Performance of Smart Contract-based Digital Twins for the Internet of Things

Chalima Dimitra Nassar Kyriakidou*, Iakovos Pittaras* Athanasia Maria Papathanasiou*,
George Xylomenos* and George C. Polyzos†
*Mobile Multimedia Laboratory
Department of Informatics, School of Information Sciences and Technology
Athens University of Economics and Business, Greece
{dnassar,pittaras,sissypapathanasiou,xgeorge}@aueb.gr
† School of Data Science
Chinese University of Hong Kong, Shenzehn, China
polyzos@acm.org

***Abstract*—Despite its rapid growth, the Internet of Things (IoT) still faces significant challenges related to interoperability, transparency and security. To address these issues, we propose the utilization of smart contract-based Digital Twins (DTs) "hosted" in the Hyperledger Fabric blockchain network, while leveraging the Web of Things paradigm for interoperability. Thus, our solution includes several notable features, such as decentralization, auditability and security. However, implementing DTs using Distributed Ledger Technologies (DLTs) introduces certain overheads. In this paper, we assess the feasibility and evaluate the performance of smart contract-based DTs using a set of Key Performance Indicators (KPIs). Our results demonstrate that, although DLT-induced overheads, such as latency, are present, they remain manageable for IoT use cases.**

***Index Terms*—Hyperledger Fabric, Kubernetes, Internet of Things, Digital Twins, Hyperledger Caliper**

I. Introduction

The *Internet of Things* (IoT) is envisioned to be an ecosystem of interconnected devices, merging the cyber with the physical world to provide a multitude of services in a variety of use cases, such as smart cities, healthcare, etc. However, there are still many challenges that need to be addressed. IoT systems remain fragmented, as IoT devices from different manufacturers use different protocols and standards. Additionally, regarding security, the solutions proposed are often incompatible with the resource-constrained nature of IoT devices. We argue that *Digital Twins* (DTs), with the right design, can address both these challenges, enhancing the transparency, interoperability, availability, auditability, and security of IoT systems.

DTs are typically deployed in secure, centralized environments, such as fixed Web servers, but are often tied to specific vendors, leading to issues of vendor lock-in and limited transparency. Furthermore, these centralized systems can be prone to occasional outages, affecting the reliability of DT operations. To address these challenges, we leverage *Distributed Ledger Technologies* (DLTs) and the emerging paradigm of the *Web of Things* (WoT) [1] to create and "host" secure DTs [2]. Initially, each IoT device exposes a *Thing Description* (TD) [3], which is used by the WoT gateway in order to create the DTs. Subsequently, the permissioned blockchain Hyperledger Fabric [4] is utilized to instantiate the DTs, which are deployed as smart contracts and "hosted" on the blockchain network. Users in our system interact directly with each DT rather than the actual device. Consequently, the DT relays all valid state modifications to the physical device, which carries out the required actions.

As shown in our previous work [2], this design has many appealing (security) properties. However, the overhead added by each DT, which acts as a proxy, should be acceptable for real-world use-cases. Therefore, in this paper, we aim to assess the feasibility and evaluate the performance of our smart contract-based DTs. In particular, the contributions of this paper are as follows:

- We utilize a number of high-level *Key Performance Indicators* (KPIs) for the smart contract-based DTs.
- We design a variety of experiments using the blockchain performance benchmarking tool, Hyperledger Caliper.[1]
- We evaluate the smart contract-based DTs based on the defined experiments and established KPIs.

II. Related Work

Several studies have explored the integration of blockchain with DTs, focusing on enhancing data storage and sharing capabilities. In particular, Yaqoob *et al* [5] and Khan *et al* [6] proposed utilizing blockchain as a secure storage solution for DT data, while Putz *et al* [7] and Dietz *et al* [8] introduced Decentralized Applications (DApps) to enable secure data sharing without trusted third-parties. These efforts emphasize the benefits of combining DLTs with DTs, but they mainly focus on data management.

Furthermore, there are several research efforts that evaluate the performance of Hyperledger Fabric. Specifically, Nasir *et al.* [9] conducted a performance analysis on a single Blockchain peer node by using a simple smart contract. Their study utilized a modified version of Caliper in order to assess KPIs such as execution time, latency, and throughput, focusing on different versions of Fabric (v0.6 and v1.0). The results

[1] https://hyperledger.github.io/caliper/

showed significant improvements in throughput and latency in the latest version; however, as the transaction load increased, performance deteriorated.

Another study by Baliga *et al.* [10] presented an experimental analysis of Hyperledger Fabric v1.0, demonstrating that throughput scales linearly for reads and almost linearly for writes for a transaction rate below 1000 TPS, after which performance degrades. The study highlights that tuning orderer settings and reducing endorsement requirements can improve throughput and latency, although this may raise security concerns in certain high trust scenarios.

Thakkar *et al.* [11] identified performance bottlenecks in Hyperledger Fabric v1.0 and proposed optimizations, such as aggressive caching and parallelized endorsement verification, which significantly improved performance. These optimizations were later incorporated into newer versions of Hyperledger Fabric, enhancing scalability and performance.

Additionally, Dreyer *et al.* [12] compared Fabric versions v0.6, v1.0, and v2.0 in terms of throughput, latency and error rate, showing that v2.0 offered notable improvements, particularly with setups involving multiple peers and organizations. Shalaby *et al.* [13] conducted experiments with HyperLedger Fabric v1.4 and showed that increasing the batch timeout leads to increased latency and decreased throughput, due to the time each block needs to wait for the timeout value, even in cases, where this block has already received all of the transactions. Increasing the batch size results in lower latency and higher throughput, which leads to better performance, as each block contains a greater number of transactions and thus more transactions will be simultaneously validated.

Our approach differentiates itself from existing solutions by addressing general IoT and WoT use cases, such as smart homes, cities and buildings, where interoperability, availability and security are essential. Instead of solely focusing on secure data storage and sharing for DTs, our design leverages WoT and the latest version of Hyperledger Fabric, namely v2.5.9, to create DTs that act as secure proxies, enabling transparent and auditable interactions, ensuring only authorized users can initiate state changes.

III. Evaluation Methodology

We evaluate two different versions of Hyperledger Fabric within the v2.x release series (v2.0.1 and v2.5.9). Notable differences between these two versions include the deprecation of gossip-based block dissemination and the Kafka consensus algorithm, which has been replaced by the more efficient Raft consensus mechanism. Additionally, v2.5.9 eliminates the need for a system channel, introduces multi-architecture binaries and docker images, and an improved smart contract lifecycle governance model. These new features are expected to improve the performance of Hyperledger Fabric.

The network topology, depicted in Fig. 1, consists of a WoT gateway, the blockchain network, and a Node.js client application. The Hyperledger Fabric blockchain network includes three organizations; one orderer organization with three orderer nodes (org0) and two peer organizations with two peer nodes each (org1 and org2). The whole blockchain network

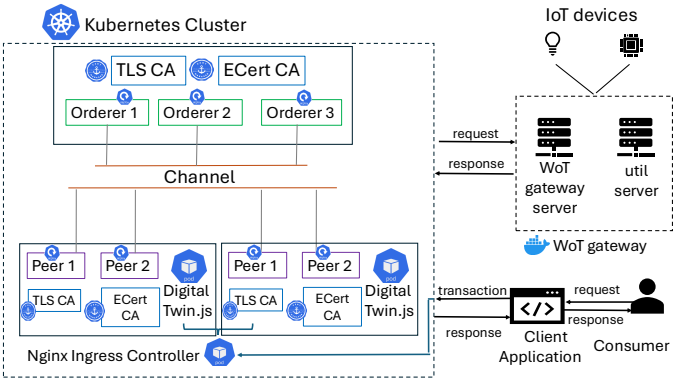

Fig. 1: Network Topology

is deployed in a Kubernetes cluster, which we chose due to its efficient resource management, scalability, and security properties, including resilience against DDoS attacks through traffic distribution and workload isolation. Finally, the WoT gateway runs in a Docker-compose setup, while the Node.js client application is responsible for submitting transactions and interacting with the network.

To evaluate the performance of our smart contract-based DTs in an IoT system, we utilize the following KPIs:

- Throughput: This metric indicates how many transactions (requests) the system can handle in a unit of time. We measure throughput as:
$$Throughput = \frac{N_{succ}}{t_{lc}\text{-}t_{fs}},$$
where N_{succ} represents the number of successful transactions, t_{lc} is the time when the last transaction was committed, and t_{fs} is the time when the first transaction was submitted.
- Latency: This metric measures the time it takes for an issued transaction to be completed and for a response to be made available to the application that initiated the transaction. Note that this takes into account the time required to communicate with and receive a response from the WoT gateway.
- Scalability: This metric assesses how well the system responds to an increase in the number of organizations and peers in the blockchain network, indicating the system's ability to maintain performance as the network grows.

The aforementioned KPIs are important for the evaluation of our implementation in IoT environments, where users often need to register dynamically, devices are constrained, and there is often a high volume of requests. Throughput, latency, and scalability allow us to understand how the system performs under realistic conditions and reflect the user experience in terms of responsiveness.

IV. Evaluation Results and Discussion

For the experiments, we deploy the network topology, presented in Section III, in a single physical machine in a local testbed. The machine is equipped with an Intel i7 processor, 8 CPUs, 16GB RAM, and runs Ubuntu 22.04. To optimize performance, we modify a few key settings across Hyperledger Fabric, Hyperledger Caliper, and Kubernetes. For

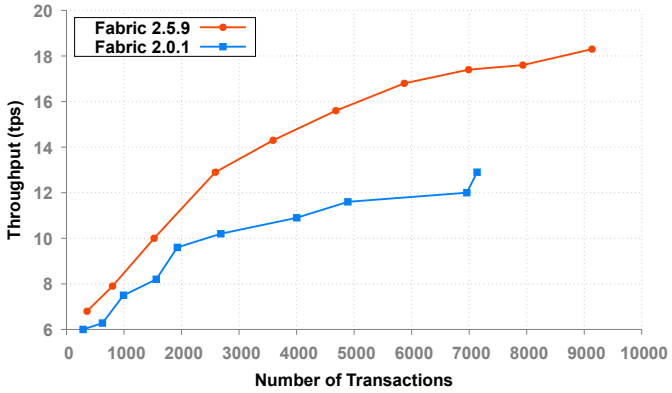

Fig. 2: Invoke Average Throughput with varying time-based test runs in Fabric v2.5.9 and Fabric v2.0.1

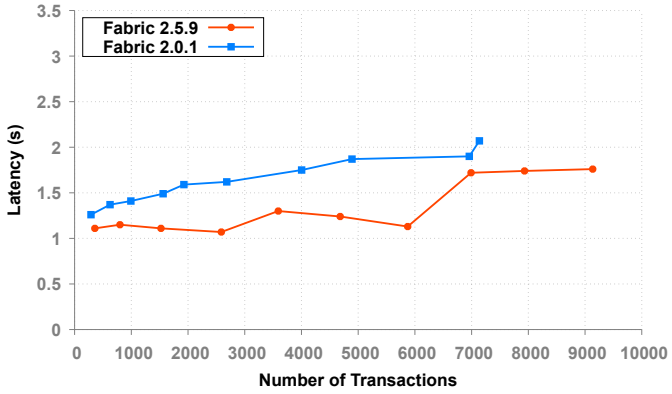

Fig. 3: Invoke Average Latency with varying time-based test runs in Fabric v2.5.9 and Fabric v2.0.1

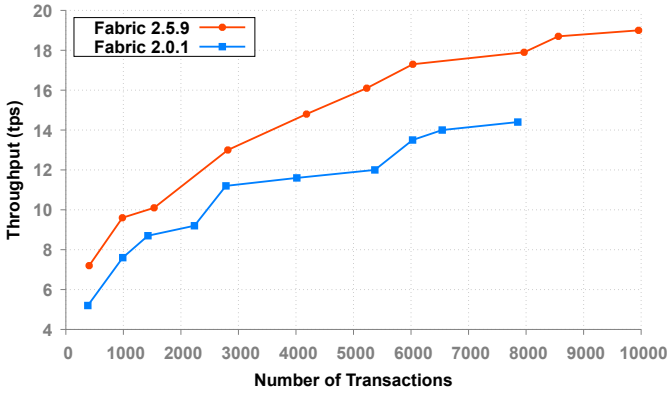

Fig. 4: Query Average Throughput with varying time-based test runs in Fabric v2.5.9 and Fabric v2.0.1

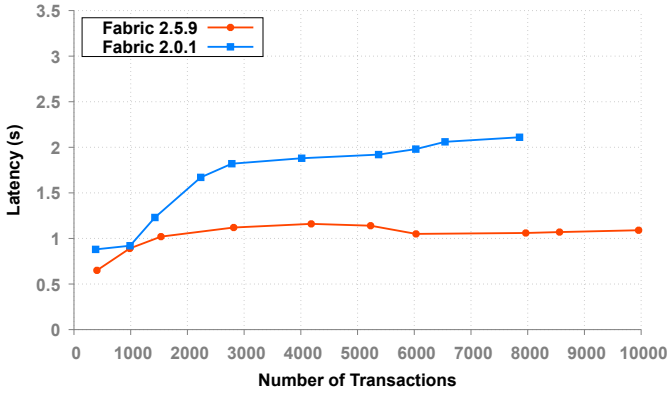

Fig. 5: Query Average Latency with varying time-based test runs in Fabric v2.5.9 and Fabric v2.0.1

instance, the cache size is increased from the default 64MB to 128MB, and the endorsement timeout, which specifies how long the gateway waits for a response from endorsing peers, is extended from 30 to 120 seconds. Additionally, we adjust the concurrency limit settings to increase the number of concurrently running requests to a service on each peer.

A. *Latency and Throughput*

In the first two sets of experiments we focus on measuring latency and throughput for both invoke (actuation) transactions that modify the ledger, and query (sensing) transactions that only read from the ledger, in Fabric v2.5.9 and v2.0.1. Each set of experiments consists of 10 test cycles with varying workloads. The workloads are defined by utilizing Caliper's txDuration parameter to perform time-based test runs. For each test cycle, we use an increasing number of workers, starting with 5, then progressing to 7, 9, 11, 13, 15, 17, 19, 21, and finally 25 workers in the last test cycle. The duration in the first test cycle is 50 seconds; it is increased by 50 seconds with each subsequent cycle. Therefore, the final test cycle, which uses 25 workers, lasts 500 seconds.

For invoke transactions, throughput and latency are depicted in Fig. 2 and Fig. 3, while for query transactions, the same metrics are depicted in Fig. 4 and Fig. 5. As expected, version 2.5.9 outperforms version 2.0.1 in terms of latency, throughput, and number of transactions processed, across all test cycles. For each version, as the duration increases across the test cycles, the number of transactions processed also increases, leading to an improvement in throughput and latency values that do not exceed 2 seconds in v2.5.9. This suggests that as the workload increases, the system maintains good performance. Query transactions exhibit better performance compared to invoke transactions, as they are read-only operations and involve less computational overhead. In both cases, we interact with the WoT gateway, which contributes to the higher-than-expected latency results.

B. *Batch Configuration*

In the next set of experiments, we evaluated the impact of both the Batch Timeout (BT) and batch size to our system's performance in v2.5.9 of Hyperledger Fabric; we did not test the older version, since the results above show that it clearly performs worse. Notably, when only changing the values for the BT, while the batch size remained at its default value, we observed an increase in latency and a decrease in throughput leading to an overall decline in the system's performance. In contrast, when only the batch size was changed, while keeping the BT at its default value, we observed an increase

in the number of transactions, as we increased the txDuration, leading to lower latency and higher throughput. This means that the system's performance improved, as also observed in the paper by Shalaby *et al.* [13].

Batch size in Hyperledger Fabric is configured using three parameters:

- Max Message Count (MMC), which sets the maximum number of messages (transactions) allowed in a batch,
- Absolute Max Bytes (AMB), which sets a hard limit on the total size of serialized messages in a batch, and
- Preferred Max Bytes (PMB), which indicates the preferred maximum size of serialized messages, acting as a guideline for the best effort batch size.

A batch will continue to fill with messages until it reaches one of the following three conditions: the PMB limit is reached, the MMC is met, or the BT is exceeded. If a new message exceeds the PMB limit, the current batch is closed without including the aforementioned message, it is written to a block, and a new batch is started for the incoming message. Although it is rarer, it is possible for messages to be larger than the PMB, and thus exceed this limit. These batches include exactly one transaction or message, which has to be up to the AMB limit.

TABLE I: Batch Configuration experiment results

BT, MMC, AMB, PMB	Latency	Throughput	Tx Number
2s, 500, 10MB, 2MB	0.78	10.3	1045
5s, 600, 12MB, 4MB	1.53	4.5	464
8s, 700, 14MB, 6MB	1.07	7.1	721
10s, 800, 16MB, 8MB	0.98	7.9	806
12s, 900, 18MB, 10MB	0.87	9.0	915

For this set of experiments we set the duration to 100 seconds and utilized 5 worker threads, which process transactions in parallel but independently of each other; the results are depicted in Tab. I. Row 1 shows the results of the first test-cycle that used the default values for the batch configuration parameters. From a high level perspective, as we progressively increase the values of the batch configuration parameters we can make the following observations:

- Latency increases until row 2, before decreasing,
- Throughput demonstrates a decline up to row 2, and after row 3 we observe an incline, although the values remain lower than the first test-cycle.

While the aforementioned observations appear to differ from the results presented by Shalaby *et al.* [13], we believe the differences can be explained by the number of transactions being equal to the MMC in the rows where we observe an incline in performance. Specifically, in row 2, we observe that within the defined timeframe (txDuration) the number of transactions was a lot less than the MMC, and thus we can conclude that in this case the BT impacts performance more than the batch size leading to an overall worse performance. In rows 3,4 and 5, which is where we observe a pattern of improvement in both throughput and latency, within the specified timeframe the system manages to process a number of transactions that is almost equal to the MMC. In this case, we conclude that the improved performance can be attributed to the batch size and not the values that we set for the BT. This conclusion is supported by the fact that each transaction is approximately 3-4 KB in size, considering that one transaction requires 1-2 KB plus an additional 1 KB per endorsement. Given that our endorsement policy requires an endorsement from the majority of the organizations, so in this case two endorsements (one per organization), this means that each transaction reaches around 3-4 KB. Therefore, more transactions are needed to reach the PMB or the AMB, making the MMC the limiting factor. Notably, although both latency and throughput are improved in the last three test cycles, the results are still not better than in the first test cycle (row 1), where the MMC is roughly half the number of transactions. Thus, we can add to our conclusion that the system tends to perform well when the number of transactions processed is approximately a multiple of the MMC.

C. Scalability

To evaluate the scalability of our system, we conducted experiments with a txDuration of 100 seconds and 5 workers across three test cycles. The first cycle involved one orderer organization having three orderers and one peer organization comprising two peers. For the second cycle, we added a second peer organization, also with two peers, while in the third cycle, we included a third peer organization and one additional orderer to the orderer organization. The results of these experiments are depicted in Fig. 6 and Fig. 7.

Increasing the number of peers in each organization generally improves network performance. This is because transaction endorsement requests are distributed across the peers, which means that with more peers, we expect more efficient load balancing. Additionally, the number of ordering service nodes directly impacts performance, as all orderers participate in the consensus process. A minimum of three orderers is required for network consensus. Adding a fourth orderer provides crash fault tolerance for one orderer failure, while five orderers protect against two failures. However, increasing to six orderers may reduce overall network performance due to added overhead.

The results of our experiments confirm these expectations. Specifically, in the first test cycle, with a single peer organization, throughput was good and latency was acceptable. In the second cycle, with two peer organizations, performance improved slightly. Finally, in the third cycle, with three peer organizations and an additional orderer, we observe a significant improvement in performance, particularly in latency, potentially nearing the system's performance peak.

D. Transaction Delay

In our system design, communication with the WoT gateway is essential for actuation because the device's state must change in the physical world. However, in the case of sensing, we could avoid communicating with the WoT gateway and simply query the blockchain for the most recent state of the device, if it were to reduce latency and improve throughput. To confirm our expectations, we conducted experiments to measure the difference in performance when communicating with

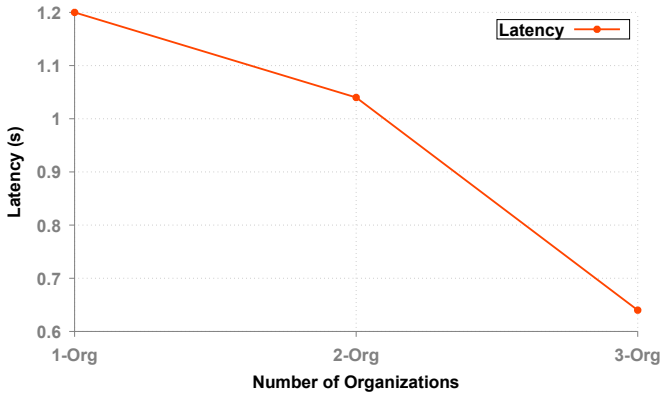

Fig. 6: Average Latency while increasing the number of Organizations

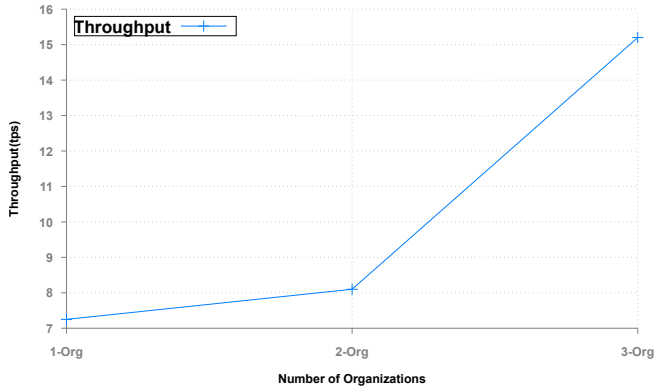

Fig. 7: Average Throughput while increasing the number of Organizations

the gateway and when only interacting with the blockchain for query transactions.

In a test cycle with 100 seconds duration and 7 worker threads, we found that when interacting with the gateway, the latency was 0.89 seconds, throughput was 9.6, and the system processed 983 transactions. However, when we avoided gateway communication during sensing, the latency dropped to 0.01 seconds, throughput increased to 484.0, and the system processed 47,729 transactions. In a second test cycle with 250 seconds duration and 13 worker threads, communication with the gateway resulted in a latency of 1.16 seconds, throughput of 14.8, and 4,183 transactions processed. Without gateway communication, latency improved to 0.03 seconds, throughput reached 509.7, and the system processed 126,112 transactions. Finally, in the last test cycle with a duration of 500 seconds and 25 worker threads, querying with the gateway led to a latency of 1.09 seconds, throughput was 19.0, and the system processed 9,955 transactions. In contrast, without gateway communication, the latency dropped to 0.06 seconds, throughput increased to 512.6, and the system processed 253,797 transactions. These results indicate a significant overhead when communicating with the gateway during sensing, with latency overhead ranging from 0.88 to 1.13 seconds, throughput overhead from 474.4 to 494.9 TPS, and transactions processed overhead between 46,746 and 243,842 transactions.

To optimize our design, we could initialize each IoT device with its current state when it is first registered on the blockchain. This way, the system would always have a state to return when a sensing (query) operation is performed, without needing to communicate with the gateway. When a device's state changes due to an actuation (invoke) operation, this update will be reflected in the blockchain. Since a query only reads the blockchain, it is not only unnecessary to contact the gateway during sensing; the performance gains from avoiding this communication are also substantial.

V. Conclusion and Future Work

Our experiments demonstrated that the transition from Hyperledger Fabric v2.0.1 to v2.5.9 introduces significant enhancements in performance, scalability, and throughput. We observed that increasing the transaction duration, which allows the system to process more transactions, may occasionally result in a slight increase in latency, but it leads to a notable improvement in throughput, indicating the system's ability to maintain good performance under higher workloads. Moreover, when conducting experiments by changing the values of the parameters related to the batch configuration, we found that the max number of messages in a batch is correlated to the number of transactions within a given timeframe. Additionally, the results of our experiments demonstrate that expanding the number of peers and orderers improves load balancing and fault tolerance, respectively. These findings align with our expectations.

We plan to leverage the Kubernetes cluster used in this paper for a more complex deployment involving virtual machines in a 6G environment, made available to us by the 6G-SANDBOX project[2], where we expect even better performance results. This future work will provide valuable insights into the potential of such infrastructures for supporting next-generation blockchain applications under real-world conditions.

Acknowledgment

The work reported in this paper was partly funded by the EU H2020 Programme through the subgrant Smart Contract-based Digital Twins for the IoT (SCDT, 6GSANDBOX-OC2) of project 6G-SANDBOX (grant agreement No 101096328).

References

[1] W3C, "Web of Things," 2017, (Accessed Dec. 22). [Online]. Available: https://www.w3.org/WoT/

[2] I. Pittaras, N. Fotiou, C. Karapapas, V. A. Siris, and G. C. Polyzos, "Secure smart contract-based digital twins for the internet of things," *Blockchain: Research and Applications*, vol. 5, no. 1, p. 100168, 2024.

[3] S. Kaebish, T. kamiya, M. McCool, V. Charpenay, and M. Kovatsch, "Web of Things Thing Description," 2020, (Accessed Oct. 24). [Online]. Available: https://www.w3.org/TR/wot-thing-description/

[4] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the EuroSys Conference*. Association for Computing Machinery, 2018.

[2]https://6g-sandbox.eu/

[5] I. Yaqoob, K. Salah, M. Uddin, R. Jayaraman, M. Omar, and M. Imran, "Blockchain for digital twins: Recent advances and future research challenges," *IEEE Network*, vol. 34, no. 5, pp. 290–298, 2020.

[6] A. Khan, F. Shahid, C. Maple, A. Ahmad, and G. Jeon, "Toward smart manufacturing using spiral digital twin framework and twinchain," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, pp. 1359–1366, 2020.

[7] B. Putz, M. Dietz, P. Empl, and G. Pernul, "Ethertwin: Blockchain-based secure digital twin information management," *Information Processing & Management*, vol. 58, no. 1, p. 102425, 2021.

[8] M. Dietz, B. Putz, and G. Pernul, "A distributed ledger approach to digital twin secure data sharing," in *Proceedings of the IFIP Data and Applications Security and Privacy Conference (DBSEC)*. Springer, 2019, pp. 281–300.

[9] Q. Nasir, I. A. Qasse, M. Abu Talib, and A. B. Nassif, "Performance analysis of hyperledger fabric platforms," *Security and Communication Networks*, vol. 2018, no. 1, p. 3976093, 2018.

[10] A. Baliga, N. Solanki, S. Verekar, A. Pednekar, P. Kamat, and S. Chatterjee, "Performance characterization of hyperledger fabric," in *Proceedings of the Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, pp. 65–74.

[11] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *Procedings of the IEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2018, pp. 264–276.

[12] J. Dreyer, M. Fischer, and R. Tönjes, "Performance analysis of hyperledger fabric 2.0 blockchain platform," in *Proceedings of the workshop on cloud continuum services for smart IoT systems*, 2020, pp. 32–38.

[13] S. Shalaby, A. A. Abdellatif, A. Al-Ali, A. Mohamed, A. Erbad, and M. Guizani, "Performance evaluation of hyperledger fabric," in *Proceedings of the International Conference on Informatics, IoT, and enabling Technologies (ICIoT)*. IEEE, 2020, pp. 608–613.