Latency Assessment of Blockchain-based SSI Applications Utilizing Hyperledger Indy

Hamza Baniata[1][a], Tamas Pflanzner[1][b], Zoltan Feher[1] and Attila Kertesz[1][c]
[1] *Department of Software Engineering, University of Szeged, Dugonics tér 13, 6720, Szeged, Hungary*

Keywords: Blockchain, Self Sovereign Identity, Hyperledger Indy, Latency Analysis.

Abstract: Blockchain is the core technology behind several revolutionary applications that require consistent and immutable Distributed Ledgers, maintained by multi-party authorities. Examples of such applications include cryptocurrencies, smart contracts, Self Sovereign Identity (SSI) and Edge/Fog-enabled smart systems (eHealth, IIoT, IoV, etc.). Hyperledger Indy and Aries are suitable open-source tools for permissioned blockchain utilization in SSI projects. Those two frameworks have gained much attraction by researchers interested in the topic, while continuously maintained under the umbrella of the Linux Foundation. However, some SSI applications require specific upper bound of response time depending on their business model. In this paper we aim at presenting a detailed latency analysis of Indy, on top of which Aries is typically built. With such an architecture, researchers and practitioners of SSI applications can decide whether this framework fulfills their application requirements. To realize our proposed architecture, we have developed a Python application with containerized Indy and Aries components. Its scripts use and build on the official open-source codes of the Hyperledger repositories. We have deployed our application on multiple virtual machines in the Google Cloud Platform, and evaluated it with various scenarios. Depending on the transaction rate, we found that the writing response latency of an Indy-based blockchain containing 4 and 8 nodes, ranges between 1–16 seconds, while the reading response latency with similar settings ranges between 0.01–5 seconds.

1 INTRODUCTION

Blockchain (BC) (Nofer et al., 2017) is a technology utilized for enhancing trust in Distributed Computing applications and managing Distributed Ledgers (DL). Applications integrated with permissionless BCs benefit from high levels of security and trust, where BCs provide fully-immutable log of transaction (TX) history without the control of a central authority. Similarly, permissioned BCs utilize a Distributed Trusted Third Party for providing services typically requested from a Central Trusted Third Party. Permissioned BCs are used in applications that require trusted BC nodes, typically referred to as Miners, Minters, Verifiers or Validaters, to maintain a trusted and consistent DL. Initially, those participants apply for becoming BC nodes to a committee that votes for inclusion. Permissionless BC solutions have been designed first for realizing digital cryptocurrencies (Abraham et al., 2016), but nowadays they address a wide variety of environments, such as document management (Karasek-Wojciechowicz, 2021), smart applications for eHealth (Transaction and MPI, 2016) or Internet of Vehicles (Javaid et al., 2020). In such access model, any party can join the BC network and collaborate in validating and confirming new pieces of data added to the DL, without a limit of network size. Permissioned BC solutions, on the other hand, are suitable for applications that require distributed yet controlled Trusted Third Party, where a limited number of verifiers vote for appending data into the DL and for accepting new verifiers. Examples of applications that require such BC model include distributed voting (Bistarelli et al., 2019), Self Sovereign Identity (SSI) (Kondova and Erbguth, 2020), and credential management (Jirgensons and Kapenieks, 2018).

Specifically, classical internet applications lack an identity layer (Tobin and Reed, 2016). Web applications, for instance, use usernames and passwords to identify their clients, yet the use of these attributes is typically limited to the specific contexts of individual

[a] https://orcid.org/0000-0003-1978-3175
[b] https://orcid.org/0000-0002-3196-9100
[c] https://orcid.org/0000-0002-9457-2928

Baniata, H., Pflanzner, T., Feher, Z. and Kertesz, A.
Latency Assessment of Blockchain-based SSI Applications Utilizing Hyperledger Indy.
DOI: 10.5220/0011082300003200
In *Proceedings of the 12th International Conference on Cloud Computing and Services Science (CLOSER 2022)*, pages 264-271
ISBN: 978-989-758-570-8; ISSN: 2184-5042
Copyright © 2022 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

applications. Such a framework is considered inconvenient and implies several security issues (e.g. identity theft and fake users) (Thomas et al., 2019). Global identity (Makri et al., 2021) providers, like Google or Meta, attempt to develop their access framework to enhance its usability. However, data leaks are still argued to remain as an open-issue (Boysen, 2021). Additionally, the loss of clients' control over their private data, which is typically saved on centralized servers, implies trust issues. Some solutions allow users to request the deletion of their data, in order to comply with the European General Data Protection Regulations (GDPR), yet users have to also trust the provider to delete them. SSI concepts enable users to anonymously identify themselves, and verify some private attributes about them, while maintaining full control and permissions over their private data.

Although permissioned BCs have shown unprecedented abilities to maintain DLs with high security and trust measures, compared to non-BC based DLs, they still suffer from high latency compared to centralized solutions. Furthermore, different permissioned BC solutions are designed differently according to their application and the different layering of the solution. This usually causes a fluctuation of latency measures among different permissioned BC solutions and even for differently parameterized solution. Such issue mainly appears due to several system properties and limitations, such as the minimum time needed to reach a consensus on a piece of data (i.e. Finality time), the average transmission delay between network entities, the total network size and the average number of neighbors per miner (Kertesz and Baniata, 2021). This being said, researchers and practitioners tend to model their applications and test several BC solutions for a decision to be made regarding the deployment of a specific BC framework. As many solutions are being continuously proposed, it must be a burden to utilize and test several frameworks before assuring the application requirements are fulfilled by a selected BC framework.

As several recent works have utilized Hyperledger Indy (Linux Foundation, 2020), and several projects are currently investigating its deployment for their SSI applications, the aim of this work is to analyse Hyperledger Indy in terms of latency. To approach our objective, we propose a deployment architecture using Indy and Aries suitable for SSI applications. We have developed a Python application with containerized Indy and Aries components, building on the official open-source Hyperledger projects. We have deployed our application onto multiple virtual machines in the Google Cloud Platform, and performed a detailed evaluation with different scenarios varying the number of BC nodes and TX arrival rates.

The remainder of this paper is as organized as follows: Section 2 provides a brief architectural and technical background of Hyperledger Indy. Section 3 presents the research methods and design decisions. Section 4 presents the results we obtained by running our experiments, which are discussed in Section 5. Section 6 discusses recent related works and, finally, Section 7 concludes the paper.

2 BACKGROUND ON HYPERLEDGER INDY

Hyperledger Indy (Linux Foundation, 2020) is an open source project, administered by the Linux foundation, which aims at supporting Verifiable Credential (VC) systems based on public-permissioned BC approach. The project implementation provides a platform for Decentralized Identifiers (DIDs) rooted on BCs, or other DLs, so that they are inter-operable across administrative domains, applications, etc. and usable for validating VCs. The project deploys privacy preserving mechanisms such as the Zero Knowledge Proofs and Digital Signatures. Additionally, Indy takes good care of what is saved on-chain, aiming at the adherence with the GDPR. The Plenum Consensus Algorithm is used in Indy, which is a special purpose Redundant Byzantine Fault Tolerance Consensus Algorithm (Aublin et al., 2013). A live example of an Indy based solution is the Sovrin BC (Windley, 2016) providing a general purpose, global DID-supportive platform.

As depicted in Figure 1, Indy requires a small group of miners that serves as a Distributed Trusted Third Party. Indy miners, or validators, are computers administered by publicly known parties, added by a voting scheme between miners themselves. The miners main purpose is to maintain the liveness and consistency of the DL by accepting new TXs. Several types of BCs consist of different types of TXs that need to be saved on-chain. Three of those BCs are depicted in the figure which are the Domain TXs BC, the Pool TXs BC and the Config TXs BC. On these BCs, public data about admins of miners, TXs meta data and network configurations are saved, respectively.

Indy miners are added to the system in a decentralized fashion, if they fulfil certain conditions. Indy allows that by a voting scheme, where available miners accept or decline a new miner application. A steward miner is an authorized Indy node that is needed to be able to add new nodes to the network. Only one node can be added per steward, so an equal number of stewards and miners need to be present. The avail-

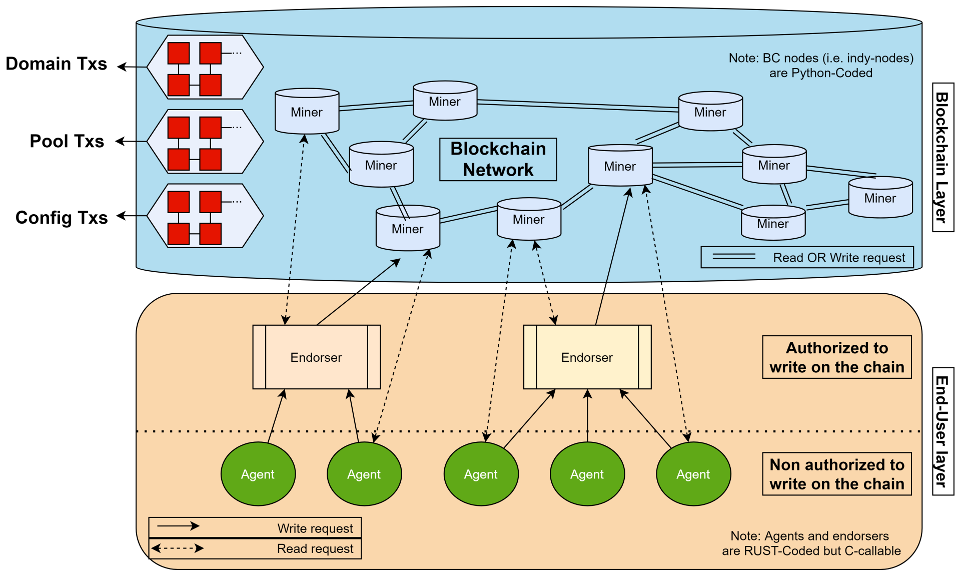
Domain Txs
Pool Txs
Config Txs
Note: BC nodes (i.e. indy-nodes) are Python-Coded
Blockchain Layer
Miner
Blockchain Network
Read OR Write request
Endorser
Authorized to write on the chain
Non authorized to write on the chain
End-User layer
Agent
Write request
Read request
Note: Agents and endorsers are RUST-Coded but C-callable

Figure 1: Our proposed Blockchain architecture based on Indy.

able sample code at the official Indy repository creates four steward nodes. To add the first four nodes of the network, four wallets and four pairs of PKI keys need to be created. However, a genesis block is created along with the initial stewards DIDs at the initialization phase of the system. Here, the stewards data are generated by seed defined by the developer. Then, NYM requests are sent by the wallet admins to the network of stewards who accept/reject the TXs. Once accepted, a new wallet is created and a block is added to the BC. Once a minimum of four miners are active, the BC network can accept new DID and schema TXs from end-users. Endorsers and agents are end-user entities of the system that should be implemented using the Aries scripts. The only main difference between these two types of end-users is that agents are only authorized to read on-chain data while endorsers are authorized to both read and write.

3 THE PROPOSED ARCHITECTURE AND METHODS

To realize the deployment of Indy and Aries nodes, we have designed and developed a validation architecture as depicted in Figure 1. We implemented an Aries agent that is capable of connecting to an Indy network, and of generating sample data (i.e. TXs), to be submitted to it. We extracted the initial codes from the Indy-SDK repository, which is implemented using Python. Our agent is capable of sending up to 250 TX requests per second, using a multi-threaded approach, to examine the system's read and write processing speeds. The sample TXs are constructed similarly to the standard TX formats generated by an Aries agent. However, the codes available at the Aries official repository allows for only 20 TXs/second, thus we had to implement our own agent.

We have implemented Indy node scripts that are suitable to be run individually on different machines. That is, the available modules we found, at the official Indy repository, run four nodes on a single machine and they were not ready for production. As a demonstration of the network is needed to control different nodes by different admins, we have also utilized the Admin UI from the VON network repository, which we configured to connect to the created Indy network.

All our implemented application components are containarized using Docker, which makes them easier to deploy for present, and future works. Our implemented application is publicly available at Github[1], and its main management scripts are as follows:

- indy_config.py: This file contains the configuration to run the Indy-node, i.e. the name of the network and the installation parameters.
- requirements.txt: This file contains system dependencies. We used Flask to create a Web-Service, which displays the genesis file extracted from the container, according to which the admin application could connect.
- indy-node-start.sh: This script starts the server.py in a container to create a Blockchain node.

[1]https://github.com/sed-szeged/IndyPerf

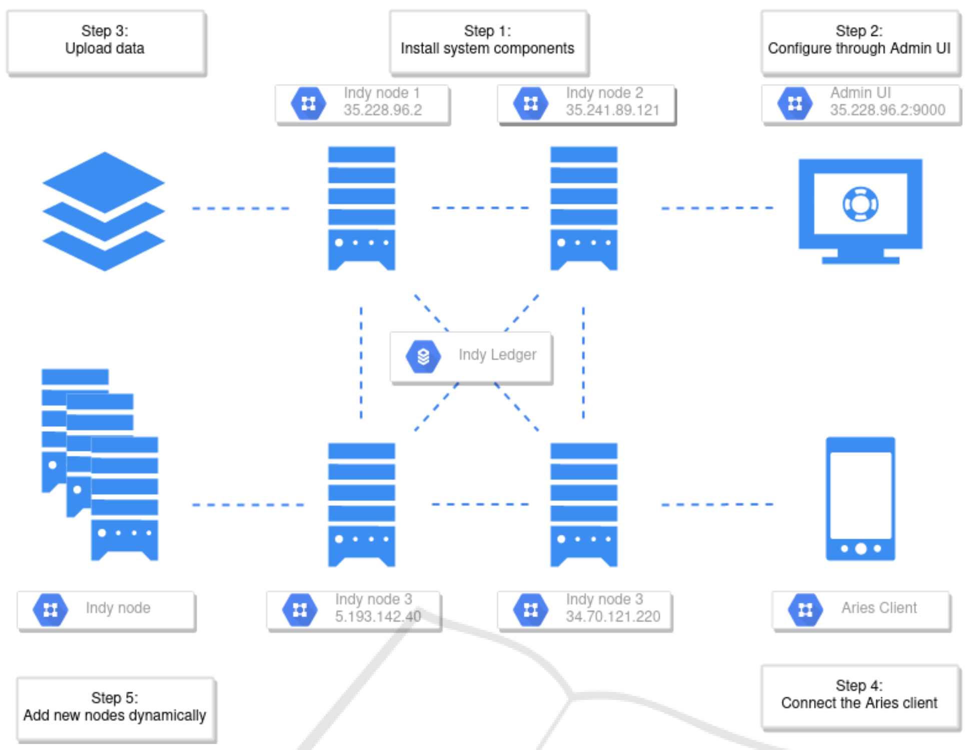

Figure 2: Evaluation architecture and installation steps.

- create_steward.py: This script can be used to add new stewards, thus new nodes to the system.
- test_transactions.py: This script can be used to perform the experiments (to be detailed in the next section). It creates connection with the BC network, and sends the specified TXs in multiple threads.

We performed several scenario runs using our multi-threaded Aries agent with different TX arrival rates (1–250 TX/s) and different network sizes (4–8). Each scenario run is repeated 10 times, which gives a total BC height of 10–2500 blocks. This way, an average latency for different heights of the BC, with different network sizes, can be extracted. We measured the elapsed time for a TX to be sent, processed and responded to. We have collected the minimum and maximum latency measures, and computed their average values. We separately examined the read and write speeds to guarantee the accuracy of our measurements. All nodes were deployed and run individually on different virtual machines, located in different regions of the Google Cloud Platform. We used E2-medium VMs (up to 3.8 GHz, 2 vCPUs, 4 GB memory) running Ubuntu 16.04 OS. Visualizations of the evaluation architecture and installation steps are provided in Figure 2.

4 EVALUATION RESULTS

In this section, we present the latency measures we obtained by deploying our scripts on multi-regional cloud servers.

4.1 Four-node Measurements

Table 1 provides the detailed read and write latency measures. We have computed those measures for an Indy network that consists of four nodes. We found that the size of data stored on-chain does not affect the speed as the ten measures we have collected per scenario were very close to each other. However, we can see a proportional relation between the TX arrival rate and the average latency for both read and write TXs. It is also worth noting that the average write latency fluctuates from 2.7 to 6.3 seconds per TX, depending on the TX arrival rate. On the other hand, the average read latency fluctuates from 0.088 to 1.56 seconds per TX

4.2 Eight-node Measurements

Table 2 provides the detailed read and write latency measures. We have computed those measures for an

Table 1: Latency measures, in seconds, for read (R) and write (W) TXs where network size = 4.

TX type	Arrival rate (TXs/s)					
	1	***50***	***100***	***150***	***200***	***250***
Min (W)	1,1014	1,1014	1,1014	1,1014	1,1014	1,1014
Max (W)	2,9832	2,9138	2,9490	5,8131	6,8542	13,3811
Avg (W)	2,7388	2,5284	2,2197	4,4763	5,3511	6,3534

TX type	Arrival rate (TXs/s)					
	1	***50***	***100***	***150***	***200***	***250***
Min (R)	0,0108	0,0671	0,1058	0,0841	0,2469	0,0800
Max (R)	0,2160	1,1387	1,4655	2,2378	2,5302	4,7086
Avg (R)	0,0881	0,3961	0,5815	0,8964	1,1408	1,5562

Table 2: Latency measures, in seconds, for read (R) and write (W) TXs where network size = 8.

TX type	Arrival rate (TXs/s)					
	1	***50***	***100***	***150***	***200***	***250***
Min (W)	0,2614	1,7064	1,5032	1,3114	1,2914	1,5289
Max (W)	2,9949	2,9607	5,2011	8,5050	7,4365	16,9094
Avg (W)	2,6118	2,5993	2,5347	4,8506	5,5883	10,4323

TX type	Arrival rate (TXs/s)					
	1	***50***	***100***	***150***	***200***	***250***
Min (R)	0,0092	0,0567	0,0927	0,0667	0,0350	0,0219
Max (R)	0,7147	1,1744	1,4469	1,9189	2,4340	5,6863
Avg (R)	0,1203	0,3718	0,5430	0,8163	1,0652	2,4928

Indy network that consists of eight nodes. Here, we have also found that the size of data stored on-chain does not affect the latency. We can also see that increasing the arrival rate increases the average latency for both types of requests. It is observable in the table that the average write latency fluctuates from 2.6 to 10.4 seconds per TX depending on the arrival rate. On the other hand, the average read latency fluctuates from 0.12 to 2.49 seconds per TX.

5 DISCUSSION

As we can observe in the results presented in the previous section, increasing the number of Indy nodes and/or the arrival rate increase the average response latency. These measures are important to consider when Indy is considered as a framework for a given BC-based application. The scenarios we tested, and their results, shall help those who tend to utilize Indy and Aries for a decision to whether adopt or exclude this platform from their project. To further clarify the results we obtained, we present our results in Figures 3 and 4.

In the case of read TXs, we can observe that the average latency measurement is almost linearly proportional to the arrival rate up to 200 TXs/s. After that, the latency starts to increase according to the number of nodes deployed. In the case of write TXs, we can observe that the average latency measurement is almost constant up to an arrival rate of 100 TXs/s. After that, the latency starts to increase depending on the system buffer size and the utilized processing power.

Furthermore, one can see that the average write latency is generally higher when deploying more nodes for different arrival rates. The read latency, on the other hand, is generally lower when deploying more nodes as more servers are available to process the received TXs. A utilization example of these results may be a latency-sensitive application that requires an average write latency that is less than 2 seconds per TX shall not adopt the Indy framework. Note that this measurement is for the least number of Indy nodes (i.e. 4 nodes). Adding more nodes shall increase the write latency as discussed above.

During the implementation and deployment of Indy and Aries nodes, we faced the following drawbacks/challenges. DLs in Indy are maintained linearly, which implies that high DL consistency depends on high block finality time (Baniata et al., 2022). This causes additional write latency on top of the latency required for reaching a consensus. Providing global information regarding what accreditation bodies approve/disapprove, and on which reference criteria blocks are confirmed, cannot be done us-

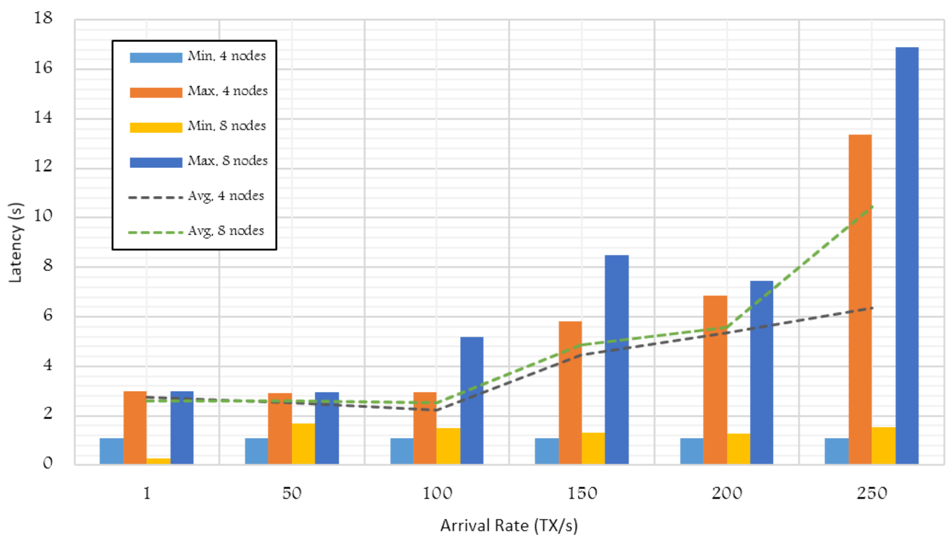

Figure 3: Latency of write TXs (in seconds) measured with different Indy-based Blockchain network sizes (4 and 8) and different arrival rates (1–250).

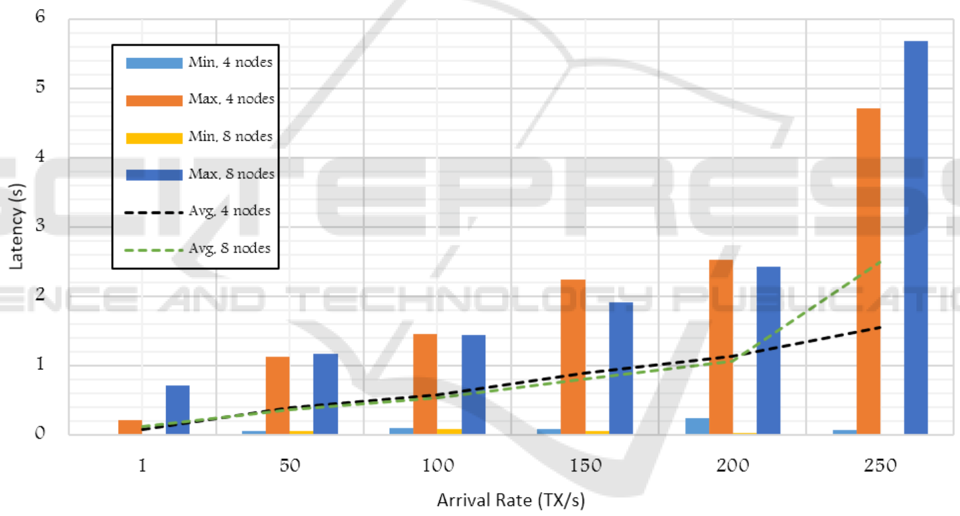

Figure 4: Latency of read TXs (in seconds) measured with different Indy-based Blockchain network sizes (4 and 8) and different arrival rates (1–250).

ing the currently available open-source code of Indy and Aries. The maximum recommended number of nodes participating as BC miners is 25. Adding more nodes is declared to introduce an inefficient system with very high latency. The open-source code currently available allowed us to perform very limited modifications, and we found that the deployment of an enterprise Indy solution requires deep background knowledge of its technicalities. For example, we tried to deploy the Indy node codes on a recent version of the Ubuntu OS but we could not because of very restrictive dependencies of the deployed libraries. We have not found a clear and proactive trust model for adding new Endorsers or Stewards. Thus, adding Endorsers does not necessarily mean they are accredited in the associated legacy system.

6 RELATED WORKS

EBSI[2] and BCDiploma[3] are examples of services built for providing VC supportive platforms utilizing DIDs. The BC acts as the TTP where system entities save their data. However, the standard recom-

[2]https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/EBSI

[3]https://www.bcdiploma.com/en-GB

mendation of W3C[4] and the solutions following it did not consider the issuer accreditation issue assuming any entity should be able to own a DID and issue any type of VCs. The BCDiploma platform directly saves all issued VC hashes on the Blockchain which means that they can never be deleted, and it uses a PoW-based BC with linear DL model. Additionally, issuer accreditation service is not provided. Specifically, EBSI utilizes the general purpose Hyperledger Fabric[5] which uses the Raft Consensus algorithm.

In (Sopek et al., 2018b) and (Raclawickie, 2019), a BC-based data management system that could be used for the Global Legal Entity Identifier System (GLEIS) is proposed. For realizing this, the authors utilized Hyperledger Indy and their previously proposed GraphChain (Sopek et al., 2018a). The implementation and utilization model of the proposed solution were presented, in which some challenges were faced such as the limited message size of Indy.

Bhattacharya et al. (Bhattacharya et al., 2020) examined certain scenarios of personal data disclosure via credential exchanges between different identities and the risks of man-in-the-middle attacks in a BC-based identity system utilizing Hyperledger Indy. On the basis of the findings, the authors proposed enhancing Indy with a novel attribute sensitivity score model for SSI agents to ascertain the sensitivity of attributes shared in credential exchanges. Additionally, they proposed a method for mitigating man-in-the-middle attacks between peer SSIs. Finally, they proposed a novel quantitative model for determining a credential issuer's reputation based on the number of issued credentials in a window period, which is then utilized to calculate an overall confidence level score for the issuer.

Prakash et al. (Prakash et al., 2020) work proposed Indy and Aries agents that are implemented and utilized to realize a connected vehicle information network solution. The authors described several challenges they faced when utilizing those two projects, including the high latency and scalability limitations. However, the paper has not provided specific latency measures and settled for a description of the solution.

Malik et al. (Malik et al., 2021) proposed an architecture for decoupling identities and trade activities on blockchain-enabled supply chains, namely TradeChain. The authors demonstrated the feasibility of TradeChain by implementing a proof of concept implementation on Hyperledger Fabric and Indy. The limited latency evaluations provided for this solution are within the ranges we obtained by our experiments.

A detailed and comprehensive details related to different BC platforms, including Indy and Aries are presented in (Mohanty, 2018). Here, well documented tutorials and implementations can be found, as well as descriptive methods and algorithms for different consensus and block confirmation approaches. Additionally, architectural and low-level details have been differentiated between the studied BC platforms.

Though these works investigate SSI issues with Indy and Aries, none of them has performed detailed latency analysis of them with different system settings concerning network sizes and TX arrival rates.

[4]https://www.w3.org/TR/vc-data-model/#introduction

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

7 CONCLUSION

In this paper, we proposed a deployment architecture for SSI applications, and developed a Python application with containerized components to realize it. We evaluated our proposed architecture in terms of latency by deploying its components in the Google Cloud Platform, at different regions with individual VMs. We executed several test scenarios to obtain average latency measures for different system settings, namely different network sizes and different TX arrival rates. For arrival rates between 1 and 250, we found that the write response latency of an Indy-based BC containing 4 and 8 nodes, ranges between 1 to 16 seconds, while the read response latency with similar settings ranges between 0.01 to 5 seconds. In the future, we plan to enhance the scalability of our application by dynamic, automated BC node addition and removal.

ACKNOWLEDGMENT

The research leading to these results has received funding from the national project TKP2021-NVA-09 implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, and from the UNKP-21-4 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund. The experiments presented in this paper are based upon work supported by Google Cloud.

REFERENCES

Abraham, I., Malkhi, D., Nayak, K., Ren, L., and Spiegelman, A. (2016). Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus. *CoRR, abs/1612.02916*.

Aublin, P.-L., Mokhtar, S. B., and Quéma, V. (2013). Rbft: Redundant byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 297–306. IEEE.

Baniata, H., Anaqreh, A., and Kertesz, A. (2022). Dons: Dynamic optimized neighbor selection for smart blockchain networks. *Future Generation Computer Systems*, 130:75–90.

Bhattacharya, M. P., Zavarsky, P., and Butakov, S. (2020). Enhancing the security and privacy of self-sovereign identities on hyperledger indy blockchain. In *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–7. IEEE.

Bistarelli, S., Mercanti, I., Santancini, P., and Santini, F. (2019). End-to-end voting with non-permissioned and permissioned ledgers. *J. Grid Comput.*, 17(1):97–118.

Boysen, A. (2021). Decentralized, self-sovereign, consortium: The future of digital identity in canada. *Frontiers in Blockchain*, 4:11.

Javaid, U., Aman, M. N., and Sikdar, B. (2020). A scalable protocol for driving trust management in internet of vehicles with blockchain. *IEEE Internet of Things Journal*, 7(12):11815–11829.

Jirgensons, M. and Kapenieks, J. (2018). Blockchain and the future of digital learning credential assessment and management. *Journal of teacher education for sustainability*, 20(1):145–156.

Karasek-Wojciechowicz, I. (2021). Reconciliation of anti-money laundering instruments and european data protection requirements in permissionless blockchain spaces. *Journal of Cybersecurity*, 7(1):tyab004.

Kertesz, A. and Baniata, H. (2021). Consistency analysis of distributed ledgersin fog-enhanced blockchains. In *European Conference on Parallel Processing*. Springer.

Kondova, G. and Erbguth, J. (2020). Self-sovereign identity on public blockchains and the gdpr. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 342–345.

Linux Foundation (2020). Hyperledger indy. hyperledger.org/use/hyperledger-indy.

Makri, K., Papadas, K., and Schlegelmilch, B. B. (2021). Global social networking sites and global identity: A three-country study. *Journal of Business Research*, 130:482–492.

Malik, S., Gupta, N., Dedeoglu, V., Kanhere, S. S., and Jurdak, R. (2021). Tradechain: Decoupling traceability and identity inblockchain enabled supply chains. *arXiv preprint arXiv:2105.11217*.

Mohanty, D. (2018). *Blockchain From Concept to Execution: BitCoin, Ethereum, Quorum, Ripple, R3 Corda, Hyperledger Fabric/SawTooth/Indy, MultiChain, IOTA, CoCo.* BPB Publications.

Nofer, M., Gomber, P., Hinz, O., and Schiereck, D. (2017). Blockchain. *Business & Information Systems Engineering*, 59(3):183–187.

Prakash, N., Michelson, D. G., and Feng, C. (2020). Cvin: Connected vehicle information network. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–6. IEEE.

Raclawickie, A. (2019). Legal entity identifier blockchained by a hyperledger indy implementation of graphchain. In *Metadata and Semantic Research: 12th International Conference, MTSR 2018, Limassol, Cyprus, October 23-26, 2018, Revised Selected Papers*, volume 846, page 26. Springer.

Sopek, M., Gradzki, P., Kosowski, W., Kuziski, D., Trojczak, R., and Trypuz, R. (2018a). Graphchain: a distributed database with explicit semantics and chained rdf graphs. In *Companion Proceedings of the The Web Conference 2018*, pages 1171–1178.

Sopek, M., Grakadzki, P., Kuzinski, D., Trojczak, R., and Trypuz, R. (2018b). Legal entity identifier blockchained by a hyperledger indy implementation of graphchain. In *Research Conference on Metadata and Semantics Research*, pages 26–36. Springer.

Thomas, K., Pullman, J., Yeo, K., Raghunathan, A., Kelley, P. G., Invernizzi, L., Benko, B., Pietraszek, T., Patel, S., Boneh, D., et al. (2019). Protecting accounts from credential stuffing with password breach alerting. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1556–1571.

Tobin, A. and Reed, D. (2016). The inevitable rise of self-sovereign identity. *The Sovrin Foundation*, 29(2016).

Transaction, C. P. and MPI, M. P. I. (2016). Blockchain: Opportunities for health care. *CP Transaction*.

Windley, P. J. (2016). How sovrin works. *Windely.com*.