2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS) | 978-1-6654-9172-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/DAPPS55202.2022.00015

Gromit: Benchmarking the Performance and Scalability of Blockchain Systems

Bulat Nasrulin, Martijn De Vos, Georgy Ishmaev, Johan Pouwelse
Delft University of Technology
{b.nasrulin, m.a.devos-1, g.ishmaev, j.a.pouwelse,}@tudelft.nl

***Abstract*—The growing number of implementations of blockchain systems stands in stark contrast with still limited research on a systematic comparison of performance characteristics of these solutions. Such research is crucial for evaluating fundamental trade-offs introduced by novel consensus protocols and their implementations. These performance limitations are commonly analyzed with ad-hoc benchmarking frameworks focused on the consensus algorithm of blockchain systems. However, comparative evaluations of design choices require macro-benchmarks for uniform and comprehensive performance evaluations of blockchains at the system level rather than performance metrics of isolated components. To address this research gap, we implement Gromit, a generic framework for analyzing blockchain systems. Gromit treats each system under test as a transaction fabric where clients issue transactions to validators. We use Gromit to conduct the largest blockchain study to date, involving seven representative systems with varying consensus models. We determine the peak performance of these systems with a synthetic workload in terms of transaction throughput and scalability and show that transaction throughput does not scale with the number of validators. We explore how robust the subjected systems are against network delays and reveal that the performance of permissoned blockchain is highly sensitive to network conditions.**

***Index Terms*—benchmark, blockchain performance, reproducibility, stress testing**

I. Introduction

The rapid growth in the number of blockchain protocols in the past few years has been boosted by the interest in cryptocurrencies, decentralized finance, and identity systems. The solutions are mostly empirically driven, with direct economic incentives stimulating engineering experiments. To date, there are more than 700 different blockchain and distributed ledger platforms offering native digital assets and products.Many of these solutions deploy original families of consensus protocols or significant modifications of popular protocols, with variations in scalability, performance, and decentralization guarantees. These developments outpace systematization and research on inherent trade-offs of different design choices [1].

The absence of benchmarking solutions for comprehensive comparative analysis of various protocols is a particularly problematic omission, given the cumulative marketcap of 1,468 trillion $ for these projects. More fundamentally, this systematization gap hampers our ability to tackle the increasing complexity of blockchain systems and make conscious design choices in blockchain engineering. Developers of these protocols often provide performance metrics of blockchain solutions as declarative whitepapers that do not pass the standards of peer review and reproducibility, calling into question the reliability and objectivity of these measurements. For instance, it is a common practice to provide performance metrics of an isolated component and report them as a system-wide performance metric [2]–[4]. This practice often leads to false impressions of the end-to-end system performance.

Few available benchmarking solutions, such as Blockbench [5] and Hyperledger Caliper [6] focus on narrow sets of permissioned consensus protocols or DAG-based protocols as DAGBENCH [7]. There is also a noticeable deficit of academic research in macro benchmarks for blockchain systems. Existing studies are rather limited in scope either focusing on specific protocols such as Hyperledger [8], [9] or Ripple [10], or reusing Hyperldeger Caliper [11] and Blockbench [5]. BCTMark is one of the few comparative benchmark studies which compares three different protocols, including permissionless Ethereum blockchain [12]. The authors in this study highlight the necessity to extend the comparison set and include more metrics such as partition tolerance. The most recent systematic survey on performance evaluation of blockchain systems demonstrates that available comparative studies are rather limited in scope both in terms of compared systems and depth of analysis, focusing on isolated layers of blockchain systems [1]. To address this research gap, we design a benchmark that is comprehensive in scope, allowing us to stress-test the system under different network conditions.

We introduce *Gromit*, a generic benchmarking framework that allows a performance evaluation of *any* blockchain solution. Gromit treats each system under test as a transaction fabric, which means a transaction processing system where a group of peers continuously reach a consensus on transactions submitted by clients. Gromit analyses performance metrics related to transactional data, specifically throughput and latency. As we will show, this data alone can reveal the limitations of various aspects of the system.

We show the applicability of Gromit and conduct the *largest blockchain benchmarking study to date*. Our benchmark involves seven major blockchain solutions with different consensus algorithms. We determine the peak performance of these blockchain systems for different numbers of peers and without any modifications to the source code. A key finding is that the performance for most evaluated blockchain systems *degrades* when the number of peers increases. We also

This work was funded by NWO/TKI grant BLOCK.2019.004

DOI 10.1109/DAPPS55202.2022.00015

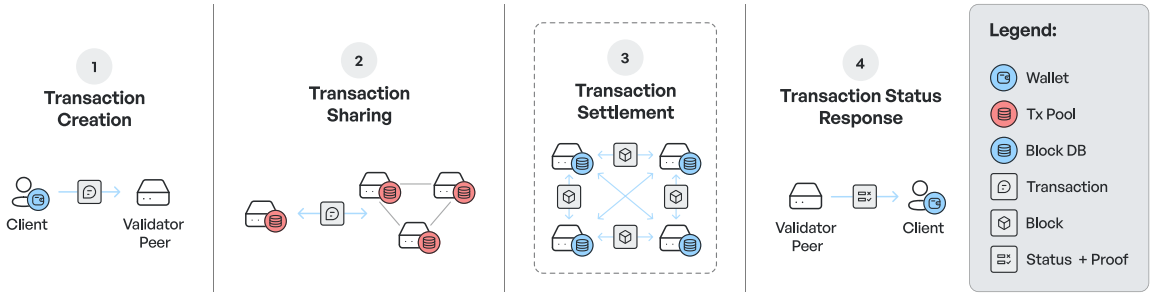


Fig. 1: Our abstraction model of blockchain system as a transaction fabric, comprising four stages. The dotted line highlights the focus of most whitepaper benchmarks.

reveal the effect of network delays and an increase in system load on the distribution of end-to-end transaction latencies, yielding valuable insights into the real-life bottlenecks of underlying consensus mechanisms. While most performance bottleneck research focuses on the consensus layer, a system-wide stress test can reveal bottlenecks in other layers, e.g., in the persistence layer.

The contribution of this work is two-fold:

1) We design and implement *Gromit*, a benchmarking framework that enables an analysis of *any* blockchain system (Section III).
2) We conduct *the largest blockchain benchmark to date*, measuring the performance of seven prominent blockchain systems (Section IV).

II. Blockchain as a Transaction Fabric

In this work, we view a blockchain system as a *transaction fabric*. Figure 1 visualizes this abstraction model, which illustrates a typical transaction life cycle.

In our model, we distinguish between clients and peers. Clients are instances that create transactions and send them to peers. An example of a client is a light wallet or lightweight nodes in a Bitcoin network. Peers in our model are responsible for processing and validating transactions in a shared, decentralized network. Thus, we call them *validator peers*. In some blockchain systems, they are also referred to as miners.

A. Transaction Life Cycle Model

1) Transaction Creation: A *transaction* contains logic that modifies the system state, e.g., by transferring an asset to another account. Each transaction is cryptographically signed with the private key of the issuing client to ensure authenticity. Blockchain solutions usually provide *Wallet* API's for clients to submit their transactions, e.g., with an RPC endpoint or REST endpoint.

2) Transaction Sharing: Blockchain systems employ complex transaction sharing mechanisms. Permissionless blockchains typically use a global gossip protocol to share transactions over a structured or unstructured overlay. Permissoned blockchains are deployed in a more controlled network environment and, as a result, might share transactions using a broadcast algorithm.

The transactions are stored in a datastore, often called a transaction pool (*TxPool*). The transaction pool is a temporary store used to queue or preprocess transactions before the network validates them.

3) Transaction Settlement: A consensus algorithm is a crucial part of the blockchain system as a mechanism for achieving security and liveness [13]. The intermediate outcome of the consensus process in blockchain systems is a set of valid transactions. Valid transactions are stored in a tamper-proof distributed ledger, a replicated data structure maintained by validators. The system discards invalid transactions.

Blockchain solutions typically bundle valid transactions in *blocks*, interlinked in a hash chain, and stored in a local database (*Block DB*). Each block in a hash chain contains the cryptographic hash of the previous block, making illegitimate modifications of the chain detectable. Some blockchain-like systems adopt a different organization of the distributed ledger, e.g., by maintaining a Directed Acyclic Graph (DAG) [7].

4) Transaction Status Response: After the transaction is settled the client waits for a transaction approval (or rejection), received from validator peers. Some blockchains also include a *proof* in the response that proves that a transaction is finalized.

B. Transaction Performance Indicators

Our approach is to analyze the performance of current blockchain solutions through *transaction benchmarking*. The speed at which a blockchain system processes transactions is a defining metric for blockchains. High transaction latencies directly impact end-users experience, e.g., the relatively high finalization times of Bitcoin transactions (10 minutes) make it unsuitable for interactive trade [14]. We include all stages of the transaction life cycle in our measurements, rather than focusing on the performance of the consensus layer only.

We obtain insights into the limitation of blockchain systems by measuring peak performance and associated transaction latencies. We also measure performance under different network conditions, such as changes in the geographical distribution of the overlay network. Our experiments in Section IV highlight that these metrics can reveal performance bottlenecks and act as a guideline for optimization efforts.

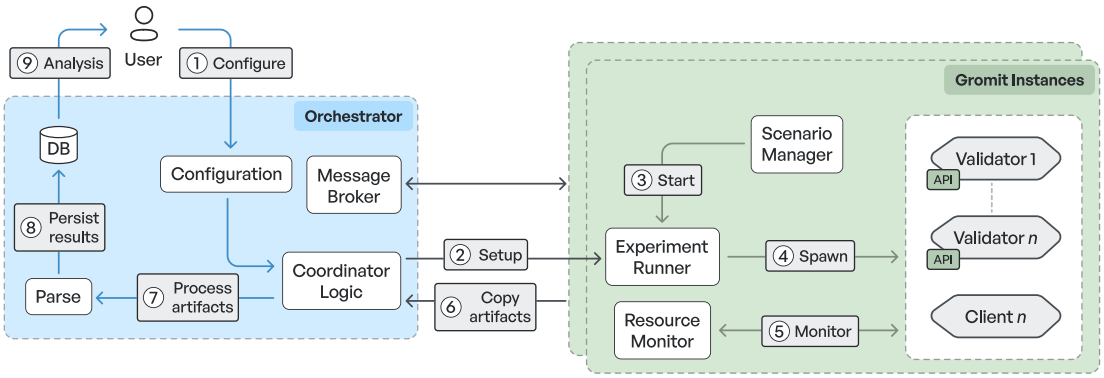


Fig. 2: Architectural overview and process flow of Gromit, our benchmarking framework for blockchain systems.

III. Gromit: A Generic Framework for Blockchain Benchmarking

We design and implement *Gromit*, the first generic framework for blockchain benchmarking. Gromit is designed for quick experiment iterations, utilizing a domain-specific language to devise experiments. Our framework, implemented in Python, is based on the abstraction model discussed in Section II. To encourage the adoption of Gromit, its source code is published on Github.[1] Gromit provides developers and researchers with the necessary tools to design benchmarks targeting blockchain performance. Gromit spawns client and validator processes on remote servers and coordinates interactions between running processes, e.g., transaction issuing.

A. Experiment Flow

Figure 2 shows the architecture of the framework and the flow of an experiment. This diagram is described next.

Experiment Setup. Before an experiment starts, it spawns a dedicated process, called the *orchestrator*, that setups the environment on remote servers and handles cross-server communication. The orchestrator reads the configuration file associated with the experiment. This file specifies the network addresses of remote servers and the number of clients and validators that should be started (step ①). The orchestrator then copies the source code and all necessary experiment files to the remote servers using `rsync`. Next, the coordinator setups the environment on the specified servers over an SSH connection (step ②). This step includes the installation of required system packages and the generation of a genesis file. This file describes the initial state of the blockchain system and can pre-load specific accounts with assets. The orchestrator then starts a *instance* for each client or validator on the remote server and assigns an identifier to each running instance. The scenario manager, part of the logic of a Gromit instance, parses a provided scenario file and starts the experiment (step ③).

Scenario Files. A user describes the actions performed during an experiment with a *scenario file*. The scenario manager parses this file and schedules actions using the `asyncio` library. Actions and their timestamps are explicitly denoted. A user can schedule an action to execute on a subset of all validators. Scenario files can be machine-generated and are a flexible approach to devising and conducting experiments.

Experiment Runner. The experiment runner spawns validators and clients as a subprocess during each experiment run (step ④). Clients interact with validators through the exposed API endpoint. We simulate a client in Gromit as a procedure that issues transactions. During an experiment, Gromit instances can share data over a TCP connection through a message broker. We use the message broker functionality to share the credentials of pre-defined blockchain accounts with clients. Gromit contains tools to gracefully terminate a running experiment, e.g., when a particular condition is not met.

Gromit also provides utilities to track system resource usage. Gromit instances monitor CPU, memory, disk, and network usage using the `procfs` library (step ⑤). These metrics allow us to estimate the system resource usage of blockchain systems. Developers can easily extend Gromit to monitor specific metrics, for example, the number of inbound network messages for a particular blockchain system.

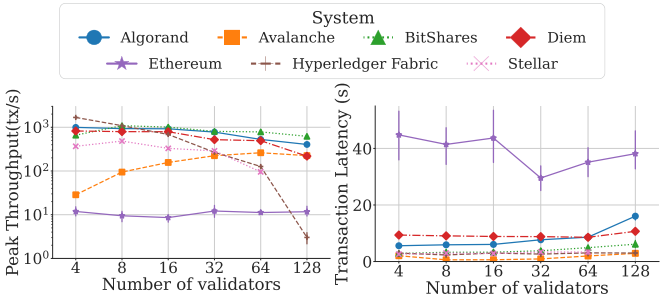
Collecting Experiment Results. When the experiment ends, the orchestrator copies all generated artifacts from the remote nodes using `rsync` (step ⑥). These artifacts include the data generated by the blockchain systems and the data output by the Gromit instances, e.g., monitoring statistics. This data is parsed by the orchestrator (step ⑦) and generates human-readable graphs. Finally, the data is stored in a database (step ⑧) and is ready for analysis by the user (step ⑨).

B. Integrating Blockchain Systems into Gromit

To show practicality of Gromit, we have integrated seven prominent blockchain systems into it. The integration requires no change in the source code of the blockchain systems. This allows to benchmark the blockchain system as close to the deployed systems as possible.

The design of Gromit is modular, and developers can implement *modules* that enrich an experiment with more functionality, for example, bandwidth monitoring. Integration of a blockchain system requires a developer to subclass the `BlockchainModule` and to implement the `init_configuration`, `start_validator`,

[1] See https://github.com/grimadas/gromit



(a) Peak transaction throughput (b) Transaction latency

Fig. 3: The peak throughput and transaction latencies of evaluated blockchain systems with the number of validators.

`stop_validator`, and `parse_ledger` methods. Gromit enables developers to specify custom network topologies to connect validators among each other. We refer readers who wish to integrate a particular blockchain system to the Gromit documentation.

Besides supporting blockchain systems, Gromit has support for generic experiments with distributed systems. For example, we have used Gromit to conduct experiments with peer-to-peer protocols on custom infrastructure.

C. Transaction Analysis

A workload during a Gromit benchmark consists of transactions issued by one or more clients. All transactions can be submitted to one validator peer or can be evenly spread among the peers. In line with related work, we mainly gauge blockchain performance using two metrics, transaction throughput, and transaction latency. Specifically, we consider the peak transaction *throughput*, which is the maximum rate at which the system can process transactions before getting congested. Second, we analyze the *latency* of transactions, which is the time between submitting a transaction and its irreversible inclusion in the distributed ledger.

We store the timestamp at which a client has submitted the transaction to a validator to determine transaction latency. However, determining the finalization time of transactions can be complicated since some blockchain systems do not expose granular, temporal information. We use the following two approaches to determine the latency of individual transactions. Our first approach is to inspect the resulting distributed ledger after all transactions have been issued (this logic should be part of the `parse_ledger` method). We then compute transaction latency based on timestamps included in ledger data elements (e.g., blocks). However, this approach is useful only if the blockchain system annotates data elements in the distributed ledger with a timestamp. The second approach suggests clients periodically poll the blockchain system to determine if a transaction has been confirmed.

IV. Blockchain Performance Evaluation

We conduct a diverse set of experiments with Gromit to reveal the performance characteristics and limitations of seven blockchain solutions. To the best of our knowledge, We are the first to perform a large-scale performance benchmark of blockchain solutions.

Since the introduction of Bitcoin in 2008, there have been many proposals for new blockchains and consensus mechanisms. Some proposals have materialized into operational systems, whereas other ones are only theoretically analyzed. The proliferation of blockchain solutions makes it infeasible to conduct a benchmarking study with all available systems. Based on taxonomy of different consensus mechanisms used in blockchain systems [13] we consider seven prominent consensus models and select a representative blockchain system for each consensus model. Our selection process is based on the economic magnitude, adoption, maturity of the system, and the protocol's academic significance. We note that the primary focus of our benchmarking study is on deployed layer-one blockchain systems. To this end, we integrate the Algorand (v2.3.0) [15], Avalanche (v1.1.1) [2], BitShares (v5.0.0) [16], Diem (v1.1.0) [17], Ethereum (v1.9.24) [18], Hyperledger Fabric (v1.4.9) [19], and Stellar (v15.1.0) [3] blockchain systems into Gromit.

Our experiments answer the following questions: How does the increase in the number of validators affect the system performance? What is the peak performance of each system under a heavy system load? What is the impact of a network delay on systems' performance? How consistent are our performance results compared to previously reported values?

Throughout the section we use two variables for our experiments: n indicates the number of validators and λ signifies the transaction throughput.

A. Setup and Transaction Workload

All experiments are conducted on four HPE DL385 Gen10 servers, located within the same data centre and interconnected with 10GB Ethernet links. Each server is equipped with 128 AMD EPYC 7452 CPUs, has 512GB of DDR4 memory, and runs Debian 10. During our experiments, we deploy each blockchain system with its source code unmodified. For each system, we use the default settings provided by the systems. Each experiment starts with only the genesis block included in the blockchain. We use a random network topology where each validator is connected to 10 other random validators.

We use Gromit to subject each system to a synthetic transaction workload, issued by up to 64 clients. To ensure an equal load on each validator, a client submits transactions to the validator with ID $i \equiv c \pmod n$ where c is the ID of the client and n the total number of validators. Each transaction is submitted to exactly one validator. Transactions are submitted during a two-minute period, after which we wait an additional minute for all transaction to be finalized.

We use simple asset transfers as a performance baseline. In our workload, a transaction issued by a client involves an asset transfer of a small, fixed amount to another account; the client counterparty is fixed throughout the experiment. We ensure that each client has sufficient funds to spend

during the experiment. For Ethereum and Hyperledger Fabric, transactions involve the transfer of an ERC20 token.

B. Determining Peak Transaction Throughput

To determine peak transaction throughput, we gradually increase the system transaction rate in steps of 100 transactions per second (tx/s). Based on the reported statistics by Gromit, we estimate the peak transaction load that each system is still able to process during a sustained period. If the system has any unconfirmed transactions after our two-minute period, we consider the system as "saturated". We evaluate the peak throughput of each system with an increasing number of validators (n). We provide each system with an equal amount of resources and ensure that the resource usage of evaluated systems (CPU power, disk space, and memory) does not exceed the available resources. Due to excessive resource usage of the Stellar software, we are only able to run Stellar with up to 64 validators. We run each experiment at least five times and average all results. Appropriate graphs are annotated with 95% confidence interval markers.

Finding 1. *Adding validators does not have a significant positive effect on the achievable peak transaction throughput of the evaluated systems.*

Figure 3 shows the result of our scalability experiment as n grows, in terms of peak transaction throughput and transaction latency. Figure 3a shows the peak transaction throughput of evaluated systems (with a horizontal and vertical log-axis). We notice that none of the evaluated systems can process over 1'000 tx/s with $n = 128$. In general, the transaction throughput of most of the systems is capped between 500 and 1'500 tx/s. Except for Ethereum, the peak transaction of all blockchains is *decreasing* as n increases. Specifically, Hyperledger Fabric shows a severe degradation in performance when $n > 8$, and is just capable of processing 2 to 4 tx/s with $n = 128$. We believe that this is caused by the underlying consensus model of Hyperledger Fabric, Raft, which does not scale well with the number of validators [20]. Of all systems, Ethereum has the lowest transaction throughput (around 10-20 tx/s), yet manages to keep stable throughput with the increase in the number of validators. Peak throughput of Avalanche increases up to $n = 64$, but then degrades for $n = 128$.

Finding 2. *For Avalanche, BitShares and Hyperledger Fabric, we observe significant discrepancies between the peak throughput found by us and previously reported values.*

The previous experiment estimates the peak throughput of blockchain systems, without modification to the source code and under default settings. We now compare the performance results with values reported by other literature. The performance, i.e. peak throughput, is typically determined through an evaluation by system developers themselves. For Stellar, we could not find reliable benchmarking results, making this work the first benchmarking study of Stellar. We are interested to see if there are significant inconsistencies with performance metrics reported by these studies.

Our performance results are comparable with results reported for Ethereum (4-40 tx/s [21]), Algorand (880 tx/s [15]) and Diem (200-1'000 tx/s [22]). However, we notice that previously reported values for some systems are significantly higher than our findings, specifically for Avalanche (7'000 tx/s, 26x higher [2]), BitShares (3'300 tx/s, 3x higher [23]) and Hyperledger Fabric (3'500 tx/s, 2 times higher [19]). For each system, we now explain these inconsistencies with additional experiments and analysis.

Avalanche. The relative low throughput of Avalanche surprises us and warrants further performance analysis. Since we noticed that each validator node is fully utilizing a CPU core, even with $n = 4$ and 32 tx/s, we perform a CPU analysis of deployed validators using the `pprof` profiler. We find that around 60% of CPU time is spent on hash computations using the `argon2` algorithm [24]. This CPU consumption originates from the API provided by Avalanche validators. Specifically, each validator maintains a keystore with credentials that is managed by end users; interactions with that keystore, e.g., accessing a private key, requires the user to include the password hash in the request. Consequently, many parallel requests to the API by clients cause severe performance degradations.

To analyse the impact of password hashing, we recompile Avalanche with this hash verification disabled and re-run our experiment. We do not observe a significant increase in transaction throughput. However, this reveals another performance bottleneck, originating from the verification of transactions, consuming around 90% of CPU time. Since Avalanche transactions are linked in a DAG structure, incoming transactions require the validation of parent transactions, which is a resource-intensive, recursive operation. We believe this can be addressed with further engineering efforts, e.g., queueing the verification of transactions.

BitShares. We further analyse the reported throughput of BitShares and found that the peak transaction throughput (3'300 tx/s) is not an accurate performance indicator since the sustained throughput throughout the experiment is only around 450 tx/s. Specifically, the achievable throughput of the BitShares consensus algorithm seems to be predicated by the speed of the slowest consensus participants in terms of connectivity and CPU resources. When operating BitShares in a heterogeneous environment, this can result in significant deviations in transaction throughput.

Hyperledger Fabric. We further analyse the results reported in the work of Androulaki et al. [19] and Blockbench [5]. We find that their work evaluates an early implementation of Hyperledger Fabric using a different consensus model (Zookeeper or PBFT). As such, these results are not directly comparable.

We also present the following two reasons to explain the discrepancies in reported and observed throughput numbers:

1) *Experimental Software vs Production*. Many of the throughput numbers reported by system developers are

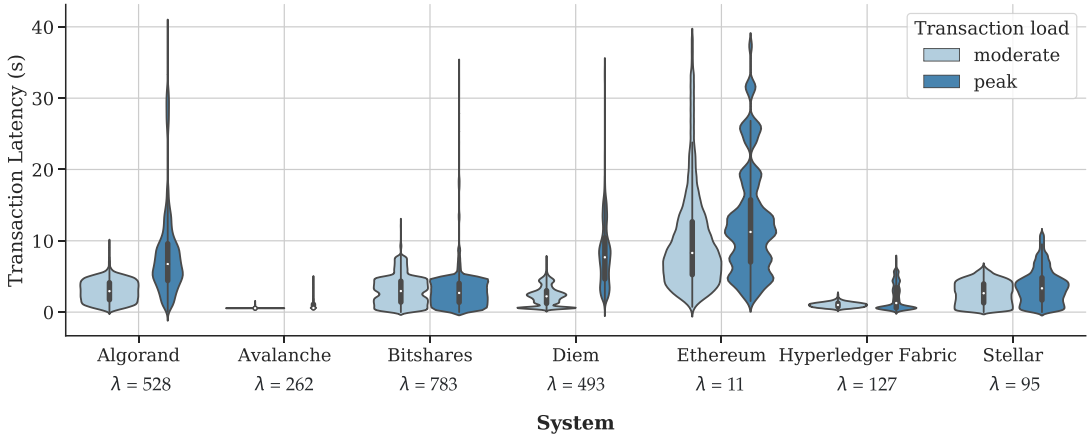


Fig. 4: The distribution of transaction latencies for systems, under peak load (λ tx/s) and moderate load ($\frac{\lambda}{2}$ tx/s) ($n = 64$).

extracted using a premature or even incomplete software implementation. As such, we argue that the values found by our experiments are a more accurate reflection of the achievable throughput in a production environment. Additionally, we noticed that some solutions (Diem and Stellar) have built-in measures that artificially lower the achievable throughput, likely to ensure safety properties or to prevent attacks in a production environment. A similar insights were observed in [25].

2) *Client-Validator Interaction*. In our experiments, clients submit transactions to the API exposed by the system, whereas other studies might directly inject transactions in the validator process. API-based interaction adds additional overhead as the request needs to be processed, and this approach therefore is likely to lower the peak throughput of the system. However, this approach resembles how users interact with validators when a blockchain system is Internet-deployed.

C. Transaction Latency

Finding 3. *For all evaluated systems, the average transaction latency under peak load is largely independent of the number of validators.*

Figure 3b shows the average transaction latency under peak load, as n increases. Except for Ethereum, the average transaction latency of evaluated systems is around or below ten seconds. The variance between different runs is relatively low, except for Ethereum. In general, the average transaction latency increases as the number of validators grows. The consensus algorithm underlying BitShares, Algorand and Stellar progresses in five-second rounds, theoretically resulting in an average transaction latency of 2.5 seconds. We see that the transaction latency of Algorand increases from 5 seconds for $n = 4$ to 15 seconds with $n = 128$, suggesting that consensus rounds take longer to complete. For BitShares and Stellar, this increase is less pronounced.

Finding 4. *The variance of transaction latencies for Algorand and Diem increases significantly under peak load, compared to a moderate load. However, the transaction latencies of BitShares and Ethereum are largerly indifferent towards the system load.*

We visualize the distribution of transaction latencies for each system to explore further the effects of increasing the system load on the transaction latency. We consider both peak and moderate loads, the latter being defined as half the determined peak load. Figure 4 shows this distribution in a violin plot. We observe that Algorand, BitShares, Hyperledger Fabric, and Stellar transaction latencies are roughly uniformly distributed under moderate load. These systems adopt a round-based consensus approach, with a target of around five seconds for Algorand, BitShares, and Stellar, and one second for Hyperledger Fabric. For these systems, most transactions are usually confirmed within the current or next consensus round relative to transaction submission. The distribution of transaction latencies transforms as the system is subjected to a peak load. Figure 4 shows that the finalization of Algorand transactions is being deferred to later rounds: 6% of Algorand transactions have a transaction latency above 20 seconds. This effect is less pronounced for BitShares and Stellar.

Increasing the system load impacts the latency distribution of Diem transactions. Further investigation reveals that the round duration in Diem adjusts to the system load. As more validators join the network and as more transactions are submitted to Diem validators, the round duration increases to ensure transactions can be processed on time. Nonetheless, 30% of all issued transactions in Diem are confirmed only after 10 seconds under peak load, whereas the system can handle all submitted transactions within 7 seconds under moderate load.

D. Impact of Network Delays

Finding 6. *Adding network delays has a minimal effect on the transaction latencies of Algorand and Stellar. However, Avalanche and Diem are extremely sensitive to network delays.*

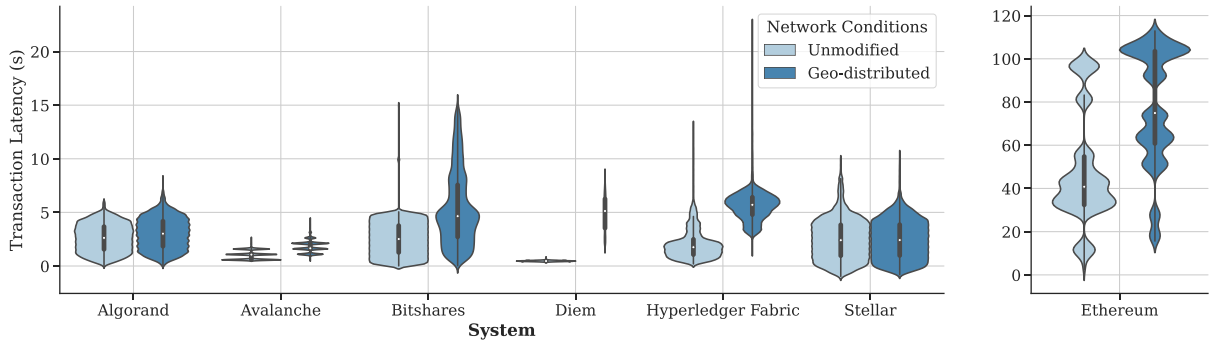


Fig. 5: Transaction latencies without network modifications and in a geo-distributed setting ($\lambda = 64$ tx/s, $n = 32$). The transaction latencies of Ethereum are shown in the right plot.

Finally, we modify the network settings using the `netem` Linux kernel module[2] and measure the impact of a network delay on the transaction latency for integrated blockchain systems. This experiment reveals how different blockchain systems are sensitive to deployment in geo-distributed settings. To replicate realistic network delays, we extract ping statistics for 32 cities worldwide from WonderNetwork [26] and assign each validator to a city in a round-robin fashion.

Figure 5 shows the distribution of transaction latencies of systems deployed in one data center with unmodified network settings and when deployed with geo-distributed settings. For each experiment run, we use 32 validators and fix the transaction load to 64 tx/s, which all evaluated systems can handle without congestion. Except for Algorand and Stellar, network delays visibly impact transaction latencies.

The effect is the most pronounced for the permissoned systems Diem and Hyperledger Fabric. This suggests that these systems can only operate in controlled/closed network environments, as any change in the network significantly affects the performance. We also observe a significant impact of network conditions on Avalance. This is due to the poll-based nature of metastable consensus. The poll rounds are visible in Figure 5.

For BitShares, we observe that validators occasionally fail to produce a block with higher network latency. For Diem, consensus progress stales a few seconds after starting the experiment, and rounds are timing out without confirming any transaction, violating system liveness.

E. Network and CPU Utilization

Finding 5. *Algorand, Stellar, Diem show high network utilization under idle load. Ethereum, Stellar and Diem consume significant CPU resources under idle load.*

We track the total network usage (inbound and outbound traffic) and average CPU utilization for each system while increasing λ and fixing n to 32. To obtain insights into the system under idle load, we also run blockchain systems with $\lambda = 0$ tx/s. The results are presented in Figure 6. Figure 6a shows that the network usage per validator quickly grows for Avalanche and Stellar as the transaction load increases. For $\lambda = 256$ tx/s, both Avalanche and Stellar use over 800 MB of network traffic per validator process. BitShares is the most network-efficient, using only 80 MB per validator for $\lambda = 256$ tx/s. We also observe that Algorand, Diem, and Stellar show 10x to 100x more bandwidth consumption under no transaction load compared to other systems.

Figure 6b shows the average CPU utilization when increasing λ. The mining process of Ethereum is continuously utilizing a single CPU. Under $\lambda = 256$ tx/s, BitShares and Hyperledger Fabric are the most CPU-efficient compared to other systems. Our synthetic workload results are consistent with real-world observations of liveness issues for Avalanche and Stellar [27], [28].

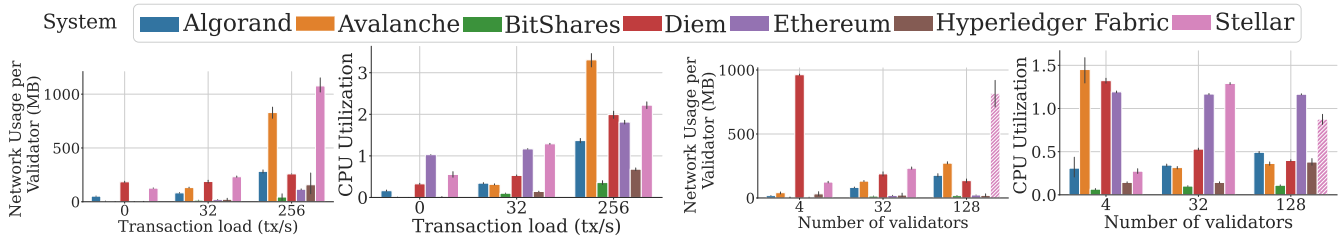
Figure 6c shows that Diem is using significant network resources for $n = 4$: 970 MB per validator process. This number decreases quickly when n increases. We explain this behavior by the self-adjusting round times of the Diem consensus mechanism: as n increases, rounds take longer to complete, lowering the bandwidth usage. For Stellar, we see the opposite effect: network usage becomes significant for $n = 128$. Although Stellar cannot process transactions for $n = 128$, we report its resource usage nonetheless. Inspection of Stellar logs reveals that validators lose track of consensus, clogging the network with resynchronization messages.

Figure 6d shows how CPU utilization behaves when increasing n. BitShares is the most CPU-efficient for all evaluated values of n. The CPU load of validators decreases for Avalanche and Diem as n increases. Since we fix the transaction load, adding more validators decreases the individual load.

V. Related Work

Blockchain benchmarking and its associated challenges has received attention from other researchers. Fan et al. present an extensive survey outlining methods for evaluating blockchain performance [1]. The work of Wang and Ye describes benchmarking tools and consensus mechanisms and outlines techniques to improve the throughput of blockchains [29]. The

[2]TC documentation: https://www.linux.org/docs/man8/tc-netem.html



(a) Network usage vs λ (b) CPU utilization vs λ (c) Network usage vs n (d) CPU utilization vs n

Fig. 6: The resource utilization of the evaluated systems with increase in transaction load (with fixed $n = 32$) or with the increase in number of validators (with fixed $\lambda = 32$).

authors of these studies summarize work on blockchain performance but do not conduct benchmarking studies themselves.

Popular blockchain benchmarking tools are Blockbench [5], Hyperledger Caliper [6], and DAGBench [7]. Blockbench, introduced in 2017, is the earliest benchmarking framework for blockchain and is specifically designed to evaluate permissioned blockchains [5]. Blockbench measures the performance of components commonly found in blockchains, e.g., the transaction execution engine). The authors of Blockbench evaluate the performance of Hyperledger Fabric, Ethereum, and Parity. Hyperledger Caliper is a benchmarking tool for the performance evaluation of specific systems with a set of pre-defined use cases [6]. Hyperledger Caliper primarily supports projects by the Hyperledger Foundation. DAGBench is a benchmarking tool for DAG-based blockchains [7]. The authors evaluate three popular DAG-based blockchain implementations [30]: IOTA, Nano, and Byteball. However, DAGBench does not allow for a comparison of other blockchains. The authors of BCTMark [12]. present an Ethereum performance evaluation.

VI. Conclusion

We have presented Gromit, a generic benchmarking framework for blockchain solutions. By treating each blockchain system as a transaction fabric, our framework enables any blockchain system's integration, benchmarking, and performance analysis. We leverage the functionalities of Gromit and conduct the largest blockchain benchmark to date, involving seven prominent blockchain systems. Our main finding is that none of the evaluated solutions can handle beyond 1'000 transactions per second as the number of validators increases. Yet, they show relatively low transaction latencies on average. We also find that synthetic workloads can provide accurate predictions of performance limitations, as our findings are consistent with real-world observations on performance incidents on Avalanche and Stellar networks.

References

[1] C. Fan *et al.*, "Performance evaluation of blockchain systems: A systematic survey," *Access*, 2020.

[2] T. Rocket *et al.*, "Scalable and probabilistic leaderless bft consensus through metastability," *arXiv preprint arXiv:1906.08936*, 2019.

[3] M. Lokhava *et al.*, "Fast and secure global payments with stellar," in *SOSP*. ACM, 2019.

[4] B. Cao *et al.*, "Performance analysis and comparison of pow, pos and dag based blockchains," *Digital Communications and Networks*, 2020.

[5] T. T. A. Dinh *et al.*, "Blockbench: A framework for analyzing private blockchains," in *ICMD*, 2017.

[6] "Hyperledger Caliper," https://github.com/hyperledger/caliper, 2018.

[7] Z. Dong *et al.*, "Dagbench: A performance evaluation framework for dag distributed ledgers," in *CLOUD*. IEEE, 2019.

[8] H. Sukhwani *et al.*, "Performance modeling of hyperledger fabric (permissioned blockchain network)," in *NCA*. IEEE, 2018.

[9] M. Kuzlu *et al.*, "Performance analysis of a hyperledger fabric blockchain framework: throughput, latency and scalability," in *IEEE international conference on blockchain (Blockchain)*, 2019.

[10] M. Touloupou *et al.*, "Towards a framework for understanding the performance of blockchains," in *BRAINS*. IEEE, 2021, pp. 47–48.

[11] M. Dabbagh *et al.*, "Performance analysis of blockchain platforms: Empirical evaluation of hyperledger fabric and ethereum," in *IEEE IICAIET*, 2020, pp. 1–6.

[12] D. Saingre *et al.*, "Bctmark: a framework for benchmarking blockchain technologies," in *AICCSA*. IEEE, 2020.

[13] S. Bano *et al.*, "Sok: Consensus in the age of blockchains," in *AFT*, 2019.

[14] T. Bamert *et al.*, "Have a snack, pay with bitcoins," in *P2P*. IEEE, 2013.

[15] Y. Gilad *et al.*, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*. ACM, 2017.

[16] F. Schuh and D. Larimer, "Bitshares 2.0: General overview," 2017.

[17] M. Yin *et al.*, "Hotstuff: Bft consensus with linearity and responsiveness," in *SPDC*, 2019.

[18] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[19] E. Androulaki *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *EuroSys*. ACM, 2018.

[20] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX ATC*, 2014.

[21] S. Pongnumkul *et al.*, "Performance analysis of private blockchain platforms in varying workloads," in *ICCCN*. IEEE, 2017.

[22] J. Zhang *et al.*, "Performance analysis of the libra blockchain: An experimental study," in *HotICN*. IEEE, 2019.

[23] "BitShares Industrial Performance and Scalability," https://bitshareshub.io/industrial-performance-and-scalability/, 2017.

[24] A. Biryukov *et al.*, "Argon2: new generation of memory-hard functions for password hashing and other applications," in *EuroS&P*. IEEE, 2016.

[25] J. A. Chacko, R. Mayer, and H.-A. Jacobsen, "Why do my blockchain transactions fail? a study of hyperledger fabric," in *SIGMOD'21*, 2021.

[26] WonderNetwork, "Global ping statistics," https://wondernetwork.com/pings, accessed: 2022-05-12.

[27] P. O'Grady. Preliminary analysis of the invalid minting bug. Accessed: 2022-05-12. [Online]. Available: https://medium.com/avalancheavax/preliminary-analysis-of-the-invalid-minting-bug-bee940cbd9e9

[28] M. McSweeney, "Stellar hit by transaction issues, developers say 'network is still online' despite node outage," *The Block*, 2021, accessed: 2022-05-12. [Online]. Available: https://www.theblockcrypto.com/linked/100649/stellar-network-stoppage-developers-investigation

[29] R. Wang *et al.*, "Performance benchmarking and optimization for blockchain systems: A survey," in *ICBC*. Springer, 2019.

[30] H. Pervez *et al.*, "A comparative analysis of dag-based blockchain architectures," in *ICOSST*. IEEE, 2018, pp. 27–34.