2019 International Conference on Networked Systems (NetSys)

Performance Perspective on Private Distributed Ledger Technologies for Industrial Networks

Fabien Geyer*†, Holger Kinkelin*, Hendrik Leppelsack*,
Stefan Liebald*, Dominik Scholz*, Georg Carle*, Dominic Schupke†

*Technical University of Munich
Emails: lastname@net.in.tum.de

†Airbus, Munich
Emails: firstname.lastname@airbus.com

***Abstract*—Blockchain-based Distributed Ledger Technology (DLT) is a novel paradigm to create tamper-resistant execution environments and data storage for distributed applications on top of a peer-to-peer network. This technology has shown to be of interest in many use-cases, especially in industrial processes where multiple shareholders would like to process and share data in a secure and accountable way.**

In this work, we evaluate the performance of a DLT-based system via modeling and a quantitative performance evaluation, focusing on the impact of the underlying communication network. Our numerical evaluation is based on the Hyperledger Fabric DLT framework, its benchmarking tool Caliper, and a dedicated test bed, where network properties such as latency or packet loss can be artificially influenced.

Our experiments show that the validation of the transactions in a DLT-based system is the main contributor in the transaction latency. We also demonstrate that the properties of the communication network can influence the performance largely, even in the case where only one of the participants in the DLT system has poor network access.

I. Introduction

In 2008, Nakamoto [1] proposed the decentralized payment system *Bitcoin*, whose essential idea is to use a *blockchain-based distributed ledger* to store verified financial transactions between its users. Both, the logic that verifies financial transactions as well as the logic that maintains the blockchain are executed on nodes of a peer-to-peer (P2P) network. The most important properties of this novel paradigm of providing a service are *immutability* and *indelibility* of the data stored in the blockchain and that the entire system can be operated without any trusted elements.

Several years later, in 2015, *Ethereum* [2] was the first *Distributed Ledger Technology (DLT)* that separated the logic of an application from the algorithms that maintain the blockchain. So instead of implementing just one application as Bitcoin does, Ethereum is a framework that enables its users to specify own application logic in so called *smart contracts*, and to store and execute these smart contracts in the Ethereum P2P network. A smart contract can be executed by users by publishing a *transaction* into the Ethereum network. The data contained in the transactions, as well as the output computed by the smart contracts, are persisted in the blockchain.

Ethereum is a highly interesting technology for companies that want to execute code or to store data in a highly trustworthy and tamper-proof manner. However, Ethereum is inherently limited concerning *performance* and *confidentiality*, which both hinder its applicability in certain scenarios.

The performance of Ethereum with regard to transaction throughput is limited because of the *consensus algorithm* and the *block size*. Consensus algorithms are used in DLTs to agree on the next block of transactions. As of October 2018, Ethereum uses consensus by *proof of work (PoW)*, which is a race of peers trying to solve a cryptographic puzzle depending on the transactions included in a new block. While solving the puzzle requires brute force, other peers can verify the solution efficiently.

The average time needed to create a block and also the block size can be adjusted. As of October 2018, the average time for creating a new block is about 15 s[1] and the average block size is around 17 000 transactions[2]. As the queue of pending transactions has about 40 000 elements[3], a transaction should be completed after 30 s to 40 s, which is inappropriate for certain applications, especially in industrial use-cases.

The maybe more serious problem is that Ethereum is a *public* system, which means that everybody can participate in the network and download a copy of the blockchain. For this reason, it is problematic to store confidential information as plain text in the ledger. Encryption of certain data elements contained in a transaction is possible. This approach, however, creates the dilemma that smart contracts are limited to process plain text information only.

To overcome these and other restrictions, various projects with the aim of creating a new class of DLT, often referred to as *corporate blockchains*, were launched. Corporate DLTs are typically *private* networks, which means that some type of identity management system is employed that allows peers and clients to authenticate themselves as being part of the same private network. One prominent example of a corporate DLT and also the subject of our study is Hyperledger Fabric [3, 4], a project from the Hyperledger project of the Linux Foundation.

A. Contributions and Structure

In this work, we conduct experiments with the corporate DLT framework Hyperledger Fabric. As a difference and addition to related work, we focus on performance impacts in DLTs caused by effects in the network such as increased latency

[1] https://etherscan.io/chart/blocktime
[2] https://etherscan.io/chart/blocksize
[3] https://etherscan.io/chart/pendingtx

and packet loss rates. These can e.g. occur if DLT clients or peers are connected via wireless technologies like Wi-Fi or 4G/5G to the rest of the DLT network. We show that those parameters heavily influence the transaction latency. We also demonstrate that the communication network properties of a single shareholder in the DLT system can influence the performances of other shareholders in the system.

The rest of this paper is structured as follows. In Section II we give an overview of the different elements which have been used in our experiments, namely Hyperledger Fabric and Hyperledger Caliper, a DLT benchmarking system.

In Section III, we review related work evaluating distributed ledger performance. We discuss metrics suitable to measure DLT performance in Section IV and propose a mathematical model based on queuing theory for modeling the system. We analyze factors that influence DLT performance and metrics to asses performance in Section V. Our experiment system, which extends Caliper with the ability of conducting experiments with artificially influenced network properties, and our test bed setup are explained in Section VI. The subsequent Section VII presents our experiments and discusses findings. The paper is concluded in Section VIII.

II. Background

A. Hyperledger Fabric

Hyperledger Fabric [3] is a framework for operating a private distributed ledger developed by the *Hyperledger* project of the Linux Foundation [5]. Hyperledger Fabric allows to specify business logic in *chaincode* that runs on a distributed network of Hyperledger Fabric peers. In contrast to Ethereum's smart contracts, chaincode is only installed on a subset of all available peers called *endorsing peers*, see Figure 1. For each chaincode there is also an *endorsement policy* that specifies how many and which endorsers have to verify and endorse a transaction.

1) Transaction Flow: Transactions execute chaincode to create a new data asset or to modify an existing one in the blockchain. Hyperledger Fabric organizes the transaction flow in three phases. In the *execute* phase, a Hyperledger Fabric client *invokes* a chaincode operation by sending a *transaction proposal* (step ① in Figure 1) to the endorsing peers. Each of them *simulates* the transaction and – if the transaction is appropriate – returns a signed *endorsement* to the client ②. Endorsements contain a so-called *read/write set*, i.e., values read and written by the chaincode.

If the read/write sets contained in all endorsements are equal, the *ordering* phase commences. Now, the client sends the transaction together with the endorsements to the *ordering service* ③, an entity responsible to collect, order and release new blocks containing transactions of all clients in the network. Also see Section II-A2.

In the final *validation* phase, the block is delivered to *all* peers in the network ④ – also to so-called *committing peers* that do not have the chaincode installed. All peers append the new block to their local blockchain copy. Subsequently, they validate each transaction in the block. This includes checks if signatures are correct, if the read/write sets are identical, and if

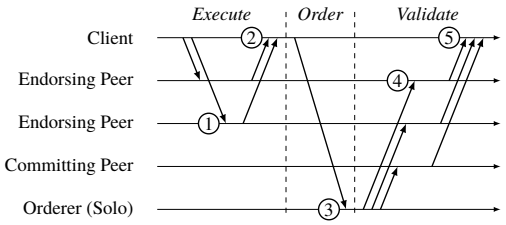

Figure 1: Hyperledger Fabric message flow

the endorsement policy is fulfilled. Furthermore, peers check if the read set of a transaction matches to their local *world state* to prevent version conflicts caused by race conditions. The world state is a local database that contains the accumulated result of sequentially applying transactions contained in all blocks ever created in the network. If all checks are passed, the write set of the transaction is applied to the node's world state and the client is notified about the successful execution of the transaction ⑤. If a check fails, the transaction is marked as being invalid and not applied to the world state.

2) Ordering Service: Hyperledger Fabric uses a *ordering service* to create new blocks. However, Hyperledger Fabric does not depend on a specific implementation. As of October 2018, three implementations exist or are under development[4]: 1) *Solo*, a centralized, not fault tolerant ordering service for testing purposes; 2) a distributed, fault tolerant orderer based on Apache Kafka; and 3) a distributed, Byzantine fault tolerant orderer, which is still under development.

B. Caliper

Caliper [6] is a Hyperledger project with the aim of creating an agreed-on benchmarking tool that helps to compare the performance of *different* DLT implementations for a specific application. During experiments, Caliper acts as a DLT client and automatically performs pre-defined DLT operations, like deploying chaincode to peers, invoking chaincode with a transaction, or querying the blockchain's state.

Caliper is also able to measure various performance-related properties, such as throughput, transaction latency, resource utilization, etc. However, Caliper's view on the DLT-based system is limited to clients when executed in a distributed setup, i.e., peers and the ordering service *cannot* be observed.

Another limitation of Caliper is that it is not able to influence the test bed the DLT is running on. For this reason, no external performance influencing factors, see Section V-A, can be adjusted automatically, which would allow to observe potential performance implications.

III. Related work

Dinh et al. [7] proposed BLOCKBENCH, one of the first frameworks for evaluating the performance of different blockchain platforms, namely, Ethereum, Parity and Hyperledger Fabric using various benchmarks. Their set of benchmarks include both specific ones targeting the evaluation of

[4]https://hyperledger-fabric.readthedocs.io

a specific aspect of the blockchain (eg. processor utilization, network), as well as benchmarks based around realistic patterns from realistic workloads. They identified the consensus protocols as the main bottleneck for Hyperledger Fabric and Ethereum, and transaction signing for Parity. The authors further investigated the performance of blockchains in [8], by including additional results such as the comparison between two different Hyperledger Fabric versions.

Pongnumkul et al. [9] investigated the performance of Hyperledger Fabric and Ethereum by measuring transaction deployment and completion time. Compared with [7], they showed that the difference between Ethereum and Hyperledger Fabric becomes even more significant with larger number of transactions. Their measurements is based on a single machine setup, meaning that effects of the communication network is not taken into account.

Yasaweerasinghelage et al. [10] proposed to model the latency of blockchain-based systems using architectural performance modeling and simulation. Their main goal for those models was to support architectural design and what-if evaluations for design changes.

Shbair et al. [11] presented a framework to test blockchains and blockchain applications based on the Grid'5000 large-scale test bed. As a showcase for their framework, they evaluated a specific banking use-case and measured transaction latency.

Thakkar et al. [12] analyzed the impact of various configuration parameters of Hyperledger Fabric such as block size, endorsement policy, channels, resource allocation, state database choice on transaction throughput and latency. They identified three main bottlenecks, namely cryptographic operations, validation of transactions, and database calls. Compared with our work, they mainly focused on CPU utilization and assumed that the network is not a bottleneck.

Selimi et al. [13] recently performed an evaluation of Hyperledger Fabric in a wireless mesh network based around Raspberry Pis, with the goal of measuring transaction latency and CPU resource consumption. Based on their finding, they proposed a placement scheme for software components that optimizes the performance of the blockchain protocol.

Göbel et al. [14] studied the effect of communication delay on the evolution of the Bitcoin blockchain based on a Markov model and validated their results against two different simulators. They showed in their analysis that nodes with high communication delay may be at a disadvantage compared to other nodes participating in the network.

Kawase and Kasahara [15] recently proposed a mathematical model of the performance of blockchain based on queuing theory using a single-server queue model with batch service M/G^B/1. They validated their findings using simulation as well as a trace of the Bitcoin network from October 2013 to September 2015.

IV. Model

We present in this section a mathematical model for estimating the average transaction latency.

A. Transaction latency

In order to measure the transaction latency, we split this latency in the three following steps:

- *Endorse*: After the DLT client creates a transaction (corresponding to step ① in Figure 1), the endorsing peers execute the related chaincode and answer with endorsements (step ② in Figure 1).
- *Order*: Subsequently, the DLT client sends the transactions together with the related endorsements to the ordering service (step ③ in Figure 1).
- *Validate*: The orderer creates a new block containing the transaction and delivers this block to all peers in the test network (step ④ in Figure 1). Finally, all peers validate and apply the transactions contained in this block and signalize the completion of the transaction to the DLT client (step ⑤ in Figure 1).

The total transaction latency corresponds to the sum of the three steps.

In case neither the underlying network nor the CPUs constitute a bottleneck in the system, the main contributor to the transaction latency is the validation. This is illustrated later in Section VII and was also noted in [7, 8, 12]. This behavior is explained by the fact that the validation step constitutes a single point where all transactions have to be processed.

In order to write the different transactions, the ordering service batches multiple ones into blocks and outputs a hash-chained sequence of blocks containing transactions. The goal of this batching process is to improve the the throughput of the broadcast protocol of Hyperledger Fabric.

This process is illustrated in Figure 2. Transactions arrive at different time at the orderer and are queued. Once the queue reaches a given threshold, the orderer at once processes K transactions, where K corresponds to the block size.

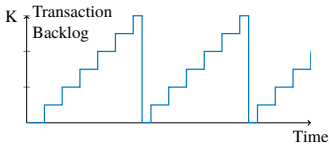

Figure 2: Illustration of batch service of the orderer

The service of our system can then be modeled as a single-serve queue model with bulk service. Such modeling is also found in other applications, such as traffic lights, service facilities or railroads [16]. We propose in this work to model our system as a $M/M^B/1$ system. This system describes a process where transactions arrive according to a Poisson process and are processed according to exponential service time, with service taking place in batches of size B. While we restrict here the model to exponential service, we note that a $M/G^B/1$ model was used in [15] for analyzing the Bitcoin network. We derive the performance model of the orderer based on the results from Medhi [17], which studies the $M/M_{a,b}/1$ queue, where batch sizes may take values between a and b. In our case, we have $a = b = K$. Let λ represent the submitted

transaction rate and μ the service rate. The average transaction time corresponds to the average waiting time in the system:

$$E[T] = \lambda P_{0,0} \frac{d}{d-1} \left[\frac{1}{\mu^2(1-d^{-B})} + \frac{B(B-1)}{2\lambda^2} - \frac{(d^B-1) - B(d-1)}{\lambda^2 d^B (d-1)^2} \right] \quad (1)$$

with

$$P_{0,0} = \frac{d^{B-1}(d-1)}{(B+\lambda/\mu)d^B - (d^{B-1} + d^{B-2} + \ldots + 1)} \quad (2)$$

$$\frac{\lambda}{\mu} = \frac{d^{-B}(d^B-1)}{d-1} \quad (3)$$

We refer to [17] for the formal proof of Equation (1).

Figure 3 illustrates Equation (1) for different values of K and different values of λ, i.e. the submitted transaction rate. As expected, increasing the rate results in the faster processing time since blocks can be generated faster.

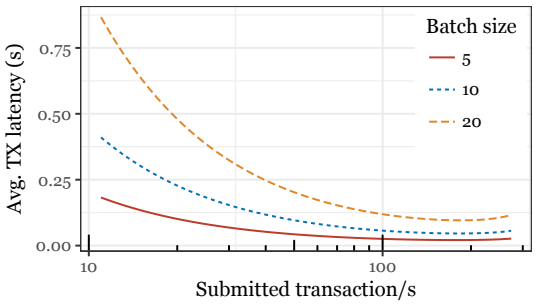

Figure 3: Transaction latency according to $M/M^K/1$ model

V. Performance Indicators

A. Performance Influencing Factors

Next to the model described in Section IV, additional parameters have be taken into account for evaluating the performance influencing the DLT. To facilitate the analysis of influencing factors, we divided the entire system in different layers, as illustrated in Figure 4.

The *underlay network* is the bottom-most one and used to transport packets between clients, peers and the ordering service via TCP/IP. Factors expected to influence performance include available bandwidth, latency, and packet loss, as illustrated later in Section VII. The underlay network is the main focus of the evaluation in Section VII.

The *overlay network* is a logical topology of clients and peers participating in the DLT network. The design of the overlay is expected to affect performance as it determines which peers interact with each other. Depending on the peers' locations, effects on the network layer are expected to be more or less pronounced. For instance, a local network connecting a DLT-based system is expected to be more reliable and performant compared to a wide area network.

The *node level* consists of physical or virtual machines that run peer processes and chain code, and store the DLT state. The properties of nodes, i.e., their available computing power, memory, system load, I/O performance, etc. are expected to have an influence on DLT performance.

On the *ledger level*, performance of the DLT-based system can be influenced by different configurations. In the example of Hyperledger Fabric, this includes the endorsement policy, which dictates how many and which peers must endorse a transaction. A further factor expected to influence performance is the block size, as modeled in Section IV.

On the *application layer*, the business logic implemented in chaincode is expected to influence the DLT-based system's performance. This is because different applications induce different *workloads* as, for instance, computations of varying complexity are involved or because varying amounts of data need to be transferred.

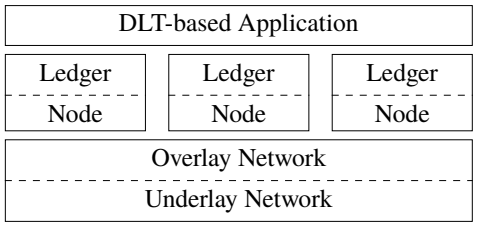

Figure 4: Layers of DLT-based systems.

B. Performance Metrics

In the same way as having specified factors that can influence performance by layers, additional metrics may be used to measure performance. On the network layer, the number of packets sent between clients, peers and ordering service, the amount of data in bytes and finally the amount of transactions can be measured.

On the node layer resources consumed by the different entities, such as CPU time, memory (RAM/cache/persistent memory), network and disk I/O, can be determined.

On the ledger layer of a Hyperledger Fabric network, we must differentiate between different entities, namely clients that issue transactions, endorsers that validate transactions, committers that apply the transaction, and the orderer service that creates blocks of transactions. For our purpose, it is most valuable and comparably easy to observe the network from the standpoint of a client. This is because a client is highly involved in processing the transaction it has proposed as described in Section II-A. For this reason, we observe the different transaction phases and the outcome of the transaction from the client. With this ability, it becomes possible to measure read and write transaction latencies, transaction throughput and backlog, as well as the transaction success rate.

VI. Implementation of Experiments

Albeit we have designed our experiment system to be as generic as possible with regard to the used experimental environment and the actual system under test, we limit ourselves to discussing our actual setup and measurements.

A. Experiment Workflow

The overall workflow of conducting experiments consists of three phases, namely *deployment*, *measurement*, and *evaluation* phase. While the first two phases are executed in the test bed, the evaluation of measurement results is executed on a single machine.

a) Deployment Phase: The deployment phase prepares the available test bed for subsequent experiments. At first, each host is booted from a pre-prepared, customized Debian 9 image with a minimal footprint. Next, Caliper is installed to a *benchmark host*, Hyperledger Fabric to several *ledger hosts* and finally the Solo orderer to the *orderer host*.

b) Measurement Phase: The measurement phase is more complex, as depicted in Figure 5. At first, we configure the measurement environment. This includes settings concerning the ledger, like the block size, or settings that concern the network, like loss rates, latency, etc. Now the measurement is started. We first synchronize the time of all used machines towards a local NTP server. After these steps, the actual experiment is started and Caliper executes workloads we have pre-defined by executing transactions. During this process, Caliper is logging transaction- and ledger-related information, like transaction state and timestamps of the different events associated to transactions. Lastly, Caliper logs are gathered from the experiment hosts.

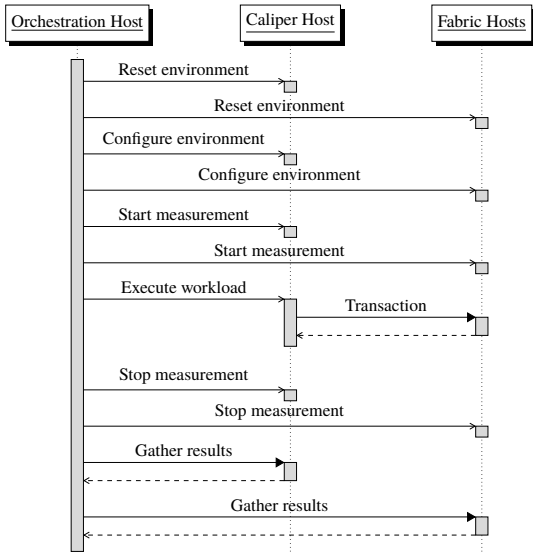

Figure 5: Measurement Workflow.

c) Implementation of Test Automation: We automated the deployment and measurement phases using two tools: the physical test bed itself is prepared using a management framework running on the orchestration server, which we developed to orchestrate experiment test beds at our chair, called plain orchestrating service (pos) [18]. The goal of pos is to provide a reusable infrastructure that enables experiment automation. It provides test bed users an abstraction layer for regular experiment tasks like restarting test nodes with a clean image or gathering experiment artifacts, including executed scripts, configuration files and generated experiment outputs. This fosters repeatable and reproducible experiments. pos can be easily adapted to meet requirements of other test beds.

As soon as the physical test bed is prepared, all remaining steps are automated using *Ansible* [19], which is an open source orchestration tool widely used in DevOps. Besides deploying software and configurations, Ansible allows to execute arbitrary scripts or commands on remote hosts and it can transport data to and from remote machines.

VII. Numerical evaluation

In this section, we present the experiments that we have conducted and discuss and interpret their outcomes.

A. Experimental Environment

To develop our test system and to perform experiments, we used a small test bed consisting of 12 identical machines (quad core Intel Xeon CPU E3-1265L V2 @ 2.50GHz, 16 GB memory, 120 GB SSD) linked via multiple Ethernet switches connected in a daisy chain and 1 Gbit/s links.

The available machines for experiments were assigned to different roles as depicted in Figure 6: one machine runs the Solo orderer, one machine runs Caliper, the remaining machines are used as peers assigned to two different organizations.

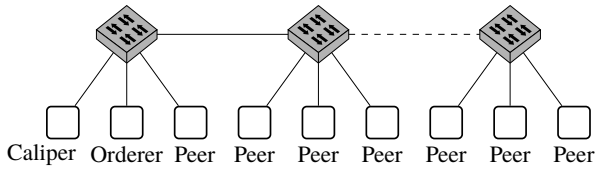

Figure 6: Topology used for the performance evaluation

B. Workloads and chaincodes

As mentioned before, different applications can induce different workloads in the DLT network. As we are not interested in measuring the performance of a particular DLT-based application, we dissect individual properties of complex workloads to define two “elementary” workloads that our test framework can execute reproducibly.

The *DoNothing* workload is the most simple chaincode that still allows a transaction to take place, i.e., no data is read or written and the transaction validation logic is emulated with a simple `return true` statement.

The *Simple* workload is used to emulate a simple banking application, where accounts are either opened, closed, or transfer of money happens between accounts. Each action in this workload induces a single change in the distributed ledger. Unless specified otherwise, we use *Simple* chaincode for the different measurements, since each transaction induces a change in the ledger.

C. Influence of transaction rate

To obtain an understanding of the basic behavior of Hyperledger Fabric, we first conducted an experiment with 10 peers and without altering the performance of the network links. In

each round of this experiment, transactions were executed at different rate for a duration of 10 s in total.

In order to evaluate the maximum workload that our experimental environment is able to process, we first measure how many transactions per seconds the system is able to process. For that we gradually increase the submitted transaction load until the system reaches a bottleneck. Figure 7 illustrates that using the default configuration provided by Hyperledger Fabric, the system is able to process a maximum of 275 transactions per seconds. Further evaluations of the cause of this bottleneck are conducted in Section VII-G.

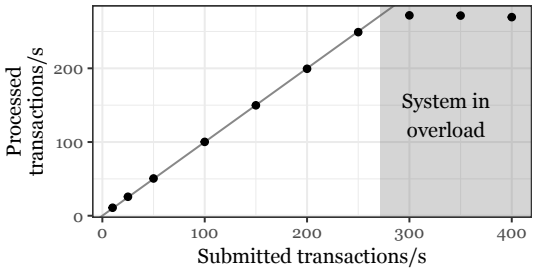

Figure 7: Maximal transactions processed with the default configuration of Hyperledger Fabric. The line represent an ideal system able to process the submitted rate without bottleneck.

Based on those results, we evaluated the influence of submitted transaction rate on transaction latency, where we gradually increased the transaction rate starting from 10 up to 250 transactions per second. Figure 8 illustrates timings of the three phases of the Hyperledger Fabric transaction flow at different transaction rates. The measurement values were created by timing different communication events as explained in Sections II-A1 and IV and illustrated in Figure 1.

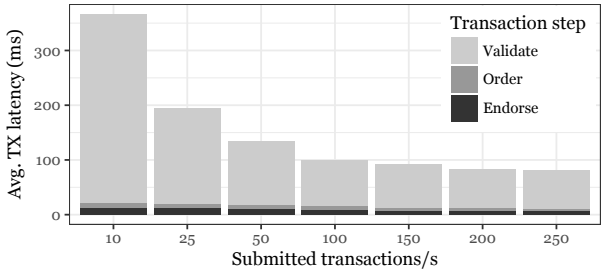

Figure 8: Transaction rate against latency of Hyperledger Fabric communication events

Figure 8 illustrates that the main contributor to transaction latency is the validation step. As the submitted transaction rate increases, we notice that the transaction latency decreases, demonstrating the bulk service of the validation step, as explained in Section IV and illustrated in Figure 3.

D. Influence of chaincode

As explained in Section VII-B, we evaluate here the differences between the two described chaincodes and compare the measured performance against the model presented in Section IV. Results are presented in Figure 9. As expected, the *Simple* chaincode results in larger validation latencies compared to the *DoNothing*, since it requires changes in the distributed ledger. Compared to the measurements, we note that the model predicts the validation latency with good accuracy.

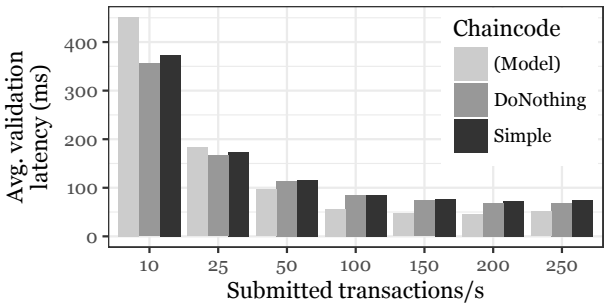

Figure 9: Comparison of the validation latency of the two evaluated chaincodes

E. Influence of network properties

In this section we evaluate the influence of network parameters on the transaction latency. We first evaluate the influence of network latency, by adding additional latency on the different hosts. This network delay was emulated via `netem`, the network emulation functionality of Linux.

Figure 10 shows the increase of transaction latency when the network latency is growing from 0 ms to 200 ms by 20 ms intervals. As expected, increases in network latency result in increased transaction latency. As demonstrated in Figure 8, the validation step is the main contributor to the transaction latency, illustrating that the validation step requires multiple network interactions and round trips.

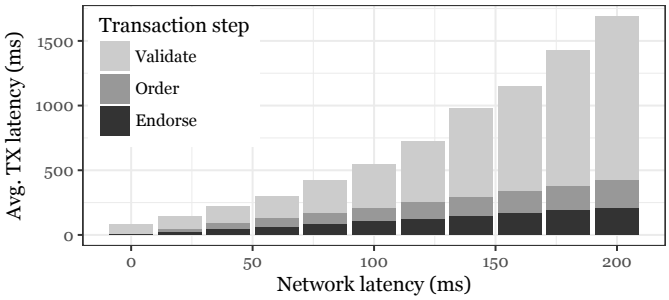

Figure 10: Transaction time against network latency

Secondly, we evaluate the influence of network loss by simulating packet losses on the different hosts in the network. Those packet losses were emulated via the statistics module of `iptables`, enabling us – as opposed to `netem` – to generate losses only on the relevant network packets for transactions and avoid potential shortcomings of Caliper. Figures 11 and 12 show the increase of transaction latency when the loss rate is growing from 0 % to 5 % by 0.5 % intervals. Compared to the Endorse and Validation steps detailed in Figure 12, we notice in Figure 11 that the network loss has a large influence

on validation time, going up to 40 s in the worst-case of 5 % packet loss. This is explained by larger messages required by the validation step which lead to more delays due to more occurrences of TCP retransmissions.

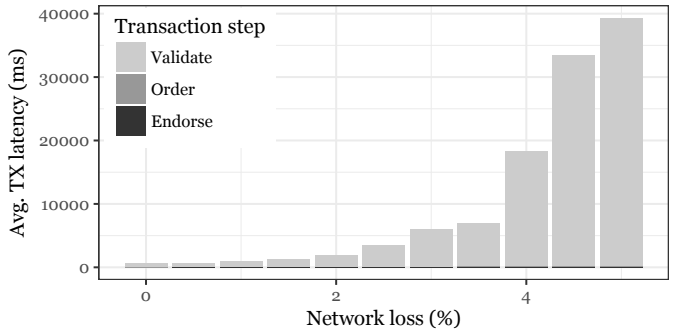

Figure 11: Transaction time against network loss. Details for the Endorse and Order steps are presented in Figure 12

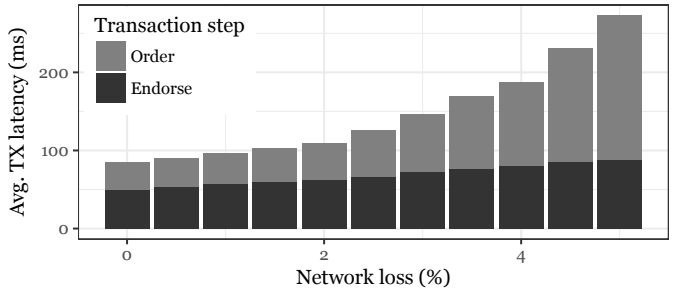

Figure 12: Details of the Endorse and Order steps of the transaction time presented in Figure 11

The findings presented in Figures 10 to 12 illustrate the importance of correctly dimensioning the network on which the DLT is running in order to minimize the transaction latency. This is especially relevant in industrial use-cases, where timing and deadline requirements have to be met in order for various processes to run correctly.

F. Influence of local network impairment

Based on the previous findings, we evaluate here the use-case where organizations participating in the DLT have heterogeneous connectivity to the orderer. This scenario is illustrated in Figure 13 with two organizations, where at least one peer of each organization is an endorsing peer.

This scenario is relevant in industrial use-cases where multiple shareholders use the same DLT to manage various collaborations. Since the shareholders may have different network providers, it is important to understand if the poor connectivity of one shareholder can influence the global performance of the DLT.

In Figures 14 and 15 we illustrate the following three cases:

- *No impairment*: both organizations have access to the network without emulated impairment.

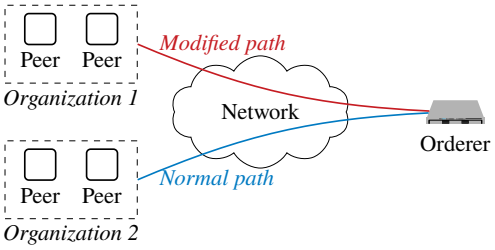

Figure 13: Illustration of simulated scenario where the network path of one organization is manipulated

- *One organization with impairment*: one of the two organizations participating in the DLT is affected either by increased delay (80 ms) or increased packet loss (2.5 %).
- *Both organizations with impairment*: both organizations are affected by the same increased delay or packet loss.

Since the Endorse step of a transaction requires the participation of all endorsement nodes in the DLT, we note in Figure 14 that its duration is the same for the two use-cases where additional delay is emulated. This illustrates that one organization with high network latency may actually slow down the performance of another organization with normal network latency. A similar behavior is illustrated in Figure 15 in case of network loss.

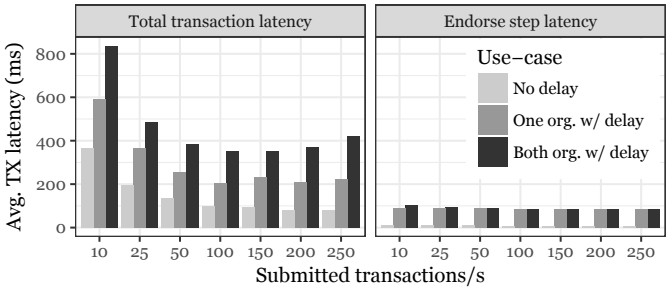

Figure 14: Transaction time against 80 ms network latency in the use-case presented in Figure 13

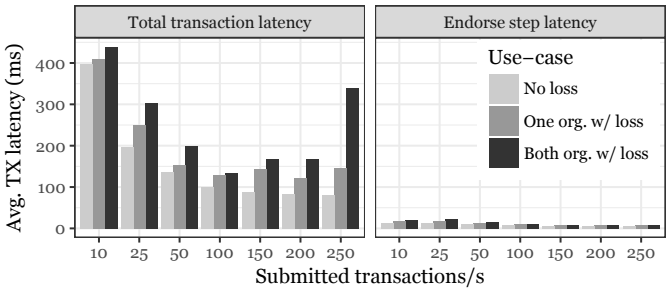

Figure 15: Transaction time against 2.5 % network loss in the use-case presented in Figure 13

We note that the effects of network latency and loss could also be used by a fraudulent stakeholder for intentionally slowing down particular transactions. This, in turn, can potentially cause invalid and failed transactions. Such manipulation of

network properties can lead to potential security attacks and denial of service of the DLT. A concrete attack has also been recently illustrated by Apostolaki et al. [20] in case of the Bitcoin network, where manipulation of the Internet routes of some users lead to delayed transactions of up to 20 min.

G. Influence of block size

We evaluate in this section the main bottleneck in our evaluated test bed, namely the ordering service. In order to better understand the role of the orderer, we evaluated different block size values and measured the rate at which transactions could be processed. Results are presented in Figure 16.

While we notice a linear trend between block size and rate of processed transaction for block sizes below 20, we notice that a bottleneck is reached for block sizes larger than 25.

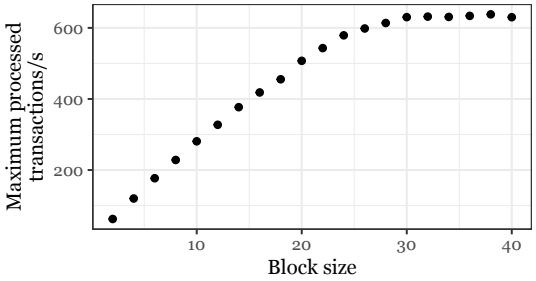

Figure 16: Influence of block size on maximal rate of transactions processed

VIII. Conclusion

In this paper we evaluated the performance of Hyperledger Fabric, a blockchain-based distributed ledger. We identified various bottlenecks which may impact its performance. We focused mainly on two metrics for evaluating this system: rate of processed transaction and transaction latency, two metrics which are especially relevant in industrial use-cases where critical latency requirements have to be met.

After an explanation of the DLT system and the methodology used for evaluating it, we propose a simple queuing model based on a bulk service. Via measurements made in our test bed, we evaluated the DLT system in various use-cases. We focused especially on the influence of communication network properties on the performance, namely link latency and packet loss. We also demonstrate how the network properties of one participant can influence the global performance of the DLT system, leading to potential attacks against some transactions.

In future work, we want to evaluate our system using a larger test bed where we plan to utilize virtualization to upscale the amount of hosts. In this environment, we also plan to conduct experiments with other ordering services than Solo, namely the Kafka-based orderer and – if available – the PBFT orderer. We want to understand how these distributed orderer implementations are influenced by network effects and in turn influence the performance of the DLT. A further interest of ours is to get more insights in potential security problems caused by delaying transactions; a threat that we expect to be even more pronounced when distributed orderer services are employed.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008, [Online] https://bitcoin.org/bitcoin.pdf, last accessed on October 15, 2019.

[2] E. Foundation, "A Next-Generation Smart Contract and Androulaki2018Decentralized Application Platform," 2018, [Online] https://github.com/ethereum/wiki/wiki/White-Paper, last accessed on October 15, 2019.

[3] 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 *Proc. of the 13th EuroSys Conf.*, Apr. 2018.

[4] M. Vukolić, "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication," in *Proc. of the International Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.

[5] The Linux Foundation, "Hyperledger Fabric," 2018, [Online] https://hyperledger.org/projects/fabric/, last access October 15, 2019.

[6] ——, "Measuring Blockchain Performance with Hyperledger Caliper," 2018, [Online] https://www.hyperledger.org/blog/2018/03/19/measuring-blockchain-performance-with-hyperledger-caliper, last access October 15, 2019.

[7] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "BLOCKBENCH: A Framework for Analyzing Private Blockchains," in *Proc. of SIGMOD*, 2017, pp. 1085–1100.

[8] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 7, pp. 1366–1385, 2018.

[9] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, "Performance Analysis of Private Blockchain Platforms in Varying Workloads," in *Proc. of the 26th Int. Conf. on Computer Communication and Networks (ICCCN)*, Jul. 2017, pp. 1–6.

[10] R. Yasaweerasinghelage, M. Staples, and I. Weber, "Predicting Latency of Blockchain-Based Systems Using Architectural Modelling and Simulation," in *Proc. of the 2017 IEEE Int. Conf. on Software Architecture (ICSA)*, 2017, pp. 253–256.

[11] W. Shbair, M. G. Steichen, J. François, and R. State, "Blockchain Orchestration and Experimentation Framework: A Case Study of KYC," in *Proc. of the 2018 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2018, pp. 1–6.

[12] P. Thakkar, S. Nathan, and B. Vishwanathan, "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform," 2018.

[13] M. Selimi, A. R. Kabbinale, A. Ali, L. Navarro, and A. Sathiaseelan, "Towards Blockchain-enabled Wireless Mesh Networks," 2018.

[14] J. Göbel, H. Keeler, A. Krzesinski, and P. Taylor, "Bitcoin Blockchain Dynamics: the Selfish-Mine Strategy in the Presence of Propagation Delay," *Performance Evaluation*, vol. 104, pp. 23–41, Oct. 2016.

[15] Y. Kawase and S. Kasahara, "Transaction-Confirmation Time for Bitcoin: A Queueing Analytical Approach to Blockchain Mechanism," in *Queueing Theory and Network Applications*. Springer International Publishing, 2017, pp. 75–88.

[16] M. L. Chaudhry and J. G. C. Templeton, *A First Course in Bulk Queues*. Wiley New York, 1983.

[17] J. Medhi, "Waiting Time Distribution in a Poisson Queue with a General Bulk Service Rule," *Management Science*, vol. 21, no. 7, Mar. 1975.

[18] S. Gallenmüller, D. Scholz, F. Wohlfart, Q. Scheitle, P. Emmerich, and G. Carle, "High-Performance Packet Processing and Measurements (Invited Paper)," in *Proc. of the 10th Int. Conf. on Communication Systems & Networks (COMSNETS 2018)*, Jan. 2018.

[19] Ansible Inc. / Red Hat Inc., "Ansible," 2018, [Online] https://www.ansible.com/, last access October 15, 2019.

[20] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies," in *Proc. of the IEEE Symp. on Security and Privacy (SP)*, 2017.