Architecture and Performance Comparison of Permissioned Blockchains Platforms for Smart Contracts

Guilherme A. Thomaz, Gustavo F. Camilo, Lucas Airam C. de Souza, Otto Carlos M. B. Duarte

Universidade Federal do Rio de Janeiro - GTA/COPPE/UFRJ

Abstract—**Blockchain and Smart Contracts ensure security and automation in trustless scenarios, leading to innovative solutions in various industry branches. The Hyperledger open-source project adopts these technologies in the corporate business, providing platforms for developing distributed applications. This paper analyses and compares two widely used platforms to develop applications based on permissioned blockchains: Hyperledger Sawtooth and Hyperledger Fabric. We implement two prototypes based on the same smart contract to evaluate the performance of each tool. The results show that: i) Sawtooth parallel transaction execution performs up to 30% better than serial execution only if the number of conflicting transactions remains low, and ii) Fabric has a much faster consensus protocol, but presents a low performance if the transactions are conflicting.**

I. Introduction

Blockchain is a disruptive technology for providing auditability, authentication, pseudo-anonymity, and non-repudiation in the transfer of digital assets with distributed trust [1], [2], [3]. Each network member maintains a copy of an immutable record of all transactions signed by their emitters. A consensus protocol guarantees that the same transactions are applied or discarded in every network node [4]. The blockchain can execute smart contracts, which implement arbitrary logic based on the transactions, allowing the development of distributed applications that go beyond cryptocurrencies [5].

The Linux Foundation Hyperledger project provides open-source platforms to promote permissioned blockchain and smart contracts technologies to corporate use cases. Permissioned blockchains require identification and permission from the user to access the network, creating a scenario where participants present a high level of trust. The project maintains six platforms to date, and two of them stand out as the pioneers and most used ones: Hyperlerledger Sawtooth, led by Intel, and Hyperledger Fabric, led by IBM. Identifying differences in architecture and evaluating the performance of the two platforms is essential for the developer.

This paper compares and analyses the architecture and performance of two Hyperledger permissioned blockchains platforms for developing smart contracts: Sawtooth and Fabric. The article presents the architecture's main differences. We develop two prototypes that execute smart contracts for the same application on both platforms to evaluate their performance. We use permissioned blockchains and smart contracts for providing access control to Internet of Things (IoT) data as a use case for the experiments [6]. The achieved results present a higher transaction throughput on Sawtooth using parallel transaction processing when the number of conflicting transactions remains low and a much higher throughput using Fabric Raft consensus protocol.

The remainder of this paper is organized as follows. Section II discusses related works. Section III details the architecture of Hyperledger Sawtooth and Fabric platforms. Section IV presents the performance evaluation results based on the prototypes developed in both platforms. Section V concludes the paper and presents the directions for future work.

II. Related works

Blockchain technology is rising, and much research focuses on solving several problems in telecommunications infrastructure, smart cities, and access control. Rebello *et al.* ensures security and transparency in the orchestration of virtual network functions (*Virtual Network Functions* - VNF) [7]. The authors use immutability and auditability properties of the blockchain to record operations in an environment with multiple market competing service providers. Camilo *et al.* propose a system that allows the safe commercialization of data based on a reputation and trust model [8]. The authors use blockchain to mitigate the problems of expensive service fees, single points of failure, and compromised privacy due to the centralization of data access control management.

Innovative proposals in permissioned blockchains and smart contracts use the Hyperledger Fabric and Hyperledger Sawtooth platforms [9]. Camilo *et al.* implement an architecture on the Fabric platform to store IoT data access permissions in the blockchain and provide data sharing through a smart contract [6]. The prototype achieves a transaction throughput of up to 70 transactions per second, higher than the current e-commerce systems in Brazil. Rebello *et al.* develop an architecture based on blockchain to ensure isolation between network slices and implement a prototype in Fabric [10]. The proposal stands out for using the platform's isolated channels to ensure privacy and for using smart contracts to automate the management and configuration of virtual network functions (VNF).

Performance evaluations of blockchains implementations help developers to choose the right platform for each use case and optimize the performance of smart contracts applications. Caro *et al.* present a blockchain use case for tracking the agricultural and food supply chain and compares the performance of Sawtooth and Ethereum implementations [11]. The results show more than seven hundred times less latency and more than six times less CPU consumption in Sawtooth, demonstrating the platform's performance potential in practical scenarios. Shi *et al.* Evaluate the performance of the Blockchain as a Service (BaaS) in the cloud using Sawtooth [12]. The authors find out that the platform is scalable by increasing the number of virtual machines and that the Sawtooth performance is affected according to the network conditions and resources of the cloud provider. The work, however, is restricted to an evaluation of Hyperledger Sawtooth in cloud environments and does not compare the platform with others available.

Androulaki *et al.* describe Hyperledger Fabric architecture in detail and highlight the logic behind design decisions [13]. The authors demonstrate platform functionality from a financial system use case and develop a smart contract that implements *Fabcoin* currency. Despite the extensive analysis, the work does not focus on a comparative analysis between platforms. Kuo *et. al.* compare ten popular blockchain platforms, emphasizing healthcare applications [14]. However, the authors consider Hyperledger as a single platform and do not delve into the important differences in architecture and performance between the tools. Moreover, they do not implement experimental scenarios to compare performance metrics. Muller *et. al.* compare Sawtooth and Fabric platforms and consider a fish supply chain use case [15]. Although the thesis presents general aspects of the platforms, the focus is on comparing the transaction processing flow. The authors do not implement the use case and do not compare the performance of the platforms.

This paper compares the architecture of the two pioneers and widely used platforms maintained by the Hyperledger project. We describe network architecture, smart contracts implementation, blockchain state storage, transaction processing model, consensus protocols, and permission control for each platform. The paper also implements smart contracts based on an access control use case and demonstrates the impact on performance due to transaction processing schemes and consensus protocol.

III. Hyperledger Sawtooth and Hyperledger Fabric Platforms

Hyperledger is a Linux Foundation project that aims to develop platforms, frameworks, tools, and open-source libraries for the diffusion of blockchain technologies with enterprise-quality levels. Hyperledger platforms are modular and guarantee flexibility in configuring consensus options, smart contracts, and access permission policies, thus serving many applications. The two main permissioned blockchain platforms maintained by the Hyperledger project are Hyperleger Sawtooth[1] and Hypeledger Fabric[2].

[1]https://www.hyperledger.org/use/sawtooth.

[2]https://www.hyperledger.org/use/fabric

Permissioned blockchains focus on business applications in which nodes identify themselves to participate in the network. These applications usually employ quorum-based consensus protocols, which have higher throughput in transactions per second compared to proof-based protocols from permissionless networks such as Bitcoin and Ethereum [16].

A. The Hyperledger Sawtooth Platform and the OX Model

The Sawtooth platform architecture is composed of validator nodes and clients, as shown in Figure 1. Validator nodes store a copy of the ledger, validate transactions and blocks, and communicate with the other validator nodes in a peer-to-peer network through a gossip protocol. On the other hand, clients communicate with validators through a Representational State Application Program Interface (REST API).

Sawtooth smart contracts are called transaction families, and they consist of two components: a transaction processor and a client application. The transaction processor receives transactions from clients and applies changes to the blockchain, following the rules defined by the smart contract. All validators must have the same set of transaction processors installed. The client application sends transactions to a validator in the format defined by the smart contract. The platform provides some transaction families for configuration and testing and a software development kit (SDK): in Go, JavaScript, Python, and Rust to develop new transaction families.

We store data concerning global state, which are frequently accessed, using a Merkle tree structure. This approach allows validator nodes to avoid extensive searches and provides an agile way of checking the integrity of the global state. A transaction may depend on changes applied to the global state by another transaction, called a predecessor transaction. We call these transactions conflicting transactions, as they must be applied in order.

Transaction processing follows the traditional order-execute (OX) model [15] which steps are described as follows:

1) **Sending**: clients propose one or more transactions and send them to the validator in a signed structure called a batch. The entire batch is discarded if a transaction is invalid. For all validators to process all transaction batches, one validator receives the batches from the others.
2) **Ordering**: validators ensure that conflicting transactions are applied in the proper order. The transactions sent by the clients contain the state addresses to be read and written, and the validators use this information to determine the predecessor transactions.
3) **Execution**: the corresponding transaction processors compute changes in global state. The transaction execution follows a serial model, in which all transactions are executed sequentially in the defined order, or parallel, in which non-conflicting transactions are executed concurrently. This execution order prevents transactions from being discarded due to a lack of information about changes applied by predecessors. However, in the case of many conflicting transactions in the same block,

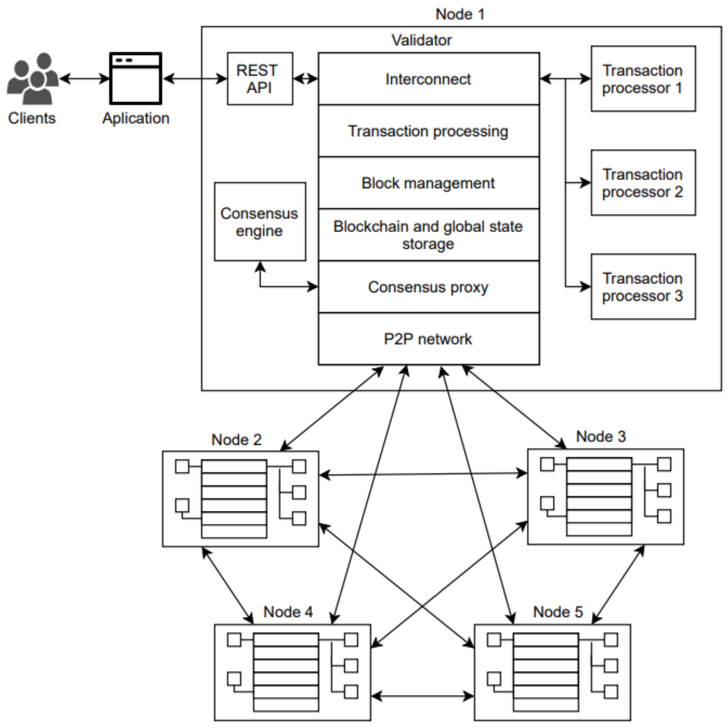


Fig. 1: Hyperledger Sawtooth architecture. Validator nodes process clients' transactions, store the blockchain and communicate in order to achieve consensus in a peer-to-peer network

the transaction throughput is considerably lower due to sequential execution.

4) **Confirmation**: a validator proposes and broadcasts a block according to the consensus protocol. The broadcasted block contains the identifiers (ID) of the transaction batches, and only valid transactions are applied to the blockchain. Other nodes add the new block when all transaction batches are executed.

The main consensus protocol for the Sawtooth is the Proof of Elapsed Time (PoET). The participant must prove that a random timer has finished proposing a block. This protocol requires the use of specific hardware with support for Intel Software Guard Extensions (SGX) technology so that malicious nodes do not gain control over the block generation process [17]. An alternative to the specific *hardware* in a development scenario is using the PoET simulator, which executes the exact same steps as the SGX version of PoET, but does not tolerate malicious participants. The platform also supports Practical Byzantine Fault Tolerant protocol (PBFT), a quorum-based protocol in which participants need to identify themselves and authenticate for the exchange of messages[3].

Validators and clients are identified by their public keys that are used to define which clients can submit transactions and which validators can connect to the network. Sawtooth can also support public network scenarios where all clients can submit transactions and all validators can participate on consensus.

[3]Sawtooth Raft consensus protocol is still under development according to the online repository available at: https://github.com/hyperledger/sawtooth-raft. Accessed on: 03/31/2021

B. The Hyperledger Fabric platform and the XO Model

Fabric architecture consists of clients, peer nodes, and ordering nodes as shown in Figure 2. Fabric identifies all participants as members of an organization. Clients interact with the blockchain through transactions sent to peers. Peer nodes are responsible for storing a copy of the blockchain and executing client transactions. The ordering nodes ensure consensus on transactions order and disseminate transaction blocks to the network [13]. Unlike Sawtooth, in which validator concentrates all transaction processing, Fabric separates roles between peers and orderers. Anchor peers are responsible for communicating with peers from other organizations through gossip protocol. Peers responsible for communicating with an ordering pair are called leaders. In Figure 2, nodes 1, 2, 3 and 5 are anchor peers and nodes 1, 2, 4 and 5 are leader peers.

Fabric Chaincodes are equivalent to Sawtooth transaction processors. Peers with a chaincode installed receive transaction proposals from clients and return a signed response called an endorsed transaction. In Fabric, not all peers need to install the same chaincodes and perform the same transactions. This procedure reduces the number of transactions executed by each peer. An endorsement policy defines the minimum number of peers who need to execute a transaction proposal to be added to the blockchain.

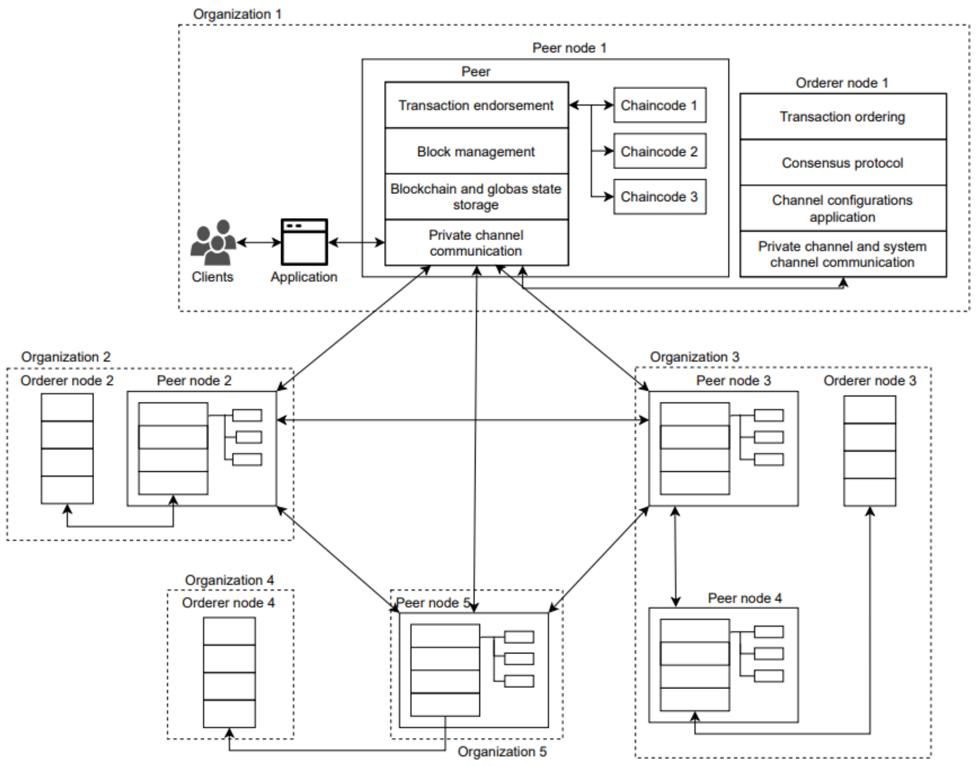


Fig. 2: Hyperledger Fabric architecture. Peer nodes clients execute transactions and can communicate with peers from another organization or with the ordering service. The ordering nodes guarantee the order of executed transactions in blocks and communicate through a system channel, omitted in the figure.

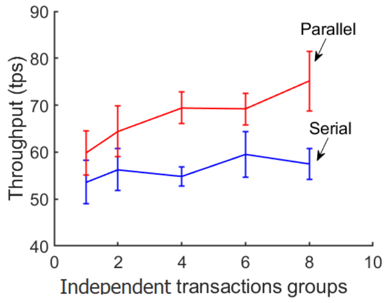
Fabric proposes a novel transaction processing model called execute-order (XO) in which peers execute all transactions in parallel, aiming to increase the transaction throughput [18]. The steps performed by the Fabric presented in sequence are:

1) **Sending**: clients send transaction proposals to the peers. Peers that receive transaction proposals must have a chaincode installed and are called endorsing peers.
2) **Execution**: the endorsing peers receive transaction proposals and compute the changes to be applied to the blockchain, according to the smart contract logic. Transactions are executed in parallel as they arrive in pairs. Therefore, endorsing peers return signed responses called endorsed transactions to the clients, containing the transaction results.
3) **Ordering**: clients send the endorsed transactions to the ordering nodes which are responsible for ordering transactions on a block candidate. The ordering nodes agree on a block by executing the consensus protocol.
4) **Confirmation**: the block is broadcasted to the leaders of each organization, which therefore forward it to the other peers. Peers validate transactions and apply changes to their copies of the ledger. Unlike Sawtooth, invalid transactions are recorded in the blockchain, ensuring auditability.

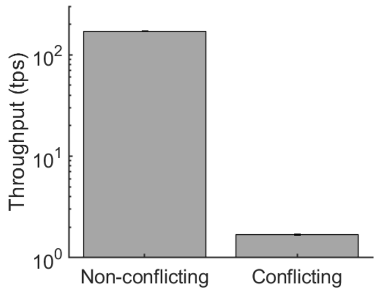
The global state stores information about the blockchain in a database using key-value pairs. Each transaction specifies a read set and a write set containing the data read and written from the global state. Unlike Sawtooth, which avoids conflicts by executing ordered transactions, Fabric follows a model based on executing all transactions concurrently and can generate conflicts. If a transaction is executed without its predecessors being applied, peers will read information from an outdated version of the global state. The pairs perform a Multi-Version Concurrency Control (MVCC) in the confirmation phase to prevent concurrent transactions inconsistencies . Transactions that read the current version of the global state are validated, and dependent transactions on the same block become invalid [19]. When a transaction is considered invalid, the client resubmits the transaction proposal to be processed and ordered again, reducing network performance, as participants waste computational resources processing invalid transactions.

The consensus protocol available on Fabric is Raft[4]. The Fabric protocol is quorum-based, in which participants exchange messages to vote for the next block. This type of protocol is deterministic and, therefore, does not suffer from forks, which occurs in proof-based protocols when two participants propose blocks simultaneously without being aware of each other. However, unlike PBFT, Raft only tolerates crash failures

[4]Fabric's Solo and Kafka protocols have been deprecated according to the online repository available at https://github.com/hyperledger/fabric/releases. Accessed on: 04/18/2021



(a) OX-Sawtooth serial execution presents a stable performance. Parallel execution improves the performance with the number of non-conflicting transactions.



(b) Low transaction throughput in XO-Fabric model caused by discarded conflicting transactions

Fig. 3: OX-Sawtooth and XO-Fabric execution models impact on transaction throughput caused by conflicting transactions.

and is vulnerable to malicious (Byzantines) actors. Crash Fault Tolerant protocols are much simpler than Byzantine Fault Tolerant protocols and achieve higher throughputs in permissioned networks.

Organizations use Certification Authorities (CA) to identify participants, and a Membership Service Provider (MSP) is defined in the network configuration to define the permissions of each identity. Members communicate through a private channel represented in Figure 2 by the arrows. The pair only makes available the blockchain information relative to the channel to the other participants. Multiple logically isolated channels can be implemented in the same Fabric network, allowing private communications between organizations that decide to form a consortium.

IV. Results

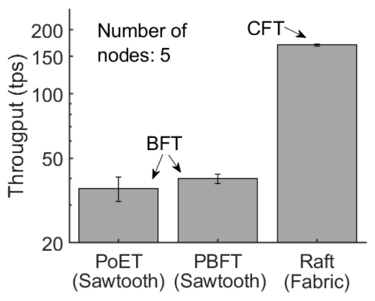
The performance evaluation aims to compare the throughput, represented by valid transactions per second, of Sawtooth and Fabric platforms in different practical scenarios, based on an access control use case. Camilo *et al.* propose the storage of access permissions using blockchain and the automation of the advertisement and purchase of data through a smart contract [6]. We implement two smart contracts in both platforms that define an advertisement transaction and a buy transaction as well as data to be read and written from the global state. When a client sends a buy transaction, the smart contract must read the balance of the negotiation participants and the advertisement information previously recorded and write the new balances and purchase information. We implement the prototypes on an Intel i9-10900 CPU 2.80GHz PC with 32GB RAM and 20 threads using Docker containers. We present results with a 95% confidence interval.

The first experiment evaluates Sawtooth throughput with serial and parallel transaction execution. A client sends 800 buy transactions to a validator node that executes a simplified random-leader consensus algorithm dedicated to development scenarios called dev mode. The maximum number of transactions per block is set to 1000 as used in other works [20]. The client first sends 1 group of 800 transactions, followed by 2 groups of 400 transactions, and so on. Transactions in the same group write to the same organization balance and therefore are conflicting, while transactions in separate groups are independent of each other. Figure 3a shows that parallel transaction execution achieves a throughput up to 30% higher than serial execution. Due to the OX model, when the number of transaction conflicts is high, the validator will execute dependent transactions sequentially, and parallel processing will not present a big advantage over serial processing.

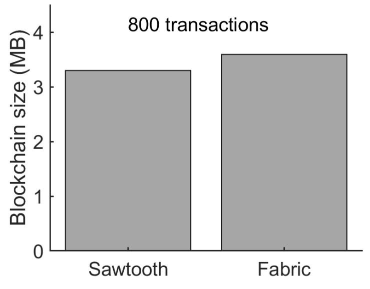
The second experiment evaluates Fabric XO model performance with conflicting and non-conflicting transactions. The Fabric network contains two peers and five orderer nodes executing the Raft consensus protocol. The number of transactions per block is set to 100 as used in other works [20]. Eight clients send a total of 800 transactions concurrently and measure the processing time. Figure 3b shows that Fabric validates one concurrent transaction in each of the ten proposed blocks and the other 99 get invalid due to MVCC, leading to a 100 times higher throughput with non-conflicting transactions. This represents a significant limitation of Fabric architecture, because applications that modify the same balance very frequently will present a poor performance. Fabric developers can improve the platform by providing the ability to choose between OX and XO models.

The third experiment compares the platform's consensus protocol in a permissioned scenario. Five nodes participate in the consensus, and Sawtooth executes non-conflicting transactions in parallel similarly to Fabric to achieve analogous scenarios in both platforms. Figure 4a shows that Fabric Raft achieves a four times higher throughput because it is much simpler than byzantine fault tolerant consensus protocols like PBFT and PoET. The PBFT presents a lower throughput than PoET because PBFT is designed to present a better performance than proof-base protocols in small permissioned networks. PoET results relative uncertainty is greater then the others because it is a probabilistic protocol where participants wait random times in each experiment round.

Figure 4b compares Sawtooth and Fabric blockchain sizes in MB after processing the same 800 transactions. These results reveal that the storage usage needed to store the ledger in Fabric is very similar to Sawtooth.



(a) Consensus protocol impact on Fabric and Sawtooth performance with five consensus participants.



(b) Sawtooth blockchain size is similar to Fabric.

Fig. 4: Consensus protocol impact on performance and blockchain size comparison.

V. Conclusion

This paper presented the main characteristics of Hyperledger Sawtooth and Fabric platforms for smart contracts development. We analyze the prototype performance based on a data-access commercialization use case, and we develop a smart contract for this purpose. Experiments reveal that Sawtooth achieves up to 30% higher throughput with parallel transaction execution if the number of transaction conflicts remains low. Therefore, Fabric Raft, despite being more insecure, presented a much higher performance compared to both Sawtooth consensus protocols in equivalent scenarios. Nevertheless, the XO model can lead to many invalid conflicting transactions and, therefore, a poor valid transaction throughput. In both cases, the developer determines how information is read and written into the blockchain and chooses the right consensus protocol for each scenario. Developers must reduce the frequency at which transaction conflicts happen and optimize application performance.

As future works, we pretend to extend the comparative analysis to other blockchain platforms.

VI. Acknowledgment

This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

References

[1] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Integration of blockchain and cloud of things: Architecture, applications and challenges," *Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2521–2549, 2020.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, disponível em https://bitcoin.org/bitcoin.pdf. Acessado em 18 de abril de 2021.

[3] Y. Dong and R. Boutaba, "Elasticoin: Low-volatility cryptocurrency with proofs of sequential work," in *ICBC*. IEEE, 2019, pp. 205–209.

[4] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which, and how," *Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3796–3838, 2019.

[5] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[6] G. F. Camilo *et al.*, "AutAvailChain: Automatic and Secure Data Availability through Blockchain," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[7] G. A. F. Rebello, I. D. Alvarenga, I. J. Sanz, and O. C. M. B. Duarte, "BSec-NFVO: A Blockchain-Based Security for Network Function Virtualization Orchestration," in *2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[8] G. F. Camilo *et al.*, "A Secure Personal-Data Trading System Based on Blockchain, Trust, and Reputation," in *IEEE Blockchain*, November 2020.

[9] L. A. C. de Souza *et al.*, "DFedForest: Decentralized Federated Forest," in *2020 IEEE Blockchain*, 2020, pp. 90–97.

[10] G. A. F. Rebello *et al.*, "Providing a sliced, secure, and isolated software infrastructure of virtual functions through blockchain technology," in *IEEE HPSR*, 2019, pp. 1–6.

[11] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda, "Blockchain-based traceability in agri-food supply chain management: A practical implementation," in *2018 IoT Vertical and Topical Summit on Agriculture-Tuscany (IOT Tuscany)*. IEEE, 2018, pp. 1–4.

[12] Z. Shi *et al.*, "Operating permissioned blockchain in clouds: A performance study of Hyperledger Sawtooth," in *ISPDC*. IEEE, 2019, pp. 50–57.

[13] Androulaki *et al.*, "Hyperledger Fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the 13th EuroSys Conference*, 2018, p. 30.

[14] T.-T. Kuo, H. Zavaleta Rojas, and L. Ohno-Machado, "Comparison of blockchain platforms: a systematic review and healthcare examples," *Journal of the American Medical Informatics Association*, vol. 26, no. 5, pp. 462–478, 2019.

[15] R. R. H. Muller, "Demystification of the blockchain phenomenon: A matter of state management," Master's thesis, Radboud University, 2019.

[16] G. A. F. Rebello, G. F. Camilo, L. Guimaraes, L. A. C. de Souza, and O. Duarte, "Security and performance analysis of quorum-based blockchain consensus protocols," Electrical Engineering Program, COPPE/UFRJ, Tech. Rep., 2020.

[17] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "SGAxe: How SGX fails in practice," 2020.

[18] C. Gorenflo, L. Golab, and S. Keshav, "XOX Fabric: A hybrid approach to blockchain transaction execution," in *2020 IEEE ICBC*. IEEE, 2020, pp. 1–9.

[19] J. A. Chacko, R. Mayer, and H.-A. Jacobsen, "Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric," *arXiv preprint arXiv:2103.04681*, 2021.

[20] A. Corso, "Performance Analysis of Proof-of-Elapsed-Time (PoET) Consensus in the Sawtooth Blockchain Framework," 2019.