On using Hyperledger Fabric over networks: Ordering phase improvements

Stavros Dimou, Kostas Choumas and Thanasis Korakis
Dept. of ECE, University of Thessaly, Volos, Greece
Email: dstavros, kohoumas, korakis@uth.gr

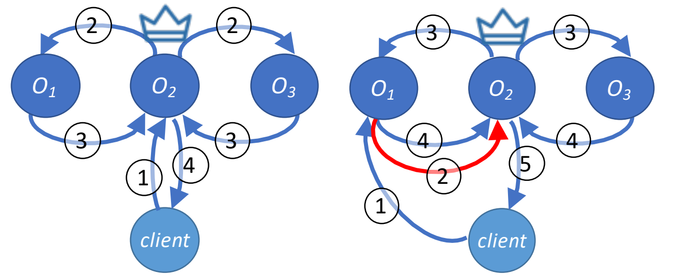
***Abstract*—Blockchain is increasingly being used in various research disciplines, such as Internet of Things (IoT), Software Defined Networking (SDN), logistics, etc. Hyperledger Fabric is a popular enterprise-grade blockchain framework that falls into the category of the permissioned blockchains, making it particularly effective at ensuring transparency for secure communication among peers, such as the SDN controllers of an IoT network. One of the most prominent aspects of Hyperledger Fabric is its three-phase transaction flow architecture, which consists of the execution, ordering and validation phases. The ordering phase involves communication between the client and the ordering service, as the latter is responsible for the transaction assembly and distribution. This study reconstructs the ordering phase and proposes a mechanism for faster communication between the client and the ordering service. While adjusting various blockchain parameters, a performance analysis is carried out to examine the blockchain behavior with the applied mechanism. The experiments reveal performance improvements on various metrics when compared to the existing ordering phase. Notably, the proposed mechanism can be easily integrated in a future Hyperledger Fabric release.**

***Index Terms*—Blockchain, Hyperledger Fabric, Raft, Ordering Service, Performance Analysis.**

I. Introduction

Blockchain technology was first introduced in 2008 as the distributed ledger behind Bitcoin transactions. Since then, the technology has grown in popularity and interest for a variety of applications, including secure communication between SDN controllers in an IoT network [1]. Blockchain creates decentralized ledgers used to record transactions (TXs) between peers without the requirement for external validation. The TXs cannot be modified and are collected into blocks that form a chain of immutable records. Each associated peer has a unique copy of the ledger, and each copy is accurate and consistently updated. For this purpose, blockchain uses a computational process called consensus, where each TX is verified and confirmed by all the peers involved.

Numerous blockchain models have emerged in response to the diverse needs of blockchain users. All of these models are classified into two broad categories: permissionless and permissioned. Permissionless blockchains allow anyone to join the network, making the ledger public. Permissioned blockchains, on the other hand, do not allow anyone to join the network and are only accessible to a specific set of known-identified peers. The permissioned model is adopted by Hyperledger Fabric [2], [3], an open-source project created by the Linux Foundation for secure information sharing among various entities, such as SDN controllers [4]. Hyperledger Fabric is regarded as one of the most efficient blockchain platforms today, employing a three-phase TX flow architecture of *Execution-Ordering-Validation*. This architecture states that during the execution phase, clients send their TX proposals to a specific group of peers for endorsement, and after that, smart contracts are executed generating ledger updates. The ordering phase follows, during which the endorsed TXs are submitted to another specific group of peers, named *ordering service*, to be assembled into blocks and distributed to all other peers. The peers consisting the ordering service are called *orderers*. Finally, during the validation phase, the peers validate the received blocks to ensure that all ledgers remain consistent.



(a) Leader receives TX directly from the client. (b) Leader receives TX indirectly from the client.

Fig. 1. Hyperledger Fabric network with 3 Raft-operated orderers. O_2 is always Leader. The number of each arrow indicates its time-sequence.

It is clear from the TX flow architecture of Hyperledger Fabric that the ordering service has a significant impact on its performance. Hyperledger Fabric provides several protocols for achieving consensus among the orderers, including Solo, Kafka and *Raft* [5], [6]. Raft is the most widespread option at the moment, being Crash Fault Tolerant (CFT) and following a *"Leader & Follower"* model. According to this model, a single orderer is elected as Leader and all others are Followers. All TXs sent to the ordering service, even those delivered to a Follower, are directed to the Leader. The *indirect* communication with the Leader, through a Follower, causes an extra delay in the TX submission, which is depicted with the red arrow of Step #2 in Figure 1(b). This step does not occur in Figure 1(a), where the TX is transmitted *directly* to the Leader.

This paper introduces a mechanism that allows the clients to communicate directly with the Leader. A comprehensive performance analysis of the TX flow architecture of Hyperledger Fabric is presented, showing how the direct communication with the Leader enhances it under various blockchain configurations. The performance analysis is done with the official Hyperledger benchmarking tool, named Hyperledger Caliper. The paper is divided into the following sections. Section II presents related work and Section III describes in detail the proposed mechanism for direct communication with the Leader. Section IV focuses on the benchmarking and the experimentation results. Finally, Section V concludes the paper and presents implications for future work.

II. Related Work

Multiple academic sectors have recently given a lot of attention to the benefits of the blockchain technology, and thus there has been a great deal of interest in the scalability and performance traits of the blockchain networks, like the ones of Hyperledger Fabric [2], as well as the influence of the deployed consensus protocols [7]. Concerning Hyperledger Fabric, there have been many attempts to optimize it's performance, like altering the state of GoLevelDB and the size of StateDB [8], not to mention the experimention with I/O, caching, parallelism and efficient data access to support up to 20K tps [9]. Regarding the validation phase, the authors in [10] propose its improvement by utilizing a chaincode cache during the TX validation and executing StateDB reads in parallel with the validation of transactions, as well as parallel writes to the ledger and the databases. In [11], the authors experiment with similar enhancements, implementing an improved caching mechanism and a validation mechanism that is executed in parallel with the endorsement phase.

As for the ordering phase, in [12], the performance of Fabric++ [13] is evaluated using conflict graphs for each transaction in a block, noting the cycles in these graphs during the ordering phase and creating acyclic graphs that are sent to the validation phase. In [14], a mechanism is proposed that improves the ordering phase but on an older version of Hyperledger Fabric (v1.3) that does not support Raft. Our work differs from theirs, as we extend the latest version of Hyperledger Fabric, by enhancing the Raft-based ordering service with respect to the underlying network, applying the results of [15] and [16] to the blockchain environment.

Apart from the optimization, a significant amount of effort has been devoted to the performance evaluation of Hyperledger Fabric. Following a more theoretical approach, both [17] and [18] create theoretical models concerning Hyperledger Fabric's performance, while experimenting with the three-phase TX flow architecture of Execution-Ordering-Validation. Furthermore, [19] provides the first study to introduce network delays on a PBFT-based blockchain, giving a better inside on how the networking side of blockchain can effect it's performance, while [20] analyzes each part of the TX life cycle separately. Contrary to those, our work is more specific as it gives an overall performance analysis of the blockchain network, while experimenting with network delays and also providing an enhanced ordering service. Lastly, the performance analysis conducted on this paper observes the blockchain's behaviour with the proposed ordering service being integrated, while experimenting with different configurable parameters as explained in [21].

III. Direct Communication with Leader

Most ordering service details are currently hidden from the blockchain clients, which do not know how many orderers exist or which one is the Leader. Each client generates and submits TXs to a randomly chosen orderer, which, unless it is the Leader, forwards the TXs to the Leader (as it is depicted in Figure 1(b)). This simplified approach is well suited to a variety of situations where the latency among the orderers is negligible. However, there are instances where the orderers, for added resilience, may be geographically dispersed, and their interconnections exhibit non-negligible delays. In this case, it is very critical that the proposed mechanism provides the clients with the opportunity to interact directly with the Leader. This way, the overhead caused from the TX redirection is eliminated. Apart from that, the Leader that is in charge of grouping the TXs into blocks handles them more quickly. Considering the aforementioned points, it is anticipated that the *direct communication* with the Leader enhances blockchain performance over the *indirect communication*. Direct communication has the disadvantage of stressing the Leader more in terms of CPU utilization, as we will find out later in our experimentation, but this is a small price to pay for cutting the time needed for each TX and so improving the operability of blockchain.

The fundamental premise of the proposed mechanism is that each client is able to learn the current Leader after communicating with any orderer. The mechanism requires minor adjustments to the communication protocol between clients and orderers. In Hyperledger Fabric, clients submit TXs to an orderer sending a *BroadcastClient* message, and the orderer responds with a *BroadcastResponse* message. Both messages are serialized based on Protobuf and then encapsulated in a TLS header, followed by the TCP/IP headers. The BroadcastResponse message has a *Info* field that may contain additional information about the returned status of the TX submission, but it is not currently used. Our proposal is that the replying orderer should enter the IP address and TCP port of the known Leader in the Info field. In this way, the BroadcastResponse messages delivered by each orderer broadcast the associated connection details of the new Leader, each time the Leader in the ordering service changes. As follows, clients always learn about the new Leader and can interact with it directly, with the exception of the initial TX submissions sent after the Leader changes.

It is also important to point out the differences between the *write* and *read* TXs. The write TXs follow the three-phase flow architecture of Execution-Ordering-Validation, while the read TXs are actually chaincode invocations that are executed without being ordered. The read TXs do not modify the current ledger state and their ordering is not required to keep the ledgers consistent. As a result, read TXs are typically only sent to the execution step of the three-phase flow and are not sent to the ordering service. It is possible for a client to submit a read TX for ordering, though this is uncommon. As follows, our enhancements to the ordering service affect only the write TXs or the few read TXs that are ordered. Next Section IV sheds some light on the performance improvements brought about by our mechanism, as well as how they are influenced by several blockchain parameters.

IV. Performance Evaluation and Analysis

A. Experimental Setup

In this work, Hyperledger Fabric version 2.2.5 is used to build a modified version of the basic blockchain network given by the *test-network* example in the *fabric-samples* directory. Instead of the one orderer found in the original test-network, its modified version has three orderers. As shown in Figure 2, the deployed network is comprised of one channel that connects two peer-organizations, org1 and org2, and an orderer-organization. Each peer-organization consists of one peer, while the ordering organization provides an ordering service built by three orderers.

During the execution phase, the TXs are *submitted* by the client to both peer1 and peer2 for their endorsement, and then, they are sent to one of the three orderers. Whatever is sent by an orderer is delayed with the use of *tc*, simulating the link delays in the ordering service. TXs are sent directly to the Leader in the direct communication, whereas TXs are sent to Follower1 and then forwarded to the Leader in the indirect communication. Finally, the TXs are *committed* by the Leader to all peers and the client, concluding with the validation phase.

The ordering service utilizes the Raft protocol and TLS is enabled on every component, including the client, peers, and orderers, to create a more realistic environment. All components are running in separate Docker version 20.10.12 containers on a single local machine. The local machine contains Intel(R) Core(TM) i7-8550U CPU@1.80GHz $\times 8$, x86_64 architecture, 8 GiB of RAM and runs Ubuntu 22.04 LTS with a disk capacity of 256 GB.

B. Hyperledger Benchmarking Tool

In the context of this work, Hyperledger Caliper (2019) implements the client role. It is the official Hyperledger benchmarking tool, allowing users to analyze the performance of the blockchain networks. Based on customized use cases, Hyperledger Caliper generates performance analyses that include their input parameters and different performance indicators, such as the send rate, the latency, the throughput and resource utilization (more details below). It is configured to submit write TXs, each of which adds a new asset to the ledger. No read TXs are involved in the evaluation of the proposed mechanism, since they are not affected by the improved ordering service.

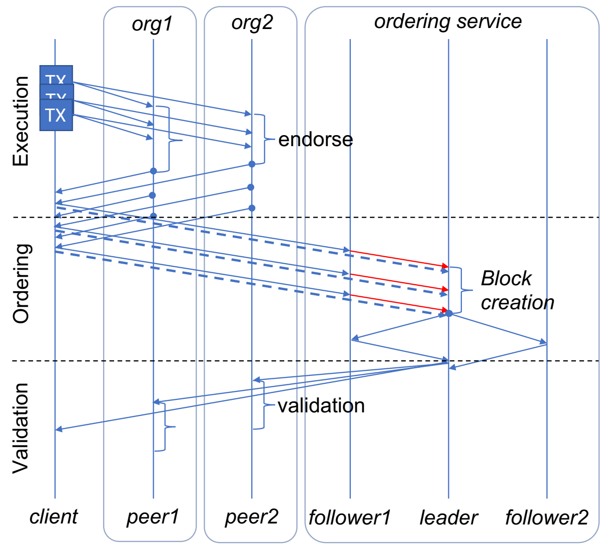


Fig. 2. The three-phase TX flow. The red arrows represent the extra step caused by the indirect communication with the Leader.

C. Evaluation Metrics

The following metrics offered instantly or easily generated by the outputs of Hyperledger Caliper, and will be referred to in the evaluation results[1]:

1) *Successful-TXs*: The number of successfully committed TXs as a percentage of all submitted TXs.
2) *Latency*: The average difference in time between the submission and the commitment of each individual TX, measured in seconds.
3) *Throughput*: The rate at which TXs are committed. It is measured in TXs per second (tps) and it is equal to (successful + failed TXs)/(last TX committing time − first TX submitting time).
4) *CPU-usage*: The Docker CPU usage of an orderer due to the blockchain operations.

D. Experimentation Results

Multiple series of experiments are conducted to show how the proposed mechanism affects the performance of the modified test-network. The blockchain performance has been extensively tested with various values for parameters such as the *BatchSize*, the *BatchTimeout* and the *SendRate*. The BatchSize defines the number of TXs in a block, while BatchTimeout describes the amount of time to wait for additional TXs before creating a new block, after the first TX arrives. The SendRate is the rate that Hyperledger Caliper is generating and submitting TXs. Various link delays are also configured with the tc assistance.

The experiments conducted are initially executed with BatchSize equal to 1 and BatchTimeout configured to its default value of 2 seconds. For each of those experiments, the link delay is either 100, 200 or 300 milliseconds. SendRate

[1]Having in mind that when a TX is submitted, processing of that TX begins, and when a TX is committed, processing of that TX ends.

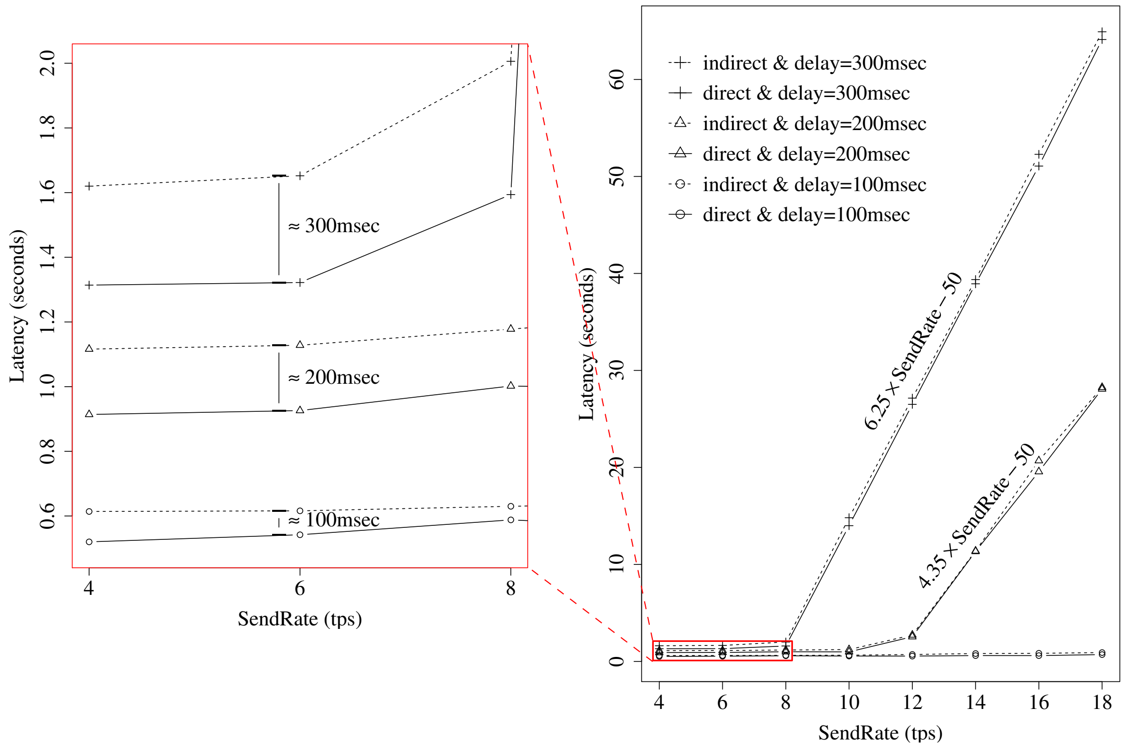


Fig. 3. Latency for various SendRates, under three link delays and either indirect or direct communication with the Leader.

is either an even number ranging from 2 and 18 tps or a multiple of 10 ranging from 20 and 50 tps. The duration of each experiment is set to 100 seconds. The rate control of Hyperledger Caliper is configured to create TXs with fixed rate.

For each configuration, the experiment is run under either the direct or the indirect communication. The presented experiments lead to some intriguing results. The main deduction is that the direct communication with the Leader results to an improvement in the average TX latency. As shown in Figure 1, the in between step to deliver the TXs to the Leader from another orderer, causes an extra delay. Thus, the Latency of the indirect communication will be increased by the time needed for the communication between the two orderers. This is evident on Figure 3, where the difference in Latency between the direct and the indirect communications is almost equal to the link delay, which is either 100, 200 or 300 milliseconds. This fact is even more evident in the left plot of Figure 3, which provides a closer view of a portion of the right plot, zooming in a smaller range of SendRates (4, 6 and 8 tps).

In addition, as link delay and, in particular, SendRate increase, many TXs fail, especially in the case of indirect communication. The orderers cannot handle the increasing load on time and the corresponding timeouts of Hyperledger Caliper and Hyperledger Fabric expire. Specifically, Hyperledger Caliper uses a timeout for bounding the time waiting for a TX to be committed, which is equal to 60 seconds by default. Hyperledger Fabric uses another timeout, with a default value

TABLE I
SUCCESSFUL-TXS FOR VARIOUS SENDRATES (IN TPS), LINK DELAYS (IN MILLISECONDS), BATCHSIZE = 1 AND EITHER INDIRECT OR DIRECT COMMUNICATION WITH THE LEADER.

method / delay / SendRate	indirect			direct		
	100	200	300	100	200	300
20	99.9%	13.1%	6.9%	100%	74.2%	42.3%
30	20.5%	6.0%	2.9%	99.8%	35.9%	20.4%
40	10.0%	3.8%	0.3%	70.9%	24.4%	15.2%
50	1.9%	3.0%	0.0%	45.9%	18.6%	10.8%

of 7 seconds, which bounds the time an orderer waits a response from the Leader. Table I shows the Successful-TXs for various experiments and, obviously, direct communication outperforms indirect, with BatchSize = 1.

Under relaxed timeouts (configured with their maximum values), the TXs are always successful and Latency increases with SendRate. For high SendRates, Latency is given by the following equation,

$$\begin{aligned}\text{Latency} &= \sum_{n=1}^{N}\left(\frac{n}{\mu}-\frac{n-1}{\lambda}\right)/N \\ &= \frac{1}{\lambda}+\frac{N+1}{2}\left(\frac{1}{\mu}-\frac{1}{\lambda}\right),\ \forall\lambda\geq\mu, \qquad (1)\end{aligned}$$

where N is the number of packets sent during the experiment, λ is the SendRate and μ is the maximum Throughput that could be achieved. For low SendRates, where $\lambda<\mu$, Latency

TABLE II

SUCCESSFUL-TXS FOR VARIOUS SENDRATES (IN TPS), INTER-ORDERER DELAYS (IN MILLISECONDS), BATCHSIZE $= 2$ AND EITHER INDIRECT OR DIRECT COMMUNICATION TO THE LEADER.

method	indirect			direct		
delay / SendRate	100	200	300	100	200	300
20	100%	100%	30.8%	100%	100%	100%
30	100%	26.2%	8.5%	100%	99.9%	69.1%
40	99.9%	10.5%	5.3%	100%	86.4%	44.3%
50	26.5%	6.4%	3.7%	100%	50.6%	29.3%

TABLE III

SUCCESSFUL-TXS FOR VARIOUS SENDRATES (IN TPS), INTER-ORDERER DELAYS (IN MILLISECONDS), BATCHSIZE $= 5$ AND EITHER INDIRECT OR DIRECT COMMUNICATION TO THE LEADER.

method	indirect			direct		
delay / SendRate	100	200	300	100	200	300
20	100%	100%	100%	100%	100%	100%
30	100%	100%	14.4%	100%	100%	100%
40	100%	21.4%	4.0%	100%	100%	90.2%
50	100%	10.6%	4.8%	100%	100%	62.1%

is equal to $1/\mu$. By increasing the SendRate and tracking the increasing Throughput until an upper threshold, the rate μ is experimentally determined to be equal to this threshold. For higher SendRates, where $\lambda > \mu$, Throughput is always equal to μ and does not increase. For link delay equal to 100, 200 and 300 milliseconds, μ is experimentally measured to be equal to 21.1, 11.5 and 8.0 tps respectively. Defining $t = 100$ seconds as the time duration of each experiment, then $N = t\lambda$ and Eq. 1 becomes equivalent to the following equation,

$$\text{Latency} = \frac{1}{\lambda} + \frac{t\lambda + 1}{2}\Big(\frac{1}{\mu} - \frac{1}{\lambda}\Big) \approx \frac{t}{2}\Big(\frac{\lambda}{\mu} - 1\Big). \qquad (2)$$

From Eq. 2, for SendRates higher than the maximum Throughput ($\lambda > \mu$) and link delay equal to 100, 200 and 300 milliseconds, Latency is approximately given by the following linear models: $50(\lambda/21.1-1) = 2.37\lambda - 50$, $50(\lambda/11.5-1) = 4.35\lambda - 50$ and $50(\lambda/8.0-1) = 6.25\lambda - 50$ respectively. Two of these three models are depicted in Figure 3.

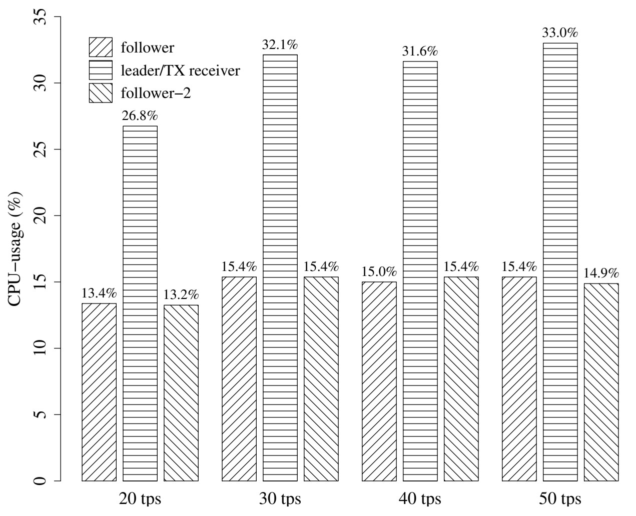
During all experiments, the CPU-usage for all the orderers remain on normal levels, without being stretched. On the direct communication with the Leader, the Leader has a higher CPU-usage than the other orderers, due to the TX traffic being directly forwarded to it, while on the default case the Leader together with the orderer receiving this traffic have slightly higher CPU-usage than the others. This indicates that the ordering nodes are not stretched from heavy load and are able to cope with that workload, if the default timeouts are altered to higher values. Figure 4 portrays the CPU-usage of each orderer for various SendRates and link delay equal to 100 milliseconds, either during the direct communication (Figure 4(a)), with the Leader having higher percentages, or during the indirect communication (Figure 4(b)), where the Leader and the first contacted orderer (TX receiver) have almost similar CPU-usages and higher than the third orderer. Thus, in terms of CPU-usage, indirect communication appears to have an advantage in the need for load balancing and resource sharing among the orderers, which is most likely the rationale for choosing an orderer at random to receive TXs.

Another important conclusion obtained by the experiments is regarding the BatchSize. As stated in [21], increasing BatchSize leads to higher Successful-TXs, because there are more TXs per block and the number of blocks eventually decreases. Moreover, as it is depicted in Figure 2, the ordering phase is repeated once per block, thus its delay is introduced less times during the whole experiment duration and its negative effect to Successful-TXs is getting less and less significant. The experimentation results presented in Table II and Table III, indicate that for increasing BatchSize but low SendRates, assuming that the delay is equal to e.g. 100 milliseconds, the Successful TXs rate is 100% for both direct and indirect communications. This makes the performance of Successful-TXs to be irrelevant to the usage of the direct or indirect communication method for higher BatchSizes. Consequently, for lower BatchSizes the direct communication will have a higher impact on Successful-TXs, compared to the indirect one. However, the threshold between low and high BatchSizes increases with SendRate and link delay, making the direct communication beneficial for even higher range of BatchSizes.

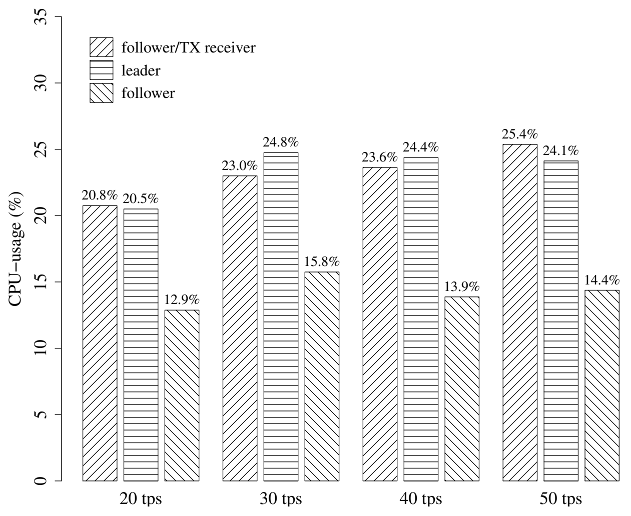
V. CONCLUSIONS & FUTURE WORK

In this paper, we focused on improving the three-phase TX flow architecture of Hyperledger Fabric, and specifically the ordering phase. Our main contribution is to propose a mechanism implemented on the clients and the Raft-operated orderers of Hyperledger Fabric, in order to enable the direct communication between the clients and the orderer that is the Raft Leader. This way, the overhead caused by the indirect communication with the Leader, through another intermediate orderer, is eliminated. In order to observe the blockchain performance with the implemented mechanism, a performance analysis study was conducted, by configuring blockchain parameters, like BatchSize and BatchTimeout, as well as parameters outside of the blockchain control, like the SendRate and the link delay between the orderers. The proposed mechanism leads to hopeful results, improving the average TX latency and the number of successful TXs on several occasions.

As part of future work, we will investigate the effects of the proposed mechanism on Hyperledger Fabric by utilizing various blockchain networks. For instance, we will incorporate several numbers of organizations with multiple nodes and we will increase the number of nodes in the ordering service. To satisfy this purpose, the experimentation will rely on a testbed, which will provide a more realistic networking environment. Furthermore, as in [9], the blockchain network will be tested with a much higher workload to simulate an even more realistic setup, consisting of multiple parallel TXs. On top of that, there is an intention to extend the study on other Raft-



(a) Under the direct communication.



(b) Under the indirect communication.

Fig. 4. The CPU-usage of the three orderers.

operated blockchain technologies and distributed systems that can be optimized.

Acknowledgment

The research leading to these results has received funding from the European Horizon 2020 Programme for research, technological development and demonstration under Grant Agreement Number No 101058580 (H2020 RECONMATIC). The European Union and its agencies are not liable or otherwise responsible for the contents of this document; its content reflects the view of its authors only.

References

[1] A. Yazdinejad, R.M. Parizi, A. Dehghantanha, Q. Zhang, and K. Choo. An Energy-Efficient SDN Controller Architecture for IoT Networks With Blockchain-Based Security. *IEEE Transactions on Services Computing*, 13(4):625–638, 2020.

[2] Elli Androulaki et al. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proc. of EuroSys*, 2018.

[3] Hyperledger Fabric. https://www.hyperledger.org/use/fabric.

[4] M. Hajizadeh, N. Afraz, M. Ruffini, and T. Bauschert. Collaborative Cyber Attack Defense in SDN Networks using Blockchain Technology. In *Proc. of IEEE NetSoft*, 2020.

[5] D. Ongaro and J. Ousterhout. In Search of an Understandable Consensus Algorithm. In *Proc. of USENIX ATC*, 2014.

[6] The Raft Consensus Algorithm. https://raft.github.io/.

[7] X. Jiang, M. Liu, F. Zhao, Q. Zhou, and R. Wang. Fast Adaptive Blockchain's Consensus Algorithm via Wlan Mesh Network. In *Proc. of SICBS*, 2020.

[8] T. Nakaike, Q. Zhang, Y. Ueda, T. Inagaki, and M. Ohara. Hyperledger fabric performance characterization and optimization using goleveldb benchmark. In *Proc. of IEEE ICBC*, 2020.

[9] C. Gorenflo, S. Lee, L. Golab, and S. Keshav. FastFabric: Scaling Hyperledger Fabric to 20000 Transactions per Second. *International Journal of Network Management*, 30:e2099, 2020.

[10] H. Javaid, C. Hu, and G. Brebner. Optimizing validation phase of Hyperledger Fabric. In *Proc of. IEEE MASCOTS*, 2019.

[11] P. Thakkar, S. Nathan, and B. Viswanathan. Performance benchmarking and optimizing Hyperledger Fabric blockchain platform. In *Proc. of IEEE MASCOTS*, 2018.

[12] J. Chacko, R. Mayer, and H. Jacobsen. Why do my blockchain transactions fail? A study of Hyperledger Fabric. In *Proc. of SIGMOD*, 2021.

[13] A. Sharma, F.M. Schuhknecht, D. Agrawal, and J. Dittrich. Blurring the Lines between Blockchains and Database Systems: the Case of Hyperledger Fabric. In *Proc. of SIGMOD*, 2019.

[14] M. Kwon and H. Yu. Performance Improvement of Ordering and Endorsement Phase in Hyperledger Fabric. In *Proc. of IOTSMS*, 2019.

[15] K. Choumas and T. Korakis. When Raft Meets SDN: How to Elect a Leader over a Network. In *Proc. of IEEE NetSoft*, 2020.

[16] K. Choumas and T. Korakis. On using Raft over Networks: Improving Leader Election. *IEEE TNSM*, 19(2):1129–1141, 2022.

[17] H. Sukhwani, N. Wang, K.S. Trivedi, and A. Rindos. Performance Modeling of Hyperledger Fabric (Permissioned Blockchain Network). In *Proc. of IEEE NCA*, 2018.

[18] X. Xu, G. Sun, L. Luo, H. Cao, H. Yu, and A.V. Vasilakos. Latency performance modeling and analysis for hyperledger fabric blockchain network. *Information Processing & Management*, 58:102436, 2021.

[19] T. Nguyen, G. Jourjon, M. Potop-Butucaru, and K. Thai. Impact of network delays on Hyperledger Fabric. In *Proc. of IEEE INFOCOM Workshops*, 2019.

[20] C. Wang and X. Chu. Performance Characterization and Bottleneck Analysis of Hyperledger Fabric. In *Proc. of IEEE ICDCS*, 2020.

[21] S. Shalaby, A. Abdellatif, A. Al-Ali, A. Mohamed, A. Erbad, and M. Guizani. Performance Evaluation of Hyperledger Fabric. In *Proc. of IEEE ICIoT*, 2020.