Multilayer Environment and Toolchain for Holistic NetwOrk Design and Analysis

Filip Rezabek, Kilian Glas, Richard von Seck, Achraf Aroua, Tizian Leonhardt, and Georg Carle
TUM School of Computation, Information, and Technology, Technical University of Munich
Germany

Abstract

The recent developments and research in distributed ledger technologies and blockchain have contributed to the increasing adoption of distributed systems. To collect relevant insights into systems' behavior, we observe many evaluation frameworks focusing mainly on the system under test throughput. However, these frameworks often need more comprehensiveness and generality, particularly in adopting a distributed applications' cross-layer approach. This work analyses in detail the requirements for distributed systems assessment. We summarize these findings into a structured methodology and experimentation framework called METHODA. Our approach emphasizes setting up and assessing a broader spectrum of distributed systems and addresses a notable research gap. We showcase the effectiveness of the framework by evaluating four distinct systems and their interaction, leveraging a diverse set of eight carefully selected metrics and 12 essential parameters. Through experimentation and analysis we demonstrate the framework's capabilities to provide valuable insights across various use cases. For instance, we identify that a combination of Trusted Execution Environments with threshold signature scheme FROST introduces minimal overhead on the performance with average latency around 40 ms. We showcase an emulation of realistic systems behavior, e.g., Maximal Extractable Value is possible and could be used to further model such dynamics. The METHODA framework enables a deeper understanding of distributed systems and is a powerful tool for researchers and practitioners navigating the complex landscape of modern computing infrastructures.

Keywords: Evaluation, Distributed Systems, Reproducability, Testbed

1 Introduction

Since the introduction of Bitcoin in 2008 [64], we have seen increased activity in distributed and decentralized systems research and solutions. The area of Blockchain-based solutions is of particular interest, with the introduction of Layer-1 protocols such as Algorand [55], Aptos [37], Cosmos [61], or Ethereum [44] offering advances and extensions on consensus and execution layers. Furthermore, there is a notable emphasis on advanced cryptographic protocols. Some of these protocols rely on threshold cryptography for secure private key protection, while also incorporating privacy features through Zero Knowledge Proofs (ZKPs) at the application layer for users [40]. Recent developments have also focused on scalability, utilizing Zero Knowledge (ZK) rollups to increase Transaction per Second (TPS) [73]. Additionally, Trusted Execution Environments (TEEs) are employed for secure computation [63]. Other innovations include privacy-preserving networks like Nym [48] and distributed storage solutions such as the InterPlanetary File System (IPFS) [41]. Often, these heterogenous systems interact on different application stack layers for improved performance, security, usability or privacy. For instance, ZK rollups built on top of underlying Layer-1 solutions, TEEs, or threshold cryptosystems, should run within the consensus mechanisms of a corresponding blockchain [42] or as an off-chain solution. Privacy-preserving networks, e.g., Nym, offer unlinkability on the networking layer for various deployments.

As modern distributed systems continue to evolve, they become increasingly complex. To grasp their practical implications and potential improvements, it's crucial to understand each component in detail. This requires precise control over the deployment environment. Such insights can help the core developers of given systems to identify and document relevant parameters for the best performance. Simultaneously, research can identify optimization approaches on individual layers, do system modeling, and investigate particular extensions and their impact on the system.

When examining cross-layer approaches, we identified that existing frameworks exhibit shortcomings in terms of modularity, upgradeability, and their ability to comprehensively assess broader distributed protocols, like threshold cryptography. Some solutions tend to focus on specific aspects or a restricted set of metrics and parameters [56, 65]. Notably, prior evaluations of large-scale systems, particularly those tied to cloud deployments [56], or those constrained by limited hardware resources for scalability assessments [65], may not accurately reflect real-world system dynamics and could introduce artifacts that affect measurement results.

We propose a methodological approach to assess various solutions on a common platform and observe their interactions and possible implications on scenario-specific Key Performance Indicators (KPIs). The methodology considers the deployment strategies, suitable experiment design, and systematizes experiment metrics and parameters.

We focus on local deployments that limit artifacts, affecting the reproducibility and interpretability. Also we need to ensure scalability and versatility, comparable to cloud deployments. Also, to handle the complexity of large-scale distributed systems, we have to define fitting experiment methodology applicable to current and future versions of the systems. This goes in hand with having a tight and granular control not only over the load generation [56], but even more importantly, on the cross-layer setup phase of such systems, as the conditions under which a system is tested can affect the results.

In the related work analysis, we investigate various frameworks towards facilitation of the identified requirements and suitability for future extensions. Rezabek et al. [67, 68] introduced a framework called ***En**vironment for **G**eneric **I**n-vehicular **N**etworking **E**xperiments* (EnGINE). Its implementation [16] relies on Ansible [1] and provides the foundation for basic infrastructure deployments, with an emphasis on finely-tuned experiment setups. Its extensibility allows it to not only support current e.g., blockchain systems, but also be future-proof. This sets it apart from other single-layered frameworks [56, 65]. While EnGINE does have some limitations, which we address in this paper, we recognized its potential for large-scale deployments. We recognize EnGINE as a suitable base for extension towards large-scale deployments.

In this paper, we present the **M**ultilayer **E**nvironment and **T**oolchain for **H**olistic **N**etw**O**rk **D**esign and **A**nalysis (METHODA), an extension of EnGINE. We integrate TEE and lightweight virtualization solutions for scalable systems into the base system. Since the focus is on distributed solutions, we decided to integrate three sample services - Algorand, Ethereum 2.0, and the FROST [59] threshold signature scheme based on Schnorr's algorithm [71]. We also conduct a baseline evaluation of TEE-based deployments and combine it with the FROST [59] threshold signature scheme. Lastly, we validate our approach and the METHODA capabilities by evaluation in a dedicated Hardware (HW) infrastructure. The results show an initial assessment of the capabilities and a verification of the methodological approach. The TEE in a combination with threshold protocol introduces negligible overhead on the End-to-End (E2E) delay, which was already shown by the baseline evaluations. Next, we are able to emulate a particular scenario in the blockchain space focusing on Maximal Extractable Value (MEV) extraction on Ethereum 2.0. This can serve as a base for further modelling of such dynamics on Ethereum 2.0 but can be also applied to other solutions. We do a detailed parameter study for Algorand with evaluation of latency TPS based on various HW specifications and peer's conditions. We further publish the codebase, presented results, and additional documentation as an online repository.

Overall we present the following Key Contributions (KCs):

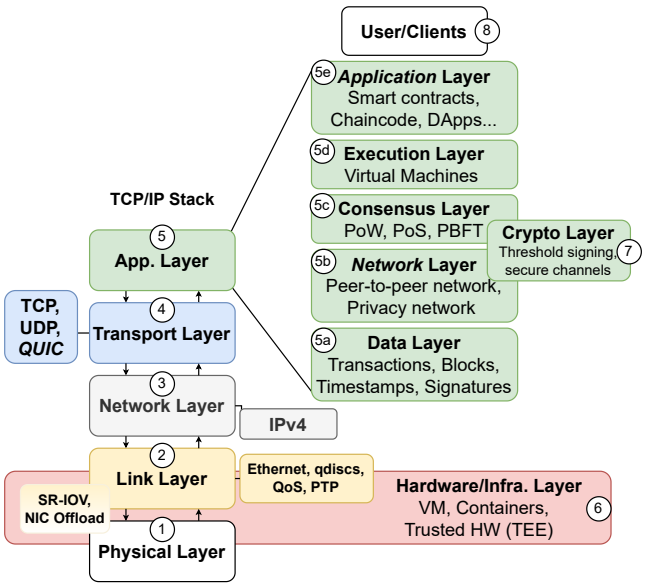


Figure 1. Deployment Stack and Layers, cf. [51]

KC1 Methodological approach to assess various distributed systems performance and their possible integration
KC2 Detailed analysis of related work focusing on blockchain frameworks
KC3 METHODA - an enhancement to the EnGINE framework for deploying large-scale distributed systems
KC4 Comparison of two recent blockchain systems and one threshold signature scheme
KC5 Evaluation of a TEE relying on recent Virtual Machine (VM) based solutions

2 Problem Analysis

This section offers a comprehensive view of the METHODA distributed system stack. We introduce a methodological approach and requirement description to address the challenges we identified in Section 1. Additionally, we conduct a comparative analysis with related studies, quantifying the research gap that motivates our approach. Finally, given the plethora of research on evaluating the performance and scalability of blockchain and agreement systems, we distinctly recognize their limitations our work aims to solve.

2.1 Distributed Systems Application Stack

Distributed systems, reliant on the TCP/IP stack, facilitate data exchange on a global scale. As illustrated in Figure 1, experimental frameworks must accommodate the entire application and ISO/OSI stack for comprehensive distributed system assessment. Within this stack, blockchains and the deployment of side-chain or Layer-2 solutions, can be viewed as separate systems. These systems need to interact with one another, and our goal is to facilitate this interaction within

the proposed framework. In the following, we provide additional information on layers shown in Figure 1.

① refers to the Physical Layer, a supplementary consideration. Nevertheless, if compatible with upper-layer protocols, the underlying transport medium can be modified. In ②, Ethernet is envisioned with support for queuing discipline (qdisc) configuration, Precision Time Protocol (PTP) [21] usage on Network interface cards (NICs), Single Root I/O Virtualization (SR-IOV), and specific NIC configuration and offload capabilities. SR-IOV enables scalability of interfaces and better utilization of the underlying link and PTP enables precisely synchronized clocks. On the Network Layer ③, we focus on IPv4 utilization, where IP addresses are directly assigned to corresponding interfaces. The Transport Layer ④ should be protocol-agnostic but provide options for execution via e.g., TCP, UDP, or QUIC.

Distributed systems predominantly operate on the Application Layer ⑤, relying on underlying network protocols and infrastructure. The Data Layer ⑧ handles various data types, distributed among individual peers in an overlay Network Layer ⑦ using schemes like gossip or privacy-preserving networks. In permissioned and permissionless ledgers, the Consensus Layer ⑨ deploys Sybil resistance mechanisms (e.g., Proof-of-Stake (PoS), Proof-of-Work (PoW)) and consensus algorithms (e.g., Practical BFT (PBFT) [46], Gasper [45], or Algorand's Byzantine Agreement (BA) [55]). Resilience against crash or byzantine failures can be achieved via the State Machine Replication (SMR) approach [70]. Therefore, for Crash Fault Tolerant (CFT) and Byzantine Fault Tolerant (BFT) systems, we aim to identify configurations of thresholds under which the system behaves as expected or starts to deteriorate. The Execution Layer ⑩ runs smart contract logic in a corresponding ledger VM, e.g., Ethereum Virtual Machine (EVM), Algorand Virtual Machine (AVM). The Application Layer ⑪ houses the logic of smart contracts and Distributed Applications (DApps).

To ensure secure data exchange beyond distributed ledgers, incorporating additional distributed protocols like threshold cryptosystems is required. Therefore, we introduce a parallel Cryptography Protocol Layer ⑦ positioned between layers ⑦ and ⑧. We note that in native deployments (e.g., only running a threshold signature protocol), the additional layers may not carry the same level of importance.

Another essential element is the underlying Hardware and Infrastructure Layer ⑥. Consequently, we not only facilitate bare-metal deployments but also extend support to various lightweight container solutions such as Linux Container (LXC) [20], Kata [19], or Docker [2]. While the system may potentially be improved by Linux Container Daemon (LXD) [31] for comprehensive VM deployments in the future, the current selections offer all requisite features without additional overhead. Additionally, the underlying infrastructure provides supplementary features, including the recent integration of confidential computing capabilities via TEE, as well as other hardware acceleration capabilities provided by the Central Processing Unit (CPU) or Graphics Processing Units (GPUs). Finally, it is crucial to emulate users/clients in the system ⑪ and ascertain their interaction with it, as they furnish vital insights regarding the load and its pattern.

2.2 Potential Use-Case Scenarios

We now introduce some exemplary use cases for METHODA. For evaluation of both distributed system in general, and Blockchain systems in particular METHODA facilitates thorough assessment of Peer-to-Peer (P2P) network properties, such as latency, bandwidth, and node connectivity, crucial for a comprehensive understanding of network performance. Such insights can be later used to confirm or model metrics, such as worst-, average-, and best-case latencies, among others. In context of MEV analysis, our solution allows to shed light on transaction ordering strategies, their influence on blockchain security, and the overhead impact of various privacy-preserving techniques. Furthermore, the framework facilitates examining node operator overhead, offering insights into resource requirements to ensure optimal system scalability and reward calculation. It also allows analysis of communication protocol replacement, extension, and configuration, providing a controlled environment for rigorous experimentation with various networking approaches.

2.3 Evaluation Methodology

This section outlines the methodology we employ to effectively evaluate distributed systems' performance and functionality. We discuss deployment strategies, considered experiment design, as well as metrics and parameters.

2.3.1 Deployment Strategies. In terms of deployment capabilities, we identify two options – simulation and emulation. For emulation we further consider either a private network or data centers (cloud). Based on the related work, we see that simulators [72, 74] focus only on particular scenarios and might provide limited insights to cross-layer and infrastructure impact, including networking properties of the system. However, their main advantage is a pure focus on a given algorithm's scalability without introducing noise and other artifacts affecting involved peers. For the emulation deployments, we see both examples in the related work, e.g., Gromit [65] relies on private infrastructure, whereas Diablo [56] on cloud deployments. The main argument for cloud deployments is simple access to more computational nodes, which can improve the scalability of the evaluation. However, a critical shortcoming is experiment reproducibility, especially for configurations and investigation of layers ② - ④, as the underlying experiment conditions change depending on too many factors, such as a load in the used data centers, latencies among the networks, and many others. Such noise and instabilities make collecting precise insights challenging. In comparison, private networks do not have such issues as

all experiments are executed in a controlled environment. Therefore, our approach focuses on private networks and enabling scalability within them while ensuring tight control over the environment and removing unwanted noise. Nevertheless, the framework should be extensible to deployments in the cloud, as it can still be a relevant approach, for instance, long-term probes used to collect data about active systems or single-node experiments.

2.3.2 Experiment Design. Our approach uses private network infrastructure to guarantee precise insights and control. To address scalability, we first consider the experiment design for large-scale systems. First, we discuss the applicability of white box and black box testing. White box testing requires a detailed code analysis and allows interaction with code to collect new insights, e.g., how expensive a given function is. When considering white box testing, introducing code changes is not welcome as in fast-paced environments, e.g., blockchain codebases, introduced changes might soon become obsolete. As an example of such changes, Ethereum's Geth client [8] observed 14 releases and more than 800 commits in one year (1 October 2022-2023). Similar holds for Algorand's client [4] with 17 releases. Nevertheless, white box testing methods are encouraged in the sense of understanding the code's functionality, undocumented behavior, and configuration parameters. In case one needs to profile a given code, using external tools, e.g., perf [26], would be a preferred option. On the other hand, black box testing considers only the system's external behavior based on a provided input such as a load. Therefore, aiming at black box testing for fast-paced systems without code modification is more suitable. As a result, we use white box understanding of the code and use it to improve experiment campaigns while relying on the systems' output to collect relevant insights.

For a suitable experiment methodology it is essential to address the diversity in hardware capabilities among peers within the system. Identifying the minimal hardware specifications that still allow for targeted system performance and sustainable operation is critical. Thus, the ability to emulate various configurations becomes crucial for pinpointing the optimal balance between performance and operational costs of the underlying infrastructure. Furthermore, for scenarios where the performance of specific peers in the production system deteriorates, perhaps due to outages or deteriorating network conditions, it is essential to be able to emulate the system's properties under such adverse circumstances.

To implement these approaches, we consider both local and global perspectives within the system. The local view emphasizes individual peer performance, optimizing resource allocation. In contrast, the global view assesses overall performance based on collective peer contributions. This dual approach offers a comprehensive understanding of system behavior. Additionally, this strategy necessitates a thorough examination of each layer. A feasibility study for every component ensures fair evaluation conditions and provides insights into their combined performance potential.

Term	Definition
	Metrics
Throughput	Rate of executed target operations
Latencies	Latency between e.g., adding a transaction to a block, round-trip-time, processing time, end-to-end latency,...
Finality	DLT and Blockchain context, Timeout for trust in transaction fixation.
Queue size	Amount of data in a queue/mempool
CPU,RAM,I/O	Local view compute time, RAM usage, I/O
	Parameters
Node number	Number of participants in different roles
Thresholds	Threshold values that determine the system's security and availability guarantees (e.g., amount of redundancy in BFT context)
Runtime con g	Config options that determine networking, processing, and protocol versions
Message size / payloads	Message size and contents of, e.g., votes, transactions, blocks, signatures
HW specs	HW specification of system nodes
Network params	Applicable to private networks (but also cloud and simulations), component specific configuration of delays, packet loss, ...
NIC con g	NICs configuration and offloading features
Load	Workload generation profile, e.g, requests, transactions and smart contracts
Fees	DLT and Blockchain context, dynamic (transaction) fee configuration
Faults	Introduction of faults to assess SUT capabilities in edge cases

Table 1. Identified Metrics and Parameters

2.3.3 Experiment Metrics & Parameters. To comprehensively evaluate a System Under Test (SUT), we define key metrics and parameters tailored to specific experiments. While TPS and finality are commonly discussed in blockchain contexts [56, 65], they offer only partial insights. To address this, we extend the configuration space to allow for evaluation of additional KPIs. Table 1 outlines eight identified metrics and ten parameters. Our aim is not to provide an exhaustive list, but to demonstrate possible experiment dimensions when designing an evaluation framework. Given the rapid evolution of systems, periodic updates to measurements and parameters will be essential. Therefore, the evaluation framework should be designed with flexibility and extensibility in mind to adapt to present and future requirements.

2.4 Requirements Definition

We formalize METHODA requirements (**R**) based on the defined application stack in Figure 1 and the evaluation methodology introduced in Section 2.3. The EnGINE [67, 68] authors introduced 13 relevant requirements, but their definitions must be extended to fit our context and needs, as discussed in Section 2.1. First, we cover the requirements introduced by EnGINE, which we do not amend. The general motivation and focus on ***R1: Repeatability***, ***R2: Reproducibility***, and ***R3: Replicability*** in experiments is highly applicable to our scope. To enable those, deployment in private infrastructure, avoiding (e.g., geodistributed) cloud infrastructure is valuable [36]. For usability, we rely on ***R4: Openness*** (e.g., open source) solutions that are, and use, openly available components. Similarly, to handle large-scale deployments, the experiments must run in ***R6: Autonomy*** (without human interaction) once defined. Second, we list amended and newly defined requirements. Insights about the system's behavior in ***R7: Malfunction Scenarios*** is essential. The framework must offer capabilities to emulate crash and byzantine failures in SUTs to study their robustness and the effect of relevant configuration parameters (e.g., redundancy). This directly leads to ***R9: Granular Control***. Starting on the node level, we need to have the option to directly allocate given resources, e.g., number of CPU cores, RAM, or NIC configurations. Continuing up the deployment stack (Figure 1), each individual SUT offers distinct configuration parameters. These encompass e.g., the number of peers within the system, communication protocols between them, or transaction fees in blockchain environments. It is imperative to permit the adjustment of such parameters to study their implications in real-world deployments. The objective is to attain a high level of control over as many aspects of the SUT as possible.

Experiment scalability (***R12: Scalability***) stands as an essential requirement. Our focus extends beyond support and actual measurement of large node count, also including factors like system load (e.g., TPS), wallet numbers, and the overall volume of requests and interactions within the system. Our approach complements real-world deployments, owing to the ***R14: Diversity*** of SUT types, log data formats, and number of experiment metrics it supports. Unlike centralized applications, distributed systems, typically lack a global perspective on system state and configuration. The operator can gather local logs, traffic loads, and general telemetry. However, given the distinct data generated by systems like Algorand or Ethereum, this presents challenges. To enable meaningful comparisons between different approaches and systems, a ***R15: Standardized*** configuration scheme is an essential criterion. This means that, similar configuration and postprocessing options (e.g., according to an abstract template) can be applied across various SUTs. The aim is to minimize variability from configuration differences, allowing results to rather reflect the SUTs intrinsic capabilities.

Fulfillment of the discussed requirements aids with achieving a set of general evaluation goals. Given ***R6: Autonomy***, ***R7: Malfunction Scenarios***, ***R9: Granular Control***, and ***R12: Scalability***, a framework can facilitate experiments mirroring real-world environments, accounting for factors like peer ratios in production versus local setups, corresponding workloads, network characteristics, etc. A modular framework architecture allows for easy integration of new SUTs. It also facilitates maintaining compatibility to dynamically changing SUT upstream codebases. All of these characteristics help acquiring insights into bottlenecks, robustness and performance behavior of target SUTs.

2.5 Related Work Analysis

Over the past years, we have seen many evaluation solutions focusing on classical BFT systems and permissioned or permissionless blockchains. In our assessment, we have identified distinct categories of frameworks. Some are tailored to single systems or their specific families [7, 17, 38, 52, 53, 57], while others, like Gromit[65] and Diablo[56], aim to provide a more generalized evaluation approach for specific blockchains. An assessment, if these systems facilitate the requirements from Section 2.4, is summarized in Table 2.

2.5.1 Classical Agreement-Focused Frameworks. BFT-Bench introduced in [57] emulates various types of faults, collects metrics such as delay and throughput, and considers underlying system CPU or network utilization. The authors claim implementation of six BFT protocols, including PBFT [46] and Zyzzyva [60]. Since the source code is not available, actual scalability potential and standardization are unknown. Details on post-processing are not provided. In a similar direction, Paxi [38] implements Paxos [62] and its variants [3], allowing for linearization checks and framework-level fault injections, for e.g., network and node failures. Presented measurements were conducted in a geodistributed cloud deployment. More recent works introduce and use the Bamboo [52, 53] evaluation framework, a rework of the Paxi codebase, to prototype and test chained-BFT protocols. The authors implement support for multiple HotStuff [77] variants as well as Streamlet [47]. The source code is available online[10]. Bamboo allows for load definition and distribution of additional configuration files among peers. It supports a simulation and deploy mode, running on several physical nodes or a single machine. The authors state that their measurements were conducted in a cloud deployment, albeit with all machines placed in the same datacenter. Lastly, a pure simulation approach is introduced by BFTSim [72] building on the NS-2 network simulator [22]. It allows for measurement of various latencies and throughput and various workloads to be provided. No reference to simulator sources is given. Tool [6, 74] is a more recent, open-source simulation framework, implementing a selection of classical agreement algorithms and variants (PBFT,

	[57]	[38]	[52]	[72]	[74]	[49]	[65]	[56]	Us
Type	⊘	y	y	ζ	ζ	z	z	z	z
R1: Repeat	✓	×	✓	✓	✓	✓	✓	×	✓
R2: Reproduce	×	×		×	✓	✓		×	✓
R3: Replicate	×	×	×	×	✓	✓	×	×	✓
R4: Openness	⊘	✓	✓	⊘	✓	✓	✓	✓	✓
R6: Autonomy	✓		✓	✓	✓	✓	✓	✓	✓
R7: MalScenario	✓			✓	✓		×	×	✓
R9: Gran.Con.				⊘					✓
R12: Scalable	⊘			✓	✓		✓	✓	✓
R14: Diversity		×	×						✓
R15: Standard	⊘			⊘	×				✓

Table 2. Analysis of Related Work Solutions. ✓ full, partial, × no, ⊘ unknown satisfaction, ζ pure simulation, y Implement algo in framework sourcecode, z Runs third-party code

HotStuff) but also protocols, aimed at large-scale blockchain deployments (Algorand [55] Consensus). The system allows additional integration of various attacker types.

2.5.2 Blockchain-Focused Evaluation Frameworks.

BlockBench [49] introduces a benchmarking framework using smart contract cost. It allows for tunable workloads and collection of various metrics, e.g., scalability and fault tolerance insights [14]. It works with PoW Ethereum and Hyperledger Fabric [39]. For Hyperledger evaluation the authors introduce Hyperledger Caliper [17] that recently added support for Ethereum. A purely EVM focused solution, starting with Ethereum, is Chainhammer [7]. According to the paper, measurements were conducted in private network of commodity machines. Gromit [65] aims to be a generic evaluation framework for seven blockchain systems [11]. While the authors conducted measurements with emulated network delay, automatable fault injection is not implemented in the framework. Experiments were run in a cloud deployment, with machines located in the same datacenter. Lastly, the Diablo framework [56] similarly studies latencies and throughput for a range of Blockchain systems. Additionally, general load profiles, smart contracts, and regular transactions are considered. Measurements were conducted in geodistributed cloud deployment. An open-source implementation is available [13], automatable fault-injection is not implemented.

2.5.3 Additional Systems.

Additionally, we explore evaluations of other systems that our framework aims to support. A simulator for Nym is introduced in [66], which measures the system's performance and models its latency based on the number of mix nodes and modeled anonymity set size. In the realm of TEE, [75] introduces TEE-based evaluations of various VM-based solutions. For low-level assessments of Zero-Knowledge Proofs (ZKP), zk-Bench [50] presents a framework to evaluate ZK circuits and arithmetic.

2.5.4 Summary.

The analyzed systems from Table 2 come in three types: Pure simulators (ζ), frameworks that require implementation of a target algorithm within the native framework source code (y), and frameworks that orchestrate and run third-party code (z). While simulators help avoiding setup-related artifacts, insights into complex, real-world deployments is limited. The best approximation of realistic scenarios can be provided by frameworks that orchestrate original codebases in real-world stacks (z). Systems, relying on (geodistributed) cloud deployments, potentially offer less strict service guarantees, thus allowing for measurement artifacts and variance. Private network deployments with granular control better facilitate ***R1-R3***. Frequent partial satisfaction in ***R9*** results from limited configuration capabilities below the application layer. No competitor to METHODA offers automated HW allocation or configuration. While only true orchestrators (z) effectively handle SUT codebases, logs, and post-processing, most competitors focus on a very limited set of evaluation metrics (e.g., throughput and latency), affecting ***R14***. In conclusion, our analysis highlights the demand for a true orchestrator (z), facilitating replicability, that also excels in ***R9***, ***R14***, and ***R15***. This includes the ability to evaluate a wider range of metrics, parameters, and experiment methods. Such a framework should not only address classical BFT or permissioned and permissionless blockchains but also support a broader spectrum of distributed systems to enable research of potential synergies between them.

3 Design & System Architecture

To satisfy all the outlined requirements defined in Section 2, we outline our design decisions and provide more details about the METHODA architecture.

3.1 Requirements Satisfaction

To ensure ***R1***, ***R2***, and ***R3***, we start by clearly outlining the capabilities of the employed HW and Software (SW) versions. All experiments are conducted in a controlled environment and, free from additional noise. Our code repository is publicly accessible. The SW artifacts we used are predominantly open-source solutions and Commercial off-the-Shelf (COTS) hardware, ensuring both ***R4*** and helping to establish ***R6***.

3.1.1 Scalability.

For large-scale deployments, an experiment setup with up to tens or hundreds of nodes is crucial (***R12***). Starting with the Link Layer (⊘ , Figure 1), we use SR-IOV. Unlike traditional virtualized environments, where network traffic passes through the hypervisor, SR-IOV bypasses this bottleneck by direct communication with the physical NIC. As a result, it significantly reduces CPU overhead and enhances network performance [54]. This is achieved by partitioning the resources of a physical NIC to multiple Virtual Functions (VFs), where each VF acts as an independent Peripheral Component Interconnect (PCI) function with its

own configuration space and capabilities, e.g., qdisc configurations. The VFs can be assigned to individual virtualization solutions, granting them direct access to the physical NIC.

To emulate realistic deployments and increase the number of peers, we introduce, among others, lightweight virtualization on the Infrastructure Layer (⑥ , Figure 1). From many variants among application- or system containers, we selected LXC system containers. LXC is an Operating System (OS) system-level lightweight virtualization method that allows multiple isolated Linux systems, known as containers, to run on a single host. Unlike an application container, such as e.g., Docker [2], LXC's lifecycle is longer and are managed similar to VMs. On the other hand, unlike traditional VM, LXC operates at the kernel level, enabling efficient resource utilization [76]. Key components for LXC are Linux namespaces and Control Groups (cgroups) to create isolated application environments. Namespaces provide process and network isolation, file system views, and more, while cgroups manage resource allocation, including CPU, memory, and disk I/O. Each LXC container is allocated one of the created VF interfaces enabled for the corresponding NIC. LXC supports a variety of Linux distributions, allowing containers to run different OS distributions and versions. Also, LXC can be extended by LXD for deployments of full VM if needed. Similarly, to integrate and support TEEs at scale, we use Kata containers [19] tailored for the use-case of confidential computing. Kata is an open-source project and runs containers in its own lightweight VM, leveraging hardware virtualization technology that can be fully deployed in a TEE environment. This approach ensures strong isolation, making it more challenging for potential attackers to compromise the guest or host system and isolation among various containers running on the same system. Kata Containers' are compatible with orchestration SW like Kubernetes. They are also Open Container Initiative (OCI) [24] compliant. Therefore, the runtime environment supports various container images, facilitating a smooth transition from or to, e.g., Docker containers.

We rely on the VM-based solutions, introduced above, for TEE integration. The process-based solutions, exemplified by Intel Software Guard Extension (SGX), enable the creation of secure enclaves with restricted interaction. In contrast, VM-based deployments, such as Intel Trusted Domain Extensions (TDX) or AMD Secure Encrypted Virtualization (SEV), enhance virtual machine security through Secure Nested Paging, encrypting and isolating guest VMs from the hypervisor, and supporting nested virtualization. The Kata containers then fully run inside the trusted enclave provided by the underlying CPU, which is, in our case, AMD SEV-SNP [58]. Nevertheless, Kata containers allow for other TEEs technologies such as the Intel TDX once present in COTS.

3.1.2 Realistic Deployments. Emulating system realism in a controlled environment is a continuous process, especially when the deployed systems evolve. Therefore, it is crucial to have granular control over individual components in the infrastructure. In addition, it is important to design experiments that can reflect and properly abstract the complexity and state one can observe in the systems. For our experiments, we mainly rely on loads identified by related work and do additional analysis of *mainnet* (Blockchain context: Production network) data for certain Algorand and Ethereum 2.0 blockchains. We deploy several strategies to ensure proper emulation of such setups.

Starting on the Link Layer (② , Figure 1), we can emulate various network conditions using the Network Emulator (netem) [33] in combination with Multiqueue Priority Qdisc (MQPRIO) [32] qdisc. The module is controlled by the traffic control (tc) [34] functionality of the networking stack. qdiscs allow prioritizing and manipulating network traffic to ensure shaped transmission. Different qdiscs employ various algorithms to determine how they are processed. MQPRIO is a qdisc module that facilitates multi-queuing with prioritization. It divides network traffic into different queues, each with their own priority level. This allows for fine-grained control over which packets are processed first. Depending on the requirements, it allows for Quality of Service (QoS) and combination with additional child qdiscs. A relevant child qdisc is netem, which allows to introduce various network conditions like delay, jitter, packet loss, and packet reordering. By default, the same rule used is applied to all outgoing interfaces. Therefore we combine netem with MQPRIO, introducing the capability to modify traffic based on e.g., source IP address or port. We apply various netem configurations to each HW queue and use nftables [23] to map the corresponding traffic to the relevant HW queue on the Application Layer (⑨). We offer multiple options to configure dedicated resources on each node/peer/container, modify the NIC configuration, and scale the number of peers.

Similarly, due to the tight control on the Infrastructure Layer (⑥ , Figure 1), we can allocate dedicated resources to each node using either LXC or dedicated NIC configurations to emulate not only delays, but also, for instance, lower the throughput of a given interface and control what HW offload capabilities are being used. We use native interactions with each system (e.g., provided Software Development Kits (SDKs) or native calls on exposed Application Programming Interfaces (APIs)) and do not rely on any middleware layer. Therefore, any type of interaction can be emulated.

As outlined, a big challenge are versioning and regular changes. So, for each protocol we evaluate – e.g., Algorand and Ethereum 2.0 – we can also easily select different versions of node and protocol to evaluate and compare their current, previous, and, more importantly, future performance. METHODA supports collection of the metrics defined in Table 1 and due to tightly synchronized clocks, we allow for nanosecond precision on NICs that support IEEE 802.1AS [69]. We rely on the Linux PTP project [21] to handle time synchronization. The PTP project implements the

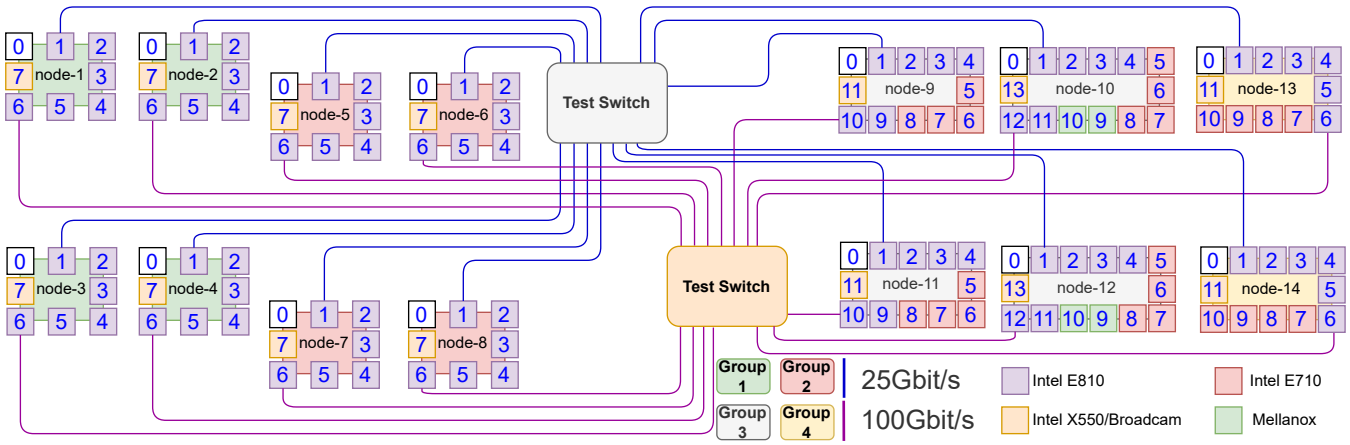


Figure 2. Simplified Topology used in our Deployments, Groups corresponds to HW Specification in Table 3

IEEE 1588 standard for PTP and can function over Ethernet, IPv4, or IPv6 networks. The *ptp4l* [29] daemon must run on all interfaces, enabling it to synchronize the system's clocks and identify the PTP Grandmaster Clock (GM), which serves as the reference time for the entire system. Additionally, the *phc2sys* [27] tool can synchronize clocks within a single node and operate in automatic mode, leveraging information from the *ptp4l* daemon to achieve synchronization. Due to the precise synchronization, we can inject specific faults and emulate dynamic changes in the system precisely at a given time and observe its effects. This allows for valuable insights into the reliability and security of the SUT (***R7***). Furthermore METHODA supports modification of the various threshold parameters, such as the number of expected active participants, stake distributions in the PoS systems, waiting parameters, expected synchronicity models in BFT settings, and others.

As outlined in Section 2.4, satisfying ***R9*** and ***R14*** facilitates realistic emulation of complex distributed systems. We employ Ansible and YAML for a standardized structure of experiments (***R15***) that can be easily ported and to ensure fair comparisons. Lastly, to support extentions with new metrics, parameters, and services, we strive for modularity the design of METHODA. This contributes to easier maintenance and compatibility to upstream changes of SUTs.

3.2 Integration into EnGINE

EnGINE is a key component of a broader orchestration approach for consistent, replicable, and verifiable networking experiments [67, 68]. It offers an adaptable experiment coordination tool, implemented in Ansible, which can be easily integrated with COTS hardware. Experiments conducted in EnGINE follow a structured process orchestrated through a management node, encompassing four key phases (Figure 3). The **1. Install** phase involves HW machine allocation, configuration, OS deployment, and is something we did not modify.

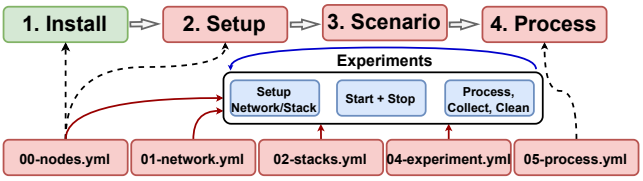


Figure 3. Overview of Extended Modules (highlighted in red) to EnGINE Framework [43]

The **2. Setup** phase installs all necessary dependencies. The actual measurements are executed in the **3. Scenario** phase, covering all experiments within a given campaign. The phase also includes network configuration, as well as the preparation and execution of applications for traffic generation and data collection. Each experiment is initiated and concluded, with this cycle repeating until all defined experiments for the scenario are finished. The collected results are then processed in the final **4. Process** phase.

We now discuss our modifications of EnGINE modules, starting with the **2. Setup** step. We implemented additional features related to SR-IOV, container technologies, and other aspects of virtualiziation. We also extended the experiment flow by relevant dependencies as well as network configuration changes. We no longer rely on Open vSwitch (OvS) since we want to generalize the framework with a focus on functionalities from the Network Transport Layer and above. We already identified suitable technologies that achieve scalability and realism using the Linux network stack (Section 3.1).

Similarly, the **3. Scenario** module now features new experiment campaigns for which we defined all relevant applications and services. For definition of individual experiments, we extended the `00-nodes.yml` by new options, related to e.g., the use of containers, but also network modes. After the configuration changes containers are treated similar to full nodes. Through this abstraction, no per-service changes are

Table 3. Node families used for experiments with HW specs, their count, and (relevant) supported standards by NICs

	CPU (cores/threads)	RAM	NICs
Group 1 (4x)	24C/48T Intel® Xeon Gold 6312U	512 GB DDR4	4 25 GbE E810-C[y], 2 100 GbE E810-XXV[y] 2 10 GbE X552[y]
Group 2 (4x)	32C/64T AMD EPYC 7543	512 GB DDR4	4 25 GbE E810-C[y], 2 100 GbE E810-XXV[y] 2 10 GbE BCM574[y]
Group 3 (4x)	32C/64T AMD EPYC 9354	768 GB DDR5	4 25 GbE E810-C[y], 2 100 GbE E810-C[y] 2 10 GbE BCM574[y], 4 10 GbE X710[y]
Group 4 (2x)	32C/64T Intel® Xeon Gold 6421N	512 GB DDR5	2 100 GbE E810-XXV[y], 2 100 GbE MT28908[y z] 2 10 GbE X552[y], 4 10 GbE X710[y], 4 25 GbE E810-C[y]
Total capacity	**416C/832T**	**8192 GB**	**On average 9 ports per node**

[y]IEEE 802.1AS, E810 family, E710, X552, and I210 are manufactured by Intel®
[z] Only two out of four nodes have this NIC

necessary and all deployments and tasks in the pipeline can be reused. The same applies to both LXC and Kata containers and Docker, respectively. `01-network.yml` now supports additional flags for interface and extended qdiscs configuration for the usage of netem. `03-stacks.yml` structure itself has not been extensively modified but instead extended by more than *ten* new services and additional flags for the processing pipeline. Lastly, we newly introduced `05-process.yml`, which contains additional metadata relevant to each scenario and its experiment run. Combining these aspects, we defined *15* experiment scenarios to collect relevant insights and validate our methodology and approach. We enclose a repository with artifacts and source code.

Last, we extended **4. Process** to support large-scale deployments and easier data access and processing via database storage. This includes various applications and various formats such as packet captures, logs, or .csv. These results are correlated and used for visualizations of our metrics (Table 1).

3.3 Infrastructure

Combining all design decisions, we deploy the framework to a private testbed. A simplified topology setup is shown in Figure 2, with all 14 nodes interconnected via dedicated test switches. The nodes are grouped based on their HW specifications and NICs. Table 3 introduces the NIC types and each group HW specifications in addition to the total capacity of the testbed. Figure 2 uses colors to differentiate NIC ports. For clarity, we mark each NIC family with a corresponding color. Of note, there is additional wiring between the nodes, but for readability reasons it is omitted. The additional connections are essential for the PTP synchronization. Depending on the system, we can scale our experiments to tens or hundreds of nodes represented by the LXC containers, based on the total capacity (Table 3).

For our experiments, we use Ubuntu 22.04 with a 5.15-lowlatency Linux kernel. This image our base OS for the Algorand, Ethereum 2.0, and FROST Signature scheme experiments. For Ethereum 2.0, we use go-ethereum version 1.20 [8] as an execution client and Prysm as a consensus client 4.0.4 [15]. For the Proposer-Builder Separation (PBS) relay we use version 1.11.5-0.2.3 [9]. Algorand uses the go-algorand release 3.18.1 [4]. The FROST library builds on top of the FROST-Dalek implementation [12] but was significantly extended by the communication stack and other parts. For Kata we also use Ubuntu, but with a version of 20.04 and Linux kernel version 5.19 with additional AMD patches to allow for deployments in the AMD SEV-SNP enclave [5]. Additional details can be found in the enclosed repository.

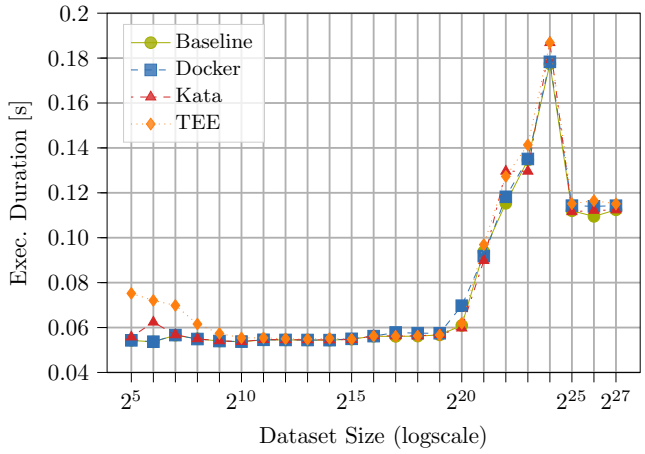
4 Evaluation & Validation

This section introduces the experiments outlined in Section 3.3, serving as a validation to our outlined methods.

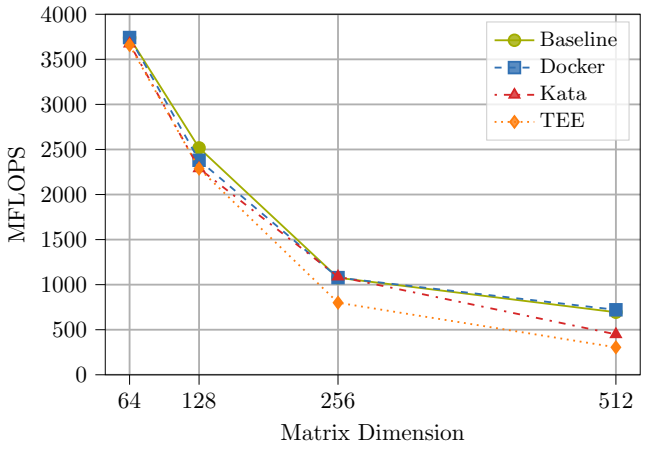
4.1 Feasibility Study

In this study, our objective is to analyze the effects of implementing the FROST scheme within a TEE to enhance the security of individual private key shares. TEEs are being explored for their potential applications in distributed systems, ranging from privacy preservation to security optimization [63]. We conduct separate performance assessments of both TEEs and the FROST threshold signature scheme, as well as their combined impact. This includes overhead comparisons for Kata containers in TEE vs. native deployments, and white box vs. black box testing for FROST. We maintain consistent configurations, varying only the virtualization technique. TEE-related experiments run on Group 3 nodes, while FROST without TEE runs on Group 2 nodes (Table 3).

In initial TEE experiments, we evaluate the application execution impact. For CPU-bound evaluations, we utilize the *triad* benchmark [25], while matrix multiplication is employed for memory-bound CPU tests, as illustrated in Section 4.1. The results depicted in Figure 4a reveal comparable



(a) Compute bound CPU Test - Triad Experiment



(b) Memory bound CPU Test - Matrix dimension

Figure 4. Comparison of Bare Metal, Docker, Kata, and Kata in TEE - RAM and CPU Tests

behavior across baseline (bare metal), Docker and Kata native deployments. Kata within the TEE, shows a spike for 2^{24} elements, attributed to caching effects. Similarly, Figure 4b displays consistent performance across various matrix dimensions and Mega Floating-Point Operations Per Second (MFLOPS) in the memory-bound test, indicating no significant performance degradation for Kata within the TEE.

4.1.1 White box and Black box testing.

After establishing the baseline performance of the TEE, we proceed with the evaluation of Schnorr's signature scheme. This involves employing both white box and black box testing methods to measure the E2E latency. The micro-benchmarking is supported by FROST-Dalek [12] and results are shown in Figure 5. This setup emulates the operations of a distributed scenario on a single node without a networking stack. Notably, nonce generation (preprocessing phase) is excluded.

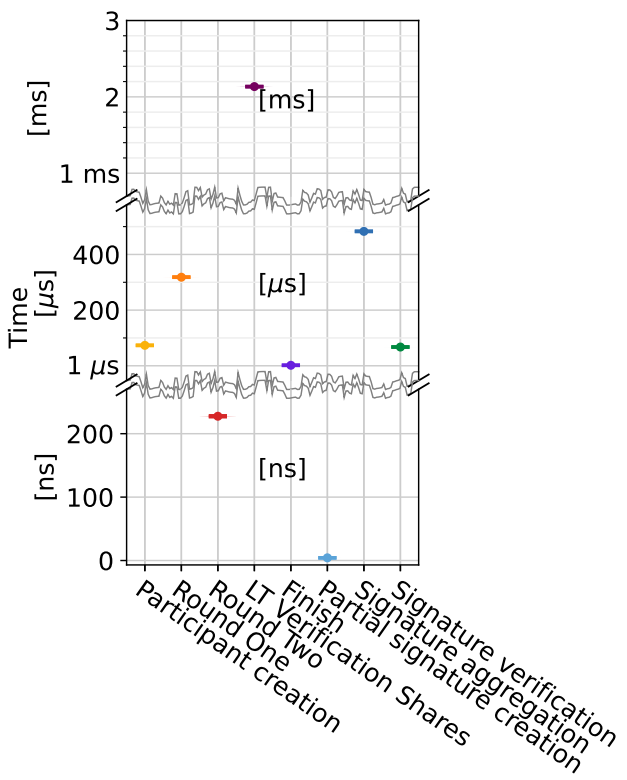
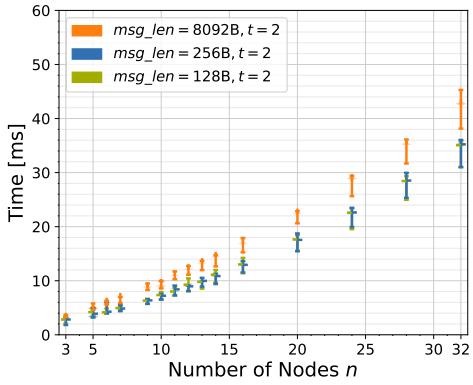


Figure 5. Threshold Schnorr, Whitebox Testing, $n = 3$, $t = 2$

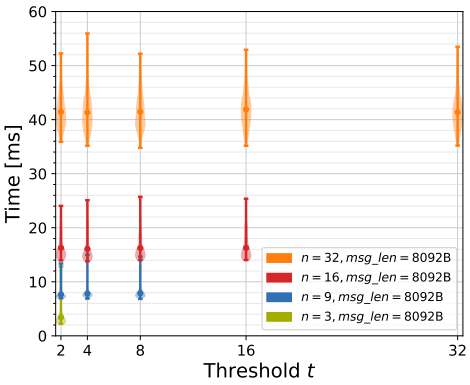
The benchmark breaks down performance into individual steps within each phase, providing a detailed assessment. Conducted on a Group 2 node (Table 3), each operation is executed approximately 500 times, with $n = 3$ and $t = 2$.

The Distributed Key Generation (DKG) phase consists of initial operations from *Participant Creation* until the *Finish* step. This phase also encompasses the *LT Veri cation Shares* step, where a participant computes long-term public key shares (Y_i) of its peers. Although not strictly part of DKG, our system includes it. The sign phase involves the *Partial signature creation* that is executed on each peer, followed by *Signature aggregation*—theoretically involving the broadcast of individual partial signatures to all other peers. Finally, the client executes the last step in our system, *Signature veri cation*. In summary, FROST-Dalek's DKG operation, without network communication, takes approximately 2 ms, while threshold signature generation requires around 0.5 ms.

Using black box testing, we assess the E2E latencies for signing operation while varying the parameters n and t, message size m, and execution within or without TEE (Section 4.1.1). Utilizing hosts from Group 2, we deploy up to eight LXC containers per node, each with fixed resources and without threading the signature application. To evaluate the scheme's scalability, we increase the node count to 32 and message size up to 8092 B. Using METHODAs dynamic



(a) Delay for n and Message Sizes - Without TEE



(b) Delay for Various t - With TEE

Figure 6. Threshold Schnorr signing End-to-End delay for various Message Sizes, n, and t values w/ and w/o TEE

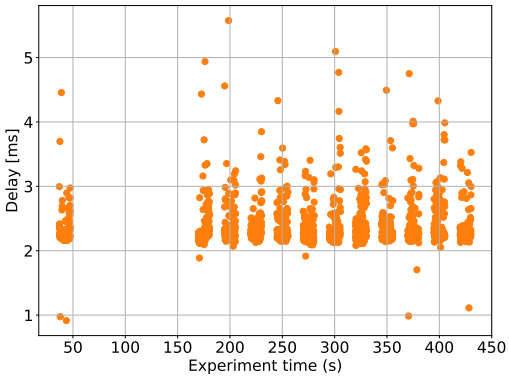


Figure 7. Ethereum 2.0 Client to Mempool E2E Latency using tcpdump

fault injection capabilities, we employ netem to introduce a delay among peers on the same physical machine, mirroring the delay among the rest of the peers.

In Figure 6a, we observe that the value of t has a less significant impact on signature generation. This is attributed to its influence being confined to the summation of partial signatures—an operation with low cost (as seen in Figure 5). Consequently, the summation of t partial signatures exhibits nearly constant complexity due to the marginal increase in t. Similarly, while the differences in means for various message lengths are measurable, they are less critical. Building on the findings from the study of Kata in TEE, we present Figure 6b. Here, the mean delay for specific values of n, t, and message size remains consistent, albeit with more outliers.

In summary, threshold Schnorr, in conjunction with TEE, achieves mean delays of up to 42 ms for 32 nodes and a message size of 8092 B. Comparing this with Figure 5, we observe an approximate 5x increase for signature generation, primarily attributable to communication overhead.

4.1.2 Applicability to Blockchain. From the perspective of a single peer, we are interested in measuring how long a client takes to process a single transaction. Instead of modifying the code of a client node, we want to measure the processing time of a node. Therefore, we measure E2E memory pool (mempool) confirmation measured on a client node using tcpdump [35]. Figure 7 indicates that the majority of transactions exhibit processing delays ranging from 2 to 3 ms. Note that no transaction is confirmed between 20 and 140 s, which is due to the execution clients waiting for the Terminal Total Difficulty (TTD) to be reached to switch to PoS [30]. A similar approach can be applied to metrics supported by the framework, especially due to the deployment of PTP for high precision for raw network data collection.

4.2 Use-case: Emulation of Ethereum's PBS

In a specific use-case, we emulate an MEV scenario of Ethereum 2.0. To mitigate the adverse effects of MEV, Ethereum 2.0 introduced PBS, enabling clients to send transactions directly to block builders [28], instead of validators directly creating blocks. Consequently, a double-order auction occurs before a block is dispatched to the proposer, involving three parties: searchers, builders, and relays. Each relay is linked to a group of block builders and selects the most profitable block from the connected set of builders. Given that searchers are continually on the lookout for MEV opportunities, they forward a transaction bundle (a list of transactions) to one or more builders. The builder's goal is to maximize profits by constructing the most lucrative block. This constitutes the first auction, where searchers bid for their bundles in competition with one another. Subsequently, builders compete for the relay's selection of their block, constituting the second auction. We now shift our focus to the first-level auction, specifically on the builder side, to extract the maximum value. We simplify the setup by assuming only a single builder.

To investigate this behavior, we emulate a scenario with continuously increasing MEV profit space, leading to increased transaction count and gas value increasing per block. Consequently, certain transactions arriving at the mempool

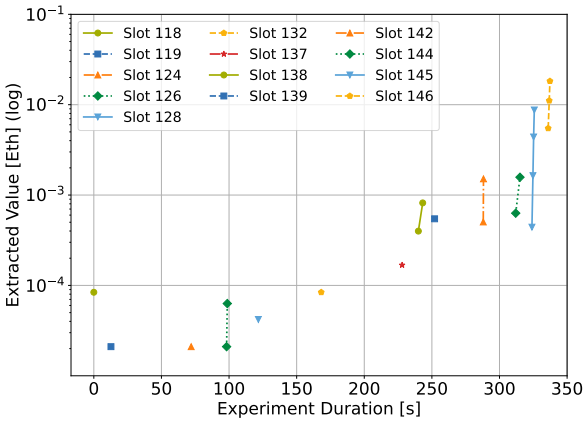


Figure 8. Ethereum Builder-Relay Extracted Value

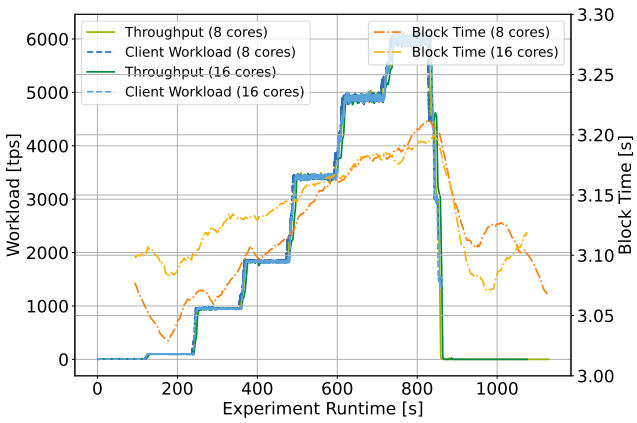


Figure 9. Algorand Throughput under varying Workload and HW Specifications

have higher gas fees. Out of a load of 20 TPS, 15 % have higher gas fees. The builder then dispatches updated blocks, extracting increased value in each slot, as depicted in Figure 8. This value incrementally rises with each new proposed block within the same slot, and cumulatively over the course of the entire experiment. This underscores the builders' objective to generate blocks prioritizing transactions with higher gas fees. Overall, this scenario emulates increasing MEV profit space, e.g., as present in a bidding competition for transaction inclusion in the subsequent block. As such, it is a basic representation that can be further refined with more sophisticated strategies and the integration of a second-order auction in the future.

4.3 Optimal Resources

Algorand's network differentiates between relay and non-relay (consensus participation and non-participation) nodes, where relays are mainly used to forward traffic to other nodes. Therefore, we investigate the impact of the HW specifications of relay nodes on the performance. Our Algorand network comprises four relay nodes, eight participation (ρ), and eight non-participation (η) nodes. Those nodes are distributed as LXC containers across four physical nodes with 64 virtual cores each, s.t. each physical node hosts one relay and two ρ-nodes and η-nodes each. Each η-node has a client container connected to it. Those clients generate payment transactions and forward them to the η-nodes. Both η-nodes and ρ-nodes have always eight virtual cores assigned to them. For the relays, we consider 8 and 16 virtual cores. Algorand recommends eight virtual and 16 cores for η-/ρ- and relay nodes, respectively [18]. Figure 9 shows the system's performance under increasing load profiles. We observe that both the relay setups can operate loads of up to 6000 TPS without reaching network congestion. The block time increases slightly with increasing load in both scenarios. In our restricted setting, we conclude that the number of relay cores does not significantly impact performance. In practice, a relay has to handle large numbers of concurrent connections to ρ and η nodes, which is not reflected in our experiment.

5 Challenges

We encountered several notable challenges while designing METHODA. The testbed's diverse hardware specifications led to varying base performances and capabilities. This diversity is common in real-world deployments, allowing us to conduct assessments that closely resemble realistic scenarios.

For experiment execution, we used a fixed distribution and kernel. Yet, many distributions with potentially distinct performance profiles are common in real-world deployments. Nevertheless, METHODA facilitates highly-automated and standardized testing of different systems and their respective versions, offering valuable insights into their impact.

6 Conclusion & Future Work

In this study, we unveil METHODA, an extension of the EnGINE framework, tailored to heterogeneous, large-scale distributed systems. This advance encompasses a comprehensive requirements and related work analysis, as well as the introduction of a sophisticated application stack and experiment methodology. These components cover a spectrum of experiment strategies and their corresponding metrics and parameters. Additionally, we discuss prerequisites and design choices for EnGINE integration in detail. We conduct a series of experiments to showcase the effectiveness and potential of our methodology within the framework. We present measurements of Ethereum 2.0 and Algorand, the FROST threshold cryptography scheme, and assess performance overhead of processing within a TEE. METHODA offers protocol developers and researchers a holistic approach to acquire in-depth system insights. We

validate METHODA on four SUTs, collecting *eight* metrics and modifying *12* parameters. For the integration of TEE with FROST, we observe minimal overhead on the performance with average latency around 40 ms. For Algorand, we see less impact by weaker HW specifications on the throughput but rather on latency. Last, emulation of realistic systems behavior e.g., MEV is possible and could be used to further model such dynamics.

For future work, we plan to integrate an even broader spectrum of applications, spanning from permissioned and permissionless blockchains to threshold cryptosystems and privacy-preserving networks, among others. Also, while cloud deployments may not be fitting for reproducible experiments, they hold promise as a pertinent infrastructure for sustained data collection, especially with the inclusion of network probes. Consequently, we envision extending the framework to facilitate deployments of such experiments in the cloud.

References

[1] [n. d.]. Ansible is Simple IT Automation. https://www.ansible.com/. (Accessed on 10/12/2023).

[2] [n. d.]. Docker: Accelerated Container Application Development. https://www.docker.com/. (Accessed on 10/12/2023).

[3] [n. d.]. GitHub - ailidani/paxi: Paxos protocol framework. https://github.com/ailidani/paxi. (Accessed on 10/15/2023).

[4] [n. d.]. GitHub - algorand/go-algorand: Algorand's official implementation in Go. https://github.com/algorand/go-algorand. (Accessed on 10/15/2023).

[5] [n. d.]. GitHub - AMDESE/AMDSEV: AMD Secure Encrypted Virtualization. https://github.com/AMDESE/AMDSEV. (Accessed on 10/15/2023).

[6] [n. d.]. GitHub - csienslab/BFT-Simulator: A simulator for testing/verifying/benchmarking Byzantine Fault-Tolerant (BFT) protocols. https://github.com/csienslab/BFT-Simulator. (Accessed on 10/15/2023).

[7] [n. d.]. GitHub - drandreaskrueger/chainhammer: fire many transactions at Ethereum node, then produce diagrams of TPS, blocktime, gasUsed and gasLimit, and blocksize. https://github.com/drandreaskrueger/chainhammer. (Accessed on 10/09/2023).

[8] [n. d.]. GitHub - ethereum/go-ethereum: Official Go implementation of the Ethereum protocol. https://github.com/ethereum/go-ethereum. (Accessed on 10/15/2023).

[9] [n. d.]. GitHub - flashbots/builder: Flashbots MEV-Boost Block Builder. https://github.com/flashbots/builder. (Accessed on 10/15/2023).

[10] [n. d.]. GitHub - gitferry/bamboo: Bamboo is a prototyping and evaluation framework that studies the next generation BFT (Byzantine fault-tolerant) protocols specific for blockchains, namely chained-BFT, or cBFT. https://github.com/gitferry/bamboo. (Accessed on 10/15/2023).

[11] [n. d.]. GitHub - grimadas/gromit: Decentralized Systems Benchmarking and Experiment Runner Framework. https://github.com/grimadas/gromit. (Accessed on 10/15/2023).

[12] [n. d.]. GitHub - isislovecruft/frost-dalek: An Rust implementation of FROST: Flexible Round-Optimised Schnorr Threshold signatures using the Ristretto group. https://github.com/isislovecruft/frost-dalek. (Accessed on 10/15/2023).

[13] [n. d.]. GitHub - NatoliChris/diablo-benchmark: The "DIstributed Analytical BLOckchain" Benchmark Framework. (Diablo). Measures blockchains with a focus on real-world applications and workload generation. https://github.com/NatoliChris/diablo-benchmark. (Accessed on 10/15/2023).

[14] [n. d.]. GitHub - ooibc88/blockbench: BLOCKBENCH: A Framework for Analyzing Private Blockchains. Blockbench contains workloads for measuring the data processing performance, and workloads for understanding the performance of different layers of Blockchain systems. https://github.com/ooibc88/blockbench. (Accessed on 10/15/2023).

[15] [n. d.]. GitHub - prysmaticlabs/prysm: Go implementation of Ethereum proof of stake. https://github.com/prysmaticlabs/prysm. (Accessed on 10/15/2023).

[16] [n. d.]. GitHub - rezabfil-sec/engine-framework. https://github.com/rezabfil-sec/engine-framework. (Accessed on 10/12/2023).

[17] [n. d.]. Hyperledger Caliper. https://hyperledger.github.io/caliper/. (Accessed on 10/09/2023).

[18] [n. d.]. Install a node - Algorand Developer Portal. https://developer.algorand.org/docs/run-a-node/setup/install/. (Accessed on 10/18/2023).

[19] [n. d.]. Kata Containers - Open Source Container Runtime Software | Kata Containers. https://katacontainers.io/. (Accessed on 10/12/2023).

[20] [n. d.]. Linux Containers. https://linuxcontainers.org/. (Accessed on 10/12/2023).

[21] [n. d.]. linuxptp. https://sourceforge.net/projects/linuxptp/. (Accessed on 10/12/2023).

[22] [n. d.]. The Network Simulator - ns-2. https://www.isi.edu/nsnam/ns/.

[23] [n. d.]. nftables(8) — nftables — Debian testing — Debian Manpages. https://manpages.debian.org/testing/nftables/nftables.8.en.html. (Accessed on 10/12/2023).

[24] [n. d.]. Open Container Initiative - Open Container Initiative. https://opencontainers.org/. (Accessed on 10/12/2023).

[25] [n. d.]. Optimizing Memory Bandwidth on Stream Triad. https://www.intel.com/content/www/us/en/developer/articles/technical/optimizing-memory-bandwidth-on-stream-triad.html. (Accessed on 10/19/2023).

[26] [n. d.]. Perf Wiki. https://perf.wiki.kernel.org/index.php/Main_Page. (Accessed on 10/12/2023).

[27] [n. d.]. phc2sys(8): synchronize two clocks - Linux man page. https://linux.die.net/man/8/phc2sys. (Accessed on 10/12/2023).

[28] [n. d.]. Proposer-builder separation | ethereum.org. https://ethereum.org/nl/roadmap/pbs/. (Accessed on 10/15/2023).

[29] [n. d.]. ptp4l(8): PTP Boundary/Ordinary Clock - Linux man page. https://linux.die.net/man/8/ptp4l. (Accessed on 10/12/2023).

[30] [n. d.]. Ropsten Merge Announcement | Ethereum Foundation Blog. https://blog.ethereum.org/2022/05/30/ropsten-merge-announcement. (Accessed on 10/18/2023).

[31] [n. d.]. Run system containers with LXD | Ubuntu. https://ubuntu.com/lxd. (Accessed on 10/12/2023).

[32] [n. d.]. tc-mqprio(8) - Linux manual page. https://man7.org/linux/man-pages/man8/tc-mqprio.8.html. (Accessed on 10/12/2023).

[33] [n. d.]. tc-netem(8) - Linux manual page. https://man7.org/linux/man-pages/man8/tc-netem.8.html. (Accessed on 10/12/2023).

[34] [n. d.]. tc(8) - Linux manual page. https://man7.org/linux/man-pages/man8/tc.8.html. (Accessed on 10/12/2023).

[35] [n. d.]. tcpdump(1) man page | TCPDUMP & LIBPCAP. https://www.tcpdump.org/manpages/tcpdump.1.html. (Accessed on 10/15/2023).

[36] 2020. Artifact Review and Badging - Current. https://www.acm.org/publications/policies/artifact-review-and-badging-current.

[37] 2022. The Aptos Blockchain: Safe, Scalable, and Upgradeable Web3 Infrastructure. https://aptos.dev/assets/files/Aptos-Whitepaper-47099b4b907b432f81fc0effd34f3b6a.pdf. (Accessed on 10/13/2023).

[38] Ailidani Ailijiang, Aleksey Charapko, and Murat Demirbas. 2019. Dissecting the Performance of Strongly-Consistent Replication Protocols. In *Proceedings of the 2019 International Conference on Management of Data*.

[39] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger fabric. In *Proceedings of the Thirteenth EuroSys Conference*. ACM. https://doi.org/10.1145/3190508.3190538

[40] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. Cryptology ePrint Archive, Paper 2014/349. https://eprint.iacr.org/2014/349 https://eprint.iacr.org/2014/349.

[41] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. arXiv:1407.3561 [cs.NI]

[42] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. 2023. SPRINT: High-Throughput Robust Distributed Schnorr Signatures. Cryptology ePrint Archive, Paper 2023/427. https://eprint.iacr.org/2023/427 https://eprint.iacr.org/2023/427.

[43] Marcin Bosk*, Filip Rezabek*, Johannes Abel, Kilian Holzinger, Max Helm, Georg Carle, and Jörg Ott. 2023. Simulation and Practice: A Hybrid Experimentation Platform for TSN. In *22nd International Federation for Information Processing (IFIP) Networking Conference*. Barcelona, Spain.

[44] Vitalik Buterin. 2013. Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform. (2013). https://github.com/ethereum/wiki/wiki/White-Paper

[45] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. 2020. Combining GHOST and Casper. arXiv:2003.03052 [cs.CR]

[46] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (New Orleans, Louisiana, USA) *(OSDI '99)*. USENIX Association, USA, 173–186.

[47] Benjamin Y Chan and Elaine Shi. 2020. Streamlet: Textbook Streamlined Blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 1–11.

[48] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. 2021. The Nym Network: The Next Generation of Privacy Infrastructure. https://nymtech.net/nym-whitepaper.pdf.

[49] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM international conference on management of data*. 1085–1100.

[50] Jens Ernstberger, Stefanos Chaliasos, George Kadianakis, Sebastian Steinhorst, Philipp Jovanovic, Arthur Gervais, Benjamin Livshits, and Michele Orrù. 2023. zk-Bench: A Toolset for Comparative Evaluation and Performance Benchmarking of SNARKs. Cryptology ePrint Archive, Paper 2023/1503. https://eprint.iacr.org/2023/1503 https://eprint.iacr.org/2023/1503.

[51] Caixiang Fan, Sara Ghaemi, Hamzeh Khazaei, and Petr Musilek. 2020. Performance Evaluation of Blockchain Systems: A Systematic Survey. *IEEE Access* 8 (2020), 126927–126950. https://doi.org/10.1109/ACCESS.2020.3006078

[52] Fangyu Gai, Ali Farahbakhsh, Jianyu Niu, Chen Feng, Ivan Beschastnikh, and Hao Duan. 2021. Dissecting the Performance of Chained-BFT. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 595–606.

[53] Fangyu Gai, Jianyu Niu, Ivan Beschastnikh, Chen Feng, and Sheng Wang. 2022. Scaling Blockchain Consensus via a Robust Shared Mempool. *arXiv:2203.05158* (2022).

[54] Sebastian Gallenmüller, Florian Wiedner, Johannes Naab, and Georg Carle. 2022. How Low Can You Go? A Limbo Dance for Low-Latency Network Functions. *Journal of Network and Systems Management* 31, 20 (28 Dec. 2022). https://doi.org/10.1007/s10922-022-09710-3

[55] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 51–68.

[56] Vincent Gramoli, Rachid Guerraoui, Andrei Lebedev, Chris Natoli, and Gauthier Voron. 2023. Diablo: A Benchmark Suite for Blockchains *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 540–556. https://doi.org/10.1145/3552326.3567482

[57] Divya Gupta, Lucas Perronne, and Sara Bouchenak. 2016. BFT-Bench: Towards a practical evaluation of robustness and effectiveness of BFT protocols. In *Distributed Applications and Interoperable Systems: 16th IFIP WG 6.1 International Conference, DAIS 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings 16*. Springer, 115–128.

[58] Felicitas Hetzelt, Martin Radev, Robert Buhren, Mathias Morbitzer, and Jean-Pierre Seifert. 2021. VIA: Analyzing Device Interfaces of Protected Virtual Machines. *CoRR* abs/2109.10660 (2021). arXiv:2109.10660 https://arxiv.org/abs/2109.10660

[59] Chelsea Komlo and Ian Goldberg. 2020. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. Cryptology ePrint Archive, Paper 2020/852. https://eprint.iacr.org/2020/852 https://eprint.iacr.org/2020/852.

[60] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2010. Zyzzyva: Speculative Byzantine Fault Tolerance. *ACM Trans. Comput. Syst.* 27, 4, Article 7 (jan 2010), 39 pages. https://doi.org/10.1145/1658357.1658358

[61] Jae Kwon and Ethan Buchman. 2019. Cosmos Whitepaper. https://wikibitimg.fx994.com/attach/2020/12/16623142020/WBE16623142020_55300.pdf.

[62] Leslie Lamport. 1998. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)* 16, 2 (1998), 133–169.

[63] Rujia Li, Qin Wang, Qi Wang, David Galindo, and Mark Ryan. 2022. SoK: TEE-assisted Confidential Smart Contract. arXiv:2203.08548 [cs.CR]

[64] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. (May 2009). http://www.bitcoin.org/bitcoin.pdf

[65] Bulat Nasrulin, Martijn De Vos, Georgy Ishmaev, and Johan Pouwelse. 2022. Gromit: Benchmarking the Performance and Scalability of Blockchain Systems. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 56–63.

[66] Ania M. Piotrowska. 2021. Studying the Anonymity Trilemma with a Discrete-Event Mix Network Simulator *(WPES '21)*. Association for Computing Machinery, New York, NY, USA, 39–44. https://doi.org/10.1145/3463676.3485614

[67] Filip Rezabek, Marcin Bosk, Thomas Paul, Kilian Holzinger, Sebastian Gallenmüller, Angela Gonzalez, Abdoul Kane, Francesc Fons, Zhang Haigang, Georg Carle, and Jörg Ott. 2021. EnGINE: Developing a Flexible Research Infrastructure for Reliable and Scalable Intra-Vehicular TSN Networks. In *3rd International Workshop on High-Precision, Predictable, and Low-Latency Networking (HiPNet 2021)*. Izmir, Turkey.

[68] Filip Rezabek*, Marcin Bosk*, Thomas Paul, Kilian Holzinger, Sebastian Gallenmüller, Angela Gonzalez, Abdoul Kane, Francesc Fons, Zhang Haigang, Georg Carle, and Jörg Ott. 2022. EnGINE: Flexible Research Infrastructure for Reliable and Scalable Time Sensitive Networks. *Journal of Network and Systems Management* 30, 4 (08 Sept. 2022), 74. https://doi.org/10.1007/s10922-022-09686-0

[69] Filip Rezabek, Max Helm, Tizian Leonhardt, and Georg Carle. 2022. PTP Security Measures and their Impact on Synchronization Accuracy. In *18th International Conference on Network and Service Management (CNSM 2022)*. Thessaloniki, Greece.

[70] Fred B Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A tutorial. *ACM Computing Surveys*

(CSUR) 22, 4 (1990), 299–319.

[71] Claus-Peter Schnorr. 1989. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings (Lecture Notes in Computer Science, Vol. 435)*. Springer, 239–252. https://doi.org/10.1007/0-387-34805-0_22

[72] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. 2008. BFT Protocols Under Fire.. In *NSDI*, Vol. 8. 189–204.

[73] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. 2022. Blockchain Scaling Using Rollups: A Comprehensive Survey. *IEEE Access* 10 (2022), 93039–93054. https://doi.org/10.1109/ACCESS.2022.3200051

[74] Ping-Lun Wang, Tzu-Wei Chao, Chia-Chien Wu, and Hsu-Chun Hsiao. 2022. Tool: An Efficient and Flexible Simulator for Byzantine Fault-Tolerant Protocols. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 287–294. https://doi.org/10.1109/DSN53405.2022.00038

[75] Xingyu Wang, Junzhao Du, and Hui Liu. 2022. Performance and isolation analysis of RunC, gVisor and Kata Containers runtimes. *Cluster Computing* 25, 2 (jan 2022), 1497–1513. https://doi.org/10.1007/s10586-021-03517-8

[76] Florian Wiedner, Max Helm, Alexander Daichendt, Jonas Andre, and Georg Carle. 2023. Containing Low Tail-Latencies in Packet Processing Using Lightweight Virtualization. In *2023 35rd International Teletra c Congress (ITC-35)* (Turin, Italy).

[77] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus in the Lens of Blockchain. arXiv:1803.05069 [cs.DC]