PAPER SUBMISSION

Blockchain for Economically Sustainable Wireless Mesh Networks

Aniruddh Rao Kabbinale[1] | Emmanouil Dimogerontakis[2,3] | Mennan Selimi[2,3] | Anwaar
Ali[1] | Leandro Navarro[2,3] | Arjuna Sathiaseelan[3]

[1]University of Cambridge, Cambridge, UK
[2]Universitat Politècnica de Catalunya. Barcelona, Spain
[3]Ammbr Research Labs, Cambridge, UK

Summary

Decentralization, in the form mesh networking and blockchain, two promising technologies, is coming to the telecommunications industry. Mesh networking allows wider low cost Internet access while blockchain enables complete transparency and accountability for investments and revenue or other forms of economic compensations from sharing of network traffic, content and services. Crowdsourcing network coverage combined with crowdfunding costs can create sustainable yet decentralized Internet access infrastructures, where every participant can invest in resources, and pay and be paid for usage. While mesh networks and mesh routing protocols enable self-organized networks that expand organically, cryptocurrencies and smart contracts enable the economic coordination among network providers and consumers. We explore and evaluate two existing blockchain software stacks, Hyperledger Fabric (HLF) and Ethereum geth with Proof of Authority (PoA), deployed in a real citywide production mesh network, and in a centralized laboratory network. We quantify the performance, bottlenecks and identify the current limitations and opportunities for improvement to serve the needs of wireless mesh networks.

KEYWORDS:
Mesh networks, Blockchain, Performance evaluation, Ethereum, Hyperledger Fabric

1 | INTRODUCTION

Network infrastructures are critical to provide local and global connectivity that enables access to information, social inclusion and participation for everyone. Local connectivity largely relies on access networks. Wireless mesh networks (WMNs) are a kind of access networks comprising of wireless nodes namely wireless mesh routers, wireless mesh clients, and network gateways. A client (wired to a mesh router or through a WiFi access point) can access the Internet across a WMN (Akyildiz u. a., 2005). These are self-organized networks that can grow organically: new network links can expand the coverage of the network or increase the capacity when links get overused. The routing protocol runs in every router by measuring the performance and quality of links and coordinates distributed decisions about the best network paths every time. The result is that, once a routing protocol is adopted, the development and operation of the network only depends on pooling routers and links with local decisions, without any central planning or management.

These decentralized networks are essential to develop community access networks, network infrastructure commons, built by citizens and organizations which pool their resources and coordinate their efforts, characterized by being open, free and neutral (Baig u. a., 2016). These decentralized access networks have been identified as one way to connecting the next billion

people that are still without the Internet access (ITU). Guifi.net[1] is an example of such a community effort which is one of the biggest community networks in the world, with more than $34,000$ participating routers, combining technologies including wireless mesh and fibre. However the main challenge of these peer-to-peer socio-technical structures are around trust among agreements between peers and how to ensure the economic sustainability of this collective effort and the balance between contribution and consumption(Baig u. a., 2016).

As an example scenario and mechanism for economic sustainability is the economic *compensation system* used in Guifi.net (Baig u. a., 2016). An answer to the lack of incentives to invest in network infrastructure, it was introduced in 2011 as a cost sharing mechanism. The idea of the compensation system is to balance between total resource contribution and its consumption. The economic cost of any contribution and consumption of network resources by each participant in a given locality are recorded. The overall result is a zero-sum computed periodically, from monthly to quarterly, where the participants with over-consumption or negative balances have to compensate those with over-contribution or positive balances.

Currently the above described economic compensation system is done manually: each participant declares its costs and consumption and then the Guifi.net foundation[2] validates this claim by cross checking it with their own network traffic measurement data and network inventory, according to the agreed list of standard costs. Any disparities between these two records are flagged, clarified or raised to a conflict resolution mechanism. There is, however, room for error or intentional false or exaggerated claims put forward by a participant, the recorded data being tampered with, or simply mistrust among the parties. The correct application of the compensation system is critical for the economical sustainability of the network, ensuring its proper operation, as well as future investments. Therefore, we argue that there is a need for an automated system where participants can trust that the consumption of resources is being accounted in a fair manner, and that these calculations and money transfers are automated to avoid the cost, delays, errors and potential mistrust from manual accounting and external payments.

Blockchain technologies offer solutions that seems apt to make the peer-to-peer nature of access networks trusted and economically sustainable. Blockchain (more details in Section 2) is an immutable and distributed data storage without the provision of retrospective mutation in data records. However, most blockchain networks are open and public (permissionless) that encourage the users to be anonymous (Nakamoto, 2008). This implies that anyone, without revealing their true identity, can be part of such a network and make transactions with another similarly anonymous peer of the network.

In the perspective of community networks such as Guifi.net, however, every participant who joins the network to contribute and benefit from the infrastructure must first register its identity and the identity of the resources that it contributes to the wider pool. This is particularly needed so that any malicious entity, such as hidden nodes in Guifi.net used by other ISPs, can be filtered out (Neumann u. a., 2016). Because of such registration process one also needs an efficient identity mechanism on top of blockchain's immutable record keeping. *Permissioned blockhains* are part of such solutions, mostly envisioned for business networks where there is often a stringent requirement of *know your customer* in addition to keeping the intra- and inter-business transactions confidential.

In this study, we extend our previous work Selimi u. a. (2018b) by exploring the plausibility of combining decentralized access networks with an permissioned blockchain running on servers inside the access network, that would result in a model for economically self-sustainable decentralized mesh access networks, guaranteeing trust among participants, allowing economic profitability, and enabling at the same time easier Internet connectivity. We study the viability of such an approach, by evaluating two of the most prominent platforms for building local blockchain applications. These platforms are Hyperledger Fabric (HLF)[3], an industry-oriented modular, and permissioned distributed ledger (see Section 2.1 for details) and Ethereum[4] (see Section 2.2), a general-purpose, bussiness-oriented nonetheless, platform.

We deploy the Hyperledger Fabric and Ethereum platform in a centralized network in our laboratory, and well as in a decentralized production wireless mesh network that is part of Guifi.net. Our key contributions are summarized as follows:

- First, we analyze the performance of both platforms in terms of metrics such as transaction latency, CPU and memory utilization of Hyperledger Fabric and Ethereum components. To the best of our knowledge, this is the first Hyperledger Fabric and Ethereum deployment made in a production wireless mesh network. Our results show that both Hyperledger Fabric and geth Ethereum network can be deployed on even resource constrained devices like RPI3 boards or router boards with limited computational capability. Both the blockchain software stacks perform well without saturation and much delays for a moderate load of up to 100 transactions fired in the network at a time. In Hyperledger Fabric, our

---

[1]https://guifi.net/
[2]https://fundacio.guifi.net/Foundation
[3]https://www.hyperledger.org/projects/fabric
[4]https://www.ethereum.org/

measurements reveal that endorsers are the bottleneck and care has to be taken in designing endorsement policy for scaling the network. In case of Ethereum, our results show that a there is a limit on the number of requests a node can support and can only be scaled vertically i.e. by increasing computational capability of serving node

- Second, driven by the findings in a mesh network, we propose a placement scheme for Hyperledger Fabric and Ethereum components that optimizes the performance of the blockchain components in mesh networks.

2 | BLOCKCHAIN: THE UNDERPINNING TECHNOLOGY

Blockchain is an *append-only* immutable data structure. Its first incarnation was in the Bitcoin cryptocurrency network (Nakamoto, 2008). Blockchain was used to enable trust in financial transactions among different non-trusting parties in a pure peer-to-peer fashion without the need for going through a third financial party like e.g., a bank. Such trust is provided in terms of immutability of blockchain's data structure. Each *block* in blockchain contains information that is immutable. The immutability aspect is rendered true by including the hash of all the contents of a block into the next block which also chains the blocks together. Tampering with one block disturbs the contents of all the following blocks in the chain. Each block in the chain is appended after a *consensus* is reached among all the peers of the network. The same version of a blockchain is stored in a distributed manner at all the peers of the network. That is why it is sometimes referred to as *distributed ledger* as well.

In this section, we briefly discuss the target blockchain platforms. Two blockchain platforms are chosen for evaluation in wireless mesh networks namely Hyperledger Fabric and Ethereum, due to their popularity and potential to be used in different applications.

2.0.1 | Permissionless vs Permissioned, Public vs Private

Bitcoin (Nakamoto, 2008) and Ethereum[5] (Wood, 2014), as various other blockchains, are considered as *permissionless*, meaning that anyone has "write" access to the blockchain. As a result anyone can be a part of the network, mining and performing transactions with other parties. The consensus in such an open environment is tackled with algorithms like the Proof-of-Work(PoW) protocol. Some degree of anonymity is also at the heart of such platforms. A user (or in general an entity) usually uses the hash of its public key as its identifier as opposed to using its real-world credentials.

In the aspect of "write" openness, *permissioned blockchains* are in sharp contrast with public blockchains which we discuss next. Permissioned blockchains, a concept particularly popularized by the Linux Foundation's Hyperledger, are usually considered for business applications. In such applications the identity of users, in addition to trusted and immutable data storage, is also important such as the stringent requirement of *know your customers* for many businesses. Hyperledger tries to leverage the best of both worlds by implementing a cryptographic *membership service* on top of blockchain's trusted, immutable, and distributed record keeping.

Another categorization can be done based on the openness of reading from the blockchain. In the case where a blockchain exposes its data publicly it is characterized as *public*. On the other hand, blockchains that prohibit access to its data are called as *private*.

In our study, the requirement of both users' identity and trusted record keeping is of paramount importance and that is why we decided to conduct our study using private permissioned blockchains. Hyperledger Fabric fulfills by default these properties. On the other hand, while Ethereum is not primarily destined to serve this purposes, it can also be used as private permissioned blockchain. Nevertheless, executing resource-full consensus algorithms in a permissioned environment where the participants are known has no application except experimentation with the protocols themselves. On the other hand, some protocols, like Ethereum, offer inexpensive consensus algorithms, like the Proof-of-Authority (POA) protocol, that are ideal for a private permissioned instances.

2.1 | Hyperledger Fabric (HLF)

Hyperledger Fabric (HLF) (Androulaki u. a., 2018) is an open source implementation of a permissioned blockchain network that executes distributed applications written in general-purpose programming languages (e.g., Go, Java etc) (Androulaki u. a.,

---

[5]https://ethereum.org/

2018). HLF's approach is modular, which implies that the platform is capable of supporting different implementations of its different components (such as different consensus protocols) in a *plug-and-play* fashion.

The HLF architecture comprises of the following components:

Peers: Peers can further be of two types namely *endorsers* and *committers*. A peer is called a *committer* when it maintains a local copy of the ledger by committing transactions into its blocks. A peer assumes the role of an *endorser* when it is also responsible for simulating the transactions by executing specific chaincodes and endorsing the result (see the next subsection 2.1.1). A peer can be an endorser for certain types of transactions and just a committer for others.

Ordering service: The role of this component is to order the transactions chronologically by time stamping them to avoid the *double spend problem* (Nakamoto, 2008). The ordering service creates new blocks of transactions and broadcast them to the peers which then append these blocks to their local copy of the blockchain (or ledger). The ordering service can be implemented as a centralized or decentralized service (Sousa u. a., 2017). It is at the ordering service level where the consensus (like proof-of-work in Bitcoin (Nakamoto, 2008)) related to the state of a blockchain takes place.

Chaincode: A chaincode or a *smart contract* is a program code that implements the application logic. It is run in a distributed manner by the peers. It is installed and instantiated on the network of HLF peer nodes, enabling interaction with the network's shared ledger (i.e., the state of a database modeled as a versioned key/value store).

Channel: A channel provides a higher layer of confidentiality abstraction. A channel can be considered as a subnet on top of a larger blockchain network. Each channel has its own set of chaincodes, member entities (peers and orderers), and a distinct version of a distributed ledger. This should not be confused with a similar term, payment channels, used to make multiple off-chain micro-payments, multiple transactions, without committing all to a blockchain.

Membership service provider (MSP): HLF makes use of a dedicated and exhaustive Membership Service Provider (MSP)[6], which is based on public-key infrastructure (PKI) and hierarchical certificate authorities (CAs), to define roles and security clearance (for different channels) of different entities for a particular use case. The goal of such a dedicated MSP is to realize the concept of an organization-like hierarchical security infrastructure in the form of a hierarchical and permissioned version of blockchain.

2.1.1 | HLF Protocol

Figure 1 depicts the sequence of transaction execution steps in HLF's environment. The description of these execution steps are as follows:

1. Transaction (Tx) proposal: In this step clients access the HLF blockchain to submit a proposal for a Tx to be included in one of the blocks of the HLF blockchain. Clients propose a transaction through an application that uses an SDK's (Java, Python etc) API. This is shown as the first step in Figure 1 .

2. Endorsement and Tx simulation: The transaction proposal from the above step is then broadcasted to the endorsing peer nodes in the HLF blockchain network. Each endorsing peer verifies the Tx proposal in terms of its correctness (i.e., its structure, the signatures that it contains, and the membership and permission status of the client that submits the transaction) uniqueness (i.e., this proposal was not submitted in the past).

After the above checks comes the *transaction simulation step*. Endorsing peers invoke a relevant chaincode (as specified in the Tx proposal by the submitting client). The execution (as per specific arguments specified in Tx proposal) of this chaincode produces an output against the current state of the database (ledger). Without updating the ledger's state, the output of the Tx simulation is sent back in the form of *proposal response* to the client through the SDK. In Figure 1 this is shown by the second step.

3. Inspection of proposal response: After the above step the client-side application collects the responses from the endorsement step. Afterwards all the responses are cross checked (in terms of the signatures of the endorsing peers and the content of the responses) to determine if there are any disparities among the content of the responses. If the content of all the

---

[6]https://hyperledger-fabric.readthedocs.io/en/release-1.3/membership/membership.html
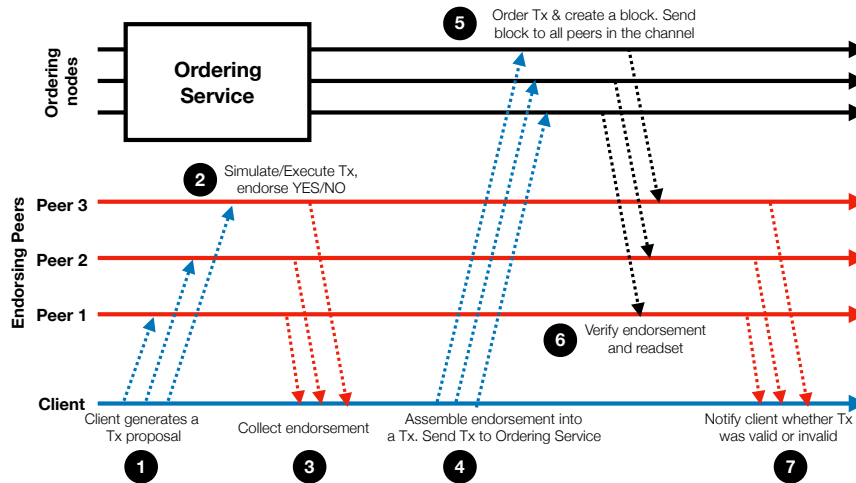
FIGURE 1 Hyperledger Fabric Protocol

responses are the same and according to the pre-defined *endorsement policy* (i.e., number of peers whose endorsements—in terms of their signatures—are necessary) then the client submits this Tx to the Ordering Service (more on it in the next step) that will in turn ultimately update the ledger's state as per the Tx simulation outcome in the last step.

It can also happen that in the Tx proposal, made in the last step, only the current state of the ledger was queried. In this case there will be no need to update a ledger's state and hence there is no submission to the Ordering Service by the client. In Figure 1 this is shown by step three.

4. Tx submission to the Ordering Service: The Ordering Service collects various Txs after the last step via various channels. This is step four in Figure 1 .

5. Tx ordering: Ordering Service *orders* various Txs according to their receiving times. This ordered set of Txs is then included in a block, specific to a channel, which will later be appended to the channel's ledger. This is covered by step five in Figure 1 .

6. Tx validation and committing: In this stage all the peers belonging to a particular channel receive a block containing Txs specific to this channel. Each peer then checks all the Txs in terms of their validity. Valid Txs are those that satisfy an endorsement policy. If the Txs pass the validity test then they are tagged as valid otherwise invalid in a block and then this block is finally appended to the ledger maintained by the peers of this channel. This is covered by step six in Figure 1 .

7. Ledger update notification: Finally, after the ledger update in the last step the client of the submitting Tx is notified about the validity or invalidity of the Tx that was included in the latest block of the channel's distributed ledger. This is step seven in Figure 1 .

2.2 | Ethereum

Ethereum is an open-source blockchain platform that can be used in a public or private setting and adds the provision of building decentralized value-transfer applications (dApps). Ethereum builds upon the Bitcoin system and introduces the concept of Ethereum Virtual Machine (EVM). EVM implements the Ethereum protocol (discussed next) which is responsible for handling the state transitions and associated computations without the involvement of third party intermediaries. The logic that powers a dApp is written in the form of a set of computer programs, so called *smart contracts*, that are being executed by the EVM. The concept of a smart contract can be understood as the algorithmic enforcement of policy agreements among, often mutually non-trusting, peers of a consortium (Wood, 2014). A set of smart contracts for a dApp, in turn, can be considered as a state machine, which is executed by the EVM of all the participating nodes. While the main Ethereum platform is a public blockchain network, the core platform software is open source and allows developers to configure and deploy a private and permissioned blockchain network (test networks) where only authorized nodes are allowed to participate.

2.2.1 | Ethereum Protocol

In Ethereum's ecosystem, there are two main types of entities namely: i) an externally owned account (EOA) with an address and a ii) smart contract written in a contract-specific programming language, such as Solidity, and is compiled into byte code which gets executed by an EVM [7]. In addition to an EOA, a smart contract is also assigned an address when it is deployed on the blockchain, however, it is used in a nuanced manner when compared to the address usage of an EOA. Anyone in possession of an EOA's address credentials can make a value-transfer transaction with another EOA by specifying its blockchain address. In such transfers the overall systems' state remains unchanged. However, in contrast, it is also possible for an EOA to make a transaction with a smart contract. In these types of transactions a specific function of a smart contract is invoked that usually triggers a state change in the overall EVM. It is also possible that one smart contract invokes a function of another smart contract possibly executing another associated EVM. It should be noted here that in Ethereum, each time a piece of code is invoked for execution (such as a smart contract's function) all the nodes of the network execute the same piece of code ensuring the correct execution of a program's logic. The state change, in turn, is then recorded in a decentralized manner in the form of mined (more on mining later), which are mutually agreed-upon, blocks ensuring immutability of such records. This way Ethereum enables a trusted and decentralized environment to automate a consortium-based application with trusted value-transfer transactions among the (potentially mutually non-trusting) peers of such a consortium.

Looking closely, a transaction-based state change in Ethereum's ecosystem can be understood with the help of Eq. 1 (Wood, 2014).

$$\sigma_{t+1} \equiv \Gamma(\sigma_t, T) \tag{1}$$

In Eq. 1, Γ represents a state-transition function and σ represents an arbitrary state. A state (σ), in general, can be defined as a collection of different types of records. As an example a state, in Ethereum, can consist of account balances, operations on a piece of data, specifics of agreements between two transacting parties etc (Wood, 2014).

Consensus engines in Ethereum: Presently Ethereum predominantly uses a *PoW*-based consensus engine called *Ethash*. PoW can be understood as a lottery-based consensus protocol introduced and popularized by Bitcoin (Nakamoto, 2008). The primary purpose of PoW is to avoid double spending of digital assets. PoW provides a trust guarantee to a payee which helps her establish the absence of a double spend of a unit of a digital asset. The actual proof is provided in the form of an integer so called a *nonce* which if, together with all the data contained within a block, is hashed produces an output string of characters which matches a predefined pattern. Such a pattern of hash outputs determines the computational difficulty of finding such a nonce. The process of finding a nonce is referred to as *mining*. More specifically the difficulty of mining a block (i.e., finding a relevant nonce) is determined by number of leading zeros of a hash output. The peer node of a blockchain network who finds a nonce is often referred to as a *miner*.

One of the other consensus engines currently in use in Ethereum's universe is called *Clique*. Clique engine makes use of a consensus protocol called *Proof-of-Authority (PoA)*. In contrast with PoW, PoA is computationally less expensive and eases the process of scaling a network. PoA-based consensus engines help to establish a private and permissioned version of a blockchain. In PoA, in contrast with PoW, the nodes who can have a say in appending new blocks to a blockchain are carefully chosen with known identities are referred to as *sealers*. In turn, the process of appending a new block to a blockchain running a PoA-based consensus engine is called *sealing*. Such nodes are also sometimes referred to as *authorities*. Specifically, sealing implies that if a block contains digital signatures of majority of authorities then it is considered as a valid block. However, PoA has proven to be less secure as compared to PoW[8] and that is the reason that it is predominantly being used by test networks and private chain setups for experimental purposes. In this paper we also make use of PoA-based Clique engine for our experiments. Since we have setup a local and private Ethereum blockchain we conjecture that security is not going to be an issue as far as our empirical analyses are concerned.

As a final note on Ethereum's consensus engines, we would like to briefly mention *Proof-of-Stake (PoS)*. Ethereum's community is planning an eventual migration from its PoW-based Ethash to a PoS-based consensus engine primarily because of network scalability issues prevalent in a computationally intensive PoW. PoS can be understood as a version of PoA where instead of an identity of a node the monetary value in the form of digital assets that a node owns in the network is at stake. The nodes with biggest stake in the network will have a bigger say when it comes to appending a new

---

[7]http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html
[8]https://medium.com/poa-network/exploiting-consensus-vulnerability-in-a-parity-client-to-hard-fork-an-ethereum-based-network-d2c368bf0bac

block to the chain. However, the idea of a PoS-based Ethereum is still in its infancy and does come with its fair share of problems most notably a mismatch of interests of nodes in the underlying network with equal stake in the network[9].

2.3 | Comparison of Hyperledger Fabric and Ethereum

The main differences between HLF and Ethereum derive from their inherent approaches of adopting a permissioned vs open blockchain paradigm respectively. HLF, as we see above in Section 2.1, has been developed mainly to promote a closed and permissioned version of blockchain with stringent confidentiality guarantees among a set of transacting member peers, by segregating them in different channels and associated chain code or a distributed application. On the other hand Ethereum proposes an open and generic flavour of blockchain, as a platform to build distributed applications and promote automation. The evolution of both of these platforms is heavily influenced by the original premises described above.

Consensus: In its first incarnation, i.e., in Bitcoin (Nakamoto, 2008), Proof of Work (PoW) based consensus mechanism was originally proposed to solve the double-spending problem (see Section 2.2 for details) in an asynchronous manner. Tracing the evolution ladder up one step, Ethereum can be considered as an evolved and more generic version of Bitcoin which popularized the concept of smart contracts with an associated Turing complete programming language. Since Ethereum builds upon Bitcoin's core, it is quite natural for it to inherit many of the Bitcoin's legacy traits: PoW-based asynchorous consensus mechanism, being public and permissionless are among the most notable ones. We did, however, see Ethereum making use of a Proof of Authority(PoA) based consensus engine in the last section to introduce a permissioned flavour of its blockchain. Such proposals are still in their infancy and we conjecture that they need time to get mature and widely adopted. In comparison, HLF has adopted a more hierarchical and synchronous approach in achieving consensus and appending new blocks to a blockchain. This is mainly due to the Ordering Service (as described in Section 2.1), which is responsible to timestamp the transactions and avoid double-spends in the network. The same Ordering Service introduces, up to some extent, centralization in the way HLF achieves consensus on a particular state of a blockchain. If a single instance of an Ordering Service is used then it raises the concern for a single-point-of-failure as well. However, single-point-of-failure is not a major concern in the asynchronous PoW-based blockchains such as Ethereum. In our experiments we deploy only one instance of Ordering Service to keep the setup simple and perform experiments to evaluate HLF's core architecture rather than auxiliary problems like the single-point-of-failure.

Architecture: • *Public vs private*: HLF and Ethereum's design differ in their approach in addressing the pool of usecases in public and enterprise domain. Ethereum was designed to be completely decentralised setup in public domain with all nodes in the network being equal and then slowly being adopted to private or enterprise usecases. Ethereum also has a native currency called *ether*. HLF was designed focusing to solve enterprise usecases, rather than providing a public blockchain platform. It does not have inbuilt currency, but allows for creation of coins on top of the network through chaincode.

   • *Extent of Decentralisation*: Not all nodes are equal in HLF's network. There are set of privileged nodes - *endorsers, orderer, membership providing service*, that have more control and access than other nodes - *peers*, making the system more centralised as compared to Ethereum. Further, in contrast with Ethereum's flat approach to reaching consensus, HLF has two main interlinked levels where consensus is reached in an hierarchical order. HLF follows an order of *execution of the chaincode and validation of transaction with endorsement policy by multiple endorsers, order the transaction and put in a block*. More specifically, the first level involves satisfying an endorsement policy where signatures from a pre-set number of endorsing peers are collected for a transaction proposal (see Section 2.1.1 for details). The second step occurs at the level of Ordering Service, which orders the transactions in block. On the other hand, Ethereum follows an order where *each node should check and execute the transaction, generate/propose a block, validate the block and broadcast it*. The initial check happens by the node where the transaction is submitted, and not by a set of endorsing nodes as in case of Hyperledger Fabric. This approach of Ethereum leads to higher collision during reaching consensus, as well as to higher chances of forks in a large network. Ethereum handles forking using GHOST protocol. While HLF's design makes it less prone to forking and with careful design of endorsement policies, the forking problem can be completely avoided. At the ordering level, HLF provides a modular plug-and-play approach where a consensus mechanism can be chosen from a set of available mechanisms that can

---

[9]https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256

be deployed pertaining to a specific use case at hand. Currently the default option is a single orderer setup, while Kafka based multi-orderer setup and PBFT techniques are also popular.

- *Confidentiality*: When it comes to the permissioned blockchain paradigm, HLF's approach to implement a private and permissioned blockchain is more exhaustive and fine grained as compared to Ethereum. As we discussed in Section 2.1, channels and MSPs implement an intricate and hierarchical, much akin to an actual organization, permissioned infrastructure with clearly defined roles and security clearance for different entities in the network. In Ethereum, the closest deployment to a permissioned blockchain would be adopting a PoA-based consensus engine, which is still quite a flat approach as compared to HLF's MSPs and channels with their dedicated ledgers at different levels with associated set of chain codes.

3 | CASE STUDY: BLOCKCHAIN IN THE QMPSU MESH NETWORK

The Quick Mesh Project (qMp) [10] develops a firmware based on OpenWrt Linux with the aim to ease the deployment of mesh networks by the users who are willing to interconnect in an area, and pool their Internet uplinks (Cerdà-Alabern u. a., 2013). qMp was initiated in 2011 by a few Guifi.net activists.

The qMp firmware has enabled to deploy several mesh networks with actual end-users (e.g., more than 250 active locations, typically households) in several parts surrounding the city of Barcelona[11]. At the time of this writing, there are 10 different neighbourhood mesh networks, and the largest (Sants-UPC or QMPSU) has 85 operational nodes. In that network, there are two gateways that connect the QMPSU network to the rest of guifi.net and the Internet. Users join the mesh by setting up *outdoor routers* (i.e., antennas) that automatically establish router-to-router links. The outdoor routers are connected through Ethernet to a home network, with an indoor AP (access point) where the edge devices and services are running: home-servers such as Raspberry Pi's or Cloudy devices (Baig u. a., 2018).
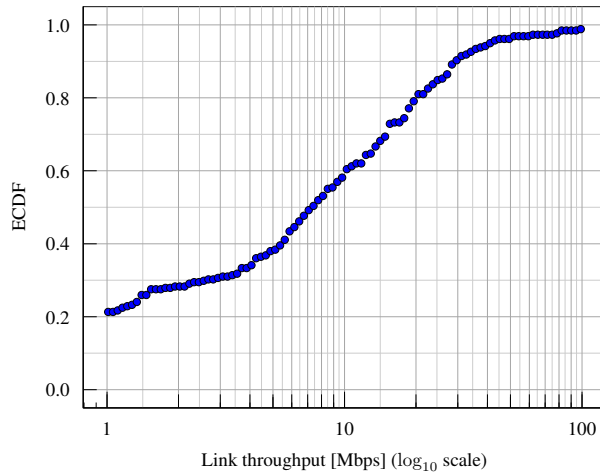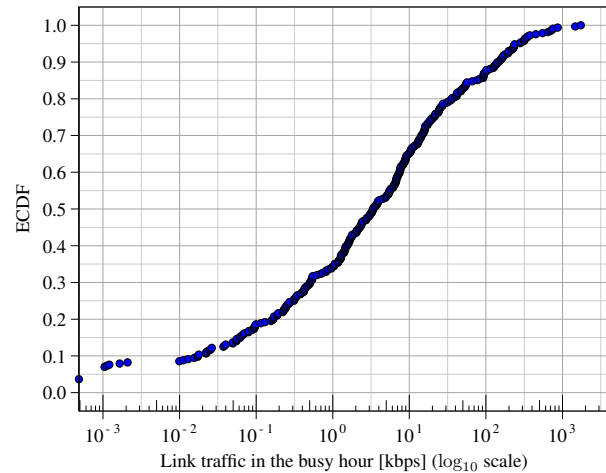

FIGURE 2 Bandwidth ECDF

FIGURE 3 Traffic ECDF

Network performance: We monitored the QMPSU mesh network for a period of one month. We took hourly captures from the network for the entire month of March 2018. Figures 2 and 3 depict the bandwidth and traffic distribution of all the links in the network. Figure 2 shows that the link throughput can be fitted with a mean of 13.6 Mbps. At the same time Figure 2 reveals that 60% of the nodes have 10 Mbps or less throughput. Figure 3 demonstrates that the maximum per-link traffic in the busiest hour is 1736 kbps. We observed that the resources are not uniformly distributed in the network. There is a highly skewed bandwidth and traffic distribution.

---

[10]http://qmp.cat/Overview
[11]http://dsg.ac.upc.edu/qmpsu

Node deployment: Based on the network measurement analysis we strategically deployed 10 Raspberry Pi (RPi3) devices on the outdoor routers to cover the area of the QMPSU network as presented in Figure 4 . We use our previous work (Selimi u. a., 2018a) on service placement to determine nodes in the network. In this set, we cover nodes with different properties: with higher bandwidth (Selimi u. a., 2018a), nodes that are highly connected (i.e., with high degree centrality) (Coimbra u. a., 2018), nodes acting as bridges (with high betweenness centrality), and nodes not well connected. After the nodes were chosen, we deployed 10 RPi boards in the community users home.
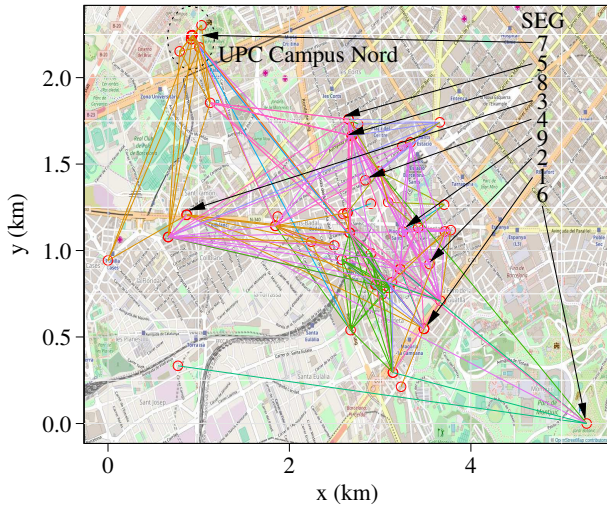

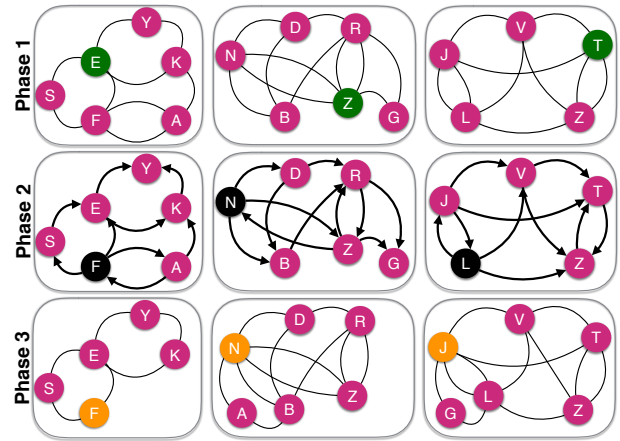**FIGURE 4** Topology of the deployed nodes in Barcelona.

FIGURE 5 BASP Phases

BASP: Bandwidth Aware Service Placement In order to determine the best nodes in the QMPSU network where to place the Hyperledger Fabric and Ethereum components, we use the BASP heuristic from our previous work (Selimi u. a., 2018a) and Selimi u. a. (2017). The BASP (Bandwidth and Availability-aware Service Placement) service placement heuristic takes into account the bandwidth of the network, node availability and CPU of the nodes to do smart node selection/placement. BASP is executed every single time a (new) service or node deployment is about to be made. BASP runs in three phases. In the first phase, BASP partitions the network topology into k (maximum allowed number of service replicas) and removes the nodes that are under the pre-defined availability threshold. In this phase, BASP uses the naive K-Means partitioning algorithm in order to group nodes based on their geo-location. The idea is to get back clusters of nodes that are close to each other. In the second phase, BASP estimates and computes the max bandwidth of the nodes in the network. The bandwidth between two nodes is estimated as the bandwidth of the link having the minimum bandwidth in the shortest path. In the third phase, BASP re-assigns nodes with higher CPU and availability to the selected clusters formed in the second phase. Figure 5 demonstrates the phases of the BASP.

4 | EVALUATION

We evaluate the performance of Hyperledger Fabric and Ethereum in a wireless mesh setting for parameters like CPU load, memory consumption and transaction latency with varying number of transactions as well as varied placement strategies of blockchain nodes. For this, we setup a testbed network comprising RPi3 boards in the QMPSU network[12]. Each RPi3 board runs either a Ethereum node or a component of Hyperledger Fabric (see Section 2.1 and Section 2.2 for details). The RPi3 boards have 1.2GHz 4 core ARM cortex A53 processor, a RAM memory of 1GB and run *raspbian-stretch* OS. In parallel, we also deployed a similar setup in the lab environment (for performance comparison purposes) and evaluated the performance in both environments.

---

[12]http://dsg.ac.upc.edu/qmpsu/index.php

We perform identical experiments in permissioned blockchain network setup for both Ethereum and Hyperledger Fabric. The scenario of a typical experiment in both Ethereum as well as Hyperledger Fabric is - a client sends *N* balance transfer request between two parties/accounts to a node in the blockchain network. In Ethereum *sendTransaction* operation is used to transfer funds between two accounts. While in the case of Hyperledger Fabric, a chaincode, deployed in endorsers, is executed to transfer funds between two parties e.g., Alice sends 10 tokens to Bob [A, B, 10].

4.1 | Evaluation Metrics

We consider the following parameters to evaluate the targeted blockchain platforms:

1. **Transaction Latency:** This is the time taken to complete a set number say N = 100 transactions. Transaction latency is measured differently in Hyperledger Fabric and Ethereum platforms.

   - In Hyperledger Fabric, this is the time taken to endorse and to commit a transaction to the ledger. We plot both time to endorse and time to commit in our plots.
   - In Ethereum, this is the total amount of time to execute, seal and confirm the transaction. We consider a total of 12 confirmations as successful commit of a transaction(Weber u. a., 2017).

2. **CPU and Memory Utilization:** Load on CPU and memory is measured for nodes when idle and during transactions.

   - In Hyperledger Fabric, CPU load and memory usage are measured for endorser, orderer and committing peers.
   - In Ethereum, CPU load and memory usage are measured for the sealer node and non sealer node to which the transaction is fired to.

4.2 | Hyperledger Fabric Setup

In our experiments, we deploy a HLF blockchain network[13] consisting of a single organizational entity. All the transactions happen among the members of this single organization. The HLF components, namely peer (we deploy multiple instances of this component), orderer, and client are deployed in different RPi3 boards connected to each other in QMPSU wireless mesh network. We perform experiments by placing different Hyperledger Fabric components at different physical (RPi3) nodes and by varying the number of peers from 1 to 4. We evaluate the setup in QMPSU network comparing transaction latency for 100 transactions fired in parallel when components of HLF are placed randomly in the network and in nodes with best connectivity (according to BASP). For the best connectivity deployment, we also evaluate transaction latencies in HLF for a 2 peer setup when the block size is varied from 10 to 100 transactions per block. Our experiments comprise of 5 runs (taken in different time slots) and the presented results are averaged over all the runs.

4.2.1 | Hyperledger Fabric Experimental Results

Table 1 lists the transaction completion time (referred to as *Time-to-Commit (TCC)*) for 100 transactions, initiated in parallel, between the two peer nodes in the lab environment and in the QMPSU network respectively with block sizes ranging from 10 to 100 transactions per block. It can be observed that, as the block size increases, the transaction completion time increases in the QMPSU network.

Transaction Latency: In Hyperledger Fabric, transaction latency is defined as the total time taken to endorse and to commit a transaction to the ledger. Figure 6 shows the comparison of transaction latency observed for two different placements of HLF ordering service. We measure transaction latency when the HLF ordering service is placed randomly in the network (Random) and when it is placed at the node chosen with a heuristic that considers the node with higher bandwidth and degree centrality (BASP) (Selimi u. a., 2018a). The results of Figure 6 are obtained when a client initiates 100 transactions sequentially. This Figure reveals that the gain brought by BASP, for the case when we have one endorser in the network, is a 30.8% reduction. For the case when we have four endorsers in the network, the gain of BASP over Random is 24% reduction. Further, we can deduct from Figure 6 that in the QMPSU network it takes around 1 second for a single transaction to be appended to the distributed ledger.

---

[13]https://github.com/anirudhkabi/HLF

Block Size	Time-to-Commit (Lab)	Time-to-Commit (QMPSU)	# of Txs
10	33.4 s	64.2 s	100
20	35.0 s	69.7 s	100
50	39.2 s	75.3 s	100
100	45.3 s	84.8 s	100

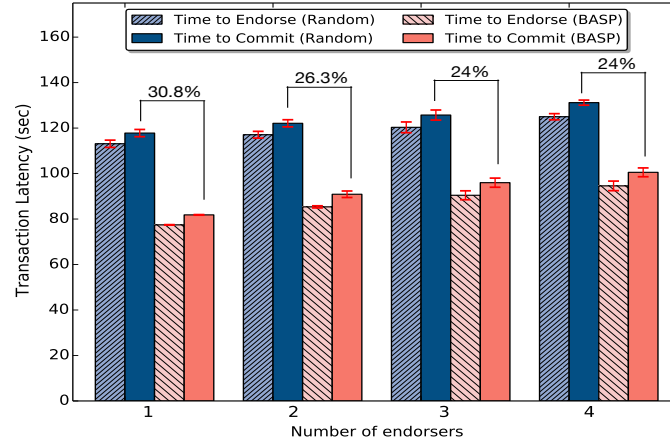**TABLE 1** Transaction delivery time (parallel transactions).

FIGURE 6 Transaction latency (QMPSU)

Resource Consumption: Figure 7 shows CPU utilization by various components of the HLF network namely: an orderer, a client and two peers (an endorser and a committer). CPU utilization of all nodes is monitored for a time period of 60 seconds during which 100 transactions are fired in parallel (by the client) and all the transactions are completed. 100 parallel transactions took around 40 seconds to complete. We chose to monitor the nodes for a time period of 60 seconds to show idle phase usage and busy phase usage of each node. In the graph, transactions are initiated at the 11th second and all the transactions get completed at 50th second. It can be observed that the endorser is the node with the highest CPU utilization whereas the orderer utilizes the least of CPU.

Figure 7 shows that, for 100 transactions initiated at the same time, the endorser's maximum CPU utilization reaches 96%. The maximum CPU utilization is 81% for the committer while it is 71% for the orderer. The reason that the endorser has the highest CPU consumption, among other HLF components, is because of the chaincode execution at the endorsing peer, which does not happen at the committer and the orderer.

In HLF, each component usually runs in it own Docker container[14]. The chaincode container executes the chaincode for each incoming transaction which is something that does not happen at the committer node. When multiple transactions take place in parallel, concurrent execution of the chaincode happens for all transactions thus, in turn, increasing the load on the endorsing peer. With 100 parallel transactions, we observe that the CPU load reaches 96% at the endorser. However, the load on each endorser can be reduced by deploying multiple endorsers in the network. The load on different endorsers can be balanced by designing a suitable endorsement policy and devising a strategy at the client to request endorsements from different set of endorsers each time a transaction is initiated.

Similarly, memory usage is the highest by the endorser and the least by the orderer. Memory usage of committing peer falls in between of endorsing peer and the orderer. At the orderer and the committing peers, memory usage remains almost the same level between the idle phase and during transaction execution. Memory usage at the orderer mostly falls in the range of 57%-58% while the memory usage at the committer is in the range of 57%-60%. At an endorsing peer the memory usage increases during transaction execution as the execution of a chaincode also takes place at the same time. The memory usage by the endorser is about 60% during the idle phase and reaches to a maximum of 65% during the chaincode execution.

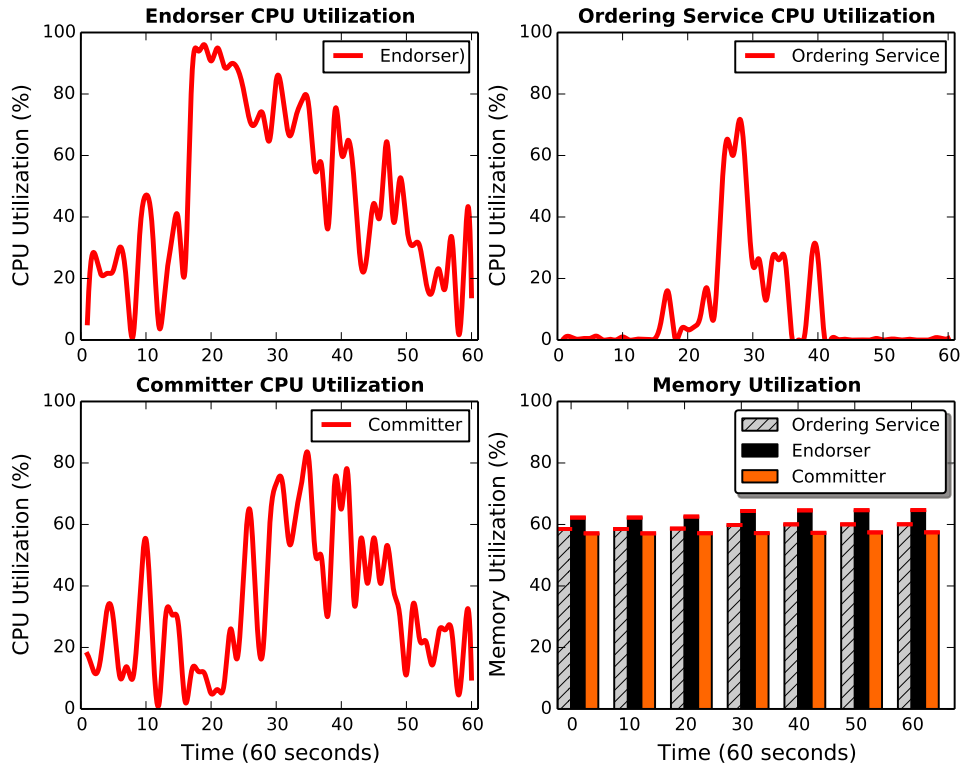
[14]https://www.docker.com/what-docker

FIGURE 7 CPU and memory utilization

4.3 | Ethereum Setup

In order to evaluate the Ethereum platform, we construct a synthetic application as a cash (Ether) transfer application where Ether token is transferred from one account to another. We create two accounts i.e., source and the target account, and cash is transferred between accounts by calling *sendTransaction*, an inbuilt function available in Ethereum implementation *geth* to transfer funds (in Ether) between two accounts. For Ethereum, Proof of Authority consensus mechanism **Clique**, implemented in geth is used and experiments are performed with 1, 2 and 4 sealers/validators. The results of each experiment are averaged over 5 independent runs.

We deploy a PoA-based Ethereum network with a blocktime of 5 seconds for our experiments as PoA consensus mechanism is more suitable to permissioned blockchain networks than the default PoW consensus mechanism. There are two kinds of nodes in a PoA network - *Validators or Sealers*, who sign and create new blocks; - *Non-Validators or Clients*, who do not have the authority to create new blocks and are mostly deployed in the network as interface for users to connect to blockchain network and submit transactions. We perform experiments in both lab and QMPSU for various configuration as listed below.

- **Baseline-lab setup**: 2 validator nodes co-located in the same host (Minix Device[15]). Transactions are generated from within the host, and sent to one of the validators through Inter-process communication (IPC). Experiments are performed measuring transaction latency for 1, 10, 100, 1000 and 10000 transactions fired in parallel.

- **Mesh-lab setup**: In order to evaluate the effect of the mesh, we perform the same experiments of Baseline-lab setup, but launching the transactions using WebSockets, from a powerful (Desktop machine) node that is located 2 (short) mesh hops away from the Minix Device.

- **QMPSU setup**: We perform experiments in QMPSU in line with experiments performed for Hyperledger Fabric in QMPSU. To mimic single organisation scenario of Hyperledger Fabric, we authorise only one sealor/validator account and run multiple instances of the validator by varying number of validator instances from 1 to 4. We compare transaction

---

[15]http://minix.com.hk/

latency for 100 transactions fired in parallel when validator instances are placed randomly and when validator instances are placed in RPi3 nodes with the BASP heuristic. We also vary number of transactions from 1 to 1000 and record transaction latency. Apart from transaction latency, we measure CPU load and memory usage of validator and non-validator nodes when idle and busy.

4.3.1 | Ethereum Experimental Results

Transaction Latency: In Ethereum, transaction latency is measured in multiple ways. Intuitively, transaction latency is the time between firing a transaction to the time it gets sealed. However, there is a significant probability that the mined block may not end up in the chain due to forking. Therefore, as mentioned earlier, it is a standard practice to consider confirmation of next 12 blocks as finality for a transaction. In our experiments we measure both *sealing time* and *completion time*, which we define as the time from firing of transaction upon receiving the 12th confirmation, with the transaction under consideration being part of the chain.
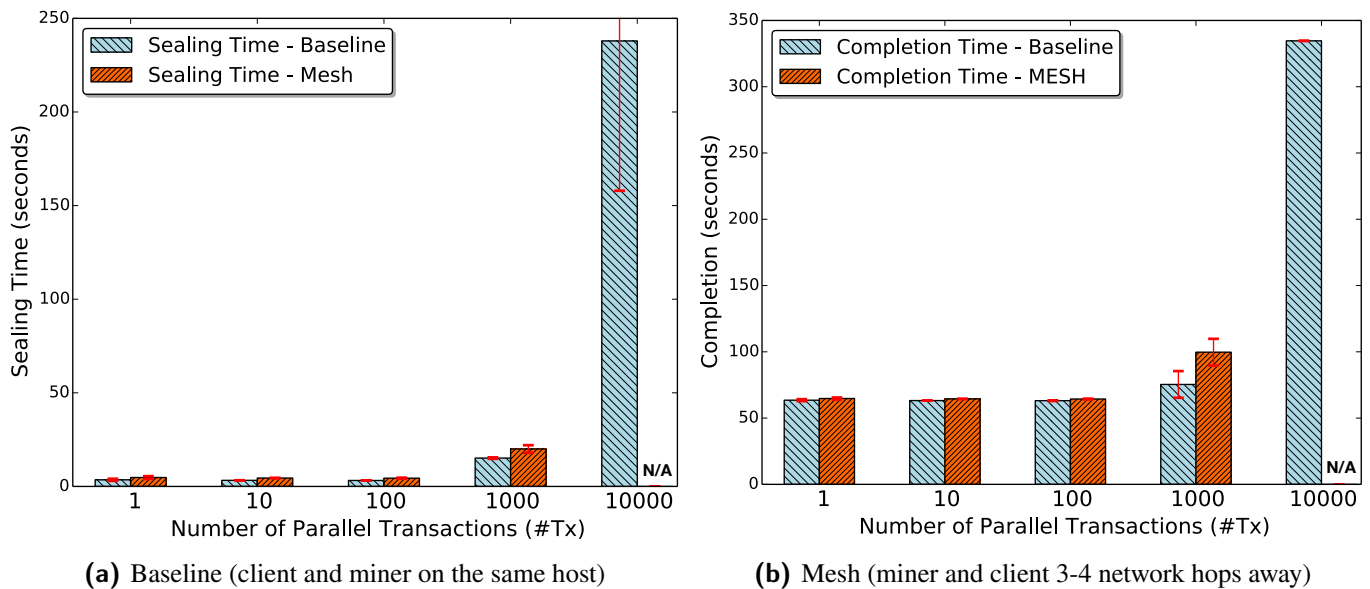

(a) Baseline (client and miner on the same host) **(b)** Mesh (miner and client 3-4 network hops away)

FIGURE 8 Sealing and Completion Time for Baseline/centralized and Mesh/distributed environment.

As an exploratory study, before jumping into QMPSU, we setup PoA based Ethereum private network following the Baseline-lab and Mesh-lab setups, measuring the sealing and completion time for 1, 10, 100, 1000 and 10000 transactions. Figures 8 a and 8 b compare the transaction latencies for the two different setups. As expected, the Baseline setup shows lower latency than the Mesh one. Considering a blocktime of 5 seconds, then in normal situations we expect a $sealingTime \in [0, blockTime] \equiv [0, 5]$ and that $completionTime >= 12 * blockTime = 60$. For both the sealing and completion time we observe that they show a normal behaviour for up to 100 parallel transactions. However, at 1000 transactions, we already note increased delays in both the cases. It is interesting to point out here that an increase of 5 seconds can be translated as a delay for the next block to be sealed. At 10000 transactions, the multinode setup is completely saturated and does not respond. We observed that the majority of the transactions do not manage to get included in next 50 blocks and are dropped, which in turn causes a timeout of the connector library, exiting with an exception. Even with the baseline setup, at 10000 transactions, there is a large delay in getting the responses of block being sealed and confirmed. As a result, the rest of the experiments are performed with a maximum of 1000 parallel transactions.

After the exploratory study, we performed experiments in QMPSU, obtaining the Figures 9 a and 9 b, that show respectively the sealing and completion time for 1, 10, 100 and 1000 transactions fired in parallel with 1, 2 and 4 validator instances. Considering a blocktime of 5s and similarly to the exploratory experiments, Figure 9 a shows that up to 100 transactions, all the transactions are verified and sealed in 1 block. Beyond 100 transactions, the number of blocks needed to accommodate all transactions fired increases beyond 3 blocks. In the case of 1000 transactions, that are accommodated in more than 3 blocks, the

total sealing time increases with more number of validator instances. The delay may be attributed to the latency generated by broadcasting the pending transactions to different validator nodes, once the current validator node reaches the block gas limit and cannot accommodate anymore transactions in the block. As far as the completion time is concerned, plotted in 9 b, we observe a similar behaviour to the sealing time. Between 1 and 100 parallel transactions fired, completion time is almost constant as empty blocks are sealed irrespective of number of transactions fired earlier, while shows an increased value for 1,000 transactions.
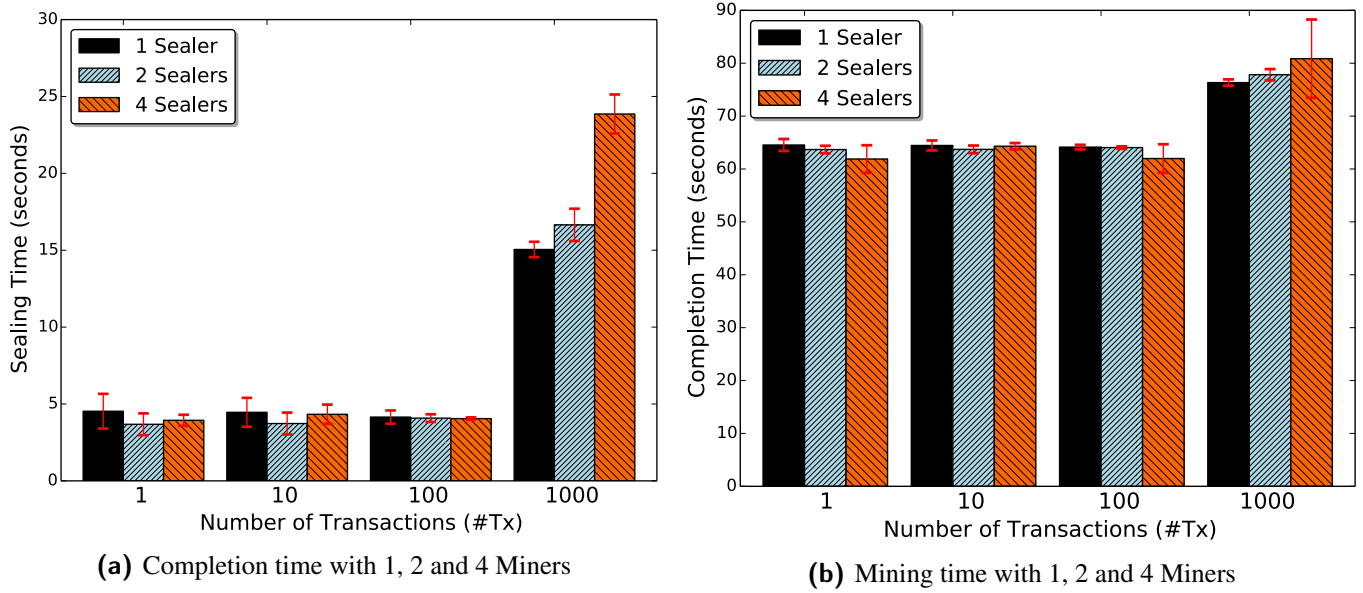

(a) Completion time with 1, 2 and 4 Miners

(b) Mining time with 1, 2 and 4 Miners

FIGURE 9 Sealing and Completion time for 1, 2 and 4 Sealers. 1, 10, 100 and 1000 transactions used.

Placement: Figure 10 a and Figure 10 b depicts the sealing and completion time for different number of sealer nodes in the network, when nodes are placed randomly and with the BASP heuristic respectively. The figures reveal that BASP outperforms the Random placement when using up to 4 sealer nodes. Moreover, as the number of sealer node increases, the gain tends to increase accordingly. For instance, when having up to 4 sealer nodes, Figure 10 a shows that the gain brought by BASP over random is 3 seconds which is 40% improvement. The same thing happens with completion time in Figure 10 b, where the gain brought by BASP over random is 12 seconds which is 26% of improvement.

These figures demonstrate the importance of the sealer node location in the network. In a challenging environment such as wireless mesh network, the placement heuristics that are agnostic to the state of the underlying network may lead to important inefficiencies. Our result demonstrates that placement of sealer nodes can become even more crucial when number of transaction is higher (e.g., 1,000, 10,000 etc).

Resource Consumption: In order to understand the resource usage of the participating nodes, we measured CPU Load and memory usage in sealer and non-sealer nodes when 100 and 1000 transactions are fired, as plotted in Figure 11 . We record the measurement from the time we fire transactions to the time the transactions are considered completed (i.e, 12 confirmations in Ethereum). The procedure we follow here is that, we fire transactions to non-sealer node; non-sealer node broadcasts the transactions to sealer node where it is sealed. We follow this strategy to get an idea of load generated by independent processes like accepting transactions and sealing. We observe that when 1000 transactions are fired in parallel, the non-sealer node is saturated heavily in all the 4 cores of the RPi3 board. Even with 100 transactions, the CPU load on non-sealer node is pretty high. While the validation process loads the CPU only moderately for both 100 and 1000 transactions. It is also demonstrated that the non-sealer has higher memory consumption compared to the sealer node in both cases. However, the maximum memory usage is still below 20% and is not a bottleneck in the blockchain network. As expected, the sealing process, following a PoA scheme, is not demanding in terms of resources. On the other hand, accepting transactions and forwarding them to the sealer nodes, as the non-sealer node of our experiment does, seems to be very resourceful, and can saturate the node.
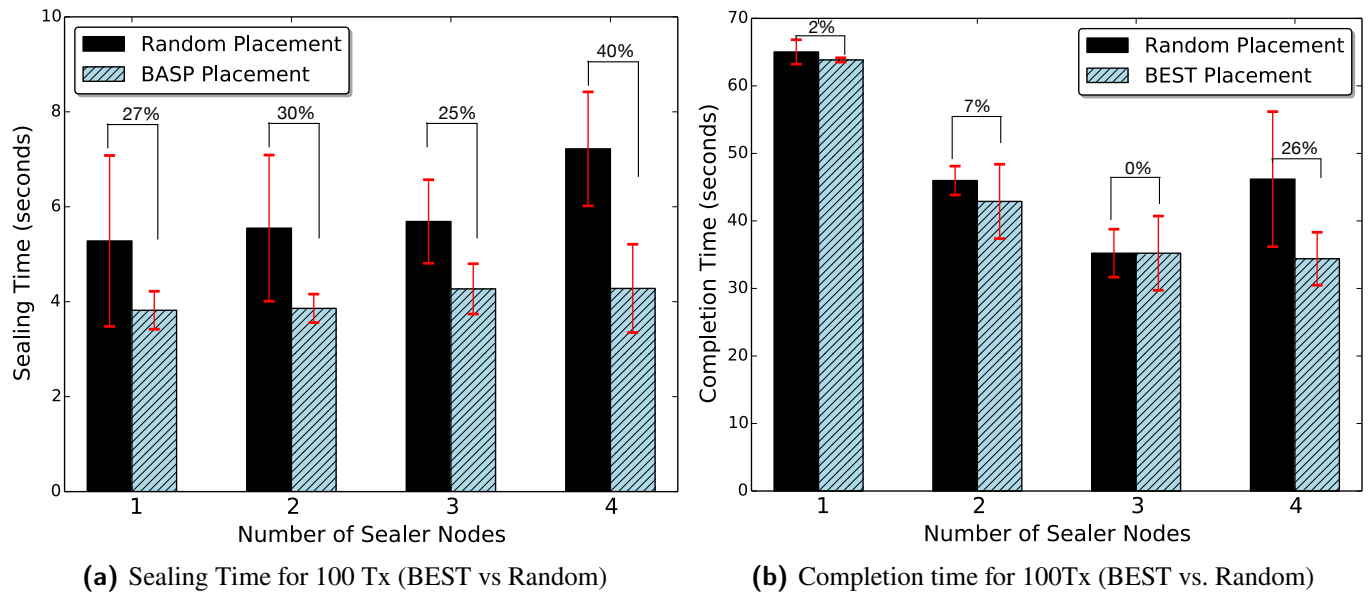
(a) Sealing Time for 100 Tx (BEST vs Random)

(b) Completion time for 100Tx (BEST vs. Random)

FIGURE 10 Sealing and Completion Time for 100 Transaction (BASP vs. Random)

4.4 | Discussion

Hyperledger Fabric: As we observed in our experiments, in terms of resource consumption, the endorser nodes can prove to be a bottleneck. We believe that this bottleneck is because of the execution of an additional chaincode container at each endorsing node. In our current study we only considered one endorser node to study the resource utilization with a simple endorsement policy encoded in the corresponding chaincode. It might get more complicated when we consider more than one endorser, and more sophisticated endorsement policies. However, as discussed in the paper, if done right it can actually improve performance. In addition to this, the actual distribution of endorsing peers in a production network, such as QMPSU, might also affect the network performance (both in terms of CPU utilization and transaction latency). Therefore we advise caution when in designing an endorsing policy that is also cognizant of the underlying network infrastructure (i.e, topology, capacity, performance, etc), especially in the resource constrained nature of CMNs. A deployment strategy and an apt endorsement policy balancing the load on various endorsers in the network can improve the performance of the blockchain network and allow scaling of the blockchain network without forming a bottleneck. As far as the orderers are concerned, horizontal scaling by adding more nodes is possible, nevertheless, this would need some sort of mechanism for syncing between instances. For instance, it is possible to have multiple instances of ordering service nodes all connected to a single fault tolerant service (Kafka) that would do the ordering (crash fault tolerant).

Ethereum: The results presented earlier concerning Ethereum show that it can be used successfully as private permissioned blockchain in a mesh environment, using PoA consensus. Nonetheless, there are various parameters to be adjusted and bottlenecks that need to be discussed. Unlike HLF, in Ethereum there is no clear horizontal scaling pattern. While having a lot of sealers could balance the incoming transactions, the transaction throughput is largely affected by the hardware resources like CPU and memory of the nodes who accept the transactions and less affected by the number of nodes . This, depending on the frequency of transactions generated, can be a significant issue for mesh like environments, since the hardware used is usually low-power/low-cost devices. Moreover, the broadcasting of the pending transactions between the sealers can become problematic over non-stable mesh connections, especially between remote nodes, or nodes connected with lossy links. This situation could also deteriorate by an increased number of nodes and small blocktimes, leading to higher frequency and higher number of message exchanges between the sealers. On the other hand, these effects could be moderated by utilising smart placement algorithms like BASP, which would play a significant role in avoiding network saturation, by placing the sealers in locations that would minimise the overhead of the blockchain. Finally, while we deploy multiple clones of one sealer, other approaches are possible, like having multiple sealer accounts, considering that a minimum of $N/2 + 1/$ instances of them are always available(Developers).
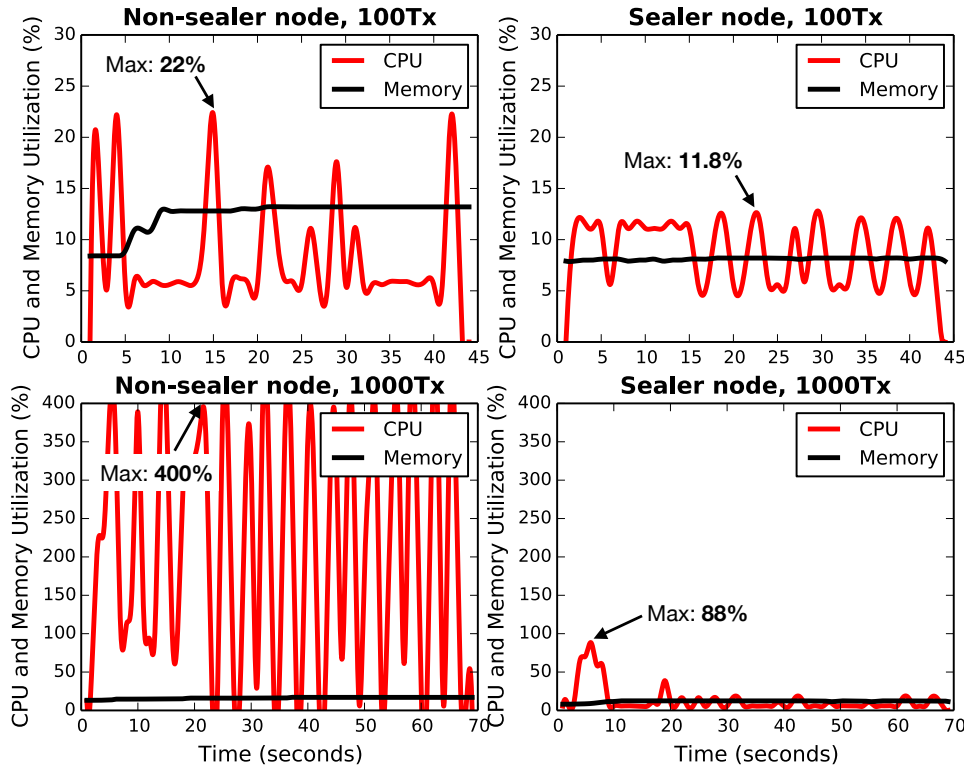
FIGURE 11 CPU and Memory utilization with 100 and 1000 Tx

5 | RELATED WORK

The work of Suporn et al. (Pongnumkul u. a., 2017) presents performance analysis of Hyperledger Fabric and Ethereum as private blockchain platforms with varying number of transactions. They conduct their experiments in Amazon AWS EC2 instances. Their assessment shows that Hyperledger Fabric consistently outperforms Ethereum across all evaluation metrics such as execution time, latency and throughput. Further, they claim that both platforms are still not competitive with current database systems in terms of performance when using high workloads. The work in (P. u. a., 2018) discusses various consensus protocols used in blockchain and comparative analysis of Hyperledger Fabric and Ethereum. The study (Scherer, 2017) compares the public blockchain with permissioned blockchain and discusses the trade-offs among decentralization, scalability and security in the two approaches. Sousa et al. (Sousa u. a., 2017) present the design, implementation and evaluation of a BFT ordering service for Hyperledger Fabric based on the the BFT-SMART state machine replication/consensus library. Their results show that Hyperledger Fabric with their ordering service can achieve up to ten thousand transactions per second and write a transaction irrevocably in the blockchain in half a second, even with peers distributed over different continents. The Blockbench (Dinh u. a., 2017) is a framework for analyzing private blockchains. It serves as a fair means of comparison for different platforms and enables deeper understanding of different system design choices. They use Blockbench to conduct comprehensive evaluation of three major private blockchains: Ethereum, Parity[16] and Hyperledger Fabric. Their results demonstrate that these systems are still far from replacing the current database systems in traditional data processing workloads. In contrast to most of the works mentioned in this section, we specifically consider the implications of deploying the blockchain paradigm to a, still in use, production environment such as that of CMNs.

---

[16]https://www.parity.io/

6 | CONCLUSION

The missing ingredient for widespread adoption of decentralized access networks (such as community mesh access networks) has always been the issue of economic sustainability. In this paper, we take on the issue of addressing trustworthy economic sustainability by proposing the need for an economic substrate built using blockchain that can keep a record of the transactions related to the contributions (of nodes, links, Internet gateways, maintenance), consumption of communication network's resources as its economic compensation in a decentralized and trusted manner. The evaluation of the Hyperledger Fabric and Ethereum blockchain deployment in a centralized network i.e. laboratory, and a decentralized network i.e. in a real production mesh network, gives us an understanding of the performance, overhead, influence of the underlying network, and limitations of the two platforms. The results show critical aspects that can be optimized in a Hyperledger Fabric and an Ethereum deployment, in the perspective of decentralized networks, where several components can prove to be bottlenecks and therefore put a limiting effect on the rate of economic transactions in a mesh network. Future work will expand the evaluation to a wider range of hardware and network configurations considering real and synthetic transaction traces. We will also consider the influence of the execution of non-trivial smart contracts, with a more realistic design of an endorsement policy (chaincode).

References

[Akyildiz u. a. 2005]   AKYILDIZ, Ian F. ; WANG, Xudong ; WANG, Weilin: Wireless mesh networks: a survey. In: *Computer networks* 47 (2005), Nr. 4, S. 445–487

[Androulaki u. a. 2018]   ANDROULAKI, E. u. a.: Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In: *ArXiv e-prints* (2018), Januar

[Androulaki u. a. 2018]   ANDROULAKI, Elli ; BARGER, Artem ; BORTNIKOV, Vita ; CACHIN, Christian ; CHRISTIDIS, Konstantinos ; DE CARO, Angelo ; ENYEART, David ; FERRIS, Christopher ; LAVENTMAN, Gennady ; MANEVICH, Yacov ; MURALIDHARAN, Srinivasan ; MURTHY, Chet ; NGUYEN, Binh ; SETHI, Manish ; SINGH, Gari ; SMITH, Keith ; SORNIOTTI, Alessandro ; STATHAKOPOULOU, Chrysoula ; VUKOLIĆ, Marko ; COCCO, Sharon W. ; YELLICK, Jason: Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In: *Proceedings of the Thirteenth EuroSys Conference*. New York, NY, USA : ACM, 2018 (EuroSys '18), S. 30:1–30:15. – URL http://doi.acm.org/10.1145/3190508.3190538. – ISBN 978-1-4503-5584-1

[Baig u. a. 2016]   BAIG, Roger ; DALMAU, Lluís ; ROCA, Ramon ; NAVARRO, Leandro ; FREITAG, Felix ; SATHIASEELAN, Arjuna: Making community networks economically sustainable, the guifi. net experience. In: *Proceedings of the 2016 workshop on Global Access to the Internet for All* ACM (Veranst.), 2016, S. 31–36

[Baig u. a. 2018]   BAIG, Roger ; FREITAG, Felix ; NAVARRO, Leandro: Cloudy in guifi.net: Establishing and sustaining a community cloud as open commons. In: *Future Generation Computer Systems* (2018), 01/2018. – URL http://people.ac.upc.edu/leandro/docs/cloudy.pdf. – ISSN 0167-739X

[Cerdà-Alabern u. a. 2013]   CERDÀ-ALABERN, Llorenç ; NEUMANN, Axel ; ESCRICH, Pau: Experimental Evaluation of a Wireless Community Mesh Network. In: *Proceedings of the 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. New York, NY, USA : ACM, 2013 (MSWiM '13), S. 23–30. – URL http://doi.acm.org/10.1145/2507924.2507960. – ISBN 978-1-4503-2353-6

[Coimbra u. a. 2018]   COIMBRA, Miguel E. ; SELIMI, Mennan ; FRANCISCO, A. P. ; FREITAG, Felix ; VEIGA, Luís: Gelly-Scheduling: Distributed Graph Processing for Service Placement in Community Networks. In: *33rd ACM/SIGAPP Symposium On Applied Computing (SAC 2018)*, ACM, April 2018

[Developers]   DEVELOPERS, Ethereum: *Clique PoA protocol & Rinkeby PoA testnet.* https://github.com/ethereum/EIPs/issues/225

[Dinh u. a. 2017]   DINH, Tien Tuan A. u. a.: BLOCKBENCH: A Framework for Analyzing Private Blockchains. In: *Proceedings of the 2017 ACM International Conference on Management of Data.* New York, NY, USA : ACM, 2017 (SIGMOD '17), S. 1085–1100. – URL http://doi.acm.org/10.1145/3035918.3064033. – ISBN 978-1-4503-4197-4

[ITU]   ITU: *International Telecommunications Union, ICT Facts and Figures 2016.*   http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2016.pdf

[Nakamoto 2008]   NAKAMOTO, Satoshi: Bitcoin: A peer-to-peer electronic cash system. (2008)

[Neumann u. a. 2016]   NEUMANN, Axel ; LÓPEZ, Ester ; CERDÀ-ALABERN, Llorenç ; NAVARRO, Leandro: Securely-entrusted multi-topology routing for community networks. In: *2016 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)* IEEE (Veranst.), IEEE, 2016. – URL http://dsg.ac.upc.edu/sites/default/files/dsg/07429050.pdf

[P. u. a. 2018]   P., Sajana ; M., Sindhu ; SETHUMADHAVAN, M.: On Blockchain Applications: Hyperledger Fabric And Ethereum. In: *International Journal of Pure and Applied Mathematics* (2018), 03/2018. – URL https://acadpubl.eu/jsi/2018-118-18/articles/18c/84.pdf. – ISSN 1314-3395

[Pongnumkul u. a. 2017]   PONGNUMKUL, S. ; SIRIPANPORNCHANA, C. ; THAJCHAYAPONG, S.: Performance Analysis of Private Blockchain Platforms in Varying Workloads. In: *2017 26th International Conference on Computer Communication and Networks (ICCCN),* July 2017, S. 1–6

[Scherer 2017]   SCHERER, Mattias: *Performance and Scalability of Blockchain Networks and Smart Contracts*, UmeÃě University, Department of Computing Science, Diplomarbeit, 2017. – 46 S

[Selimi u. a. 2017]   SELIMI, M. ; CERDA-ALABERN, L. ; SANCHEZ-ARTIGAS, M. ; FREITAG, F. ; VEIGA, L.: Practical Service Placement Approach for Microservices Architecture. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID),* May 2017, S. 401–410

[Selimi u. a. 2018a]   SELIMI, Mennan ; CERDÀ-ALABERN, Llorenç ; FREITAG, Felix ; VEIGA, Luís ; SATHIASEELAN, Arjuna ; CROWCROFT, Jon: A Lightweight Service Placement Approach for Community Network Micro-Clouds. In: *Journal of Grid Computing* (2018), Feb. – URL https://doi.org/10.1007/s10723-018-9437-3. – ISSN 1572-9184

[Selimi u. a. 2018b]   SELIMI, Mennan ; KABBINALE, Aniruddh R. ; ALI, Anwaar ; NAVARRO, Leandro ; SATHIASEELAN, Arjuna: Towards Blockchain-enabled Wireless Mesh Networks. In: *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems.* New York, NY, USA : ACM, 2018 (CryBlock'18), S. 13–18. – URL http://doi.acm.org/10.1145/3211933.3211936. – ISBN 978-1-4503-5838-5

[Sousa u. a. 2017]   SOUSA, J. ; BESSANI, A. ; VUKOLIĆ, M.: A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform. In: *ArXiv e-prints* (2017), September

[Weber u. a. 2017]   WEBER, Ingo ; GRAMOLI, Vincent ; PONOMAREV, Alex ; STAPLES, Mark ; HOLZ, Ralph ; TRAN, An B. ; RIMBA, Paul: On availability for blockchain-based systems. In: *Reliable Distributed Systems (SRDS), 2017 IEEE 36th Symposium on* IEEE (Veranst.), 2017, S. 64–73

[Wood 2014]   WOOD, Gavin: Ethereum: A secure decentralised generalised transaction ledger. In: *Ethereum Project Yellow Paper* 151 (2014), S. 1–32