Adaptive Block Size and Block Timeout for Hyperledger Fabric Networks

Ericksulino Manoel de A. Moura[1], **Ramon Vítor C. de Abreu**[1]
Franciso Airton Silva[1], **André Soares Castelo Branco**[1]
Allan Edgard Silva Freitas[2], **Glauber Dias Gonçalves**[1]

[1]Universidade Federal do Piauí (UFPI), Teresina, Piauí, Brasil

[2]Instituto Federal da Bahia (IFBA), Salvador, Bahia, Brasil

{ericksulino,ramon.abreu}@ufpi.edu.br

{faps,andre.soares,ggoncalves}@ufpi.edu.br, allan@ifba.edu.br

Abstract. *Permissioned blockchains must sustain stable performance under time-varying transaction arrival rates. In Hyperledger Fabric (HLF), block configuration parameters, such as block size (BS) and block timeout (BT), strongly affect performance, particularly throughput and transaction latency. We propose an adaptive framework that monitors network workload and automatically reconfigures BT and BS across consecutive time windows to improve HLF performance. As a proof of concept, we implement two adaptation strategies from the literature within the framework: aHLF (adapted from a batch control for the Practical Byzantine Fault Tolerance protocol) and FabMAN (originally designed for HLF). We then compare the performance of these strategies with the default fixed-block configuration in HLF, based on experiments on an AWS-based HLF network deployment under a dynamic workload profile. Results show that the adaptive strategies outperform the default configuration. In particular, at the peak workload, the best-performing strategy (FabMAN) achieves 44% higher throughput and 38% lower latency than the default configuration, indicating the benefits of the proposed adaptive blockchain framework.*

1. Introduction

Blockchain is a disruptive technology, particularly for the industrial and service sectors, as it provides robust solutions for secure, decentralized data management [Xu et al. 2019]. Blockchain enables secure recording of transactions between entities, such as individuals or organizations, even without mutual trust. This became possible through the evolution and unification of technologies such as asymmetric cryptography, distributed consensus protocols via peer-to-peer communication, and a structure of blocks chained by cryptographic digests (i.e., hashes) that are the very essence of blockchain. These technologies enable immutability, auditability, and consistency in transaction records.

In public blockchain networks, where cryptocurrencies such as Bitcoin or Ethereum mediate transaction consensus, participation is open to any entity. This openness often leads to large-scale scenarios in which network performance becomes constrained. In contrast, consortium scenarios with a limited set of participants can employ classical consensus protocols such as PBFT [Castro et al. 1999] and RAFT [Ongaro and Ousterhout 2014]. These protocols require participant management, or

membership, but support a higher transaction throughput and improved performance. Such scenarios typically utilize private, or permissioned, blockchain networks. Among these platforms, Hyperledger Fabric is currently one of the most widely adopted[1]. Hyperledger Fabric provides resources for deploying private network infrastructure and for developing applications on this network[2]. In such configurations, network participants form a consortium and share infrastructure costs, aiming to achieve performance gains that surpass those of public blockchain networks.

Performance is a critical factor in ensuring the effectiveness of permissioned blockchains for organizational consortia, particularly in industrial applications where transactions require rapid recording to achieve high throughput and low latency. Beyond the consensus protocol, parameters such as block size and block timeout are also essential, as they determine the number of transactions recorded and directly affect throughput and latency [Liu et al. 2021, Wang et al. 2024]. In public blockchains, performance during periods of congestion is typically managed through built-in adaptation mechanisms, such as dynamic fees and, in some cases, automatic block adjustments [Liu et al. 2022]. These mechanisms discourage excessive demand and contribute to stabilizing transaction latency.

In Hyperledger Fabric, *block size* (BS) and *block timeout* (BT) directly govern block processing efficiency, yet they are commonly kept static even when transaction input rates vary over time. It motivates the need for an approach that monitors workload and continuously reconfigures BT and BS to maintain improved throughput and latency as workload varies. In this sense, a natural question arises: if public systems incorporate mechanisms to adapt to congestion, *why should permissioned deployments keep block parameters fixed under workload-varying demand?*

In this work, we propose an adaptive framework that automatically adapts block parameters in Hyperledger Fabric based on observed workload. The framework continuously monitors the network and applies reconfigurations, using adaptive strategies to estimate near-future transaction input rate and adjust BT and BS between consecutive time windows. As a proof of concept, we implement two adaptive strategies within the framework: FabMAN [Roy and Ghosh 2024], originally designed for Hyperledger Fabric, and aHLF, adapted from an adaptive PBFT batch control approach [de Sá et al. 2013] to HLF. These strategies encompass distinct adjustment algorithms for BS and BT, enabling their performance to be analyzed within the framework.

To support this analysis, we adopt an evaluation methodology to validate the proposed framework and compare the performance of the two strategies with the default fixed-block configuration in HLF. We deploy an AWS-based Hyperledger Fabric network and conduct controlled experiments with a varying workload profile that stresses the blockchain nodes as transaction volume increases and then decreases. During workload execution, we collect performance indicators such as average latency, throughput, and success rate, as the framework automatically adjusts BT and BS values across consecutive time windows. Unlike prior works that focus on isolated strategies, our contribution lies in a unified closed-loop framework and a reproducible evaluation pipeline for fair comparison under dynamic workloads. In summary, this paper contributes with (1) design

[1]https://www.ibm.com/topics/hyperledger

[2]https://hyperledger-fabric.readthedocs.io

and implementation of a framework for automatic adaptation of block parameters in HLF based on pluggable adaptation strategies,[3] and (2) a reproducible evaluation methodology for such framework proposals and their adaptive strategies, considering dynamic workloads to quantify performance.

Overall, the results indicate that both adaptation strategies outperform the default fixed-block parameters in HLF (i.e., the baseline). Specifically, FabMAN achieved the highest throughput (180 transactions per second) and the lowest transaction latency (2.5 seconds) at the highest evaluated transaction input rate, yet exhibiting the most robust transaction success rate (61%), despite slightly lower performance at lower workloads. Compared to the baseline, it represents 44% higher throughput, 38% lower latency, and 61% higher success rate. In turn, aHLF achieved the lowest latency (0.11 seconds) at low-to-medium workloads due to its fine-grained adaptation strategy, which tends to produce small block sizes. These results are consistent with the strategies' distinct BT and BS adaptation values and are further supported by CPU measurements on the blockchain nodes, which reached 100% in the majority of time-window measurements, indicating that these nodes achieved maximum throughput and latency relative to their computational capacity.

The paper presents the following sections: Section 2 reviews the literature on blockchain performance evaluation. In section 3, we describe the proposed framework, explaining its policies and the fundamentals used to predict the Block Size and Block Timeout parameters. Section 4 details the performance evaluation method. Section 5 presents the main results obtained and their analyses. Finally, Section 6 presents our conclusions.

2. Related Work

This section reviews related efforts on parameter self-configuration in permissioned blockchains and positions our contribution, emphasizing the framework itself. Unlike most prior work, which typically evaluates parameters offline or proposes a single tuning mechanism in isolation, we develop a unified adaptive framework that integrates (i) workload enforcement, (ii) online monitoring of system performance metrics, and (iii) automated actuation over HLF block parameters (BT and BS), enabling a reproducible end-to-end evaluation of multiple controllers under the same execution pipeline.

In PBFT-style systems such as Hyperledger Fabric [Androulaki et al. 2018] and Tendermint [Kwon 2014], BT and BS strongly influence performance and stability. [de Sá et al. 2013] is an early reference on feedback-based adjustment of batching parameters, proposing aPBFT to adapt BS/BT under dynamic conditions. In our work, we operationalize this idea within our framework by implementing *aHLF*, an adaptation of aPBFT to HLF, so it can be executed as a controller within the same closed-loop pipeline used for other strategies.

Several studies characterize HLF behavior under different configurations and workloads and, in some cases, propose implementation-level optimizations. For instance, [Thakkar et al. 2018] explores multiple configuration dimensions and bottlenecks, then

[3]The framework source-codes are available in the project repository `https://github.com/B-MonAnalys/blockchain_performance`.

proposes changes to the HLF stack. In contrast, our framework treats HLF as a black box and focuses on runtime parameter tuning via BT and BS, without modifying consensus, validation, or platform internals.

Complementary to direct experimentation, analytical and modeling-based approaches estimate latency as a function of block formation parameters [Xu et al. 2021] or use stochastic Petri nets to guide configuration choices [Silva et al. 2023]. These efforts are valuable for understanding trends and planning, whereas our framework is designed for online control in a running network, where parameter updates must be applied between consecutive adaptation windows.

Beyond performance-centric tuning, prior work has investigated transaction failures and their relation to configuration and policies [Chacko et al. 2021], as well as learning-based controllers that rely on predictive models or cognitive loops [Wang et al. 2024, Ameri and Meybodi 2024]. In contrast, we use the success rate only as a robustness indicator. The adaptation remains lightweight, focusing on closed-loop BT and BS tuning without modifying endorsement policies, consensus membership, or *chaincodes*.

Finally, [Roy and Ghosh 2024] proposes FabMAN, an HLF-specific adaptation strategy that adjusts BT and BS to improve behavior under changing conditions. Our contribution is to embed FabMAN into a general framework and compare it directly with aHLF using an identical end-to-end pipeline, thereby highlighting differences in controller behavior and robustness under the same dynamic workload profile. Our earlier study [Moura et al. 2025] compared aHLF and FabMAN in Hyperledger Fabric under fixed transaction rates and discussed when each is preferable. Here, we extend it with a unified closed-loop framework and a time-varying workload profile (increasing and decreasing), and we complement the comparison with CPU measurements on the bottleneck host to relate performance to resource pressure at peak load.

In summary, existing work provides important tuning mechanisms and experimental evidence but often lacks an end-to-end framework that integrates workload control, monitoring, and automated reconfiguration into a single, reusable pipeline. Our work fills this gap by providing a unified adaptive framework for HLF and using it to compare aHLF and FabMAN under the same closed-loop methodology.

3. Proposed Framework

In this section, we describe our proposed adaptive framework, which is conceptually designed to operate agnostically to blockchain platforms. Thus, it currently focuses on the *block size* (BS) and the *block timeout* (BT) parameters, whose appropriate adjustment might improve the performance of blockchain systems, independent of the consensus mechanism and network configuration [Melo et al. 2022, Thakkar et al. 2018]. In this work, we implement the framework on the Hyperledger Fabric (HLF) blockchain, a well-known open-source project for developing permissioned blockchain networks for government and industry. Besides, there are no tools for adapting BS and BT parameters in the original HLF project available on the Internet[4], leaving it to developers and the research community to make this relevant contribution.

[4]https://hyperledger-fabric.readthedocs.io/en/latest/

3.1. Overview

The proposed framework organizes its operation into three core stages: monitoring, prediction, and adjustment. These stages are carried out within a time window, a fixed interval managed by the framework. In this work, we adopt a 60-second time window, enabling robust performance evaluation of the framework across different workload input rates. Alternatively, a target number of transactions confirmed on the blockchain can serve as a window for carrying out the three stages. However, it may lead to a lower transaction sample for performance evaluation at a higher input rate, as observed in our previous work [Moura et al. 2025].

Thus, the framework carries out the three stages within every time window as follows: The *monitoring stage* consists of collecting metrics from approved transactions and generated blocks. These data support the next stages at the end of the time window. We use Block Explorer[5] during the monitoring stage, a well-known tool for collecting block statistics that is supported by HLF with minimal processing overhead. At the *prediction stage*, an adaptation strategy algorithm processes the observed metrics to predict the transaction input rate for the next window, thereby estimating the appropriate BT and BS parameters. Finally, at the *adjustment stage*, the framework applies the estimated parameters to the blockchain, triggering a continuous cycle of monitoring, prediction, and adjustment. Specifically, the monitoring and prediction stages must use simple, fast adaptive strategies that deliver BT and BS estimates to achieve adjustment at the end of each time window.

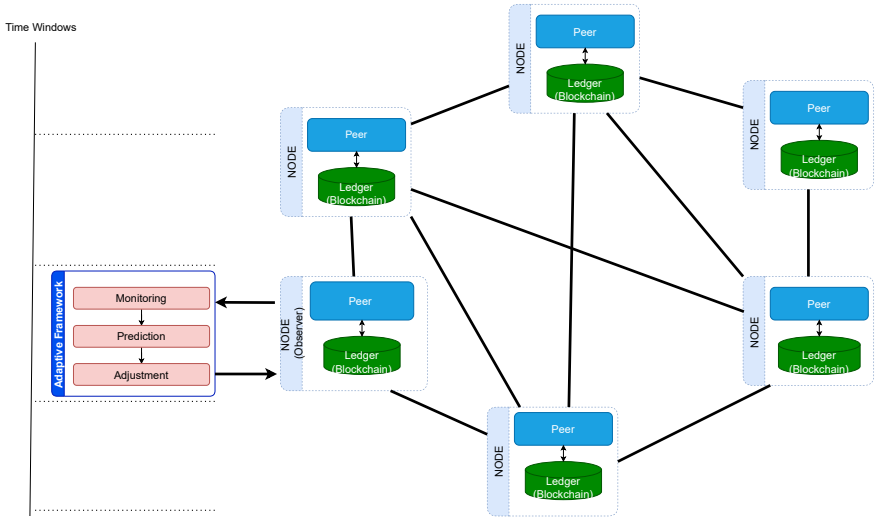


Figure 1. Integration of the proposed adaptive framework into a blockchain Network

Figure 1 conceptually illustrates how the proposed framework integrates with an existing blockchain network. The framework is implemented within an *Observer Node*, shown on the left, which connects to the blockchain nodes to perform the monitoring, prediction, and adjustment stages described. Each node in the blockchain, on the right side, comprises a peer responsible for transaction processing, block validation, and maintenance of a replica of the ledger (i.e., the blockchain), ensuring decentralization and

[5]https://github.com/hyperledger-labs/blockchain-explorer

network resilience. Communication among nodes is represented by links that depict the typical peer-to-peer (P2P) connection. Thus, the Observer Node participates in the blockchain network, ordering adjustments to BT and BS parameters, without performing block processing and validation tasks like the other nodes.

To achieve BT and BS adjustment, the Observer must have been granted the right to set parameters at the *HLF orderer's channel* [Androulaki et al. 2018]. This HLF architecture component is responsible for preparing the transaction block and distributing it to peers via P2P communication. It is based on crash-fault-tolerant (e.g., RAFT) or byzantine-fault-tolerant (e.g., PBFT) protocols, with replicas distributed across a subset of the blockchain nodes, as shown in Figure 2 for simplicity. Thus, the Observer fetches the current block's configuration from the channel (in protobuf format), updates its parameters (new BS and BT values), resubmits it to the channel, and then updates the block's configuration. The framework source code regarding the three stages and their connection with the HLF blockchain is available in the project repository.[6]

3.2. Adaptive Strategies

Now, we describe two strategies to predict transaction input rate and estimate BS and BT at the prediction stage of the framework. These parameters regulate the transaction flow for block construction in Hyperledger Fabric. Whether the number of input transactions reaches BS or the block's waiting time exceeds BT, the system creates a new block. Adapting BS and BT to near-future predicted transaction input rates might improve blockchain performance, resulting in lower transaction latency and higher blockchain throughput.

The strategies below use exponentially weighted moving average (EWMA), which is a fast and efficient method to predict a time series's value ($\hat{Y}_{t+1}$) from the previously observed values (Y_t):

$$\hat{Y}_{t+1} = \alpha Y_t + (1-\alpha)\hat{Y}_t, \tag{1}$$

where the factor α for older observed values decreases exponentially. Recall that observed values used in predictions are extracted from block metadata monitored by the *Block Explorer* within 60-second time windows.

aHLF: this strategy is based on aPBFT [de Sá et al. 2013] to estimate BS and BT from the mean time to order and execute blocks (MTE) and the mean time between transaction arrivals (MTA). We compute *MTE* from recent confirmed blocks using their timestamps, and smooth it with EWMA controlled by a given factor α. The *MTA* is computed as the ratio between the window size (60 seconds) and the number of successfully confirmed transactions within the window, reflecting the transactions' arrival rate per window.

Considering MTA and MTE above described, the new BS estimate is given by:

$$\text{BS}_{new} = \left\lceil \frac{\text{MTE}}{\text{MTA}} \right\rceil. \tag{2}$$

The *block timeout* is set to track MTE, except when the system is underutilized:

$$\text{BT}_{new} = \begin{cases} \text{MTE}, & \text{if MTA} < \text{MTE} \\ 0, & \text{if MTA} \geq \text{MTE} \end{cases} \tag{3}$$

[6]https://github.com/B-MonAnalys/blockchain_performance

FabMAN: This strategy [Roy and Ghosh 2024] performs abrupt updates of BS and BT based on short-term estimates of the transaction input rate (T_{rate} and the network delay (N_delay). Its goal is to keep transaction latency within a predefined, tolerable bound (T_delay) while reacting to changes in the transaction input rate through multiplicative BS adjustments.

We predict the transaction input rate T_{rate} as a function of the number of transactions and the execution time for each recently confirmed block, and smooth it using EWMA with factor α. Then, assuming T_{rate} as the ideal block size, we estimate the new BS by the multiplicative policy:

$$\text{BS}_{new} = \begin{cases} 2 \cdot \text{BS}_{current}, & \text{if } T_{rate} > \lambda \cdot \text{BS}_{current} \\ \frac{1}{2} \cdot \text{BS}_{current}, & \text{if } T_{rate} < \frac{\lambda}{2} \cdot \text{BS}_{current} \\ \text{BS}_{current}, & \text{otherwise} \end{cases} \tag{4}$$

where λ acts as a sensitivity threshold: higher λ makes BS changes less frequent (more stable but slower to adapt), while lower λ increases responsiveness at the cost of potentially more oscillations.

In turn, the new BT estimate subtracts from the pre-defined maximum delay the network delay due to communication among nodes, represented by:

$$\text{BT}_{new} = T_{delay} - N_{delay}, \tag{5}$$

where N_{delay} is computed by the time gap between consecutive committed blocks.

4. Evaluation Methodology

In this section, we describe the methodology for evaluating the proposed framework, using Hyperledger Fabric (HLF). First, we construct an experimental environment based on the HLF network to deploy the framework. Next, we conduct a set of experiments by varying the transaction input rate to evaluate the framework's performance.

4.1. Environment Configuration

In this environment, we integrate (i) a HLF test network, (ii) the proposed adaptive framework, and (iii) a workload generator as shown in Figure 2.

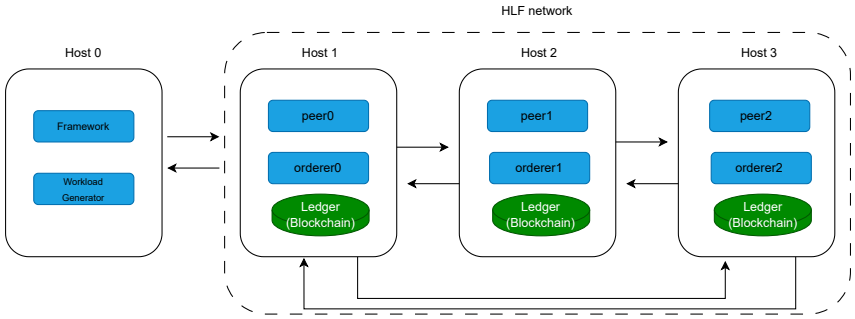


Figure 2. Overview of experimental environment that integrates: (1) HLF network, (2) proposed framework, and (3) workload generator.

The HLF environment was deployed on AWS virtual machines with homogeneous hardware (2 vCPUs, 4 GB RAM, and 100 GB storage). The HLF network topology and deployment layout were inspired by the experimental architecture presented in [Mendonça et al. 2023]. This HLF network was deployed using Docker container images running Hyperledger Fabric's official software (version 2.5). The Hyperledger Foundation provides one container to act as an ordering service node based on the Raft protocol [Ongaro and Ousterhout 2014] and another container to act as a peer in the P2P network, responsible for transaction endorsement and validation. Thus, we instantiated one peer container and one orderer container on the same host. These components communicate through network interfaces and TCP/UDP ports defined by HLF. We deployed three hosts to build a permissioned blockchain network, since HLF requires at least three orderers to create blocks. In this case, we keep the default HLF configuration that uses the RAFT crash-fault tolerance protocol, allowing failures up to 2 orders. An overlay network using the Docker Swarm container orchestrator was used to enable communication among the containers across the three hosts (three orderers and three peers).

The proposed framework and the workload generator were deployed as separate modules on a single host of the environment, shown in Host 0 of Figure 1. The proposed framework provides an automated pipeline that enforces the three stages described in Section 3 to adapt the BS and BT block parameters in HLF at the end of each time window. In this study, the framework instantiates the two adaptive strategies, described in Section 3.2, in addition to the default HLF configuration that uses fixed BS and BT values. In turn, the workload generator HLF-PET, developed in our previous work [Moura et al. 2024], injects synthetic transactions into the network according to predefined workload profiles. HLF-PET enables reproducible workload submission under a given transaction inter-arrival time distribution. For the experiments described in the following, we set the time between transactions to be uniformly distributed for a given set of transaction-per-second (TPS) rates defined in the workload profile.

4.2. Experiments and Metrics

We conduct experiments by setting a dynamic workload profile in HLF-PET with both increasing and decreasing phases, allowing us to assess how the adaptive strategies behave not only as the transaction input rate increases, but also as it decreases. The profile starts with a warm-up period, then applies progressively higher transaction per second (TPS) levels to a peak, followed by a symmetric decrease back to the lowest level.

Specifically, the experiment begins with a window warm-up at 10 TPS, before any adaptations take place. After the warm-up, the workload profile follows a dynamic profile with an increasing and a decreasing phase. In the increase, the TPS doubles at each level (10, 20, 40, 80, 160, and 320 TPS). In the decrease, the TPS is halved symmetrically (320, 160, 80, 40, 20, and 10 TPS). Each level lasts 2 minutes, i.e., two consecutive 60-second time windows. We use this design to monitor the framework, at least, before the adjustment to a new BT and BS blockchain network. Thus, the first window at a given level is used to monitor the current conditions and predict the transaction input rate to the next window. While in the second window, we measure the performance outcome after applying the reconfiguration.

Overall, the workload profile comprises 23 consecutive time windows, organized into one warm-up window, five increasing TPS levels, a peak TPS level, and five de-

creasing TPS levels. This profile is executed for the two adaptation strategies (aHLF and FabMAN) and for the baseline, enabling comparison. The baseline uses the default Hyperledger Fabric parameters, i.e., $BT = 2$ seconds and $BS = 10$ transactions, remaining fixed throughout the experiment. For the adaptive configurations, the same initial BT and BS values are used to ensure a fair comparison. In addition, the strategies are initialized with their tuning parameters: aHLF uses $\alpha = 0.4$, while FabMAN uses $\alpha = 0.4$, $\lambda = 0.9$, and $t_{delay} = 2.0$ s.

The workload generator HLF-PET measures the mean latency, mean throughput, and transaction success rate for each time window during the experiment. The mean latency is calculated by the measured latency of each transaction that started within the window, whereas the mean throughput is calculated by the transactions per second rate for all transactions that started and succeeded within the window. Both mean measures are referred to as *latency* and *throughput* for simplicity. In turn, the success rate is calculated as the ratio of all transactions that start within the window and succeed at any window. Additionally, CPU utilization is collected on each host of the HLF network by an independent Python script executed locally on each machine using the *psutil* library. This instrumentation characterizes the computational cost associated with different strategies and reconfigurations for later analyses.

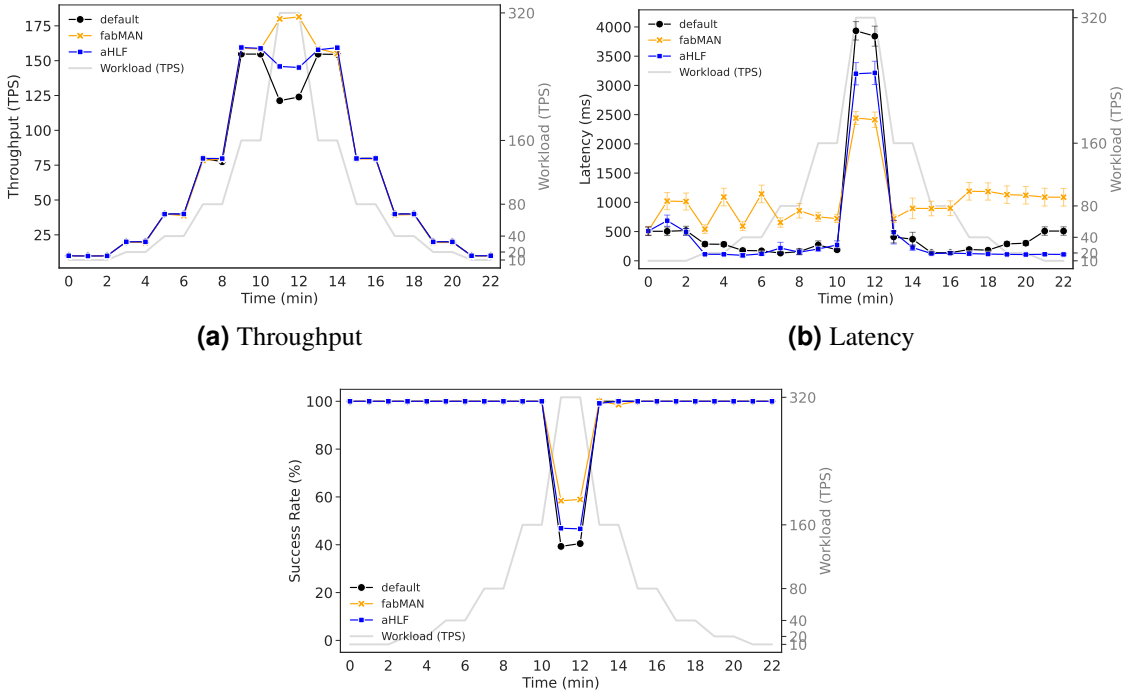
5. Results

In this section, we present the results obtained with the evaluation methodology for the proposed adaptive framework on Hyperledger Fabric (HLF). First, we evaluate performance in terms of throughput, latency, and success rate. Next, we discuss how BT and BS adjustments explain the observed behaviors. Finally, we analyze the computational cost of reconfigurations based on CPU usage on the blockchain nodes.

5.1. Throughput, Latency and Success Rate

In Figure 3, we analyze the achieved performance of the framework in terms of metrics throughput, latency, and success rate. The x-axis shows the total experiment duration, whereas the primary and secondary y-axes show the metrics and the applied workload profile, respectively. The framework's performance is represented by the aHLF (blue) and FabMAN (orange) adaptive strategy curves, while the HLF default configuration (black) serves as the baseline for comparison. Recall that, out of the first window (warm-up), the framework monitoring, prediction, and adjustment continuously. These stages are executed at each minute (time window) shown on the x-axis, considering the increasing and decreasing levels of the workload profile as shown by the light-gray curve for the secondary y-axis of the figure.

Regarding throughput, the FabMAN strategy achieved the best overall performance during the workload peak, outperforming both the baseline (default HLF) and aHLF at the highest workload, i.e., 320 TPS. The aHLF strategy, in turn, achieves the lowest observed latencies at low-to-medium workload levels, reaching approximately 110 ms across workloads of 20-160 TPS. Both strategies also achieve higher success rates at the workload peak, i.e., 59% and 46% for FabMAN and aHLF, respectively, whereas the default configuration achieves only 40%. In sum, we observe that both adaptive strategies outperform the baseline, particularly in terms of latency. FabMAN tends to perform best



(a) Throughput **(b)** Latency

(c) Success Rate

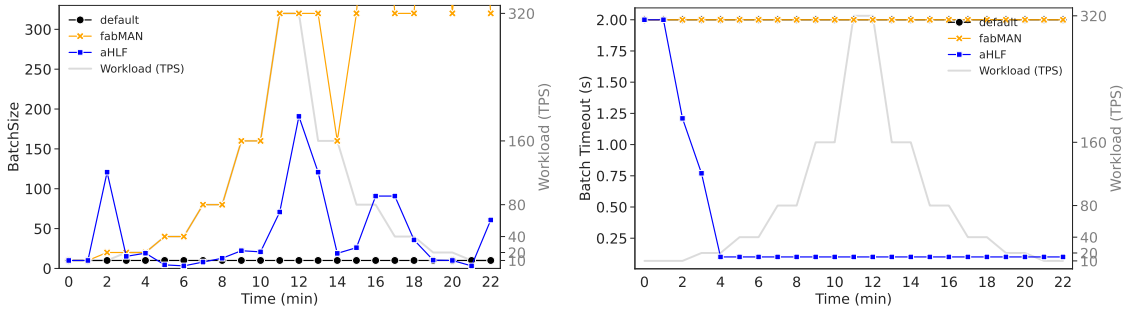
Figure 3. Performance under increasing and a decreasing workload profile for the baseline, aHLF, and FabMAN strategies.

at the highest workload level because its large BS adjust is more effective at handling a large volume of simultaneous transactions, keeping the HLF orderer's channel highly utilized without transaction drops due to queuing. In contrast, aHLF tends to deliver the lowest latency at low workload levels because its tight adjustment of BS and BT favors faster confirmations when the network is not congested.

5.2. Adaptive Behavior: BT and BS Evolution

Figure 4 shows how the blockchain parameters evolve as the workload varies: BS and BT evolutions are shown separately in Figures 4(a) and 4(b), respectively, and both use the same x- and y-axis scales as the performance analysis discussed in the previous section. Recall that the gray curve (secondary y-axis) shows workload levels over time, allowing us to assess the performance of each adaptation strategy at each workload level.

First, we analyze BS adjustments in Figure 4a. As can be observed, the multiplicative policy adopted by FabMAN (orange curve) can mimic workload increases and decreases up to 14 minutes. This period comprises the greatest variation in the workload. Note that FabMAN extends BS to 320 transactions at the highest workload (320 TPS), thus fitting most of that high workload while achieving high throughput and low latency, without overloading blockchain nodes with a large number of blocks. After 14 minutes at the decreasing workload, BS shows a "sawtooth" pattern (omitted in Figure) consistent with prediction inertia due to dominance of the initial timeout ($BT = 2$ seconds), i.e., predictions do not reach the threshold to reduce BS even when blocks are almost empty. This behavior does not significantly impact performance. Throughput follows the workload levels, and latency remains below the predefined bound ($T_{delay} = 2$ seconds). In



(a) Evolution of *block size* (BS). **(b)** Evolution of *block timeout* (BT).

Figure 4. Evolution of BS and BT parameters: aHLF and FabMAN adjust BT and BS over time, by contrast, the baseline that remains them fixed.

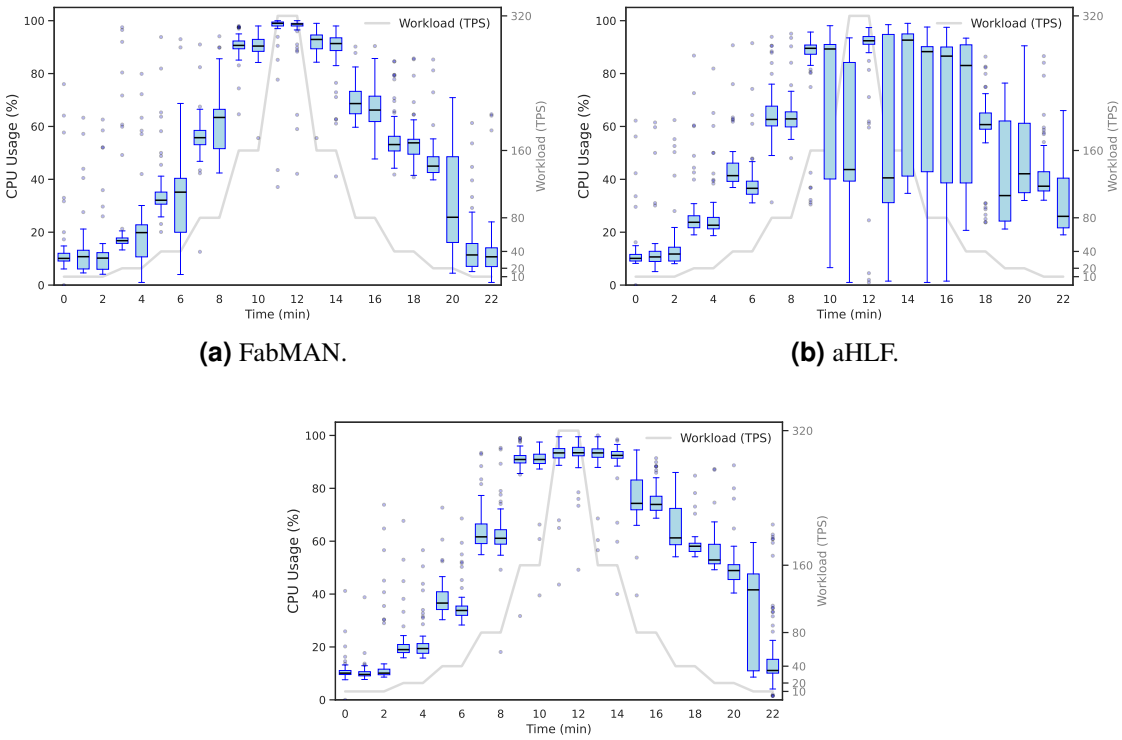
turn, the fine-grained adjustment strategy of aHLF (blue curve) tends to reduce BS beyond the workload level, which reduces transaction delay, but increases node overhead and degrades performance at peak workload.

Next, the BT adjustments in Figure 4b corroborate the observations. FabMAN BT does not vary from its initial value because communication delays among peers in the experimental environment are negligible relative to the predefined transaction latency bound. On the other hand, aHLF BT decreases progressively, which dominates BS estimates and feeds back into the decrease of BT. The reconfiguration process introduces only minor transient effects, without observable instability during execution.

5.3. Computational Cost: CPU Usage

Finally, we evaluate the computational cost associated with the adaptation strategies. Although the framework can improve system performance through block reconfigurations, such adjustments may impose additional pressure on node resources. Therefore, we analyze CPU usage as a supporting metric. Figure 5 summarizes Host 1 utilization using boxplots, which capture the distribution of CPU values observed across the adaptation windows. A gray line also indicates the workload level (right y-axis) along the workload profile. Unlike previously analyzed metrics, CPU usage is monitored independently on each host by a local Python script using the *psutil* library. For space, we show only the results for Host 1, which serves as the first HLF gateway to receive transactions and forward them to other nodes, resulting in slightly higher CPU usage than other nodes. Thus, we use the CPU behavior of Host 1 to compare baseline, aHLF, and FabMAN, relating the observed computational cost to the BT and BS adaptation patterns and the resulting performance under different workload levels.

Figure 5a indicates that FabMAN achieved the lowest latency for an HLF network; interestingly, we have processing power according to the experimental environment built. Note that at peak load, FabMAN achieves lower latency and higher throughput, while the closed boxplots (i.e., low variability) indicate that virtually all CPU frequencies are concentrated near 100% CPU usage. Therefore, the blockchain network nodes are overutilized, with no additional resources to increase the block processing rate. Consequently, latency cannot be reduced, and throughput cannot be increased beyond the achieved levels. In other words, performance improvements at the evaluated peak load



(a) FabMAN.

(b) aHLF.

(c) Baseline (default HLF).

Figure 5. CPU utilization on Host 1 (bottleneck host) for the baseline, aHLF, and FabMAN.

would require increased computational power on the nodes, given the planned strategies.

In turn, Figure 5b indicates that aHLF operates under greater availability of computational resources at low load levels (i.e., 10-80 TPS), given the use of up to 60% CPU for half of the halves (median line of the boxplots). In this case, aHLF already tends to split BS into smaller blocks (Figure 4(b)), which explains its better latency performance without requiring excessive computational resources. Conversely, also in this figure, note that between 160 and 320 TPS, aHLF begins to show predominantly above 80% CPU usage, but does not achieve the lowest latency at peak load, consistent with the adaptation tending towards small blocks that increase computational overhead under high transaction input rates.

Finally, Figure 5c shows the use of CPU for the baseline, i.e., the default Hyperledger configuration. As can be seen, the baseline reaches a high CPU plateau, but its median is lower than FabMAN at the workload peak. It indicates that the CPU is underutilized at that point due to a low BS configuration, suggesting limitations of a fixed-block configuration. Therefore, performance at peak load is primarily constrained by hardware capacity rather than solely by the adaptation strategies.

6. Final Considerations

This paper presented an adaptive framework for closed-loop parameter tuning in permissioned blockchain networks, using Hyperledger Fabric (HLF) as the target platform. The framework combines workload enforcement, online monitoring of system performance

(achieved throughput, confirmation latency, and success rate), and automated actuation over the ordering service parameters, enabling dynamic adjustment of *block size* (BS) and *block timeout* (BT) across consecutive monitoring windows in a reproducible experimental pipeline.

Experiments under a dynamic workload profile with increasing and decreasing phases highlight distinct operating regimes in Hyperledger Fabric. At low transaction rates, the baseline and *aHLF* tend to be more favorable in latency, indicating that conservative operation is sufficient when the system is far from saturation and aggressive adaptation is unnecessary. As transaction pressure increases toward the peak workload, *FabMAN* becomes more effective, sustaining higher throughput with fewer transaction losses and better latency control, indicating greater robustness in congestion regimes. The CPU behavior reinforces this interpretation by showing that, near the peak, the environment approaches its processing limit, suggesting that performance becomes bounded by available resources rather than by configuration alone. As future work, we plan to evaluate the framework on a more robust infrastructure to explore higher-load regimes. We also intend to integrate reinforcement learning to learn tuning policies that adapt to workload context and performance targets. These results are obtained under a controlled and resource-constrained environment, which may influence system behavior under extreme workloads.

References

Ameri, R. and Meybodi, M. R. (2024). Cognitive blockchain and its application to optimize performance in blockchain systems. *Transactions on Emerging Telecommunications Technologies*, 35(7):e5009.

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. (2018). Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM.

Castro, M., Liskov, B., et al. (1999). Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186.

Chacko, J. A., Mayer, R., and Jacobsen, H.-A. (2021). Why do my blockchain transactions fail? a study of hyperledger fabric. In *Proceedings of the 2021 international conference on management of data*, pages 221–234.

de Sá, A. S., Silva Freitas, A. E., and de Araújo Macêdo, R. J. (2013). Adaptive request batching for byzantine replication. *ACM SIGOPS Operating Systems Review*, 47(1):35–42.

Kwon, J. (2014). Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11):1–11.

Liu, C.-M., Badigineni, M., and Lu, S. W. (2021). Adaptive blocksize for iot payload data on fabric blockchain. In *2021 30th Wireless and Optical Communications Conference (WOCC)*, pages 92–96. IEEE.

Liu, Y., Lu, Y., Nayak, K., Zhang, F., Zhang, L., and Zhao, Y. (2022). Empirical analysis of eip-1559: Transaction fees, waiting times, and consensus security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2099–2113.

Melo, C., Araujo, J., Dantas, J., Pereira, P., and Maciel, P. (2022). A model-based approach for planning blockchain service provisioning. *Computing*, 104(2):315–337.

Mendonça, R., Moura, E., Gonçalves, G., Vieira, A., and Nacif, J. (2023). Comparação e análise de custo e desempenho entre nós de redes blockchain permissionadas e públicas. In *Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 141–154, Porto Alegre, RS, Brasil. SBC.

Moura, E., Melo, C., Gonçalves, G., Silva, F., and Soares, A. (2024). Uma ferramenta de avaliação de desempenho para plataforma blockchain hyperledger fabric: Hlf-pet. In *Anais do II Colóquio em Blockchain e Web Descentralizada*, pages 8–13, Porto Alegre, RS, Brasil. SBC.

Moura, E. d. A., Gama, F. S., Gonçalves, G. D., Freitas, A. S., and Soares, A. (2025). Rumo à blockchain adaptiva para a plataforma hyperledger fabric. In *Anais do VIII Workshop em Blockchain: Teoria, Tecnologias e Aplicações*, pages 15–28, Porto Alegre, RS, Brasil. SBC.

Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *2014 USENIX annual technical conference (USENIX ATC 14)*, pages 305–319.

Roy, U. and Ghosh, N. (2024). Fabman: A framework for ledger storage and size management for hyperledger fabric-based iot applications. *IEEE Transactions on Network and Service Management*.

Silva, F., Gonçalves, G., fé, I., Feitosa, L., and Soares, A. (2023). Avaliação de desempenho de blockchains permissionadas hyperledger orientada ao planejamento de capacidade de recursos computacionais. In *Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 71–84, Porto Alegre, RS, Brasil. SBC.

Thakkar, P., Nathan, S., and Viswanathan, B. (2018). Performance benchmarking and optimizing hyperledger fabric blockchain platform. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 264–276. IEEE.

Wang, J., Wang, Y., Zhang, X., Jin, Z., Zhu, C., Li, L., Zhu, R., and Lv, S. (2024). Learningchain: A highly scalable and applicable learning-based blockchain performance optimization framework. *IEEE Transactions on Network and Service Management*, 21:1817–1831.

Xu, X., Sun, G., Luo, L., Cao, H., Yu, H., and Vasilakos, A. V. (2021). Latency performance modeling and analysis for hyperledger fabric blockchain network. *Information Processing & Management*, 58(1):102436.

Xu, X., Weber, I., and Staples, M. (2019). *Architecture for blockchain applications*. Springer.