Cypherium Whitepaper 2.0

Introduction

Since the inception of Bitcoin, the first public blockchain, researchers, engineers, and entrepreneurs throughout the world have wanted to expand this technology to a broader range of applications. Ethereum, the world's first blockchain with Turing-complete smart contracts, made a bold attempt to transform blockchain into a general-purpose computing platform.

Although the early pioneers first mapped out blockchain's potential as a digital payment network and a collaborative data sharing platform, poised to reshape human society entirely, the stark reality of the last ten years has shown us that these technologies were simply not ready to achieve these lofty goals. Blockchain has yet to see"killer dApps" other than the issuance of digital currencies. Even Bitcoin has not attained its original goal of "a peer-to-peer cash payment system".

As regulatory bodies around the world began to impose stricter regulation over ICOs, with some completely banning them, blockchain and crypto fell into a long period of hibernation.

However, blockchain technology's revolution has never stopped. Two of the most prominent digital currency projects were announced in 2019: Facebook's Libra and People's Bank of China's DC/EP. Although these projects appear to have abundant resources and a huge potential user base, they are not the best use cases of blockchain technology because they revert back to the use of trusted third parties to expedite their regulatory compliance. However, true decentralization should not run into conflict with any regulator. Just as the internet is permissionless, blockchains can be both fully realized in their decentralization and governed with meaningful oversight.

We, the Cypherium team, hereby set forth five goals for our project:

1. An instant ledger to process real-time transactions for billions of users.
2. A smart contract platform to enable enterprise use cases for all industries.
3. A trusted database to connect isolated data islands around the world.
4. An open network to enfranchise any participant or contributor.
5. A secure vault to combat the increasing threats to data privacy.

To achieve our goals, we have built Cypherium only on a foundation of the most cutting-edge and bulletproof technologies to date.

Technical Background

A blockchain is a data ledger shared across a network of nodes, which store and replicate a consistent state of transactions. According to the admissions mechanism of the network, a blockchain can be

classified into two categories: permissionless (open to any participant) and permissioned (open only to credentialed members). Typically, a blockchain consists of three layers: network, consensus, and smart contract.

A network is a collection of interconnected nodes that exchange information. The nodes can include computers, servers, mobile devices, or any other device that can be configured to communicate with other nodes in the network. The network may be configured as a centralized system, such as a client-server system having one or more central servers in communication with one or more client devices, or may be configured as a decentralized system, such as a peer-to-peer (P2P) network. A P2P network is a system in which each network node, sometimes called a "peer," may be directly connected to one or more other nodes in the network. Peers typically have similar privileges and may share a portion of their resources with other peers in the network without coordination by any centralized servers or hosts. P2P networks have been implemented using various technologies, including for example file-sharing systems, such as Napster, and blockchain technologies, such as Bitcoin.

A transaction-based system is a system that processes data in units of "transactions," which may have different contents, formats, or other characteristics depending on the system. An example of a conventional transaction-based system is Sabre, which is a system used by travel agents and companies to process transactions that are used to book airline tickets. A transaction may be a message, event, or process initiated or invoked by a user or computer program and may provide a single unit of work in the system. For example, a transaction may contain data corresponding to an electronic funds transfer between different bank accounts. A transaction could also correspond to an event when two or more parties enter into an agreement, such as a legal contract. In this example, the action of entering into the agreement is processed and recorded as a single transaction, and the transaction data may correspond to the agreement itself and/or any other information relating to the agreement. In another example, each transaction may correspond to the identity of the party or parties that have signed an agreement, in which case the signing of the agreement by each party is processed and recorded as one transaction. A transaction, alone or combined with other transactions, may be represented or transmitted as a unit of data. In a Bitcoin setting, a transaction usually is defined as the transfer of a certain amount of Bitcoin cryptocurrency from one Bitcoin account to another.

To facilitate a demand for higher speed processing and convenience in transaction-based systems, some transaction-based systems have been built on network infrastructures. Originally and conventionally, such systems have been implemented in centralized networks: all the transactions were processed by centralized servers and the related transaction data was stored in centralized databases. The system reliability thus depended solely on the reliability of the centralized system. A failure of the centralized server could cause catastrophic results to the transaction-based system.

A decentralized transaction-based system, sometimes referred to as a distributed transaction-based system, is desirable because the system depends less on centralized servers and, thus, may be more reliable. Implementing such a distributed system on a P2P network is often preferred because the necessity of using a centralized host server is eliminated and the system is reliable until many network nodes fail at the same time, which is typically unlikely. However, implementing such a distributed transaction-based system is challenging because the lack of a centralized host may result in complicated and dynamic component interdependencies. Several critical issues must be solved: for example, how is transaction data organized and stored in the system; how is a consensus reached by the network nodes

to confirm or reject a new transaction request; and how are network nodes in the system authenticated and/or their authentication verified.

Unlike in a centralized system, data in a distributed P2P system may be stored as many copies, for example, by one or more of the network nodes. For example, each node may have a copy of the entire data set in a transaction-based system or it may store only a portion of the data. Many schemes have been designed to ensure that each network node may effectively store and verify the system data through cooperation with the other nodes.

One exemplary scheme uses blockchain technology, which is widely known as the technology behind the popular cryptocurrency Bitcoin. In a blockchain scheme, transaction data can be organized into discrete "blocks," each of which may be linked with a predecessor ("parent") block by storing a cryptographic hash value of the parent block, or by using other techniques such as using hash values and/or digital signatures of one or more preceding blocks. The contents in each block can be verified against its cryptographic hash value stored in a subsequent adjacent ("child") block. Therefore, once the accuracy of the current block of the blockchain is verified, a network node can verify every preceding block contained in the blockchain without having to contact any of the other nodes.

To ensure that data is valid and consistently replicated over all nodes in the network, a blockchain requires that a consensus mechanism determine the validity of the blockchain data. Bitcoin introduced the Nakamoto consensus, named after its pseudonymic author. In the Bitcoin network, nodes recognize the version of the chain with the greatest number of valid blocks as the "real" blockchain.

Consensus

Problem

The Bitcoin system is one of the most well-known implementations of blockchain technologies in distributed transaction-based systems. In Bitcoin, each network node competes for the privilege of storing a set of one or more transactions in a new block of the blockchain by solving a complex computational math problem, sometimes referred to as a mining proof-of-work (POW). Under current conditions, a set of transactions is typically stored in a new block of the Bitcoin blockchain at a rate of about one new block every 10 minutes, and each block has an approximate size of one megabyte (MB). Accordingly, the Bitcoin system is subject to an impending scalability problem: only 3 to 7 transactions can be processed per second, which is far below the number of transactions processed in other transaction-based systems, such as the approximately 30,000 transactions per second in the Visa™ transaction system.

The most significant drawback of the Nakamoto consensus is its lack of finality. Finality means once a transaction or an action is performed on the blockchain, it is permanently recorded on the blockchain and impossible to reverse. This is vital to the safety of financial settlement systems as transactions must not be reserved once they are made. In Bitcoin's case, malicious actors can tamper with the transaction history given enough hash power, causing a double-spending attack, provided that there is enough incentive and financial viability to carry out such attacks. Given that mining equipment renting and botnets are currently prevalent world-wide, such an attack has become feasible.

Due to this lack of finality, Nakamoto consensus must rely on extra measures, such as proof-of-work to prevent malicious activities. This impedes the ability ofNakamoto consensus to scale because a transaction must wait for multiple confirmations before reaching "probabilistic finality". Therefore,

safety is not guaranteed by Nakamoto consensus, and in order to protect the network, each transaction must undergo additional time to process. In Bitcoin's case, a transaction is not considered final until at least six confirmations. Since Bitcoin can only process a few transactions per second, the transaction cost is outrageously high, making it impractical for small payments like grocery shopping or restaurant dining. This greatly hinders Bitcoin's use as a payment method in the real world.

Attempted Solutions

Many distributed ledger projects have attempted to solve the challenges Bitcoin faces. However, these solutions have yet to resolve the so-called blockchain trilemma: to preserve speed, security, and decentralization at the same time. Often, these solutions must come down to a trade-off among these three vectors.

Direct Methods

The easiest way to solve the transaction speed problem is to reduce the time taken to generate a block. Ethereum generates a new block every 12 seconds. However, as Ethereum is also based on the Nakamoto consensus, it requires more confirmations, usually over 50, for transactions to gain probabilistic finality. Bitcoin Cash introduced bigger block sizes to include more transactions in a single block, but this does not reduce the confirmation time of blocks and can cause network connection problems.

DPoS

EOS's delegated Proof-of-Stake (DPoS) consensus involves a three-tiered process. The first tier has EOS token holders hold a referendum to choose 21 Block Producers (BPs). Those elected Block Producers are then delegated to perform the second tier of the consensus by generating blocks. Lastly, the third tier consists of the 21 BPs conducting Byzantine Fault Tolerant consensus, wherein approved blocks can be irreversibly admitted to the blockchain history. The block may be considered truly confirmed having completed this process.

EOS boasts that it can produce a new block every half-second. However, the true confirmation of these blocks cannot be considered final until their BFT consensus has been performed, and that process takes roughly three minutes. To propose and proliferate new, unconfirmed blocks has little bearing on the health of the blockchain itself.

Bitcoin-NG

Bitcoin-NG, or the Bitcoin Next Generation system, attempts to solve this scalability problem by introducing a network node in the P2P system that acts as a leader node that processes transactions and adds new blocks to the Bitcoin blockchain. This system includes two separate blockchains: a Transaction blockchain for storing the transaction data and a Keyblock blockchain for publishing a public key, which may be a temporary "epoch" public key assigned to the current leader node. In Bitcoin-NG, the Keyblock blocks are generated in the same way as transaction blocks are generated in the conventional Bitcoin network, except the Keyblock blocks contain only the public key that the leader node will use to sign new blocks added to the Transaction blockchain. Thus, the leader node is the node that generated the latest block in the Keyblock blockchain and that digitally signs new blocks added to the Transaction blockchain using a private key corresponding to its published epoch public key.

While Bitcoin-NG does solve conventional Bitcoin's scalability problem, as the leader node in Bitcoin-NG can process new transaction blocks more efficiently than using the POW approach of conventional Bitcoin, it somewhat reverts Bitcoin from a distributed system to a system by having a single leader node at any given time. As such, the reliability and the security of the Bitcoin-NG system depends, at least partially, on the reliability and security of that leader node. The failure of the leader node may cause a catastrophic failure of the system. The Bitcoin-NG system is described, for example, in the paper by Ittay Eyal et al., "Bitcoin-NG: A Scalable Blockchain Protocol," 13th USENIX Symposium on Networked System Design and Implementation (NSDI '16), pp. 45-59 (2016), which is incorporated by reference herein.

PBFT

Another issue rooted in a conventional distributed system is the Byzantine fault problem, which addresses how a system of multiple nodes can reach a consensus for agreeing when to change the system's state. The Byzantine fault problem requires that nodes must agree on a concrete strategy to reach a consensus to avoid a system failure, even where some of the nodes may be unreliable. Castro et al., "Practical Byzantine Fault Tolerance," Proceedings of the Third Symposium on Operating System Design and Implementation, New Orleans, USA (Feb. 1999), which is incorporated by reference herein, introduced a two-round peer-to-peer message exchange technique to achieve a consensus. Initially, a leader node sends a system state-transition request to each of the other nodes in the network. In the first round, each node communicates with each of the other nodes to confirm that enough nodes have received the state-transition request, and then the nodes validate the request after confirming a first threshold number of nodes received it. In the second round, each node communicates with each of the other nodes to confirm that a second threshold number of nodes were able to validate the request. If the second threshold is satisfied, the nodes finalize the requested system state-transition based on the second round of confirmation. Because each node communicates with each of the other nodes in the system to reach a consensus, such a system is referred to as a mesh communication network. Because only a threshold number of nodes are required to reach a consensus, a fault tolerance that permits a certain number of faulty nodes can be achieved in the Practical Byzantine Fault Tolerance (PBFT) system. However, PBFT's solution is not efficient when the system is scaled up. The communication costs necessary to form a consensus in the mesh network increases exponentially as the number of nodes increases in the PBFT system, again resulting in a scalability problem. Further, the identities of the nodes are known to other nodes in the network, exposing all the nodes to potential attacks from other nodes in or out of the network. Given the nodes in PBFT are not reconfigurable, the failure of sufficient number of nodes may also cause catastrophic failure of the system.

In the PBFT system, a "view" refers to a period of time that a node serves as the leader node. When a node in the PBFT system discovers that the leader node has failed, it broadcasts a request to each of the other nodes in the system to change the leader node, which is a "view change" request. When a node receives enough view-change requests, it first determines a new leader node based on predefined rules and sends a view-change confirmation to the new leader node. The next "view" starts when the new leader node receives enough view-change confirmations from the other nodes in the PBFT system.

Tendermint/Casper

Tendermint and Casper, two blockchains that rose to prominence in 2018, brought to public view two new variants of PBFT that simplify a traditional aspect of the consensus mechanism. These projects retrofitted PBFT's leader-replacement protocol with linearity, meaning only linear communication cost would be levied on network participants. However, in order to add this economic aspect to their

networks, the two projects sacrificed a cornerstone of practical consensus called responsiveness. In brief and plain terms, responsiveness refers to a dynamic wherein appointed leaders of a PBFT consensus may generate after receiving a fixed number of communications, instead of waiting for a mandatory fixed delay. Tendermint and Casper, in this way, brought to the PBFT space a trade-off between linearity and responsiveness. The HotStuff algorithm, which has been adopted by Facebook's Libra, as well as some other recent blockchains, have resolved this trade-off.

HotStuff/LibraBFT

As a recent development, "HotStuff" introduced a protocol that modifies the mesh communication network in the PBFT system, merging the view-change process for changing a leader node with generation of transaction blocks, resulting in a three-phase protocol. HotStuff is described, for example, by Maofan Yin et al., "HotStuff: BFT Consensus in the Lens of Blockchain," document no. arXiv:1803.05069 (March 2018), which is incorporated by reference herein. Because each node communicates all of its messages (including transaction requests) directly to the leader node instead of indirectly sending them to the leader node via one or more other nodes, the communication complexity of the HotStuff system is greatly reduced. However, even with such reduced communications, the HotStuff network still faces scalability issues. For example, the number of nodes is limited to the capability of the leader node to process transaction requests from a large number of nodes. In addition, as the number of nodes grows, the system slows because of the time and resources consumed while processing and waiting for digital signatures for messages requiring multiple cosigners. Further, because the identity of the leader node is public to all of the nodes in the network, the leader node may be subject to attacks by malicious third parties, which also may decrease the reliability of the system.

Libra's consensus algorithm, LibraBFT, is an implementation of the HotStuff algorithm that completes HotStuff's "pacemaker" process. The LibraBFT is permissioned and nodes require authorization of the Libra Association to become a committee member. Although Libra and some other HotStuff based projects state that they will become permissionless in the future, their technical documents do not specify how they will implement the validator committee reconfiguration mechanism to allow open participation. Moreover, Libra is backed by the Libra Reserve, which is pegged to a number of fiat currencies. The centralized nature of Libra has diminished its value as an implementation of blockchain technology.

DAG

There are a number of new distributed ledger technologies that do not use traditional blockchain data structures, favoring instead Direct Acyclic Graphs (DAGs), which allow blocks to be authored and accepted simultaneously, instead of a single, total-ordered chain.

Hashgraph and Conflux might be two of the most notable examples of such projects. However, these graphs are not without their weaknesses. One particularly notable shortcoming of DAGs is the recovery of data. If a network participant loses connection to the network for whatever reasons, it is much harder to recover the graph information than it is to retrace the history of a true blockchain. The lack of total-ordering also presents a significant barrier for DAG to support smart contracts, limiting its utility.

CypherBFT

Cypherium's proprietary consensus algorithm, CypherBFT overcomes disadvantages of the prior art by providing a distributed transaction system including a group of validator nodes that are known to each other in a network but are indistinguishable to the other network nodes in the network. As used herein,

the group of validator nodes may be referred to as a “Committee” of validator nodes. In some embodiments, the system reconfigures one or more validator nodes in the Committee based on the results of proof-of-work (POW) challenges. According to some disclosed embodiments, a network node that is not already a validator node in the Committee may be added to the Committee if it successfully completes a POW challenge. In such an event, the network node may become a new validator node in the Committee, replacing an existing validator node. In alternative embodiments, a network node may become a new validator node in the Committee based on a proof-of-stake (POS) consensus. In yet another embodiment, a network node may become a new validator node in the Committee based on a proof-of-authority (POA) consensus. In other alternative embodiments, a network node may become a new validator node in the Committee based on a combination of any of POW, POA, and POS consensus.

In some disclosed embodiments, the new validator node replaces a validator node in the Committee. The replacement may be based on a predetermined rule known by all the nodes in the network. For example, the new validator node may replace the oldest validator node in the Committee. According to another example, the new validator node may replace a validator node that has been determined to have gone off-line, become compromised (e.g., hacked), failed (e.g., due to hardware malfunction), or otherwise is unavailable or no longer trusted. In the exemplary embodiments, the distributed system assumes that for a fault-tolerance of f nodes, the Committee includes at least $3f+1$ validator nodes. Because the validator nodes in the Committee may be frequently replaced, for example, depending on the amount of time required to complete the POW challenges, it is difficult for malicious third parties to detect the complete set of validator nodes in the Committee at any given time.

Cypherium, on the other hand, runs its proprietary CypherBFT consensus, anchored by the HotStuff algorithm, and can authentically offer instant finality for its network users. With its HotStuff-based design, the CypherBFT’s runtime lasts only 20-30 milliseconds (ms). Two-to-three confirmations are all that is required to permanently accept a proposed block into the blockchain, and it only takes 90ms for these confirmations to transpire, making the process significantly faster than the two-minutes required by EOS. Cypherium’s CypherBFT, which also utilizes HotStuff, does not need to choose between responsiveness and linearity.

Cypherium’s dual blockchain structure includes the speeds of a dag, but its recall for users can happen much simpler and faster, which adds to the availability of information and makes the information more decentralized.

In accordance with some disclosed embodiments, the validator nodes in the Committee may receive transaction requests from other network nodes, for example, in a P2P network. The Committee may include at least one validator node that serves as a “Leader” validator node; the other validator nodes may be referred to as “Associate” validator nodes. The Leader node may be changed periodically, on demand, or sporadically by the members of the Committee. When any validator node receives a new transaction request from a non-validator node in the network, the transaction request may be forwarded to all of the validator nodes in the Committee. Further to the disclosed embodiments, the Leader node coordinates with the other Associate validator nodes to reach a consensus of a disposition (e.g., accept or reject) for a transaction block containing the transaction request and broadcasts the consensus to the entire P2P network. If the consensus is to accept or otherwise validate the transaction request, the requested transaction may be added in a new block of a blockchain that is known to at least some of the network nodes in the network.

In accordance with some embodiments, the Leader node may use improved protocols for reaching a consensus with the Associate validator nodes in the Committee when determining the disposition of a received request, such as a transaction request to add one or more transactions to an existing blockchain or a candidate request to request that a node join the Committee as a new validator node. For example, in some embodiments, the Leader node may use aggregate signatures in reaching a consensus for a new transaction request, allowing the consensus to be reached by validating digital signatures from fewer than all of the validator nodes in the Committee. In contrast with prior systems, the Leader node's use of aggregate signatures may permit the Committee to reach a consensus of whether to accept or reject a preliminary transaction block faster and/or more efficiently, for example, in the event that one or more of the validator nodes in the Committee do not respond sufficiently quickly or at all. In some embodiments, as long as the number of responding validator nodes reaches a predetermined threshold, the Leader node can conclude that a consensus in the Committee has been reached. In some exemplary embodiments, in a network having a fault-tolerance of f nodes, and the Committee includes $3f+1$ validator nodes, this predetermined threshold may be set equal to $2f+1$ validator nodes.

In accordance with some disclosed embodiments, the improved protocols used by the Leader node for reaching a consensus based on receipt of a new transaction request or candidate request may comprise a plurality of phases. In some embodiments, the Leader node employs a three-phase protocol, for example including a Prepare phase, a Pre-Commit phase, and a Commit phase, for determining whether to accept or reject a new transaction request or candidate request. In other exemplary embodiments, the Leader node may determine the disposition of the new request using a two-phase protocol, for example including a Prepare phase and a Commit phase.

Advantageously, the disclosed embodiments provide a distributed-system architecture and related protocols that allow a distributed system, such as a blockchain system, to scale up without incurring an unacceptable increase in decision-making complexity while also preserving the benefit of using a decentralized system. The disclosed embodiments reduce the distributed system's reliance on the stability of any particular node(s), as the validator nodes in the Committee may be changed at a sufficient frequency to remove unreliable, unavailable, or otherwise untrusted nodes. Further, the system and method of the disclosed embodiments provides a scheme that helps ensure the Leader node, as well as the other Committee members, functions properly.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the disclosed embodiments or the scope of the inventions as claimed. The concepts in this application may be employed in other embodiments without departing from the scope of the inventions.

The CypherBFT process flow is described as follow:

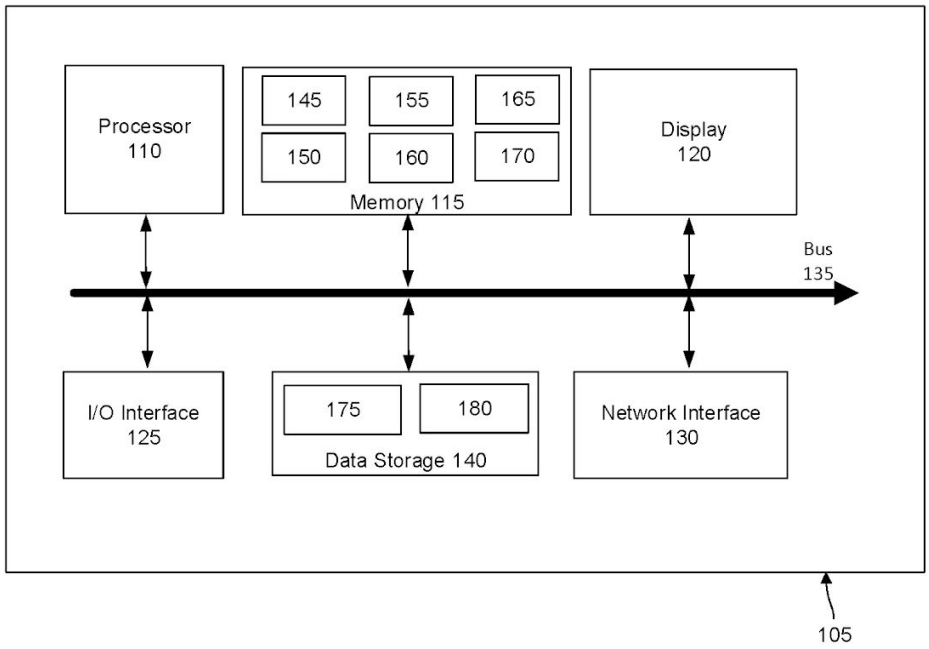


FIG. 1

Figure 1 is a schematic of an exemplary computing system 105 that may be used as a network node in a network 200 in accordance with the systems and methods of the disclosed embodiments. The computing system may be configured to function, for example, as any of a Client node, Common node, Associate node, or Leader node in the disclosed embodiments. Computing system 105 includes, for example, one or more processors 110, a memory 115, a display 120, input/output (I/O) interface(s) 125, network interface(s) 130, and data storage 140. These components in the computing system may communicate with each other via a system bus 135, wirelessly, or via one or more wired or wireless communication paths. The components in Figure 1 may reside in a single device or in multiple devices.

Consistent with the disclosed embodiments, a processor 110 may comprise at least one of a central processing unit (CPU), graphical processing unit (GPU), or similar microprocessor having one or more processing cores. Computing system 105 may include one or more processors 110 and may further operate with one or more other processors that may be located in one or more remote systems. Memory 115 may include non-volatile and/or volatile memory configured to store software instructions for execution on the processor 110, and may include one or more of random access memory (RAM), solid state memory, magnetic memory, register memory, cache memory, or any other type of memory.

In some embodiments, memory 115 may comprise one or more removable modules or may at least partially reside in a remote storage device (not shown) outside of system 105. In some embodiments, the memory 115 may be configured to store data and instructions, such as software programs. For example, memory 115 may be configured to store data and instructions. In some embodiments, processor 110 may be configured to execute instructions and/or programs stored in memory 115 to configure computing system 105 to perform operations of the disclosed systems and methods. In various aspects, as would be recognized by one of skill in the art, processor 110 may be configured to execute at least some instructions and/or software programs stored on a remote memory to perform operations of the disclosed systems and methods.

In accordance with certain disclosed embodiments, the memory 115 may include one or more of a campaign engine 145, candidate-selection engine 150, vote-decision engine 155, coordination engine 160, signature engine 165, and view-change engine 170, which may be embodied in one or more software programs, modules, libraries, and/or applications stored in memory 115. The software also may be stored in one or more other data storage devices 140, such as a hard drive or other non-volatile memory for longer term storage.

Campaign engine 145 may generate candidate requests during a first stage of a reconfiguration process, for example, as described in detail with reference to Figure 7. Candidate-selection engine 150 may verify candidate requests (such as by verifying digital signatures in those requests) and select one or more "best" candidate nodes, for example, as described with reference to Figure 8. Vote-decision engine 155 may verify information transmitted from a Leader node (such as by verifying a digital signature associated with the information) and generate a voting decision based on the verification, for example, as described with reference to Figure 10. Coordination engine 160 may determine the content of a block of a blockchain and/or generate new blocks for a blockchain. Signature engine 165 may be configured to generate digital signatures using a private key associated with the computing system 105, such as generating partial signatures and aggregated signatures. The signature engine also may be configured to verify received digital signatures using the public keys of other nodes, such as verifying received partial signatures.

View-change engine 170 may detect the need for a view change in a transaction-based system, select a new Leader node, and/or vote for a new Leader node. Depending on the type of a node, certain engines may be loaded and activated, and other engines may be deactivated or deleted from the memory. For example, when a node is configured as a Common node, campaign engine 145 may be loaded and activated and one or more of the other engines 150, 155, 160, 165, and 170 may be deactivated and/or removed from the memory 115. In some embodiments, when a node is configured as an Associate node, it may load and activate the candidate-selection engine 150, vote-decision engine 155, and view-change engine 170. As another example, a Leader node may load and activate the candidate-selection engine 150, coordination engine 160, and signature engine 165. As explained herein, a node may be configured as more than one type of node, in which case the engines corresponding to the configured types may be loaded to the memory 115 and be activated.

Display 120 may be any device adapted to provide a visual output, for example, a computer monitor, an LCD screen, etc. The I/O interfaces 125 may include hardware and/or a combination of hardware and software for communicating information between the computing system 105 and a user of the computing system. The I/O interfaces may include, for example, devices such as a keyboard, mouse, trackball, audio input device, touch screen, infrared input interface, or any other input or output device.

Display 120 is an optional component. Some nodes may include display 120 while other nodes may omit this device.

Network interface 130 may be a network adapter that includes hardware and/or a combination of hardware and software for enabling the computing system 105 to exchange information with other network nodes in the network 200. For example, the network interface 130 may include a wireless wide area network (WWAN) adapter, a Bluetooth module, a near field communication module, an Ethernet interface, a local area network (LAN) adapter, or any other network interface known in the art for communicating over a network. The network adapter 130 may transmit and receive information from another node in a P2P network.

Data Storage 140 may include one or more data storage devices, such as a computer hard disk, random access memory (RAM), removable storage, or remote computer storage, or an array of such memory devices. In some embodiments, the data storage 140 and memory 115 may share at least one memory device. In other embodiments, the data storage 140 and memory 115 may use separate memory devices. The data storage 140 may include Transaction storage 175, which may be configured to store transactions and/or transaction data, such as a TransactionBlock blockchain in a blockchain implementation. The data storage 140 may also include Key storage 180, which may be configured to store cryptographic key information and/or a KeyBlock blockchain in a blockchain implementation. While Transaction and Key information 175 and 180 are preferably stored in the data storage 140, they also may be stored in the memory 115, for example, as a duplicated copy for the purpose of access or processing efficiency. It should be understood that although the blockchain data structures are explained in the exemplary embodiments, any other data structure alternatively may be adopted, such as doubly-linked lists, heaps, trees, and so forth.

Network Architecture

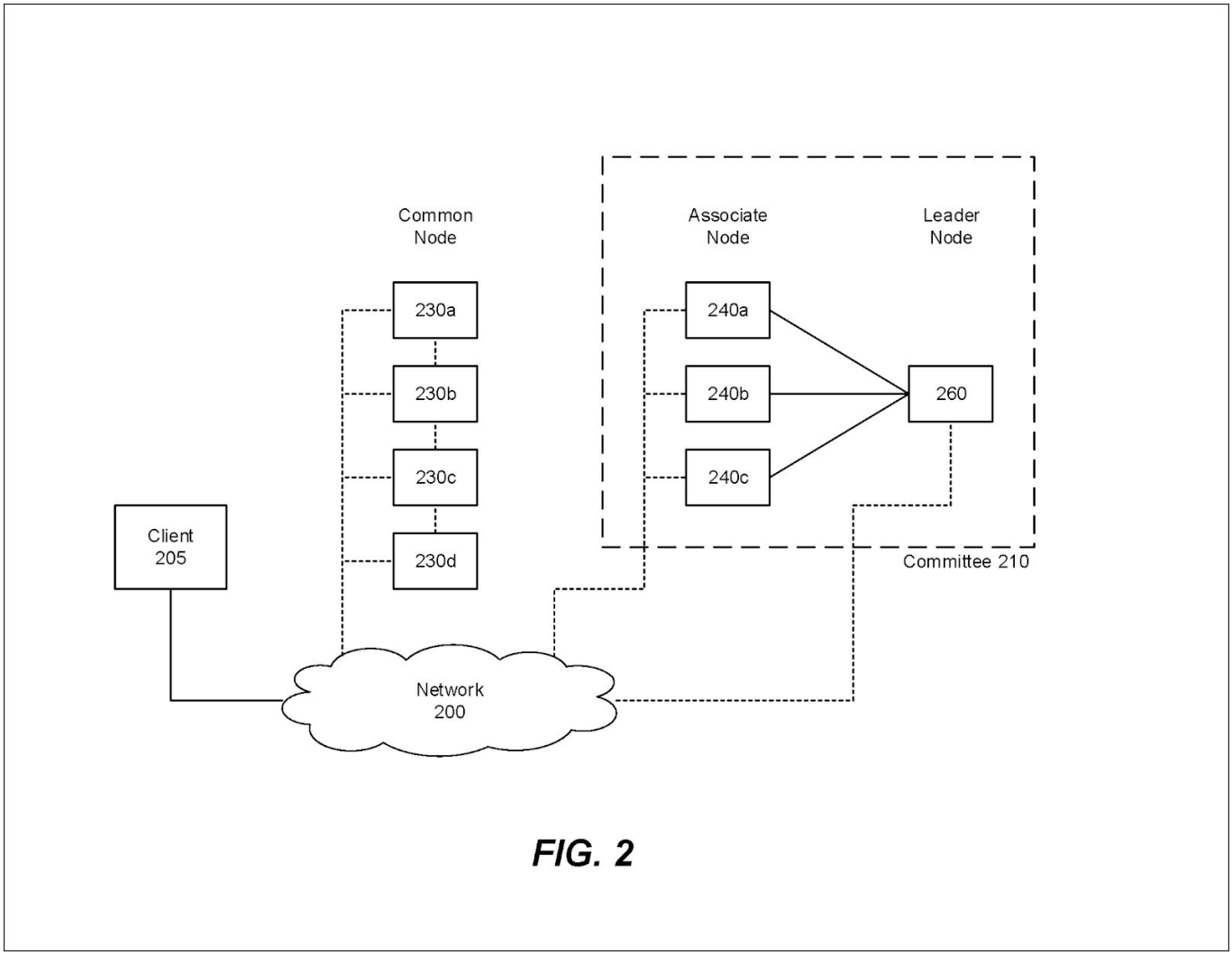


FIG. 2

Figure 2 depicts a schematic block diagram of an exemplary network architecture that may be used in accordance with the disclosed embodiments of the invention. In Figure 2, a network 200 contains a plurality of network nodes, each of which may be directly connected to at least one of the other nodes in the network. Accordingly, at least a portion of the network 200 may be configured as a P2P network, and in the disclosed embodiments, the network 200 also may be referred to as the P2P network. The network 200 includes one or more of each of the following types of nodes: Client node 205, Common node 230, Associate node 240, and Leader node 260. Each of the Associate nodes 240 and Leader node 260 are validator nodes that collectively form a Committee 210 of validator nodes. In some cases, the same physical device or devices may be configured to serve as different types of network nodes simultaneously. For example, a computer may be configured to function as a Client node 205 for certain users and also may otherwise function as a Common node in the network 200.

Client node 205 is a network node from which a user may submit a new transaction request. For example, the Client node 205 may provide a transaction request including data for a new transaction block that may be added to a blockchain. In the disclosed embodiments, for example, the user may be

any person, software application, or other entity that can generate or otherwise provide a transaction request at the Client node 205.

Each Common node 230 may serve as a bridge, relay, or router configured to forward one or more transaction requests throughout the network 200. Figure 2 illustrates exemplary Common nodes 230a-d, although there may be one or more Common nodes in the network. A Common node 230 may receive a transaction request from the Client node 205 or another Common node 230 in the network 200. In operation, a transaction request may be originated at a Client node 205, routed through the network 200 via one or more Client and/or Common nodes, until it is received by each of the validator nodes in the Committee 210. In accordance with the disclosed embodiments, a Common node 230 may be a candidate to become a new validator node in the Committee 210. Conversely, an Associate or Leader node may be removed from the Committee and, in some cases, may become a Common node.

In the Committee 210, the Leader node 260 coordinates with the other Associate nodes 240 to reach a consensus of a disposition (e.g., accept or reject) for each new transaction block and in some disclosed embodiments, broadcasts the consensus decision to one or more network nodes in the network 200. For example, in certain embodiments where Common nodes 230 maintain a copy of a current blockchain, the consensus decision from the Committee whether to add a new transaction block based on a received transaction request may be communicated to at least those Common nodes that store copies of the relevant blockchain. While Figure 2 illustrates a set of Associate nodes 240a-c, those skilled in the art will appreciate there may be one or more Associate nodes in the Committee; there is preferably a single Leader node 260, although it is also contemplated that other embodiments (not shown) could employ more than one Leader node. In some embodiments, each Associate node 240 is configured to communicate directly with the Leader node 260. Further, as part of certain network events or procedures, an Associate node also may communicate directly with one or more other Associate nodes, such as in response to a “view change” event where the validator-node membership of the Committee is modified to select a new Leader node 260.

Further to the disclosed embodiments, the term of validator nodes in the Committee 210 may be limited. For example, in some embodiments, the Committee may have a predetermined term limit, e.g., measured in time or transactions, after which the specific validator nodes constituting the Committee may be updated or adjusted. One or more members 240 and 260 in the Committee 210 may be replaced at the end of each Committee term. In the disclosed embodiments, the term length for an Associate or Leader node may be defined as a fixed time interval, a predetermined number of transactions or events, or may be in response to one or more predetermined triggering events. In some cases, an Associate or Leader validator node may be removed from the Committee before the end of its term as a result of a discovered malfunction, period of non-responsiveness, malicious behavior, or other event or action deemed unacceptable for participation in the Committee.

Preferably, each Associate and Leader node stores information about each of other validator nodes in the Committee 210. For example, each of the Associate and Leader validator nodes may store the IP address of each of other validator nodes in the Committee. Each Associate and Leader node also may store public keys corresponding to each of the other validator nodes in the Committee. In some embodiments, the validator nodes may maintain “live” or “active” communications connections or sessions with each other and transmit signals from time to time to keep those connections or sessions active. Alternatively, the Associate and Leader validator nodes may not communicate with each other until one or more events trigger a need for such communications, such as in response to a view-change

event or any other procedure to select a new Leader node in the Committee. removed from the Committee before the end of its term as a result of a discovered malfunction, period of non-responsiveness, malicious behavior, or other event or action deemed unacceptable for participation in the Committee.

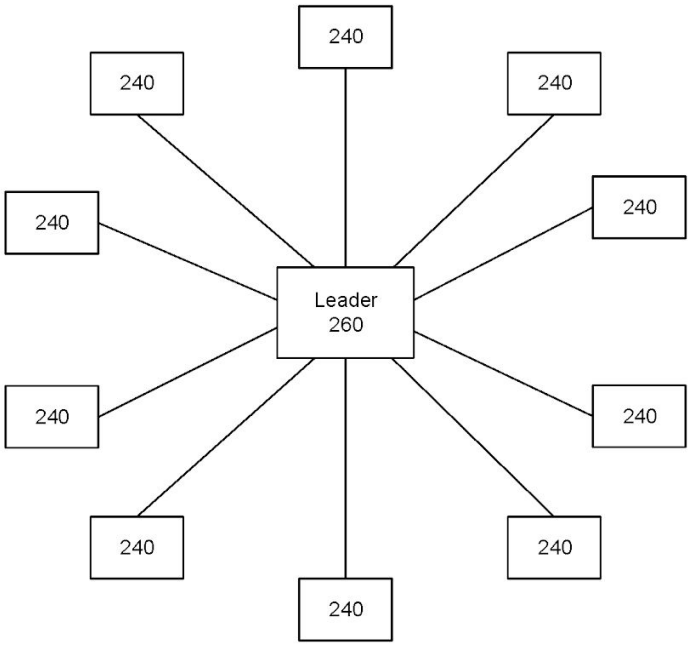


FIG. 3A

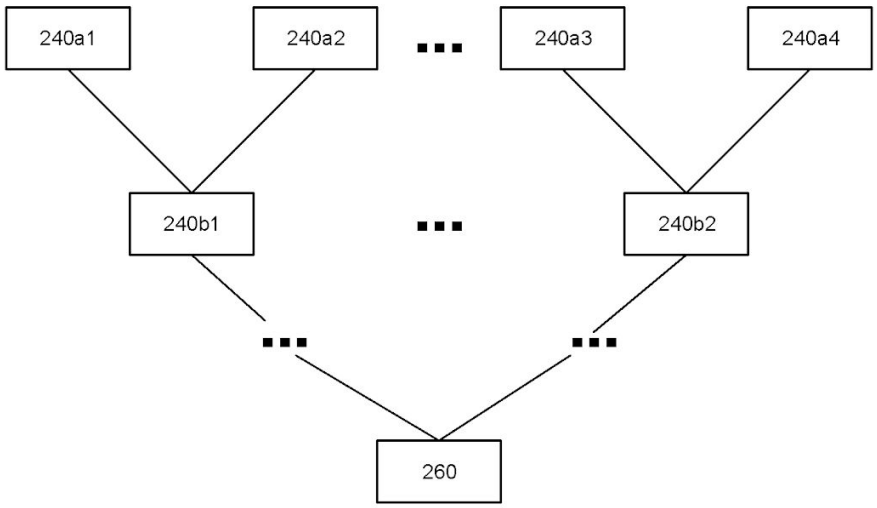


FIG. 3B

Figure 3A and 3B each depict a schematic block diagram of an exemplary network architecture of Committee 210 that may be used in accordance with the disclosed embodiments of the invention. In Figure 3A, the Committee 210 includes a Leader node 260 and a plurality of Associate nodes 240, each of which may be directly connected to the Leader node 260. In the exemplary embodiment of Figure 3A, the Associate nodes may be configured to communicate directly with the Leader node 260 until a certain event, such as a view-change event or procedure for changing the Leader node in the Committee, in which case each Associate node in the Committee may communicate with other Associate nodes in the Committee.

In the exemplary embodiment of Figure 3B, the Committee 210 contains the Leader node 260, the plurality of Associate nodes 240a1, 240a2, 240a3, 240a4, 240b1, 240b2, and one or more other Associate nodes arranged in a tree structure. Each Associate node may be logically connected in the tree structure with at least one parent and/or child Associate nodes. For example, in Figure 3B, Associate nodes 240a1 and 240a2 are connected with their parent Associate node 240b1, and Associate nodes 240a3 and 240a4 are connected with their parent Associate node 240b2. Associate nodes 240b1 and

240b2 are further connected with one or more additional levels of parent nodes, which eventually are connected to the Leader node 260.

In some embodiments, each parent node in the exemplary tree structure of Figure 3B is connected with same number of child nodes, and each Associate node other than the root (i.e., the Leader node 260) may have same number of sibling nodes; in other embodiments, the number of child nodes of each parent node may vary. The number of levels in the tree structure and the number(s) of nodes at each level also may vary depending on the implementation of the Committee. Further, the arrangement of the validator nodes in the tree structure may be ordered, for example, where each node in the tree is assigned with a sequence number. Using an ordered data structure, such as an ordered tree in Figure 3B, for the validator nodes may form a first-in, first-out queue that can serve various purposes described further in the disclosed embodiments.

In some embodiments, the Associate and Leader nodes may each store information about each of all other validator nodes in the Committee 210. In other embodiments, each validator node may store information about only some of the other validator nodes in the Committee 210. For example, an Associate or Leader validator node may be configured to only store information about its parent node, child nodes, sibling nodes, or combinations of such nodes, as defined by the logical arrangement of nodes in the exemplary tree structure shown in Figure 3B. The stored information may include one or more of the IP address of another validator node, the public key corresponding to that other validator node, and/or other information that may help identify that other validator node. In some embodiments, a validator node may maintain live connections with its parent and child nodes and transmit pulse signals or other messages from time to time, for example, to keep those connections active. Alternatively, a validator node may not communicate with any of the other validator nodes until a certain event, such as a view-change event or procedure for changing the Leader node in the Committee.

Data Structure

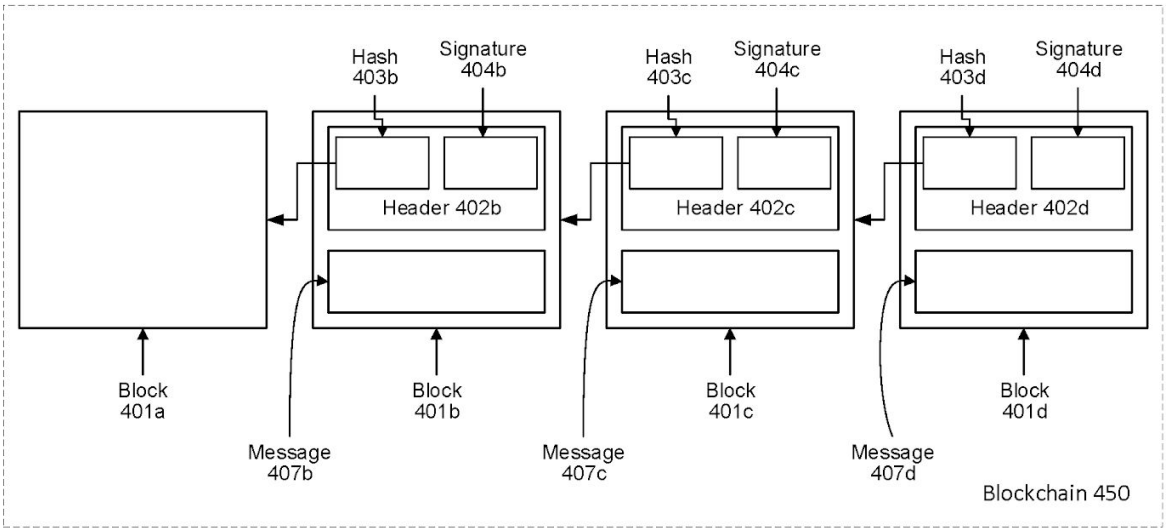


FIG. 4

Figure 4 illustrates a schematic block diagram of an exemplary blockchain 450 that may be used in accordance with the disclosed embodiments. Copies of the blockchain 450 may be maintained by many different nodes (e.g., Common nodes 230, Associate nodes 240, or Leader node 260) in the network 200. The exemplary blockchain 450 comprises one or more blocks, such as blocks 401a - 401d. A block 401 may include at least one message, such as message 407b, 407c, or 407d. In this context, the message contains any type of data, such as transaction data that may have been generated by one or more Client nodes in the network.

A block also may include a header, such as the header 402b in block 401b. The header may include, for example, at least one of: a hash value 403b of a previous block in the blockchain 450, a digital signature 404b, a hash value (not shown) based on at least one message 407b, and a timestamp (not shown), which may indicate a date and/or time when the block was added to the blockchain. In some embodiments, the header may be digitally signed with a cryptographic key of an authorized system, and the digital signature may be included in the header. Preferably, this digital signature may be verified

using a public key of the authorized system, which may be a node in the network or any other trusted authority.

In accordance with some of the disclosed embodiments, there may be at least two types of blockchains 450: a KeyBlock blockchain and a TransactionBlock blockchain. Although the two blockchains are described separately below, alternatively they could be merged into one blockchain including multiple types of blocks: KeyBlock blocks (KBs) and TransactionBlock blocks (TBs). In some embodiments, there may be a third type of blockchains 450: a CommitteeBlock blockchain.

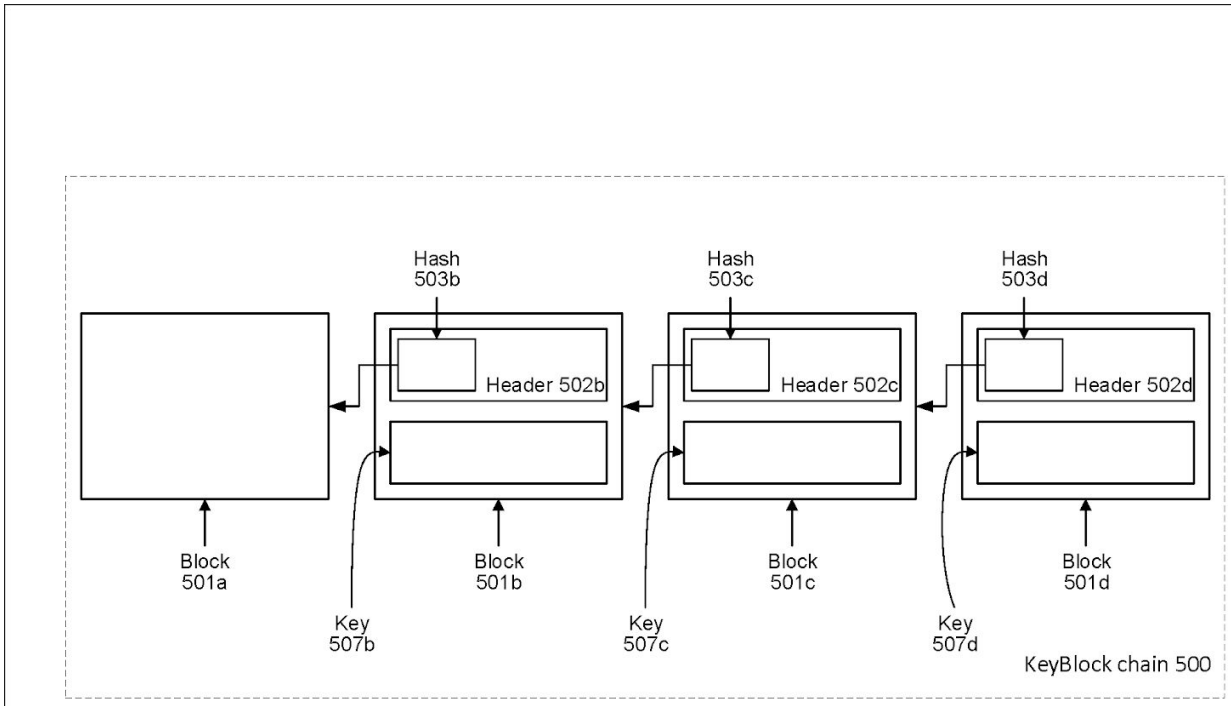


FIG. 5A

Figure 5A depicts an exemplary KeyBlock blockchain 500 that may be used in accordance with certain disclosed embodiments. In some embodiments, the KeyBlock blockchain is formed using individual blocks, such as the exemplary Keyblock blocks 501a, 501b, and 501c, which store one or more cryptographic public keys 507b, 507c, and 507d that can be used by at least some of the network nodes in network 200. Each of the KeyBlock blocks may include a header portion containing a hash value, which may be digitally signed in some embodiments, of one or more preceding blocks in the KeyBlock blockchain, such as the hash values 503b, 503c, and 503d within the KeyBlock block headers 502b, 502c, and 502d. In an exemplary embodiment, each Keyblock block (KB) 501 in the KeyBlock chain may

correspond to a different term of the Committee 210. The message data stored in each KB 501 may include one or more public keys for the validator nodes in the Committee 210. When a new Committee 210 is formed, or the validator nodes in the Committee are changed or reassigned, a new block 501 is added to the KeyBlock chain corresponding to the new composition of the Committee. Particularly, a KeyBlock block 501 preferably stores a public “epoch” key, which is a cryptographic key that is used by the validator nodes during the term of the corresponding Committee 210. In some embodiments, each block 501 in the KeyBlock chain also may store the public keys of the individual validator nodes in the block’s corresponding Committee 210.

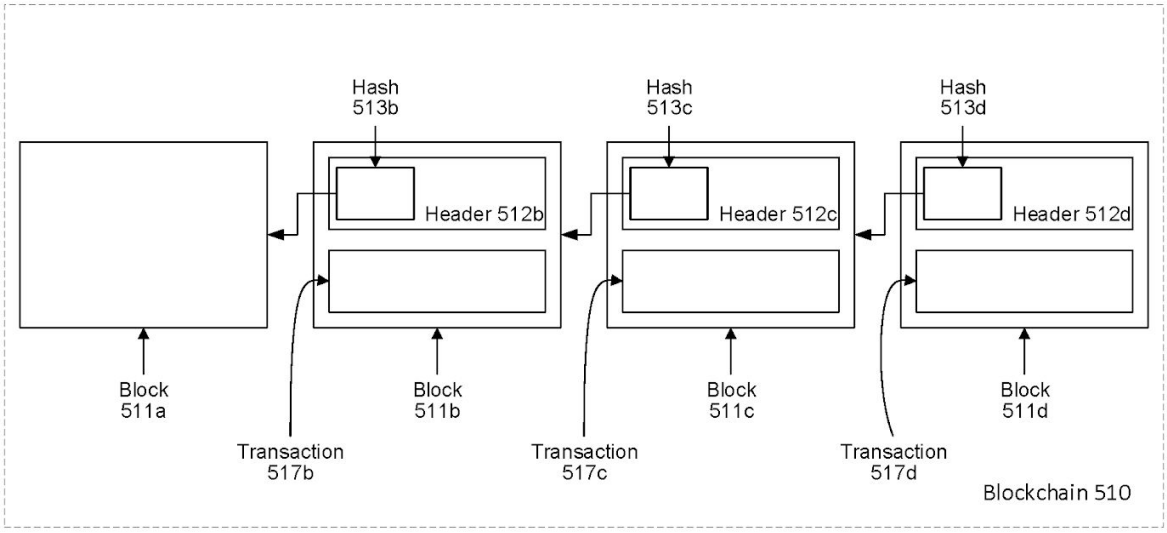


FIG. 5B

Figure 5B depicts an exemplary TransactionBlock chain 510 that may be used in accordance with the disclosed embodiments of the invention. A TransactionBlock chain is a blockchain in which individual blocks, such as exemplary TransactionBlock blocks 511a, 511b, 511c, and 511d, store messages containing any type of transaction data that may be processed by the system. A TransactionBlock block may contain transaction data, such as Transactions 517b-d, corresponding to one or more transactions in the network. In this context, the transaction data is any data that may be stored in the blockchain 510, and is not limited to any particular type of transaction data. In accordance with the disclosed embodiments, each block 511 in the TransactionBlock blockchain 510 stores information about one or

more transactions processed by the Committee. For example, a transaction may correspond to currency or electronic funds transferring, where the information may include payer account information, payee account information, amount to be transferred, and/or the type of currency. By way of another example, a transaction may be a legal agreement, such as a patent assignment agreement, where in such an example the information may include the patent number, the assignor and the assignee of the patent. It may also include any witness that has witnessed the transaction or may include the entire contract document. The transaction data in the TransactionBlock blocks (TB) 511 generally may correspond to data for one or more financial, legal, commercial, collaborative, or any other types of transactions.

Further to the disclosed embodiments, the TBs 511 are generated by the Committee 210. The Committee 210 may generate a TB in certain time intervals. For example, the Committee 210 may generate a TB every ten minutes or other predetermined time interval, and in this example, the TB may store data for all transactions that the Committee has processed during that time interval. In some embodiments, the Committee 210 may generate a new TB when it has received a certain amount of transaction requests or has processed a certain number of transactions. In some embodiments, the Committee may generate a new TB 511 in the TransactionBlock blockchain 510 when a certain amount of transactions are processed or when a certain period of time has lapsed. In yet other exemplary embodiments, a new TB is generated when the size of the transaction data to be included in a TB reaches a threshold size. For example, a TB may be generated as a result of the Committee having received a predetermined number, such as 1024, currency transactions, whereas another exemplary implementation may generate a new TB when the Committee has received a different number, such as 10, patent transactions to include in a patent assignment, since the data for each patent assignment transaction may be larger than the data required for a currency transaction. Preferably, the TransactionBlock blockchain may be modified in the same manner that the KeyBlock blockchain is modified in the steps described with reference with Figures 9-11.

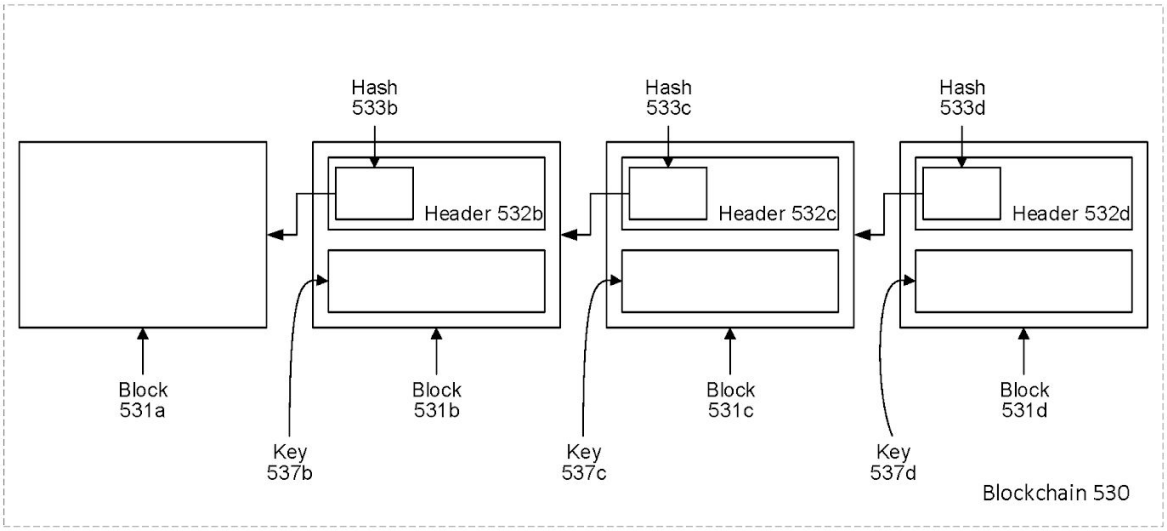


FIG. 5C

Figure 5C depicts an exemplary CommitteeBlock blockchain 530 that may be used in accordance with certain disclosed embodiments of the invention. In some embodiments, the CommitteeBlock blockchain contains individual blocks, such as exemplary CommitteeBlock blocks (CB) 531a, 531b, 531c, and 531d, that store information of members of Committee 210. For example, the CBs 531b-d may store cryptographic key information 537b-d for members of the Committee 210 and also may comprise block headers 532b-d storing hash values 533b-d of one or more preceding CBs in the blockchain.

In some exemplary embodiments, each CB 531 in the CommitteeBlock chain 530 may correspond to a different KB 501 in the KeyBlock chain 510 (Figure 5B), in which case each CB contains information for the same term of the Committee 210 as its corresponding KB. Accordingly, when a new KB 501 is added to the KeyBlock chain corresponding to a new composition of the Committee 210, a new CB 531 is also added to the CommitteeBlock chain 530. In some embodiments, the message data stored in the new KB 501 includes a hash of its corresponding new CB 531. In some embodiments, each validator node may generate CB 531 when verifying the new PKB-1. Each validator node may calculate a hash value of the CB and compare it with the hash value of CB stored in the PKB-1.

Further, in some embodiments, the CommitteeBlock chain 530 may be encrypted and/or stored separately from the other blockchains in the system. The encrypted message data of each CB 531 is only accessible to validator nodes in the Committee 210 during a term corresponding to the CB. In some other embodiments, each CB 531 may be transmitted among validator nodes without encryption.

The message data stored in each CB 531 may include information of the validator nodes in the Committee 210 during the corresponding term. In some embodiments, the message data stored in each CB 531 may include the IP addresses (or other network addresses or network-node identifications) of all the validator nodes in the Committee 210 during the term.

Reconfiguration

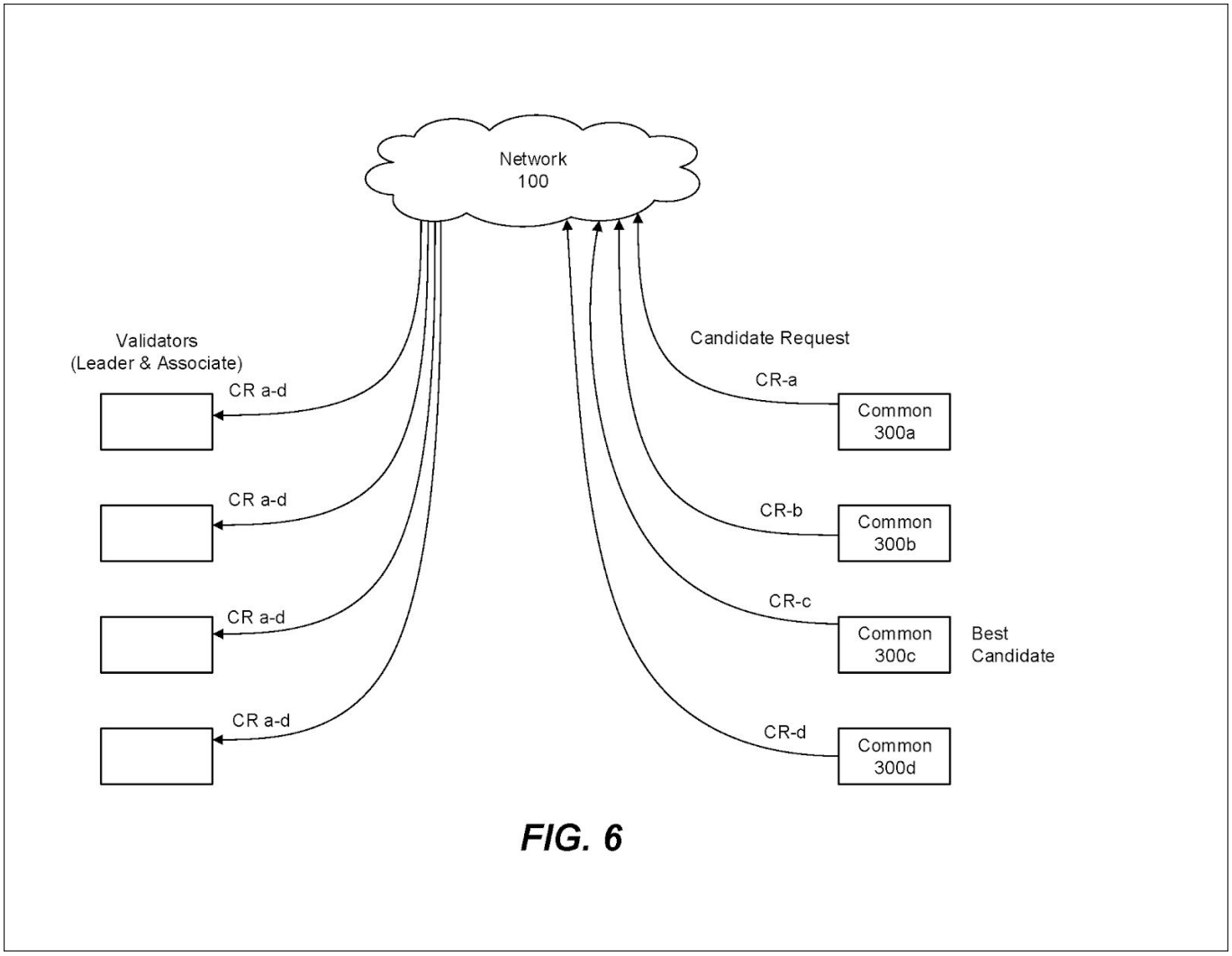


FIG. 6

Figure 6 depicts exemplary communications that may be exchanged among network nodes in network 200 using an exemplary process for reconfiguring the membership of validator nodes on the Committee 210 in accordance with certain disclosed embodiments. The first stage of reconfiguring membership of the Committee 210 may be referred to as a "campaign," where Common nodes 230 may compete or otherwise request to become a new validator node in the Committee 210. After a new campaign has started, a Common node 230 may submit a request ("candidate request") to become a new validator

node to at least some of the other nodes in the P2P network 200, and the request is received by at least one of the validator nodes (Associate nodes 240 and Leader node 260) in the Committee 210. Multiple candidate Common nodes 230, for example nodes 300a, 300b, 300c, and 300d in Figure 6, may send their respective candidate requests CR-a, CR-b, CR-c, and CR-d, to the validator nodes in the Committee after the campaign is started. In some embodiments, the campaign to select a new validator node may end based on the expiration of a certain time period, triggering event, or reception of a message indicating an end of the campaign in the P2P network 200. For example, the campaign may end based on the triggering event that the number of new TBs added to the TransactionBlock blockchain 510 during the term of the current Committee 210 (i.e., since the previous new block addition to Keyblock chain) has exceeded a threshold number. In other exemplary embodiments, however, a campaign may be ongoing without a predetermined termination. In some disclosed embodiments, a candidate Common node 230 may encrypt its candidate request using an epoch public key and transmit the encrypted candidate request through the P2P network. The candidate Common node 230 may be configured to only encrypt a portion of information in the candidate request, such as its IP address. In such embodiments, because the corresponding epoch private key is known only to the validator nodes, only the Associate and Leader nodes may successfully decrypt the received candidate request. But the Common nodes can verify at least a portion of information stored in the candidate request because the corresponding epoch public key is known by all the Common nodes, as described in detail with reference to step 740. For example, in a POW network, a nonce value stored in the candidate request may not be encrypted so the Common nodes can access the nonce value by using the epoch public key and verify if the candidate request is valid under the POW protocol. Each Associate and Leader node 240 and 260 in the Committee 210 selects what it considers is its preferred candidate to join the Committee as a new validator node based on the candidate requests it has received during the campaign.

A candidate request may include a proof of work (POW) or a proof of stake (POS) or a proof of authority (POA), or a combination of any of the above. Based on the POW, POA, or POS data in the received candidate requests, each Associate and Leader validator node determines a “best” (preferred or optimal) candidate Common node to join the next Committee. In accordance with the disclosed embodiments, each of the Leader node and Associate nodes preferably uses the same criteria for selecting the best candidate Common node to join the next Committee.

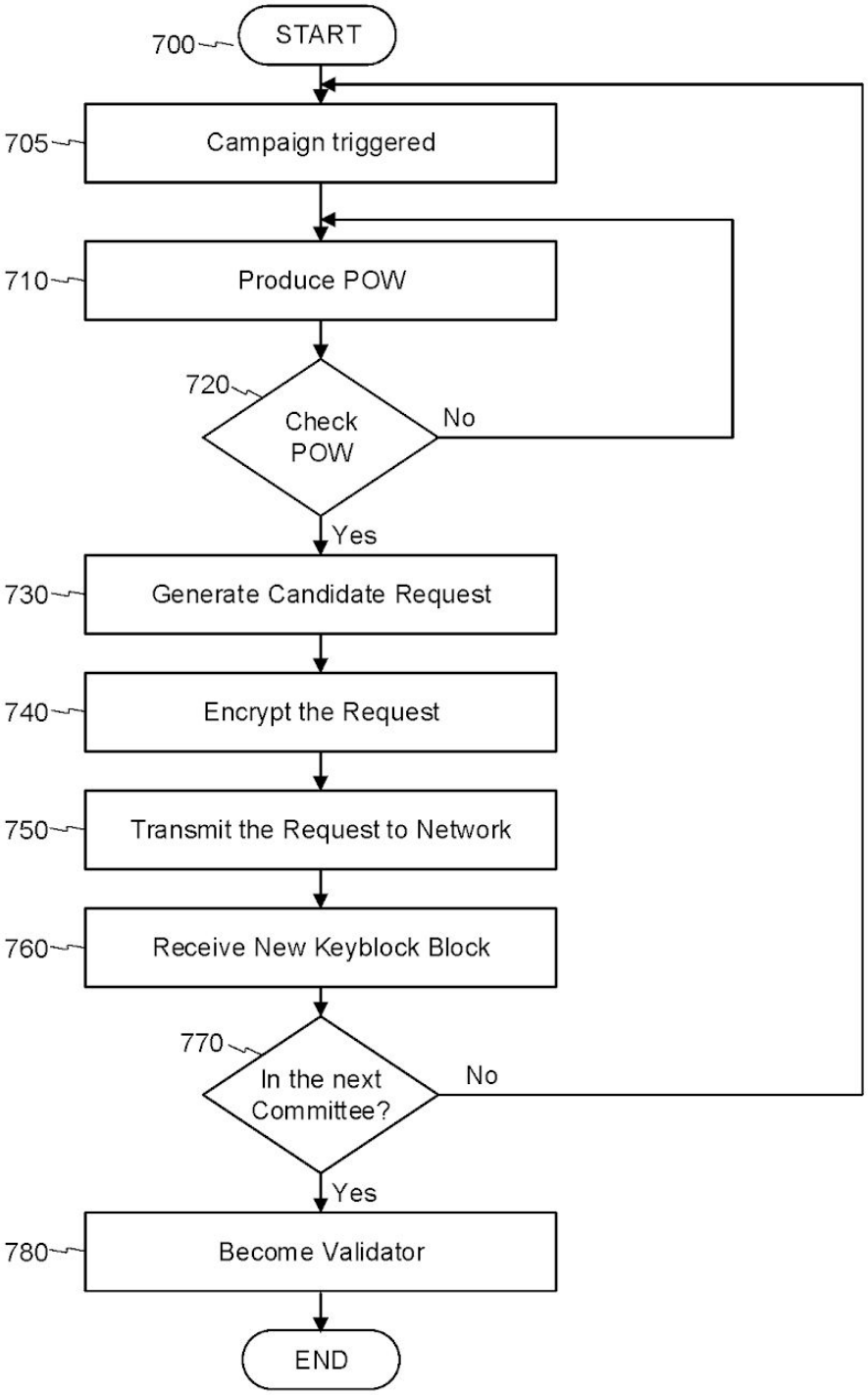

700
START
705
Campaign triggered
710
Produce POW
720
Check POW
No
Yes
730
Generate Candidate Request
740
Encrypt the Request
750
Transmit the Request to Network
760
Receive New Keyblock Block
770
In the next Committee?
No
Yes
780
Become Validator
END

FIG. 7

Figure 7 depicts a flowchart showing a sequence of steps that may be performed by a Common node 230 according to a first stage of the exemplary reconfiguration process in accordance with certain disclosed embodiments. In this context, the Common node is a candidate to become a new Associate validator node in the Committee 210. The sequence starts at step 700 and proceeds to step 705, where a campaign is triggered. There are several ways that a campaign may be triggered in the network 200. In some embodiments, a campaign is always triggered, in which case Common nodes may send candidate requests at any time. For example, in such exemplary embodiments, a Common node may create a new candidate request whenever it has completed one or more POW challenges. In some embodiments, a campaign may be triggered by certain events, such as a failure of a validator node, an update of a network protocol, or any other predefined event(s). In some embodiments, a campaign may be triggered by a certain time interval. In some embodiments, at least one of the validator nodes in the Committee may communicate the start of a new campaign to the Common nodes in the network.

In some disclosed embodiments, a Common node 230 operates in the campaign by producing at least one proof of work, which is a piece of data that is difficult to produce but easy for others to verify. For example, a POW on message *m* may include a hash value of the combination of message *m* plus a nonce value, where the resulting hash value may have to meet certain conditions, for example, being smaller than a threshold value.

In one illustrative example, a candidate Common node 230 may produce its POW (step 710) by repeatedly generating a hash value using different nonce values and each time checking whether the generated hash value using the selected nonce value alone or in combination with a message *m* satisfies a predetermined condition (step 720). If the condition is satisfied, and thus a valid POW is produced, the candidate Common node proceeds to the next step 730. In this example, many algorithms may be used to generate the hash value. For example, in the campaign, the validator nodes may require the hash value to be produced using a SHA-256 algorithm applied to a message *m* and a nonce value, where the hash value must be smaller than a threshold, such as less than 2^{240}. Other algorithms could also be used for producing the POW, such as but not limited to Scrypt, Blake-256, CryptoNight, HEFTY1, Quark, SHA-3, scrypt-jane, scrypt-n, and combinations thereof. More generally, the validator nodes in the Committee may require a POW based on any arbitrary selected piece of verifiable information. For example, the network may be configured to require candidate Common nodes 230 to provide a POW based on the first 80 bytes of a current transaction block or on a hash value of the previous block. In another exemplary embodiment, the network may be configured to require candidate Common nodes 230 to provide a POW based on a snapshot of the current system state. In yet other exemplary embodiments, the candidate Common nodes 230 may run the campaign by producing a proof of stake, for example, based on their age, value, or other weighting algorithm(s). In some other embodiments, the candidate Common nodes 230 may run the campaign by producing POA, POS and/or POW data that are included in their candidate requests.

At step 730, the candidate Common node 230 generates a candidate request. As noted, the candidate request may include a POW, POA, or POS or a combination thereof, and it preferably also includes information about the Common node 230 itself, such as one or more of its Internet Protocol (IP) address or other network address, a network or user identifier, a network or user account number, a public key for the candidate Common node, and/or any other relevant information that may be used for identifying the node. The Common node also may digitally sign at least some of the information in its candidate request using its private key.

At step 740, the candidate Common node 230 preferably encrypts a portion of information stored in the generated candidate request with an epoch public key that is known to all of the validator nodes. The portion of information encrypted may include the IP address of the candidate Common node 230. In some embodiments, the epoch public key may be stored in a current KB 511 of a Keyblock blockchain, which the Common node has accessed to obtain the epoch public key. The encryption may utilize any appropriate encryption algorithm, such as but not limited to SHA3, ECC25519, ED25519, etc. An encryption algorithm is preferably selected so the candidate Common node may encrypt information using the public epoch key, but each of other network nodes in the P2P network 200 cannot decrypt the information unless it knows the epoch private key.

At step 750, the candidate Common node 230 transmits its encrypted candidate request to at least one adjacent ("peer") node in the P2P network 200, wherein the other nodes in the network 200 relay the candidate request to distribute it throughout the network. The candidate request may reach at least one validator node, which in turn makes a decision on whether to add the Common node 230 to a current candidate "pool" (e.g., a list of candidate Common nodes requesting to join the Committee) based on the candidate request it received, e.g., using an exemplary process described with reference to Figure 8.

At step 760, the candidate Common node 230 may receive a new KeyBlock block (in this example called "KB-1" to distinguish it from KBs already part of the KeyBlock chain) that may be created by the Leader node, e.g., using an exemplary process described with reference to Figure 10. The KB-1 corresponds to the next term of the Committee 210 and contains the new validator-node membership of the Committee. KB-1 contains one or more public keys for each of the validator nodes in the next term of the Committee 210. Preferably, KB-1 stores a public "epoch" key, which is a cryptographic key that will be used by the validator nodes during the next term of the Committee 210, e.g., after the membership of the Committee has been modified to account for any new validator nodes and/or any validator nodes from the Committee's previous term that have been removed. In some embodiments, the public epoch key may also be used by the Common nodes. In some embodiments, KB-1 also may store the public keys of the individual validator nodes in the KB-1's corresponding Committee 210.

At step 770, after receiving the new KB-1, the candidate Common node 230 may determine whether it is in the next term of the Committee 210 based on the composition of the Committee corresponding to the information stored in the KB-1. If the candidate Common node 230 determines that it is in the next term of the Committee 210, it proceeds to step 780. If the candidate Common node 230 determines that it is not in the next term of the Committee 210, it returns to step 705 where another campaign may be triggered and then the process may proceed to the next steps with updated information from KB-1.

If the candidate Common node 230 determines that it is included in Committee membership in KB-1, then at step 780, the candidate Common node becomes a validator node. In some exemplary embodiments, the candidate Common node 230 may not become a Leader node in the first one or more terms of the Committee 210 after it becomes a validator node based on predetermined rules. In other exemplary embodiments, the candidate Common node 230 may be permitted to become the Leader node or an Associate node after joining the Committee.

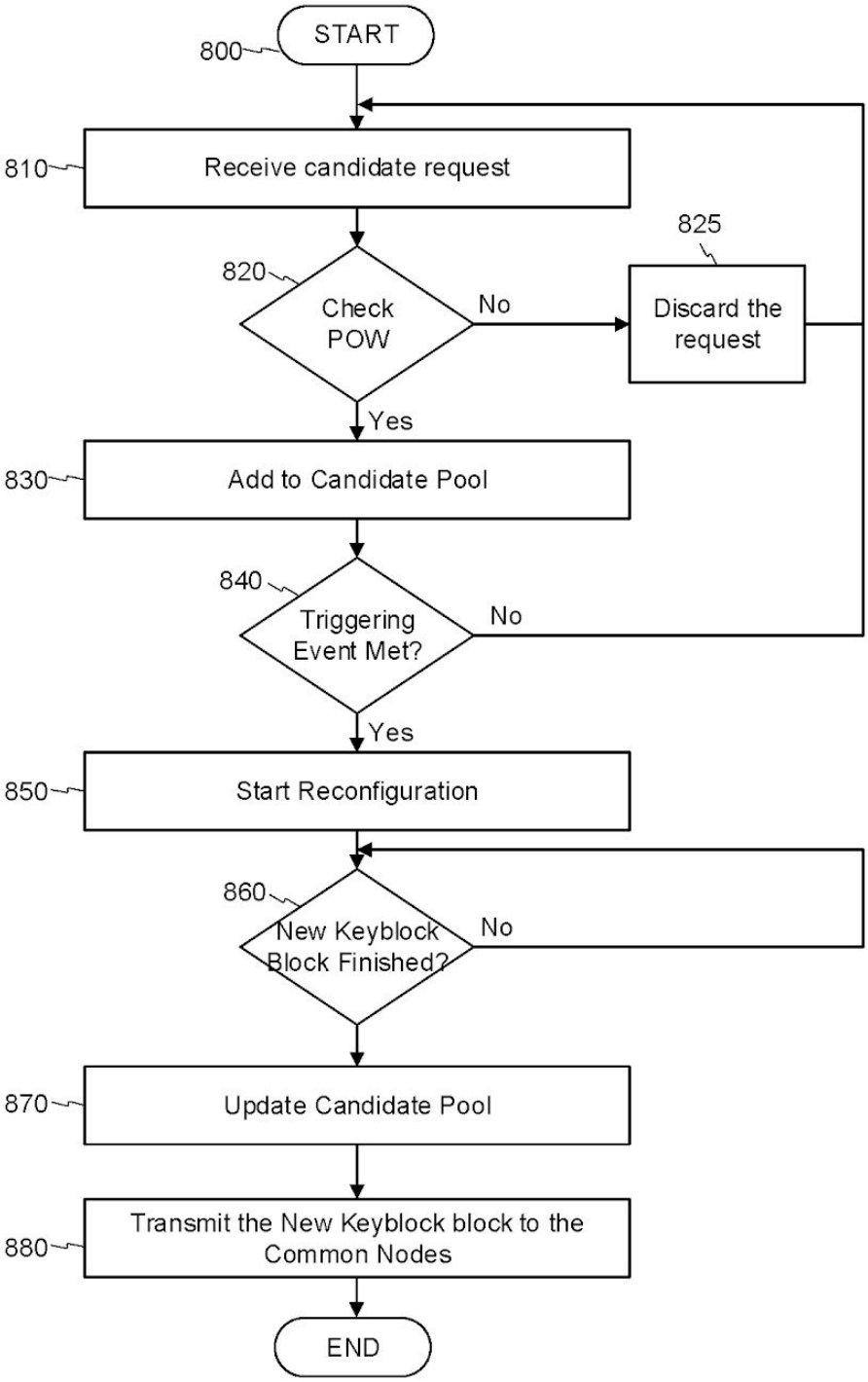

START
800
Receive candidate request
810
825
820
Check POW
No
Discard the request
Yes
Add to Candidate Pool
830
840
Triggering Event Met?
No
Yes
Start Reconfiguration
850
860
New Keyblock Block Finished?
No
Update Candidate Pool
870
Transmit the New Keyblock block to the Common Nodes
880
END

FIG. 8

Figure 8 depicts a flowchart showing a sequence of steps that a validator node may perform as part of the first stage of an exemplary process for reconfiguring membership of the Committee 210 in accordance with certain disclosed embodiments. With respect to this exemplary embodiment, the Leader node 260 and Associate nodes 240 perform substantially same steps in the first stage of the reconfiguration process 800. For this reason, the exemplary sequence of steps in Figure 8 will be generally described below as steps performed by a validator node, without distinguishing steps performed by Associate nodes relative to the Leader node.

At step 810, a validator node receives a candidate request that had been sent from one of the Common nodes 230 in the network 200. At step 820, the validator node verifies the candidate request. In some embodiments, the validator node verifies the candidate request by decrypting it using an epoch private key, e.g., that it previously received from the Leader node 260 or it previously generated if the validator node is the Leader node or it obtains from the current CB of the CommitteeBlock chain 530. If the received candidate request is determined to be valid based on one or more of a POW, POS, or POA at step 820, then the validator node adds the candidate Common node corresponding to the candidate request to a candidate pool at step 830. In this exemplary embodiment, the validator node performing the process 800 maintains its own candidate pool, although in other exemplary embodiments the candidate pool may be accessed and/or modified by one or more validator nodes in the Committee. If the candidate Common node's candidate request is not determined to be valid at step 820, then the validator node discards the received candidate request at step 825 and the process returns to step 810. In some embodiments, as will be described in detail with reference to step 1105, a **Heartbeat** monitor may start right after the campaign starts, or at a certain point before the start of the reconfiguration.

At step 840, the validator node may determine whether a process of request-fulfillment process may begin. The process may preferably begin following the end of the campaign of selecting a new validator node. As described in detail with reference to Figure 6, the campaign may end based on the expiration of a certain time period, triggering event, or reception of a message indicating an end of the campaign in the P2P network 200. In some embodiments, the triggering event may be that the candidate pool contains a sufficient number of candidate Common nodes (or validated candidate requests), for example, exceeding a threshold number. The determination at step 840 may be related to step 1105. In some embodiments, as will be described in detail with reference to step 1105, if the Heartbeat monitor is running and the validator node determines that it has received a sufficient number of candidates, and if the validator node does not receive a PKB-1 from the Leader node 260 by the end of a time period, the validator node may transmit a request for view change to the Committee 210. The validator node will transmit to the Leader node the candidate requests corresponding to the Common nodes that the validator node has added to the candidate pool. If the validator node determines, at step 840, that it has not received a sufficient number of candidates, it may return to step 810 and receive and process more candidate requests. In some embodiments, if the validator node has determined that it has received a sufficient number of candidates, it may stop adding new candidates into its candidate pool and thereby stop sending new candidate requests to the Leader node. In some embodiments, the Common nodes may perform the same process as the validator nodes with respect to steps 810 to 840, such that each Common node also validates and sends to the Leader node 260 its received candidate requests and may have a candidate pool like the validator nodes.

The size of the candidate pool may vary. In some exemplary embodiments, the size of the pool is one, which means the validator node only sends the Leader node a single candidate request corresponding to

the first candidate Common node that the validator node has validated at step 820. In some exemplary embodiments, the size of the candidate pool may be a number selected based on the POW, POA and/or POS being evaluated, or otherwise may be selected based on experience with the particular POW, POA, or POS used to validate candidate requests at step 820. In other embodiments, the size of the candidate pool may be indefinite. In yet other embodiments, the validator nodes may accept candidate requests for the duration of a predetermined time interval. For example, each validator node may accept candidate requests within ten minutes after it receives its first candidate request. In some embodiments, the validator nodes may not perform step 840, and instead send the Leader node any number of Common nodes from its candidate pool.

At step 850, the reconfiguration process is started by the Leader node. Steps 850 -880 will be described in detail in Figures 9, 10, and 11.

Because it may take some time for a candidate request to be transmitted from a candidate Common node to the validator node through the P2P network, in some embodiments, the receiving validator node may wait at least a minimum amount of time before deciding which candidate Common node will be its best candidate, so an earlier-generated but later-received candidate request can be considered, reducing the impact of the P2P network delay to the validator node's decision. In some embodiments, the validator node's selection may be based on elements other than, or in addition to, POW, such as POS or POA. In some embodiments, the validator node may use an equation or formula, such as a weighted equation based on different weights assigned to one or more POW, POS, or POA received with the candidate request, to select a best candidate Common node to send to the Leader node. In accordance with the disclosed embodiments, the validator nodes preferably employ the same criteria for selecting a best candidate Common node to join the next Committee.

At step 860, the validator node determines whether the Leader node has added a new KB-1 to the KeyBlock blockchain, for example, before a predetermined condition has been reached. In some embodiments, the predetermined condition can be a certain period of time or a certain number of new TBs added to the TransactionBlock blockchain. During the reconfiguration process, there may be one or more "timeout" determinations, which will be described in further detail with reference to Figures 10 and 11. If the validator node determines that a KB-1 has not been added to the KeyBlock blockchain before a predetermined condition has been reached, it will generate a new view-change message which it sends to the other validator nodes. If the validator node determines that a new KB-1 has been added to the KeyBlock blockchain before a predetermined condition has been reached, the validator node will proceed to step 870.

At step 870, the validator node updates the candidate pool. In some embodiments, the validator node may clear all candidate Common nodes from its candidate pool. In some embodiments, the validator node may decide whether to remove a candidate Common node from the candidate pool based on whether that Common node had been added to the candidate pool based on a validated POW, POA and/or POS. In some embodiments, for example, the validator node may remove all Common nodes from the candidate pool that had been added to the pool based on their POW candidate request; in some exemplary embodiments, the validator node may keep all Common nodes that were added to the pool based on a POS determination and update the stake of each such Common node based on their remaining stake percentage in the pool.

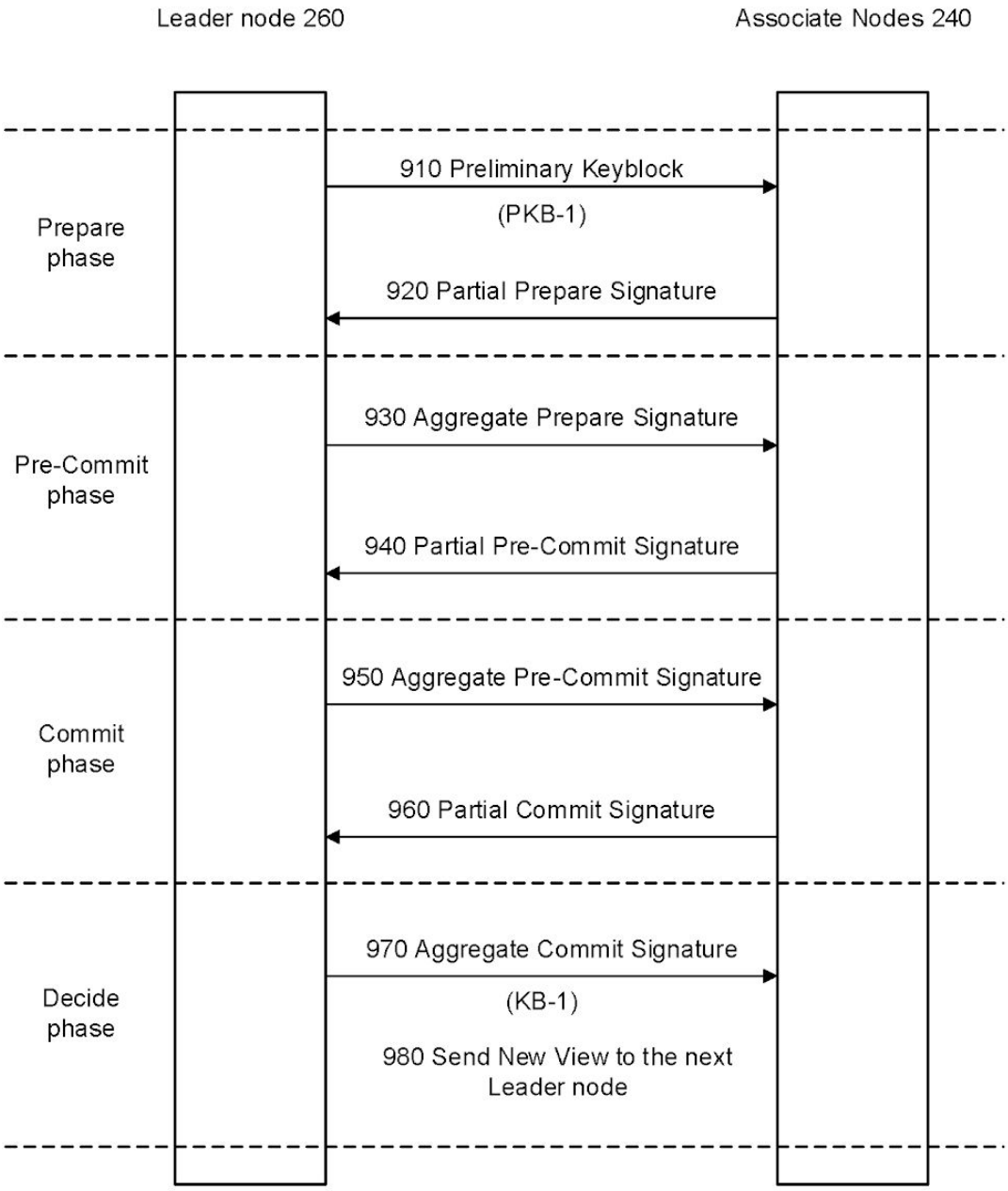

Leader node 260
Associate Nodes 240
Prepare phase
910 Preliminary Keyblock
(PKB-1)
920 Partial Prepare Signature
Pre-Commit phase
930 Aggregate Prepare Signature
940 Partial Pre-Commit Signature
Commit phase
950 Aggregate Pre-Commit Signature
960 Partial Commit Signature
Decide phase
970 Aggregate Commit Signature
(KB-1)
980 Send New View to the next Leader node

FIG. 9

Figure 9 depicts an exemplary exchange of communications among validator nodes that may be used for the exemplary process of reconfiguring the validator-node membership of the Committee 210 in accordance with certain disclosed embodiments. In some embodiments, the Leader node may use a three-phase process to complete the processing (or fulfilling) of a new received transaction request or candidate request. In such embodiments, the three phases are a Prepare phase, Pre-Commit phase, and Commit phase. In some embodiments, at each phase each Associate node may preferably cast vote based on their decision making by generating and transmitting a partial signature to the Leader node. In some other embodiments, each Associate node may cast vote by partial signature and/or other methods.

Before the Prepare phase begins, the candidate Common nodes 230 may transmit their candidate requests through the P2P network 200 to one or more validator nodes as described, for example, at step 750 of Figure 7. The validator nodes that receive the candidate request may in turn make decisions on whether to add the Common node 230 to their candidate pool(s) and whether to send the received candidate request to the Leader node for potentially adding the Common node to the next Committee.

In the Prepare phase of reconfiguring the membership of the Committee 210, the Leader node 260 coordinates with the Associate nodes 240 to reach a consensus over the membership and/or logical configuration of the next Committee 210. The Leader node 260 conducts the coordination by generating and transmitting PKB-1 to the Associate nodes 240 in the Committee. In some embodiments, the Leader node 260 may also transmit each of the candidate requests it receives from the candidate pools of the Associate nodes 240 to every other Associate node in the Committee. In this manner, the Leader node distributes all of the candidate requests that had been successfully validated by validator nodes in the Committee to each of the Associate nodes.

It is possible that not all the Associate nodes in the Committee may have received each of the candidate requests distributed by the Leader node because of various predictable or unpredictable failures in the network. It is also possible that some validator nodes may receive incorrect or illegal candidate requests, for example, from a malicious network node or a malfunctioning node. In either situation, a validator node that fails to receive a proper candidate request may not function properly in participating in the selection of the membership and/or logical configuration of a new Committee. The disclosed embodiments prevent catastrophic failures in the selection of a new Committee and allow the validator nodes to reach a consensus about the composition of the new Committee using a multi-round signature protocol that provides fault tolerance even when one or more of the validator nodes has failed or otherwise becomes compromised. Preferably, in some embodiments, for a fault tolerance of f failed validator nodes, the Committee must include at least $3f + 1$ validator nodes.

At step 910, after the campaign described in Figure 6 has completed, the Leader node 260 verifies the candidate requests it has received, selects a best candidate Common node, and creates a Preliminary KeyBlock block (PKB). For the purpose of illustration and discussion in the exemplary embodiment of Figure 9, the newly-created PKB is referred to as “PKB-1.” The Leader node 260 then transmits PKB-1 and all the candidate requests it received from all Associate nodes to each Associate node 240 in the Committee 210. As described above, in some embodiments, the Leader node 260 and Associate nodes 240 may be connected in a star configuration, in which case PKB-1 is transmitted directly from the Leader node to each of the Associate nodes. In alternative embodiments, for example, the Leader node 260 and Associate nodes 240 may be connected hierarchically, such as in a tree structure, in which case the PKB-1 is transmitted from the Leader node at the root of the tree structure and relayed by Associate

nodes at the various levels of the hierarchical tree structure, until the PKB-1 has been received by each of the Associate nodes.

In some exemplary embodiments, the Leader node 260 may select the best candidate using a verifiable random function (VRF). In other exemplary embodiments, the Leader may select the best candidate using one or more VRFs, one or more nonces (e.g., if the corresponding candidate is added based on POW), one or more weights (e.g., if the corresponding candidate is added based on POS), or any other predetermined criteria, or combinations thereof.

At step 920, each Associate node receives and stores the PKB-1 and candidate requests received from the Leader node 260 in its local memory or in any external memory (local or remote) accessible to that node. The Associate node verifies the candidate requests it receives from the Leader node, for example by verifying the Leader node's digital signature in the received PKB-1, and then chooses its own selection of a best candidate Common node from among all the candidate requests it has received from nodes in the network, including from the Leader node. The Associate node then verifies the accuracy of the received PKB-1 by comparing the best-candidate data in the PKB-1 block with the best candidate Common node that it has selected based on its own determination.

The data in PKB-1 corresponding to the Leader node's selection of a best candidate Common node may include, for example, an IP address, public key, or other identifying information relating to the selected candidate Common node. Based on the verification performed by each Associate node, at step 920 each Associate node may separately generate a Partial Prepare Signature (PPS) and transmit its generated PPS to the Leader node 260, either directly or indirectly depending on the logical configuration of the Committee 210. In some embodiments, each Associate node may only generate or transmit the PPS when the Associate node successfully verifies and approves the data in the PKB-1. The PPS signature may comprise a digital signature of at least some information stored in the PKB-1 block, and may be generated using the Associated node's private key so the Leader node can subsequently verify the PPS signature using the Associate node's corresponding public key.

The Prepare phase in the process of fulfilling a transaction request to add a TB to the TransactionBlock chain 510 is similar to the above-described Prepare phase of fulfilling a candidate request for the process of reconfiguring the membership of the Committee 210, except that the Leader node 260 further coordinates with the Associate nodes 240 to reach a consensus over which transaction request(s) to include in the next TB.

In some embodiments, the Leader node determines whether a new transaction request should be contained in the next TB. Therefore, in such embodiments, every transaction request the Leader node 260 receives from an Associate node may trigger the Leader node to proceed to the Prepare phase. In other embodiments, the Leader node may determine which transaction request or transaction requests should be contained in the next TB. For example, there may be predetermined rules that over a certain time period and/or after receiving a number of transaction requests equal to or above a threshold number, the Leader node 260 may proceed to the Prepare phase. In accordance with certain disclosed embodiments where the validator nodes of the Committee determine whether multiple transaction requests may be contained in the next TB, the PPS may contain the Associate node's votes with respect to each transaction request.

At step 930, at the beginning of the Pre-Commit phase, the Leader node 260 collects all of the PPS signatures it has received from the Associate nodes. In some embodiments, when the Leader node has

received more than a predetermined threshold number of PPS signatures, the Leader node 260 aggregates the received partial signatures from the Associate nodes to create an Aggregate Prepare Signature (APS). The Leader node may transmit the APS signature to the Associate nodes in the same manner that it transmitted the PKB-1.

At step 940, each Associate node 240 verifies the Aggregate Prepare Signature, for example, using the validator node's public key. By verifying the received APS signature, the Associate nodes 240 learn whether a sufficient number of Associate nodes have received and verified the PKB-1. If an Associate node 240 successfully verifies the APS signature, it may generate a Partial Pre-Commit Signature (PPCS) and transmit the PPCS signature to the Leader node 260 in the same manner that it generated and transmitted a PPS signature.

The Pre-Commit phase in the process of fulfilling a transaction request is similar to the Pre-Commit phase in the process of reconfiguring the validator-node membership of the Committee 210, except that the Leader node 260 coordinates with the Associate nodes 240 to reach a consensus over which transaction request(s) to include in the next TB.

At step 950, at the beginning of the Commit phase, the Leader node 260 collects the transmitted PPCS signatures from the Associate nodes. When it receives a threshold number of PPCS signatures, the Leader node 260 generates an Aggregate Pre-Commit Signature (APCS), indicating that the Leader node has verified that a sufficient number of Associate nodes 240 has received and verified the APS signature, which indicates a sufficient number of Associate nodes have received and verified the PKB-1. The Leader node may transmit the APCS signature to the Associate nodes in the same manner that it transmitted the PKB-1 and APS signature.

At step 960, each Associate node 240 verifies the APCS received from the Leader node 260, for example, using the Leader node's public key. In this exemplary process, by verifying the received APCS signature of the Leader node, the Associate nodes 240 learn whether a sufficient number of Associate nodes have received and verified the APS. If an Associate node 240 successfully verifies the APCS signature, it may generate a Partial Commit Signature (PCS) and transmit the PCS signature to the Leader node 260 in the same manner that it generated and transmitted the APCS signature.

The Commit phase in the process of fulfilling a transaction request is similar to the Commit phase in the process of reconfiguring the membership of the Committee 210, except that the Leader node 260 coordinates with the Associate nodes 240 to reach a consensus over which transaction request(s) to include in the next TB.

At step 970, at the beginning of the Decide phase (which is the phase when the candidate request or transaction request is fulfilled), the Leader node 260 collects the transmitted PCS signatures from the Associate nodes 240 in the Committee. When it receives a threshold number of PCS signatures, the Leader node 260 generates an Aggregate Commit Signature (ACS), indicating that the Leader node has finalized the PKB-1 for addition as a new block to add to the Keyblock blockchain 510. The Leader node 260 may broadcast the ACS and the finalized PKB-1, which is referred to as "KB-1" in Figure 9, to each of the Associate nodes and any Common nodes that store copies of the KeyBlock blockchain in the network 200. After receiving KB-1, each Associate or Common node may verify the authentication of the KB-1 by checking the ACS signature and then, if the KB-1 block is verified, store KB-1 in its storage and update its

operation in accordance with the information in KB-1. For example, future candidate requests from the Common nodes 230 may be encrypted using a new public epoch key included in KB-1.

At step 980, each validator node may determine a new Leader node and may also transmit a request for view change to the other validator nodes in the Committee. Step 980 will be described in detail with reference to Figure 12.

The Decide phase in the process when a transaction request is fulfilled is similar to the Decide phase in the process of reconfiguring the membership of the Committee 210, except that the Leader node 260 coordinates with the Associate nodes 240 to reach a consensus over which transaction request(s) to include in the next TB. In some embodiments, the Decide phase for a transaction block may not contain step 980 where a view change may not be present at process of fulfilling every transaction request.

The new Committee 210 may start operating as soon as the KB-1 is formed and is transmitted to at least one of the Associate or Common nodes. In this example, the new Committee includes the newly-selected Common node, which now becomes an Associate node in the Committee. In forming the new Committee, the Leader node also may have removed a previous Associate node to replace it with the newly-selected Common node, for example, based on the age, priority level, or any other criteria associated with the Associated node being removed from the Committee. The new Committee 210 may, in response to future candidate requests, generate the next KeyBlock block, for example KB-2, and establish the next Committee using the same procedure discussed above.

Although the exemplary embodiment above is described in the context of a blockchain implementation, persons skilled in the art will understand that other data-management schemes can also benefit from using the multi-round signature protocol described above in connection with a Committee having a Leader node and one or more Associate nodes. For example, instead of transmitting the PKB-1 to each Associate node, the Leader node 260 instead may transmit a Structured Query Language (SQL) instruction to the Associate nodes at step 910. In such an alternative embodiment, at step 970 the Leader node may transmit at least one finalized SQL instruction associated with its ACS signature for each Associate and/or Common node that receives the finalized SQL instruction to execute.

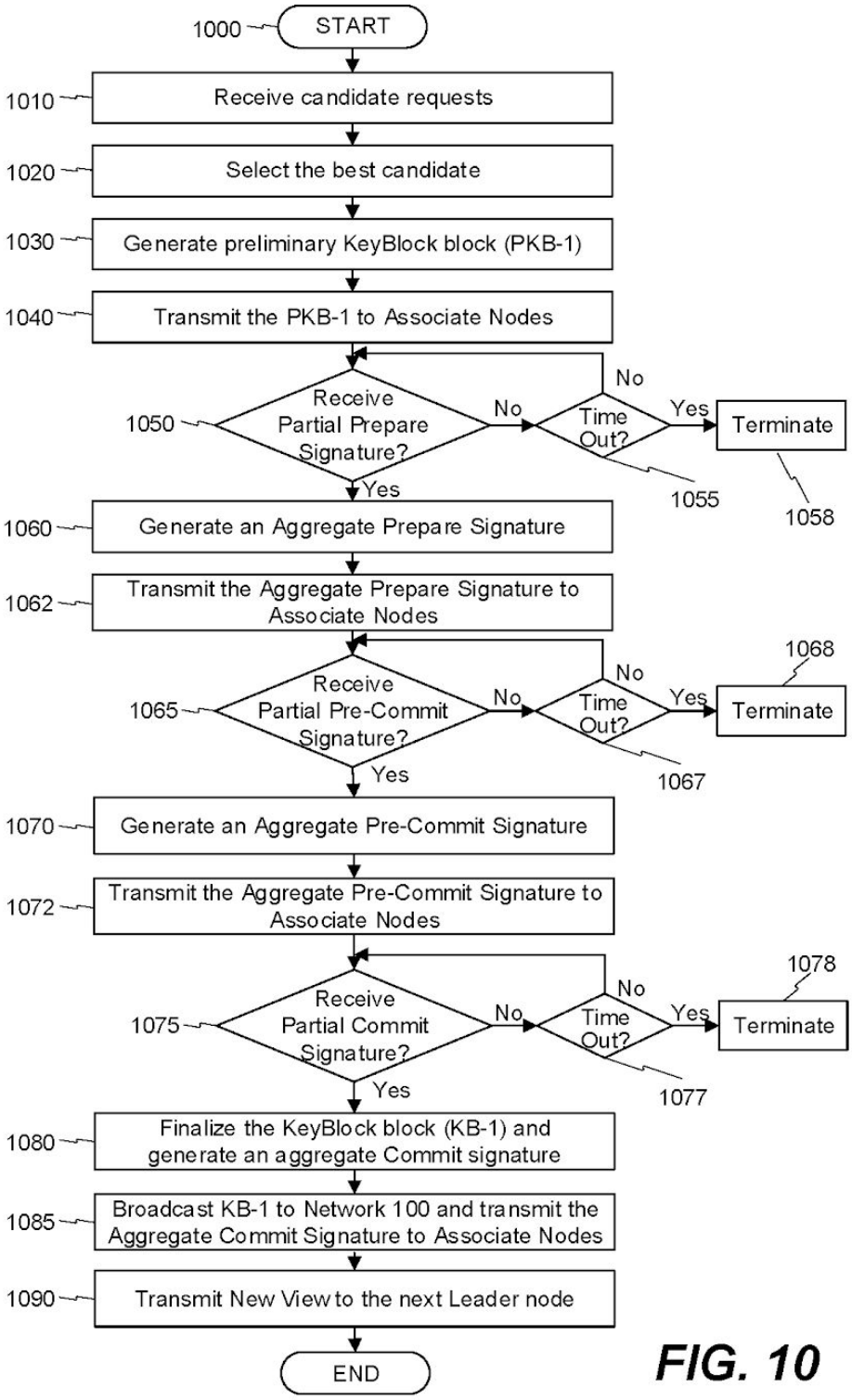

1000
START
1010
Receive candidate requests
1020
Select the best candidate
1030
Generate preliminary KeyBlock block (PKB-1)
1040
Transmit the PKB-1 to Associate Nodes
1050
Receive Partial Prepare Signature?
No
Time Out?
No
Yes
Terminate
1055
1058
Yes
1060
Generate an Aggregate Prepare Signature
1062
Transmit the Aggregate Prepare Signature to Associate Nodes
1065
Receive Partial Pre-Commit Signature?
No
Time Out?
No
Yes
Terminate
1067
1068
Yes
1070
Generate an Aggregate Pre-Commit Signature
1072
Transmit the Aggregate Pre-Commit Signature to Associate Nodes
1075
Receive Partial Commit Signature?
No
Time Out?
No
Yes
Terminate
1077
1078
Yes
1080
Finalize the KeyBlock block (KB-1) and generate an aggregate Commit signature
1085
Broadcast KB-1 to Network 100 and transmit the Aggregate Commit Signature to Associate Nodes
1090
Transmit New View to the next Leader node
END

FIG. 10

Figure 10 is a flowchart showing an exemplary sequence of steps that the Leader node 260 may perform as part of a second stage of a process for reconfiguring the membership of Committee 210 in accordance with certain disclosed embodiments. The sequence starts at step 1000 and proceeds to step 1010 where the Leader node 260 may receive candidate requests from Associate nodes 240 during a campaign. As explained with reference to Figure 6, the campaign may end based on the expiration of a certain time period, triggering event, or reception of a message indicating an end of the campaign in the P2P network 200. In some embodiments, the campaign may end based on a triggering event due to the number of new TBs added to the TransactionBlock blockchain 510 during the term of the current Committee 210 (i.e., since the previous new block addition to Keyblock chain) has exceeded a threshold number of TBs.

At step 1020, after receiving the candidate requests, the Leader node 260 may verify the candidate requests and select a best candidate Common node to join the next Committee. In some embodiments, the Leader node 260 may discard a candidate request if the request does not correspond to a Common node. The Leader node may verify each candidate request against information included in one or more previously-received candidate request(s). In some other embodiments, whether the Leader node has previously selected the Common node does not affect the Leader node's verification of the Common node's candidate request, e.g., based on POW, POS, and/or POA, as long as the current candidate request provides sufficient information for verification. The Leader node 260 alternatively, or in addition, may utilize other verification methods to ensure the authentication and accuracy of each candidate request that it receives.

It is possible that the Leader node 260 may receive only one candidate request from Associate nodes 240 during the first stage of the reconfiguration process. In this case, the Leader node 260 may determine that the Common node corresponding the only candidate request is the best candidate after verifying that candidate request. In some embodiments, the Leader node 260 may receive multiple candidate requests and may select the best candidate among the multiple candidate requests. In some embodiments, the Leader node 260 may select the best candidate based on the type of the candidate requests. For example, if the candidate requests are based on POW, the Leader node may select the one that is the earliest mined and has a valid nonce value. If the candidate requests are based on POS, for example, the Leader node may select the one that has the highest weight percentage of the stake. In other exemplary embodiments, the Leader node may select the best candidate based on VRF methods. In such embodiments, the Leader node may keep a copy of the private key that can compute the hash of the VRF, and store the corresponding public key in the PKB-1 so that Associated nodes 240 can use the public key in the PKB-1 to verify the Leader node's digitally-signed hash created using the VRF methods. In some embodiments, the Leader node may select the best candidate Common node based a combination of the methods above and/or other techniques known in the art. As noted above, in accordance with the disclosed embodiments, the Leader node and Associate nodes preferably use the same techniques for selecting the best candidate Common node to join the next Committee.

In some embodiments of step 1020, after selecting the best candidate, the Leader node 260 also determines other members of the next Committee. In some embodiments, the next Committee is preferably formed by replacing one or more of the existing the validator nodes in the Committee with the candidate Common node(s) corresponding to the best candidate request(s) identified as part of the reconfiguration process. For example, the Leader node may replace an existing Associate node with the selected best candidate Common node. In some embodiments, the oldest validator node will be

removed from the next Committee and be replaced with the selected candidate Common node. In a blockchain implementation, for example, the validator node that consecutively appears in a furthest block of the KeyBlock blockchain and/or CommitteeBlock blockchain may be removed and replaced. In other embodiments, the Leader node 260 may generate a queue of validator nodes according to their age or other indication of seniority, for example, where the oldest validator node is at the rear of the queue and the youngest validator node is at the front. The age of the validators may be defined by the length of continuous time or terms that the validator node has served on the Committee. In some embodiments, the Leader node 260 may take other factors, such as POS or past performance or priority levels, into consideration when generating the order of validator nodes in a queue of validator nodes to replace. For example, the Leader node may modify the order of the queue by moving a validator node to the rear of the queue if it has not been working properly, the PKB-1 to verify the Leader node's digitally-signed hash created using the VRF methods. In some embodiments, the Leader node may select the best candidate Common node based on a combination of the methods above and/or other techniques known in the art. As noted above, in accordance with the disclosed embodiments, the Leader node and Associate nodes preferably use the same techniques for selecting the best candidate Common node to join the next Committee.

In accordance with the exemplary embodiments, the validator node at the rear of the queue may be removed from the next Committee and be replaced with the selected best candidate Common node. Those skilled in the art will appreciate that the queue in the exemplary embodiments may be replaced with any other suitable data structure that the Leader node may use to keep track of an order of validator nodes to remove from the Committee, for example, to replace with a selected candidate Common node. e.g., the validator node has not responded in a previous round of voting, or if the validator node has responded in an incorrect way, or if the validator node has acted suspiciously or maliciously in other ways or has failed to act at all.

In some embodiments of step 1020, the Leader node 260 may determine the next Leader node to serve on the next Committee according to predetermined rules. The next Committee may be referred to as a "new view." For example, in some embodiments, the Leader node 260 may determine the selected best candidate Common node (i.e., the youngest validator node to participate in the next Committee) will be the next Leader node. In some other exemplary embodiments, where a queue of validator nodes has been created, the Leader node 260 may pick a validator node located at a predetermined position in the queue to be the next Leader node. In yet other embodiments, the Leader node may choose a new Leader node according to a predetermined protocol, such as based on a POW challenge among the Associate nodes in the Committee. In yet other embodiments, the Leader node may select a validator node to serve as the next Leader node according to a predetermined function or equation, such as but not limited to a weighted equation involving POS, POA, age, or other parameters. In some other embodiments, the predetermined rules may not require a change of the Leader node for every request fulfillment. For example, the predetermined rules may require the Leader node 260 to determine a next Leader node every ten transaction blocks added.

In some embodiments, the Leader node 260 may collect the public keys of all the validator nodes of the next Committee after the Leader node has determined which network nodes will be on the next Committee. The public keys may have been previously received and stored in Leader node's local memory, or another local or remote memory accessible to the Leader node, or the Leader node may request the public keys from each network node that will serve as a validator node on the next

Committee. The Leader node may further generate a new epoch public/private key pair. The new pair of epoch keys may be used during the term of the next Committee to digitally sign and validate transaction blocks and/or other blockchain blocks. Alternatively, the next Leader node may generate the new pair of epoch public and private keys and transmit, or have transmitted, the new pair of epoch public/private keys to the current Leader node 260. In some embodiments, the new epoch public/private keys are generated by the next Leader node, encrypted using the current epoch public key, and transmitted to the current Leader node, who can validate the encrypted new-epoch keys using the current epoch private key. In some further embodiments, the new epoch public/private keys may also be transmitted to the current Associate nodes, who can also validate the encrypted new-epoch keys using the current epoch private key.

At step 1030, the Leader node 260 generates the PKB-1, which is described above with reference to step 910 in Figure 9. PKB-1, as well as all PKBs, preferably has a structure described with reference Figure 5A. The PKB-1 may contain the public key of the new epoch key pair. In some embodiments, it also may include public keys of all of the validator nodes of the next Committee. In some other embodiments, it may include only the public key of the new Committee member and/or the public key of the Committee member that will be removed in the next Committee. In some embodiments, PKB-1 may contain the height (e.g., length) of a current blockchain and/or other system information or other desired information. PKB-1 may further contain a hash value of the highest (most recent) KeyBlock block in the KeyBlock chain, i.e. the parent KeyBlock block, which may be referred to as KB-0. In this case, the KeyBlock blocks may be linked through their hash values and copies of the KeyBlock blockchain stored and distributed among the validator nodes in the Committee.

After the PKB-1 is generated, the Leader node 260 transmits the PKB-1 to each Associate node in the Committee, either sending it directly or indirectly to the Associate node depending on whether the Leader node has a direct connection to that Associate node (step 1040). In some embodiments, for example, the Leader node 260 may be directly connected to each of the Associate nodes 240, for example, forming a star network. In some embodiments, Associate nodes 240 and the Leader 260 may be connected in a tree structure, in which the Leader node serves as a root node, to which one or more of the Associate nodes are connected. Each Associate node connected to the root then serves as the root node for the next level, to which one or more of the other Associate nodes are connected. In these embodiments, the PKB-1 may be first transmitted to the Associate nodes directly connected to the Leader node 260 and then may be further transmitted by these Associate nodes to other levels of Associate nodes.

At step 1050, the Leader node 260 collects Partial Prepare Signatures from the Associate nodes 240. The PPS signatures may be generated by the Associate nodes using the exemplary process described with reference to Figure 11. The Associate nodes may transmit their PPS signatures to the Leader node 260 directly or route them to the Leader node via the exemplary tree structure described above with reference to Figure 3B. To generate an aggregate signature in certain embodiments, the Leader node 260 must receive a threshold number of partial signatures from the Associate nodes. The threshold number of PPS signatures that the Leader node is configured to receive may vary depending on the selection of the multi-signature scheme and/or the fault tolerance of the system. In some embodiments, the Leader node is preferably configured such that at least $2f+1$ signatures are required to create an aggregate signature in a system tolerating f faults. The Leader node 260 may wait, for example, until expiration of a time-out period that terminates the reconfiguration process or until it has received a

predetermined threshold number of PPS signatures, at step 1055. If the time-out period expires, then the process 1000 may terminate at step 1058 or the Associate nodes may send view-change requests and a view-change process may be started.

At step 1060, the Leader node 260 generates an aggregate prepare signature using a multi-signature scheme. In the disclosed embodiments, many algorithms may be used to generate the APS signature, such as using Schnorr multi-signatures or a Boneh-Lynn-Shacham (BLS) signature scheme. The multi-signature scheme may provide that the Leader node waits until it receives a threshold number of signatures from its Associate node co-signers, and when it does, it can generate an APS signature for the entire Committee of validator nodes. In an exemplary embodiment, the threshold number is $2f+1$ for a system that can tolerate f faulty validator nodes. This threshold may vary depending on the setup of the system. In some embodiments, the aggregate signature generated by the Leader node at step 1060 may be subsequently verified using an epoch public key for the validator-node Committee or by a set of public keys corresponding to individual validator nodes in the Committee.

At step 1062, the Leader node 260 transmits the generated APS signature to the Associate nodes 240 in the same manner as it transmitted the PKB-1 block at step 1040, and, at step 1065, the Leader node collects Partial Pre-Commit Signatures (PPCS) from the Associate nodes in the same manner as it collected the PPS signatures in step 1050. The PPCS signatures may be generated by the Associate nodes using the exemplary process described with reference to Figure 11. The Associate nodes may transmit their PPCS signatures to the Leader node 260 directly or route them to the Leader node via the exemplary tree structure described above with reference to Figure 3B. The Leader node 260 may wait, for example, (1) until expiration of a time-out period that terminates the reconfiguration process or that starts a view-change process; or (2) until it has received a predetermined threshold number of PPCS signatures, at step 1067. If the time-out period expires, then the process 1000 may terminate at step 1068 or a view change process may be started. In some embodiments, the Leader node 260 may transmit the private epoch key by storing it in the PKB-1 that is transmitted to Associate nodes. In some other embodiments, the Leader node 260 may transmit the public epoch key to the Associate nodes at another time in the process 1000, for example, to an Associate node in response to receiving a PPCS signature from that Associate node.

At step 1070, when the Leader node 260 receives at least a predetermined threshold number of PPCS signatures from the Associate nodes, the Leader node 260 generates an Aggregate Pre-Commit Signature (APCS) in the same manner as it generated the APS signature in step 1060. The APCS indicates that the Leader node has verified that a sufficient number of Associate nodes 240 has received and verified the APS signature, which indicates a sufficient number of Associate nodes have received and verified the PKB-1. At step 1072, the Leader node may transmit the APCS signature to the Associate nodes in the same manner that it transmitted the PKB-1 and APS signature in the earlier steps 1040 and 1062.

At step 1075, the Leader node collects Partial Commit signatures (PCS) from the Associate nodes in the same manner as it collected the PPS signatures in step 1050 and PPCS in step 1065. The PCS signatures may be generated by the Associate nodes using the exemplary process described with reference to Figure 11. The Associate nodes may transmit their PPCS signatures to the Leader node 260 directly or route them to the Leader node via the exemplary tree structure described above with reference to Figure 3B. The Leader node 260 may wait, for example, (1) until expiration of a time-out period that terminates the reconfiguration process or that starts a view-change process; or (2) until it has received a

predetermined threshold number of PPCS signatures, at step 1077. If the time-out period expires, then the process 1000 may terminate at step 1078. In some embodiments, the Leader node 260 may transmit a private epoch key to an Associate node in response to receiving a PPCS signature from that Associate node. In alternative embodiments, the Leader node 260 may transmit the private epoch key to the Associate nodes at another time in the process 1000.

At step 1080, when the Leader node 260 receives at least a predetermined threshold number of PCS signatures from the Associate nodes, the Leader node 260 generates an Aggregate Commit signature (ACS) in the same manner as it generated the APS signature in step 1060 and APCS in step 1070. The Leader node finalizes the PKB-1 block by incorporating the generated ACS signature into the PKB-1 block, thereby creating a finalized KeyBlock block (KB-1). At step 1085, the Leader node 260 transmits the KB-1 block to each of the Associate nodes and, optionally, transmits KB-1 to at least one of Common nodes. In the event that a Common node becomes a member of the next Committee, the Leader node 260 may also transmit the ACS and the KB-1 block to the Common node. The Leader node may broadcast the KB-1 block or otherwise transmit it so the new KeyBlock KB-1 is received by each of the Associate nodes and any Common node(s) storing a copy of the KeyBlock blockchain. After receiving the new KB-1 block, the Associate nodes and Common nodes may further transmit the new received KB-1 block to other network nodes in the network 200 to ensure it is added to every copy of the KeyBlock blockchain in the P2P network 200.

In some embodiments, if the request is a transaction request and a new TB is broadcast by the Leader node, the client nodes and common nodes in the network may not need to store the new TB. In some other embodiments, the client nodes and common nodes may store the new TB in the same way as they store KB-1 when the request is a candidate request. Associate nodes and any Common node(s) storing a copy of the KeyBlock blockchain. After receiving the new KB-1 block, the Associate nodes and Common nodes may further transmit the new received KB-1 block to other network nodes in the network 200 to ensure it is added to every copy of the KeyBlock blockchain in the P2P network 200.

A client node or a common node may not need to verify every new transaction block. In some embodiments, when a client node or a common node intends to verify a transaction block, it may access the transaction block and get information about the corresponding KeyBlock, and then access the corresponding KeyBlock to retrieve public epoch key or public keys, then it just need to verify the Aggregate Commit Signature (ACS) in the corresponding TransactionBlock (TB-1) using the public epoch key or public keys. This is because the new transaction block's corresponding KeyBlock has the same committee as the new transaction block. Once the ACS in the TransactionBlock (TB-1) is successfully verified, the client node or the common node has successfully verified the transaction block. Similarly, if a client node or a common node intends to verify a KeyBlock, it may access the KeyBlock to retrieve public epoch key or public keys and verify the ACS in the KeyBlock using the public epoch key or public keys.

This provides a couple advantages over the art. For example, in traditional blockchain protocols such as Bitcoin and Ethereum, to be secure against malicious behaviors such as double spending, when receiving a new block NB-1 in the network, each node needs to wait multiple blocks before it can confirm that the new block NB-1 is acceptable. And this means each node need to store information of at least the multiple blocks for verification and conduct necessary computation based on the information of at least the multiple blocks. In some other traditional protocols, the node need to store information in all the

past blocks in the blockchain, which incurs even more burden on storage and computation power. These issues are avoided by the present disclosures.

At step 1090, the Leader node 260 may transmit the new view to the next Leader node. As described in detail with reference to step 1020, the Leader node 260 may determine the next Leader node according to predetermined rules in step 1020. In some embodiments, the information of the next view may be stored in KB-1 and such that step 1090 may be skipped. In some embodiments, the predetermined rules may not require a change of the Leader node for every request fulfillment and such that step 1090 may be skipped.

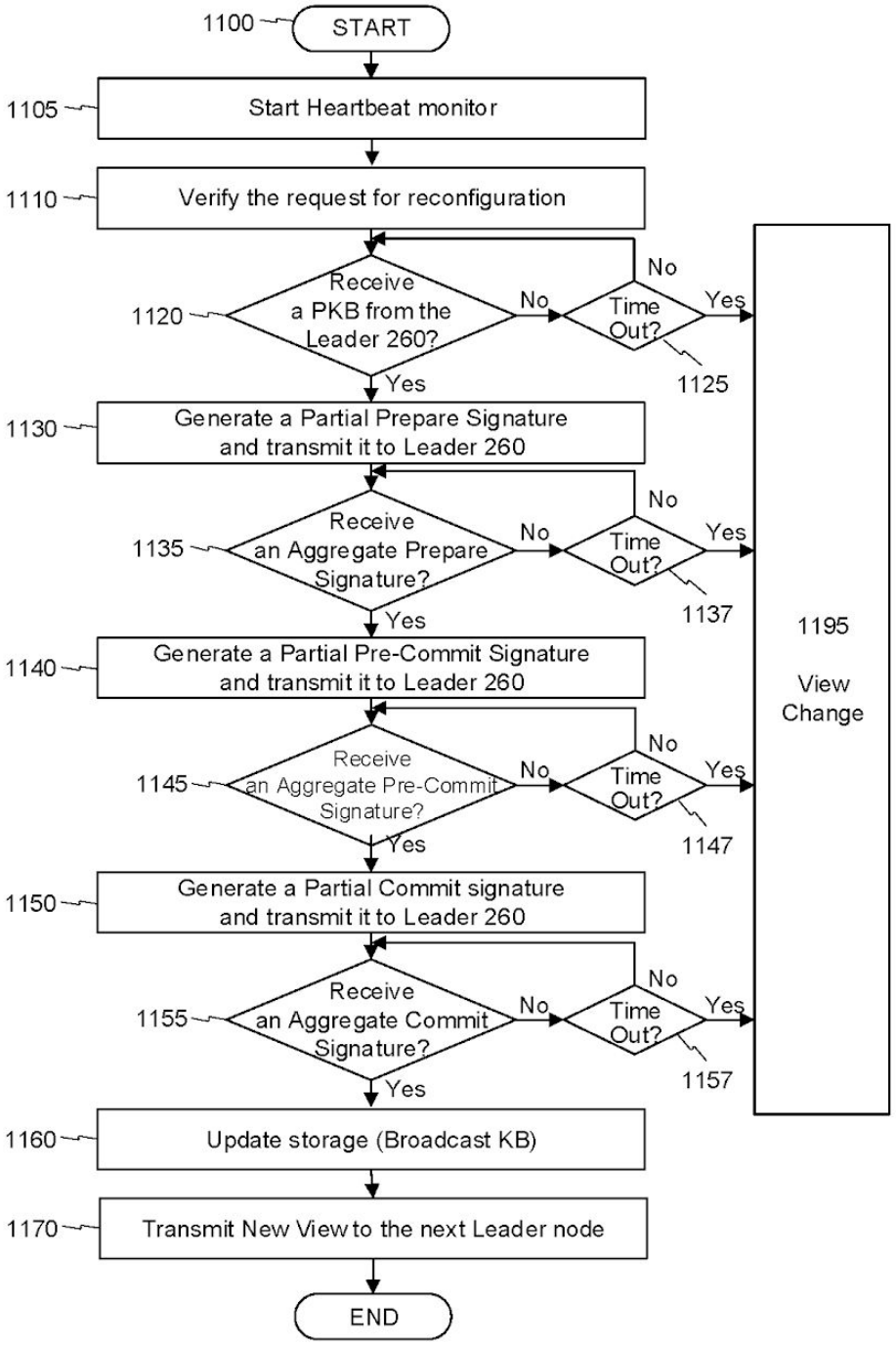

1100
START
1105
Start Heartbeat monitor
1110
Verify the request for reconfiguration
1120
Receive a PKB from the Leader 260?
No
Time Out?
Yes
1125
Yes
1130
Generate a Partial Prepare Signature and transmit it to Leader 260
1135
Receive an Aggregate Prepare Signature?
No
Time Out?
Yes
1137
Yes
1140
Generate a Partial Pre-Commit Signature and transmit it to Leader 260
1145
Receive an Aggregate Pre-Commit Signature?
No
Time Out?
Yes
1147
Yes
1150
Generate a Partial Commit signature and transmit it to Leader 260
1155
Receive an Aggregate Commit Signature?
No
Time Out?
Yes
1157
Yes
1160
Update storage (Broadcast KB)
1170
Transmit New View to the next Leader node
END
1195
View Change

FIG. 11

Figure 11 depicts a flowchart showing an exemplary sequence of steps that an Associate node 240 may perform as part of a second stage of a process for reconfiguring the membership of the Committee 210 in accordance with certain disclosed embodiments. The sequence starts with step 1100, where the campaign begins, such that Associate nodes 240 may receive candidate requests from Common nodes and may transfer some of the received candidate requests to the Leader node 260 after validation. Leader node 260 may receive candidate requests from Associate nodes 240 or Common nodes 230 during the campaign. As explained with reference to Figure 6, the campaign may end based on the expiration of a certain time period, triggering event, or reception of a message indicating an end of the campaign in the P2P network 200.

At step 1105, the Associate node may start a Heartbeat monitor. In some embodiments, the Heartbeat monitor may be used by each Associate node to monitor the “liveness” (e.g., as determined based on responsiveness or operation) of the Leader node 260 and/or other validator nodes in the Committee 210. The Heartbeat monitor is preferably independent from the Leader node 260. The Heartbeat monitor may start before the start of the reconfiguration, and is preferably started at or right after the beginning of the campaign, such that a view-change process can be started, for example, as early as when the Leader node fails to generate and transmit PKB-1 to the Associate nodes 240. In some embodiments, the liveness of the Leader node may relate to whether the Leader node has timely generated a new KB, TB, or performed other tasks that need to be performed by the Leader node, or the like, or any combination thereof. In some embodiments, the Heartbeat monitor may monitor whether the Leader node has timely generated a new KB based on the expiration of a certain time period, triggering event, or reception of a message, or a combination thereof. In some embodiments, the Heartbeat monitor may relate its decision regarding the liveness of the Leader node based on a triggering event of the end of a campaign. For example, if the campaign may end based on the triggering event that the number of new TBs added to the TransactionBlock blockchain 510 during the term of the current Committee 210 has exceeded a predetermined threshold number, the Heartbeat monitor may determine the liveness of the Leader node based on whether a new KB was added to the KeyBlock blockchain a certain time period after the triggering event. In other exemplary embodiments, the Heartbeat monitor may determine liveness of the Leader node based on a combination of factors. For example, the Heartbeat monitor may generally determine the liveness of the Leader node by verifying whether a new KB is added after the number of new TBs exceeds a predetermined threshold (such as 100), but if the network 200 is busy, the Heartbeat monitor may increase the predetermined threshold number of TBs (such as 150). In some embodiments, in the event that the Heartbeat monitor determines that the liveness of the Leader node does not meet its requirement, the Associate node may generate a new view-change message and send it to the other validator nodes in the Committee to start the process of selecting a new Leader node.

In some embodiments, the Associate node performing the process 1100 may receive a request to start a reconfiguration process from another validator node in the Committee, for example, in response to a new campaign or based on the Heartbeat monitor indicating that the Leader node is not sufficiently “live” and should be replaced. The Associate may verify a digital signature in the received request for reconfiguration at step 1110, for example, using a public key of the validator node in the Committee that sent the request to start the reconfiguration process.

At step 1120, the Associate node 240 may receive a preliminary KeyBlock PKB-1. In some embodiments, the Associate node 240 may also receive candidate requests from the Leader node 260 to reduce the

possibilities of failure to reach consensus owing to Associate nodes' failure of receiving candidate requests. Further to the disclosed embodiments, the Associate node may verify whether the received PKB-1 was generated by the Leader node 260 by verifying a digital signature in the PKB-1, for example, using the Leader 260 node's public key.

Further to the disclosed embodiments, the Associate node 240 also may verify whether the PKB-1 was generated in accordance with the candidate requests. In some embodiments, the Associate node may independently select the best candidate based on all of the candidate requests corresponding to its candidate pool and/or it receives from the other nodes, and compare its determined best-candidate Common node with the best-candidate Common node that was selected by the Leader node in the received PKB-1. In other embodiments where the Leader node uses VRF to select a best candidate, the Associate node may use the public key in the PKB-1 to verify the digitally-signed VRF hash and the Leader node's best-candidate selection. In yet other exemplary embodiments, the Associate node 240 may generate its own preliminary KeyBlock block and compare its generated KeyBlock block with the received PKB-1. In some embodiments, if the Associate node does not receive any PKB-1 from the Leader node 260 before expiration of a timeout period at step 1125, the Associate node may initiate a view-change procedure to select a new Leader node at step 1195. The view-change procedure may comprise the Associate node transmitting a request for a view change to the other Associate nodes in the Committee. In some other embodiments, if the PKB-1 fails to pass the Associate node's verification(s), the Associate node also may initiate a view-change procedure. Otherwise, the Associate node 240 proceeds to step 1130.

In some embodiments, the Associate nodes preferably may independently select the best candidate based on the same predetermined rules that the Leader node uses to select its choice of the best candidate Common node. For example, the Associate nodes and the Leader node may select the earliest candidate request that it received to be its selection of the best candidate Common node. In some other embodiments, the Associate nodes may independently select the best candidate based on certain rules that are different from the rules that the Leader node uses to select its choice of the best candidate Common node.

In some embodiments, as described in detail with reference to step 1020, the Leader node 260 may select the next Leader node according to predetermined rules. At step 1020, the Associate node may verify the selection of the next Leader node received from the Leader node. The Associate node may conduct the verification based on the same predetermined rules as the Leader node 260 to select the next Leader node. In some embodiments, the Associate node conducting verification may send their approval to the node that it selected. If an Associate node has received a threshold number of approval, it may proceed to 1130. The threshold number is preferably configured to be $2f+1$.

At step 1130, the Associate node generates a Partial Prepared Signature using its private key. In some embodiments, the PPS signature may be generated using a multi-signature scheme. The Associate node 240 then transmits the generated PPS signature to the Leader node 260. In some embodiments, the Associate node may transmit the generated PPS signature directly to the Leader node 260 or indirectly to the Leader node, for example, when the validator nodes are organized in a tree structure as illustrated in Figure 3B. In an exemplary embodiment, the Associate node may first transmit the PPS signature to its parent node, which in turn relays this partial signature to the next level until the signature reaches the Leader node 260. For example, with reference to Figure 3A, the Associate node 240a1 generates a PPS signature and transmits the partial signature to the Associate node 240b1. The

Associate node 240b1 may generate its own PPS signature, which it may transmit separately or together with the PPS signature that it received from Associate node 240a1, to the Leader node 260.

In another embodiment, multiple Associate nodes may transmit PPS signatures to the same parent Associate node in the hierarchical arrangement of the Committee's validator nodes. The parent Associate node may generate an Aggregated Partial Signature using the PPS partial signatures that it has received from its child nodes and transmit the APS to its parent Associate node at the next level of the hierarchical arrangement. The APS may be a concatenation of multiple PPS signatures (signed or unsigned) or may be otherwise derived from the multiple PPS partial signatures. For example, the Associate node 240a1 may generate a partial signature and transmit the PPS signature to Associate node 240b1. Associate node 240b1 generates an Aggregate Partial Signature using PPS signatures it has received from Associate nodes 240a1 and 240a2 and, optionally, a partial signature generated by itself. The Associate node 240b1 then transmits the generated aggregated partial signature to the next level, for example the Leader node 260, which further aggregates received PPS partial signatures from the Associate nodes.

At step 1135, the Associate node 240 receives an Aggregate Prepare Signature from the Leader node 260. After the APS signature is received, the Associate node verifies the authentication of this APS signature. Depending on the multi-signature scheme used by the validator nodes in the Committee for this stage of the reconfiguration process, the Associate node may verify the APS signature using one or more public keys it has previously received. In some embodiments, if the Associate node does not receive any APS signature before the expiration of a time-out period at step 1137, the Associate node may initiate a view-change process at step 1195. In some other embodiments, if the received APS signature fails the Associate node's verification, the Associate node also may initiate a view change procedure.

After the Associate node has verified the Leader node's APS signature at step 1135, the Associate node 240 can be confident that a sufficient number of Associate nodes in the Committee have received and verified the PKB-1 block. Then, at step 1140, the Associate node 240 generates a Partial Pre-Commit Signature (PPCS) in the same manner as it generated a PPS in step 1130, and then it transmits the PPCS signature in the same manner as it transmitted the PPS signature in step 1130. In some embodiments, the Associate node 240 may generate a PPCS signature using a different algorithm than it uses to generate a PPS signature.

In step 1145, the Associate node 240 receives an Aggregate Pre-Commit Signature from the Leader node 260. After the APCS signature is received, the Associate node verifies the authentication of this APCS signature. Depending on the multi-signature scheme used by the validator nodes in the Committee for this stage of the reconfiguration process, the Associate node may verify the signature using one or more public keys it has previously received. In some embodiments, if the Associate node does not receive any APCS signature before the expiration of a time-out period at step 1147, the Associate node may initiate a view-change process at step 1195. In some other embodiments, if the received APCS signature fails the Associate node's verification, the Associate node also may initiate a view change procedure.

After the Associate node has verified the Leader node's APCS signature at step 1145, the Associate node 240 can be confident that a sufficient number of Associate nodes in the Committee have received and verified the APS signature, which indicates a sufficient number of Associate nodes have received and verified the PKB-1.

Then, at step 1150, the Associate node 240 generates a Partial Commit Signature (PCS) in the same manner as it generated a partial prepare signature in step 1130 and a PPCS in step 1140, and then it transmits the PCS signature in the same manner as it transmitted the PPS signature in step 1130 and the PPCS in step 1140. In some embodiments, the Associate node 240 may generate a PCS signature using a different algorithm than it uses to generate a PPS or PPCS.

In step 1155, the Associate node receives an Aggregate Commit Signature from the Leader node 260. In some embodiments, the Leader node may transmit the finalized the KB-1 with the ACS signature. Further, in some embodiments, the Leader node may only transfer the ACS signature, and each Associate node adds the received ACS signature to the PKB-1 block that it previously received at step 1120, thereby creating the same KB-1 block at each of the Associate nodes. The ACS signature may be verified in the same manner as described in step 1150. In some embodiments, if the Associate node does not receive any Aggregate Commit Signature before the expiration of a time-out period at step 1157, the Associate node may initiate a view-change process at step 1195. In some other embodiments, if the received ACS signature fails the Associate node's verification, the Associate node also may initiate a view change process at step 1195.

After the Associate node has verified the ACS signature, the Associate node may be confident that a sufficient number of Associate nodes have confirmed that they have learned that a sufficient number of Associate nodes received the proper PBK-1 block. At step 1160, the Associate node may add the new Keyblock block KB-1 to its copy of the KeyBlock blockchain and also may update its database in accordance with at least some information contained in the received PBK-1. In some embodiments, the Associate node may further transmit the KB-1 to at least one Common node or other Associate node in the network 200. In some embodiments, as described in detail with reference to Figure 5C, when KB-1 is added to the KeyBlock chain, a new CB 531 is also added to the CommitteeBlock chain 530. In some embodiments, the message data stored in the KB-1 includes a hash of its corresponding new CB 531. In some embodiments, each validator node may generate CB 531 when verifying the new PKB-1, and may add its generated CB 531 to its copy of CommitteeBlock chain 530 when adding KB-1 to its copy of the KeyBlock blockchain. In some other embodiments, the Leader node may transmit the KB-1 with its corresponding CB 531 to the Associate nodes, such that each Associate node may add the CB 531 received from the Leader node to its copy of CommitteeBlock chain 530. Although the KeyBlock data structure described in the context of Figures 10 and 11 is in the form of a blockchain, those skilled in the art will understand that the disclosed embodiments may be implemented when adding new information to other types of data structures that could also be benefit from the reconfiguration processes described above.

At step 1170, the Associate node may transmit the new view to the next Leader node. As described in detail with reference to steps 1020 and 1120, the Leader node 260 may determine the next Leader node according to predetermined rules at step 1020 and the Associate node may verify the Leader node's selection at step 1120. In some embodiments, the predetermined rules may not require a change of the Leader node for every request fulfillment and such that step 1170 may be skipped.

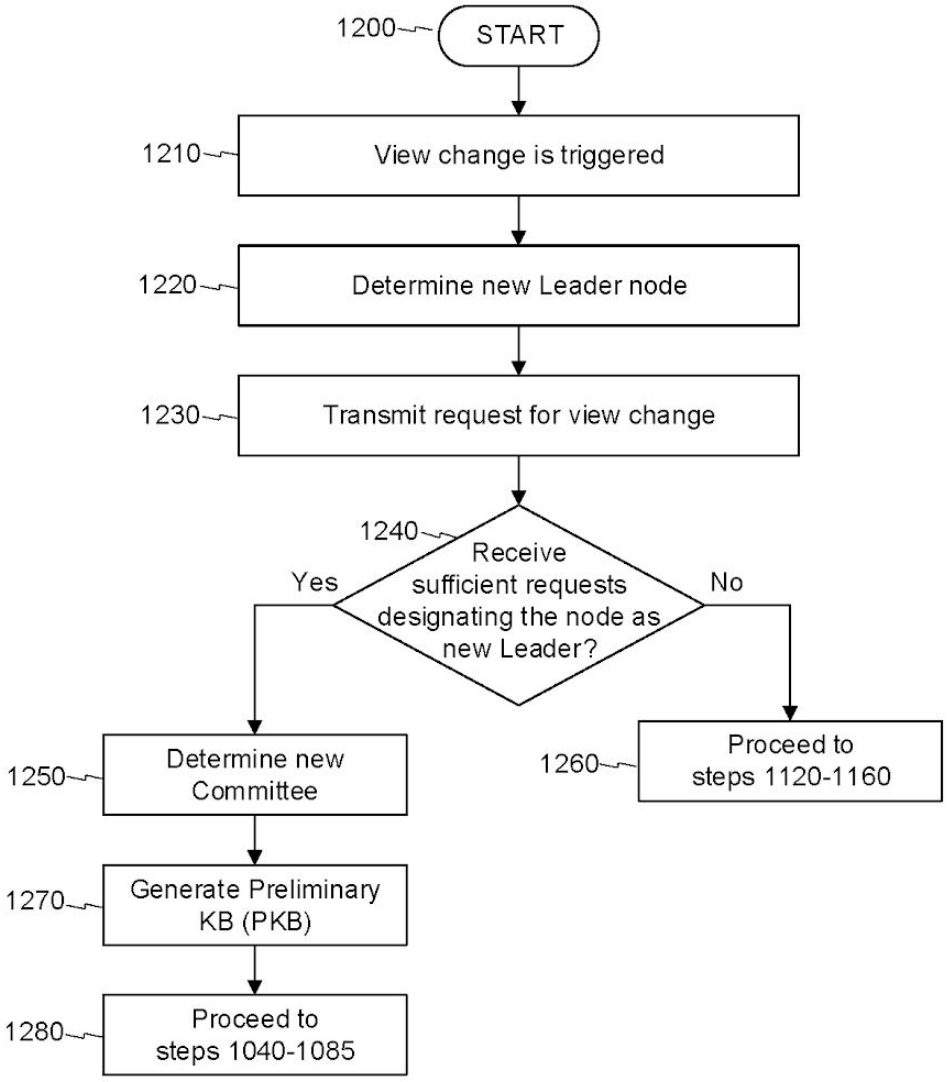

1200
START
1210
View change is triggered
1220
Determine new Leader node
1230
Transmit request for view change
1240
Receive sufficient requests designating the node as new Leader?
Yes
No
1250
Determine new Committee
1260
Proceed to steps 1120-1160
1270
Generate Preliminary KB (PKB)
1280
Proceed to steps 1040-1085

FIG. 12

Figure 12 depicts a flowchart showing an exemplary sequence of steps that an Associate node 240 may perform as part of a view-change process to select a new Leader node in accordance with certain disclosed embodiments of the invention. The view-change process may be used to change the Leader node 260 in cases where the Leader node is deemed inappropriate or unable to continue serving as the Leader node. For example, the Leader node may have stopped responding because of a denial of service (DoS) or distributed DoS (DDoS) attack. In other embodiments, one or more of the Associate nodes may detect that the Leader node 260 has been compromised, for example, by software and/or people within or outside of the entity, such as an enterprise or government, that manages the network 200. An Associate node also may determine that the Leader node is malfunctioning in various scenarios, including the scenarios described with reference to Figure 11. For example, the Associate node may detect that the Leader node has not sent any new KeyBlock and/or TransactionBlock blocks for a predetermined amount of time, which may be a configured time limit set by a system administrator or otherwise agreed upon by the Associate nodes. The Associate node also may detect that the Leader node may have sent out one or more erroneous preliminary blocks, such as blocks including a wrong signature or one or more blocks with incorrect information.

In some embodiments, multiple Associate nodes that detect a malfunctioning of the Leader node 260 may initiate a view-change process at or around a same time. Further, in some embodiments, each Associate node 240 may select a new Leader node according to a set of predetermined rules. For example, a new Leader node preferably may be selected based on a consensus by a majority of the Associate nodes in the Committee. When a consensus is reached to identify the new Leader node, although it has not yet been formally recorded as the Leader node in a new KeyBlock block of the KeyBlock blockchain, the new Leader node may act as a Leader node and generate the new preliminary KeyBlock block PKB-1, which will identify the new Leader node for the next Committee 210. Thereafter, the new Leader node may coordinate with the Associate nodes to reach a consensus decision to finalize and adopt the new KB-1 block recording the composition of the new Committee including the new Leader node.

The sequence starts at step 1200 and proceeds to step 1210, where a view-change process is triggered. A view change may be triggered when an Associate node 240 determines the Leader node is malfunctioning, compromised, or otherwise untrusted or unavailable. The view change also may be triggered when the Associate node receives requests for a view change from a threshold number of Associate nodes. In some embodiments, for example, a predetermined threshold number of Associate nodes required to trigger a view change is $f + 1$ for a system having fault tolerance of f validator nodes.

At step 1220, an Associate node 240 determines, for example according to one or more rules, or otherwise selects a new Leader node. As described with reference to step 1020 of Figure 10, there may be various strategies to select a new Leader node. In some embodiments, the Associate node selects a new Leader node using the same strategy that the Leader node 260 selects the best candidate in step 1020. For example, the Associate node 240 may use a VRF to select the new Leader node and send the public key used for signing a VRF hash to the other Associate nodes for verification. In some embodiments, the Associate node may select a new Leader node using a different strategy than the Leader node 260 uses. By way of example and not limitation, the current Leader node in an exemplary embodiment may be configured to select the youngest validator node in the Committee to be the new Leader node, while the Associate node may be configured to select the oldest validator node in the Committee to be the new Leader node.

At step 1230, the Associate node 240 transmits a request for view change to the other validator nodes in the Committee. The request for view change may include information about the new Leader node, such as but not limited to its IP address, public key, or other identifying information. The request for view change may be encrypted or signed by the Associate node using a known algorithm, using either its own private key or the Committee's epoch private key. In some embodiments, an Associate node may perform this step before performing step 1220 and, thus, the request for view change may not include information about the new Leader node. If a request for view change is transmitted before the Associate node determines the new Leader node, the Associate node may send an updated request, or any other notice or message indicating the Associate node's selection for a new Leader node, after it has determined the new Leader node. In such embodiments, the Associate node may transmit its updated request for view change (or other notification or message) identifying its selection for a new Leader node to its selected new Leader node and, optionally, also to the other Associate nodes. Even if an Associate node determines that it should be the new Leader node, the Associate node still may transmit the request for view change to the other validator nodes in the Committee.

In some embodiments, the Associate node 240 may transmit its request for a view change to the other validator nodes directly, regardless of the network structure of the Committee 210. In some embodiments, for example where a tree structure is adopted for the logical arrangement of validator nodes in the Committee, the Associate node may only transmit the request to its child, parent, and/or sibling nodes, if such nodes exist. In these exemplary embodiments, an Associate node receiving such a request for view change may further transmit the request to its own child, parent, and/or sibling nodes to propagate the request throughout the hierarchical tree structure.

At step 1240, the Associate node 240 receives and verifies requests for view change that it receives from the other validator nodes. It examines each received request to determine whether it has been selected as the new Leader node in the request. In some embodiments, if the Associate node determines that it has received a predetermined threshold number of requests for a view change that designates itself as the new Leader node, then the Associate node assumes it has become the new Leader node in the Committee. The predetermined threshold number for this determination is preferably greater than or equal to $2f+1$ requests identifying the Associate node as the new Leader node, where f is an intended fault tolerance of the system. At step 1260, if the Associate node determines that it has not received the predetermined threshold number of requests and a predefined time-out period is expired, it may continue serve as an Associate node in the Committee or start another round of view-change, choosing a second best leader based on the rules described at step 1220, if it determines a new Leader node has not been properly selected by the network.

At step 1250, the Associate node 240 assumes that it is the new Leader node and determines the validator-node membership of the new Committee 210. In some embodiments, the membership of the Committee does not change except that the Associate node becomes the new Leader node and the existing Leader node becomes an Associate node. In other embodiments, the existing Leader node may be removed from the Committee 210. If the existing Leader node is removed, a new campaign may be triggered to identify a Common node to add to the new Committee as a replacement for the removed Leader node. In alternative embodiments, a new campaign is not triggered when a new Leader node is selected and the old Leader node is removed, in which case the new Committee may contain one less validator node. In such alternative embodiments, one less validator node may be removed in the next reconfiguration process.

At step 1270, the assumed new Leader node generates a Preliminary KeyBlock block PKB-1 reflecting the determined membership of the new Committee as described in steps 1020 of Figure 10. In step 1280, the assumed new Leader node coordinates with the other Associate nodes to perform the process of the second stage of reconfiguration to finalize the new Committee in the same manner as described with reference to Figures 9-11.

At step 1280, where the Associate node determines it is not selected as the new Leader node, the Associate node may act as an ordinary Associate node and coordinate with the new Leader node to perform the process of the second stage of reconfiguration to finalize the new Committee in the same manner as described with reference to Figures 9-11.

After a new KeyBlock block is signed by the new Committee, including the new Leader node, with an Aggregate Commit Signature, the new KB block is broadcasted or otherwise transmitted to network nodes in the network 200 and the view-change process is completed.

Although the steps described with reference to Figure 12 are sequenced, these steps are not necessarily performed in the order of the exemplary sequence. In some embodiments, for example, some of the Associate nodes in the Committee may perform certain steps in Figure 12 first while other Associate nodes may perform these steps later.

The pseudocode for the reconfiguration process is as follows:

Algorithm 1 CypherBFT Reconfiguration

input: $KeyBlock_k$, $Candidate_{i,k+1}$
output: $KeyBlock_{k+1}$

while keyBlockNumber $k = 0, 1, 2, ...$ **do**
  ▷ New-View Phase
  for each committee member from $i = 0 \rightarrow n - 1$ **do**
    if EpochArrived **then**
      $leader_i \leftarrow leader(currentKeyblock_i)$
      $state_i \leftarrow state(currentKeyblock_i, currentTxBlock_i)$
      send MSG(NewView, $state_i$, $candidate_{i,k+1}$) to $leader_i$
    end if
  end for
  ▷ Prepare Phase
  As Committee Leader
    Wait (n - f) New-View messages, $M \leftarrow \{MSG(NewView, state_i, candidate_{i,k+1})\}$
    $candidate_{best} \leftarrow best(\{m.candidate_{k+1}\})$
    $keyBlock_{k+1} \leftarrow nextKeyBlock(candidate_{best})$
    $qc \leftarrow aggregate(\{m.signature\})$
    broadcast MSG(Prepare, $keyBlock_{k+1}$, qc)
  As Committee Member
    Wait Prepare message: $m \leftarrow MSG(Prepare, keyBlock_{k+1}, qc)$ from leader
    $proposedCandidate \leftarrow keyBlock_{k+1}.candidate$
  if qc.valid & $proposedCandidate$ is better than $candidate_i$ **then**
    $sig \leftarrow sign(keyBlock_{k+1})$
    send MSG(VotePrepare, sig) to $leader_i$
  end if
  ▷ Pre-commit Phase
  As Committee Leader
    Wait (n - f) VotePrepare messages, $M \leftarrow \{MSG(VotePrepare, signature)\}$
    $qc \leftarrow aggregate(\{m.signature\})$
    broadcast MSG(Pre-commit, qc)
  As Committee Member
    Wait Pre-commit message: $m \leftarrow MSG(Pre - commit, qc)$ from leader
    $sig \leftarrow sign(qc)$
    send MSG(VotePre-commit, sig) to $leader_i$
  ▷ Commit Phase
  As Committee Leader
    Wait (n - f) VotePre-commit messages, $M \leftarrow \{MSG(VotePre - commit, signature)\}$
    $qc \leftarrow aggregate(\{m.signature\})$
    broadcast MSG(Commit, qc)
  As Committee Member
    Wait Pre-commit message: $m \leftarrow MSG(Commit, qc)$ from leader
    $sig \leftarrow sign(qc)$
    send MSG(VoteCommit, sig) to $leader_i$
  ▷ Decide Phase
  As Committee Leader

Wait (n - f) VoteCommit messages, $M \leftarrow \{MSG(VoteCommit, signature)\}$
$qc \leftarrow aggregate(\{m.signature\})$
broadcast MSG(Decide, qc)
As Committee Member
Wait Decide message: $m \leftarrow MSG(Decide, qc)$ from leader
Inert $Keyblock_{k+1}$
end while

Pipelined CypherBFT

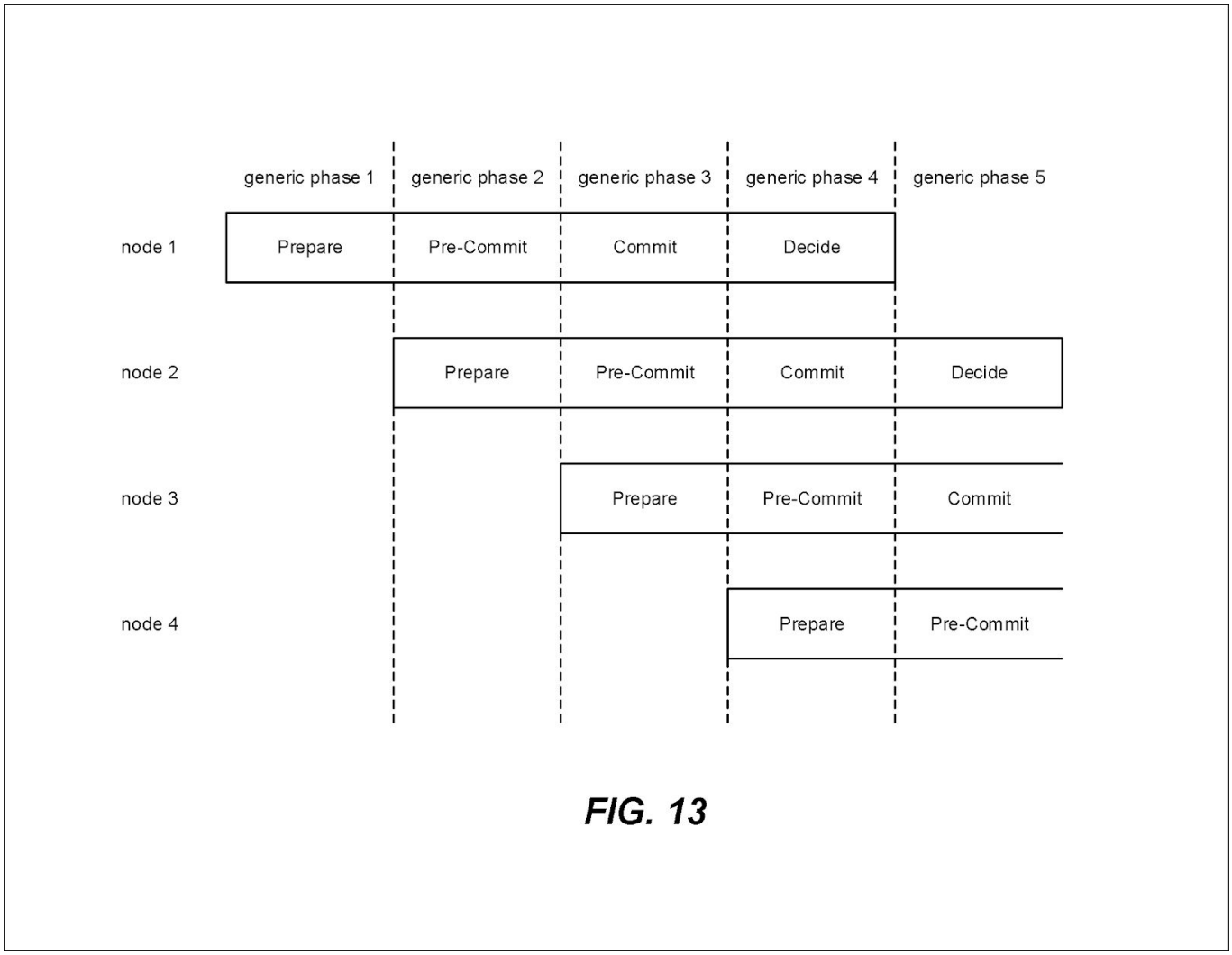


FIG. 13

Figure 13 depicts an exemplary pipeline process for a request-fulfillment process that may be performed in accordance with certain disclosed embodiments of the invention. As used herein, the request-fulfillment processes may be used for processing transaction requests and/or candidate requests in accordance with the disclosed embodiments. The exemplary pipeline process in Figure 13 may be performed using the multiple processing phases that were described above with reference to Figure 9. As explained above, Figure 9 depicts an exemplary sequence of communications among validator nodes that may be used for fulfilling candidate requests or transaction requests in accordance with certain disclosed embodiments. The exemplary process in Figure 9 comprises at least three phases (Prepare, Pre-Commit, Commit) for fulfilling requests and further includes a Decide phase where the request-fulfillment process may be completed.

The pipelined process of the Prepare, Pre-Commit, Commit, and Decide phases, as shown in Figure 13, may allow multiple communications and fulfillment processes to proceed concurrently without interfering with each other. In some embodiments, for example, this exemplary pipelined process may allow multiple concurrent request-fulfillment processes to be performed simultaneously, or at least substantially in parallel, in a pipelined fashion with each concurrent process having a different respective

temporal phase relative to the other processes. In the example of Figure 13, there are four validator nodes that operate in parallel, in a pipelined configuration, to perform respective request-fulfillment processes. Those skilled in the art will appreciate that other embodiments (not shown in Figure 13) may employ a different number of pipelined request-fulfillment processes and would be consistent with the disclosed embodiments.

In some embodiments, a validator node may not be the Leader node for more than one of the pipelined request-fulfillment processes. Further to some of these embodiments, there may be a view-change process to select a new Leader node at a certain predetermined phase of each request-fulfillment process. In other embodiments, for example, a validator node may be the Leader node of one or more of the concurrent request-fulfillment processes. In accordance with certain disclosed embodiments at steps 1020, the predetermined rules may not require a change of the Leader node for every request-fulfillment process. In such embodiments, there may or may not also be a view-change process at a certain phase of the request-fulfillment process.

The Associate nodes' functions in each request-fulfillment process in the embodiment of Figure 13 are not changed relative to their description in reference to Figures 9 and 11 above, except that in the pipelined process the Associate nodes may use aggregated signatures (and in some embodiments, also signatures in the PKB-1, candidate requests, and/or KB-1) based on a Generic Aggregate Signature (GAS) technique rather than using a specific aggregate signature (such as APS, APCS, ACS). The Leader nodes' role in each request-fulfillment process in the exemplary pipelined embodiment also is not changed from Figure 9 and Figure 10 above, except with respect to aggregate signatures as explained below. For example, at a Decide phase of each request-fulfillment process, the Leader node may proceed as described in detail with reference to steps 1075, 1080, 1085 and 1090, except that the Leader node may not be generating and transmitting a specific ACS of its own request-fulfillment process.

Compared to a validator node in a single request-fulfillment process as depicted in the exemplary embodiments of Figures 9, 10, and 11, a validator node in the exemplary pipelined process in Figure 13 may not be an Associate node only or a Leader node only at any given moment. Instead, it is possible that a validator node may be simultaneously an Associate node for some of the concurrent request-fulfillment processes and a Leader node for some of the other pipelined request-fulfillment process(es). For example, as depicted in the generic phase 4 of Figure 13, node 1 may be the Leader node for process 1 while, at the same time, node 1 is also an Associate node for processes 2, 3, and 4.

In accordance with certain disclosed embodiments, the pipelined process may comprise multiple request-fulfillment processes as depicted in Figure 9, each having a different Leader node. In the disclosed embodiment in Figure 13, for example, there are four concurrent request-fulfillment processes that are separately executed by nodes 1-4, such that each of these parallel processes may have a different Leader node. At each moment, there will be four nodes running as the Leader node of their own request-fulfillment process.

In Figure 13, for example, nodes 1, 2, 3, and 4 are validator nodes at least during generic phase 1 through generic phase 5, and they are each the Leader node of a request-fulfillment process starting at a different generic phase. At generic phase 4, for example, node 1 is the Leader node of process 1 that is at the Decide phase; node 2 is the Leader node of process 2 that is at Commit phase; node 3 is the Leader node of process 3 that is at Pre-Commit phase; node 4 is the Leader node of process 4 that is at Prepare phase. In this example, at generic phases 1, 2, 3, and 5, there are also four request-fulfillment

processes but some of the Leader nodes may be validator nodes other than the nodes 1, 2, 3, and 4. For example, at generic phase 1 of Figure 13, the Leader node of process 1 that is at Prepare phase is node 1; the Leader node of a process that is at the Decide phase in generic phase 1 (not shown in Figure 13) may be a validator node other than nodes 1, 2, 3, or 4; the Leader node of a process that is at Commit phase in generic phase 1 (also not shown) may be another validator node other than nodes 1, 2, 3, or 4; the Leader node of a process that is at Pre-Commit phase during generic phase 1 (also not shown) may be yet a different validator node other than nodes 1, 2, 3, or 4.

In accordance with certain disclosed embodiments, at each generic phase of the pipelined process, a generic GAS signature may be generated. For example, at generic phase 1 of the exemplary pipelined process in Figure 13 a GAS-1 signature may be generated; at generic phase 2 of the pipeline a GAS-2 may be generated; at generic phase 3 of the pipeline a GAS-3 may be generated; at generic phase 4 of the pipeline a GAS-4 may be generated. In some embodiments, the GAS signature may comprise an APS, an APCS, and an ACS aggregated signature from three ongoing processes. In other embodiments, the GAS signature may comprise an APS, an APCS, an ACS, a PKB-1, a KB-1, or any combination thereof, from one or more of the pipelined request-fulfillment processes.

In accordance with certain disclosed embodiments, at each generic phase the Leader node of the request-fulfillment process that is at a Prepare phase (“the Leader node in charge”) may be the Leader node to generate the GAS signature at this generic phase. For example, at generic phase 4 of Figure 13, process 4 is at its Prepare phase. Therefore, node 4 may generate GAS-4 which comprises ACS of process 1, APCS of process 2, APS of process 3, and PKB-1 of process 4. Nodes 1, 2, and 3 may not participate in generating GAS-4. Process 1 is at the Decide phase and node 1 is the Leader node of process 1. Node 1 may proceed with steps 1075, 1080, 1085, and 1090 except generating and transmitting a specific ACS of process 1, such that node 1 may generate KB-1 for process 1 upon receiving GAS-4 from node 4. In some embodiments, the GAS generated by the Leader node may be a single value from which each node in each process may interpret the meaning with respect to their process. For example, the GAS-4 may be a single value. Upon receiving GAS-4 as the single value, the validator nodes of process 1 may interpret the value as that there are sufficient number of PCS received for process 1; the validator nodes of process 2 may interpret the value as that there are sufficient number of PPCS received for process 2; the validator nodes of process 3 may interpret the value as that there are sufficient number of PPS received for process 3, etc.

In some preferred embodiments, node 4 may receive partial signatures from other Associate nodes of process 4; node 4 may also receive PCS from other Associate nodes of process 1; node 4 may also receive PPCS from other Associate nodes of process 2; node 4 may also receive PPS from other Associate nodes of process 3. Node 4 may then aggregate all partial signatures node 4 has received , and generate a GAS-4 signature. Node 4 may also generate a PKB-1 at its Prepare phase. In some other embodiments, the Leader nodes of the multiple concurrent request-fulfillment processes may first each collect partial signatures of their own processes, then send their received partial signatures to the Leader node in charge, then the Leader node in charge aggregates the received partial signatures of all the concurrent processes into a single GAS signature for this generic phase. For example, at generic phase 4 of Figure 13, node 1 may send its received PCS to node 4; node 2 may send its received PPCS to node 4; node 3 may send its received PPS to node 4. Node 4 may generate a PKB-1 for process 4, then aggregate the PKB-1 and its received PCS, PPCS, PPS partial signatures to generate GAS-4. In some other embodiments, the Leader node in charge may directly collect partial signatures from all concurrent request-fulfillment

processes and then generate the GAS signature based on the collected partial signatures. For example, at generic phase 4 of Figure 13, process 4 is the Leader node in charge. Node 4 may directly receive partial signatures from Associate nodes of processes 1, 2, and 3, and use them to generate GAS-4. In yet other exemplary embodiments, node 1 may first generate an ACS at its Decide phase, node 2 may first generate an APCS at its Commit phase, node 3 may first generate an APS at its Pre-Commit phase, node 4 may generate a PKB-1 at its Prepare phase, then nodes 1, 2, 3, may send their signatures to node 4 and node 4 may generate a GAS-4 signature.

In some embodiments, the process of generating the GAS signature is performed by each Leader node in turn. For example, at generic phase 4 of Figure 13, in accordance with certain disclosed embodiments, node 1 may first generate an ACS, generate a first partial GAS-4, and then send the first partial GAS-4 to node 2; node 2 may then generate an APCS, aggregate it with the first partial GAS-4 received from node 1 and generate a second partial GAS-4, and then send the second partial GAS-4 to node 3; node 3 may then generate an APS, aggregate it with the second partial GAS-4 received from node 2, generate a third partial GAS-4, and then send the third partial GAS-4 to node 4; node 4 may then generate a PKB-1 and aggregate it with the third partial GAS-4 received from node 3, and then generate a GAS-4 and transmit to all validator nodes. In some embodiments, the GAS signature may be updated and generated simultaneously, or substantially simultaneously, by all the Leader nodes in the different pipelined processes, and there may be a triggering event when the GAS signature is finalized. In some embodiments, the order of the Leader nodes participating in generating the GAS signature may be based on predetermined rules. For example, the Leader node at the earliest phase of its respective request-fulfillment process, such as the Prepare phase, may be the last Leader node to participate in preparing and finalizing the GAS signature for a particular generic phase.

In some embodiments, the Leader node of a request-fulfillment process in the pipeline also may generate a snapshot containing information relating to state of the pipelined system. In some embodiments, only a request-fulfillment process in a pipeline for fulfilling candidate requests may generate a snapshot. In some embodiments, the Leader node may generate the snapshot at the Decide phase of the process for fulfilling candidate requests when it generates the KB-1. The Leader node may then transmit the KB-1 and the snapshot to the new Committee.

The snapshot, for example, may comprise data related to each validator node, status of the ongoing processes in the pipeline, a copy of one or more current blockchains, and the height of one or more blockchains. The data related to each validator node may comprise the identity, the IP address, public keys corresponding to each validator node, and data related to the candidate requests or transaction requests depending on whether each validator node is a Leader node or an Associate node with respect to each concurrent process in the pipeline. The status of the ongoing processes in the pipeline also, or alternatively, may comprise aggregate signatures of all ongoing processes in the pipeline. In some embodiments, a snapshot may be taken by any validator node that is involved in any of the ongoing request-fulfillment processes based on predetermined rules. For example, the predetermined rules may require a snapshot for every 100 new blocks successfully added to the KeyBlock blockchain and/or TransactionBlock blockchain. In this example, the Leader node that added the 100th new block since last snapshot may generate the next snapshot at its Decide phase where the 100th new KB-1 or TB is added.

In some embodiments, in forming a new Committee where the Leader node may have removed a previous validator node to replace it with a newly-selected Common node, the removed validator node also may be the Leader node of another ongoing process in the pipeline. In accordance with the

disclosed embodiments of the invention, after a Leader node of a candidate request-fulfillment process generates a snapshot and transmits it to the new Committee, the ongoing process ("affected process") where the removed validator node is the Leader node of another request-fulfillment process may be treated based on predetermined rules.

In some embodiments, at each Decide phase of a process in the pipeline there may be a view change and the new Leader node may be the Leader node of the affected process. The view change may be based on a predetermined leader-rotation rule. In some embodiments, the leader-rotation rule may be a cyclical sequence of each validator node corresponding to the sequence of request-fulfillment processes. In other exemplary embodiments where there may not be a view change at every Decide phase, the predetermined rules may ensure there is a view-change process by the end of the generic phase where the Leader node is the removed validator node, and the new Leader node may be the Leader node of the affected process.

In some embodiments, the new Leader node of the affected process may take over the aggregated signature from the removed previous validator node and continue with the affected process. In accordance with certain disclosed embodiments, the aggregated signature may be formed using a GAS signature. For example, process 1 in Figure 13 may be a candidate request fulfillment process and node 5 may be determined to be the newly-added Common node replacing node 4 in the Committee. At generic process 4 where process 1 is at Decide phase, GAS-4 is generated and transmitted to validator nodes in the new Committee that includes node 5 but excludes node 4.

In some other embodiments, the affected process is abandoned and the new Leader node may restart the affected process from its Prepare phase. For example, process 1 in Figure 13 may be a candidate request-fulfillment process and node 5 may be determined to be the newly-added Common node replacing node 4 in the Committee. At generic process 4 where process 1 is at Decide phase, a GAS-4 signature may be generated and transmitted to validator nodes in the new Committee that includes node 5 but excludes node 4. Node 5 also may be determined by the view-change process to be the new Leader node of process 4. In this example, process 4 may be terminated and the GAS-4 signature does not include an aggregate signature generated at generic phase 4. Instead, Associate nodes of the process 4 may send transaction requests to node 5. Then, at generic phase 5, during the Prepare phase of process 5, node 5 may generate a PKB-1 and transmit the PKB-1 to its Associate nodes. In any event, the affected process 5 need to be restarted, the Committee and the next Leader node may be able to conduct the process at any time because the snapshot may contain the related data.

While illustrative embodiments have been described herein, the scope of any and all embodiments having equivalent elements, modifications, omissions, combinations (e.g., of aspects across various embodiments), adaptations and/or alterations as would be appreciated by those skilled in the art based on the present disclosure. The limitations in the claims are to be interpreted broadly based on the language employed in the claims and not limited to examples described in the present specification or during the prosecution of the application. The examples are to be construed as non-exclusive. Furthermore, the steps of the disclosed routines may be modified in any manner, including by reordering steps and/or inserting or deleting steps.

For example, the methods and systems described herein may be deployed in part or in whole through a machine that executes computer software, program codes, and/or instructions on a processor. The present invention may be implemented as a method on the machine, as a system or apparatus as part of

or in relation to the machine, or as a computer program product embodied in a computer readable medium executing on one or more of the machines. In some embodiments, the processor may be part of a server, cloud server, client, network infrastructure, mobile computing platform, stationary computing platform, or other computing platform. The storage medium associated with the processor for storing methods, programs, codes, program instructions or other type of instructions capable of being executed by the computing or processing device may include but may not be limited to one or more of a SSD, optical disk, flash memory, hard disk, RAM, ROM, cache and the like.

The methods and/or processes described above, and steps associated therewith, may be realized in hardware, software or any combination of hardware and software suitable for a particular application. The hardware may include a general- purpose computer and/or dedicated computing device or specific computing device or particular aspect or component of a specific computing device. The processes may be realized in one or more microprocessors, microcontrollers, embedded microcontrollers, programmable digital signal processors or other programmable devices, along with internal and/or external memory. The processes may also, or instead, be embodied in an application specific integrated circuit, a programmable gate array, programmable array logic, or any other device or combination of devices that may be configured to process electronic signals. As noted above, it will further be appreciated that one or more of the processes may be realized as a computer executable code capable of being executed on a machine-readable medium.

Smart Contract

Since the advent of Ethereum, smart contracts have become a standardized feature of later, more advanced blockchains. A smart contract is a set of machine instructions stored on the blockchain ledger and executed in the virtual machine.

Because Ethereum emerged much later than Bitcoin, it has significantly improved the functionality, positioning, and design architecture of smart contract platforms, while avoiding the defects of the Bitcoin script. Vitalik and company have brought our industry's vision of a scalable, enterprise-ready smart contract platform much nearer to reality — especially with their invention of the Ethereum Virtual Machine (EVM). Virtual machines play an integral role in the successful implementation of smart contracts, as they allow any network participant to access and run a smart contract's code while securing the integrity of that contract. In other words, anyone can execute the contract, but no one can violate its design or change its intended outcomes.

Problem

Ethereum has several serious problems that are also commonly found in other smart contract platforms:

1. Using 256-bit integers results in serious performance degradation. Most popular computer processors are built on 32-bit or 64-bit RISC architecture. As a result, most devices, unable to support 256-bit operations, slow down the computation speed dramatically.
2. Missing standard libraries and tool libraries. In the development of their smart contract platform, Ethereum developers realize that no standard library presents in their language, Solidity. They can only copy and paste the code from some open source software. Firstly, the security of these codes is not guaranteed, and people may modify the code unwisely for smaller Gas consumption, which may introduce more serious security issues to their contracts.

3. No data object support. Data objects are nowadays widely used in object-oriented programming. Common data formats such as JSON, XML are used by most APIs to deliver data. The lack of data object support has made Solidity extremely burdensome to process data input and integrate with external APIs.
4. Difficult to debug and test. This problem not only presents a design flaw of EVM, but also relates to its difficulty of implementation. The only exception that EVM can throw is OutOfGas, and there is no debug log or external code available.
5. Floating point and fixed-point operations are not supported.
6. Poor security. Since safe math is not supported by default, there are often "one-hundred-million-dollar code losses" caused by integer overflow vulnerabilities. And because there is no sandbox-style security isolation, it has recently been said that the EVM has been completely broken by a "nuclear bomb" of security exploitation.
7. Unreasonable billing, the high cost of application. EVM not only makes writing good code difficult, it also makes it awfully expensive. For example, storing data on a blockchain requires a lot of Gas. This means that the cost of caching data in a smart contract can be extremely high, so data is often recalculated each time the contract runs. As the contract is continuously executed, more and more gas and time are spent on repeatedly calculating the same data, and after all that, the cost of the code cannot be simulated adequately offline.
8. Only limited storage. No consideration for secure docking of external resources such as IPFS, etc.

Cypherium Virtual Machine

The consistent execution of smart contracts must derive from the deterministic nature of its main blockchain's execution results, which means that all transactions must be handled strictly by their chronological order (total order).

Many of the public blockchains that have been launched now support smart contracts, but after careful analysis of their structure, most of them have not overcome the above-mentioned defects of Ethereum, let alone true innovation. Some do not have general purpose functionalities and focus on specific domains only, such as Libra's MOVE.

By way of introduction, here are some features of the Cypherium Virtual Machine (CVM):

1. Based on Java. Cypherium adopts the popular Java Virtual Machine (JVM) architecture for its smart contract computation environment. The JVM has the biggest number of device installations in the world. The world's most popular operating system, Android is a JVM variation. Cypherium Virtual Machine (CVM) supports the complete Java instruction set and can seamlessly integrate with other Java APIs, including Java SE, Java EE and Android. The Java language is a platform independent programming language adopted by Cyherium smart contracts. Once the Java program is compiled into the Java bytecodes, it can run on any JVM-enabled devices. In addition to being the most widely used, the Java language has the largest developer community in the world. There are a large number of third-party libraries on the network for quick development and use. Cypherium's smart contract also supports data types such as XML and JSON, which are used by most financial systems. There are already many powerful IDEs available, such as IntelliJ Idea, Eclipse, VS Code, etc. Running on Java allows for high portability, support for floating point operations, and easy CPU optimization. And, of course, there's all those killer apps.

2. Hierarchical calculation. At present, all known blockchains handle smart contracts in unified deployment, with unified computing and unified consensus processing. In real life, because of the different application scopes, the power capacities of all of our computing devices are quite varied. Cypherium allows all of these computing devices to run the CVM, without affecting the consensus processing of the blockchain's computation results. Cypherium separates the calculation of the smart contract from the consensus of calculation results, which allows the network to categorize computation according to the power of the particular device. Nodes are only responsible for BFT consensus on all computation results to ensure consistency. This design allows the Cypherium chain to adapt to a wide range of application scenarios and apply smart contracts at multiple layers. For example, the mainframe can develop complex contracts with huge amounts of computing; PC nodes can develop contracts suitable for PC computing; mobile devices can run smart contracts for mobile devices; and so on.
3. Native 64-bit integer support and fixed-point representation. The CVM has been implementing native acceleration processing on 64-bit integer and fixed-point operations according to different CPU types, thereby improving the computing speeds of all smart contracts.
4. Support for runtime and compile-time security checks. The risks of Ethereum smart contracts come mainly from one of two sources: the vulnerability caused by the flaws in the design of the smart contract language itself, or more importantly, the code loophole caused by the coders' unfamiliarity with the smart contract language. To avoid these problems, the CVM provides an automatic security check mechanism during both compile and execute phases, checking for buffer overflows, stack overflows, excessive computation, excessive memory consumption, and unsafe external requests. Therefore, the reliability and security of the execution of the smart contract are guaranteed, and the robustness of the whole system is ensured to a large extent.
5. Transparent billing mechanism. CVM only bills the computing workload, memory consumption, and storage capacity, and can provide external tools for calculation. Users can clearly calculate the GAS cost without accessing the network. If the network is congested, the user can pay a fixed rating priority fee. The priority fee (temporarily set to an exponent of 2, depending on the appropriate level 1, 2, 3, 4, 5, or 6) is prioritized for queuing, and the fee structure is fixed and transparent.
6. Smart contracts can be updated. In the case where all relevant accounts vote to agree, the smart contract containing breached vulnerability can be deprecated, and the newly deployed contract can inherit the relevant status and data of the prior contract.
7. Fully compatible with all smart contracts in Ethereum. There may be nearly a million smart contracts that have matured on Ethereum. In order to attract these and future users, the CVM provides a fully compatible Ethereum smart contract mechanism that allows users to deploy directly to the Cypherium chain without any code changes. The logic and results are exactly the same.
8. Enhanced security. All operations Cypherium virtual machines are executed under sandbox isolation, and internal procedures will always provide its own operating status to an external monitor. If unforeseen circumstances are found, such as attacks, infinite loops, the occurrence of astronomically high numbers (often caused by memory overflow), the CVM will immediately halt its operation. This design guarantees the normal operation of a given node and protects the user's on-chain assets from accidental loss. At the same time, the validation committee performs BLS signature processing on the execution result, so that the execution result state

data cannot be tampered with during the network communication process, thereby effectively preventing data forgery and ensuring the consistency of our results.

9. Self-contained cross-chain atomic operation package. With the diversification of blockchain applications, there are more and more public chains, consortium chains, and private chains. However powerful a chain may be, if it cannot communicate with other chains, it will become isolated, and the application community will not flourish. Cypherium will provide interfaces to COSMOS and ILP ecosystems. Users can easily use those interfaces in smart contracts, and can even perform decentralized multi-currency trading in the contract, so that the Cypherium ecosystem will grow and thrive with an active, diverse user community.
10. Secure docking and external storage package. Storage in the blockchain is very expensive, but the current IPFS solution can solve this problem well. IPFS is a peer-to-peer distributed file system that attempts to connect to the same file system for all computing devices. The system utilizes the advantages of blockchain protocol and network infrastructure to store immutable data, while automatically removing duplicate files from the network. When we talk about the underlying blockchain technology of IPFS, we have to mention FileCoin, which provides a market for host and uploader to transact, built on IPFS. The cost of storage can adjust according to the market, and the uploader can select the speed by redundancy and cost. It divides nodes into two types: storage and retrieval. Anyone can become a storage node through leasing out the extra storage space on their own disk on their own extra storage space leased out. Retrieval nodes, which need to be as close to the storage nodes as possible, also require a higher bandwidth and lower latency, anasd the user will pay for the retrieval node that returns the file fastest. Cypherium-IPFS docking mainly refers to docking the above-mentioned retrieval nodes and providing query functions, so that Cypherium smart contracts can check in a timely manner the various states of software, audio, and video on IPFS, thus facilitating related transactions.
11. Nodes can upgrade new virtual machine functionality without downtime. When designing a virtual machine, the Cypherium team built the core functionalities internally, such as our instruction set, consensus mechanism, signature system, and consistency checks. Most of the functional modules, however, are externally linked. When running, the CVM will dynamically search for related modules as needed. This feature makes it easy to upgrade non-core features of the CVM, which becomes especially important for large collaborative computing scenarios. Cypherium provides a call stack depth adjustment mechanism that allows miners to make appropriate adjustments based on node performance and user-facing population.
12. Functionality modularization. Most of the current blockchain designs use unified computing of all smart contracts. In hard-copy contracts, different contracts often correspond to different application user groups. For example, gambling contracts often only correspond to fans and gamblers, and option contracts often only correspond to enterprises employee groups. Accordingly, different contracts should correspond to components with different computing capabilities. The Cypherium team separates the calculation and consensus consistency by design. Nodes are responsible for the results of the operation. This makes it unnecessary to require all nodes' code modules to be the same. The nodes can select relevant functional modules according to their own user groups. These functional modules can run without permission on the test network, but in order to run on the official network, they must obtain the official certification of the Cypherium organization. However, the verification node must be fully functional to ensure consistent execution results. In this way, users can participate extensively in the development of the virtual machine and its smart contracts, and they can release smart

contracts on the test network, which can form a functional component to be bought and sold within the community ecosystem, so that our functions grow more and more powerful, offering the best possible experience of a smart contract platform.

13. Compatibility with other VMs and smart contract languages. Because Cypherium's consensus layer is agnostic to the smart contract layer, Cypherium also supports any VM of the user's choice, including EVM. Any programming language through a translator, can be compiled to CVM bytecode and executed by the CVM.

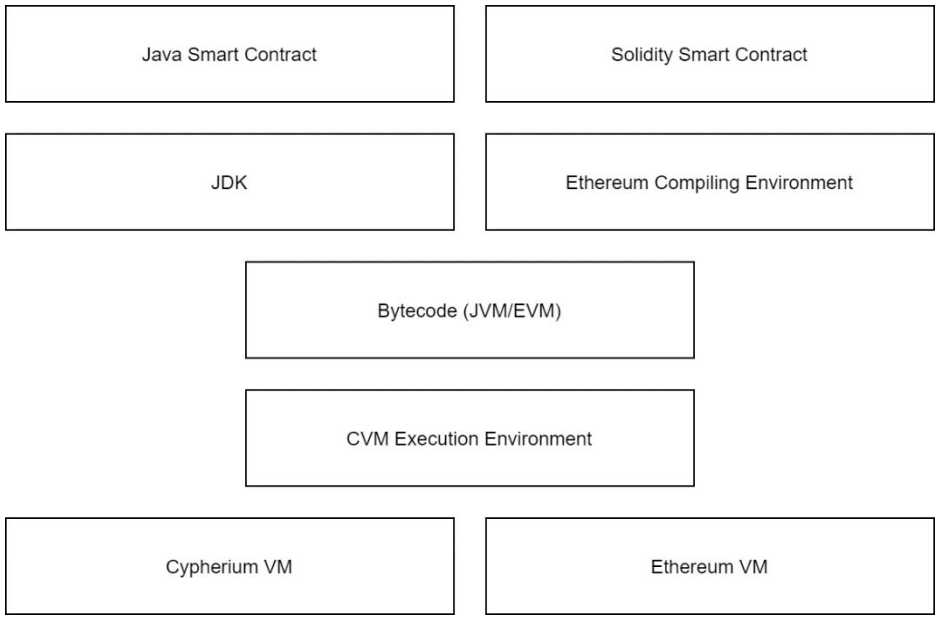


Fig. 14 Cypherium smart contract layer

Cross-chain support

For example, cannot connect with the SWIFT system, and . In order to establish a seamless transition from the current financial system landscape to a global digital asset network, blockchains must be able to interact with assets running on various legacy systems. To date, three types of cross-chain scheme exist:

Notary scheme

This method can be understood as a Witness mode of centralized or multi-signature functionality, where the witness is a legitimate user of chain A, responsible for monitoring the events and status of chain B, and then operating chain A. The essential feature here is that there is no need to pay attention to the structure and consensus characteristics of the cross-chain mechanism separate from the normal workings of each blockchain. Assuming that A and B cannot trust each other, this method simply introduces a third party that both A and B can trust together to act as a notary public. In this case, A and B can indirectly trust each other. This method does not propose a ledger in itself and does not seek any consensus. On the contrary, it provides a top-level encrypted escrow system called a "connector." With the help of this intermediary agency, different ledger systems can freely transfer money to each other through a third-party "connector" or "verifier." The ledger system does not need to trust the "connector" because the protocol uses a cryptographic algorithm to create funds for each of the two ledger systems with their own connectors. When all parties agree on the transaction, they can interact

with each other. The agreement removes the trust required by transaction participants, as the connector cannot lose or steal funds, which means that such transactions do not need to be protected by legal contracts and excessive review, which greatly reduces the thresholds for security and cost. Moreover, only the ledger systems involved can monitor the transaction, meaning that the details of the transactions can be hidden. The “validator” operates through an encryption algorithm, so it will not directly see the details of the transaction. Theoretically, the protocol can be compatible with any digital ledger system, and a bank’s existing ledger system can use the protocol with only minor changes. Thus, banks would be able to trade directly without a central counterparty or correspondent bank. The main representative projects of the notary scheme are: Interledger agreement, 0x agreement.

Hash-locking

Hash locking originated from the HTLC (Hashed TimeLock Contract) of the Lightning Network, and is now widely used. The process of its realization is as follows:

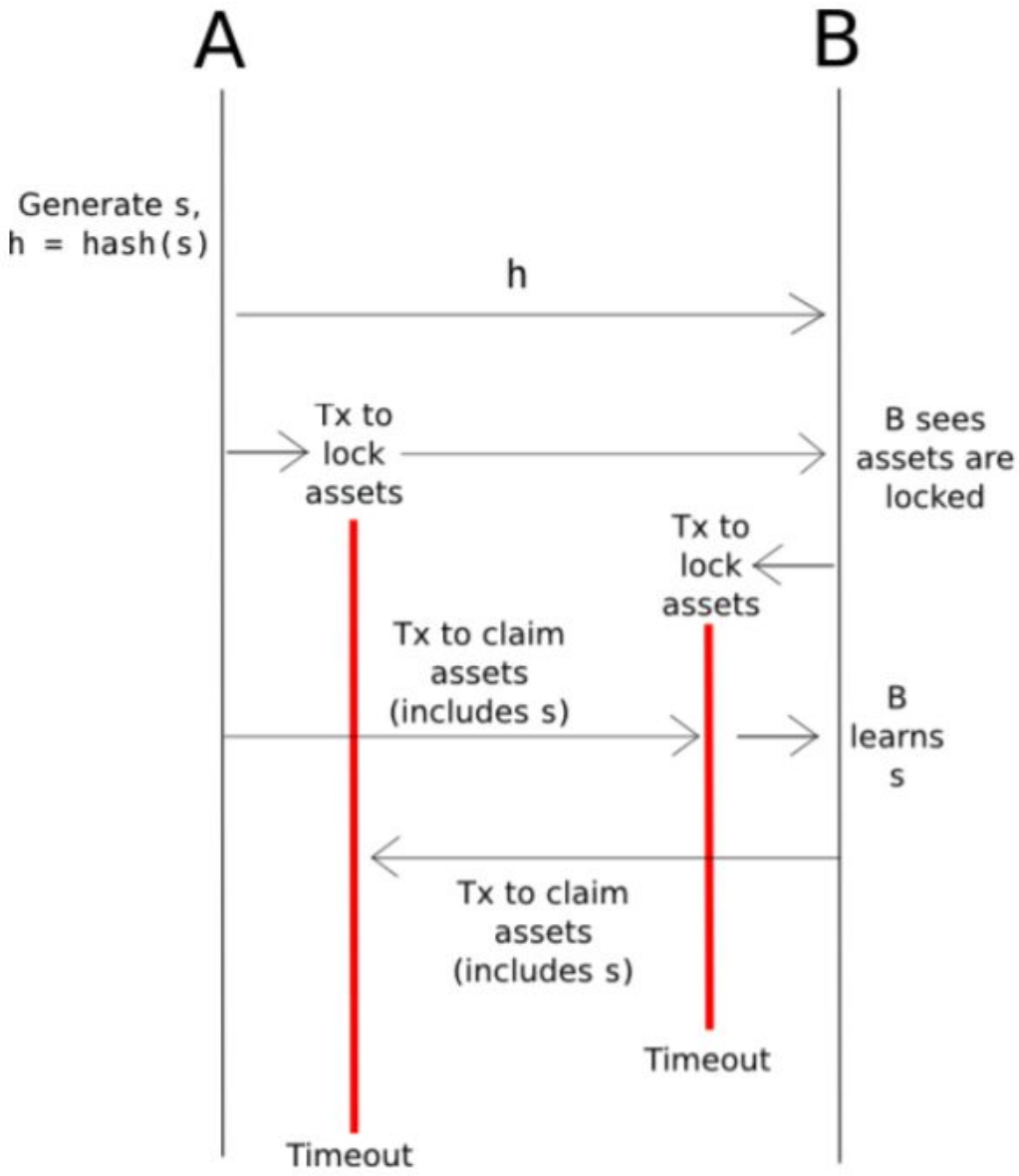


Fig. 15 Hash locking steps. Source: DREP

To walk through it step-by-step, one can use Hash-locking to realize an atomic exchange process of 20ETH and 1BTC, for example:

1. A generates a random number **s**, calculates **h** = hash (**s**), and sends **h** to B;

2. A generates HTLC, the overtime is set to: 2 hours, if B guesses the random number **s** within 2 hours, then take 1BTC, otherwise A retrieves 1BTC; here A locks the BTC contract with **h**, and B also has the same **h**. So A and B both have the same lock **h**, but A has the key **s**

3.B deploys a smart contract in Ethereum, if anyone can provide a random number **s** within 1 hour, so that the hash value is equal to **h**, that person can take away 20ETH from the smart contract;

4. A calls the smart contract deployed by B to provide the correct **s** and takes 20 ETH;

5. B knows **s**, there is still 1 hour left, and B can easily cash out 1BTC of A's HTLC.

Once it times out, the transaction fails and is atomic.

Note: The time parameter is introduced here mainly so that when the time expires, the current user can withdraw his coins. Otherwise, your own coins may be maliciously locked indefinitely.

Sidechain/relay

The side-chain system can read the events and status of the main chain and operate distinctly based on that information. That is, a side-chain system operates through SPV (Simple Payment Verification) and can verify the information of the header and Merkle tree of the block. The distinguishing feature here is that one must pay unique attention to the structure and consensus characteristics of the cross-chain mechanism. In general, the main chain is oblivious to the existence of the side chain, while the side chain depends entirely upon the existence of the main chain; similarly, the double chain does not know the existence of the relay, and the relay must know the two chains.

The sidechain is a blockchain based upon the anchoring of tokens on a certain original chain, just as fiat currencies would anchor gold (before Bretton Woods). The technology was originally designed to solve the scalability issues of first-generation blockchains. Each blockchain can implement mandatory consensus through a protocol. The performance of one such blockchain system can understand the consensus systems of other blockchains, and can automatically release Bitcoin after obtaining the locked transaction proof provided by other ledgers.

Comparison of various schemes

	Hash-locking	Notary	Sidechain / Relay
Interoperability	Cross dependency only	All	All (requires relays on all chains, otherwise only supports a single chain)
Trust model	Chain will not fail or 51% attack	Most notaries are honest	Chain will not fail or 51% attack
Suitable for cross-chain exchange	Supported	Supported	Supported
Applicable to cross-chain asset transfer	Unsupported	Supported (requires trusted long-term notary)	Supported
Applicable to Oracles	Not directly supported	Supported	Supported
Applicable to cross-chain asset mortgage	Mostly supported but difficult	Support (requires trusted long-term notary)	Supported
Difficulty of realization	Easy	Medium	Difficult
Practical application level	Mainly used in the Lightning Network	Ripple and more than 200 banks in the world	Basically no practical application, still in development

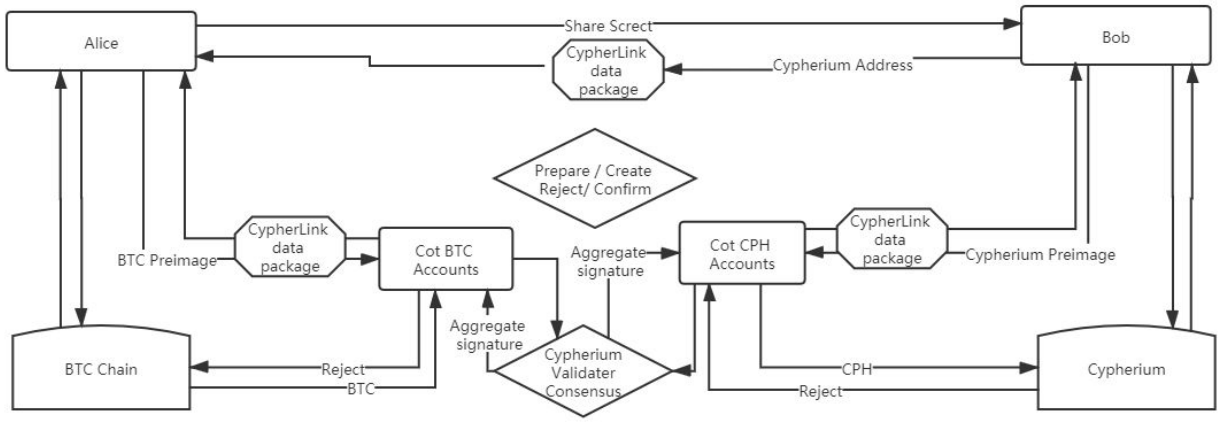
Cypherium Notary Connector

Cypherium introduces the Notary Connector Component to enable cross-chain transactions. The notary could be any trusted third party. Compared with many others and taking into account the characteristics of its particular network, Cypherium has developed a notary mechanism **CypherLink**, which is based on the InterLedger protocol, an open protocol initiated and led by the Ripple company. Any company or individual can participate in joint development and use it for free. It is not linked to any blockchain or XRP. Its original intention was to establish a global unified payment standard. It has been supported by

Microsoft and World Wide Web (W3C) since its inception and it has already accessed and traded among many different financial networks..

The notary is a component Cypherium dedicated to connection, focusing on the payment standard and unified protocol for connecting various ledgers. Its supporting communication objects include not only the blockchain but also various types of ledgers (usually the internal ledger systems of various banks), which are connected to each other through a trusted third-party “connector”. CypherLink protocol is an implementation model of hash-locking. In this system, two different ledger systems can freely transfer money between each other through third-party “connectors” or “verifiers.” The ledger system does not need to trust the “connector”, as the protocol uses cryptographic algorithms to create fund custody for the two ledger systems and connectors. When all parties agree on the amounts of funds, they can transact with each other across the ledger and the funds can flow immediately and automatically. And only participating parties in the ledger system can track the transaction; in other words, the transaction details can be hidden. The “validator” is operated by encryption algorithms, so, again, the transaction details will not be directly visible. This method is particularly suitable for existing banking systems. As a general rule, banks do not like to use other companies’ machines to verify their own transactions. In simple terms, they do not like public processing methods or semi-public processing methods, because these two methods may give outsiders opportunities for peeping at their internal data. This method alleviates that concern.

The following is a schematic diagram of architecture using CypherLink for Cypherium’s and BTC cross-chain transfer:



Use Case

Because Cypherium is a general purpose blockchain, it supports most common blockchain applications, such as issuance of customized tokens. We hereby only highlight use cases that are unique to Cypherium.

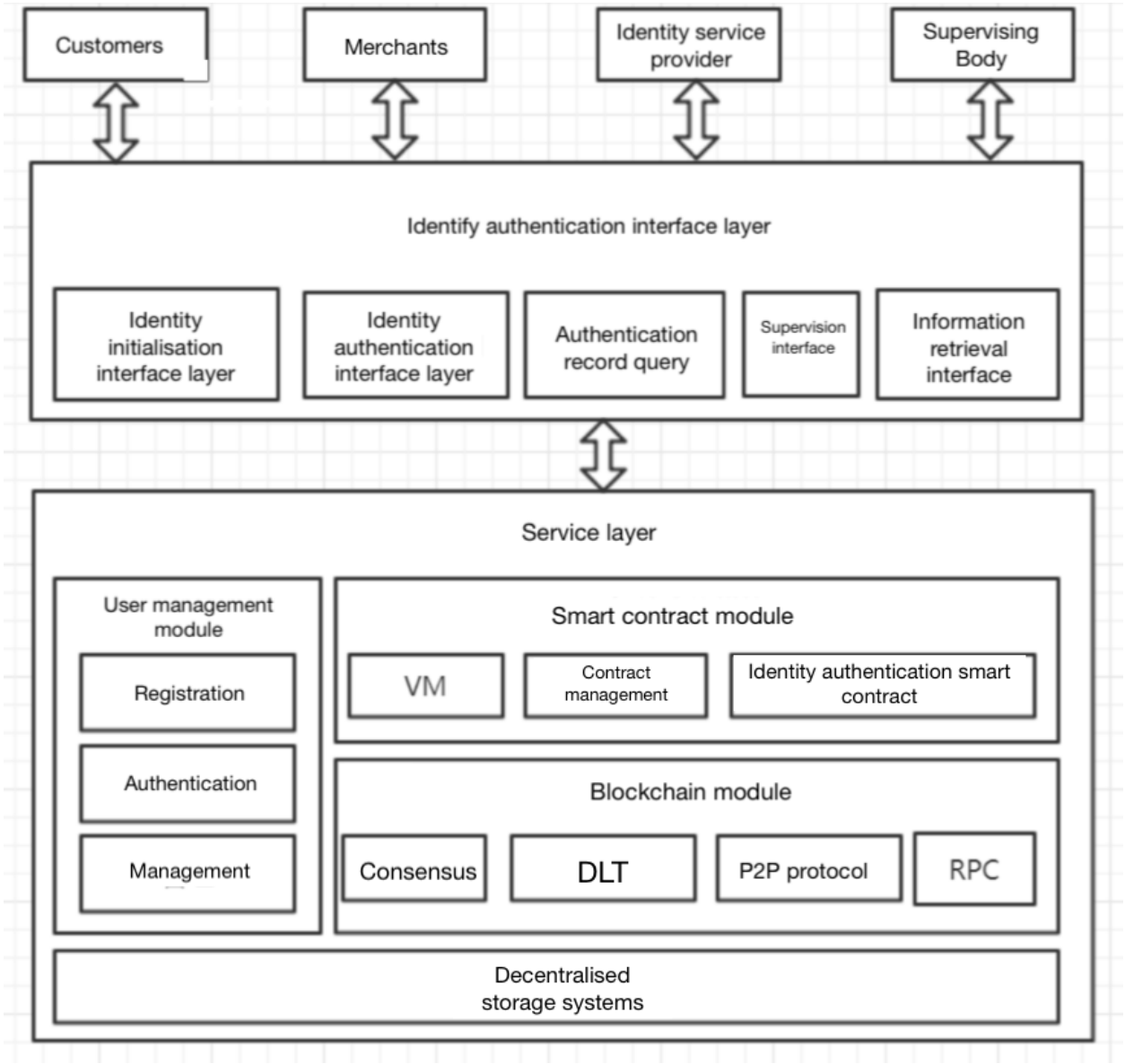
Digital identity

The protection of private data remains a big challenge in modern cybersecurity. Traditional centralized data servers have become easy targets of cyber criminals. In the past few years, massive data leaks of big enterprises including Facebook, Yahoo, Marriott and so on, have resulted in billions of user data records being stolen and insurmountable financial losses. As software and devices become more and

more complex, it is practically impossible to eliminate all security vulnerabilities in a system. A new form of identity must be implemented to prevent further data breaches.

Decentralized ledger technology handles identities via shared root of trust instead of centralized authority or a single point of failure. The Decentralized Identifiers (DIDs) is a standard facilitated by the Internet body World Wide Web Consortium (WC3) that allows users to own and manage their personal data. Cypherium can fully incorporate with open identity protocols including, but not limited to, the DID standard.

Cypherium ID solution overview:



Service layer provides a basic blockchain solution, including three types of logic structure: a blockchain service module, a smart contract service module, and an administrator management module. With

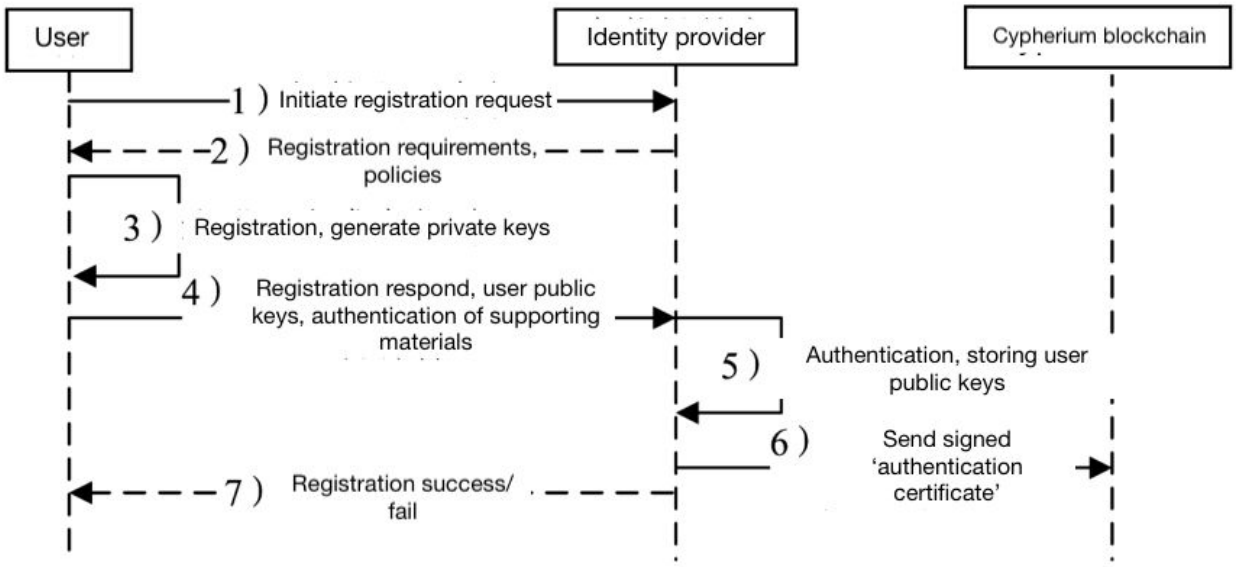
different timestamps and scenarios, different modules will be activated. For instance, the inclusion of new nodes will activate the registration function under the administrator-management module.

Interface layer provides the basic blockchain application interface in the upper layer. We have identified entities such as merchants, users, identity service providers, regulatory bodies so that the interface layer can provide basic identity authentication service. For instance, providing an authentication interface to merchants and users and providing supervision authentication to regulatory bodies, while at the same time allowing for the initial registration and distinguishment of identities.

The interface layer and service layer together create a trust model and provide basic blockchain services for external applications. This model will disrupt existing centralized identity management systems and at the same time take into account the needs of user privacy protection and supervision requirements by regulatory bodies.

We will then explain the process of registration and authentication below.

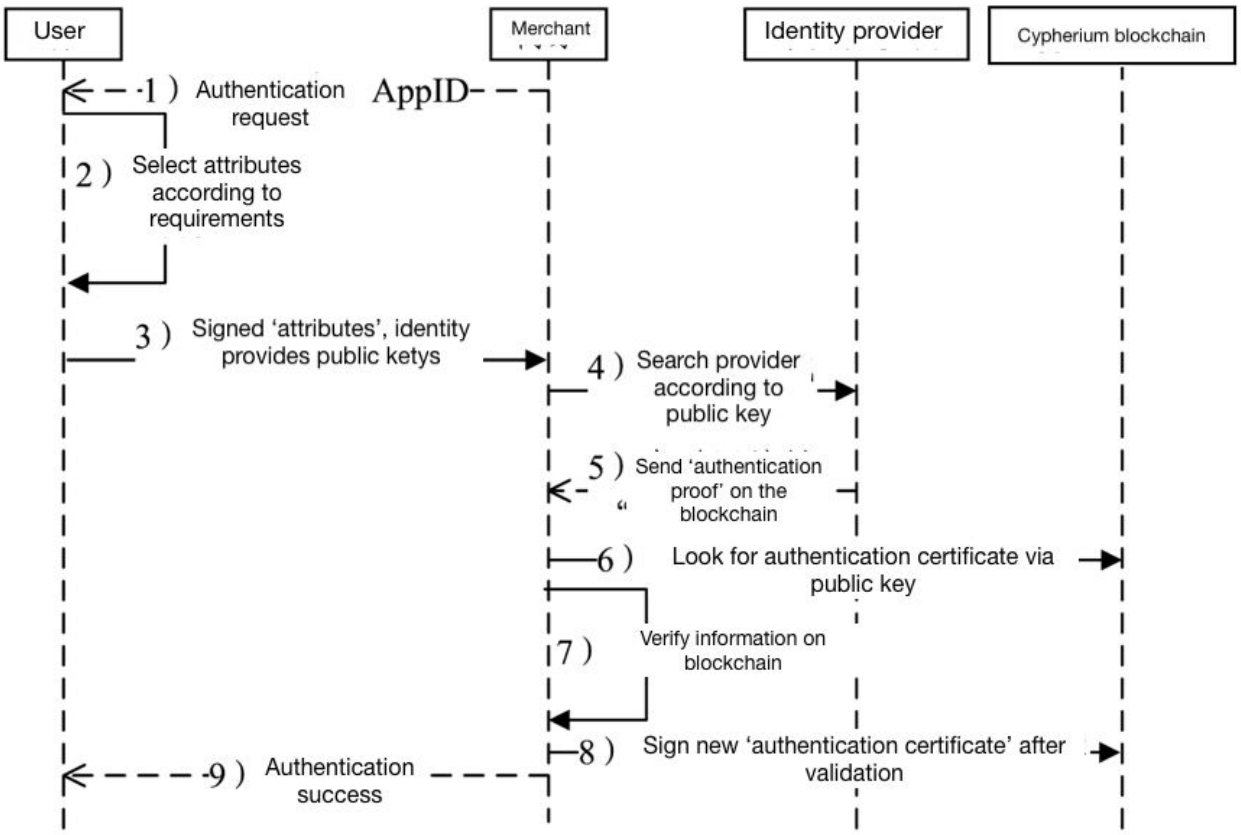
User registration process flowchart is as follows:



1. The identity service provider receives a registration request initiated by the user.
2. Identity service provider then selects user registration requests and sends back a set of policy requirements to the user.
3. A set of private and public keys will be created at the user's end and this set of private and public keys is unique to the user, identity service provider, and the blockchain.
4. According to the policy requirements, the user makes required selections, chooses a personal public key and sorts through other optional attributes, and sends it back to the identity service provider. At the same time, the user is also required to provide various supporting documents.
5. Identity service provider verifies the supporting documents provided by the user and stores the user's public key and related user profiles after documents are verified. User data is not stored in the local server but rather through hashing the attribute data and adding the signatory properties, in order to be 'authenticated'.

6. Authenticated data will be broadcast onto the blockchain and cryptographically stored.
7. Notice on user registration success will be sent.

User authentication process flowchart is as follows:



1. The merchant sends a randomized challenge to the user and requests the user to authenticate data requested per policy requirement.
2. According to the authentication request, the user selects the data attributes authenticated by the identity service provider from his end to fulfill policy requirements.
3. The user provides signatory, uses his public key to cryptographically hash attribute data requested by the merchant and provides a public key provided by the identity service provider and other related information to the merchant.
4. According to the information given by the identity service provider, merchant requests for the public key of the user, blockchain 'authentication' information and other related information.
5. Identity service provider responds to information requested by the merchant.
6. Merchant's end application automatically scouts the blockchain for 'authentication' information.
7. Merchant hashes the supporting data provided by the user and compares it to the blockchain 'authentication' proof publicly signed by the identity service provider in order to verify the validity of the 'authentication' information.
8. After verification success, user data is not stored locally but rather through encryption through hashing of valid data and adding of signatory properties, thereby creating new 'authentication proof' (with timestamps and other metadata) and sent to the blockchain for record keeping.

9. Notice on user authentication success will be sent.

Through identity registration and authentication protocol, Cypherium uses asymmetrical cryptography to ensure the security of the value transfer when users, merchants, identity service providers are exchanging information. The sender must use his private key to decrypt the data, then encrypt the data using the recipient's public key and send it over to the recipient. The recipient must authenticate the data using the sender's public key and finally uses his private key to decrypt the data.

Step 2 in the authentication process exhibits the user's control and permission to his own data. Trust being realized through the decentralized exchange of data through the blockchain is exhibited in step 8 in the process where the 'authentication proofs' signed and stored in the blockchain in the authentication by the current merchant can be used by other merchants when they want to authenticate the same user.

A protocol that considers all participant's registration and authentication needs is made possible due to the base infrastructure Cypherium provides.

Leveraging on the Cypherium blockchain service module which enhances the reliability and anti-attack capabilities of the system to realize a decentralized user data management.

Using Cypherium smart contract module, we are able to customize scenarios to create 'contracts' that fulfill the requirements of both parties. Therefore, using code to enforce conduct and law is not too distant from us anymore and it can better assure fairness. Administrator management module maintains the order of all participants and increases the scalability of the system.

Firstly, the existing mechanism of services based on trust is not mature. Based on market demand, interests between identity service providers and identity service users, we can integrate the strengths of Cypherium blockchain into existing trust services and identity authentication solutions to create a new trust model based on the blockchain and provide a unified trust model across the entire network. This will support common access among various identity service providers, providing various formats of identity authentication, as well as a fusion of authentication methods of the identity information source.

Next, all authentication is peer-to-peer, with user data being stored within the mobile phone terminal and blockchain only acts as an authenticator. Data management organizations can save hefty costs without having to maintain a centralized database because the validity of the authentication is both verified and maintained by participants of the network, therefore third-party trust agents have lost their value.

Under the Cypherium framework, the exchange of information between systems will not be interrupted by compatibility and mutually exclusivity issues which would result in high cost and difficulty in connection. As all systems use the same technology protocol, rules of authentication between participants will strictly follow the protocol consensus written into the blockchain and cannot be tampered with.

Lastly, programmability of Cypherium smart contract can allow complete automation of the authentication process customized based on different scenarios set by the management organization. Through embedded preset authentication rules programmed into the smart contract, the authentication

can be automatically completed if all the preset requirements are fulfilled. This increases user experience and work efficiency.

Central Bank Digital Currency

Countries around the world are paying more attention to the concept of Central Bank Digital Currencies (CBDCs), trying to capitalize on this clear and exciting trend, and China is no exception. In fact, China is eager to lead the charge with its DC/EP project. China has been researching and developing the idea of a CBDC as far back as 2014, and its national efforts to adopt digital technologies continue to grow.

CBDCs are positioned to revolutionize the global financial system. These technologies will serve as a new form of fiat currencies, but in their implementation, they will alter the very concept of currency and what it can mean to society. It will do through its five primary design principles:

1. Distributed Payments that eliminate unnecessary intermediaries.
2. Financial Inclusivity for the unbanked and underserved.
3. Efficient Cross-Border Payments for an increasingly globalized economy.
4. Digitized Management of Monetary Policy for more effective and immediate regulation.
5. New Framework to provide for the next generation of financial service innovation.

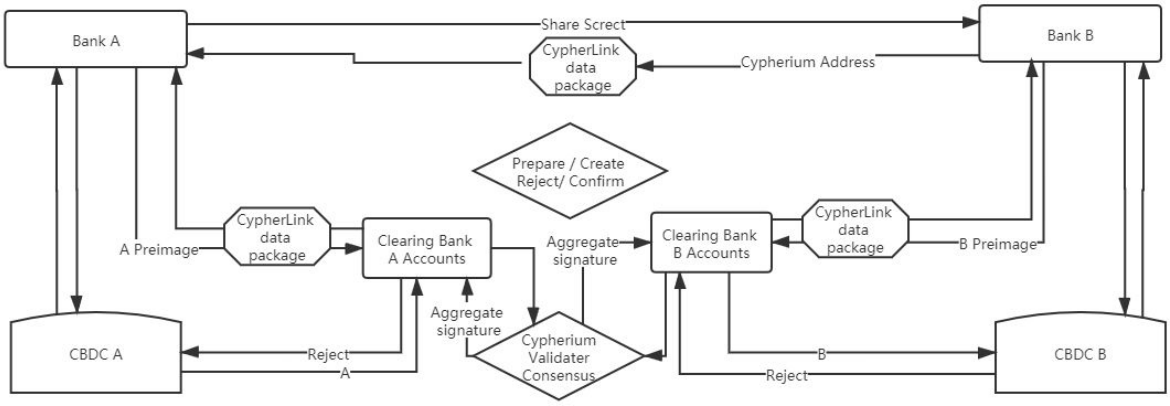
To achieve this, the People's Bank of China (PBoC) announced that its DC/EP project would take on a two-tiered operating architecture that would maximize resource utilization, foster collaboration, and avoid "disintermediation" or bottle-necking. Together these two tiers allow for consistent, safe financial service--the first-tier being, of course, the government-controlled PBoC, and the second, commercial banks.

The network itself is underpinned by three central authorities: the Certification Center, which manages the relationship between the customers' identities and their anonymous digital wallets; the Registration Center, which manages records of ownership and transference; and the Big Data Analysis Center, which monitors the entire digital currency environment to support the central bank's monetary policy and macro-prudential supervision.

Cypherium is a blockchain and smart contracting platform capable of fulfilling these technical requirements in order to meet the expectations and needs of the world's first operational CBDCs. Through a number of strategic design choices, Cypherium expresses the cutting-edge features of a blockchain without sacrificing the security and scalability required of a national currency. These features include bifurcated identity-authentication (for both practical anonymity and retrievable KYC necessary to combat criminality and terrorism) as well as crucial integration with business applications, legacy transaction systems, and parallel blockchain systems.

A blockchain system will need cross-chain operability to verify its own wallet and central bank digital currency. Through CypherLink, Cypherium can serve as a connector to any type of digital ledger, including RTGS, CBDCs, or other types of virtual currencies such as Bitcoin.

The architecture diagram is as follows, illustrated by an example of digital euro (DC / DE) to Chinese CBDC (DC / EP) shown below:



Parties: sender: Bank A; receiver: Bank B; and connector: Clearing bank.

Ledger relationship: Bank A has an account with CBDC DE (Digital Euro), Bank B has an account with CBDC DC/EP (Chinese Central Bank Digital Currency) , and the clearing bank has both DE and DC/EP accounts.

Scene: Bank A to send DE to complete the purchase of a laptop priced in DC/EP online, the price is 2000 DC/EP.

(1) . Bank A obtains a “shared password” provided by Bank B through encrypted communication means. The communication must be encrypted, so that after the communication, only Bank A and Bank B know the “shared password”; at the same time, Bank B will tell Bank A corresponding unique address in the CypherLink network, such as a52813334ee89485d661fee989a0e75402b2eeea.

(2) . Bank A inquires Clearing Bank about the transaction cost of sending 2,000 DC/EP in DE. At this time Clearing Bank will calculate the fee by real-time DC/EP and DE market rate, while Clearing Bank will overcharge one DE handling fee, and the final amount Bank A needs to pay Clearing Bank is 1001 DE.

(3) . Bank A generates the required CypherLink package according to the message format of the CypherLink specifics. The CypherLink package indicates that the target address is Clearing Bank. At the same time, a “conditional preimage” is generated based on the private content of the CypherLink package and the “shared password”. The preimage is hashed to obtain the “conditions” of a “custodial” transaction.

(4) . Bank A initiates a “custodial” creation operation on the DE ledger system, sets the “custodial” conditions in step 3 and a timeout period, and sets the ILP package at the same time.

(5) . Clearing Bank detects a “custodial” creation operation involving itself on the DE ledger.

(6) . Clearing Bank parses the ILP package, and figures out it should credit Bank B 2,000 DC/EP, and modify the CypherLink destination address to Bank B.

(7) . Clearing Bank initiates a “custodial” creation operation on the Cypherium ledger, sets the “custodial” conditions in step 3 and a timeout period, which is less than the timeout period in step 4, and sets the ILP package.

(8) . Bank B detected a “custodial” creation operation involving himself on Cypherium ledger.

(9) . Bank B parses the ILP package and uses his “shared password” and private content in the ILP package to generate a “conditional preimage” and corresponding “conditions”. By comparing whether the “custodial” creation operation’s “conditions” are the same as the “condition” created by Bank B, along with verifying whether the amount in the “custodial” transaction equals 2,000, the “custodial” transaction is either accepted or rejected. Here we assume the transaction is accepted.

(10) . Bank B launches a “custodial“ confirmation operation on Cypherium ledger, sets “conditional preimage”. Cypherium ledger’s “custodial” transaction is completed; Bank B receives 2,000 DC/EP.

(11) . Clearing Bank detects a “custodial” confirmation operation involving itself on the Cypherium ledger.

(12) . Clearing Bank analyzes the contents of the “custodial” confirmation operation and obtains the “conditional preimage”.

(13) . Cot initiates a “custodial” confirmation operation on the DE ledger, sets the “conditional preimage”, the “custodial” transaction on the Bitcoin ledger is completed, and Clearing Bank receives 1001 DC/EP.

Big Data Analysis

The status of the big data center is unique among the three centers of DC/EP, as it is charged with conducting anti-money laundering, anti-fraud, and other security monitoring. As the data aggregation office of the other two centers, the big data center also needs to analyze and govern all the data being collected and processed, in order to help policy formulation with the output of many indicators.

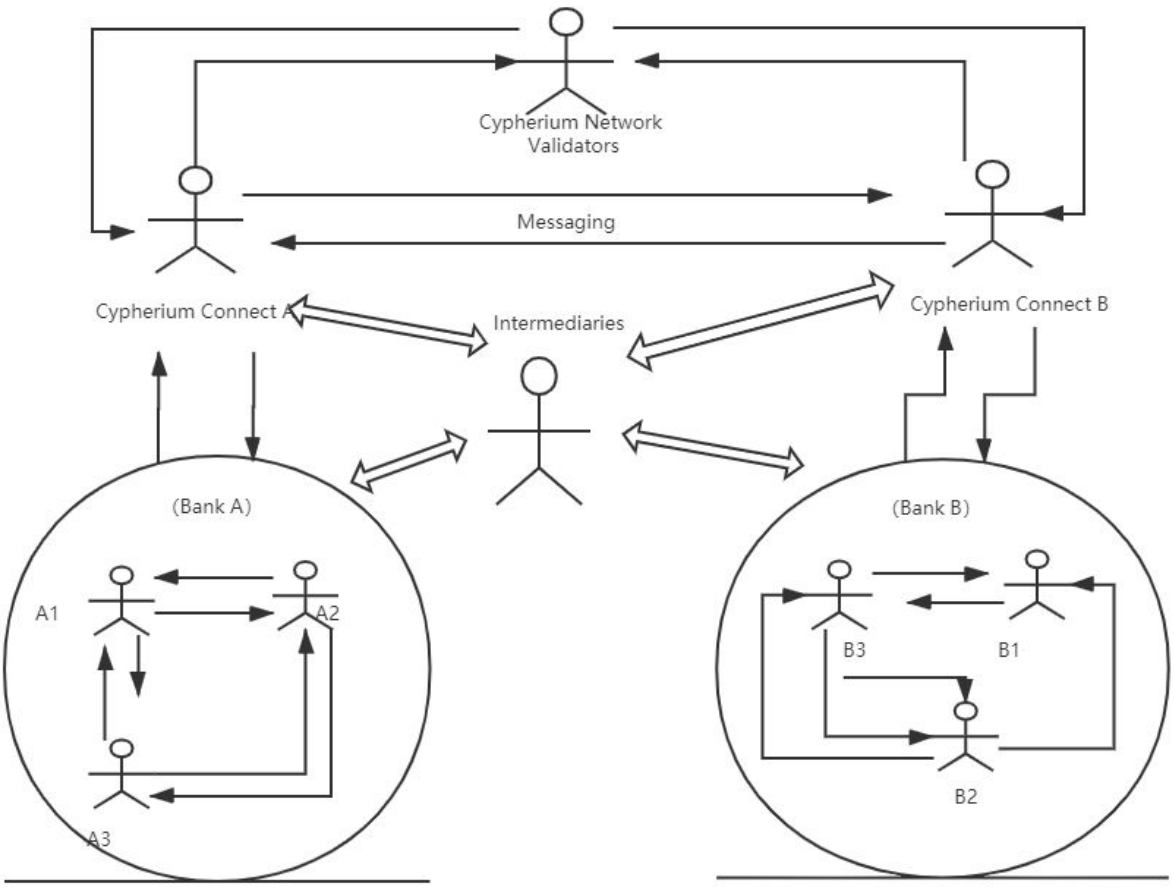
Due to this important office and sensitive nature of the big data center, in order to ensure transaction security and information security, the center must abide by a special mandate: for all data, the big data center can only read, not modify.

Cypherium reserves a monitoring interface for on-chain activities, and can promptly warn against illegal transactions through the big data center.

Clearing System

Considering that there may be problems in directly connecting to the central bank's digital currency system without special permission, Cypherium has also established a clearing system architecture, wherein intermediaries (which can be financial institutions with financial strength)

perform the final clearing of funds to ensure that user accounts have a relatively smooth experience. The software architecture is as follows:



The most important parts of the solution are Cypherium Connect and Cypherium Validator. The two core components are each described below.

Cypherium Connect

Cypherium Connect is a plug-in module that processes Cypherium payment transactions in the banking system (it's a bit similar to the payment front-end system). Between the remittance bank and the receiving bank, Cypherium Connect has established an information channel for exchanging KYC / AML, risk control information, handling fees, exchange rates, and other payment-related information. In order to attract banks to join, Cypherium has created adaptive designs on KYC / AML that can be personalized by both banks. Before the transaction is initiated, Cypherium Connect sends this information to the counterparty of the transaction. Only by confirming OK can one execute the transaction and clear the funds.

Cypherium Validator

Cypherium Validator is a verification machine. Before the transaction enters the Central Bank and blockchain ledger system, it must be confirmed by Validation. This machine has a strict

authentication mechanism, and its verification rules can be customized according to specific system requirements.

Cypherium also introduces the scalable collective signature scheme CoSi. All communication parties must jointly sign the key information received and sent. Without increasing communication overhead, the system may always verify the correctness of the communication messages of all parties in the clearing system.

The Cypherium chain uses the BFT + BLS consensus model to solve double-spending, and the ledger nodes also perform additional version checks to ensure data integrity. There is also a queue manager in the system, which packages signed transactions into blocks and broadcasts each block to all participant nodes in the same channel. After receiving the broadcast, the nodes in the channel will verify the transaction again, and then update the transaction block to the ledger.

In order to ensure the privacy of transactions, a two-way channel must be established between every two banks in the system. In each channel, the bank must establish a channel account and allocate appropriate funds in the channel account to ensure that the transaction can be completed. Each transaction must be signed by the two banks in the channel to be verified.

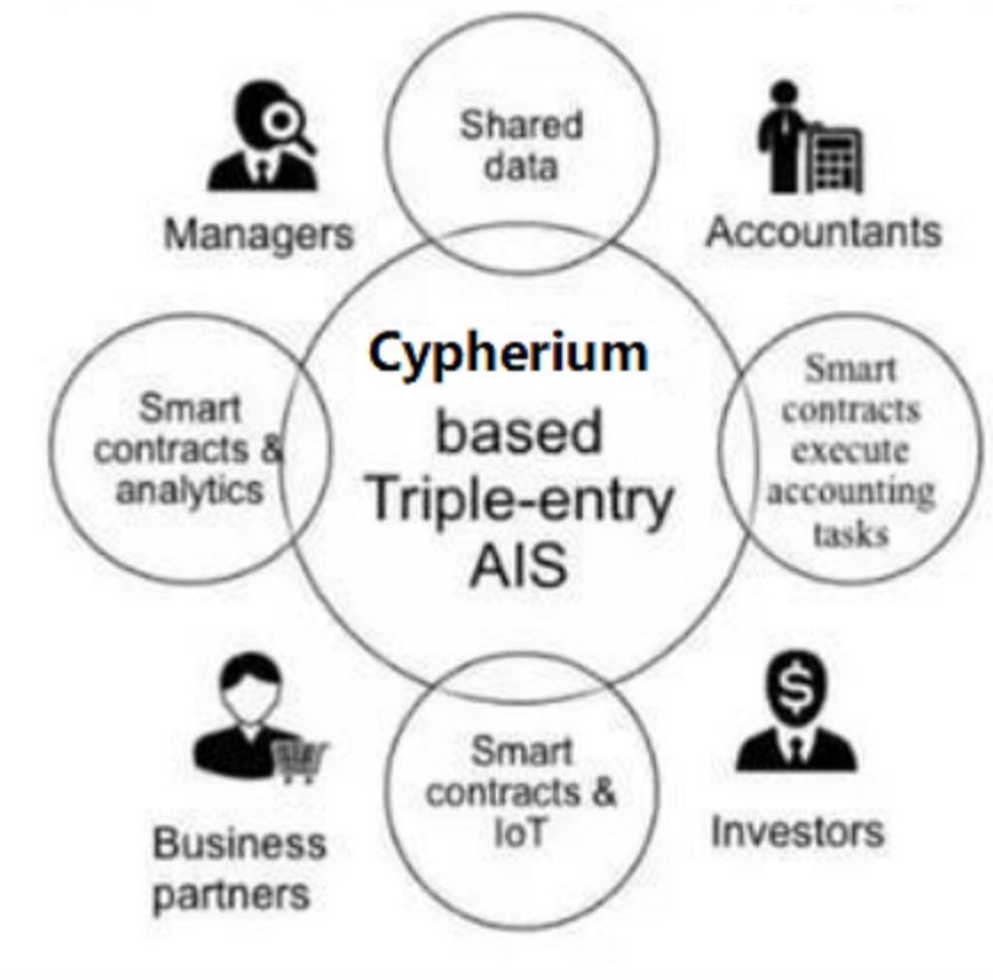
Triple-entry bookkeeping

Issues of accounting have plagued us for the whole of human history. After thousands of years of human commercial activities, the double-entry bookkeeping system emerged around the 13th century among the merchants in Venice, Italy and remains the most widely used accounting method for businesses today. In this system, every financial transaction gets a debit entry and a credit entry. The total debits and the total credits should always be equal. The double-entry system can detect common errors. However, the system depends on the trustworthiness of the accountant, and the ledger can be gamed to defraud external viewers.

Triple-entry bookkeeping improves the double-entry bookkeeping by applying all accounting entries with a cryptographic third entry, making them permanently recorded without a trusted third party.

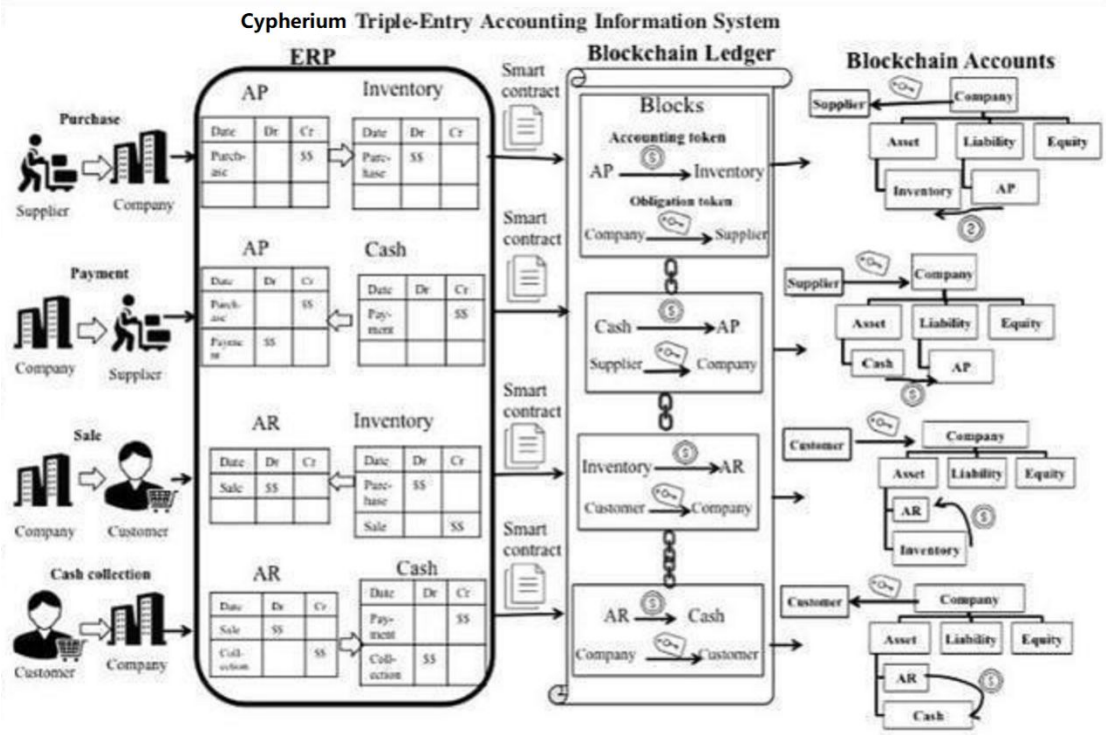
Cypherium blockchain distributes the power of transaction verification, storage, and management in the accounting information system among a network of computers to prevent any unauthorized data changes, while integrating other emerging technologies (such as the Internet of Things). This enables the system to meet the needs of today's businesses, such as , real-time tracking and monitoring of activities, and automatically recording and measuring operational performance. Also, by smart contract's predetermined rules, businesses can automate business processes, further implementing automatic control of business execution functions.

Compared with the traditional ERP system, this method bears the advantages of decentralized and safe data storage, and resistance to tampering and manipulation. It helps to report reliable accounting information to stakeholders (such as managers, auditors, creditors, shareholders) in near real-time according to different levels of aggregation. In the Cypherium system, management, accountants, business partners and investors can actively collaborate to verify transactions and provide reliable evidence for cross-validation. These components combine to form a real-time, verifiable and transparent accounting ecosystem, as illustrated below:



The triple-entry bookkeeping system initially required transaction processing authorization from a neutral intermediary agency. Each party (the two parties involved in the transaction and the intermediary agency) created a record for the transaction with a total of three accounts. Cypherium blockchain can now play the intermediary role through automatic distribution, automatic storage, and intelligent verification mechanisms. Once the accounting entries are confirmed and added to the chain, it is difficult to modify or destroy them. The smart contract technology of Cypherium blockchain can follow its accounting guidelines or pre-specified business rules to quickly verify transaction records. By encoding a third accounting entry into the blockchain, a transparent, encrypted, secure, and self-validating accounting information system can be generated, thereby facilitating reliable data sharing between business parties and continuous reporting to shareholders and interested parties.

Cypherium's simplified triple-entry bookkeeping information system architecture follows:



The figure above uses a simple purchase-and-sale business cycle as an example to demonstrate the workflow of the system. When a company purchases goods on credit from its suppliers, it records accounts payable and inventory in its ERP system. It will simultaneously submit this matter to the blockchain ledger in the form of digital transactions transferred between the two blockchain accounts.

Accounting documents in the Cypherium chain can simply be regarded as symbols for recording and tracking purposes. Each account in the modern double-entry accounting system will have a corresponding Cypherium account. The account will form a hierarchical structure, focusing the accounting records on three levels: the personal account at the bottom; the total assets, liabilities, and equity in the middle; and the overall company at the top. This structure can use smart contracts to automatically confirm the balance sheet equation. For example, if the balance in the company account is set to the balance in the asset account minus the total balance in the liability and equity accounts, it can create a smart contract to monitor the balance in the company account and issue an alert when the balance is not equal to zero. Another benefit of this account hierarchy is that it allows data to be viewed at different levels. Different information users have different requirements and restrictions on accounting data collection, so different data views may be granted according to corresponding user roles.

Although the verification process will be automated by Cypherium blockchain technology, the process will strictly abide by role management, such as accountants, management, auditors, etc. Therefore, the blockchain ledger in this scenario belongs to the category of permissioned chain. Each party will play a specific role in the verification process, and actions and concerns may be resolved differently. For example, if the auditor suspects a transaction, the transaction may be suspended for confirmation by the accountant, and the CFO may decide to cancel the transaction altogether. Cypherium will conduct relevant authorization and transaction verification based on the responsibilities

undertaken by different roles, and automatically aggregate their signatures to a verifiable BLS signature and form a consensus record.

In order to protect the privacy of the company's sensitive data, transactions can be encrypted before uploading to the blockchain ledger, and only users with decryption keys can view the transaction content.

Cypherium smart contracts can effectively control the recording of accounting activities by enforcing accounting rules, and therefore, they can provide automatic guarantees for the processes of transaction posting, classification, and termination. This contracting system can also be combined with IoT technology, which can capture the actual situation and activities of physical objects to automate the bookkeeping process. For example, if an inventory item is known to have left the company based on its geographic information transmitted through the Internet of Things, smart contracts can be executed to publish sales records to the blockchain ledger. Cypherium smart contracts implement functional componentization, and each function can form its own independent Java class file, which saves the storage space of the smart contract and makes the actual application easier.

Conclusion and Future Works

This whitepaper has introduced the reader to Cypherium, a blockchain and smart contracting platform. Cypherium makes a number of advancements in decentralized ledger technology. Firstly, its proprietary consensus mechanism offers a unique solution to the so-called “blockchain trilemma” of scaling, security, and speed with a strong emphasis on applicability. While other third-generation blockchains have abandoned Proof-of-Work, the fundamental innovation of Bitcoin’s Nakamoto Consensus, Cypherium marries this older methodology to a more current cutting-edge technology, the HotStuff algorithm employed by Calibra. As such, Cypherium is the first public and permissionless HotStuff-based blockchain. Cypherium achieves this by rethinking the role of mining, breaking the process down into two component parts--minting new coins and verifying transactions--and supplying each with its own blockchain. This novel consensus architecture, called CyberBFT, improves upon the work of many established projects in the space including Bitcoin, ByzCoin, Bitcoin-NG, and others.

The reader has also encountered the Cypherium Virtual Machine (CVM), a Java-based virtual runtime environment for the execution of smart contracts on the Cypherium network. Just as CypherBFT emerges as a solution from a careful analysis of the pain-points of prior decentralized technologies, the CVM offers original solutions to distributed computing’s most persistent problems. Chief among these attributes is the CVM’s Java compatibility, which grants the Cypherium network access to the world’s largest pool of legacy development and computing infrastructure. This serves as a necessary access point between the legacy web and a decentralized future. Other key features of the CVM with which the reader has become acquainted include hierarchical calculation, native 64-bit integer support, fixed-point representation, increased security, compatibility with other VMs including Ethereum’s EVM, atomic swap support, IPFS docking, and more. The CVM is a key innovation that allows the Cypherium network both to support everyday enterprise and economic transactions and to provide a framework for the developments of meaningful, “killer” decentralized applications.

One of the most notable uses of Cypherium, as a highly scalable blockchain system, is its use as an interbank intermediary. Through tools like CypherLink (a notary mechanism based on the InterLedger protocol), Cypherium Connect (a third-party plug-in module for banking systems), and Cypherium Validator (a verification machine), our network can link any two banking systems; Cypherium can support cross-chain transactions among any two digital currencies. This is a vital feature of our network, particularly given the increasing promise and prominence of Central Bank Digital Currencies (CBDCs). Cypherium's architecture makes it an ideal technology to connect all kinds of data systems and enterprises, both public and private, as industries across the board turn to decentralization. We at the Cypherium network believe this on-chain migration will be a large part of the economic and technological landscape of the next five to ten years, and Cypherium provides the support necessary in this period of change.

In the coming months and years, Cypherium plans to adopt several additional features that will become necessary for the network to mature. These include horizontal scaling, increased privacy protections, and smart contract standard libraries. Horizontal scaling, or sharding, is a technique that will help our blockchain scale. In brief, sharding will allow nodes to process and store only a fraction of their data on-chain, with reference to the rest, improving the efficiency of the network. The Cypherium developers also have plans to incorporate Zero-Knowledge proof privacy, which in practice allows nodes to communicate about the existence of information without divulging the content of information. This will bring the privacy of our public network as it becomes increasingly overrun with sensitive information. And finally, Cypherium is continuing to develop smart contract libraries for the various contracting languages it supports. These standard libraries--repositories that maintain coding definitions for commonly used algorithms, data structures, and mechanisms--will play an important role in the standardization of our nascent industry. They will also foster a common understanding among our devs, which will be an absolutely necessary foundation for the development of future apps and enterprises in an open system like Cypherium's public network.

We are grateful for the interest and support of our community, partners, and advisors, and above all, we are excited to bring our technology to the world.