Frankfurt School Blockchain Center

JUNE 2017

FSBC Working Paper

Comparison of Ethereum, Hyperledger Fabric and Corda

Martin Valenta, Philipp Sandner

With this paper, we provide a brief analysis of the most notable differences between the distributed ledger technologies (DLT) Hyperledger Fabric, R3 Corda and Ethereum. Our intention is to give decision makers new to DLT guidance for what use cases Hyperledger Fabric, Corda and Ethereum are most suitable.

Three different frameworks

From the white papers of Hyperledger Fabric, R3 Corda (in the following only referred to as Fabric and Corda, respectively) and Ethereum it becomes obvious that these frameworks have very different visions in mind with respect to possible fields of application. Development of both Fabric[1] and Corda[2] is driven by concrete use cases, whereas Corda's use cases are drawn from the financial services industry. Consequently, this is where Corda sees its main field of application. In contrast, Fabric intends to provide a modular and extendable architecture that can be employed in various industries, from banking and healthcare over to supply chains. Ethereum also presents itself as utterly independent of any specific field of application.[3] However, in contrast to Fabric, it is not modularity that stands out but the provision of a generic platform for all kinds of transactions and applications. Table 1 provides a summary of the three frameworks.

Frankfurt School Blockchain Center
www.fs-blockchain.de
contact@fs-blockchain.de

Follow us
www.twitter.com/fsblockchain
www.facebook.de/fsblockchain

Frankfurt School of Finance & Management gGmbH
Sonnemannstrasse 9-11
60314 Frankfurt am Main
Germany

Table 1

Comparison of Ethereum, Hyperledger Fabric and Corda

Characteristic	Ethereum	Hyperledger Fabric	R3 Corda
Description of platform	– Generic blockchain platform	– Modular blockchain platform	– Specialized distributed ledger platform for financial industry
Governance	– Ethereum developers	– Linux Foundation	– R3
Mode of operation	– Permissionless, public or private[4]	– Permissioned, private	– Permissioned, private
Consensus	– Mining based on proof-of-work (PoW) – Ledger level	– Broad understanding of consensus that allows multiple approaches – Transaction level	– Specific understanding of consensus (i.e., notary nodes) – Transaction level
Smart contracts	– Smart contract code (e.g., Solidity)	– Smart contract code (e.g., Go, Java)	– Smart contract code (e.g., Kotlin, Java) – Smart legal contract (legal prose)
Currency	– Ether – Tokens via smart contract	– None – Currency and tokens via chaincode	– None

Participation of peers

With conventional central data storage, only a single entity, the owner, keeps a copy of the underlying database, e.g. a ledger. Consequently, this entity controls what data is contributed and what other entities are permitted to contribute. With the advent of DLT this radically changes in favor of distributed data storage where multiple entities hold a copy of the underlying database and are naturally permitted to contribute. All entities that participate in distributed data storage form a network of so-called *nodes* or *peers*. Due to distributed data storage, the difficulty arises to ensure that all nodes agree upon a common truth, e.g. the correctness of a ledger, as changes made by one node have to be propagated to all other peer nodes in the

network. The result of arriving at a common truth is called *consensus* among nodes and is described below.

With respect to participating to consensus, there are two modes of operation: *permissionless* and *permissioned*. If participation is permissionless, anybody is allowed to participate in the network. This mode is true for Ethereum as a public blockchain. On the other hand, if participation is permissioned, participants are selected in advance and access to the network is restricted to these only. This is true for Fabric and Corda. The mode of participation, permissionless or permissioned, has a profound impact on how consensus is reached.

Consensus

Ethereum. With Ethereum, all participants have to reach consensus over the order of all transactions that have taken place, irrespectively of whether a participant has taken part in a particular transaction or not. The order of the transactions is crucial for the consistent state of the ledger. If a definitive order of transactions cannot be established there is a chance that double-spends might have occurred, that is, two parallel transactions transfer the same coin to different recipients, thus making money out of thin air. As the network might involve mutually distrusting and anonymous parties, a consensus mechanism has to be employed that protects the ledger against fraudulent or adverse participants that attempt double-spends. In the current implementation of Ethereum, this mechanism is established by mining based on the proof-of-work (PoW) scheme. All participants have to agree upon a common ledger and all participants have access to all entries ever recorded. The consequences are that PoW unfavorably affects the performance of transactions processing.[5] Concerning the data stored on the ledger, even though records are anonymized, they are nevertheless accessible to all participants, which is problematic for applications that require a higher degree of privacy.

In contrast to Ethereum, Fabric's and Corda's interpretation of consensus is more refined and does not merely boil down to mining based on PoW or a derivative thereof. Due to operating in a permissioned mode, Fabric and Corda provide a more fine-grained access control to records and thus en-

hance privacy. Furthermore, a gain in performance is achieved as only parties taking part in a transaction have to reach consensus.

Fabric. Fabric's understanding of consensus is broad and encompasses the whole transaction flow, starting from proposing a transaction to the network to committing it to the ledger.[6] Furthermore, nodes assume different roles and tasks in the process of reaching consensus. This contrasts to Ethereum where roles and tasks of nodes participating in reaching consensus are identical.

Within Fabric, nodes are differentiated based on whether they are *clients*, *peers* or *orderers*.[7] A client acts on behalf of an end-user and creates and thereby invokes transactions. They communicate with both peers and orderers. Peers maintain the ledger and receive ordered update messages from orderers for committing new transactions to the ledger. *Endorsers* are a special type of peer, whereas their task is to endorse a transaction by checking whether they fulfill necessary and sufficient conditions (e.g. the provision of required signatures). Orderers provide a communication channel to clients and peers over which messages containing transactions can be broadcasted. With respect to consensus in particular, the channels ensure that all connected peers are delivered exactly the same messages with exactly the same logical order.

The mode of participation, permissionless or permissioned, has a profound impact on how consensus is reached.

At this point, the problem arises that there might occur faults in the delivery of messages when many mutually untrusting orderers are employed. As a consequence, a consensus algorithm has to be used in order to reach consensus despite faults, e.g. inconsistent order of messages, thus making the replication of the distributed ledger faults tolerant. With Fabric, the algorithm employed is “pluggable”, meaning that depending on application-

specific requirements various algorithms can be used. For example, in order to deal with random or malicious replication faults as outlined above a variant of the Byzantine fault-tolerant (BFT) algorithms could be used. Furthermore, channels partition message flows, meaning that clients only see the messages and associated transactions of the channels they are connected to and are unaware of other channels. This way, access to transactions is restricted to involved parties only with the consequence that consensus has only to be reached at transaction level and not at ledger level as with Ethereum.

The roles of nodes outlined above are now described in the context of the transaction flow: A client sends a transaction to connected endorsers in order to initiate an update of the ledger. All endorsers have to agree upon the proposed transaction, thus some sort of consensus has to be reached regarding the proposed ledger update. The client now successively collects approval of all endorsers. The approved transaction is now sent to connected orderers which again reach consensus. Subsequently, the transaction is forwarded to peers holding the ledger for committing the transaction.

Without going further into detail, it becomes clear that Fabric allows fine-grained control over consensus and restricted access to transactions which results in improved performance scalability and privacy.

Corda. Similar to Fabric, consensus in Corda is also reached at transaction level by involving parties only. Subject to consensus is transaction *validity* and transaction *uniqueness*[8]. Validity is ensured by running the smart contract code (smart contracts are described in detail below) associated with a transaction, by checking for all required signatures and by assuring that any transactions that are referred to are also valid. Uniqueness concerns the input states of a transaction. Specifically, it has to be ensured that the transaction in question is the unique consumer of all its input states. In other words, there exists no other transaction that consumes any of the same states. The reason for this is to avoid double-spends. Consensus over uniqueness is reached among participants called *notary*[9] nodes, whereas the employed algorithm is “pluggable” as with Fabric. So once again a BFT algorithm might be used.

Smart contracts

The term “smart contract” causes considerable misunderstanding when first encountered as it evokes the idea of some sort of contract that intelligently acts on one’s behalf. The contract’s nature, however, remains vague, but intuitively appears to be linked to legal matters. That said, focal contracts are neither smart in the sense that they are e.g. driven by artificial intelligence, at least not yet, nor do they generally encode obligations and rights that are legally binding. Clark and colleagues[10] provide a useful terminology by highlighting two different ways the term “smart contracts” is commonly used. The first refers to *smart contract code*, the second to *smart legal contracts*, two distinctions that prove to be beneficial in the context of this comparison.

Smart contract code simply denotes software written in a programming language. It acts as a software agent or delegate of the party that employed it with the intention that it fulfills certain obligations, exercises rights and may take control of assets within a distributed ledger in an automated way. Thus, it takes on tasks and responsibilities in the distributed ledger world by executing code that models or emulates contract logic in the real world, though its legal justification may be unclear.

Smart contract code simply denotes software written in a programming language.

All DLTs feature smart contracts in the sense of smart contract code that can be written in Go or Java for Fabric[11], in Solidity[12] for Ethereum and in Java or Kotlin for Corda[13]. In Fabric, the term “chaincode” is used as a synonym for smart contract. As an illustrative example, the reader is reminded of the usage of a smart contract code in the consensus mechanism of Corda in order to ensure transaction validity. However, there is a notable difference between Fabric and Ethereum on the one hand and Corda on the other that is connected to the second way the “smart contracts” term is used.

In Corda, smart contracts not only consist of code but additionally are allowed to contain legal prose. Thus above smart legal contracts are legal prose that are formulated in a way that they can be expressed and implemented in smart contract code. The rationale behind this is to give the code legitimacy that is rooted in the associated legal prose. Such a construct is called Ricardian Contract[14]. At this point, it again becomes clear that Corda was explicitly designed to account for the highly regulated environment of the financial services industry. Both Fabric and Ethereum lack this feature.

Built-in currency

Another noteworthy difference is that Ethereum features a build-in cryptocurrency called *Ether*. It is used to pay rewards to nodes that contribute to reach consensus by mining blocks as well as to pay transaction fees. Therefore decentralized apps (DApps) can be built for Ethereum that allow monetary transactions. Furthermore, a digital token for custom use cases can be created by deploying a smart contract that conforms to a pre-defined standard.[15] This way, own currencies or assets can be defined.

Fabric and Corda do not require a build-in cryptocurrency as consensus is not reached via mining. With Fabric, however, it is possible to develop a native currency or a digital token with chaincode.[16] With Corda, a creation of digital currencies or tokens is not intended.[17]

Summary: customized vs. generic platform

To sum up, the examined DLTs span a continuum. On the one side, there is Fabric and Ethereum. They both are highly flexible, but in different aspects. Ethereum's powerful smart contracts engine makes it a generic platform for literally any kind of application. However, Ethereum's permissionless mode of operation and its total transparency comes at the cost of performance scalability and privacy. Fabric solves performance scalability and privacy issues by permissioned mode of operation and specifically by using a BFT algorithm and fine-grained access control. Further, the modular architecture allows Fabric to be customized to a multitude of applications. An analogy to a versatile toolbox can be drawn.

Corda is located at the other end. It has been consciously designed as DLT for the financial services industry. Most notably, it takes the highly regulated environment into account by augmenting smart contracts with legal prose.

Apparently, Corda's focus solely on financial services transactions simplified its architectural design compared to Fabric. Therefore, it might offer a more out-of-the-box experience. However, it might be possible that Fabric, due to its modularity, can be tailored to resemble Corda's feature set. Efforts exist that seek to integrate Corda into the Hyperledger project. Corda therefore cannot be seen as a competitor to Fabric but more as a complement.

Martin Valenta is blockchain engineer and consultant at the Frankfurt School Blockchain Center. You can contact him via mail (martin.valenta@gmx.net). **Prof. Dr. Philipp Sandner** is head of the Frankfurt School Blockchain Center. You can contact him via mail (email@philipp-sandner.de) or follow him on Twitter (@philippsandner).

[1] https://docs.google.com/document/d/1Z4M_qwILLRehPbVRUsJ3OF8Iir-gqS-ZYe7W-LE9gnE/pub
[2] https://docs.corda.net/_static/corda-introductory-whitepaper.pdf
[3] https://github.com/ethereum/wiki/wiki/White-Paper
[4] e.g. https://github.com/jpmorganchase/quorum
[5] Vukolić M. (2016). The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication, in: Camenisch J., Kesdoğan D. (eds.) Open Problems in Network Security, iNetSec 2015, Lecture Notes in Computer Science, Vol. 9591, Springer.
[6] https://hyperledger-fabric.readthedocs.io/en/latest/fabric_model.html#consensus
[7] https://github.com/hyperledger/fabric/blob/master/proposals/r1/Next-Consensus-Architecture-Proposal.md
[8] https://docs.corda.net/key-concepts-consensus.html
[9] https://docs.corda.net/key-concepts-notaries.html
[10] http://arxiv.org/abs/1608.00771
[11] http://hyperledger-fabric.readthedocs.io/en/latest/chaincode.html
[12] http://solidity.readthedocs.io/en/latest/
[13] https://docs.corda.net/tutorial-contract.html
[14] http://iang.org/papers/ricardian_contract.html
[15] https://www.ethereum.org/token
[16] https://hyperledger-fabric.readthedocs.io/en/v0.6/FAQ/chaincode_FAQ.html
[17] https://discourse.corda.net/t/mobile-consumer-payment-experiences-with-corda-on-ledger-cash/966?source_topic_id=962