Leveraging Public-Private Blockchain Interoperability for Closed Consortium Interfacing

Bishakh Chandra Ghosh, Tanay Bhartia, Sourav Kanti Addya, and Sandip Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, India
Email: ghoshbishakh@gmail.com, tanaybhartia@gmail.com, souravkaddya@cse.iitkgp.ac.in, sandipc@cse.iitkgp.ac.in

IEEE INFOCOM 2021 - IEEE Conference on Computer Communications | 978-1-6654-0325-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOM42981.2021.9488683

***Abstract*—With the increasing adoption of private blockchain platforms, consortia operating in various sectors such as trade, finance, logistics, etc., are becoming common. Despite having the benefits of a completely decentralized architecture which supports transparency and distributed control, existing private blockchains limit the data, assets, and processes within its closed boundary, which restricts secure and verifiable service provisioning to the end-consumers. Thus, platforms such as e-commerce with multiple sellers or cloud federation with a collection of cloud service providers cannot be decentralized with the existing blockchain platforms. This paper proposes a decentralized gateway architecture interfacing private blockchain with end-users by leveraging the unique combination of public and private blockchain platforms through interoperation. Through the use case of decentralized cloud federations, we have demonstrated the viability of the solution. Our testbed implementation with Ethereum and Hyperledger Fabric, with three service providers, shows that such consortium can operate within an acceptable response latency while scaling up to 64 parallel requests per second for cloud infrastructure provisioning. Further analysis over the Mininet emulation platform indicates that the platform can scale well with minimal impact over the latency as the number of participating service providers increases.**

***Index Terms*—Blockchain; Interoperability; Multifaceted networks; Open interfacing**

I. Introduction

Business-to-Business (B2B) and Business-to-Consumer (B2C) online marketplaces have gained much attention nowadays within various sectors, including e-commerce, ride-hailing, cloud service provisioning (e.g., cloud federations), supply-chain management, etc.. However, there has been a continuing debate about the market-monopoly and unfairness they created in the digital economy [1], [2]. Such platforms typically work as the central agent or broker to interconnect various businesses and consumers. In such a firm-controlled marketplace, supporting trustworthiness and unbiased business transactions is always a concern. Blockchain is a natural extension to help trustworthy and bias-free business by allowing the stakeholders to interact over a decentralized marketplace. Therefore, various recent works advocate for developing blockchain-based electronic marketplaces [3]–[8]. However, there is a fundamental limitation of the current blockchain technologies to support this, as discussed next.

An electronic marketplace is typically a multifaceted network with one or more closed business networks collaborating through B2B transactions and, finally, an open consumer network having B2C operations [9]. For example, in a typical supply chain, manufacturers, wholesalers, and retailers form different closed business networks, and finally, the end-customers create an open consumer network. Another example is cloud federation platforms like OnApp [10], where small cloud service providers (CSPs) construct a closed consortium to provide cloud resources to customers. Depending on customer requests, the transactions flow from the open consumer network to various closed business networks, and the service is finally delivered back to the open consumer network.

Emerging blockchain networks such as *IBM Food Trust* [11], *TradeLens* [12], *Marcopolo* [13], etc., use private (permissioned) distributed ledger-based systems like *Hyperledger Fabric* [14] and *Corda* [15] to form closed consortia of businesses. However, a key limitation of the existing private blockchain platforms is the restriction of their applicability within only closed consortia where data and assets are not required to be communicated outside the network boundary. Thus, *Fabric*, *Corda*, or other existing private blockchains do not support any interface or protocols for interacting with the open network outside, which is crucial for building consortia of service providers acting together to deliver services to the consumer network.

However, there are challenges in designing such interfacing. **First**, the businesses, as well as the consumers, can exhibit byzantine behavior in the absence of a firm-controlled marketplace. Therefore they can collude to deceive and take control over the consortium decisions. **Second**, the consumers' service requests need to be agreed upon by the businesses within the closed consortium along with their ordering, before they can be processed. Otherwise, any malicious business can take priority over a profitable service request, thus affecting the fairness of the system. Although private blockchain can ensure transaction execution order within the closed network, they do not support transactions from outside the closed network pertaining to Sybil attacks from the open network participants [16]. **Third**, the service responses from the closed consortium also need to be transferred back to the consumer who requested the service. Such information must be verifiable by the consumers against the valid consensus at the business network. Further, the privacy of the information must be ensured.

Thus, towards developing a decentralized collaborative architecture for service providing consortia, we introduce ***CollabFed***, which addresses the above challenges by building a

978-1-6654-0325-2/21/$31.00 ©2021 IEEE

novel decentralized interface between the private blockchain networks and the open network of consumers. *CollabFed* ensures multi-party consensus validation and considers threats such as Sybil attacks and byzantine behaviors of the participants. The decentralized interface is engineered through a unique combination of the public blockchain and private blockchain networks by enabling interoperability between them to support trusted and secure data transfer in both the directions, that is (a) from the consumers to the businesses and (b) from the businesses to the consumers (**Contribution-1**). Our *Consensus on Consensus* mechanism handles the transfer of data from the open network into the private blockchain in a secured and verifiable manner (**Contribution-2**). We employ a novel mechanism based on *collective signing* (CoSi) technology [17] to generate verifiable results from the consortium, which is accessed securely by the consumers (**Contribution-3**). Moreover, *CollabFed* facilitates the collaboration among the participating businesses and enables fair scheduling of requests through a distributed consensus. Performance in terms of latency is of utmost importance here, so we analyze the effect of *order-execute* and *execute-order* transaction execution workflows on the performance of request scheduling.

Considering a use case of a decentralized brokerless cloud federation, we have done a proof-of-concept (PoC) implementation of *CollabFed* using *Ethereum* as the public blockchain platform and *Hyperledger Fabric*, and *Burrow* as the two different candidates for the private blockchain platform, and tested it with three emulated CSPs (**Contribution-4**). The experiments prove the viability of *CollabFed* as a platform for service provisioning consortia, which supports interaction between a private blockchain network and the end-consumers. Evaluation of the performance shows acceptable overhead on the federation, and a Mininet-based emulation with 32 CSPs also validates its scalability over a large geo-distributed setup.

II. Related Work

One of the most compelling use cases of blockchain technology is in industries and enterprise environments where multiple authoritative domains such as companies, organizations, and governments form a consortium without any central trusted mediator's involvement. Research on enabling such applications have been carried out in sectors like energy trading [18], supply chain [19], cloud [20]–[22], and many more [23]. However, almost all existing solutions consider a closed consortium of organizations that do not require communication with the outside. Some blockchain-driven systems which enable businesses to interact with consumers have been proposed, such as BlockV [5], a ride-sharing application ensuring fairness, and ArtChain [6] - a blockchain-based art marketplace. Similarly, Savi *et al.* introduced a public blockchain-based cloud brokerage platform [24] using Ethereum for sharing spare fog resources. Although connecting businesses and consumers, these platforms are based on the public blockchain only, and thus are not suitable for enterprise use cases that involve sensitive data exchange between the consortium members. Moreover, public blockchains are not ideal for complex business logic-based smart contracts since they have to be replicated and executed over the entire network, thus hampering performance.

Using private blockchain for such use cases will require some mode of interoperability with the public blockchain. Several prior works focus on cross-chain communication [25] for different applications such as cross-network asset exchange or asset transfer. Most of them such as, Tesseract [26], Herlihy [27], Xclaim [28], AMHL [29], focus on public-public blockchain interoperability for exchange of cryptocurrency, asset transfer, and payment channel networks. On the other hand, Omniledger's Atomix [30], Chainspace [31], Fabric Channels [32] enable interoperability and transactions between different shards of the same blockchain platform. Abebe *et al.* [33] proposed a protocol for trusted and verifiable data transfer across private blockchain networks using endorsement collection. Cash *et al.* [34] proposed a two-tier public-private blockchain architecture for secure data sharing. However, none of these existing works address the interoperability and data transfer between private and public blockchain platforms.

To the best of our knowledge, *CollabFed* is the first attempt to address the issue of communication of consumer requests, and processed responses between a private blockchain-based consortium and the open network through public-private blockchain interoperability.

III. System Model and Design Challenges

We consider the interconnecting network between the consumers and the closed consortium to be partially synchronous where there is an upper bound Δ on the time of message delivery [35], [36]. If a message is not received within the time-bound Δ, then it is considered as a message fault. The intuition is that in a realistic communication, the messages must have arbitrary but bounded delay. This results in challenges such as unordered message delivery and message drops. Additionally, we consider different types of attacks that might affect the above operations, as follows.

A. Threat Model

A decentralized consortium is prone to the following types of attacks, which we take care of in the design of *CollabFed*.

Byzantine participants: We consider that at most $\frac{1}{3}$ of the participants, both for businesses and consumers, may exhibit byzantine behavior [37]–[40]. A consumer can try to deceive the consortium by sending different requests to different businesses, while the businesses can collude themselves to alter the decision protocols' results to take control of the consortium.

Sybil attacks: BFT consensus protocols assume that each participant has only one distinct identity [36], [38], [39]. If somehow one participant can generate multiple identities, then using such redundancy, it can launch a "Sybil Attack" [16]. The consumers thus can launch a Sybil attack to the closed consortia by using multiple identities.

Impersonation attacks: As a decentralized architecture, the consortium does not have a single spokesperson responsible for communicating with the open network consumers. Exploiting this, a malicious business from the closed consortium might try to deceive a consumer by posing as the consortium's spokesperson and providing false information.

Leakage of sensitive information: The business and the consumers communicate over an open, unsecured channel through message passing. Therefore, sensitive information like credentials, contact information, etc., might get leaked.

B. Design Philosophy and Challenges

CollabFed's primary objective is to develop a mechanism through which any closed consortium designed using a private blockchain platform can interface with open consumer networks. Considering the threat model as discussed above and the possibility of unordered message delivery along with message drops, in *CollabFed*, the following two guarantees need to be ensured at the consortium interface.

Definition 1. Consortium Interface Safety - The interface should ensure that all the correct consortium members agree on the same set of incoming consumer requests in same order.

Definition 2. Consortium Interface Liveness - The interface must ensure that all the correct consumer requests are eventually be processed and committed by the closed consortium.

Thus, a mechanism is needed such that the interface meets the safety and liveness guarantees, and the consortium members are in a consensus on each request. To achieve consensus over the ordering of consumer requests from the open network, we propose to use public blockchain platforms [41]–[43] for interfacing the closed consortium to the consumers of the open network. The consensus algorithms over a public blockchain setting are designed to be resistant to Sybil attacks. Therefore, using a public blockchain platform, the consumers' requests from an open network can be ordered. However, merely clubbing together any public and private blockchain is not enough to enable the targeted consortium interface; there are open challenges that need to be solved.

(i) **Passing consensus of one network to another:** The public and the private blockchain networks run their own consensus protocols independently. The interface should pass the consensus information from one network to another by ensuring (i) security, and (ii) accountability. The interface should guarantee that the consensus information of one network is verifiable at the other network.

(ii) **Transferring sensitive information from the closed network to the consumers:** Once a consumer request is scheduled and processed by the closed consortia, the associated service information such as access credentials, invoice, shipping information, etc., need to be passed to only the targeted consumer who has requested for the service. Therefore, merely putting the information to the public blockchain will not help, as anyone will access it. Protocols need to be designed to share such sensitive information with the targeted consumer only.

(iii) **Verifiability of the consortium decision:** The consortium's decision of scheduling, service provisioning, etc. comes through a consensus over the private network. However, once this information is forwarded to the public network, the consumers should be able to verify such decisions to avoid any byzantine behavior from the colluded consortium members.

IV. Decentralized Consortium Interface

The functionality of *CollabFed Consortium Interface* is broadly two-fold: (a) transferring consumer requests from the open network to the closed consortium members (Fig 1), (b) transferring consortium responses to the open network consumers in a secure and verifiable way (Fig 2). The *Consortium Interface Safety* is achieved using two rounds of consensus over the consumer requests – (1) *regular consensus (mining) of the public blockchain*, and (2) A *Consensus on Consensus* mechanism. The details follow.

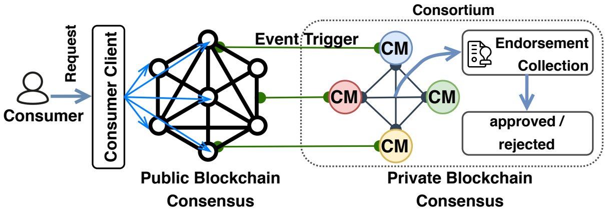

Fig. 1: Transferring Consumer Requests from Public Blockchain to the Consortium Members (CMs)

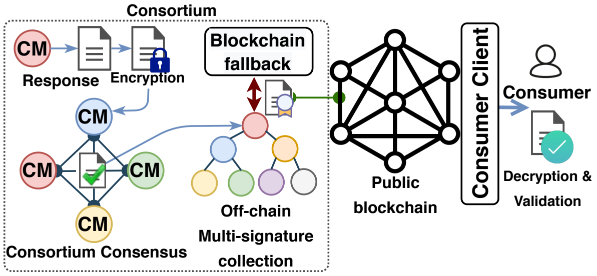

Fig. 2: Secure and Verifiable Data Transfer from CMs to Consumers

A. Regular Consensus (Mining) over Public Blockchain

Before scheduling and processing any consumer request, the consortium members must reach a consensus on the same. Moreover, there needs to be a consensus on the order in which the requests are to be considered to ensure *Consortium Interface Safety*. This ensures that a malicious member of the consortium cannot collude the network by triggering the scheduling of an invalid consumer request or take priority on a specific consumer request. *CollabFed* uses public blockchain in conjunction with the private consortium to support this. However, for supporting interoperability between the two networks, the consensus has to be propagated between them.

To interact with the consortium, consumers send their requests through the public blockchain. These requests are formed as transactions to a smart contract - *User Request Contract*, deployed in the public blockchain. Just like a web interface of a central firm-controlled platform, this smart contract acts as the communicating point for the consumers to reach the consortium, albeit in a decentralized way. The "consumer request" transactions are then committed to a block in the ledger through the public blockchain platform's mining/consensus process. For example, *Ethereum* uses a modification of the most popular consensus protocol: "proof of work" (PoW) [44], while there are many alternate consensus protocols, such as Proof of Stake [43], Bitcoin-NG [45], Byzcoin [46], Algorand [42] etc. These consensus protocols have different safety and liveliness assumptions of their own; however, their common objective is to reach consensus on a block of transactions. Moreover, since these are permissionless blockchain protocols, they are designed to resist Sybil attacks.

Once a block is mined and committed in the public blockchain, this ensures that there is a consensus on the particular block and their order in which they are committed, since each block is linked to the previous one through its cryptographic hash. Moreover, the set of transactions in each block also has a fixed packing order for the smart contracts' deterministic serial execution. Despite these properties, public blockchain consensus itself is not enough to satisfy *Consortium Interface Safety*, and consortium members cannot simply pick user requests from the public blockchain and start processing them. The reasons are as follows. (1) Due to the partially synchronous network, some consortium members might not get the mined block in time and thus cannot participate in its scheduling. (2) Malicious consortium members may introduce and schedule invalid consumer requests that are not mined at all. (3) Public blockchain consensus protocol like PoW, often goes through temporary forks [47], resulting in conflicting consumer requests or conflicting ordering in different members. Thus, *CollabFed* has to carry out a second round of consensus, which we call *Consensus on Consensus*.

B. Consensus on Consensus

In [25], the authors have shown an interesting result that states that cross-chain communication is impossible without a trusted third party. To circumvent this impossibility result, *CollabFed* uses a novel idea where the private consortium members also participate in the public blockchain to represent themselves as their own trusted agent. Whenever a new block is committed in the public blockchain, the trusted agents corresponding to the private consortium members get an event-trigger, which in turn invokes a *Propagation Contract* in the private blockchain network. Before invoking the *Propagation Contract*, the transactions of the public blockchain can be verified individually by the consortium members by existing methods such as *Simplified Payment Verification (SPV)* as used in standard public blockchain like Bitcoin [44].

The task of the *Propagation Contract* is to collect **verification endorsements** from consortium members for each consumer request. The *verification endorsements* are the digitally signed certificates from the consortium members, indicating that the corresponding members agree on the processing of a consumer request committed over the public blockchain. As per the standard BFT protocols [38], [46], a consumer request can be committed for scheduling in the private consortium if the majority ($\frac{2}{3}$rd) of the consortium members endorse the request transaction. The endorsement protocol used in the *Propagation Contract* is shown in Fig. 3. The details follow.

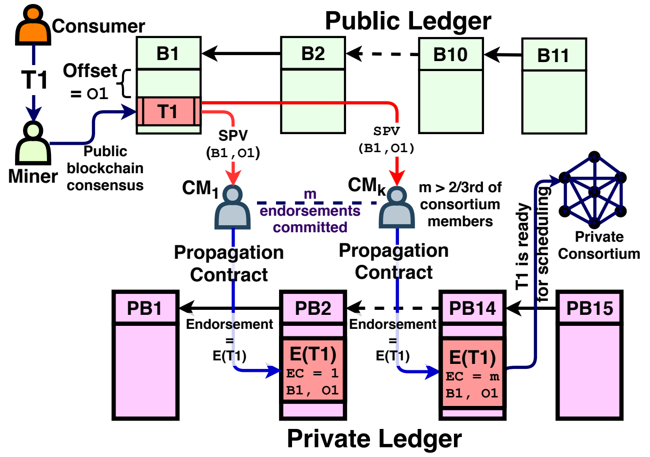

Fig. 3: Propagation Contract: *Consensus on Consensus*

Endorsement Initialization: Whenever a consortium member receives a "consumer request" transaction through the event listener of the public blockchain, it checks whether there is already an endorsement available in the private ledger corresponds to that transaction. If no endorsement is available, it initiates the endorsement collection process for that particular request by initiating the `endorsement-count (EC)` variable set to 1, and committing the signed endorsement in the private ledger. The request is also accompanied by a sequence number for representing its order. This sequence number is formed as {`blocknumber, offset`}, indicating the block in which the request transaction is committed in the public blockchain, and its packing order inside the block.

Endorsement Propagation: As other consortium members also get the same consumer request and with the same {`blocknumber, offset`} through the event listener of the public blockchain, they also execute the *Propagation Contract* for it, which adds their signed endorsements while incrementing the `EC`. Each execution of the *Propagation Contract* is also a transaction. Therefore, each endorsement also goes through the consensus process of the private blockchain.

Commitment: Thus, the number of endorsements for a request goes up until it reaches greater than two-third of the number of consortium members ($\texttt{EC} > \frac{2}{3}|consortium|$). At this point, the majority of the consortium participants have consensus on the request through endorsements, and each such endorsement has a consensus of the network. Thus, the consumer request is marked as approved and ready to be scheduled.

Theorem 1. The *Consensus on Consensus* mechanism ensures consortium interface safety and consortium interface liveness.

Proof: Whenever a transaction is committed in a block in the public blockchain, it implies all its correct participants including consortium members agree on it, along with the (`blocknumber`, `offset`). The *Consensus on Consensus* mechanism endorses the transactions from the public blockchain and then commits the endorsements in the private blockchain. A transaction is scheduled only when more than $\frac{2}{3}$ of the consortium members endorse the transaction. Given that each endorsement transaction also undergoes consensus in the private blockchain, and the given verifiability property of the private ledger, a transaction from the public blockchain is executed only when the majority of the consortium members endorse it. Further, the transactions are executed in the order of (`blocknumber`, `offset`) parameters of the public blockchain ensuring agreement on the order. Thus the *Consensus on Consensus* mechanism ensures interface safety.

Consortium interface liveness depends on the liveness of the public blockchain. The event-listeners for correct consortium members eventually trigger the propagation contract when a transaction is committed in the public ledger. Even if there is a temporary fork, the propagation contract is executed when the transaction is finally committed in the public ledger. ■

The *Propagation Contract* triggers *Scheduling Contract* that schedules the requests based on a predefined business logic. After a request is scheduled and processed over the closed consortium, the service results have to be transferred back to the consumers. The details follow.

C. Secure and Verifiable Response Transfer

A consortium is operated collectively by its participant businesses. Hence, any data/information provided by it has to be the result of the collective consensus process. Thus, in the absence of a central coordinating platform, this consensus has to be collected and verified by the consumers, without depending on any trusted agent. There can be two variations of information originating from the consortium. (1) ***Consortium information*** such as information about the participating businesses, service catalogs, etc., and (2) ***Request responses*** that are the results of scheduling and processing consumer requests such as a digital document.

Both of these kinds of data are generated collectively by the consortium members through the private blockchain's consensus process. However, this consensus information has no manifestation outside this closed network. Thus, consumers being outside the consortium and not participating in the consensus protocol cannot verify the correctness of the data that is committed through transactions in the private blockchain. A separate protocol has to be designed through which information transfer from the consortium to the consumers can be validated outside the private network concerning the consensus of the participating businesses. Moreover, although *consortium information* can be considered publicly available, the *Request responses* to the consumers may contain sensitive information that should remain confidential while being transferred across the open network of the public blockchain.

In *CollabFed*, we use the concept of *Collective Signing* (CoSi) [17] where a set of consortium members collectively sign a valid information to make it verifiable. We utilize *Boneh-Lynn-Shacham* (BLS) cryptosystem [48] for collecting and aggregating signatures from the individual participating businesses. Similar to Byzcoin [46], which uses CoSi to reach to a BFT consensus, a piece of information posted by the consortium through the public blockchain is considered to be valid, if and only if it has been signed by at least $\frac{2}{3}$rd of the consortium members. The details follow.

1) BLS Signatures: A BLS signature is computed as $\mathbb{S}_i(\mathcal{M}) = \mathcal{H}(\mathcal{M})^{\mathcal{S}_{\mathcal{C}_i}}$, where $\mathcal{M}$ is the message that is to be signed, $\mathcal{H}(.)$ is a cryptographic hash function, and $\mathcal{S}_{\mathcal{C}_i}$ is the secret key of the consortium member $\mathcal{C}_i$. The property that makes BLS signatures special is that they can readily be extended to multi-signatures. Therefore, for n members participating in the consortium, $\mathcal{C}_1, \mathcal{C}_2, \cdot, \mathcal{C}_n$, the aggregated multi-signature can be calculated as follows.

$$\begin{aligned} \mathbb{S}_{1..n}(\mathcal{M}) = \mathcal{H}(\mathcal{M})^{\mathcal{S}_{\mathcal{C}_1}+\mathcal{S}_{\mathcal{C}_2}+..+\mathcal{S}_{\mathcal{C}_n}} &= \prod_{i=1}^{n} \mathcal{H}(\mathcal{M})^{\mathcal{S}_{\mathcal{C}_i}} \\ = \mathbb{S}_1(\mathcal{M}) \times \mathbb{S}_2(\mathcal{M}) \times .. \times \mathbb{S}_n(\mathcal{M}) &= \prod_{i=1}^{n} \mathbb{S}_i(\mathcal{M}) \end{aligned} \tag{1}$$

This aggregated multi-signature $\mathbb{S}_{1..n}(\mathcal{M})$ can be verified with the help of the public keys of the individual consortium members. This verification is done by comparing the pairing operation between the aggregated signatures and the aggregated public keys. The aggregated public key for n members is calculated as $\prod_{i=1}^{n} \mathcal{P}_{\mathcal{C}_i}$, where $\mathcal{P}_{\mathcal{C}_i}$ is the public key of $\mathcal{C}_i$.

2) Posting information using BLS: Any information about the consortium is communicated to the consumers by posting the same in the public blockchain. Such information originates from the result of the *Collaboration Contract* in the private blockchain, which is responsible for reaching consensus on them. This resultant data like updated information or updated catalog, etc. must be collectively signed by at least $\frac{2}{3}$rd of the participating consortium members. This again has two different levels of security requirements for *Consortium information* and *Request responses*.

Posting Consortium Information to the Public Blockchain: Let $\mathcal{I}$ be a piece of public consortium information that is meant to be seen by all consumers. $\mathcal{I}$ is proposed by a consortium member in the private blockchain where consensus is reached over it. To post this information over the public blockchain, the consortium members over the closed network construct a *Signing-Request message* as `sign`$\{\mathcal{H}(\mathcal{I}), \mathbb{B}, [\mathcal{H}(\mathcal{I})]_{\mathbb{S}_{\mathbb{B}}}\}$ and forward it to all other consortium members. Here $\mathbb{B}$ is a bitmap indicating which members have signed the message and $[\mathcal{H}(\mathcal{I})]_{\mathbb{S}_{\mathbb{B}}}$ is the aggregated collective signature on the hash of the message $\mathcal{I}$. Every consortium member, upon receiving this message, adds its own signature through multiplication, as shown in Eq. (1), updates $\mathbb{B}$ and sends back the response. Once signatures from majority of the members have been aggregated, the final response message $\{\mathcal{I}, \mathcal{H}(\mathcal{I}), \mathbb{B}, [\mathcal{H}(\mathcal{I})]_{\mathbb{S}_{\mathbb{B}}}\}$ is posted in the public blockchain. The authenticity of this message can be easily verified using the public keys of the members who have signed the message, and the integrity can be checked by computing and comparing the hash of $\mathcal{I}$. This verification process is carried out by the *Consumer Client* and is transparent to all the consumers. The *Consumer Client* only accepts those messages which have the required number of signatures ($> \frac{2}{3}|consortium|$) along with the proper hash.

Posting Private Information for a Consumer: Posting private information to a consumer through the public blockchain requires some mechanism to preserve confidentiality. This is done by encrypting the message using the public key $\mathcal{P}_{\mathcal{U}}$ of the consumer $\mathcal{U}$. The message is also similarly authenticated using the aggregated multi-signature of the consortium members. Thus the final message which is posted in the public blockchain is $\{< \mathcal{M} >_{\mathcal{P}_{\mathcal{U}}}, \mathcal{H}(< \mathcal{M} >_{\mathcal{P}_{\mathcal{U}}}), \mathbb{B}, [\mathcal{H}(< \mathcal{M} >_{\mathcal{P}_{\mathcal{U}}})]_{\mathbb{S}_{\mathbb{B}}}\}_{\mathcal{P}_{\mathcal{U}}}$, where $< \mathcal{M} >_{\mathcal{P}_{\mathcal{U}}}$ denotes a message $\mathcal{M}$ encrypted using the key $\mathcal{P}_{\mathcal{U}}$. Thus, only the consumer $\mathcal{U}$ can decrypt the message using its secret key $\mathcal{S}_{\mathcal{U}}$. The *Consumer Client* handles the decryption and verification of authenticity.

D. *Optimizing the Latency for Signature Collection*

Since the messages to be transferred from the consortium to the consumers already have to be committed in the private blockchain, the multi-signature collection process is decoupled and carried out off-chain to improve the latency. Thus the consortium members communicate through peer-to-peer messages to form the verifiable signed message. This multi-signature mechanism's latency depends on the way the members forward the messages and collect back the signatures to generate the final payload by aggregating them. Thus a communication tree is formed along which the singing request and the signatures are exchanged. One extreme case of this is when one of the members acts as the leader, and the other members sign their messages and forward them back to it. The leader constructs the collective signature by including its own signature and validates other members' signatures against their public keys.

This strategy is likely to have low latency because of its star topology with a path length of at most one but will have high signature combination computation overhead for the leader. Another extreme is to consider a linear chain of consortium members through which the above round of messages propagate; this will have less computation overhead for each member but will have high network latency. *CollabFed* uses a M-ary tree structure to propagate multi-signature collection messages through which individual signatures are collected, and the multi-signature is constructed following Eq. (1). Interestingly, the latency for multi-signature generation changes with the value of M, which we analyze in Section VI.

Handling denial of service: Off-chain multi-signature collection improves the latency of the process. However, it introduces the risk of denial of service. Although the message to be signed is first committed in the private blockchain through the consensus process, some malicious consortium participants may try to halt the consortium through denial of service attack by not responding to signature collection requests. As a result, to prevent that and detect the faulty members to hold them responsible, *CollabFed* resorts to a blockchain contract-based signature collection after the off-chain protocol fails (possibly with a timeout). For a message, the *Signature Collection* contract is initialized in a similar way as *Propagation Contract*, and gathers BLS signatures of the members. Thus any non-cooperating member is detected through this transparent process, who can be held responsible.

Algorithm 1: Fair Request Scheduling Contract

```
Input: $\mathcal{R}_i$, $\mathbb{K}$, $\mathbb{W}$
Result: Scheduled CSP: $\mathcal{C}_s$
for $\mathcal{C}_j \in \mathcal{F}$ do
    \* Initialize current proportion of scheduled requests of $\mathcal{C}_j$ to 0 *\
    $\mathcal{G}_{\mathcal{C}_j} \leftarrow 0$
    for $l \leftarrow 1 \ to \ |\mathbb{W}|$ do
        if $\mathbb{W}[l] = \mathcal{C}_j$ then
            $\mathcal{G}_{\mathcal{C}_j} \leftarrow \mathcal{G}_{\mathcal{C}_j} + 1$
        end
    end
    $\mathcal{G}_{\mathcal{C}_j} \leftarrow \frac{\mathcal{G}_{\mathcal{C}_j}}{|\mathbb{W}|}$
    $\mathcal{D}_{\mathcal{C}_j} \leftarrow \mathcal{G}_{\mathcal{C}_j} - \hat{\mathcal{K}_{\mathcal{C}_i}}$
end
$\mathcal{C}_s \leftarrow \mathrm{argmax}_{\mathcal{C}_i \in \mathcal{F}}(\mathcal{D}_{\mathcal{C}_j})$
enqueue($\mathbb{W}$, $\mathcal{C}_s$)
if $|\mathbb{W}| > \mathfrak{K}$ then
    dequeue($\mathbb{W}$)
end
return $\mathcal{C}_s$
```

V. Use Case Implementation: Cloud Federation

To evaluate the potential of *CollabFed*, we have implemented a use-case of cloud federations like *OnApp*, where a group of CSPs participate in a single marketplace to offer cloud infrastructure such as virtual machines (VMs) as a service (IaaS) to the consumers. Traditionally cloud brokers [49] or centralized marketplaces like *OnApp* coordinate all interactions between the CSPs and the consumers. To design a fully trustless decentralized architecture for cloud federations, we use *CollabFed* to implement a private network of CSPs and a public network of consumers, called *CollabCloud*. Apart from the basic functionalities of *CollabFed*, *CollabCloud* implements a Fair Scheduling Contract within the CSP consortium to schedule the VM requests among the participating CSPs while ensuring fairness in terms of profitability of the CSPs and quality of service (QoS) for the consumers.

The *Fair Request Scheduling Contract* takes into account the contribution of the individual CSPs in the federation and schedules consumer requests in proportion to it. We define the federation $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$ as a collection of CSPs $\mathcal{C}_i$. A CSP $\mathcal{C}_i$ can support certain VM configurations which are represented by $\mathbb{V}^{\mathcal{C}_i} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m\}$. Thus the catalog of the federation is the union of all such VM configurations being offered by the individual CSPs, represented as $\mathbb{C} = \bigcup_{\mathcal{C}_i \in \mathcal{F}} \mathbb{V}^{\mathcal{C}_i}$. Similar to the catalog, the contribution of each CSP $\mathcal{C}_i$ is a set of *VM offerings*, denoted by $\mathbb{O}^{\mathcal{C}_i} = \{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_m\}$. A *VM offering* is defined as a three-tuple: $\mathcal{O} = \{\mathcal{V}, k, c\}$, where $\mathcal{V}$ denotes a VM configuration, k denotes the quantity of the VMs of the particular configuration the CSP can offer, and c denotes the expected pricing of that VM type. A consumer request for a VM is defined as a four-tuple: $\mathcal{R} = \{\mathcal{R}_{id}, \mathcal{P}_{\mathcal{U}}, \mathcal{V}_j, \mathcal{D}\}$, where $\mathcal{R}_{id}$ is the unique identifier of the consumer request, $\mathcal{P}_{\mathcal{U}}$ is the public key of the consumer making the request, $\mathcal{V}_j \in \mathbb{C}$ is the

VM configuration selected from the catalog $\mathbb{C}$, and $\mathcal{D}$ is the duration for which the VM is requested.

The fair scheduling smart contract is shown in Algorithm 1. The input to the algorithm is a consumer request $\mathcal{R}_i$, the proportions of contribution of all CSPs in the federation $\mathbb{K} = \{\hat{\mathcal{K}_{\mathcal{C}_i}} |\ \mathcal{C}_i \in \mathcal{F}\}$, and an array $\mathbb{W}$ consisting of the results of this algorithm for last $|\mathbb{W}|$ scheduled requests. We define infrastructure contribution $\mathcal{K}_{\mathcal{C}_i}$ of each CSP $\mathcal{C}_i$ as $\mathcal{K}_{\mathcal{C}_i} = \sum_{\mathcal{O}\in\mathbb{C}_{\mathcal{C}_i}} \mathcal{O}.\mathcal{V}.\mathcal{CPU} \times \mathcal{O}.k$, that is the weighted sum of the quantities of its *VM offerings* indicating the amount of IaaS capacity (hardware resources) contributed. Thus, each time the catalog is updated, the proportion of contributions are also changed. The contribution proportion is thus $\hat{\mathcal{K}_{\mathcal{C}_i}} = \frac{\mathcal{K}_{\mathcal{C}_i}}{\sum_{\mathcal{C}_j\in\mathcal{F}} \mathcal{K}_{\mathcal{C}_j}}$, for each CSP $\mathcal{C}_i$.

In essence, the scheduler works similarly to a *weighted fair queue* [50] which ensures that the rate of consumer requests received by each CSP $\mathcal{C}_i$ is proportional to $\hat{\mathcal{K}_{\mathcal{C}_i}}$. For this purpose, the scheduling contract keeps track of a window ($\mathbb{W}$) of the past scheduled results. We implement $\mathbb{W}$ as a queue containing results for past requests that is $\mathcal{R}_{i-1}, \mathcal{R}_{i-2}, \ldots, \mathcal{R}_{i-|\mathbb{W}|}$. Here each result corresponds to some CSP to which the past request was scheduled. The algorithm first computes the proportion of requests scheduled to a particular CSP as $\mathcal{G}_{\mathcal{C}_j}$ and then computes the proportion deficit as $\mathcal{D}_{\mathcal{C}_j}$. The request $\mathcal{R}_i$ is then scheduled to the CSP, having the maximum deficit in its share of past scheduled requests. Then the window $\mathbb{W}$ is updated by inserting the new result, and also removing the oldest result if $|\mathbb{W}| > \textit{some threshold } \mathfrak{K}$.

Verifiability of the Scheduling Algorithm: $\mathcal{R}_i$ is obtained from the public blockchain, and $\mathbb{K}$ is available in the private ledger. Finally, the past scheduled requests are obtained from the previous results of the *Fair Resource Scheduling* contract in the private blockchain. Thus, each CSP has access to all the information from the two blockchains. For verifiability, it must be ensured that all the CSPs act on the same version of information. With each execution of *Fair Resource Scheduling* contract, the value of $\mathbb{W}$ is altered. Therefore, the CSPs must know which version of $\mathbb{W}$ is applicable for which transaction. This is ensured in two different ways – **order-execute** and **execute-order** based executions of the contracts [14].

Case (i): order-execute – The transactions are ordered first, and the consensus is achieved on this ordering. The transactions are then executed sequentially based on the agreed order, and $\mathbb{W}$ is updated. Thus every CSP applies the transactions in the same sequence on $\mathbb{W}$, starting from the initial version.

Case (ii): execute-order – Each transaction is first simulated on a particular version of $\mathbb{W}$, and this version number is also included in the transaction. Then the simulation result (an updated version of $\mathbb{W}$) is sent for consensus. In the case of multiple such parallel transactions acting on the same version of $\mathbb{W}$, only one transaction is agreed upon during the consensus and accepted. The rest of them are rejected.

VI. Evaluation

In order to test the feasibility and practicality of *CollabFed*, we have implemented a PoC of *CollabCloud* decentralized cloud federation. Each component of *CollabFed* along with the additional cloud federation specific functionalities are developed, and the end-to-end system is deployed in a testbed (Fig. 4). Since *CollabFed* needs one public blockchain platform for providing the *Consortium Interface*, we have chosen *Ethereum* [41]. For the private blockchain, we have tested with *Hyperledger Fabric* and *Burrow* platforms. The public blockchain smart contracts are implemented using `Solidity (v0.5.0)` (https://solidity.readthedocs.io/en/v0.5.0/) language, and they are executed on the Ethereum Virtual Machine (EVM). We have used `Truffle` (https://www.trufflesuite.com/) for the development and testing of the Ethereum contracts. Two test networks (https://docs.ethhub.io/using-ethereum/test-networks/), `Ropsten` and `Rinkeby` are used to run the consortium interface. `Ropsten` uses Proof of Work (PoW) whereas `Rinkeby` uses Proof of Authority (PoA) [51] for consensus. We evaluate *CollabFed*, as well as the cloud-federation functionalities from two different setups. First, we develop an in-house testbed with three emulated CSPs over six cloud servers (each CSP having two servers). Next, to analyze the scalability of different components of *CollabFed*, we perform an emulation-based evaluation over the Mininet virtual emulation network [52].

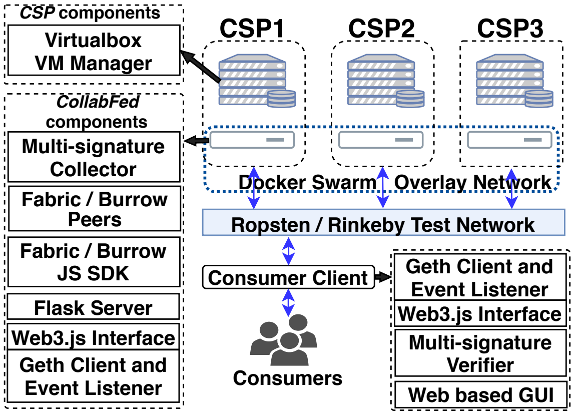

Fig. 4: *CollabCloud* modules and Testbed setup

A. Platform Setup

To test the end-to-end functionality and performance of *CollabFed* along with its various components, we set up a PoC testbed of cloud federation emulating 3 CSPs participating in the federation. Fig. 4 shows the setup where each CSP has two cloud servers – one 4-core Intel Core i5-4590@3.30GHz server with 8GB memory (Ubuntu 18.04, Linux Kernel 4.15) for running *CollabFed* services, and another 88-core Intel Xeon Gold 6152@2.10GHz server with 256GB memory (CentOS 7.7, Linux Kernel 3.10) for running the CSP's usual services including VM placement and hosting the VMs. All the services are run in `Docker` (https://www.docker.com/) containers, and the networking is established through a Docker swarm overlay network.

For implementing the CPS functionalities, we have used `VirtualBox` (https://www.virtualbox.org/) for creating

VMs, and a `Flask` (https://flask.palletsprojects.com/en/1.1.x/) server for accepting VM placement requests and interfacing with `VirtualBox`. Since each CSP has only one emulated data center, which is the host server itself, the placement algorithm does not affect our system's evaluation. However, each CSP has its own set of supported VM specifications that it offers, resulting in different catalogs. We use the *Fair Request Scheduling* contract that allocates requests based on the proportionality of the virtual CPU (vCPU) contribution in the federation by each CSP.

Apart from the testbed, to evaluate the scalability of different components of *CollabFed*, we also created a Mininet-based network topology for emulation. We created test scenarios with several *CollabCloud* CSP nodes ranging from 2 to 32 and the latency between them ranging from 50ms to 400ms to capture their performance in real-world deployments.

B. End-to-end Testbed experiments

In these experiments, we used the PoC testbed to evaluate each component's performance while doing end-to-end consumer request processing for VM provisioning. We used emulated consumers with numbers ranging from 4 to 64 and programmed them to send parallel requests at the same instance of time. We have evaluated the latency and overheads of processing these requests in each *CollabFed* module.

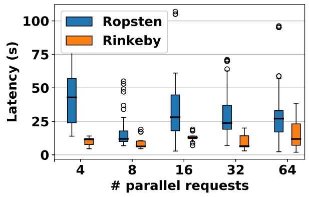

Fig. 5: Public Blockchain Latency

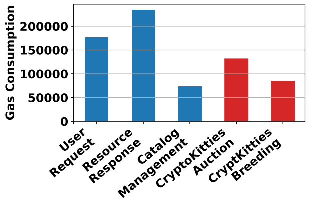

Fig. 6: Gas Consumption

Consortium Interface: Each consumer request encounters the public blockchain twice, first when it propagates from the public blockchain to the consortium, and then in the *Resource Response Contract*, when the processed result is transferred back from the private blockchain to the public one. Fig. 5 shows the distribution of latency for processing the consumer requests over the public Ethereum blockchain. The processing latency over Ethereum test networks varies widely at different times depending upon the usage by other Ethereum users across the globe. We have collected the data for two weeks at different times of the day, and the same has been plotted in Fig. 5. We observe that the PoW-based consensus process of Ropsten test network has a higher transaction processing time compared to the PoA-based Rinkeby network.

Each contract in the public blockchain requires some transaction fees proportional to its computational complexity or storage requirements. In Ethereum, this is measured as "Gas". Fig. 6 shows the gas consumption of the smart contracts of *CollabFed*, along with the cloud federation specific contracts. We observe that the *Resource Provisioning* contract is of the highest complexity since it has to store the multi-signatures for each transaction and the encrypted resource access information. To understand whether this Gas requirement is too high or too low, we benchmark these values concerning the Gas consumption by `CryptoKitties` (https://www.cryptokitties.co/) which is a common Ethereum application, and found that they are comparable.

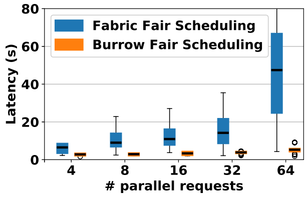

Fig. 7: *Fair Scheduling* Latency: Fabric vs Burrow

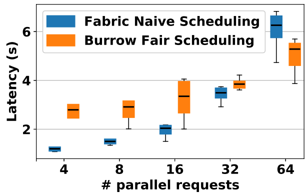

Fig. 8: Fabric Static Scheduling vs Burrow Fair Scheduling

Request Scheduling over Private Ledgers: Fig. 7 shows the time required for executing fair scheduling contracts in the private blockchain. We observe that the transaction processing time for Fabric is much higher than that of Burrow. The reason for this result is specific to the type of processing required by the *Fair Request Scheduling*. The key difference between Burrow and Fabric is the transaction execution workflow followed by them. Fabric follows *execute-order* flow, while Burrow follows *order-execute*. Executing first and then committing the results introduces a new problem for the type of contracts that read and change the system's common state, just like the *Fair Request Scheduling* uses a history of the already scheduled requests at different CSPs. The reason is as follows. While executing multiple transactions in parallel, let's assume that they get executed on the same current state $\mathcal{S}_c$, and thus the output is based on $\mathcal{S}_c$. After that, once any one of the transactions is committed, the current state is changed to $\mathcal{S}'_c$. This state change also might change the output of other transactions that would be executed after it. As a result, when the other parallelly executed transactions are processed for committing, they fail in the ordering and validation phase since their execution results do not match with the execution result on $\mathcal{S}'_c$. Fabric does not retry to execute the failed transactions by itself, so *CollabFed* over Fabric reschedules the failed transactions, thus increasing the latency.

To validate our hypothesis regarding the source of higher overhead caused by Fabric, we also tested with a naive scheduling contract that schedules the requests based on a static rule depending on its ID. This contract does not depend on the current state of the blockchain. In Figure 8, we can see that the scheduling latency of Fabric has dramatically improved. We also noticed that there are no transaction failures due to inconsistent execution results. Moreover, we saw no such latency improvements for Burrow with such a naive scheduling contract. It may be noted that for more parallel requests, Burrow still performs marginally better than Fabric. Consequently, we can conclude that the choice of private blockchain technology depends heavily on the fair scheduling contract's business logic.

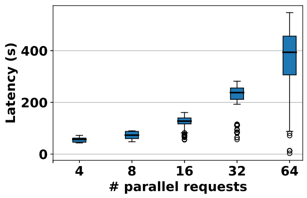

Fig. 9: VM Provisioning Latency

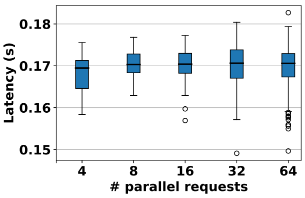

Fig. 10: Sign. Collection Latency

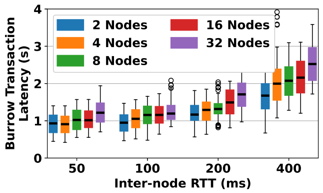

Fig. 13: Burrow scalability

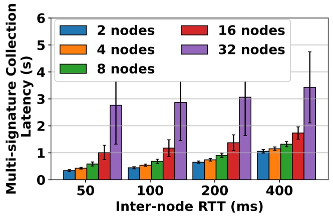

Fig. 14: BLS scalability

After a request is scheduled, a VM is provisioned accordingly, and the access information is signed through collections of BLS multi-signatures. Fig. 9 shows the distribution of the time taken for VM Provisioning. This increases with the increase in number of parallel requests, mainly due to the limited processing capability of the hardware of our setup. This latency is specific to the cloud federation application of *CollabFed*, and thus does not count towards its performance. Fig. 10 shows the distribution of latency for multi-signature collection. We see that the multi-signature collection latency remains fairly consistent.

Resource Consumption: *CollabFed* consumes CPU, memory, and network bandwidth, which are an additional overhead to normal operations of a consortium. Fig. 11 shows the box-plot distribution of CPU usage by *CollabFed* server for executing the private blockchain transactions in Fabric and Burrow, and for multi-signature collection. We observe that the CPU consumption is reasonably low, below 10% in most cases for all the services. Similarly, Fig. 12 depicts the distribution for memory requirements which stays below 200MB.

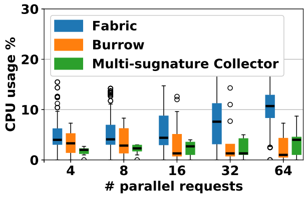

Fig. 11: CPU Usage

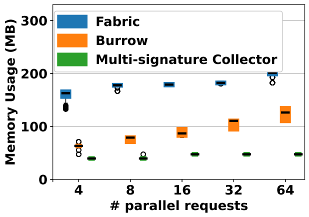

Fig. 12: Memory Usage

C. Mininet scalability experiments

The public blockchain platforms being open networks have been designed to be scalable, and extensive research has been done to study their performance [42], [46]. We focus on the scalability of the private network and the multi-signature collection. For this, we set up an experiment with 32 emulated CSPs over a Mininet [52] topology, which forms a *CollabFed* consortium. We also changed the inter-CSP network latency to emulate the CSPs' spread across different geographic regions.

Fig. 13 shows the distribution of Burrow propagation contract execution and commitment latency. The experiment has been done with inter-CSP latency varying in each case, from 50ms to 400ms. We observe that the median transaction latency lies around 2.5 seconds with 32 nodes and 400ms inter-CSP latency. Further, the increment in the transaction latency due to an increase in the number of CSPs or inter-CSP network latency is not very high, which indicates the scalability of the proposed approach.

Fig. 14 presents the mean and the standard deviation of multi-signature aggregation latency with a varying number of nodes and inter-CSP latency values. We observe that the mean latency is below 2 seconds for 16 SPs and about 3.5 seconds for 32 SPs. This also indicates the scalability of the signature aggregation scheme. However, the multi-signature collection latency can have a big impact due to the collection tree structure. To study it, we constructed a complete M-ary communication tree with 32 SPs. Table I shows the multi-signature collection latency for different values of M. The inter-CSP latency for this test is kept fixed at 400ms. We can observe a sharp improvement in the latency from linear (M=1) to binary tree (M=2) structure. The latency is more or less stable from $M = 4$. However, the multi-signature combination complexity for individual CSPs increase with the value of M. Therefore, the value of M in a real deployment can be chosen based on this trade-off.

TABLE I: Effect of communication tree on multisig collection latency

M	1	2	4	6	8	16	31
Mean Latency (s)	29.9	5.0	3.2	2.3	2.4	3.0	2.9
Standard Deviation	1.8	0.3	0.2	0.1	0.2	0.6	1.3

VII. Conclusion

Towards a fully trustless decentralized architecture for an electronic business consortium providing services to consumers, *CollabFed* introduces a public-private hybrid blockchain architecture with a unified interface between the consortia and the open network. To the best of our knowledge, this is the first attempt to fill a critical gap in the application of blockchain in the enterprise and business use cases. *CollabFed* is flexible in terms of the choice of public and private blockchain networks; however, the performance and security guarantees depend on the assumptions of those underlying blockchain technologies and consensus protocols. The PoC implementation of *CollabFed* indicates that the system is scalable and performant with Hyperledger and Ethereum – one of the most popular private and public blockchain platforms, respectively. The analysis of the impact of different blockchain protocols on the architecture is an exciting direction for our future works to develop a more robust system.

References

[1] J. Haucap and U. Heimeshoff, "Google, Facebook, Amazon, eBay: Is the Internet driving competition or market monopolization?" *International Economics and Economic Policy*, vol. 11, 2014.

[2] M. Hindman, *The Internet trap: How the digital economy builds monopolies and undermines democracy*. Princeton University Press, 2018.

[3] H. Subramanian, "Decentralized blockchain-based electronic marketplaces," *Communications of the ACM*, vol. 61, no. 1, pp. 78–84, 2017.

[4] N. Hynes, D. Dao, D. Yan, R. Cheng, and D. Song, "A demonstration of sterling: a privacy-preserving data marketplace," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 2086–2089, 2018.

[5] P. Pal and S. Ruj, "BlockV: A blockchain enabled peer-peer ride sharing service," in *IEEE Blockchain*, 2019.

[6] Z. Wang, L. Yang, Q. Wang, D. Liu, Z. Xu, and S. Liu, "ArtChain: Blockchain-enabled platform for art marketplace," in *IEEE Blockchain*, 2019.

[7] Y.-W. Chang, K.-P. Lin, and C.-Y. Shen, "Blockchain technology for e-marketplace," in *IEEE PerCom Workshops*, 2019.

[8] S. Narang, M. Byali, P. Dayama, V. Pandit, and Y. Narahari, "Design of trusted B2B market platforms using permissioned blockchains and game theory," in *IEEE ICBC*, 2019.

[9] A. Schiff, "Open and closed systems of two-sided networks," *Information Economics and Policy*, vol. 15, no. 4, pp. 425–442, 2003.

[10] "Onapp-Federation," (Last accessed: 5 Jan, 2021). [Online]. Available: https://onapp.com/onapp-federation/

[11] "IBM Food Trust," (Last accessed: 5 Jan, 2021). [Online]. Available: https://www.ibm.com/in-en/blockchain/solutions/food-trust

[12] "Tradelens," (Last accessed: 5 Jan, 2021). [Online]. Available: https://www.tradelens.com/

[13] "Marco Polo - A Trade Finance Initiative," 2020, (Last accessed: 5 Jan, 2021). [Online]. Available: https://www.marcopolo.finance/

[14] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *EuroSys*, 2018.

[15] M. Hearn, "Corda: A distributed ledger," *Corda Technical White Paper*, vol. 2016, 2016.

[16] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[17] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities "honest or bust" with decentralized witness cosigning," in *IEEE Symposium on Security and Privacy*, 2016.

[18] K. Gai, Y. Wu, L. Zhu, M. Qiu, and M. Shen, "Privacy-preserving energy trading using consortium blockchain in smart grid," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, 2019.

[19] X. Xu, F. Rahman, B. Shakya, A. Vassilev, D. Forte, and M. Tehranipoor, "Electronics supply chain integrity enabled by blockchain," *ACM Transactions on Design Automation of Electronic Systems*, vol. 24, no. 3, 2019.

[20] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *IEEE INFOCOM*, 2018, pp. 792–800.

[21] H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat, and Z. Zhao, "A blockchain based witness model for trustworthy cloud service level agreement enforcement," in *IEEE INFOCOM*, 2019, pp. 1567–1575.

[22] B. C. Ghosh, S. K. Addya, A. Satpathy, S. K. Ghosh, and S. Chakraborty, "Towards a democratic federation for infrastructure service provisioning," in *IEEE SCC*, 2019, pp. 162–166.

[23] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which, and how," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3796–3838, 2019.

[24] M. Savi, D. Santoro, K. Di Meo, D. Pizzolli, M. Pincheira, R. Giaffreda, S. Cretti, S. Kum, and D. Siracusa, "A blockchain-based brokerage platform for fog computing resource federation," in *23rd Conference on Innovation in Clouds, Internet and Networks and Workshops*, 2020.

[25] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, "Sok: Communication across distributed ledgers," Cryptology ePrint Archive, 2019, https://eprint.iacr.org/2019/1128.

[26] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *ACM CCS*, 2019.

[27] M. Herlihy, "Atomic cross-chain swaps," in *PODC*, 2018.

[28] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," in *IEEE Symposium on Security and Privacy*, 2019.

[29] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability." in *NDSS*, 2019.

[30] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *IEEE Symposium on Security and Privacy*, 2018.

[31] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," in *NDSS*. The Internet Society, 2018.

[32] E. Androulaki, C. Cachin, A. De Caro, and E. Kokoris-Kogias, "Channels: Horizontal scaling and confidentiality on permissioned blockchains," in *European Symposium on Research in Computer Security*. Springer, 2018, pp. 111–131.

[33] E. Abebe, D. Behl, C. Govindarajan, Y. Hu, D. Karunamoorthy, P. Novotny, V. Pandit, V. Ramakrishna, and C. Vecchiola, "Enabling enterprise blockchain interoperability with trusted data transfer (industry track)," in *International Middleware Conference Industrial Track*, 2019.

[34] M. Cash and M. Bassiouni, "Two-tier permission-ed and permission-less blockchain for secure data sharing," in *IEEE International Conference on Smart Cloud*. IEEE, 2018.

[35] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolić, "XFT: Practical fault tolerance beyond crashes," in *OSDI*, 2016, pp. 485–500.

[36] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[37] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: the Works of Leslie Lamport*, 2019, pp. 203–226.

[38] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[39] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "RBFT: Redundant byzantine fault tolerance," in *IEEE ICDCS*, 2013, pp. 297–306.

[40] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with bft-smart," in *IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 355–362.

[41] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[42] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *SOSP*, 2017, pp. 51–68.

[43] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[44] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[45] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *NSDI*, 2016, pp. 45–59.

[46] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *USENIX Security*, 2016, pp. 279–296.

[47] Y. Shahsavari, K. Zhang, and C. Talhi, "A theoretical model for fork analysis in the bitcoin network," in *IEEE Blockchain*. IEEE, 2019.

[48] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 514–532.

[49] J. Mei, K. Li, Z. Tong, Q. Li, and K. Li, "Profit maximization for cloud brokers in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 190–203, 2018.

[50] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Symposium Proceedings on Communications Architectures & Protocols*, ser. SIGCOMM '89. ACM, 1989.

[51] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain," in *Italian Conference on Cyber Security*, 2018.

[52] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.