SMChain: A Scalable Blockchain Protocol for Secure Metering Systems in Distributed Industrial Plants

Gang Wang, Zhijie Jerry Shi
University of Connecticut
{gang.wang,zshi}@uconn.edu

Mark Nixon
Emerson Automation Solutions
mark.nixon@emerson.com

Song Han
University of Connecticut
song.han@uconn.edu

ABSTRACT

Metering is a critical process in large-scale distributed industrial plants, which enables multiple plants to collaborate to offer mutual services without outside interference. When distributed plants measure the data from a shared common source, e.g., flow metering in an oil pipeline, trustworthiness and immutability must be guaranteed among them. In this paper, we propose a hierarchical and scalable blockchain-based secure metering system, *SMChain*, to provide strong security, trustworthy guarantee, and immutable services. *SMChain* adopts a two-layer blockchain structure, consisting of independent local blockchains stored at individual plants and one state blockchain stored in the cloud. To deal with the scalability issues within each plant, we propose a novel scalable Byzantine Fault Tolerance (BFT) consensus protocol based on *(k, n)*-threshold signature scheme to deal with the Byzantine faults and reduce the intra-plant communication complexity from $O(n^2)$ to $O(n)$. For the state blockchain, we use a cloud-based service to synchronize and integrate the local blockchains into one state blockchain, which can further be distributed back to each plant.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; • **Information systems → Hierarchical storage management**; • **Applied computing → Industry and manufacturing**;

KEYWORDS

Blockchain, Secure Metering System, Consensus Protocol, BFT

ACM Reference Format:
Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. 2019. SMChain: A Scalable Blockchain Protocol for Secure Metering Systems in Distributed Industrial Plants. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Metering, or smart metering, refers to the procedure of installing intelligent meter-reading systems, reading meters remotely, and transporting the readings to the processing devices, such as edge gateways or central data servers. Metering systems have been pervasively deployed in industrial plants. These metering systems from independent and distributed plants (entities), in some scenarios, take measurements from a common resource. For example, in an oil pipeline system, the oil is transported through long-distance pipelines to its destinations (i.e., gas stations), and distributed entities might perform independent measuring to monitor the oil states in the oil pipeline. These measurements are typically made to receive some form of compensation from the utilization or transfer of the common resource. Multiple entities participate in this arrangement to share and negotiate with each other using the measurement data to determine fair compensation. Since each entity can be operated by different vendors without a central controller, and they do not share their own information, there is a lack of trust among the participating entities. Due to the competition, malicious entities might modify the data measurements in its own plant for its own benefits. To guarantee fairness, an immutable record must be established among multiple plants. Distributed ledger technology, such as Blockchain, due to its properties of decentralization, verifiability, and immutability, provides an ideal solution to enhance security (in conjunction with cryptographic primitives), and service availability (to avoid a single point of failure).

To build a distributed ledger, multiple entities require to establish a consensus in the distributed manner. Proof-of-Concept (PoC) mechanisms, e.g., Proof-of-Work (PoW) in Bitcoin [1], cannot meet the above requirements in distributed industrial plants since PoC mechanisms require every entity to participate in the consensus process. On the other hand, Byzantine fault tolerance (BFT) scheme fits for this scenario with instant finality which can reduce the latency to confirm the transactions and improve the overall throughput. However, most BFT protocols, such as practical BFT (PBFT) [2], have been perceived to be communication-heavy, which prevents them from being used for large-scale distributed industrial plants.

To integrate distributed ledger, or more specifically, blockchain technologies, into distributed industrial plants, we need the BFT protocol to be scalable. Two metrics are directly related to BFT scalability: transaction throughput (maximum rate that the blockchain can process transactions) and latency (time to confirm that a transaction has got an agreement). Also, there exist two ways to reduce communication complexity: reducing the number of participating nodes or the number of communication messages. Since it is not practical to reduce the participating nodes for individual plants, this paper focuses on reducing the communication messages.

In this paper, we propose a novel blockchain structure, SMChain, which is devised specifically to meet data immutability and trustworthiness among industrial plants. Our proposal mainly focuses on the designs of the hierarchical chain structure and the scalable consensus protocol. Specifically, we adopt a two-layer blockchain design to implement SMChain, leveraging the local chains that form independently within its own plant. Each plant is responsible for its own local chain without sharing information with other plants, and uses a scalable BFT protocol to build the local chain. The BFT protocol leverages the crypto-primitives, threshold signatures [3], to reduce the communication within each plant. Finally, we utilize the cloud to integrate the local chains to form a single state blockchain, which is then distributed back to each plant.

Conference'17, July 2017, Washington, DC, USA
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The rest of the paper is organized as follows. Section 2 provides the preliminaries. Section 3 gives an overview of the SMChain architecture. Section 4 and Section 5 describe the two-layer hierarchical chain design and the scalable BFT protocol. Section 6 provides the security analysis and Section 7 concludes the paper.

2 PRELIMINARIES

2.1 Blockchain

A blockchain is a distributed and shared ledger that serves as an irreversible and incorruptible public repository. It enables unrelated participants to reach the consensus on the occurrence of a particular transaction or event without the need for a centralized authority. Compared to a traditional database system to keep the data consistency, blockchain offers several important features: (1) distribution – the distribution can help reduce the risk of tampering; (2) decentralization – without a single centralized authority, blockchain can increase network efficiency and security (e.g., eliminating "middle-man" attacks in P2P network), and reduce costs in centralized infrastructures; (3) trustlessness – no trusted third party needs to certify the transactions, which allows digital transactions to happen between parties who do not trust each other; (4) immutability – based on the implementation of cryptographic hash functions, the blockchain is immutable. By identity management, the blockchain technology typically can be categorized into two types: permissionless blockchain and permissioned blockchain [4]. In our framework, we consider the permissioned blockchain.

2.2 Practical BFT protocol

The practical BFT protcol (PBFT) [2] was initially designed under the assumption that replicas communicate over a LAN that has few failures. There is also a rich body of work to provide algorithmic and system design improvements based on PBFT [5–7]. Due to Fisher-Lynch-Paterson (FLP) impossibility [8], PBFT leverages the weak synchrony assumption under which messages are guaranteed to be delivered after a certain time bound [5].

PBFT can tolerate up to 1/3 Byzantine faults. In the following, we briefly describe its consensus procedures. One replica, the *primary*/*leader* replica, decides the order for clients' requests, and forwards them to other replicas, the *secondary* replicas. All replicas together run a three-phase (pre-prepare/prepare/commit) agreement protocol to agree on the order of requests. Each replica processes every request and sends a response to the corresponding client. On detecting that the leader replica is faulty through the consensus procedure, the other replicas trigger a *view-change* protocol to select a new leader. The leader-based protocol works very well in practice and is suitable in industrial cases. It however is subject to the scalability issues, which is going to be addressed in this paper.

2.3 Network and Threat Models

Network Model. In this work, we consider a peer-to-peer network with n nodes who establish identities (i.e., public/private key pairs) through each own certificate authority (CA). We assume all messages sent in the network are authenticated with the sender's private key. Also, we assume a partial synchronous model adopted by most blockchain protocols. That is, the messages are propagated through a synchronous gossip protocol [9] that guarantees a message sent by an honest node will be delivered to all honest nodes within a known fixed time, Δ. However, the order of these messages is not necessarily preserved.

Threat Model. Our protocol considers a probabilistic polynomial-time Byzantine adversary who can corrupt at most f nodes among $3f+1$ participating nodes at any time. The corrupted nodes not only may collude with each other but also can deviate from the protocol in any arbitrary manner, e.g., by sending invalid or inconsistent messages, or remaining silent. We also assume nodes may disconnect from the network during an epoch (one epoch is defined as the time to form one block) or between two epochs due to reasons such as internal failure or network jitters. In addition, we assume, at any moment, at least 2/3 of the computational resources belong to the uncorrupted participants that are online, e.g., response within the network time bound.

3 SMCHAIN ARCHITECTURE OVERVIEW

We now give an overview of the SMChain architecture. Our proposal leverages the notion of local chains that are formed independently within the industrial plants. Each local chain maintains its *own* private ledger, thus preventing any non-member (e.g., an adversary) from modifying it at any time. Different local chains may run different consensus protocols in parallel, which further allows unprecedented levels of scalability in large-scale industrial plants. Within each plant, we propose to use a scalable BFT consensus protocol to deal with the scalability issues.

Fig. 1 (a) shows the blockchain-based secure metering architecture. It has three major components: local metering networks, local blockchain network, and the cloud. A local metering network consists of various metering devices, and each metering device sends the meter readings to its corresponding edge gateway. The edge gateways are connected to each other, forming a local blockchain network. The edge gateways in individual plants play the key role to connect physical resources (e.g., metering devices) to the cloud, and build the local blocks. It leverages a consensus protocol to form the local chain within its plant. For each plant, we consider a permissioned blockchain network comprising of the registered and authorized edge gateways as its participating nodes. Each edge gateway functions as the full node, e.g., it can collect and send transactions to its peers in local blockchain network, and verify the blocks from other nodes. The cloud is responsible to construct the state blockchain from the synchronized local chains.

With the above architecture, we construct a hierarchical chain structure called *SMChain*, which aims for blockchain-based *S*ecure *M*etering systems. Fig. 1 (b) shows a conceptual structure of *SMChain*.

SMChain has a two-layer chain structure and each chain consists of blocks, chronologically chained by a hash. The first layer blockchain is called the *local chain*, which is built within an *independent* industrial plant, while the second layer blockchain is called the *state chain*. Each plant has its own local chain, however, all the industrial plants share one state chain. This two-layer blockchain includes two block types, data block and state block, as to be detailed in Section 4.1.2. A data block is proposed by the edge gateways within a single plant, while a state block is proposed by the final

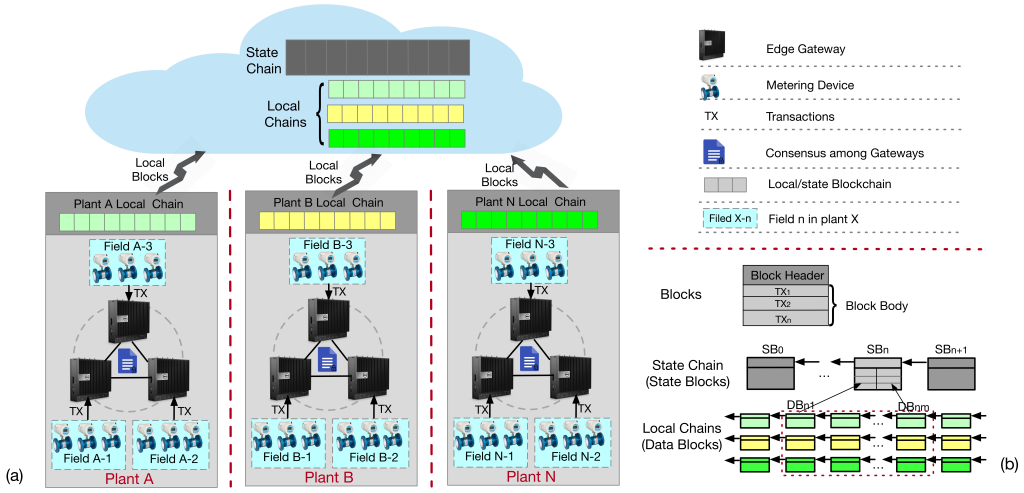

Figure 1: (a) An overview of the SMChain architecture with three components: local metering networks, local chains stored in the plants, and the state chain stored in the cloud; (b) The conceptual chain structure between local chains and the state chain.

designated cloud to include all the data blocks generated in a specific epoch. The local chains are built using scalable BFT consensus protocol, while the state chain is formed in the cloud using cloud services. The industrial plants may select one single cloud as the cloud service provider. In this case, we assume this cloud is trustworthy and resistant to attackers. Alternatively, individual plants might have different clouds or different storage nodes within one single cloud. In this case, the clouds might run a consensus protocol to get a shared state chain. In this paper, for simplicity, we assume there is only one trusted and secure cloud infrastructure.

4 TWO-LAYER BLOCKCHAIN DESIGN

This section first describes the key components of the hierarchical chain structure, and then presents the details of the chain design.

4.1 Transactions and Blocks

4.1.1 Transaction. A transaction (TX) in *SMChain* represents a meter reading from a metering device. The gateway, after receiving these transactions, uses a Merkle tree to cryptographically commit a set of transactions into a data block. And, Merkle tree allows all transactions to be committed in a single hash value stored by the Merkle root (as showing in Fig. 2).

4.1.2 Blocks. Transaction data are permanently recorded in files called blocks. New transactions are constantly being processed by consensus nodes into new blocks which are then appended at the end of the chain. In SMChain, we adopt a two-layer blockchain structure (e.g., data blockchain and state blockchain), in which each layer contains its own block information.

Data Block: the block in the first layer of the hierarchical chain structure is called data block. Data block is directly related to transactions, which come from the physical resources and local metering networks. Each data block has a block header and a block body.

State Block: the block in the second layer of the hierarchical chain structure is called state block. Similar to the data block, a state block also has a block header and a block body. The block body includes the headers of a set of data blocks from the first layer.

4.2 Details of the Chain Design

We now describe how to build the two-layer blockchain structure.

The local chain is a private ledger maintained by edge gateways in a single plant, and each plant does not share its transactions with other plants with whom it has not established any relationship yet. The edge gateways within one plant have a full connection (i.e., P2P) to run a specified consensus protocol. Typically, the consensus protocol is based on the *epoch*, which specifies the maximum time to form one block. Here we briefly describe the generation process of one data block. Edge gateways first need to elect one gateway as the primary node. The role of the primary node is to facilitate the agreement process. Then, the primary node starts to collect the transaction requests (i.e., meter readings) from the local metering networks and broadcast these requests to all other edge gateways. Following the proposed scalable BFT consensus protocol (see Section 5 for details), once the edge gateways reach an agreement, a local block is successfully formed. Following the traditional blockchain protocol, the new generated local block is appended to its local chain. Please note that each local chain functions independently in its corresponding plant.

SMChain operates based on epoch, and we distinguish the epoch to generate a data block (namely *data epoch*) from the epoch to generate a state block (namely *state epoch*). One state epoch might contain multiple data epochs. The state block describes the state information of multiple separated local chains. Within a state epoch, each plant uploads its data blocks (generated in that state epoch), via either the primary replica of the last view or a specific edge gateway, to the cloud. According to the specified local chain ID, the cloud then links the uploaded local data blocks to its corresponding local chain. After every plant uploads its local chain generated in a state epoch, the cloud extracts the first and the last data block information and inserts them to the block body of the state block. This process is called the local chain aggregation. Once a state block is formed, following the traditional blockchain mechanism, the new generated state block is chained to the state chain.

Fig. 2 shows an example of the overall chain design in SMChain. Assume we have 4 plants, named from *000* to *011*, respectively, and

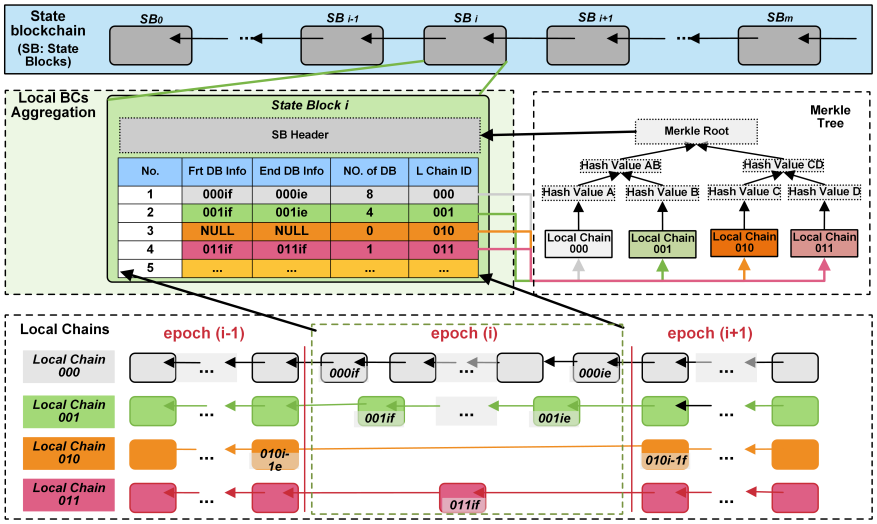

Figure 2: Blockchain design in SMChain. Each block epoch has only one state block but multiple data blocks.

each plant has its own local chain. Taking *plant 000* as an example, it has a *local chain 000*. Assuming in state epoch i, the *local chain 000* generates a local chain piece, containing 8 local data blocks. The first data block in this local chain piece is data block *000if* and the tail data block is data block *000ie*. Then, the cloud extracts the first and last data block information, and put these information into "slot 1" of the block body in state block i. Once every local chain finishes this process, the cloud computes a Merkle root from the extracted local chain pieces, and put this Merkle root in the header of the state block i. Following the traditional blockchain mechanism, the state block i is then chained to the state blockchain.

5 SCALABLE BFT PROTOCOL DESIGN

PBFT protocols are well known to become bandwidth-bound and latency-bound quickly when the number of involved network identities grows. To achieve a good balance between performance and scalability in *SMChain*, we design a scalable BFT protocol to meet the requirements in large-scale industrial settings. The key idea of the scalable BFT protocol is to utilize threshold signature, a cryptographic primitive, to avoid message broadcast, which significantly reduces the communication complexity in the consensus process.

5.1 Cryptography Primitives

We first introduce the threshold signature, which is a signature scheme used for the scalable BFT consensus.

In threshold signature, for a threshold parameter k, any subset of k from a total of n signers can collaborate to produce a valid signature on any given message, but no subset of less than k can do so. Each signer holds a distinct private signing key that it can use to generate a signature share. A (k, n) threshold digital signature scheme allows a set of signers to generate a digital signature as a single logical entity despite $(k-1)$ Byzantine faults. By dividing a private key into n shares, each one owns by a signer. Each signer uses its key share to generate a partial signature on a message m and sends its partial signature to a combiner signer, which combines the partial signatures into a threshold signature on m. In our scalable BFT protocol, we use a robust threshold signature scheme based on Boneh-Lynn-Shacham (BLS) signature [10]. BLS signature is a short signature scheme based on the computational Diffie-Hellman assumption on certain elliptic and hyperelliptic curves.

5.2 Consensus Protocol

A BFT protocol typically has two components: a consensus protocol to describe how to reach an agreement among participating nodes, and a view-change protocol to deal with abnormal cases.

In PBFT, the primary S_p decides the order for clients' requests, and forwards them to other replicas S_is. All replicas together run a three-phase agreement protocol to agree on the order of requests. Each replica then processes each request and sends a response to the corresponding client. Agreement in PBFT requires each S_i to *multicast* a commit message to all (active) replicas to signal that it agrees with the order proposed by S_p, which leads to $O(n^2)$ communication complexity. Our scalable BFT protocol uses a threshold signature to reduce the complexity to $O(n)$: during commitment, instead of using multicast to reply to all replicas, each active replica S_i sends its commit message directly to the primary S_p and the threshold signature is used to verify the committed messages. Since the secret shares are one-time, the proposed scalable BFT protocol uses an additional *pre-processing* phase to distribute secret shares from the *primary* to the participating replicas.

This consensus procedure of the scalable BFT protocol is elaborated below and illustrated on the left side of Fig. 3.

- *Pre-processing*: In this phase, S_p generates a set of random secret shares using BLS signature, and publishes the cryptographic hash of each secret. Then, S_p sends one share to each active S_i. At the end of this phase, each active replica (or signer) holds a distinct private signing key that can be used to generate a signature share.
- *Request*: In this phase, the clients (e.g., metering devices) send operation requests to S_p (e.g., metering data). Once S_p receives the requests, it creates a decision block from the gathered transactions.
- *Prepare*: In this phase, S_p sends the decision block to active replicas. Those replicas sign the requests using their private signing keys to get the signature (e.g., a *sign-share* message), and then prepare its commitment proof about its decision.

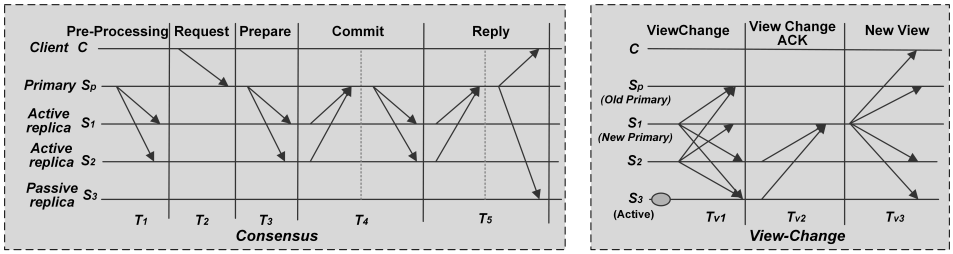

Figure 3: The consensus procedure of the proposed scalable BFT protocol.

• *Commit*: In this phase, each active S_i signals its commitment by sending its *sign-share* message, together with its commit proof (this proof is signed by S_i's own private key). S_p then gathers all the signature shares, and logs all received commit proofs together to create a succinct *full-commit-proof* for the decision block. Besides, S_p also gathers all secret shares to reconstruct the secret, which represents the aggregated commitment of all replicas. S_p multicasts the reconstructed secret and the full-commit-proof to all active $S_i s$, which is sufficient for replicas to verify.

• *Reply*: The same approach is used to aggregate reply messages from all active S_i. In this last phase, after each active S_i verifies the secret, S_i reveals its share of the secret to S_p, which reconstructs the reply secret and returns it to the client and all passive replicas. In this case, the client and passive replicas only need to receive one reply, instead of $f + 1$ replies.

Like most PBFT protocols, the primary S_p is initially assumed to be trusted when it broadcasts the client's request to all other active replicas. Each replica works independent. Only if it verifies the correctness of the blocks, it sends the response to the primary S_p during the *Commit* phase. Otherwise, the replica will hold its response, since we assume 2/3 active participating nodes are honest. Also, each message sent to its peers is signed by the sender, and the signature cannot be compromised by any adversary. By carefully designing the security parameter k in (k, n)-threshold signature, the scheme can reach the consensus among the majority of the honest participants. Typically, we set $k \geq f + 1$. Even in the presence of a malicious primary, e.g., knowing all secret shares, it cannot compromise the commit proof sent by each active replica. Without enough commit proofs, e.g., $f + 1$ commit proofs in full-commit-proof message, the honest replicas would not agree on this round of consensus process, thus invoke a view-change process.

5.3 View-Change Protocol

The replicas can trigger a view-change protocol (see the right side of Fig. 3) when the consensus procedure fails. In the consensus procedure, the client sets a timer after sending a request to S_p. It will broadcast the request to all replicas if no reply is received from S_p before the timeout. If an active replica does not receive the *pre-pare/commit/reply* messages in the corresponding phases before the associated timeouts, it will initiate a view change by broadcasting a $<$*REQ-VIEW-CHANGE*, L, $< H(L), (v) >_{\sigma_i} >$ message, where L is the message log that includes all messages it has received/sent since the latest checkpoint, $H(L)$ is the cryptographic hash function on message L, v represents the current view and σ_i represents a signature on view by S_i. During the scalable BFT consensus, we run optimistic BFT where only a subset of replicas are required to run the agreement protocol; other replicas passively update their states and become actively involved only if the agreement protocol fails. After receiving this request from the client, the passive replicas become active. Replicas can also suspect that S_p is faulty by verifying the message they received and initialize a view-change request. Besides, the passive replicas can also send *REQ-VIEW-CHANGE* messages, this guarantees that if faulty primary occurs, there will be always $f + 1$ non-faulty replicas that initiate the view-change requests.

Upon receiving $f + 1$ *REQ-VIEW-CHANGE* messages, the new primary $S_{p'}$ (assuming $p = v \ mod \ n$ and $v' = v + 1$, then $p' = v' \ mod \ n$) constructs the execution history O by collecting all pre-pared/committed/replied requests from the message logs. The constructed logs guarantee that replicas will always process the same execution history. Based on the above information, the view-change protocol consists of three major phases:

• *View Change*: The replicas receiving the exception message will broadcast a *REQ-VIEW-CHANGE* message to other consensus replicas, including passive replicas.

• *View Change ACK*: After receiving the *REQ-VIEW-CHANGE* message, each replica verifies the validity and replies a *REQ-VIEW-CHANGE-ACK* message to validate replicas.

• *New View*: The leader/primary of the next round broadcasts a *New View* message to all other consensus replicas to process the next consensus if the number of received *REQ-VIEW-CHANGE-ACK* messages is more than 2/3 of the total number of replicas.

If the above phases finish successfully, all replicas will switch to a new view with new primary $S_{p'}$.

6 SECURITY ANALYSIS

We now present the security analysis for the proposed scalable BFT protocol regarding its safety and liveness. *Safety* is the property that if any two non-faulty replicas commit on a decision block for a given sequence number, then they both commit on the same decision block; while *liveness* is to ensure that the consensus protocol makes progress in the current view and moves to a new view, which means clients eventually receive replies to their requests. In our scheme, we assume $n >= 3f + 1$, where n is the number of replicas, and f is the number of Byzantine replicas. We further assume at most f replicas are passive replicas, so that the total participating replicas are at least $2f + 1$. Let $h = SHA256(r||v)$, where r is a batch of requests (req) from clients and v is the current view number.

6.1 Safety

If a non-faulty replica commits due to a signature $\sigma(h)$ which is induced by req then at least $f + 1$ non-faulty replicas received h as a pre-prepare hash during *pre-processing* from the primary.

So at most f replicas can send a different prepare message with $h' \neq h$, hence *req* or r is the unique value in the current view v that receives at least $f+1$ prepare messages. The scheme sets the threshold signature as *(f + 1, n)*-threshold signatures. Thus, the adversary cannot create a signature $\sigma(h')$ where $h' \neq h$. Hence, no non-faulty will commit to any $h' \neq h$ at view v.

Moreover, at any view-change to a new view $v' > v$, we can show by induction that the new primary must choose the value *req*. For the base case $v' = v + 1$, there will be at least $f + 1$ non-faulty replicas that will report these prepares of h with view v to the new primary $S_{p'}$. Since the view v' is the highest new to potentially adopt, this view will be the adopted value. The inductive argument is similar, since every new leader must adopt *req*, then for any view change, there will either be $f + 1$ that report *req*, or there will be a signature $\sigma(h')$ where h' uses *req*. However, due to the setting of the threshold signature, we know that $\sigma(h')$ will not be accepted for verification. Furthermore, one important observation is that the maximality of this view value continues to hold due to induction. The proof continues by induction for any view $v' > v$. Assuming that in all views v and v', it must have been the case that the primaries have adopted the hash h that is based on *req*. From the above arguments, it is shown that the new primary for view v' must also adopt the value *req* as the decision block.

6.2 Liveness

We say the client's request completes when it accepts the reply, which needs to show that an *req* from a correct client eventually completes. If the primary is correct, then we can say a view is *stable*.

We claim that during a stable view, a *req* from a correct client will complete. If the primary S_p is correct as assumed, then a valid prepare message will be sent to the active replicas. If all active replicas behave correctly, the request will complete with at least $f+1$ replicas, and the request will get verified. Alternatively, a faulty replica, say S_f may either crash or reply with a wrong share. This behavior will be detected by the primary, and S_f will be replaced by a passive replica. If a threshold number of failure detection has been reached, correct replicas will initiate a view-change, and the view-change will succeed as long as the primary S_p is correct. Thus, the request will complete as long as the number of non-primary faulty is at most f.

If the view is not stable, and at least $f+1$ correct replicas request a view-change, then a view v eventually will be changed to a stable view. This case has three different scenarios:

(a). The new primary $S_{p'}$ is correct and all other f replicas received a valid new-view message, then they will change to a stable view successfully.

(b). None of the correct replicas received a valid new-view message, in which another view-change will start.

(c). Only a set of less than $f + 1$ correct replicas received a valid new-view message. (1) Faulty replicas can follow the protocol to make the correct replicas change to a non-stable view. (2) Other correct replicas will send new *REQ-VIEW-CHANGE* messages due to timeout, however, a view-change will not start since they are less than $f + 1$. When faulty replicas deviate from the protocol, the correct replica will trigger a new view-change. Eventually, the protocol will reach case (a), and a stable view will be reached.

If the view is not stable, and less than $f + 1$ correct replicas request a view change, a stable view can also be achieved. In this case, requests will complete if all active replicas follow the protocol. Otherwise, requests will not complete within a timeout, and eventually, all correct replicas will request view-change and the system goes to the case (a) above.

In all the three scenarios, all replicas will eventually go into a stable view and the clients' request can complete.

7 CONCLUSION AND FUTURE WORK

In this paper, we presented *SMChain*, a new blockchain design to provide immutable and trustworthy metering services among large-scale distributed industrial plants. SMChain adopts a two-layer blockchain structure, consisting of independent local chains and one state chain. The local chain is built in each plant among the edge gateways, while the state chain is formed in the cloud by aggregating individual local chains. To form a local chain efficiently, we designed a scalable BFT protocol, leveraging threshold signature, to reduce the communication complexity from $O(n^2)$ to $O(n)$ among each plant. A security analysis on safety and liveness of the proposed scalable BFT protocol is also presented.

As a future work, we plan to implement and evaluate the proposed SMChain scheme on a real-world crude oil pipeline transmission system to provide immutable and trustworthy ledger. More specially, we will design the detailed transaction and block structures for a large-scale pipeline system, and then thoroughly evaluate the performance of the proposed scalable BFT protocol in terms of the number of consensus nodes and the transaction latency.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Available: https://bitcoin.org/bitcoin.pdf*, 2008.

[2] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[3] Y. Desmedt, "Threshold cryptosystems," in *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1992, pp. 1–14.

[4] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, vol. 310, 2016.

[5] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *ACM CCS*, 2016, pp. 31–42.

[6] A. N. Bessani, M. Santos, J. Felix, N. F. Neves, and M. Correia, "On the efficiency of durable state machine replication." in *USENIX ATC*, 2013, pp. 169–180.

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

[8] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[9] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking, "Randomized rumor spreading," in *FOCS '00*. IEEE, 2000, pp. 565–574.

[10] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of cryptology*, vol. 17, no. 4, pp. 297–319, 2004.