Throughput Demands under Security Constraint. Computer Networks **269**, 109214 (2025). https://doi.org/10.1016/j.comnet.2023.109214
11. Li, J., Zhang, G., Liu, D., Zhou, F., Li, L., Meng, W.: DynaNet: A dynamic BFT consensus framework. Journal of Systems Architecture **162**, 103386 (2025). https://doi.org/10.1016/j.sysarc.2025.103386
12. Liu, M., Yu, F.R., Teng, Y., Leung, V.C.M., Song, M.: Performance Optimization for Blockchain-Enabled Industrial Internet of Things (IIoT) Systems: A Deep Reinforcement Learning Approach. IEEE Transactions on Industrial Informatics **15**(6), 3559–3570 (2019). https://doi.org/10.1109/TII.2019.2897805
13. Nasir, Q., Qasse, I.A., Talib, M.A., Nassif, A.B.: Performance Analysis of Hyperledger Fabric Platforms. Security and Communication Networks **2018**, 3976093 (2018). https://doi.org/10.1155/2018/3976093
14. Pongnumkul, S., Siripanpornchana, C., Thajchayapong, S.: Performance Analysis of Private Blockchain Platforms in Varying Workloads. In: Proc. 26th Int'l Conf. on Computer Communication and Networks (ICCCN). pp. 1–6. IEEE (2017). https://doi.org/10.1109/ICCCN.2017.8038517
15. Saltini, R., Hyland-Wood, D.: IBFT 2.0: A Safe and Live Variation of the IBFT Blockchain Consensus Protocol for Eventually Synchronous Networks. arXiv preprint arXiv:1909.10194 (2019), `https://arxiv.org/abs/1909.10194`
16. Sousa, J., Bessani, A., Vukolić, M.: A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform. In: Proc. 48th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN). pp. 51–58 (2018). https://doi.org/10.1109/DSN.2018.00018
17. Sukhwani, H., Martinez, J.M., Chang, X., Trivedi, K.S., Rindos, A.: Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Networks. In: Proc. 2017 IEEE 36th Symp. on Reliable Distributed Systems (SRDS). pp. 253–255 (2017). https://doi.org/10.1109/SRDS.2017.36
18. Sukhwani, H., Wang, N., Trivedi, K.S., Rindos, A.: Performance Modeling of Hyperledger Fabric (Permissioned Blockchain Network). In: Proc. 2018 IEEE 17th Int'l Symp. on Network Computing and Applications (NCA). pp. 1–8 (2018). https://doi.org/10.1109/NCA.2018.8548070
19. Vukolić, M.: The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In: Proc. IFIP Int'l Conf. on Open Problems in Network Security (iNetSec). pp. 112–125 (2015). https://doi.org/10.1007/978-3-319-39028-4_9
20. Xu, X., Sun, G., Luo, L., Cao, H., Yu, H., Vasilakos, A.V.: Latency Performance Modeling and Analysis for Hyperledger Fabric Blockchain Network. Information Processing & Management **58**(1), 102436 (2021). https://doi.org/10.1016/j.ipm.2020.102436
21. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT Consensus with Linearity and Responsiveness. In: Proc. 2019 ACM Symposium on Principles of Distributed Computing (PODC). pp. 347–356 (2019). https://doi.org/10.1145/3293611.3331591
22. Yuan, P., Zheng, K., Xiong, X., Zhang, K., Lei, L.: Performance Modeling and Analysis of a Hyperledger-Based System Using GSPN. Computer Communications **153**, 117–124 (2020). https://doi.org/10.1016/j.comcom.2020.01.073
23. Zhang, J., Yang, Y., Zhao, D., Wang, Y.: A node selection algorithm with a genetic method based on pbft in consortium blockchains. Complex & Intelligent Systems **9**, 3085–3105 (2023). https://doi.org/10.1007/s40747-022-00907-2