Proposal of a Permissioned Blockchain Network To Supervise Cashflow in Large-Scale Projects

Lukas Schöbel
Technical University of Munich
Garching bei München, Germany
lukas.schoebel@tum.de

Alex Kulikov
Technical University of Munich
Garching bei München, Germany
alex.kulikov@tum.de

ABSTRACT

Modern Blockchain technologies offer high potential to ensure transparency and security for transactions of money and goods among business partners. The aim of this work is to elaborate on the history, potential use cases and the technical properties of Hyperledger Fabric as an open-source project. Additionally, we propose a prototype implementation of a blockchain-based legal agreements storage built on top of Hyperledger Fabric. In particular, the suggested approach allows for the convenient supervision of cashflow within large-scale projects as well as the detection of potential cash leaks and criminal activity. In this work, we explain the motivation behind the suggested application as well as the real-life impact it can provide in projects with high expenditure. In this documentation, we present our implementation approach, encountered issues during setup and want to conclude with an outlook on Hyperledger Fabric and on our project in particular.

KEYWORDS

Hyperledger Fabric, Hyperledger, Permissioned, Blockchain, Smart Contract, Legal Agreement, Cashflow

1 INTRODUCTION: HYPERLEDGER FABRIC

In this first introductory chapter, we want to provide a brief overview about the original Hyperledger project and its history as well as to elaborate on potential use cases for the Hyperledger Fabric subproject.

1.1 History of Hyperledger and Partners

With the intention of developing industrial large-scale Blockchain applications, the Hyperledger project was initially founded by 30 companies in 2016 [17]. The initial founding members, companies as i.a. IBM, Cisco, J.P. Morgan, Deutsche Börse Group, set up the Hyperledger project to enforce trust, accountability and full transparency among business partners [24]. Today, over 250 parties from varying industries are involved in the development of the venture which is hosted by the Linux Foundation [25].

The very fast developing open source project utilizes a distributed ledger technology which ensures a transparent, decentralized, and shared cross-industry open standard. Having a high potential at the interface of various industries, the drawback of Blockchain technology is that there is no so-called one-size-fits-all approach. In order to find a solution to this problem, Hyperledger understands itself as a greenhouse for open source Blockchain development that unites companies and developers with the appropriate technologies [6, p. 8]. Currently, Hyperledger Fabric is only one out of 15 subprojects within Hyperledger and can be understood as a permissioned network with decentralized trust, where the parties know each other [19]. Based on a very active community, the current Hyperledger Fabric version supports different programming languages and offers a modular toolbox with basic building blocks. This enables rapid implementation of architectures which can be highly specialized and specifically designed for a given use case.

1.2 Classification and Differentiation from other Blockchain Systems

Blockchain systems can be *permissionless* or *permissioned*. In *permissionless* blockchain systems, anyone can join and participate in the network freely. *Permissioned* blockchains, on the other hand, have strict policies, determining which parties are allowed to join the network, and which permissions and privileges are assigned to them.

Hyperledger Fabric is designed as a permissioned platform to set up private and consortium Blockchain networks with modular consensus that is not hard-coded in contrast to most other blockchain systems [26]. Fabric does not include a cryptocurrency by default, as it is in most cases not relevant for business applications, and would create unnecessary performance overhead at transaction processing. The project was specifically created for B2B applications and is - as a permissioned Blockchain - able to utilize the traditional Byzantine-fault-tolerant consensus since the participants are identified [2]. Because the validation of transactions in permissioned blockchain systems is conducted by a subset of verified peers, private systems tend to have a higher throughput. Referring to Dinh et al. [13], Hyperledger Fabric is currently the fastest private open-source blockchain platform. Recent work examined that Fabric 1.2. is able to execute around 3,000 transactions per second *(tps)*, and that it is possible to even achieve 20,000 tps by restructuring the underlying blockchain architecture [15]. To differentiate Fabric from other common blockchain solutions, Fig. 1 shows differences and similarities between *Hyperledger Fabric*, *Bitcoin*, *Ethereum* and *R3 Corda*.

1.3 Potential Use Cases of Fabric

As a permissioned blockchain, Hyperledger Fabric can be utilized where privacy, transparency as well as the need for efficient transaction processing are paramount. This makes Fabric especially interesting for high-scaling finance, manufacturing, banking, IoT

Advanced Blockchain Technologies, IN2107, February 2020

	Bitcoin	Ethereum	**Hyperledger Fabric**	R3 Corda
Business Area	cryptocurrency	cryptocurrency, B2C	**B2B**	B2B
Type	public, permissionless	public, permissionless	**private, permissioned**	private, permissioned
Contracts	no smart contracts	smart contracts e.g. with Solidity	**smart contracts e.g. with Java, Javascript**	smart contracts e.g. with Kotlin, Java
Currency	Bitcoin	Ether	**none**	none

Figure 1: Comparison of Hyperledger Fabric to Bitcoin, Ethereum, and R3 Corda.

and insurance applications with high security standards [19]. In general, Fabric allows to tackle broad supply or production chain management problems with partners from multiple industries or locations.
Shortly after the Hyperledger project was launched, IBM started a pilot project with Walmart using Fabric to trace food and to guarantee transparency throughout the entire supply chain. While Walmart aims to detect bacteria or defects, their customers are able to track products back to the manufacturer [1]. Potentially saving lives, the traceability of food is especially important when it needs to be recalled due to production and processing defects.
Another interesting project from Ichikawa et al. [20] features the implementation of a digital passport for personal health data. Introducing an increased level of security while providing an accessible system at the same time, the Japanese researchers introduced a mobile application that exchanges health data by connecting to a private Hyperledger Fabric blockchain network.

2 TECHNICAL PROPERTIES OF HYPERLEDGER FABRIC

In order to get a deeper understanding of Hyperledger Fabric, it is essential to characterize its architecture as well as to outline its advantages and disadvantages as a permissioned, distributed ledger framework.

2.1 Architecture of Hyperledger Fabric

2.1.1 Consensus Protocols. Fabric includes different consensus protocols which allow to adjust the technology to a specific use case and trust model [6, p. 23]. Further, the platform provides either crash-fault-tolerant (CFT) or byzantine-fault-tolerant (BFT) [10] ordering. While the first one utilizes the *Raft* protocol [23], the latter one uses the *Apache Kafka* ordering service [4], which in its turn utilizes *ZooKeeper* [3] internally. By default, the byzantine-fault-tolerance consensus protocol is used for ordering [8, p.2].

2.1.2 Membership Service Provider. New nodes are included into the network by using a "Membership Service Provider" (MSP), which is the identity management solution used in Fabric [22, p. 31]. The MSP provides a pluggable API, which supports several different certificate authorities (CA). One of the commonly used external certificate authorities is TLS-CA. Through this CA, the identity of clients connecting to the system is verified over the HTTPS protocol. Additionally, Hyperledger Fabric provides its own certificate authority. It is called Fabric-CA, and is utilized by default in Fabric applications. With such a certificate authority, it is ensured that only the nodes which were previously authorized to connect to the network can actually do it.

2.1.3 Channels. The MSP also makes sure that only authorized clients can join certain channels and are able to execute chaincode. Within Fabric, Channels are a core feature and are responsible for establishing private and secure connections between peers belonging to the same group [2, p. 5]. Every such channel has a ledger associated with it, of which every connected peer maintains its own copy. This ledger stores all transactions which took place on this specific channel since its creation. Which channels can be accessed by which nodes is managed by the MSP, and hence the transactions within the channel cannot be seen from outside the channel. Each of the clients can participate on multiple channels at the same time (Fig. 2), which is frequently the case in real life. For instance, it is considered common practice for one employee to participate in multiple working groups within their company.

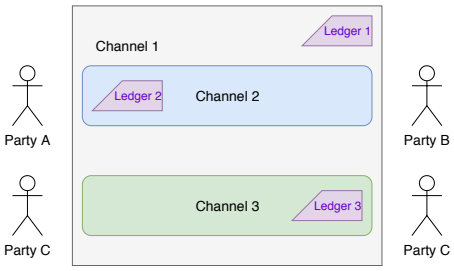

Figure 2: Channels in Hyperledger Fabric

An additional benefit of the strict separation of channels is the possibility to execute transactions concurrently whenever they take place on different channels. This is a core difference to some other Blockchain systems - like e.g. Bitcoin - and provides a significant boost to performance and scalability [21, p. 21].

2.1.4 Ledger. For its ledger, Fabric utilizes an append-only distributed physical storage [2, pp. 4]. As in most other blockchain-based systems, the ledger records all the incoming transactions by adding blocks to its existing data chain. These blocks are connected via hash pointers to ensure data integrity and immutability, and are replicated across the system to prevent data loss. The replication happens naturally, as each of the peers involved in a channel maintains its own copy of the ledger associated with this channel. The block storage of the ledger provides multiple built-in index structures which make random access queries and transaction search particularly convenient in Fabric. The physical storage behind the ledger is provided with a local key-value store based on either Apache's *CouchDB* [5] or on Google's *LevelDB* [9]. The choice of database depends on the particular application use case, as well as on the choice of programming languages.

2.1.5 Node Types. There are three different types of nodes (*peers*) in Fabric: endorsing peers, committing peers, and ordering service peers [2, p. 5].

Endorsing peers are responsible for executing proposed transactions and thus creating chaincode that can be deployed on the network. If the execution goes well, the peer sends an endorsement back to the transaction issuer. An endorsing peer is also always a committing peer.
Committing peers are responsible for validating the results of a transaction and for deploying (*committing*) it to the network.
Ordering service peers are there to prevent race conditions between transactions. They order transactions according to some earlier specified algorithm, so that all transactions are committed sequentially in one batch. Ordering service nodes are only there for the ordering, i.e. they do not maintain a copy of the ledger themselves. The most commonly used ordering service utilizes *Apache Kafka*, as mentioned in 2.1.1, which can efficiently process streams of records ordering them into blocks.

2.1.6 Transaction Processing. Whenever a transaction is created on one of the channels in Fabric, it has to follow a pre-defined process in order to be included in the network. A visualization of these steps can be seen in Fig. 3 and bases on [2, p. 6] and [21, pp. 11]:

(1) A node or application submits a transaction proposal to the network.
(2) The endorser peers test-execute the chaincode involved in the transaction.
(3) If the execution did not cause any problems, an endorsed response is returned to the transaction issuer.
(4) The transaction issuer submits the transaction to an ordering service node.
(5) The ordering service node orders this and other current transactions according to some ordering schema.
(6) A batch of ordered transactions is sent to committing nodes.
(7) The committing nodes validate the transactions and commit them to the ledger.
(8) A transaction commitment event is issued on the network [1].

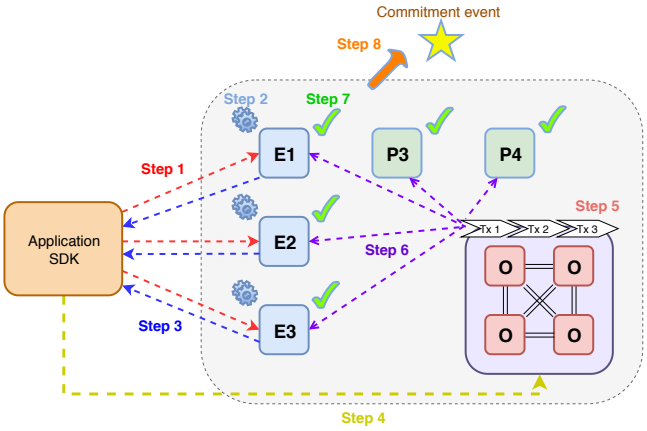

Figure 3: Transaction Processing in Fabric

In order to keep all nodes up-to-date with the current ledger state, a so-called *state transfer* is performed whenever a new transaction has been committed [2, p. 9]. For this purpose, Fabric makes use of a gossiping protocol called *Fabric gossip*, which is based on the epidemic multicast algorithm as explained in [12]. The updated nodes can easily verify that the received state data is legitimate by checking the signature of the ordering service on each of the blocks.

2.1.7 Chaincode. Hyperledger Fabric provides the basic framework for Blockchain applications, and container technology to host smart contracts which are called *chaincode*. Within chaincode, the business logic behind the blockchain network can be defined. In its current version v. 1.6, Fabric provides chaincode SDKs for general-purpose programming languages such as Go, Node.js and Java, and offers two application SDKs for Node.js and Java. For the latter, there are also unofficially released SDKs for Python and Go [18] [27].

2.1.8 Applications. Applications based on Hyperledger Fabric are somewhat similar to Ethereum's *dApps* (decentralized applications). Applications are programs that interact with the blockchain system by making calls on smart contracts [14]. As such, some applications provide a rudimentary interface, like for instance a range of console commands, to conveniently interact with the chaincode. Others include a graphical user interface (GUI), which makes them more user-friendly, and thus facilitates deployment of transactions according to some business logic. For smaller proof-of-concept applications, which aim to demonstrate the basic capabilities of Fabric, a REST API can be the interface of choice. This allows the user to interact with the ledger through a range of read and write requests.

2.2 Critical Discussion of Fabric

As of date, Hyperledger Fabric maintains a prominent position among existing Blockchain projects. In the following, we briefly discuss some of its strengths and weaknesses.

2.2.1 Advantages. Hyperledger Fabric features a very modular architecture with its main components being fully pluggable. It supports customizable solutions for ordering, membership, endorsement, gossiping, data storage, and more [2]. Additionally, it supports chaincode and application development in multiple programming languages and runs smart contracts in isolated *Docker* [2] containers. All this enables simple integration into existing business applications, as well as the implementation of complex business logic in a flexible way [11]. Furthermore, the physical separation of sensitive data through private channels and identity management makes sure that only those authorized may access critical data. Taking into account Fabric's better scalability and transaction throughput than most traditional Blockchain solutions, it is also more likely to be chosen over conventional distributed database systems in certain business models.

2.2.2 Disadvantages and limitations. Despite all its advantages, there are also negative aspects to mention about Hyperledger Fabric. As the project is still very new, it is prone to missing documentation, bugs, and a lack of referable examples. Due to new features being

[1] Applications, which will be introduced a bit later, can subscribe to certain events and react to them with some predefined functionality.

[2] Docker is a platform providing containerization of applications. Official online resource: https://www.docker.com/, last accessed 2 Feb. 2020.

added rapidly and old ones getting deprecated, the documentation and official resources of the project cannot keep up with its current state [7]. This makes it hard for developers to implement and setup the system within their own business environment. Because of this, Fabric is currently mostly used by technology pioneers and within R&D projects, but only rarely in production environments.
Furthermore, just like the other existing blockchain systems, Fabric somewhat suffers from the lack of use cases, where it would be provenly better than a traditional distributed DBMS [7]. As the latter still offer higher transaction rates than Blockchain-based storages, it is potentially hard to compete with them despite the advantages that Fabric provides.

3 THE CASHFLOW USE CASE

In this chapter, we introduce the motivation behind our Cashflow use case, as well as the implementation approach of the corresponding prototype.

3.1 Motivation and Introduction of Use Case

In today's world, countless large construction projects are designed, planned and executed every year. Carrying out such projects often involves very high organizational effort from the organizing entity and a variety of contractors who work on the construction site. In the general case, a construction company sets up a legal agreement with a contractor to capture the details of the business deal. In this work, we want to propose an alternative system to the legal agreements on paper to guarantee temporal and financial efficiency. Based on Blockchain technology, the proposed model can be utilized to trace private or public legal agreements between multiple parties. This way, it is possible to keep track of the money flow, and of further important characteristics, such as the current state of the project, or its budget. Being a redundant system to actual paper contracts, we believe that this solution could help to prevent the embezzlement of project funds, intentional delays of work or even corruption due to the increased degree of transparency.

Hyperledger Fabric allows the integration of different user roles with diverse user rights and as shown in Figure 4, we have incorporated three user roles in our project that occur in every building project. Starting from the top, an *Authority* - e.g. a state - generally commissions the construction of the building and provides the necessary budget. Within the Blockchain network, the authority is analogous to the "eyes" of the building project, since it only has read access and can oversee the contracts, but not interfere with the legal agreements directly. In contrast, the *Organizer* - e.g. a construction company - corresponds to the "brains" behind the construction which is responsible for i.a. querying, setting up, and signing legal agreements between itself and a given contractor. The organizer is provided with a budget by the authority and spends these funds on services. Further, this entity has all the necessary read and write permissions to submit transactions to the Blockchain. In our senses analogy, *Contractors* - e.g. a painter or architect - are the "hands" of the project and provide their services to advance the construction site. As a user role, they have access rights which are similar to the Organizer with the only difference that they are only able to get legal agreements in which they are involved themselves. In

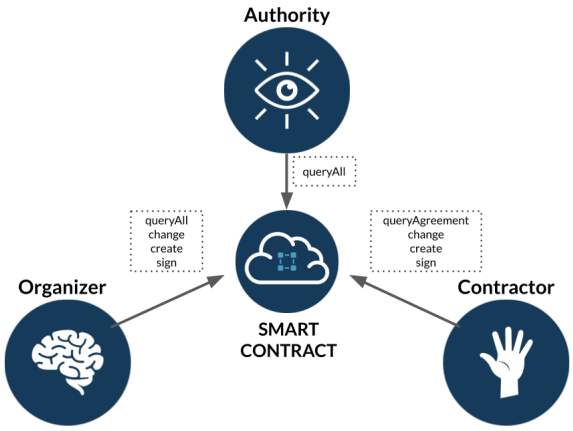

Figure 4: Visualization of Cashflow Use Case

the proposed network, contractors have the least amount of access rights and are the only entity which can exist multiple times.

As a paradigm of a very large and lengthy construction site in Germany, we have focused on the new airport of Berlin within the implementation of our prototype. When the construction began in 2006, the costs until opening in October 2011 were estimated to be 2 Billion Euros. Due to multiple scandals and several delays, the infrastructural building project is yet to be finished. The latest numbers suggest a preliminary opening date of October 2020 and estimated total costs of 6.4 Billion Euros [16]. Not explicitly suspecting any criminal activity or intended delays during the construction of the airport BER, we nevertheless want to note that the cashflow in large building projects can be complex and difficult to comprehend. Therefore, we want to suggest a solution to ease this process, increase the transparency and provide a starting point for in-depth auditioning.

3.2 Implementation

In the following, the solution approach and the implementation of our cashflow use case is explained in detail.

3.2.1 Setup. Our cashflow prototype was developed using Hyperledger Fabric version 1.4 and its JavaScript-based SDK. The operating system we used to set up the blockchain was *MAC OS 10.15 Catalina*. The setup of Hyperledger Fabric requires the developer to install the following programming languages and tools with the given versions:

- *Go* – version 1.13.4
- *Node.js* – version 10.17.0, including *NPM* – version 6.11.3
- *Python 2* – version 2.7
- *Docker* – version 18.03.1, including *Docker Compose* – version 1.21.0
- *XCode* – when working in MAC OS environment
- *Homebrew* – when working in MAC OS environment
- The tools *curl* and *wget*

Additionally, the binaries and images needed to run our Fabric example have to be downloaded by switching to the project directory and executing …

```
curl -sSL http://bit.ly/2ysbOFE | bash -s.
```

It is important to note that Fabric is still in active development and is hence very sensitive to version changes of the tools it requires. Therefore, it is advisable to always use the versions recommended in its manual, or the ones listed in this documentation.
Finally, the entire code required to run our Cashflow prototype needs to be downloaded from our *GitHub* repository[3].

3.2.2 Approach. The general approach we used to implement our Cashflow use case is as following:

(1) Take the official *fabric-samples* repository from the project's developers, and use its *fabcar* example as a starting point for our own implementation.
(2) Give the downloaded samples a test run to ensure that everything works as intended.
(3) Implement our own logic for the concepts of user registration, chaincode transactions, and the application API.
(4) Make sure that the prototype architecture reflects our use case, and remove unused elements of the original samples.

3.2.3 Starting up Fabric. In order to first start up the network, the *startFabric.sh* script, which finds itself in our *cashflow* folder, needs to be executed.

```
sh startFabric.sh
```

The script creates a small blockchain network, comprised of several peers, an orderer node, and a certificate authority. For each of these elements it starts up an own Docker container to ensure that the blockchain components are initially isolated from each other. Additionally, the script installs a JavaScript version of the smart contract on each of the blockchain nodes.
The created docker containers can easily be checked by running the `docker ps` command. It should show up the docker containers which have been set up.
After running the script, the node application dependencies now need to be installed in order to be able to interact with the instantiated ledger [14]. The following command installs all application dependencies defined in the *package.json* file …

```
npm install
```

Including the smart contract besides other dependencies, this installation is essential to make use of identities, user wallets, channel gateways as well as to submit transactions to the network. As soon as `npm install` completes, everything is ready to go.

3.2.4 Enrolling admin user. Once the blockchain network has been set up, we can now enroll our first admin user. This is done by executing the *enrollAdmin.js* file with

```
node enrollAdmin.js
```

In particular, the command creates a wallet for the admin user, which is comprised of a private and a public key, as well as a X.509 certificate [14]. Afterwards, the admin user will be responsible for the entire administration of the Blockchain network.

[3]Official GitHub repository of our Cashflow project: https://github.com/lukaschoebel/cashflow.

3.2.5 Registering users. While the admin user will be managing the system, we also need several ordinary users to interact with the ledger in the scope of our Cashflow use case. For this purpose, we create three users, who will fulfill the three different roles defined in our use case description in 3.1. In particular, we create the three users shown in Table 1. In order to register the users with their respective user roles, the following command needs to be executed:

User Name	User Role	User Permissions
State Agency	Authority	queryAll
Construction Company	Organizer	query, queryAll, create, sign, change
Architect	Contractor	query, create, sign, change

Table 1: Users to register with their respective roles and permissions.

```
node registerUser.js -a State\ Agency -o Construction\
↪  Company -c Architect
```

Here, the flags $-a$, $-o$, and $-c$, signify the user roles *authority*, *organizer*, and *contractor*, respectively. The argument after each flag gives the name of the user with the specified user role.

3.2.6 Creating the smart contract. After registering the users, the client can interact with the ledger by using transactions, which were predefined in the smart contract. For our Cashflow use case, we created the smart contract called *cashflow.js*, which encompasses all the functionality that is required for our proof-of-concept. At the beginning of the contract, there is a function called *initLedger*. This function is different from other transactions defined in the contract in that it gets called automatically each time the chaincode of this contract is deployed onto the Blockchain. This way, it can be conveniently used to predefine some original data stored on the ledger. We make use of it by creating three legal agreements between our Construction Company and some contractors, which have already been signed.
Defining the core functionality of the proposed network, the following transactions can be executed on the ledger …

- queryAgreement
- queryAll
- createAgreement
- signAgreement
- changeAgreement

Each of the transactions takes its required attributes as arguments. Also, depending on the user that issued the transaction call, it is being determined whether this user is allowed to execute this particular transaction. The access control is required to ensure that users can only execute transactions to which they are authorized, as previously explained in 3.2.5. A user with the role *contractor*, for instance, is not allowed to execute the *queryAll* transaction to gain insight into all the legal agreements on the Blockchain.

3.2.7 Creating the application. Having created the smart contract, the only piece missing to be able to interact with the ledger is the application interface. This is implemented in the *query.js* JavaScript file, which can also be found in our GitHub repository. In its core, *query.js* takes as arguments the role of the user that wants to

execute some transaction, the name of the transaction, and its parameters. These arguments are then extracted from the command line, and the corresponding transaction is called on the chaincode. At this point, two different variants are available to call a transaction. The first one is the function *evaluateTransaction*, which is used for read-only requests on the ledger. The second one is *submitTransaction*, and has to be called whenever write requests on the blockchain need to be executed. Therefore, we are choosing one of these two functions to be called on the smart contract depending on the user input. We "evaluate" the transactions *query* and *queryAll* and "submit" the transactions *create*, *sign*, and *change*.

3.2.8 Interacting with the blockchain. With all the setup and implementation being done, the Blockchain system can now be interacted with by issuing transactions to it from the command line. Assume the following example:

```
node query.js organizer create LAG4 52ABC1042 10M
↪ Construction\ Company Architect
node query.js contractor sign LAG4
node query.js contractor query LAG4
node query.js authority queryAll
```

In this example our organizer Construction Company creates and automatically signs a new legal agreement with the following parameters:

- id: "LAG4"
- hash: "542ABC1042"
- cash amount: "10M"
- partner_1: "Construction Company"
- partner_2: "Architect"

After that, the legal agreement "LAG4" also gets signed by the contractor Architect, who then retrieves the agreement to check whether the signature is already there.
Finally, the regulation authority queries all the legal agreements deployed on the network to trace the flow of money within the construction project.

3.3 Advantages and Limitations

The proposed prototype is able to make cash flow in large-scale building projects more transparent since the legal agreements and the included transactions can be traced. By introducing an increased degree of traceability, the proof-of-concept provides a first starting point for in-depth auditioning and enables to detect cash leaks or suspicious manipulations.
However, the system should be seen as a redundant alternative to the legal agreements on paper and cannot be utilized to prevent criminal activity all by itself. In addition, it needs to be noted that the suggested Fabric prototype will not work sufficiently if its environment and all included entities are entirely corrupt.

4 CONCLUSION

In this last section, we focus on the findings and the outlook of Hyperledger Fabric, and of our prototype in particular.

4.1 Summary

Fabric is a powerful state-of-the-art blockchain project, which features a modular and flexible architecture, and is suitable for various business applications. In this work, we have proposed a solution based on this technology to trace the cashflow in large construction projects, and described the utilized frameworks and ideas. As a proof-of-concept, the prototype sets up a Fabric network, is able to register users and make according queries to e.g. create or obtain certain legal agreements. It ensures that money spending is kept transparent and traceable by storing legal agreements between project parties on the blockchain. Further, our implementation incorporates access control which only allows to execute functions if a given user has the required access rights. Despite Fabric being advertised as simple to build up, we encountered some difficulties during setup of the Fabric environment. Due to partially outdated documentation and incorrectly named source files, it is first necessary to fix existing bugs, and to get a deeper understanding of the Fabric infrastructure. However, in the end, we were able to successfully implement and test our cashflow prototype, which shows that Fabric can indeed be used to create and maintain blockchain-based solutions for business applications.

4.2 Outlook

Within the scope of this seminar, we introduced a first prototype that could make large-scale building projects more transparent and effective. To advance the development of the proposed system, the implementation could be improved by connecting the Fabric network to a graphical user interface to increase the usability. Also, at the moment, the prototype stores *hashes* of legal agreements in the blockchain network. In subsequent work, we believe that it would be more efficient to set up an adjacent database in which these hashes are mapped to digital copies of the corresponding legal agreements. In addition, it could also be interesting to exploit the event services of Hyperledger Fabric and issue and react to events if e.g. a contract is created by a certain entity. Lastly, the network should be tested in terms of performance and privacy.

ACKNOWLEDGMENTS

We are very grateful to Jeeta Chacko for her valuable comments on the documentation and presentation of this project.

REFERENCES

[1] Roger Aitken. 2017. IBM & Walmart Launching Blockchain Food Safety Alliance In China With Fortune 500's JD.com. https://www.forbes.com/sites/rogeraitken/2017/12/14/ibm-walmart-launching-blockchain-food-safety-alliance-in-china-with-fortune-500s-jd-com/{#}6e575fad7d9c
[2] Elli Androulaki, Artem Barger, Vita Bortnikov, Srinivasan Muralidharan, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Chet Murthy, Christopher Ferris, Gennady Laventman, Yacov Manevich, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *Proceedings of the 13th EuroSys Conference, EuroSys 2018* 2018-Janua (2018). https://doi.org/10.1145/3190508.3190538 arXiv:1801.10228
[3] Apache Software Foundation. [n.d.]. Apache ZooKeeper. https://zookeeper.apache.org/
[4] Apache Software Foundation. 2017. Apache Kafka - A distributed streaming platform. https://kafka.apache.org/
[5] Apache Software Foundation. 2020. Official CouchDB Resource. https://couchdb.apache.org/
[6] Tamas Blummer, Sean Bohan, Mic Bowman, Christian Cachin, Nick Gaski, and Nathan George. 2018. An Introduction to Hyperledger. (2018), 33 pages. https://doi.org/10.1007/978-1-4842-4847-8_8
[7] Bill Brock. 2018. The Pros and Cons of Hyperledger Fabric. https://www.verypossible.com/blog/the-pros-and-cons-of-hyperledger-fabric

[8] Christian Cachin. 2016. *Architecture of the Hyperledger Blockchain Fabric.* Technical Report. IBM Research, Zurich, Switzerland.
[9] Carnegie Mellon Database Group. 2020. Official LevelDB Resource. https://dbdb.io/db/leveldb
[10] Miguel Castro. 2001. Practical Byzantine Fault Tolerance. (2001).
[11] Aran Davies. 2019. Pros and Cons of Hyperledger Fabric. https://www.devteam.space/blog/pros-and-cons-of-hyperledger-fabric-for-blockchain-networks
[12] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. 1987. Epidemic algorithms for replicated database maintenance. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing* Part F1302 (1987), 1–12. https://doi.org/10.1145/41840.41841
[13] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian Lee Tan. 2017. BLOCKBENCH: A framework for analyzing private blockchains. *Proceedings of the ACM SIGMOD International Conference on Management of Data* Part F1277 (2017), 1085–1100. https://doi.org/10.1145/3035918.3064033 arXiv:1703.04057
[14] Nick Gaski, Robert P. J. Day, and Et al. 2019. Writing your first application. https://hyperledger-fabric.readthedocs.io/en/release-1.4/write{_}first{_}app.html
[15] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. 2019. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. (2019), 455–463. https://doi.org/10.1109/bloc.2019.8751452 arXiv:1901.00910
[16] Robert Hartl. 2020. Flughafe Berlin Kosten. https://www.flughafen-berlin-kosten.de/
[17] Hyperledger. 2019. About Hyperledger. https://www.hyperledger.org/about
[18] Hyperledger. 2019. Hyperledger Fabric Documentation. https://hyperledger-fabric.readthedocs.io/
[19] IBM. 2019. Hyperledger Fabric: the flexible blockchain framework that's changing the business world. https://www.ibm.com/blockchain/hyperledger
[20] Daisuke Ichikawa, Makiko Kashiyama, and Taro Ueno. 2017. Tamper-Resistant Mobile Health Using Blockchain Technology. *JMIR mHealth and uHealth* 5, 7 (2017), e111. https://doi.org/10.2196/mhealth.7938
[21] Florian Matthes. 2019. Hyperledger – Fabric.
[22] Florian Matthes. 2019. Hyperledger – Introduction.
[23] Diego Ongaro and John Ousterhout. 2014. *In Search of an Understandable Consensus Algorithm (Extended Version).* Technical Report. Stanford University.
[24] The Linux Foundation. 2016. Linux Foundation's Hyperledger Project Announces 30 Founding Members and Code Proposals To Advance Blockchain Technology. https://www.hyperledger.org/announcements/2016/02/09/linux-foundations-hyperledger-project-announces-30-founding-members-and-code-proposals-to-advance-blockchain-technology
[25] The Linux Foundation. 2018. Hyperledger Fabric. https://www.hyperledger.org/projects/fabric
[26] Marko Vukolic. 2017. Rethinking Permissioned Blockchains. *IBM Research* (2017), 3–7. https://doi.org/10.1145/3055518.3055526
[27] Kazuhiro Yamashita, Yoshihide Nomura, Ence Zhou, Bingfeng Pi, and Sun Jun. 2019. Potential Risks of Hyperledger Fabric Smart Contracts. *IWBOSE 2019 - 2019 IEEE 2nd International Workshop on Blockchain Oriented Software Engineering* (2019), 1–10. https://doi.org/10.1109/IWBOSE.2019.8666486