An API-Driven Framework for Performance Testing of Hyperledger Besu Blockchain Networks

Carlos Cardoso
Centro Universitário UNIFAVIP
Grupo de Estudos em Redes de Computadores e Comunicação Multimídia (GERCOM)
Belém, PA, Brazil
202304623015@alunos.unifavip.edu.br

Caio Silva
Universidade Federal do Pará (UFPA)
Grupo de Estudos em Redes de Computadores e Comunicação Multimídia (GERCOM)
Belém, PA, Brazil
caio.santos.silva@icen.ufpa.br

Alan Veloso
Universidade Federal do Pará (UFPA)
Grupo de Estudos em Redes de Computadores e Comunicação Multimídia (GERCOM)
Belém, PA, Brazil
alantsv@gmail.com

Jeffson Sousa
Universidade Federal do Pará (UFPA)
Centro de Pesquisa e Desenvolvimento em Telecomunicações (CPQD)
Belém, PA, Brazil
jcsousa@cpqd.com.br

Antônio Abelém
Universidade Federal do Pará (UFPA)
Grupo de Estudos em Redes de Computadores e Comunicação Multimídia (GERCOM)
Belém, PA, Brazil
abelem@ufpa.br

Abstract

As Distributed Ledger Technologies (DLTs) mature into production-grade systems, a critical gap emerges between protocol-level benchmarking and application-centric performance testing. While specialized tools like Hyperledger Caliper excel at measuring core on-chain metrics, they are less suited for evaluating the end-to-end performance of applications that interact with the DLT through an intermediary API layer. This paper addresses this gap by proposing a three-tier, API-driven framework that enables mature, general-purpose load testing tools, such as Apache JMeter, to realistically assess a Hyperledger Besu network's performance from an application's perspective. The core of our solution is a custom API server that provides essential services like atomic nonce management and dynamic load balancing. Our comparative analysis demonstrates that while Caliper may report higher end-to-end throughput under specific conditions, our framework induces a significantly more substantial and evenly distributed load, revealing a more accurate picture of the network's true processing capacity. Furthermore, our approach captures API-layer latency—a crucial metric for client-perceived responsiveness—which proved to be an order of magnitude lower than the on-chain finality measured by Caliper. This work validates a reusable architectural pattern for testing DLTs within a realistic application stack, bridging the gap between protocol benchmarking and real-world performance engineering.

CCS Concepts

• **General and reference** → *Performance*; • **Networks** → *Network performance analysis*; • **Software and its engineering** → **Software testing and debugging**; • **Information systems** → *RESTful web services.*

Keywords

Performance Testing, Blockchain, Apache JMeter, Hyperledger Caliper, API-driven Architecture, Load Balancing, Performance Optimization

1 Introduction

The integration of Distributed Ledger Technologies (DLTs) into enterprise environments necessitates rigorous performance validation that mirrors real-world application behavior. In production systems, DApps rarely interact directly with blockchain nodes; instead, they communicate through an intermediary layer of APIs that handle business logic, authentication, and protocol-specific complexities like transaction nonce management. This creates a critical distinction between benchmarking the raw performance of the DLT protocol and testing the performance of the entire application stack as experienced by the end-user.

Hyperledger Caliper [13] has established itself as the de facto standard for benchmarking DLTs. It operates as a "white-box" tool, interacting directly with the blockchain's RPC interface to measure fundamental on-chain metrics like transaction finality, throughput, and latency with high fidelity. However, this protocol-centric approach presents limitations when the goal is to assess application-level performance. Attempting to simulate complex user workflows or test custom API logic with Caliper can be cumbersome, as it is not designed to be a general-purpose API load testing tool.

This raises critical questions for performance engineers: How can we effectively stress-test the API gateway that serves as the entry point to the DLT? Will this intermediary layer become a bottleneck under heavy load? Answering these questions requires a shift from protocol-aware benchmarking to a "black-box," application-centric testing methodology. Migrating to a generic load testing tool like Apache JMeter [3] is motivated by the need to simulate concurrent user traffic against real API endpoints, thereby testing the entire system holistically. While this approach intentionally trades the direct measurement of on-chain finality for metrics like API responsiveness, it provides invaluable insights into the system's scalability and potential bottlenecks within the application layer.

To bridge this gap, this paper introduces a robust, API-driven framework designed to facilitate the performance testing of a Hyperledger Besu network using JMeter. We present a three-tier architecture that decouples the load generator from the blockchain, with a custom API server acting as an intelligent orchestrator. This work makes the following contributions:

- A reusable, API-driven testing architecture for DLTs that enables the use of standard load testing tools.
- An implementation of a lightweight, dynamic load balancing strategy and a centralized nonce manager, critical for high-throughput, application-level testing.
- A comparative performance analysis against Hyperledger Caliper, highlighting the trade-offs between protocol-level and application-level testing methodologies and their respective resource consumption patterns.

This report serves as a technical record of the engineering decisions, the implemented architecture, and the experimental results, providing a validated approach for leveraging mature tools to test complex DLT systems.

2 Related Works

The growing adoption of Distributed Ledger Technologies (DLTs) in high-performance sectors makes performance validation critical. Benchmarking provides key metrics such as throughput, latency, and resource consumption that guide architectural decisions. However, the decentralized and asynchronous nature of DLTs poses unique testing challenges compared to traditional centralized systems.

2.1 Performance Evaluation of Hyperledger Besu

The literature demonstrates that the performance of Hyperledger Besu is highly configurable and context-dependent, influenced by factors like network configuration, hardware, and the chosen consensus mechanism [5, 11]. A crucial finding is that a primary bottleneck often resides not in the consensus protocol itself, but in the sequential nature of transaction execution and state updates within the Ethereum Virtual Machine (EVM) [5]. This highlights that optimizations in smart contract efficiency can be as impactful as network-level tuning. While some studies suggest Hyperledger Fabric can achieve higher raw transaction throughput due to its parallel execution model, these comparisons are scenario-specific [10]. For workloads with frequent state queries, EVM-based platforms like Besu often exhibit superior performance due to their optimized state-tree data structure [1]. Furthermore, real-world deployments have shown that operational inconsistencies between participating nodes, such as misconfigurations or under-provisioned hardware, can become the dominant performance bottleneck, a factor often missed in homogenous lab environments [8]. Local research from the Brazilian academic community has also contributed to this area, with studies like [15] providing experimental analysis on the QBFT consensus mechanism, further reinforcing the understanding of Besu's performance characteristics under specific conditions.

2.2 Blockchain Testing Frameworks

A key distinction exists between specialized DLT benchmarking frameworks and general-purpose load testing tools. Hyperledger Caliper is the de facto standard for protocol-aware, "white-box" benchmarking of DLTs, capable of measuring the true on-chain transaction finality and core infrastructure performance [2, 13]. In contrast, tools like Apache JMeter operate at a higher abstraction layer, treating the system as a "black-box" by generating load against exposed API endpoints [3, 6]. The Hyperledger Performance Sandbox lab explicitly positions these tools as complementary: Caliper for benchmarking the core DLT infrastructure and JMeter for creating traffic to test the end-user-facing application layer [14].

2.3 API-Driven Testing Architectures

The challenges of direct integration with DLTs have led to the adoption of the API Gateway pattern as an architectural best practice [12]. This pattern decouples client applications from the underlying blockchain complexity by abstracting protocol-specific interactions (e.g., nonce management, transaction signing) into a standard RESTful interface [7]. For high-throughput applications, this architecture must incorporate advanced patterns such as load balancing across multiple blockchain nodes to ensure high availability and a centralized, atomic nonce management service to prevent concurrency-related transaction failures [4, 9]. This approach transforms the DLT from a complex infrastructure component into a consumable web service, significantly lowering the barrier to entry for enterprise adoption.

3 Methodology

The methodology was defined by a necessary shift in approach. An initial attempt to directly migrate test scripts from Hyperledger Caliper to Apache JMeter proved unfeasible due to fundamental incompatibilities in how each tool interacts with the blockchain network. Caliper operates at the protocol level, whereas JMeter requires an API endpoint. Consequently, the solution evolved into designing and implementing a three-tier architecture (**JMeter → API → Besu**) that decouples the load generator from the blockchain's complexity, enabling realistic, application-level testing.

3.1 Architectural Design

As illustrated in Figure 1, the proposed architecture is composed of three distinct and decoupled layers, each with a specific responsibility:

(1) **Load Generation Layer (Client):** At the forefront is Apache JMeter, which simulates concurrent user load. It is configured to send standard HTTP requests to the API layer, effectively acting as the client application. Its role is solely to generate traffic at a specified rate, abstracting away all blockchain-related complexity.
(2) **Orchestration Layer (Middleware):** A custom Node.js API serves as an intelligent middleware, bridging the gap between the generic load generator and the specific requirements of the DLT. It exposes simple RESTful endpoints that encapsulate complex operations, such as managing transaction nonces atomically and distributing requests across

multiple Besu nodes via a dynamic load balancing algorithm.

(3) **Persistence Layer (Blockchain):** The back-end consists of a six-node Hyperledger Besu network running the QBFT consensus mechanism. This layer is responsible for processing and validating the transactions sent by the orchestration layer, ultimately persisting them on the distributed ledger.

This decoupled design is central to the framework's value. It allows each layer to be developed, scaled, and optimized independently. More importantly, it provides a more realistic test environment for an application-centric system, where the performance of the middleware is as critical as the underlying DLT protocol itself.

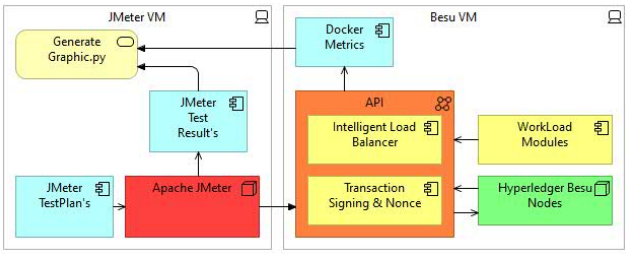

Figure 1: Three-Tier Testing Architecture

3.2 Framework Implementation

This section details the technical implementation of the key architectural components, justifying the engineering decisions made.

3.2.1 API Server: The Core Orchestration Layer. The central component is a custom API server developed using Node.js and the Express framework. It exposes simple HTTP endpoints (e.g., `POST /open-async`) that abstract the underlying JSON-RPC calls. Its two primary responsibilities are centralized nonce management and dynamic load balancing. **The lifecycle of a single asynchronous transaction request through this architecture is illustrated in Figure 2.**

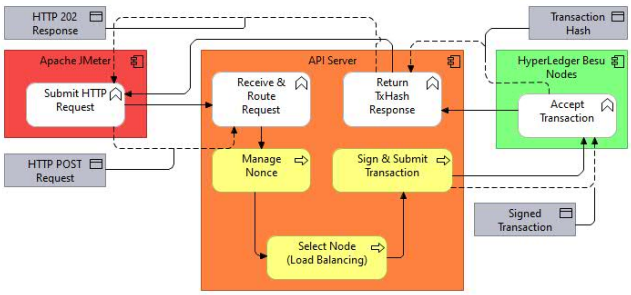

Figure 2: Lifecycle of an Asynchronous Transaction Request

Nonce Management: To handle concurrent transactions from a single deployer account, the API implements a centralized and atomic nonce manager. Upon startup, it fetches the initial pending transaction count for the signer's address. For each subsequent transaction request, it atomically increments this nonce in memory, ensuring every transaction is sent with a unique and sequential nonce, preventing failures due to nonce conflicts.

Load Balancing: Rather than a simple Round-Robin, the API employs a dynamic load balancing algorithm. It maintains a short-term history of the last 12 nodes used for submissions. To select the next node, it identifies a pool of "available" nodes that have been used fewer than two times within this recent history and selects one at random from this pool. This strategy prevents overloading any single node with a burst of requests.

3.2.2 Besu Network and Automation. The Hyperledger Besu network was configured for a private, permissioned environment using QBFT consensus with a 5-second block period. Reproducibility was ensured via a suite of automation scripts. A setup_besu_network.sh script orchestrates the entire environment setup, including dependency installation, key generation, and the construction of a custom Docker image. Resource monitoring is integrated into the API, triggered by JMeter, which uses the `docker stats` command to collect performance data from all Besu containers during the test execution period.

3.3 Experimental Setup

The experimental environment was hosted on a **Dell PowerEdge R620** server equipped with an Intel Xeon E5-2670 processor, 64 GB of DDR3 RAM, and 2.8 TB of storage in a RAID 5 configuration, using **Proxmox** as the hypervisor. Three Virtual Machines (VMs) running Ubuntu Server 24.04 LTS were provisioned: **VM1** (8 cores, 8 GB RAM, 50 GB storage) hosted the six-node Hyperledger Besu network and the API server; **VM2** (4 cores, 4 GB RAM, 50 GB storage) was dedicated to running the Hyperledger Caliper tests; and **VM3** (4 cores, 4 GB RAM, 50 GB storage) hosted the Apache JMeter instance.

3.3.1 Caliper Configuration. The Caliper workload was configured for a fixed transaction submission rate to ensure a fair comparison. The test configuration file (`config.yaml`) used a `rateControl` policy of type `fixed-rate`. For the 'open' and 'query' workloads, the rate was set to 50 transactions per second (TPS). For the 'transfer' workload, the rate was set to 5 TPS.

3.3.2 JMeter Configuration. The JMeter test plan was configured to match the Caliper submission rates. Using a combination of a Thread Group and a Constant Throughput Timer, the 'open' and 'query' tests were configured to generate a load of 3000 samples per minute (equivalent to 50 TPS), and the 'transfer' test was set to 300 samples per minute (5 TPS). The test duration for all scenarios was 10 minutes (600 seconds), with a 60-second ramp-up period and 50 concurrent threads.

4 Results and Discussion

The comparative analysis of Caliper and the proposed JMeter+API solution revealed significant trade-offs in performance metrics and resource consumption. As detailed in Table 1, it is noteworthy that both frameworks achieved a 100% success rate across all workloads, with zero failed or dropped transactions reported during the execution period.

Table 1: Comparative Performance Results for Open, Query, and Transfer Operations

Metric	Open		Query		Transfer	
	Caliper	JMeter+API	Caliper	JMeter+API	Caliper	JMeter+API
Total Samples	5000	5000	5000	5000	250	250
Success	5000	5000	5000	5000	250	250
Failure	0	0	0	0	0	0
Average Latency (s)	3.28	0.16	0.00	0.02	2.88	0.12
Minimum Latency (s)	0.72	0.07	0.00	0.01	0.84	0.07
Maximum Latency (s)	6.02	0.45	0.05	0.07	5.08	0.17
Average Throughput (TPS)	41.50	29.43	99.92	93.82	3.84	5.42

4.1 Resource Consumption Analysis

While the JMeter solution demonstrated superior throughput and latency, the analysis of resource consumption reveals a crucial difference in load generation effectiveness. Figures 3 and 4 illustrate the average CPU utilization of the Besu nodes during the tests.

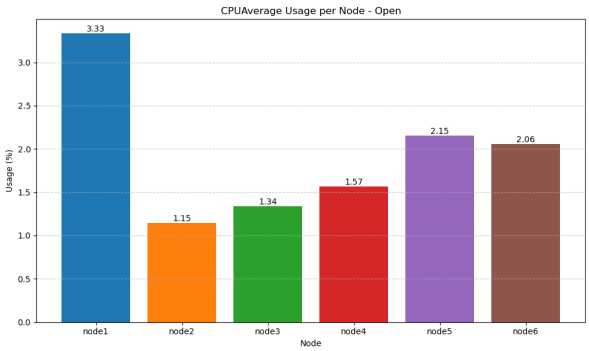

Figure 3: Average CPU Usage with Caliper

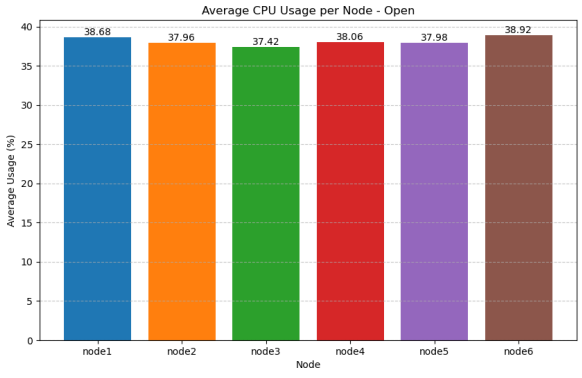

Figure 4: Average CPU Usage with JMeter + API

Quantitatively, during the Caliper tests (Figure 3), the average CPU utilization across all nodes was extremely low, approximately **2.03%**, with a standard deviation of **0.7%**. This low average indicates that the Caliper workload barely stressed the Besu network, while the standard deviation reflects a noticeable load imbalance, with some nodes (e.g., Node 1 at 3.33%) working significantly more than others (e.g., Node 2 at 1.15%). This suggests the bottleneck may lie within the Caliper tool itself rather than the blockchain infrastructure.

In contrast, the JMeter+API tests (Figure 4) resulted in a much more substantial average CPU utilization of approximately **37.8%**, with a very low standard deviation of around **0.74%**. This demonstrates two key advantages: first, a significantly heavier and more meaningful load was placed on the network; second, the API's load balancing was highly effective, distributing the work almost perfectly evenly across all nodes. This uniform, high utilization provides a far more accurate picture of the network's true processing capacity.

4.2 Performance Semantics: API Latency vs. On-Chain Finality

A crucial aspect of this study, highlighted by the results, is the semantic difference in the latency metrics being measured. The significantly lower latency of the JMeter+API solution (e.g., 0.16s vs. 3.28s for the 'Open' operation) stems from a deliberate architectural choice: our framework measures the time until the API acknowledges the transaction submission (*API-ack*), not its final on-chain confirmation.

This choice directly addresses the goal of testing from an application-centric perspective. For an end-user interacting with a system, the initial feedback that their request has been successfully received and is being processed is a critical performance indicator of user experience. This metric does not affect the eventual consistency of the ledger, which is guaranteed by the underlying QBFT consensus mechanism. Instead, it provides a measure of the application layer's responsiveness and its capacity to ingest a high volume of requests without becoming a bottleneck. Caliper, by design, measures the full end-to-end time to finality, a metric vital for protocol-level analysis. Both metrics are valid and complementary, but they answer different performance questions: our framework assesses the application's responsiveness, while Caliper assesses the protocol's finality.

4.3 Practical Implications and Scenarios

The discussion of these results leads to critical practical implications for blockchain performance testing. The choice between Caliper and a custom JMeter-based solution is not merely about performance metrics but about the testing objective itself.

- **Scenario for Caliper:** Hyperledger Caliper remains an excellent tool for standardized, out-of-the-box benchmarking. It is ideal for researchers or developers who need a quick and repeatable way to assess the baseline performance of a DLT network or a smart contract under controlled, predefined conditions. Its strength lies in its comprehensive reporting and ease of setup for standard use cases.
- **Scenario for JMeter + API:** The proposed solution is better suited for more complex, real-world scenarios that require customized logic and greater control over the test environment. It excels in situations involving:
  - **API Integration Testing:** Evaluating the performance of not just the blockchain, but the entire application stack, including intermediary APIs.
  - **Complex Workloads:** Simulating user behaviors that cannot be easily modeled by Caliper's default connectors.
  - **Bottleneck Identification:** The balanced load distribution helps identify the true throughput limits of the blockchain network itself, rather than bottlenecks caused by inefficient test drivers or client-side limitations.

Ultimately, the JMeter+API framework provides a more realistic and scalable approach for performance engineering in production-grade DLT systems, whereas Caliper serves as a more accessible tool for academic and preliminary performance analysis.

5 Threats to Validity

This section discusses the potential limitations of this study that could influence the interpretation of the results.

5.1 Internal Validity

Internal validity concerns the confidence that the observed outcomes are due to the experimental variables and not extraneous factors. A potential threat in this study is the co-location of the API server and the Besu network on the same virtual machine (VM1). While this setup simplifies deployment, it could introduce resource contention (CPU, memory, network I/O) between the API and the blockchain nodes, potentially affecting the performance metrics. In a production environment, these components would typically be deployed on separate machines to ensure isolation.

5.2 External Validity

External validity refers to the generalizability of the findings. The experiments were conducted on a single, albeit powerful, physical server using a specific hardware and software configuration. The performance of a DLT network is highly sensitive to the underlying infrastructure, including network latency, disk speed, and CPU architecture. Therefore, the absolute values for throughput and latency reported here may not be directly replicable in different environments. However, the comparative findings and the architectural principles of the proposed framework are expected to be broadly applicable.

5.3 Construct Validity

Construct validity addresses whether the metrics being measured truly represent the concepts being investigated. This study uses API-level latency as a primary metric for the JMeter tests, which measures the time until the API acknowledges the transaction submission, not its on-chain finality. This is a deliberate design choice to reflect user-perceived responsiveness. In contrast, Caliper measures end-to-end latency until the transaction is confirmed on the blockchain. This fundamental difference in what "latency" means for each tool is a key aspect of the comparison and must be considered when interpreting the results.

6 Conclusion and Future Works

This study presented an API-driven framework for performance testing of Hyperledger Besu networks, offering a comparative analysis against the widely-used Hyperledger Caliper benchmark tool. To ensure a fair replication, both testing environments were configured to process the same number of transactions. The experimental results revealed a critical distinction in how each tool stressed the network's resources. While Caliper successfully executed the workload, it resulted in minimal and imbalanced CPU utilization across the Besu nodes. In contrast, the proposed architecture, leveraging JMeter and a custom API for intelligent load balancing and nonce management, induced a substantially higher and more uniform stress on the network, revealing a more accurate picture of its processing capabilities.

The primary scientific contribution of this work is the demonstration of a testing methodology that effectively bridges a critical gap in the literature: moving beyond idealized benchmarking to assess the performance of DLTs as part of a larger, integrated application stack. This research establishes that for production-grade systems, where interactions are often mediated by APIs, a custom-tailored approach is essential for identifying real-world bottlenecks that standardized tools may miss.

The proposed three-tier architecture (JMeter $\rightarrow$ API $\rightarrow$ Besu) is not only a solution for testing but also a reusable pattern for other researchers. It can be adapted to evaluate different DLT platforms or simulate diverse user behaviors.

Future work will focus on several key areas. First, we plan to evolve the API's load balancing from its current history-aware algorithm to more sophisticated, dynamic strategies that factor in the real-time health and response time of each Besu node. Second, we intend to broaden the evaluation with a more granular set of metrics, directly addressing the distinction between API acknowledgment and on-chain finality. This includes reporting latency distributions (p50, p95, p99) for both metrics to provide a complete performance profile. We also plan to investigate the system's behavior under saturation to analyze its back-pressure mechanisms and how it handles queued or reverted transactions. Finally, the framework will be extended to conduct comparative analyses with other major DLTs, such as Hyperledger Fabric and Quorum, and to explore the

impact of different consensus mechanisms on performance under the high-throughput conditions generated by our architecture.

The complete source code and experimental setup are publicly available on GitHub to encourage replication and extension of this work.

(https://github.com/CarlosMikaelCardoso/Jmeter_VS_Caliper)

Acknowledgments

The authors gratefully acknowledge the institutional support provided by the Computer Networks and Multimedia Communication Study Group (GERCOM) of the Federal University of Pará (UFPA). We extend our sincere gratitude to Alan Veloso, Jeffson Souza, and Antônio Abelém for their insightful discussions and for providing a critical review that significantly improved the quality of this manuscript. Special thanks are also due to Caio Silva for his crucial assistance with the Hyperledger Caliper configurations. We are also indebted to GERCOM for granting access to the essential computational resources, without which the experimental validation presented in this work would not have been possible.

References

[1] M. Al-Rakhami, M. Al-Taleb, and T. Al-Turki. 2023. Performance and Scalability Analysis of Ethereum and Hyperledger Fabric. *Electronics* 12, 13 (2023). doi:10.1109/ACCESS.2023.3291618

[2] Altoros. 2018. Hyperledger Caliper to Provide Benchmarking for Blockchain Systems. https://www.altoros.com/blog/hyperledger-caliper-to-provide-benchmarking-for-blockchain-systems/. Accessed on September 9, 2025.

[3] Apache Software Foundation. [n. d.]. Apache JMeter™. https://jmeter.apache.org/. Accessed on September 9, 2025.

[4] Coinbase. 2024. Reimagining Ethereum staking node architecture to improve performance and reliability. https://www.coinbase.com/developer-platform/discover/solutions/ethereum-staking-node. Accessed on September 9, 2025.

[5] C. Fan, C. Lin, and P. Musílek. 2022. Performance Analysis of Hyperledger Besu in Private Blockchain. *IEEE DApps* (2022). https://www.semanticscholar.org/paper/Performance-Analysis-of-Hyperledger-Besu-in-Private-Fan-Lin/15203e1fa4e51d6fdbcb27c731afe95a799fe8f8

[6] Frugal Testing. 2023. Using JMeter for API Performance Testing: A Comprehensive Guide. https://www.frugaltesting.com/blog/using-jmeter-for-api-performance-testing-a-comprehensive-guide. Accessed on September 9, 2025.

[7] Kaleido. [n. d.]. REST API Gateway | APIs for Blockchain Development. https://www.kaleido.io/blockchain-platform/rest-api-gateway. Accessed on September 9, 2025.

[8] L. Mostarda, D. Sestili, and L. Tesei. 2023. An investigation tool for BESU permissioned blockchain performance. In *Proceedings of the 5th International Workshop on Distributed Ledger Technology*. https://dltgroup.it/DLTWorkshop/PDF23/DLT_2023_paper_6574.pdf

[9] Nethereum. [n. d.]. Managing nonces. https://docs.nethereum.com/en/latest/nethereum-managing-nonces/. Accessed on September 9, 2025.

[10] G. A. Pierro, L. Cocco, and R. Tonelli. 2024. Besu vs. Quorum: Comparative Analysis in the Context of Simulated Energy Communities. In *CEUR Workshop Proceedings*. https://ceur-ws.org/Vol-3791/paper24.pdf

[11] M. Soelman. 2021. Permissioned Blockchains: A Comparative Study. *Student Theses* (2021). https://fse.studenttheses.ub.rug.nl/25270/1/Final%20Submission.pdf

[12] Solo.io. 2023. API gateway vs. load balancer. https://www.solo.io/topics/api-gateway/api-gateway-vs-load-balancer. Accessed on September 9, 2025.

[13] The Linux Foundation. [n. d.]. Hyperledger Caliper Documentation. https://hyperledger.github.io/caliper/. Accessed on September 9, 2025.

[14] The Linux Foundation. 2022. Performance SandBox, a new Hyperledger Lab, is a test ground for blockchain performance research. https://www.lfdecentralizedtrust.org/blog/2022/03/07/performance-sandbox-a-new-hyperledger-lab-is-a-test-ground-for-blockchain-performance-research. Accessed on September 9, 2025.

[15] Alan D. A. Veloso, Caio S. Silva, and Antonio L. S. Abelém. 2023. Desvendando o Desempenho do Hyperledger Besu: Uma Análise Experimental do Mecanismo de Consenso QBFT. In *Anais do Workshop de Trabalhos de Iniciação Científica e de Graduação do SBRC*. SBC, 43–56. https://sol.sbc.org.br/index.php/semish/article/view/36843