Towards Scalable and Private Industrial Blockchains

Wenting Li
NEC Laboratories Europe, Germany
wenting.li@neclab.eu

Alessandro Sforzin
NEC Laboratories Europe, Germany
alessandro.sforzin@neclab.eu

Sergey Fedorov
NEC Laboratories Europe, Germany
sergey.fedorov@neclab.eu

Ghassan O. Karame
NEC Laboratories Europe, Germany
ghassan@karame.org

ABSTRACT

The blockchain emerges as an innovative tool that has the potential to positively impact the way we design a number of online applications today. In many ways, the blockchain technology is, however, still not mature enough to cater for industrial standards. Namely, existing Byzantine tolerant permission-based blockchain deployments can only scale to a limited number of nodes. These systems typically require that all transactions (and their order of execution) are publicly available to all nodes in the system, which comes at odds with common data sharing practices in the industry, and prevents a centralized regulator from overseeing the full blockchain system.

In this paper, we propose a novel blockchain architecture devised specifically to meet industrial standards. Our proposal leverages the notion of *satellite chains* that can privately run different consensus protocols in parallel—thereby considerably boosting the scalability premises of the system. Our solution also accounts for a "hands-off" *regulator* that oversees the entire network, enforces specific policies by means of smart contracts, etc. We implemented our solution and integrated it with Hyperledger Fabric v0.6 [2].

Keywords

Scalability; privacy; Private Blockchain; Hyperledger.

1. INTRODUCTION

First introduced with Bitcoin [12] in 2009, the blockchain is rapidly gaining ground as a key technology, especially in the financial and retail sectors. A number of large industrial players, such as IBM, Microsoft, Intel, and NEC are currently investing in exploiting the blockchain in order to enrich their product portfolio. Recent years witnessed the surge of a number of blockchain frameworks proposals such as Ripple [3], Ethereum [14], Corda [9], and Hyperledger [1], among others.

A number of researchers and practitioners speculate that the blockchain technology can change the way we see a number of online applications today. Although the technology is still not mature, it is expected that the blockchain will stimulate considerable changes to a large number of products, and will positively impact the digital experience of many enterprises around the globe.

However, experience with existing blockchain proposals reveals that there are still many challenges that need to be overcome prior to any large scale industrial adoption of the blockchain paradigm:

Privacy Existing blockchain deployments rely on the availability of transactions and their order of execution to all nodes in the system. Clearly, this comes at odds with current industry practices which only restrict data sharing and distribution to the intended stakeholders. While some solutions propose to selectively encrypt transactions, such approaches require delicate key management infrastructure, and still allow the remaining nodes in the system to learn about the occurrence of a particular exchange in the system.

Scalability Existing permissionless blockchains (e.g., Bitcoin) are able to scale to a considerable number of nodes at the expense of attained throughput (e.g., Bitcoin can only achieve 7 transactions per second [8]). On the other hand, permission-based blockchains can achieve relatively higher throughput, but can only scale to few hundred nodes. However, one needs to cater for both performance and scalability to meet industrial standards.

Lack of Governance One of the main attractions of the blockchain lies in its decentralized aspects. However, organizations are typically not democratic entities, and want to retain control of their systems in order to enforce specific business logics and policies.

In this paper, we address this problem, and propose a novel blockchain architecture devised specifically to meet industrial standards. Our proposal leverages the notion of *satellite chains* that form interconnected, but independent, subchains of a single blockchain system. Nodes join a given satellite chain if they want to transact with another set of particular nodes. Each satellite chain maintains its own private ledger, thus preventing any non-member node from receiving or accessing any given transaction in its ledger. Our solution supports an unbounded number of active chains at any time; different satellite chains can run different consensus protocols in parallel—thereby allowing for unprecedented levels of scalability in the system. Satellite chains can, however, transfer assets among each other at any point in time without

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

BCC'17, April 02 2017, Abu Dhabi, United Arab Emirates

© 2017 ACM. ISBN 978-1-4503-4974-1/17/04…$15.00

DOI: http://dx.doi.org/10.1145/3055518.3055531

compromising the security and soundness guarantees in the system. Finally, our solution allows a *regulator* to oversee the entire network, enforce specific policies by means of smart contracts, etc.

Our proposal can be easily integrated within existing blockchain platforms. As a proof of concept, we report on the integration of satellite chains within Hyperledger Fabric v0.6. As a by-product, we note that by effectively organizing nodes within interconnected satellite chains, our proposal emerges as the first practical solution for realizing blockchain sharding based on node relationships.

2. BACKGROUND

In what follows, we briefly overview a number of existing blockchain efforts.

Hyperledger Fabric.

Hyperledger is an open source project, managed by the Linux foundation, comprising a number of major banks and IT firms.

Hyperledger is a permission-based blockchain that requires that all prospective members register and acquire an identity (i.e., an enrollment certificate) before attempting to connect to the network and submit transactions. The fabric's *Membership Services* handle both registration process and identity management.

The system comprises clients, non-validating peers, and validating peers. Consensus agreement is performed by validating peers who validate the transactions, execute them through smart contracts (or chaincodes), and maintain the ledger. Fabric also supports "non-validating peers" which, as the name implies, assume non-consensus related tasks in the network.

Clearly, the validating peer is the core element of this architecture. However, placing key functionalities—chaincode and consensus execution—within the same blockchain node might hinder scalability. For example, validation of different transactions cannot be parallelized, because all validators must execute them sequentially by invoking chaincodes and running consensus on each of them. Hyperledger is planning the release of Fabric version 1.0 which brings major architectural changes to enhance its modularity and scalability. We discuss this extension in more details in Section 4.

Corda.

Corda is a recent distributed ledger framework proposal by R3. Corda introduces the notion of *flows* to improve transaction privacy. Flows establish point-to-point connections between nodes that wish to carry out transactions. Thus, transactions will be only visible to the contracting nodes. Corda's transactions, following Bitcoin's UTXO model [12], are linked to each other in a consumer-producer model: they take current ledger's entries as inputs to produce new ledger's entries as outputs. Nodes then verify the entire transaction graph upon the receipt of a new transaction.

The consensus protocol in Corda is run by notaries, a cluster of nodes that maintains the ledger, and ensures that no conflicting transactions is included therein. There can be multiple notaries in the network, and nodes can choose by which notary cluster they want their transactions to be finalized. During the consensus process, notaries are required to receive and verify all the transactions of their "subscribed" nodes.

However, this approach achieves privacy only for message transmission, which is orthogonal to consensus realization in the network. Namely, notaries need to see transactions to check for double spending; when traversing the transaction graph to verify a newly received transaction, nodes learn about other transactions issued by other nodes.

Blockchain Sharding.

Given the lack of scalability premises of existing blockchains, a number of recent works have proposed to shard the blockchain in order to increase the attained scalability and throughput of the system.

Elastico [11] is a permission-less blockchain that relies on network sharding. Similar to BitcoinNG [7], Elastico defines epochs within which validator nodes establish their identities in the network based on some puzzle such as Proof of Work (PoW) [5, 12]. Based on the established identities, nodes are organized into committees that process only a subset of transactions in the network. Committees then execute permission-based Byzantine consensus protocols such as PBFT [6] to confirm transactions. Finally, all blocks generated in one epoch are merged by a final committee and broadcasted into the network to update the chain.

However, such sharding protocols split the load of only transaction processing. All validator nodes still have to maintain the complete blockchain history (e.g., Bitcoin's blockchain amounts to about 120 GB at the time of writing). In fact, all nodes in the system still need to receive all confirmed transactions/blocks.

3. OUR APPROACH

In this section, we present and detail our proposed architecture. To this end, we start by introducing our system model, after which we introduce the concept of satellite chains, and discuss how to enable asset transfer among satellite chains.

System Model.

We consider a permission-based blockchain network comprising of registered stakeholders. The platform supports smart contracts in the form of distributed applications such as those found in Ethereum [14] and Fabric [2].

Our system consists of a number of nodes that can take any of the following roles: *clients* that just send transactions in the system, *validators* that participate in the consensus, *auditors* that can passively see a selected number of transactions in the system, and *regulators* that can enforce policies (e.g., ban nodes) without necessarily participating in the consensus. Notice that nodes can take more than one role in the system.

We assume that nodes are interested in joining different committees, so called *satellite chains*, in the network. Satellite chains consist of an arbitrary number of stakeholders that share a given business logic or commonly interact with each other to fulfill a desired goal. Conforming with existing industry standards, we assume that stakeholders are not interested in sharing their transactions with parties with whom they did not establish any relationship.

Satellite Chains.

We define a satellite chain to be a distributed private ledger maintained by a subset of stakeholders in the network. These stakeholders also act as *validators* in their satellite chain,

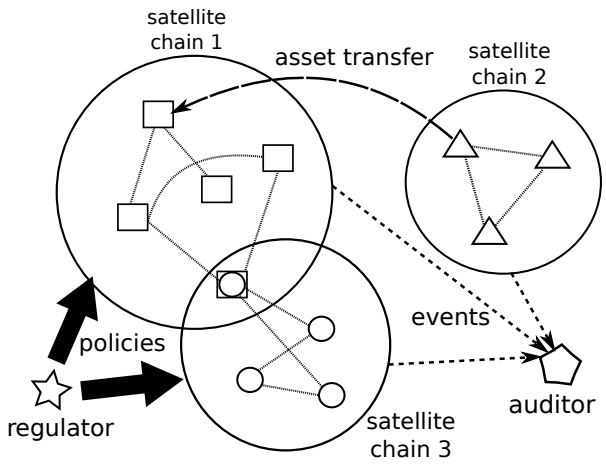

Figure 1: Our approach featuring multiple interconnected satellite chains. Validator nodes maintain the private ledger within the satellite chains while the assets can be transferred across chains. The regulators push policies into chains and the auditors subscribe to events generated by chains.

participate in the consensus, and maintain the ledger state (cf. Figure 1). Unlike Sidechains [4], our model makes no restriction on the underlying consensus layer used in each satellite chain. As we will show later, each satellite chain can decide to adopt any consensus protocol, such as PBFT [6], MinBFT [13], FastBFT [10], that does not violate the policies mandated by the regulator. However, similar to Sidechains, our solutions supports asset transfer among different satellite chains. Access control to the private ledger is maintained by each satellite chain, and is defined by a policy determined by the *validators* of the chain and the regulator. Notice that, in our solution, a single node is not bound to a single satellite chain and could join different chains simultaneously.

To form a satellite chain, a group of nodes first agrees on the validators of the chain, the consensus protocol, and the access policies. This agreement can be reached offline (i.e., using an external channel) or could be realized within the blockchain system itself. Algorithm 1 sketches a routine for forming satellite chains. This construct defines an algorithm to elect an initiator, a chain ID, and the underlying consensus protocol. All nodes involved in the chain broadcast their ID and their preferences for the consensus protocol. The sole role of the chain initiator is to facilitate the agreement process by aggregating the nodes' preferences and making a matching proposal. The initiator broadcasts the proposal to the chain nodes in order to collect their approval. Once an agreement among the nodes is reached, the satellite chain can be successfully instantiated with the agreed consensus protocol.

Notice that auditors can subscribe to any given satellite chain in order to receive notifications about selected transactions in that chain. This subscription request needs to be authorized by the chain.

Regulators.

Regulators are entities which ensure that all transactions in the network are validated and comply with some high level policy. Unlike previous proposals, regulators do not participate in the consensus protocol, and have the sole role

Algorithm 1 Satellite chain formation

Definitions:
ID_i: registered ID of node i in the blockchain
$CList_i$: proposed consensus protocol list from node i

procedure PROPOSE_CHAIN
send $ID_i, CList_i$ to all nodes
wait and collect $\langle\{ID_k\}, \{CList_k\}\rangle$ from all nodes
check $\{ID_k\}$ are all registered and retrieve their certificates
$chain_{id} \leftarrow$ getName($\{ID_k\}$)
$consensus \leftarrow$ select($\{CList_k\}$)
$chain\_info \leftarrow \langle chain_{id}, \{ID_k\}, consensus\rangle$
if isInitiator($i, \{ID_k\}$) **then**
COORDINATE($chain\_info$)
else
ACKNOWLEDGE($chain\_info$)
end if
end procedure

function COORDINATE($chain\_info$)
$sig_i \leftarrow$ sign($chain\_info$)
send $\langle chain\_info, sig_i\rangle$ to all nodes
wait and collect $\{sig_k\}$ from all nodes
if valid(sig_k) for all nodes k **then**
aggregate and send $\{sig_k\}$ to all nodes
return $\langle chain\_info, \{sig_k\}\rangle$
end if
end function

function ACKNOWLEDGE($chain\_info$)
wait for $\langle chain\_info', sig_j\rangle$ from initiator j
if isInitiator($j, \{ID_k\}$) & valid($sig_j, chain\_info'$) **then**
if $chain\_info = chain\_info'$ **then**
$sig_i \leftarrow$ sign($chain\_info$)
send sig_i to initiator j
wait for $\{sig_k\}$ from initiator j
if valid(sig_k) for all nodes k **then**
return $\langle chain\_info, \{sig_k\}\rangle$
end if
end if
end if
end function

of pushing the regulation policies to the selected satellite chains.

This is achieved by deploying the regulation policies using smart contracts. To apply these policy contracts with full flexibility, we also introduce a *policy directory contract* which is designed to manage all the *policy contracts* by providing functionalities such as registration, search by transaction type, etc. The directory contract is mandated by the regulators, and is deployed in each satellite chain. The policy directory contract listens to all publish/update events about policy contracts released by the regulator—thus ensuring that new policy contracts will be automatically deployed in their respective satellite chains and registered in the directory contract.

Moreover, we require *all* smart contracts deployed in each satellite chain to contain a hook to this policy directory contract. Namely, during transactions validation, transactions will be forwarded to the policy directory contract, which will apply appropriate policy checks before allowing the satellite chain's smart contracts to execute them. This proposal ensures that, as long as there are enough honest validators in each satellite chain, the regulation policies are enforced correctly in each chain without the need for active intervention from the regulators.

Algorithm 2 Cross-chain Asset Transfer

Definitions:
s, r: sender and recipient nodes
s_a, r_a: sender's and recipient's accounts
m: amount of transferred asset
t_i: threshold of the replies from i's chain for finality proof according to their consensus protocols
$chain\_info_i$: chain consensus configuration
$\mathsf{transfer}_f, \mathsf{reply}_f, \mathsf{reject}_f, \mathsf{receipt}_f$: transactions with finality proof

procedure TRANSFER(r_a, m)
  issue $\mathsf{transfer}(s_a, r_a, m)$ in s's chain
  wait for $\{sig_{tr}\}$ from t_s nodes
  $\mathsf{transfer}_f \leftarrow \langle \mathsf{transfer}, \{sig_{tr}\}\rangle$
  send $\mathsf{transfer}_f$ to r and **wait** for reply_f from r
  HANDLEREPLY(reply_f)
end procedure

procedure HANDLEREPLY(reply_f)
  if accept($chain\_info_r, \mathsf{reply}_f$) **then**
    if reply = receipt **then**
      store receipt
    else if reply = reject **then**
      issue reject_f in s's chain
    end if
  end if
end procedure

upon committing transfer in s's chain
  if checkPolicy(transfer) **then**
    $s_a \leftarrow s_a - m$
    $sig_{tr} \leftarrow$ sign(transfer)
    send sig_{tr} to node s
  end if

upon receiving $\mathsf{transfer}_f$ at r
  issue $\mathsf{transfer}_f$ in r's chain
  wait for $\{\mathsf{reply}, sig_{rep}\}$ from t_r nodes
  $\mathsf{reply}_f \leftarrow \langle \mathsf{reply}, \{sig_{rep}\}\rangle$
  send reply_f to s

upon committing $\mathsf{transfer}_f$ in r's chain
  reply $\leftarrow \langle$"*reject*", transfer$\rangle$
  if accept($chain\_info_s, \mathsf{transfer}_f$) **then**
    if checkPolicy(transfer) **then**
      $r_a \leftarrow r_a + m$
      reply $\leftarrow \langle$"*receipt*", transfer$\rangle$
    end if
  end if
  $sig_{rep} \leftarrow$ sign(reply)
  send $\langle \mathsf{reply}, sig_{rep}\rangle$ to r

upon committing reject_f in s's chain
  if accept($chain\_info_r, \mathsf{reject}_f$) **then**
    $s_a \leftarrow s_a + m$
  end if

Cross-chain Asset Transfer.

As mentioned earlier, each satellite chain processes transactions therein and maintains the ledger independently among the member stakeholders. However, in some use cases, assets need to flow between stakeholders across multiple satellite chains. This would be beneficial, for example, in cross-border payments between financial institutions that belong to different administrative domains (i.e., different satellite chains).

To address this problem, our proposal supports transferring assets between independent satellite chains. As shown in Algorithm 2, asset transfers involve moving a particular asset m (e.g., a payment or an investment) from a sender s to a recipient r, such that s and r belong to two different satellite chains. Figure 2 shows the work flow of the asset transferring process.

To initiate the transaction, s first issues a transfer request to its satellite chain. All validators in s's chain verify the transaction, e.g., check if m does not exceed some upper bound limit and s's account holds m assets. If the policy check (*checkPolicy*() in Algorithm 2) passes, the validators execute the transaction by removing m assets from s's account, and return to s an authenticated response approving the transfer transaction. Once s collects enough authenticated responses from the validators, s combines them into a *finality proof* that represents the aggregated approval from all validators of s's chain, and proves that the transaction is indeed finalized and executed in that chain. The number of required approvals depends on the underlying consensus protocol. For instance, if the consensus protocol of s's chain is PBFT, then, given the full list of n validator certificates, the finality proof should include at least $\lfloor (n-1)/3 \rfloor + 1$ signatures. Notice that the asset transfer token (i.e., transfer) also specifies the recipient chain where the asset should be spent. As a result, the token will not be accepted in any other chain.

s then sends this finality proof to r's chain, which verifies it given the chain information it contains, such as the list of validator certificates, and the underlying consensus protocol (*accept*() in Algorithm 2). If the finality proof is accepted, the validators of r's chain further verify if the transfer transaction from s to r is allowed by the policies. If all checks are passed, r's validators move m assets to r's account and together they return to s a receipt token with the finality proof of r's chain. Otherwise, the chain sends back a reject token that can be used by s to restore its account in s's chain. The messages transmitted between the sender and recipient chain are carried out using a direct channel between s and r. If s receives neither a receipt nor a reject token within some timeout, he can resend the transfer request to another node in r's chain.

Given the above described procedure, we argue that the assert transfer operation is atomic at all times: the sender's and recipient's accounts are either both updated in their respective satellite chains or both unchanged. We do not allow implicit roll-backs based on timeout since the process is asynchronous. Only explicit roll-backs with reject token from the recipient chain are allowed. More specifically, the sender s expects either a receipt token or a reject token from r's chain to conclude the asset transfer. Since the consensus layer of the satellite chains guarantees availability and consistency, we argue that s will eventually get a response from r's chain. If the transfer operation is successful, only one consensus round is required within each chain; if the operation fails in the recipient chain, the sender chain is required to execute an additional round of consensus to roll back the sender's account.

4. INTEGRATION WITH HYPERLEDGER

In this section, we discuss various insights with respect to the integration of our proposed architecture with Hyperledger Fabric v0.6 (see Section 2). Namely, Hyperledger Fabric does not support multiple chains/ledgers, nor does it provide functionality for cross-chain asset transfer. We therefore discuss various required adaptations of Fabric in order to implement our proposal. We additionally comment on the integration with the upcoming release of Hyperledger v1.0.

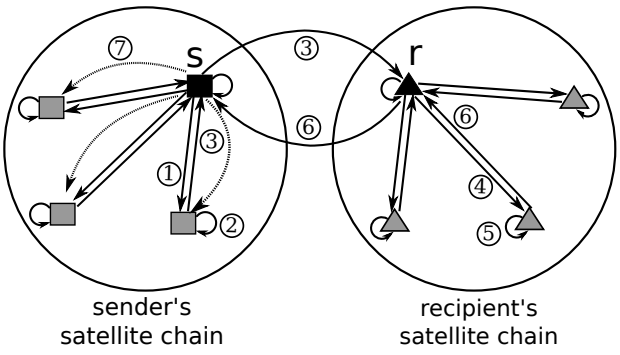

Figure 2: Atomic asset transfer between the sender node (s) and recipient node (r) across two satellite chains: ① s issues transfer transaction; ② nodes verify and execute transfer; ③ s aggregates authenticated approvals on transfer and forwards to r; ④ r issues transfer$_f$ transaction; ⑤ nodes verify and execute transfer$_f$; ⑥ r aggregates authenticated approvals on either receipt or reject and sends back; ⑦ s issues reject$_f$ to restore account in case the transaction fails in the recipient's chain.

Instantiating Regulators.

We extended Fabric's Membership Services to comply with our solution. Namely, we assume that the regulator orchestrates the membership services for the entire blockchain network. Moreover, the regulator stores a data structure listing all the satellite chains registered to it, together with information about members of each chain (i.e., IP address, TCP port, node ID). Each satellite chain must register itself to the regulator services upon formation.

Additionally, in our system, regulators enforce policies by means of chaincodes; that is, for each policy the regulator provides one (or more) chaincode(s). Each satellite chain must deploy the policy chaincodes together with their own chaincodes.

Independent Ledgers.

To instantiate independent ledgers, nodes need to maintain multiple separate ledgers, one for each satellite chain they joined. In our implementation, we adapted Fabric nodes' core code to create a new database to store the ledger whenever a node joins a new satellite chain. For this purpose, we leverage Fabric's RocksDB key-value storage to store each ledger on disk. The node then includes a pointer to the newly created database in a data structure dubbed *chain-to-ledgers*. The *chain-to-ledgers* data structure is a map that stores $(key, value)$ pairs of $(chain\_id, ledger\_db)$ listing all the satellite chains the node has joined and a pointer to the database storing their ledger, that is, the transactions exchanged within that satellite chain. This data structure is accessed whenever a newly validated transactions needs to be added to the satellite chain ledger's database.

Additionally, we added new components to the node's core code: a *chain-to-consensus* data structure, a *chain-to-ledgers* data structure and a *chain-to-peers* data structure. The *chain-to-consensus* data structure is a map that stores $(key, value)$ pairs of $(chain\_id, consensus\_plugin)$ listing all the satellite chains the node has joined and their corresponding consensus protocol. Recall that the satellite chain ID and consensus protocol are agreed with an off-chain protocol by its members. Whenever a node joins a new satellite chain, it passes this data to the fabric's peer so that it is stored in the *chain-to-consensus* map. The *chain-to-peers* data structure is a map that stores $(key, value)$ pairs of $(chain\_id, peers\_list)$ listing all the satellite chains the node has joined and a list of peers participating in that satellite chain. The list of participants of a satellite chain is obtained as a result of the same off-chain protocol that the nodes use to agree the satellite chain ID and consensus protocol. Therefore, the participant's list is also passed to the fabric's peer whenever a node joins a new chain.

Whenever a satellite chain successfully validates a transaction, each node adds it to the chain's ledger by accessing the *chain-to-ledgers* map with the *chain_id* to retrieve the pointer to the ledger's database. The transaction is then added to the ledger by querying the database.

Asset Transfers.

To incorporate the cross-chain assets transfer functionality within Fabric, we allow nodes to establish direct connections to send assets transfer messages over TLS.

Additionally, transactions now include their target satellite chain ID—the *chain_id*—in their payload. The receiving node broadcasts the transaction to only the nodes participating in the satellite chain specified in the transaction's payload. The node knows to which nodes it must send the transaction by retrieving the peers list from the *chain-to-peers* map introduced above. The receiving node than issues an *invoke* transaction to the desired chaincode using *chain_id* and *chaincode_id* as identifiers.

Enforcing Policies.

Recall that each satellite chain must deploy the policy chaincodes together with their own chaincodes.

The regulator provides a *policy directory chaincode* that manages the policy chaincodes to validate transactions, and collects their validations' results. This special chaincode is deployed in each satellite chain, and listens for update events about policy chaincodes from the regulator (cf. Figure 3). This is achieved by leveraging Fabric's event framework to push chaincode updates. Upon the receipt of an update event, the policy directory chaincode downloads the new—or the updated—policy chaincode and deploys it to the satellite chain so that it can be used to validate transactions.

Whenever a chaincode executes a transaction, it passes the transaction to the policy directory chaincode. The latter then looks up the list of deployed policy chaincodes, and forwards the transaction further. Subsequently, policy chaincodes determine if the transaction violates any of their defined policies, and return the result of the validation to the policy directory chaincode. The validation is unsuccessful if *at least one* fails. The validation is successful if *all* policy validations complete successfully. In that case, the chaincode then continues processing the transaction.

Integration with Hyperledger Fabric v1.0.

Hyperledger Fabric v1.0 plans to introduce a clear separation between node roles: clients, submitters, endorsers, and consenters. Clients connect to a submitter, issue a transaction, and wait for the result of execution. Submitters act as proxies for clients, and connect to endorsers in order to execute the transaction on behalf of the client. Endorsers process

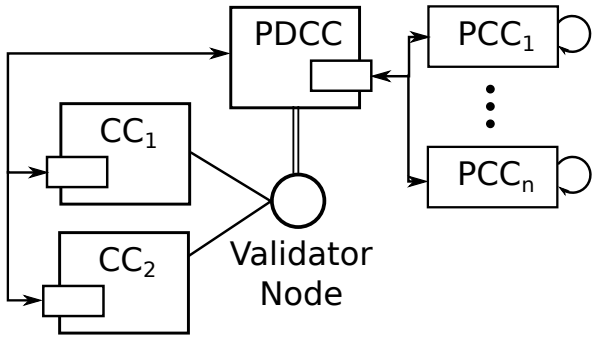

Figure 3: Deployed policy chaincodes ($PCC_1 \dots PCC_n$) are registered by the policy directory chaincode ($PDCC$), which is a default system chaincode deployed in each satellite chain in the network. Each application chaincode (CC_i) verifies the received transactions against all regulation policies registered by the $PDCC$.

the transaction by invoking smart contracts (i.e., chaincodes) executing it, and endorsing it. Notice that chaincodes may specify an endorsement policy which defines a set of requirements for a valid transaction endorsement. Finally, if the endorsement is successful, the transaction will be forwarded to the consenters who will run consensus to include it in the global ledger. Clearly, this architecture allows to parallelize endorsements execution amongst various endorsers. However, although Hyperledger Fabric allows multiple private ledgers and consensus established over multiple channels, it does not support asset transfers among these ledgers. Moreover, there is no support for a hands-off regulator to enforce policies on multiple channels: the regulator has to be involved as an active endorser to vote on all the transactions.

We argue that our proposed architecture can further improve the provisions of Fabric v1.0 to better fit the industrial use cases. Namely, satellite chains can be implemented using multiple consensus channels in the framework; we can also achieve asset transfer among different chains in a similar way as in v0.6. Moreover, the regulator policy enforcement scheme can also be easily integrated with the *system chaincode* in Fabric v1.0. We therefore foresee no major obstacles in integrating our proposal within the planned Hyperledger Fabric v1.0.

5. OUTLOOK

In this paper, we proposed a new blockchain architecture devised specifically to meet industrial standards. Our proposal leverages the notion of *satellite chains* that form interconnected but independent subchains of a single blockchain system, and supports an unbounded number of active chains that can run in parallel at any point in time. Moreover, our proposal allows different satellite chains to transfer assets among themselves without compromising the security and soundness guarantees in the system, and accounts for the role of a passive regulator that can at any point in time enforce specific network-wide policies.

Our proposal is agnostic of the underlying consensus protocol utilized in the network, and as such can be easily integrated within existing blockchain platforms such as Hyperledger Fabric and Corda. Moreover, our proposal supports heterogenous consensus protocols by allowing different satellite chains to execute different consensus protocol as long as such consensus is in line with the policies set by the regulator. As a proof of concept, we discussed various insights on its integration within Hyperledger Fabric v0.6.

Notice that by effectively organizing nodes within interconnected satellite chains, our proposal practically enables the realization of blockchain sharding based on functional requirements such as node relationships. Blockchain sharding has received considerable attention in the literature; while there are a number of security challenges that effectively hinder practical sharding (for load balancing), our solution groups relevant stakeholders in a single shard. We believe that our solution finds direct applicability in a number of emerging industrial blockchain applications, such as trade finance, asset management, supply chain management, and retail services. We therefore hope that our findings motivate further research in this area.

6. REFERENCES

[1] Hyperledger – Blockchain Technologies for Business. https://www.hyperledger.org/.

[2] Hyperledger Fabric. https://hyperledger-fabric.readthedocs.io/en/v0.6/.

[3] Ripple. https://ripple.com/.

[4] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille. Enabling Blockchain Innovations with Pegged Sidechains. 2014.

[5] A. Back et al. Hashcash-a denial of service counter-measure, 2002.

[6] M. Castro, B. Liskov, et al. Practical Byzantine Fault Tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[7] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59. USENIX Association, 2016.

[8] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[9] M. Hearn. Corda – A distributed ledger. *Corda Technical White Paper*, 2016.

[10] J. Liu, W. Li, G. O. Karame, and N. Asokan. Scalable Byzantine Consensus via Hardware-assisted Secret Sharing. *arXiv preprint arXiv:1612.04997*, 2016.

[11] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A Secure Sharding Protocol For Open Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[12] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[13] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo. Efficient Byzantine Fault-Tolerance. *IEEE Transactions on Computers*, 62(1):16–30, 2013.

[14] G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper*, 2014.