A PROJECT ON

COMPARISON OF BLOCKCHAIN PLATFORMS: ETHEREUM, HYPERLEDGER FABRIC AND CORDA

ADVANCED COMPUTER NETWORK SECURITY

COMS 4507/7507

Team Members

Saisrikar Paruchuri	44751575
Michelle Forrester	42658166
Michael Li Connor	41773080
Anant Tuli	44411600

COURSE COORDINATOR

MARIUS PORTMANN

THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Table of Contents

1. Abstract

This report analyzes Ethereum[1], Hyperledger Fabric[2] and Corda[3] blockchain platforms to enable future users in selecting a platform appropriate for their purposes. The motivation behind the creation of each of these blockchain platforms has been considered in relation to their requirements and functionality, limitations and advantages. By comparing these features, it has been concluded that Ethereum, Hyperledger Fabric, and Corda are most suitable for the purposes they were explicitly designed for, and have a varying degree of usefulness outside of this scope. Ethereum is appropriate for general use, open source projects, Hyperledger Fabric is appropriate for use in a business context requiring a greater degree of privacy, and Corda is useful in financial contexts.

2. Introduction

Conventionally, data has been stored in a relatively centralised fashion, whereby a single organisation managed and owned the database. The concept of distributed ledgers has introduced an innovative paradigm to augment and, in a multitude of cases, replace existing conventional operational models. “Distributed ledgers are a type of database that is spread across multiple sites, countries or institutions, and is typically public. Records are stored one after the other in a continuous ledger, rather than sorted into blocks, but they can only be added when the participants reach a quorum.” [4]

Many organizations (and individuals) are embracing this innovation, trying to find novel use cases to deploy this technology. Attempts at understanding this technology and potentially “revolutionizing” existing infrastructure and services has created the need for fulfilling a knowledge deficit in this area – the deficit being which platform to choose. A wide variety of DLT platforms are available to be selected; An important consideration is that they fulfil different needs, were created for different reasons and (in some cases) cater to certain specific industries. Three different distributed ledger technology (DLT) platforms, Ethereum, Hyperledger Fabric and Corda, have been selected and their notable differences will be compared. The aim of this project is to aid decision makers to choose a suitable platform which suits their specific use case.

3. Ethereum

3.1 Background and Motivation behind inception

Ethereum is a decentralised blockchain platform initially ideated by Vitalik Buterin in October 2013 and released in July 2015 as a result of a collaborative effort with Gavin Wood and Vlad Zamfir[5]. The motivation behind Ethereum is to provide an alternative protocol for building decentralised applications while providing a different set of tradeoffs to existing blockchains platforms, providing usefulness to a large class of decentralised applications. Specifically, Ethereum provides advantages where rapid development time, security to small and rarely used applications and the ability of to have important inter-application interactions are important. In order to achieve the intended specification, Ethereum was designed under the following 5 principles[6]:

1. Sandwich Complexity Model: Where the interface to Ethereum is simple and the complex components are within the middle layer away from the consensus and end users.
2. Freedom: Where the restrictions on what the Ethereum protocol can be used for and the preferential treatment of certain smart contract does not exist.
3. Generalisation: The Ethereum protocol and opcodes should be designed with a maximally low-level concept, enabling arbitrary combination and reducing the risk of incompatibility to high level functions.
4. No Features: The Ethereum protocol will implement no build in features, even the most commonly used high level features in accordance with Generalisation principle.
5. Non-Risk Aversion: In the scenario where a risk-increasing change will provide significant benefits, the risk will be taken during the development procedure.

Ethereum became a successful blockchain platform with a strong focus on private and open source peer to peer smart contract user base because of these design principles.

3.2 Design Architecture

The basic architecture of Ethereum takes after Bitcoin but with two distinct differences; the data stored in blockchain is the account information instead of transaction information and the merkle tree structure stores this account information as a single block.

3.2.1 UTXO vs Account Information

Bitcoin stores the information about the user's' balance in a structure known as unspent transaction output. Specifically, the set of unspent outputs is stored along with the owner value. A transaction simply consists of one or more inputs leading to one or more outputs under the constraints that each input must be unspent along with a valid signature matching the owner value of the input and the total input must equal or exceed the total output in the transaction.

Ethereum stores the account information of users involved in a transaction in its merkle tree, a different and simpler approach to Bitcoin. The account information stored in the merkle tree includes the account balance and some Ethereum specific data such as codes and internal storage. A transaction will be marked successful and recorded when the input account has enough balance to pay for a transaction.

There are two notable advantages of UTXO. Firstly, storing only the unspent output as a part of the merkle tree improves privacy as no information about the account holder is stored. Secondly, there are some scalability benefits regarding storing information only about a particular transaction. If the data regarding a transaction is lost in the blockchain network, only the owner of the unspent output is affected while the account approach may effect the whole account [7].

Storing account information in a blockchain provides four major benefits. Firstly, it uses less space as each account can have multiple UTXOs and storing each UTXO in the blockchain can proportionally increase the space requirement. Secondly, not storing UTXO results in greater fungibility as the origin of each unit of currency is not clearly documented, making tracing difficult and preventing the blacklisting of certain coins. Thirdly, account information storage is simpler than UTXO, making it easier to code and

understand. Lastly, storing account information means light clients can easily access information related to an account by traversing through the merkle tree unlike in UTXO where the reference/address changes with each transaction making it harder to track an account [7].

Ethereum was developed with account information as the data stored in merkle tree to take advantage of these benefits. There are weaknesses to this approach in regard to anonymity due to storing account information and defense against replay attack due to the lack of a coin validity tracker. The Ethereum development team used a timed incrementing nonce method to counter the possibility of a replay attack, but the anonymity issue was left to be addressed by the smart contract developers instead of on the protocol level in the spirit of "no features" principle [7].

3.2.2 Merkle Patricia Tree

Ethereum use a Merkle Patricia Tree (MPT) as the primary data structure for storing account information and transactions. MPT is a combination of a merkle tree and patricia tree. It takes features from both these trees to create a data structure with logarithmic computation time for change, add or delete operations and each set of key/value pair will hash uniquely to a hash root. This data structure was first implemented in Ripple protocol [6].

3.3 Consensus

The Ethereum platform reaches consensus in a very similar way to Bitcoin. It uses a proof of work system to hash a block of information onto the blockchain. The new blockchain state is shared between all nodes in the network to reach consensus. However, the hashing algorithm used for the hash process of a block is different to Bitcoin. Ethereum uses a specially designed hashing algorithm called Ethash. The design motivation behind Ethash was to create a proof of work challenge that is not solely dependent on processing power to prevent centralisation when a single entity owns the majority of the processing power. The development team created a challenge with an emphasis on a property known as memory hardness. With memory

hardness, the speed at which the mining hardware is able to read from cache becomes an important factor in the work the hardware performs eliminating the potential of pooling hardware processing power to gain a mining advantage.

The route the algorithm takes to add a block to the chain is as follows [8]:

1. Compute a seed by scanning the block headers of each block up to the point.
2. Compute a 16MB pseudorandom cache from the seed and store it in the light client.
3. Generate a 1GB dataset from cache where each item in the dataset depends on a few items from the cache.
4. Hash random slices of the dataset together and verify with the current difficulty.

With the steps listed above, the majority of the miner's effort goes into reading the dataset. When designing the Ethash algorithm, the development team made sure that the Ethash was best suited for GPU hardware, taking advantage of the optimal cache capacity and processing power ratio of GPU, preventing botnets enabled by CPU mining and the use of ASIC due to suboptimal cache read speed. Lastly, the verification procedure of the Ethash can be done efficiently on a light client, but light clients do not make efficient mining units.

Even with the memory hardness of Ethash's proof of work algorithm, it is still possible to achieve centralisation with a sufficient amount of hardware. In response, Ethereum is considering a proof of stake system.

3.4 Gas and fees

Ethereum Gas corresponds to the transaction fee of Bitcoin, it is largely controlled by the sender and is the incentive for the miners to confirm a Ethereum transaction. Unlike Bitcoin where each transaction requires a static amount of resources to compute, Ethereum with the turing-complete programming language can use an arbitrary quantity of computation steps, bandwidth and storage. Having a static transaction cost can result in vulnerability to potential denial of service attacks. To remedy this, the cost of each transaction is dependent on the number of operations the transaction requires in the form of gas with each operation typically costing 1 unit of gas. The sender of the transaction must then specify how much gas they are willing to

pay for the transaction in the form of startgas and how much ether they are willing to pay per gas in the form of gasprice. This information is made visible to the miners and if the expected reward from the startgas x gasprice is higher than the operation cost, the miner will be willing to process the transaction [7].

The transaction fee a miner can obtain per transaction is described in the following steps [6]:

1. At the start of the transaction execution, startgas x gasprice amount of ether is removed from the sender's account.
2. All operations during the execution of the transaction consume a certain amount of gas.
3. If the transaction is successfully executed and the gas required to perform all operations in the transaction is less than or equal to the startgas, the gas remains or gas_rem if any are refunded back to the sender. The miner will keep the gas consumed for operations at the gasprice specified ether value per gas.
4. In the case where a transaction runs out of gas mid execution, the operations performed thus far will be reversed and all the gas used is sent to the miner.

3.5 Smart Contract - EVM

Ethereum virtual machine (EVM) is the environment in which transactions are executed as code. The EVM can house a number of smart contracts or, in Ethereum's term, autonomous agents. These Autonomous agents have their own reserve of ether and keep a key/value pair to store persistent variables. They will execute a predefined set of code when asked to by a transaction or message operation. Unlike the Bitcoin smart contract bases which are static and were developed as part of the protocol to facilitate currency use in the form of scripts, Ethereum uses a turing-complete programming language. This allows developers to write their own custom smart contracts to reside within the EVM performing a broader range of computations. For example, using Ethereum, developers can write a smart contract to act as a multi-signature account where transactions can only be made when a certain number of account shareholders agrees [9].

3.6 Transaction

In Ethereum, a transaction, also known as a message, is a signed data package sent from one account to another which contains the following [7]:

- Recipient
- Signature of Sender
- Amount of Ether to be transferred
- Optional data field
- STARTGAS - the maximum number of computational steps for transaction
- GASPRICE - the amount of gas to be paid per computational step

In the case of a successful transaction, the Ether corresponding to the agreed amount will be debited from the sender and credited to the recipient, in the case that the recipient has code, the code will be run and internal storage of the receiving account may be debited. Once the transaction is made, the verification process is completed in similar manner to Bitcoin where a miner verifies the transaction block and appends it to the blockchain as a confirmed transaction while reaping the transaction fees.

3.7 Issues

Like many other cryptocurrency blockchain platforms, Ethereum suffers from the scalability issue. As all nodes need to come to a consensus on the state of the blockchain before new blocks can be added, the transaction rate is significantly reduced. Ethereum reached a record transaction per second (TPS) of 12.77 TPS in December 22nd 2017 [1]. This leaves much to be desired, as Ethereum, ideally, should have the capacity to perform more TPS than visa, mastercard and AMEX combined to be able to justify the potential mainstream usage of Ethereum cryptocurrency.

One potential solution to this is sharding. This is a current research topic. A vulnerability to a sudden upshift in scalability is that fewer attacks will be needed to compromise the network [1].

4. Hyperledger Fabric

The Linux Foundation presented Hyperledger Fabric in 2015 for the progression of cross-industry blockchain technologies [10]. Hyperledger Fabric uses smart contracts through which participants complete their transactions. These smart contracts will communicate the approved transactions in the Blockchain ledger. Some key highlights of the current platform are:

- A permissioned blockchain with prompt finality
- Ability to run discretionary brilliant contracts (chaincode) executed in Go [2]
  - User-defined chaincode is typified in a docker holder
  - System chaincode keeps running in a procedure indistinguishable from its companion
- Pluggable Consensus convention. Currently a usage of Byzantine blames tolerant agreement utilizing the PBFT convention is upheld, a model of SIEVE to address non- deterministic chaincode is available, and a protocol stub (NOOPS) serves for development on a solitary hub
- Security support through certificate authorities for TLS certificates, enrollment certificates, and exchange certificates
- Persistent state utilizing a key-esteem store interface, sponsored by RocksDB
- An occasion structure that backs pre-defined and custom occasions
- A customer SDK (Node.js) to interface with the texture.
- Support for essential REST, APIs and CLIs. Support for non-approving associates is insignificant in the engineer review discharge.

4.1 Background and Motivation

Hyperledger Fabric was developed to apply blockchain technology to business, with high focus on permissioned blockchains [11] to overcome the lack of privacy and security in permissionless blockchains. Hyperledger Fabric was designed with the following features [12]:

4.1.1 Identity Management

Hyperledger Fabric is designed to empower permissioned networks. It has a membership identity service to complete user ids and authenticate against all the participants in the blockchain network.

4.1.2 Privacy and Confidentiality

To provide for business interests, Hyperledger Fabric creates private transactions to any group who are on the same permissioned network. Parties outside of this network are unable to view any transactions or group information.

4.1.3 Efficient Processing

Hyperledger Fabric relegates the organization of parts by hub write. To give simultaneousness and parallelism to the system, exchange execution is isolated from exchange requesting and duty. Executing exchanges before requesting them empowers each companion hub to process numerous exchanges at the same time. This simultaneous execution builds preparing proficiency on each associate and quickens conveyance of exchanges to the requesting administration.

Notwithstanding empowering parallel preparing, the division of work unburdens requesting hubs from the requests of exchange execution and record support, while peer hubs are liberated from requesting (agreement) workloads. This bifurcation of parts additionally restrains the handling required for approval and confirmation; all companion hubs don't need to put stock in all requesting hubs, and the other way around, so forms on one can run autonomously of check by the other.

4.2 Chaincode Functionality

Chaincode applications encode rationale that is summoned by particular sorts of exchanges on the channel. Chaincode that characterises parameters for a difference in resource proprietorship, for instance, guarantee that all exchanges that exchange possessions are liable to similar principles and necessities. Framework chaincode is recognized as chaincode that characterises working parameters for the whole channel. Lifecycle and setup framework chaincode characterises the standards for the channel.

Underwriting and approval framework chaincode characterises the necessities for embracing and approving exchanges.

4.3 Modular Design

Hyperledger Fabric is implemented by a modular architecture where specific algorithms for encryption, identification and ordering can be done by any Hyperledger Fabric network. It is a universal blockchain architecture that anyone can adopt, and it assures its network is interoperable across markets.

4.4 Architecture

The peer validation nodes run a BFT consensus protocol to execute a replicated state machine which accepts the below types of transactions as operations [2]:

- Deploy Transaction: a transaction which takes a chaincode as a parameter and is installed on the peers, and ready to invoke.
- Invoke Transaction: A transaction which invokes a transaction of a chaincode which has already been installed by a deploy transaction. The arguments are very specific to the transaction type and accordingly it may read and write entries in its state. It indicates whether it is succeeded or failed.
- Query Transaction: a transaction which will read the peer's persistent state and returns an entry of the state. It doesn't determine linearizability.

Chaincode will define its persistent entries in the state. The hash chain is calculated over the result of persistent state and transactions that are executed. Approval of exchanges happens through the repeated execution of the chaincode and given the blame supposition hidden BFT agreement, i.e., that among the n approving associates at most $f < n/3$ may "lie" and carry on discretionarily, however all others execute the chaincode effectively.

At the point when executed over PBFT agreement, it is vital that chaincode exchanges are deterministic, generally the condition of the associates may wander. A measured answer for filter out non-deterministic exchanges that are verifiably wandering is accessible and has been actualized in the SIEVE convention. Enrolment among the

approving hubs running BFT agreement is static and the setup requires manual mediation. Support for progressively changing the arrangement of hubs running accord is anticipated a future form. As the texture executes a permissioned record, it contains a security foundation for confirmation and approval. It bolsters enlistment and exchange approval through open key certificates, and confidentiality for chaincode acknowledged through in-band encryption [2].

More absolutely, to connect to the system each associate needs to get an enlistment certificate from an enlistment CA that is a piece of the participation administrations. It approves a companion to associate with the system and to procure exchange certificates, which are expected to submit exchanges. Exchange certificates are issued by an exchange CA and bolster pseudonymous approval for the associates submitting transactions, in the sense that multiple transaction certificates issued to the same peer (that is, to a similar enlistment certificate) can't relate to each other.

Confidentiality for chaincode and state is given through symmetric-key encryption of transactions and states with a blockchain specific key that is available to all peers with an enrolment certificate for the blockchain. Broadening the encryption instruments towards more fine-grained confidentiality for exchanges and state passages is gotten ready for a future variant [2].

4.5 Smart Contracts

Hyperledger Fabric savvy contracts are composed in chaincode and are conjured by an application outside to the blockchain when that application needs to cooperate with the record. By and large, chaincode communicates just with the database part of the record, the world state (questioning it, for instance), and not the exchange log. Chaincode can be executed in a few programming dialects. The right now upheld chaincode dialect is Go with help for Java and different dialects coming in future discharges [10].

4.6 Consensus

Exchanges must be composed to the record in the request in which they happen, even though there may be between various arrangements of members inside the system. For this to happen, the request of exchanges must be set up and a strategy for dismissing

terrible exchanges that have been embedded into the record in blunder (or perniciously) must be instituted [10].

This is an altogether examined zone of software engineering, and there are numerous approaches to accomplish it, each with various exchange offs. For instance, PBFT (Practical Byzantine Fault Tolerance) can give a component to record reproductions to speak with each other to keep each duplicate steady, even in case of defilement. On the other hand, in Bitcoin, requesting occurs through a procedure called mining where contending PCs race to explain a cryptographic perplex which characterizes the request that all procedures therefore expand upon.

Hyperledger Fabric has been intended to permit arrange starters to pick an accord component that best speaks to the connections that exist between members. Similarly, as with security, there is a range of necessities; from systems that are exceptionally organized in their connections to those that are more distributed [10].

4.7 Issues

The texture ccenv picture which is utilized to assemble chaincode, at present incorporates the github.com/hyperledger/texture/center/chaincode/("shim") bundle. This is advantageous, as it gives the capacity to bundle chaincode without the need to incorporate the "shim". Be that as it may, this may cause issues in future discharges (as well as when endeavoring to utilize bundles which are incorporated by the "shim") [13].

So as to stay away from any issues, clients are encouraged to physically merchant the "shim" bundle with their chaincode preceding utilizing the associate CLI for bundling or potentially to install chaincode [13].

5. Corda

Corda is a platform created by R3 which uses a distributed ledger of mutually distrusting nodes to provide an authoritative and decentralized system of records which can be securely shared among firms [14, 1]. Unlike more generalized blockchain platforms, Corda was explicitly designed for use in the financial sector [3, 7]. To fulfil this purpose, Corda must exist within the

current legal framework; rely on proven technology; and follow a structure appropriate for the requirements of institutions [14, 3]. This purpose and these requirements strongly influence the structure and features of the platform and its advantages and limitations. Corda frequently repurposes existing technology and has several design features which differentiate it from more conventional blockchain platforms [14, 5].

5.1 Overview

Corda is a platform for writing applications which extend a global database with new capabilities, known as CorDapps. These apps define new data types, new inter-node protocol flows, and smart contracts which determine changes to the database [14, 6].

Data is modelled as arbitrary object graphs known as states which are stored in the ledger, form the atomic unit of data, and may declare scheduled events. Corda emphasizes integration with existing systems and, among other features such as rapid bulk data imports from other database systems, and ledger events exposed via an embedded JMS compatible message broker, all nodes are backed by a relational database. Data can be queried using SQL and joined with private tables. This data can express a wide range of types, for example time and currency.

All communication is in the form of flows. These are complex, multi-step transaction building protocols consisting of bytecode-to-bytecode transpilation and modelled as blocking code. These flows expose progress information to node administrators and users [14, 5].

Unlike Ethereum, transactions across Corda's peer-to-peer network are not broadcast to the entire network. Instead they are communicated only to the parties directly involved in the transaction [14, 10]. Admission requires obtaining an identity signed by a root authority which must be unique within the network. As the financial industry often requires some level of customer checking, Corda reuses standard PKIX infrastructure for connecting public keys to identities, which may be kept anonymous within a transaction [14, 7-8].

Corda does not use a blockchain to store and verify transactions, and by extension does not use miners or proof-of-work. Instead, each state points to a notary. A notary

is a service which will sign a transaction only if all the input states are unconsumed [14, 6].

The network itself consists of nodes communicating using AMQP/1.0 over TLS to secure messages in transit and authenticate endpoints [14, 9]. It uses a network map service publishing information about the network nodes which operates similar to Tor's directory authorities [14, 8]. Communication between nodes operates like an email network in that messages are written to disk and delivery is retried until the remote node acknowledges a message [14, 8-9].

5.2 Consensus

Rather than organising time into blocks, Corda networks have one or more notary services providing time stamping and transaction ordering. These notaries are composed of mutually distrusting parties who use a standard consensus algorithm and are identified by and sign with composite public keys. A notary may either sign a transaction or a rejection error if a double spend has occurred. A signed transaction from the state's choses notary indicates the transaction is final [14, 29].

Corda does not have an equivalent of a 'hard fork'. Fraudulent or buggy transaction chains must be discarded by mutual agreement of the involved parties. Corda provides node logs which provide sufficient information to determine the set of entities who must agree to discard a subgraph. A flow exists which will signal an administrator that a decision is required. If the administrator accepts the request, the next hops in the transaction chain are returned and the network can be semi-automatically crawled through to find all parties who would be affected by a transaction rollback [14, 19]. The involved parties may either extend the transaction chain with new transaction correcting the ledger or the involved parties, minus parties who may be uncooperative due to fraudulent activities, mark the relevant states as no longer consumed [14, 20].

5.3 Smart Contracts

Corda defines Smart Contracts under the UTXO model, as used in BitCoin. The database is a set of immutable rows which, unlike BitCoin, can contain arbitrary data and are termed states. Transactions define outputs which append new states and inputs which consume existing states. These states are associated with JVM bytecode programs which

accept a transaction if it satisfies both the input and output states simultaneously, making it valid [14, 6].

A contract is a class implementing the Contract interface which exposes a single function called verify and makes use of Java's comprehensive type system for expressing common business data. To support development on other non-JVM platforms, allowed types are restricted and a standardised binary encoding scheme is provided [14, 18]. Contracts may be stored in zip files which are attached to transactions [14, 17]. Corda doesn't implement a mechanism describing what a contract is meant to do and they are not required to carry legal weight. However, it's expected that they would be legal contracts which take precedence over software implementation in the event of a disagreement [14, 19]. Contract constraints can be implemented using composite keys, allowing multiple signatures from different private keys to be used in the standard JAR signing protocol [14, 24].

5.4 The Vault

The vault is Corda's equivalent of a wallet. It contains data extracted from the ledger pertaining to the node's owner and private key information for consuming states. The vault implements coin selection, which is the process of creating transactions sending value to someone else by combining asset states and possibly adding a change output to make the values balance. Transactions are not liable to per transaction network fees, however they must respect fungibility rules to ensure issuer and reference data is preserved between parties. Vaults manage soft locks to prevent multiple transactions from using the same output simultaneously [14, 34].

5.5 Oracle

Corda implements oracles, a network service which is trusted to sign transaction containing statements about the world outside the ledger only if the statements are true [14, 21]. This allows counter-parties to view a small part of a transaction allowing them to check signatures and sign it themselves. Oracles sign off an entire transaction and data which isn't required by the oracle is 'torn off' before sending. This is implemented a Merkle hash tree where the hash used for the signing operation is the root. The counterparty is presented with the required elements and the Merkle branches linking

them to the root hash, allowing them to sign the entire transaction without viewing all of it. The resulting signature contains flag bits indicating which parts where presented for signing. This method ensures there is only one place where signatures may be found greatly simplifying signature checking [14, 22].

5.6 Encumbrances

Corda introduces the concept of encumbrances. Within finance, it is useful to be able to limit changes to an asset using a time lock, a restriction on a state which prevents it being modified until a certain time. Encumbrances allow states a specify other states which must be present in any transaction that consumes it [14, 24].

5.7 Data Structures

5.7.1 Transactions

Transactions consume zero or more states (inputs), and create zero or more new states (outputs). States never alter and may either be current ('unspent') or consumed ('spent'). They cannot exist outside of the transaction that created them and may be identified by the identifier of the creating transaction and the index of the state in the outputs list [14, 13]. Within a financial context, contextual information is considered important. As such, transactions have input references, output states, attachments, commands, signatures, type, timestamp, and summaries [14, 13-14]. Signatures are appended to the end of transactions and contracts check each set of public keys, a composite key, for a matching signature [14, 14]. The timestamp communicates the position of the transaction on the timeline to the smart contract code for enforcement of contractual logic [14, 17]. Attachments are zip files which are stored and transmitted separately to transaction data and are intended to provide data on the ledger that parties may wish to use repeatedly [14, 18].

5.7.2 Financial Constructs

Corda provides data structures to represent several common financial constructs. The ledger may store an Ownable States, which attach ownership to assets by requiring an owner field consisting of a composite key. An ownable state has a Fungible Asset to represent assets of measurable quantity, for example pound notes [14,

[25]. Corda supports an Amount<T> type to account for difference in fungible state of cash due to differences in the issuers and their stability. This type defines an integer quantity of some token, for example Issued<T> which encapsulates what the asset is, the issuer, and an opaque reference field.

Financial parties may participate in netting, the process by which a set of gross obligations is replaced by an economically-equivalent set where eligible offsetting obligations have been elided. Corda provides an Obligation contract as a subclass of Fungible Asset which provides methods to calculate simple bilateral nettings and verify the correctness of both bi and multilateral nettings [14, 26].

Within the financial industry, Corda offers the advantages of significantly reducing the amount of time and effort required to keep multiple, manual ledgers synchronised; increasing code sharing [3, 1]; reducing financial costs; eliminating errors; and reducing operational risk [3, 3]. Corda's largest limitation is the scope of its use, unlike more generalized blockchain platform, it is exclusively for use in financial sector and isn't appropriate for other purposes [3, 15].

6. Comparison of Blockchain Platform

Before exploring the differences between the three platforms, we will look at certain commonalities among distributed ledger technologies:

- Distributed networks consist of individual members, called nodes/peers
- DLTs require verification of the data's validity across nodes/peers within the network i.e.
    - All nodes can be assured of the data's integrity and authenticity and consistency
    - All nodes should agree upon a common truth
    - This is called consensus [15]

6.1 Consensus

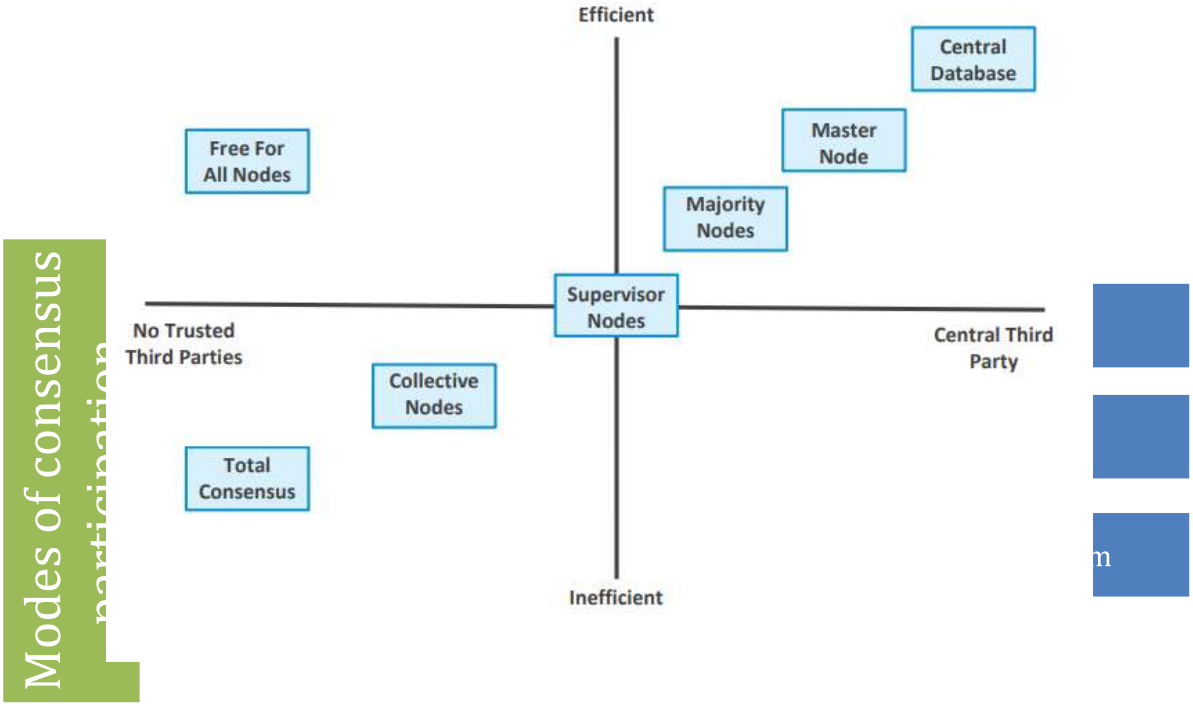
Reaching or achieving a consensus is a very fundamental requirement of using distributed systems to store data. There are two broad types of consensus-reaching mechanisms – permissioned and permissionless.

Depending on the requirements of the system, different types of consensus models are available. There are certain considerations which affect characteristics of consensus, e.g. speed of achieving consensus, level of trust attained etc.

The following grid attempts to explain the trade-offs that exist while choosing a consensus participation model.

Fig CNS. Consensus models [16]

The consensus participation models for the three selected platforms (Fabric, Corda and Ethereum) can be categorized as follows:



Ethereum

Due to the fact that Ethereum has a permissionless consensus system (which implies that there is no inherent trust that exists between nodes), Ethereum needs a proof-of-work system for validation.
Although a permissionless consensus system sounds ideal (in terms of the core beliefs and principles of distributed ledger technologies), it has an adverse impact on performance since consensus must be reached by all participants.
Although currently, Ethereum uses a proof-of-work consensus mechanism, a proof-of-stake solution is in the works.

Corda and Fabric

Corda and Fabric, on the other hand, operate on a permissioned consensus model. Even though a permissioned system could be considered a less-than-ideal distributed ledger system, there is a significant performance advantage over a permissionless consensus model like Ethereum. This is due to the fact that not all nodes/peers within the network need to reach a consensus.
Apart from the performance benefits, Corda and Fabric are more suitable for "closed"/" limited" networks where a higher degree and finer control of privacy, security and authorisation is required.

6.2 Currency

Ethereum

Ethereum operates on an inbuilt currency called Ether. This is required because Ethereum (currently) requires proof of work for consensus and thus, Ether is used to reward miners.

Fabric

Although there is no inbuilt/required currency like Ethereum, chaincode can be used to emulate a currency-based system. This is due to the fact that reaching a consensus in this system does not require proof-of-work and thus, mining.

Corda

Corda does not support use of currency.

6.3 Applications

With the various categories discussed above, we should now be able to make a decision as to which applications are ideal for which particular use case. In terms of versatility, Fabric and Ethereum are more generalized platforms which can be adapted to a multitude of scenarios. Corda, on the other hand was created with a focus solely on the finance industry.

Corda

Corda is ideal for financial services and is more tailor-made for that purpose. Therefore, Corda should be the platform of choice if administrators want a:

- Ready-to-use system focussed solely on providing financial services
- Rigid but simplified system which although reduces complexity of the system, but also offers less customizability

Ethereum

Ethereum has its advantages, due to the fact that it offers a permissionless consensus mechanism and smart contracts, but is bogged down by its scalability issues (due to the proof-of-work consensus; across all nodes). The “complete transparency” aspect of Ethereum is a double-edged sword, because even though distributed ledgers should ideally uphold complete transparency, there are certain situations where an organization requires a higher degree of privacy, security and authorization.

Fabric

Lastly, Fabric seems to be the ideal trade-off between being a generalized platform for DLT applications (customizable), being scalable and also offering better options in terms of privacy and authorization. Its modular structure makes it highly versatile. Contrarily, it can be argued that the rationale behind the inception of the distributed ledger technology was for it to be permissionless i.e. publicly owned data in the first place. In essence, third parties are a potential security flaw. Therefore, Fabric would be ideal for applications where nodes/peers are clearly identifiable (which makes it hard to be implemented as a public DLT) [17].

6.4 Comparison Table

Category	Feature	Ethereum	Corda	Hyperledger Fabric
Transactions	Cryptocurrency	Ether	None	Create any Cryptocurrency
	Unit	Gas	None	None
	Amount	Operation Dependent	None	None
Smart Contract	Runtime Environment	Ethereum Virtual Environment	IntelliJ IDEA (IDE using Java/Kotlin)	Hyperledger Composer [7]
	Programming Level	High Level Programming Language	High Level Programming Language	General Purpose Programming Language [6]
	Developed by	End user developer	End user developer	Linux Foundation [1]
Consensus	Centralisation	Decentralised	Decentralized	Decentralised
	Ownership	Public/Open source	Private	Private/Open source [7]
	Proof Concept	Currently - memory hard Proof of Work, researching into Proof of Stake being done.	Transaction level validation via notaries	Validation of transaction by peer nodes

Data Structure	Hashed transaction	Account Information	Proposed output/input states	Ordering Service
	Stored structure	Merkle Patricia Tree	State Sequence	Block contains an array of totally ordered transaction
Known Issues		Scalability - Sharding being researched as potential solution	Difficulty implementing transaction rollback	Problems when using Shim package which are included by shim[5].

7.

8. Conclusion

With the recent popularity of blockchain and the influx of platforms which support and expand on this technology, the number of platforms available to users has increased drastically. To aid future users in selecting an appropriate blockchain platform, this report has compared Ethereum, Hyperledger Fabric, and Corda based on their motivation, function, and advantages and disadvantages.

Ethereum was designed to provide an alternative protocol for building decentralized applications, creating a new set of tradeoffs to existing platforms and aiding in the creation of a wide range of applications. It uses the cryptocurrency Ether and the concept of gas units. Smart contracts are created using the Ethereum Virtual Environment by the end user. It is a decentralized and open public source platform which relies on memory hard proof of work with potential future use of proof of stake. A transaction consists of account information stored on a Merkle Patricia tree. It suffers from scalability issues.

Hyperledger Fabric was created to allow the advantages of blockchain platforms to be used in a business context, and includes features for a permissioned blockchain, identity management, privacy and confidentiality. Hyperledger Fabric allows the use of any cryptocurrency, and as such the units used are variable. The technology is open source and availability of transactions is subject to permission. Smart contracts are implemented using Hyperledger Composer by the end user developer. Consensus is achieved by peer nodes validating transactions. Data is stored in blocks containing an array of ordered transactions. Hyperledger Fabric has suffered issues related to the use of Shim.

Corda was created explicitly for use in the financial sector, and is designed to have legal accountability. Corda is a private, permissioned platform where only directly involved parties have access to transactions. It doesn't use a cryptocurrency but provides data types to express common forms of assets, such as cash and ownable states. All data is modelled as a state, an immutable data type which may either be spent or unspent.

Transactions are defined as changes to these states which must be validated by notaries to achieve consensus, and are stored as a state sequence. Smart contracts are essentially zip files implemented using Java or Kotlin, and allow more contextual information to be included than other platforms. Corda is limited in the scope of its use. It is largely only appropriate for use in a financial context, or an environment which is heavily institutionalized. Rollbacks of transactions are difficult to achieve as they require the consent of all parties involved.

These three platforms were designed for a specific reason and market. As such, their design and functionality is appropriate to particular purposes. Ethereum is a very generalized platform which can be used for many purposes. The transparency of transactions makes it less appropriate for situations requiring a high degree of secrecy, privacy, or authorization. Hyperledger Fabric's use of permissioned services and the large selection of cryptocurrencies available allow its users more privacy, security, and easier incorporation with existing systems or practices. It is more appropriate for use in business contexts. Corda's design allows the same degree of privacy but makes a greater provision for incorporation with fiscal arrangements in the real world, including data types common in finance and legal accountability. It is appropriate for the highly regulated practices of financial institutions.

9. References

[1] J. Ray, *Ethereum Introduction*. 2018. Available: https://github.com/ethereum/wiki/wiki/Ethereum-introduction

[2] C. Cachin, "Architecture of the Hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[3] R. Brown, J. Carlyle, I. Grigg and M. Hearn, *Corda: An Introduction*. 2016. [PDF] Available: https://static1.squarespace.com/static/55f73743e4b051cfcc0b02cf/t/57bda2fdebbd1acc9c0309b2/1472045822585/corda-introductory-whitepaper-final.pdf

[4] U. G. Chief Scientific Adviser, "Distributed Ledger Technology: beyond block chain".

[5] V. Buterin, *A Prehistory of the Ethereum Protocol*. 2017. Available: https://vitalik.ca/2017-09-15-prehistory.html
[6] J. Ray, *Ethereum Design Rationale*. 2018. Available: https://github.com/ethereum/wiki/wiki/Design-Rationale
[7] J. Ray, *Ethereum White Paper*. 2018. Available: https://github.com/ethereum/wiki/wiki/White-Paper
[8] KPE, Ethash. 2018. Available: https://github.com/ethereum/wiki/wiki/Ethash
[9] A. Hertig, *How Do Ethereum Smart Contract Work.* 2018. Available: https://www.coindesk.com/information/ethereum-smart-contracts-work/
[10] "Introduction — hyperledger-fabric docs master documentation", Hyperledger-fabric.readthedocs.io, 2018. [Online]. Available: http://hyperledger-fabric.readthedocs.io/en/release-1.1/blockchain.html. [Accessed: 21- May- 2018].
[11] M. Vukolić, "Rethinking permissioned blockchains," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, 2017, pp. 3-7: ACM.
[12] "Hyperledger Fabric Functionalities — hyperledger-fabric docs master documentation", *Hyperledger-fabric.readthedocs.io*, 2018. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.1/functionalities.html. [Accessed: 21- May- 2018].
[13] "Release Notes — hyperledger-fabric docs master documentation", *Hyperledger-fabric.readthedocs.io*, 2018. [Online]. Available: http://hyperledger-fabric.readthedocs.io/en/release-1.1/releases.html. [Accessed: 21- May- 2018].
[14] M. Hearn, *Corda: A distributed ledger*. 2016. [PDF] Available: https://docs.corda.net/_static/corda-technical-whitepaper.pdf
[15] D. A. Baliga, Understanding Blockchain Consensus Models (Whitepaper), 2017.
[16] Z/Yen Group Limited, "http://www.longfinance.net," July 2017. [Online]. Available: http://www.longfinance.net/LongFinance/DLTCoursePDF2.pdf.
[17] M. Scherer, "Performance and Scalability of Blockchain Networks and Smart Contracts".