(12) INNOVATION PATENT
(19) AUSTRALIAN PATENT OFFICE

(11) Application No. **AU 2016101976 A4**

(54) Title
Open Network of Permissioned Ledgers

(51) International Patent Classification(s)
G06F 17/00 (2006.01)

(21) Application No: **2016101976** (22) Date of Filing: **2016.11.11**

(45) Publication Date: **2016.12.08**
(45) Publication Journal Date: **2016.12.08**
(45) Granted Journal Date: **2016.12.08**

(71) Applicant(s)
Ivan Klianev

(72) Inventor(s)
Klianev, Ivan

(74) Agent / Attorney
Ivan Klianev, PO Box 324, Brighton-Le-Sands, NSW, 2216

Open Network of Permissioned Ledgers

Abstract

A permissioned ledger with multiple distributed replicas is designed and implemented to circumvent CAP theorem and the FLP result. Each replica implements highly available mufti-partition transactions with scaled out data operations that do not degrade throughput and latency. The ledger implements a fully asynchronous leaderless Byzantine consensus algorithm. Multiple such ledgers are integrated in a federation with open membership and no central administration where safety and liveness of distributed transactions that modify state of two ledgers are guaranteed regardless of anything that may go wrong within the tolerated limit per ledger participating in chain from sending account to recipient account.

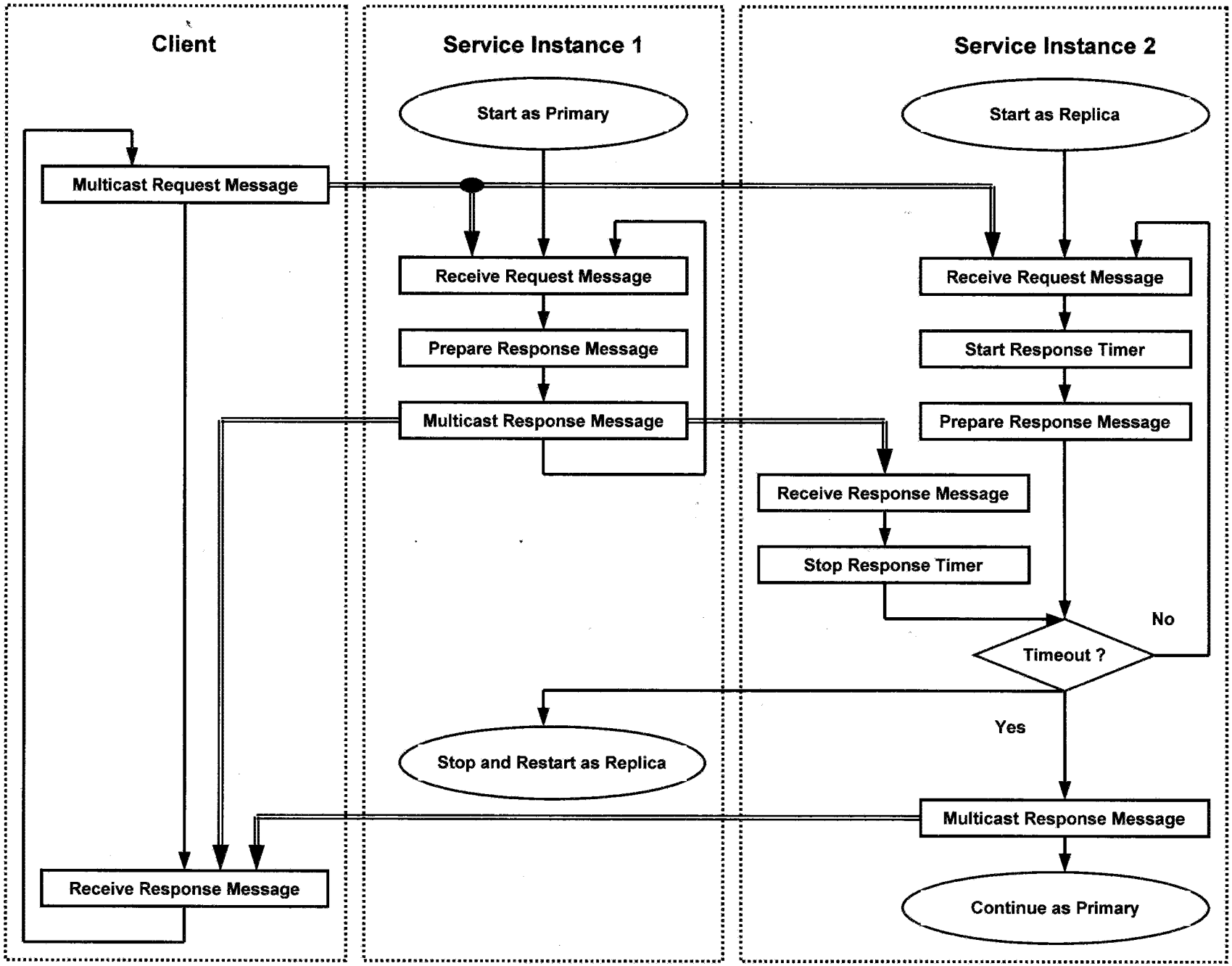
Client
Service Instance 1
Service Instance 2
Start as Primary
Start as Replica
Multicast Request Message
Receive Request Message
Receive Request Message
Prepare Response Message
Start Response Timer
Multicast Response Message
Prepare Response Message
Receive Response Message
Stop Response Timer
No
Timeout ?
Yes
Stop and Restart as Replica
Multicast Response Message
Receive Response Message
Continue as Primary

Fig. 2

Open Network of Permissioned Ledgers

Introduction

The Internet made communicating and sharing information a widely accessible privilege. As a tool for distribution of data, it served the purpose of informational democracy in an extremely efficient manner. Efficiency, however, was not a characteristic of using the Internet for business involving transfer of digital assets, such as money in bank account. Sending data is in fact sending a copy of it, not the original. As a significant consequence of that, a number of intermediaries are needed to perform multiple roles just to ensure that sending a digital asset is not a sending of its copy.

The Internet of Value Manifesto[1] outlines a vision of a Web where value is exchanged as freely and easily as information. It acknowledges that: there are many value networks and there always will be; and there are many ways to connect value networks. The Manifesto proposes six general principles to follow while pursuing the Internet of Value vision: 1. Open and accessible - nobody is arbitrarily denied access. 2. Built on secure and resilient systems. 3. Simultaneously private and transparent - while privacy is fundamental right of participants, transparency is needed when moving value between different jurisdictions. 4. No single entity is in control. 5. Built on open standards. 6. Simple and extensible. The document recognizes that much work needs to be done before the objective is achieved as the existing systems for exchange of value are proprietary, disconnected, opaque, and built with lack of common standards.

Since the effective end of the Bretton Woods system[2] of monetary management in August 1971 and the following move to fiat-based currency[3], the vast majority of money in the world are nothing more than an entry in a ledger. It is made by central banks when they create reserves and by commercial banks when they extend credit. It lives in digital ledgers inside databases spread through the financial system. Therefore, the fiat money[4] is nothing more than a promise and movement of funds is simply an adjustment of databases to reflect the changes in promises.

Two-phase-commit[5] protocol is a distributed algorithm that coordinates the database management systems[6] participating in a distributed atomic transaction[7] on whether to commit[8] or roll back[9]. The protocol is a perfect fit for double entry accounting transactions involving two separate database systems, but it is not resilient to all possible failures and thus cannot provide guaranteed reconciliation. It assumes that: participating nodes do not experience stop-faults or Byzantine[10] faults, data in the write-ahead log[11] is never lost or corrupted, and nodes can freely

[1] https://www.w3.org/Payments/IG/wiki/Internet_of_Value_Manifesto
[2] https://en.wikipedia.org/wiki/Bretton_Woods_system
[3] https://en.wiktionary.org/wiki/fiat_currency#English
[4] https://en.wikipedia.org/wiki/Fiat_money
[5] https://en.wikipedia.org/wiki/Two-phase_commit_protocol
[6] https://en.wikipedia.org/wiki/Database
[7] https://en.wikipedia.org/wiki/Distributed_transaction
[8] https://en.wikipedia.org/wiki/Commit_(data_management)
[9] https://en.wikipedia.org/wiki/Rollback_(data_management)
[10] https://en.wikipedia.org/wiki/Byzantine_fault_tolerance
[11] https://en.wikipedia.org/wiki/Write-ahead_logging

communicate with each other. Therefore, when one of those assumptions turned out incorrect, a human intervention may be needed to remedy the consequences resulting from the failure.

Execution of distributed double entry accounting transactions between two database systems with Two-phase-commit protocol over the Internet became a reality since its commercial use started. A Transaction-Internet[12] protocol was designed to make the Two-phase-commit protocol more secure when used over the Internet by combining it with Secure Socket Layer[13] (Transport Layer Security[14]) for public key encryption and authentication. Making it more secure, however, does not solve its inherent safety[15] and liveness[16] issues accelerated by the fact that in the Internet environment transaction participants cannot be expected to always cooperate.

To illustrate a possible safety issue with Two-phase-commit protocol, a stop-faulty node during the commit phase of the protocol leaves the system in inconsistent state. For example, a paying node reduces the content of the payer account but a stop-faulty payee node fails to increase the content of the payee account. Alternatively, a payee node increases the content of the payee account but a stop-faulty or a Byzantine-faulty paying node fails to reduce the content of the payer account. Prevention of flooding the financial system with non-existing money requires complex reconciliation procedures.

To illustrate a possible liveness issue, a lost connection or a stop-faulty node may prevent terminating a transaction coordinated with Two-phase-commit protocol, or a transaction manager could fake failure with an objective to block other transaction managers. It would be difficult for the blocked participants to differentiate between genuine and fake failures. These participants typically hold locks on resources of the involved accounts while being blocked, thereby placing the account owners in a state of commercial disadvantage.

Bitcoin[17] blockchain[18] technology is an alternative to globally distributed transactions coordinated with a two-phase-commit. The blockchain achieves consistency[19] in the system with locally executed but globally replicated transactions and guarantees atomic consistency (linearizability[20]) between all replicas with a mechanism employing the self interest of participants. As a by product, it provides a degree of immutability[21] of records resulting from the committed transactions and does not require a human intervention, administration, or reconciliation. Bank of England described the technology as a first attempt at an 'Internet of finance'[22] as it operates in an entirely decentralized way, without intermediaries such as banks. Deutsche Bank sees the blockchain technologies as a solution capable to guard against risks associated with IT systems failure[23].

---

[12] https://www.w3.org/2001/03/WSWS-popa/paper01
[13] https://en.wikipedia.org/wiki/SSL
[14] https://en.wikipedia.org/wiki/Transport_Layer_Security
[15] https://en.wikipedia.org/wiki/Safety_(distributed_computing)
[16] https://en.wikipedia.org/wiki/Liveness
[17] https://en.wikipedia.org/wiki/Bitcoin
[18] https://en.wikipedia.org/wiki/Blockchain_(database)
[19] https://en.wikipedia.org/wiki/Consistency_(database_systems)
[20] https://en.wikipedia.org/wiki/Linearizability
[21] https://en.wikipedia.org/wiki/Immutable_object
[22] bankofengland.co.uk/publications/Documents/quarterlybulletin/2014/qb14q3digitalcurrenciesbitcoin1.pdf
[23] https://www.db.com/newsroom_news/Deutsche_Bank_Investor_Report.pdf

Money in a blockchain is like cash, but with significant improvements[24]. Blockchain-based currencies cannot be counterfeited and can be easily traced. Blockchains function by being records of the movements of each coin from the moment it is created. Since the record is public, it is easy to ensure that each coin is only assigned to one account. Yet, these qualities were facilitated by the Bitcoin blockchain with a questionable engineering achievement. It eliminates the cheap talk[25] and circumvents CAP theorem[26], but burns electricity like a small nation[27] just to provide utility with a capacity much below the needs of that nation.

A blockchain is a single monolithic ledger for all accounts of all users, maintained by all replica nodes. It could be a proper design choice for a ledger with small number of accounts or infrequent use, otherwise, it is hardly effective or efficient. The global replication requires messaging across all continents, even for a transfer of $5 between two adjacent towns. Synchronization of multiple copies of the ledgers incurs significant costs even when all copies can be trusted as accurate and their operations are honest. The costs arise much higher when the participants cannot trust each other thereby making the social costs prohibitively high.
Blockchains operate using state machine replication[28]. Every replica must execute the transaction requests received on all replicas under an agreed common sequential order. State machine[29] is an abstract device that executes a command and produces an outcome. Starting from the same initial state, state machines executing the same command finish with the same final state. In the same manner, replicas starting from atomically consistent state and executing the same sequence of transactions finish with atomically consistent state. Agreement about the common sequential order is essential for efficient replication, especially over the Internet.

Where a network partition splits a distributed system, the CAP theorem states that if the system needs to be available to continue processing requests it has to give up the atomic consistency of replicas since there is no way to achieve an agreement about a common sequential order of execution of all transaction requests received on all replicas. Alternatively, if the system needs to preserve the atomic consistency, it has to give up the availability and stop the processing of requests. Bitcoin in effect circumvents the CAP theorem. It doesn't really care about network partitions as long as the whole world is not split in two parts that cannot communicate between themselves, which is practically impossible with the multiple alternative routs of the Internet.

In a system where anyone can freely enroll, a simple vote is not enough for an agreement since the communication between participants constitutes a cheap talk: 1) the cost to construct and delivery a message is effectively zero. 2) messages are not binding as the participants can drop out and later freely reenroll. 3) the validity of a transaction is not fully verifiable without access to all copies of the ledger. To addresses the cheap talk problem, Bitcoin stipulates that to gain the right to propose an addition to the ledger any participant must prove that it has done costly work.

[24] Building the Trust Engine. UBS Group Technology White Paper 2016
[25] https://en.wikipedia.org/wiki/Cheap_talk
[26] https://en.wikipedia.org/wiki/CAP_theorem
[27] https://karlodwyer.github.io/publications/pdf/bitcoin_KJOD_2014.pdf
[28] https://en.wikipedia.org/wiki/State_machine_replication
[29] https://en.wikipedia.org/wiki/Finite-state_machine

Instead of aiming for an impossible agreement, Bitcoin motivates the participants to compete against each other for the so called mining money[30] by performing CPU-intensive computations in searching for a cryptographic proof of work[31] and requires each participant to accept the sequential order of a group of requests received by the participant that was first to solve the proof of work. The sequential order on the winning computer becomes binding order for all other participants. The burnt energy and the limited transactional throughput are price for the eliminated cheap talk and for the achieved common sequential order.

The proof of work system allows a participant to prove the paid cost without a need to disclose identity. This allows cryptocurrencies to maintain a completely decentralized network with free entry and exit of participants but decreases the ledger's total throughput capacity. At present, Bitcoin is limited to between 7 and 10 transactions per second and it takes an hour before a transaction can be trusted. This is the price to run a decentralized database with availability, atomically consistent replicas on the majority of open membership computers, and tolerance to coordinated malicious attempts of less than half of members.

State machine replication needs a deterministic database[32] operations. Bitcoin utilizes Google's open source NoSQL data engine LevelDB[33], which doesn't have a transaction manager, but being a permissionless blockchain Bitcoin does not really need one. The low throughput makes the sequential execution of requests an affordable luxury, which makes any database perform deterministically. This is paid with some increase of latency, but with no decline of throughput.

Digital currencies aspiring for long-term success based on higher social impact need more efficient agreement solutions. Efficiency requires giving up the proof of work, which enforces giving up the permissionless networks. Ripple[34] and Stellar[35] exemplify notable endeavors to accomplish an efficient in terms of throughput agreement solution with a permissioned network, which does not restrict the number of participants. Both Ripple and Stellar, however, emulate the Bitcoin blockchain approach of using a ledger with monolithic central state. Guaranteeing safety, unrestricted growth, and decentralization are mutually exclusive objectives where a ledger with monolithic central state is used. Yet an Internet-of-Value[36] needs even more than that.

Today money resides in databases run by financial institutions[37]. In the future Internet-of-Value, money is a record in a ledger, copies of which exist on multiple computers and since the records of a user are secured by a key only the user knows, the money of a user resides with the user. Once users could keep their own accounts, banks might not be necessary. If some users no longer have accounts at a specific bank, then these users are no longer obliged to use that bank for their transactions.

---

[30] https://www.genesis-mining.com/
[31] https://en.bitcoin.it/wiki/Proof_of_work
[32] http://www.slideshare.net/abadid/the-power-of-determinism-in-database-systems
[33] http://leveldb.org/
[34] https://en.wikipedia.org/wiki/Ripple_(company)
[35] https://en.wikipedia.org/wiki/Stellar_(payment_network)
[36] https://ripple.com/wp-content/uploads/2016/08/Toward-the-Internet-of-Value.pdf
[37] Building the Trust Engine. UBS Group Technology White Paper 2016

Essential utility of the future Internet-of-Value is the ability to transfer funds with email speed across a billion of frequently used accounts around the globe. Interoperability across digital ledgers is a necessary element of a solution, but not the solution. Safety of an interoperability-based solution depends on liveness of involved ledgers and the weakest link may destroy it. Alternatively, a single monolithic ledger requires a huge amount of messages to propagate the requests and synchronize the outcome and ability to sustain the critical speed of data operations inside a node with expanding volumes of data, thus making unfeasible the essential balance of throughput and latency.

In addition to the amount of messages required to keep the replicas in sync, a monolithic ledger with global replication demands the use of a complex consensus protocol. This is inefficient The more complex the synchronization with asynchronous messaging, the more expensive it is in terms of time. Moreover, this approach is unsustainable. Bitcoin illustrates operations with monolithic ledgers. For the last 12 months, the size of its blockchain doubled[38] from 40GB to 80GB and its long term growth curve[39] has a hyper-linear shape. Ledger size of a system with thousand times higher throughput will easily grow up to a petabyte and its data operations will need scaling out with horizontal partitioning just to cope with the workload.

At present, multi-partition database transactions degrade the throughput and defeat the reason to scale out. In addition, no transaction management product currently on the market can guarantee high availability of multi-partition transactions. A blocked transaction may prevent participation in the voting rounds of a stateful consensus protocol and the subsequent update of state. Achieving the needed throughput and latency like most credit cards is unfeasible with a single ledger having a monolithic state and global replication. Such throughput, eventually achievable with consensus rounds that involve very large sets of transactions, will have prohibitively large latencies as a complex consensus with asynchronous messages cannot become more efficient and the speed of data transmission cannot exceed the speed of light. Understanding the limits imposed by the consensus protocol requires a brief description of differences between the stateful consensus protocols and the stateless consensus protocols.

A stateless communication protocol[40] is one that treats each request as independent and unrelated to any previous request, so that the communication consists of independent pairs of request and response. A stateful communication protocol is one that needs to keep the internal state, which is outcome from previous sessions, as it affects the outcome of next sessions. A session of any consensus protocol comprises multiple communication rounds with a stateful communication protocol. The stateful communication within a consensus protocol is unavoidable. The outcome of a round is affected by the outcome of previous rounds of a consensus session. A stateless consensus protocol is one where the outcome of a protocol session is completely independent from the outcome of all previous sessions. A stateful consensus protocol is one where the outcome of a session is causally dependent from the outcome of previous sessions.

In the context of payment networks, the objective of consensus is to prevent double spending. One way of accomplishing this is to sequentially order all requested transactions. A consensus about the sequential order guarantees that in case of an attempted double spending all replicas will successfully complete exactly the same transaction as first one and reject the second one.

[38] https://blockchain.info/charts/blocks-size
[39] https://blockchain.info/charts/blocks-size?timespan=a
[40] https://en.wikipedia.org/wiki/Stateless_protocol

The second way to prevent double spending is slightly more complex. Once transaction requests are received on a node, all of them are executed against the node's replica of the ledger. In a case of an attempted double spending, the first one will be completed and the second one will be rejected. The outcome is not made durable before the consensus protocol achieves its objective. What causes a problem is the fact that requests arrive to different nodes in different order. Nothing guarantees that the resulting pairs of 'one completed, one rejected' transactions are the same on all nodes, and therefore on all replicas of the ledger. The protocol aims an agreement about which one of both transaction to complete. Typically it requires multiple voting rounds.

Protocols aiming consensus about the ordering of requests are stateless. Sequential ordering of a bunch of requests is in no way related to the sequential ordering of any previous bunch. The protocols must guarantee that even the tolerated number of malicious nodes will know the agreed sequential order per each session. This is a strict safety guarantee, not a probabilistic one. The strictness of safety has its price. Every node must be aware of every other node participating in a protocol session and these protocols are clearly inapplicable with permissionless networks. Additionally, the number of the necessary per session messages is exponentially related to the number of nodes and the performance becomes inefficient with large permissioned networks.

Protocols that involve voting rounds are stateful. Detection of a double spending depends on state, which is result of previous consensus sessions. Thus, the safety of a stateful consensus protocol is strictly dependent on the atomic consistency between the state on all nodes. Otherwise, one node may detect an attempted double spending but another one may not. Protocols' safety guarantees are probabilistic. Strict guarantees are impractical – too many voting rounds take too much time and would destroy protocols' utility. On the positive side, the nodes do not need to be aware of every other node in a protocol session. Properly structured propagation of requests and voting can enable unrestricted expansion of number of network nodes. Stateful consensus is the price for unrestricted number of nodes in a ledger. Maintaining safety with a probabilistic guarantee requires maintaining a delicate and vulnerable equilibrium between the possible consequences of events beyond control. A negligible miscalculation of probabilities combined with unpredictable sequence of events may negatively affect it.

Blockchains' monolithic central state and the stateful consensus protocols are inconvenient for the Internet-of-Value. It has to be built[41] with stateless consensus protocols in the way the Internet was built with stateless communication protocols. It requires *s*afety, performance, unrestricted growth, and decentralization, but the unrestricted growth using a ledger with monolithic central state dictates the use of a stateful consensus protocol. The balance between performance and probabilistically guaranteed safety dictates the need of a central administration to configure the propagation of requests and voting. Therefore, safety with the required performance and unrestricted growth cannot coexist with decentralization. Thus, the monolithic central state precludes some of the critically important properties of the Internet-of-Value

The term Internet-of-Value originates from Ripple[42] but digital currency system of Ripple is built as ledger with monolithic central state and uses a stateful consensus protocol. As a remedy, Ripple proposes a Protocol for Interledger Payments[43]. The protocol, however, does not address the inability for decentralization and spoils the safety of participating ledgers. The protocol

[41] https://medium.com/@justmoon/the-subtle-tyranny-of-blockchain-91d98b8a3a65#.c4574d9vf
[42] http://openmarkets.cmegroup.com/10381/what-is-an-internet-of-value
[43] https://interledger.org/interledger.pdf

operates in two modes: Universal and Atomic. In Universal mode, its safety does not hold if a node that bridges two ledgers is faulty. In Atomic mode, its safety does not hold if both bridging nodes of a ledger are faulty. In both modes, safety of interledger payments depends on liveness of participating ledgers. Therefore, safety of one ledger depends on liveness of another ledger. Moreover, the protocol cannot guarantee that any number of faults per ledger within the tolerated limit of the ledger will not destroy the safety of an interledger transaction.

The Internet-of-Value needs high number of frequently used accounts, high throughput, low latency, safety, liveness, unrestricted growth, and decentralization. These cannot be achieved with a permissionless ledger with monolithic state, or with a permissioned ledger with monolithic state and a stateful consensus protocol, or with a permissioned ledger with monolithic state and a stateless consensus protocol.

One not yet explored by anyone else way of satisfying all of the above requirements is to device a composite ledger, which operates as federation of permissioned ledgers each with a monolithic state, where each ledger uses a stateless consensus protocol. With such federation, safety of transactions that modify the state of two monolithic ledgers depends on liveness of the participating ledgers as well. Safety of transactions of a composite ledger, however, does not depend on health of a single node or two nodes where two monolithic ledgers intersect. Thus, a stop-faulty or Byzantine faulty node cannot affect the safety of transactions with distributed state.

Agreement within a reasonable time frame inside a permissioned blockchain system is a feasible objective where the number of participants is limited and their identities are known in advance. A permissioned blockchain system with a restricted number of replicas can provide a built-in tolerance to a proportional number of malicious replicas, eliminate the waste of energy, and radically increase the throughput. A thousand transactions per second and few seconds latency even with replicas on different continents and tens of thousands of transactions with short distances between replicas are realistic objectives.

Execution of an asynchronous agreement protocol is the most vulnerable step in replication. Completion of this step with no risks for liveness or throughput is as important as the safety. It requires elimination of the possibility that during its execution the agreement protocol can lock and stay locked until expiring of a timeout interval. At present, however, no known technology can guarantee this without spoiling the safety or atomic consistency of replicas._Some of the brightest computer scientists devoted years to the theory and practice of replication via asynchronous network with distributed agreement. Despite these, not all issues of interdependence between safety, liveness, and throughput are fully solved.

Where a network link uses routers, no bound can be placed on the time it takes to deliver a message. Packet's delay on a router depends on how full router's queuing buffer is in packet's direction. Moreover, when the buffer overflows, the router has no other choice but to drop the packet. Similarly, a process on a healthy computer may take different time to perform a request, mainly because a different number of queued requests may wait for execution. Queuing lets cope with spikes of incoming requests. A system where no bound is placed on time to process a request or to deliver a message is qualified as asynchronous one. In contrast, synchronous system is one where performing a request or delivering a message that takes too long is considered a failure.

The necessity of a time bound is an unavoidable evil with the asynchronous systems at present. Castro-Liskov[44] and Byzantine Paxos[45] asynchronous agreement protocols cannot guarantee simultaneous safety and liveness in event of a stop-faulty or malicious computer, or a partitioned network. As Leslie Lamport[46] wrote[47], in theory these protocols tolerate faulty computers, but in practice have difficulty distinguishing a faulty computer from an ordinary delayed message. The same is true with computers maliciously delaying a reply message to block the protocol.

Castro-Liskov algorithm, for example, relies on synchrony for liveness. This lets an attacker who controls the leader to slow down the system performance. Even without a single Byzantine-faulty node in the system, a network partition that creates a minority group with more than a third of all nodes will block the entire system since the consensus algorithm cannot terminate. Alternatively, when the algorithm operates with the maximum tolerated Byzantine-faulty nodes, a single lost message will prevent algorithm termination and the system will block.

The FLP result is a short way to refer to the theorem[48] of Fischer, Lynch, and Paterson. It proves that a system cannot guarantee agreement with one faulty computer in an asynchronous network by showing that a possible blocking of the agreement protocol cannot be ruled out. Use of a time bound prevents blocking. Processing requests while a faulty or malicious replica is being recovered, however, spoils the atomic consistency between the recovered replica and the rest. Moreover, during a delayed message or network partition, a replica may be seen as stop-faulty or malicious and excluded from the protocol. This prevents locking but spoils the throughput and may impair the ability to neutralize a real malicious behavior, thus damaging the consistency.

For permissioned blockchain that aims to perform with high throughput, the sequential execution is not an option. It needs a transaction manager – to increase the throughput with a serializable[49] execution order, which ensures a serial effect with concurrent execution. Moreover, it needs the transaction manager to be deterministic – to ensure execution with exactly the same serializable order on every replica. Multiple database systems provide deterministic transaction management. The unsolved problem with each one of these systems is scalability. A blockchain database is append-only and immutable. Nothing is ever deleted. With a thousand transactions per second, the volume of data grows rapidly and beyond a certain point starts degrading the performance.

Normal response is to scale out with partitioning of data. It works well where the nature of data allows perfect partitioning. This means a transaction does not, and will never have to, access more than one partition. Blockchain use cases rarely permit perfect partitioning. Accessing multiple partitions requires distributed transactions[50], which are prone to blocking[51], thus negatively affecting the availability, and have high performance costs[52], thus defeating the reason to scale out. Scaling out is inevitable with the ever expanding data. Devising highly available high-throughput distributed transactions is a problem waiting its solution.

---

[44] http://pmg.csail.mit.edu/papers/osdi99.pdf
[45] http://research.microsoft.com/en-us/um/people/lamport/tla/byzsimple.pdf
[46] https://en.wikipedia.org/wiki/Leslie_Lamport
[47] Leaderless Byzantine Consensus, United States Patent 7,797,457
[48] https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf
[49] https://en.wikipedia.org/wiki/Serializability
[50] https://en.wikipedia.org/wiki/Distributed_transaction
[51] https://en.wikipedia.org/wiki/Two-phase_commit_protoco
[52] http://adrianmarriott.net/logosroot/papers/LifeBeyondTxns.pdf

The following non-existing technologies are necessary and sufficient where the goal is to guarantee safety, liveness, and throughput with an asynchronous agreement protocol, and availability, atomic consistency and throughput with a scaled out database:

1. Circumvention of CAP theorem for permissioned networks. It needs invention of a messaging scheme with multiple communication paths and guaranteed delivery. This is sufficient to prevent a partitioned wide area network link from evolving into partitioned distributed system.

2. Circumvention of FLP result. It needs a replica node to be a system with reliable operations, which sends the expected replies regardless of a stop-faulty or malicious computer. This, together with a guaranteed message delivery, is sufficient to prevent blocking of any asynchronous agreement protocol.

3. Implementation of new transactions. They need to be highly available deterministic multi-partition transactions, which are capable to access multiple partitions with no degrading the throughput or latency. This is sufficient to scale out the data operations without risking the throughput and availability.

4. Implementation of asynchronous leaderless Byzantine consensus algorithm. This is sufficient to preclude the possibility of attacks on liveness of the agreement when a Byzantine-faulty node becomes a leader.

A permissioned blockchain, built with reliance on these technologies and having a limited number of participating nodes, can achieve its goals. It will have the best of Bitcoin: perform with guaranteed consistency and availability, provide tolerances to network partitions and malicious attempts, and run on decentralized infrastructure with decentralized administration. It will be better than Bitcoin: run with no use of excessive computing power and electric energy, and perform with high throughput, low latency, and scalability of data.

Circumvention of the FLP result will increase the tolerance to Byzantine faults of some stateless consensus protocols from (n-1)/3 to the theoretical maximum of (n-1)/2 , where n is the number of ledger nodes. The increased tolerance to Byzantine faults is extremely important for any stateless consensus protocols as scaling the number of nodes compromises the protocol efficiency.

The successful design and implementation of a permissioned ledger with guaranteed both safety and liveness regardless of anything that possibly may go wrong is the sufficient condition that enables the design and implementation of a ledger with distributed state, which operates as federation of permissioned ledgers with monolithic state. Inside such federation, safety and liveness of distributed transactions, modifying the state of two monolithic ledgers, are guaranteed regardless of anything that may go wrong within the tolerated limit per ledger participating in the chain of ledgers between the sending and recipient ledgers.

Summary of Invention

A distributed system with asynchronous communication between known number of nodes with known identities implements a messaging scheme with multiple communication paths between any two nodes and guaranteed delivery thereby preventing a partitioned wide area network link from evolving into a partitioned distributed system. The distributed system further implements reliable node operations, which result into sending the expected replies regardless of a stop-faulty computer and thus prevents blocking of any non-Byzantine consensus protocol. The reliable node operations further include highly available multi-partition transactions with a scaled out data management with no degradation of the throughput and latency. The distributed systems implements a fully asynchronous leaderless Byzantine consensus algorithm, which precludes the possibility of attacks on liveness of the consensus algorithm when a Byzantine-faulty node becomes a leader, reduces by one the necessary messaging rounds of a typical Byzantine consensus protocol, and increases the tolerance to Byzantine faults from (n-1)/3 to the theoretical maximum of (n-1)/2 , where n is the number of system nodes. A this way implemented ledger with monolithic state and known distributed replica nodes is integrated with one of multiple this way implemented ledgers with monolithic state and known distributed replica nodes to form an open membership federation of ledgers. Within the federation, safety and liveness of distributed transactions, modifying the state of two monolithic ledgers, are guaranteed regardless of anything that may go wrong within the tolerated limit per ledger participating in the chain starting from the ledger of the sending account and finishing with the ledger of the recipient account.

In the drawings:

Fig. 1 is block diagram of a system for network-link partition-tolerant delivery of messages between two computers and between a computer and a plurality of two computers via asynchronous public network.

Fig.2 is flowchart of a method of providing detection, neutralization, and recovery of a stop-faulty process on a computer of a cluster of two computers.

Fig.3 is block diagram of a ledger node built as a cluster of computers with resilient operations guaranteed with redundant communication paths and replicated functional elements.

Fig. 4 is block diagram of a cycle of an asynchronous leaderless Byzantine agreement protocol.

Fig. 5 is block diagram of a federation with open membership between two permissioned ledgers, each one with four distributed replicas.

Fig. 6 is block diagram of a federation with open membership between three permissioned ledgers, each one with four distributed replicas.

Fig. 7 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - placing the money on escrow account of a business entity that has a node in Ledger1 and a node in Ledger2 and notification to Ledger2 that money was placed on escrow.

Fig. 8 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - placing the money on escrow account of a business entity that has a node in Ledger2 and a node in Ledger3 and notification to Ledger3 that money was placed on escrow.

Fig. 9 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - paying the money to the recipient in exchange for a digitally signed receipt, sending notifications to Ledger2 and Ledger1 about payment to the recipient and a message to Ledger2 containing the payment receipt.

Fig. 10 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - Ledger2 payment of the money from the escrow account to the account of business entity that has a node in Ledger2 and a node in Ledger3 and sending notifications to Ledger3 and a message to Ledger1 containing the payment receipt.

Fig.11 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - Ledger1 payment of the money from the escrow account to the account of business entity that has a node in Ledger1 and a node in Ledger2 and sending notifications to Ledger2.

Full Description

This invention provides a technology: a) to build a digital ledger node that tolerates computer stop-faults; b) to deploy over asynchronous networks a multi-replica permissioned ledger performing with guaranteed: atomic consistency of replicas, availability, and tolerance to network link partitions; c) to deploy an open membership network of permissioned ledgers over asynchronous networks; d) to execute interledger transactions that do not jeopardize the safety and liveness of individual consensus protocols of the participating ledgers; e) to allow gradual and decentralized deployment of an open network with unlimited expansion of the number of participating permissioned ledgers; and f) to allow fragmented changes permitting integration of solutions that implement different concepts as long as these solutions conform to a set of rules.

The correctness characteristics of performance of a permissioned ledger are dependent on accuracy of the following two assumptions: a) Operator of a node of a permissioned ledger will not undertake malicious activities aiming to prevent or delay termination of ledger's consensus protocol as long as these activities cannot be masked as a stop-faulty node or a partitioned network and such activities entail ledger executed consequences; b) At any time, number of ledger nodes that may conspire to tamper ledger's data, to censor execution of a transaction, or to affect the outcome of a transaction, are less than half of all ledger nodes.

Similarly to the decentralized configuration and operations of the Internet, no single authority is needed for configuration or operations of the open network. The connection of a device to the Internet requires human involvement in form of a permission from a service provider that operates the network to which the device will connect. Compared to it, connection of a node to a permissioned ledger is one step ahead in elimination of human involvement. It requires individual consent from each one of ledger nodes and the process can be fully executed by software algorithm according to specified rules and conditions. Human involvement is necessary only to guarantee that the joining node is independent from each one of the existing ledger nodes. Unlike the connection of a node to a ledger, connection of a ledger to an open network of ledgers

does nor require permission from all other ledgers that are already members of the open network. It requires permission from only one ledger that is already a member. Yet, to give such permission, a ledger needs consent from each one of its nodes.

To make these possible, the invention:

1. Circumvents CAP theorem for a distributed system with asynchronous communication between known number of nodes with known identities and IP addresses. The circumvention is achieved with implementation of a messaging scheme with multiple communication paths between any two nodes and guaranteed delivery, thereby preventing a partitioned wide area network link from evolving into a partitioned distributed system.

2. Circumvents the FLP result for a distributed system with asynchronous communication between known number of nodes with known identities and IP addresses. The circumvention is achieved by stepping on the circumvented CAP theorem and by implementation of each node of the system as a node with reliable operations, which sends the expected reply messages regardless of a stop-faulty computer and takes care for guaranteed delivery of each sent message regardless of a possibly partitioned wide area network link. The combined effect of the reliable node operations and the guaranteed delivery of each sent message is sufficient to prevent blocking of any non-Byzantine agreement protocol.

3. Devises highly available deterministic multi-partition transactions, which are capable to access multiple data partitions with no degradation of the throughput and latency. This is sufficient to scale out the data operations without risking the throughput and availability, thus solving the data storage challenges and performance challenges related to immutability where the record of every transaction remains permanently on the data store.

4. Devises a fully asynchronous leaderless Byzantine consensus algorithm for distributed systems with known number of nodes with known identities and IP addresses. This is sufficient to preclude the possibility of attacks on liveness of the consensus algorithm when a Byzantine-faulty node becomes a leader and to reduce by one the necessary messaging rounds of a typical Byzantine consensus protocol. The combined effect with the circumvented FLP result is increased tolerance to Byzantine faults from (n-1)/3 to the theoretical maximum of (n-1)/2 , where n is the number of nodes.

5. Devises an open membership network of permissioned ledgers, where the network does not need a central administration. For a permissioned ledger to join the network it is necessary and sufficient that a business entity has a node in a ledger, which is existing member of the network and has a node in the ledger aspiring to join the network. Safety and liveness of an interledger distributed transaction is guaranteed regardless of anything that may go wrong within the tolerated limit per ledger participating in the chain starting from the ledger of the sending account and finishing with the ledger of the recipient account. Main building blocks of an interledger distributed transaction are ledger-executed contracts, each implemented as event-driven state-machine workflow executed on each node of a ledger. A step of an event-driven state-machine workflow can modify three categories of state: control state, workflow state, and data state. A modification of data state per workflow step is performed as data transaction to increase the throughput with serializable concurrent execution. A single ledger transaction is implemented on each ledger node as a one step workflow executing a data transaction.

Fig. 1 is block diagram of a system for network-link partition-tolerant delivery of messages between two computers via public asynchronous network or between a computer and a plurality of two computers via asynchronous public network. The system comprises three computing clusters. Cluster 101 has 2 computers (110a and 110b) and 2 routing devices (111a and 111b) connected via pair of local area network segments (101a and 101b). Cluster 102 has 2 computers (120a and 120b) and 2 routing devices (121a and 121b) connected via pair of local area network segments (102a and 102b). Cluster 103 has 2 computers (130a and 130b) and 2 routing devices (131a and 131b) connected via pair of local area network segments (103a and 103b). Each computing cluster uses two independent Internet service providers, one for each routing device. On the diagram, each routing device has established two point-to-point connections, each to a routing device of another computing cluster. On the diagram: 104a denotes a connection from 111a to 121a and a connection from 121a to 111a, 104b denotes a connection from 111b to 121b and a connection from 121b to 111b, 105a denotes a connection from 111a to 131a and a connection from 131a to 111a, 105b denotes a connection from 111b to 131b and a connection from 131b to 111b, 106a denotes a connection from 121a to 131a and a connection from 131a to 121a, and 106b denotes a connection from 121b to 131b and a connection from 131b to 121b.

Each computer and routing device of a cluster runs a network-link-clustering software module. This module implements an application layer transport protocol for reliable delivery of packets via multicast communication. In addition, each routing device implements an algorithm for establishing boundaries of a message in a stream of bytes received via point-to-point communication. A network-link-clustering module constructs and uses a listen socket object per local area network segment and a send socket per multicast destination per local area network segment. In addition, each routing device constructs and uses a listen socket object per point-to-point connection and a send socket object per point-to-point connection. A network-link-clustering module transmits a message by duplicating the message on buffers of a pair of send socket objects and handling the transmission using both socket objects. A network-link-clustering module receives a message with involvement of two socked objects. A process on a computer sends a message by writing it to send buffer and reads a received message from read buffer of the network-link-clustering module.

Multicast of a message from one computer, say 110a, to every other computer in the system happens in the following manner. The network-link-clustering module on computer 110a multicasts the message to a multicast group that comprises computers 110a and 110b and routing devices 111a and 111b. On receiving the message, a routing device 111a transmits it over each one of connections 104a and 105a. On receiving the message, routing device 121a multicasts it to a multicast group that comprises routing devices 121a and 121b and computers 120a and 120b. On receiving the message, routing device 131a multicasts it to a multicast group that comprises routing devices 131a and 131b and computers 130a and 130b. Routing devices of clusters 101 and 102 take care to ensure delivery of the message from cluster 101 to cluster 102 regardless of possible partition of a link between routing device 111a and routing device 121a or a link between routing device 111b and routing device 121b. Routing devices of clusters 101 and 103 take care to ensure delivery of the message from cluster 101 to cluster 103 regardless of possible partition of a link between routing device 111a and routing device 131a or a link between routing device 111b and routing device 131b.

Fig.2 is flowchart of a method of providing detection, neutralization, and recovery of a stop-faulty process on a computer of a cluster of two computers. The method requires a minimal configuration comprising a client computer running a client process and two server computers, each running an instance of service process. All three computers are interconnected via a local area network. The flowchart represents the interaction between the client process, the primary service process, and the replica service process. Objective of this method is to guarantee that each client request will receive a response regardless of a stop-fault of the primary service process.

The method comprises: running a service process on the primary and the replica computers; maintaining state of execution of the service process on the primary and the replica computers; sending a message by the client computer for the primary computer via multicasting the message to the primary and replica computers; starting a timer on the replica computer on receiving a message that needs a reply or the last message of a plurality of messages that needs a group reply; using the timer to watch out for expiration of a timeout value specifying the maximum interval of time given to a primary computer to send a reply; sending a message for the client computer via multicasting the message to the client computer and to replica computer;

On event of a received message from the client computer: the replica computer starts a timer; the primary and the replica computers process the received message and prepare a reply message; on readiness of a reply message the primary computer multicasts it. On event of a received message from a primary computer, the replica computer stops the timer related to the received message. On event of a timeout, the replica computer triggers stop and restart of the service process on the primary computer, becomes primary, and multicasts the message that ex primary failed to multicast. On event of a completed recovery of an ex primary computer, it assumes the role of replica computer.

Fig.3 is block diagram of a ledger node built as a cluster of computers with resilient operations guaranteed with redundant communication paths and replicated functional elements. The node ensures high-availability of horizontally scaled data by performing highly available deterministic transactions requested by node clients and clients of all other nodes of the ledger that are partners in establishing consensus about a common sequential order of execution of all transactions requested by all clients of the ledger. Functional elements of the node are: clients communication, partners multicast, transactions management, and data management per data partition. As a functional element, the transaction management performs two distinctive tasks: a) executing a consensus protocol algorithm that orders the requested transactions and b) executing the actual transactions. The transaction management communicates with every other functional element using multicast messaging with guaranteed delivery.

On the diagram blocks 301a and 301b represent computers for communication with clients, blocks 302a and 302b represent computers for multicast with partners, blocks 303a and 303b represent transaction management computers, blocks 304a and 304b represent data management computers for data partition 1, blocks 305a and 305b represent data management computers for data partition 2, and blocks 306a and 306b represent data management computers for data partition n. The clients communication computers and the transaction management computers communicate between themselves over a pair of local area network segments 307a and 307b. The partners multicast computers and the transaction management computers communicate between themselves over a pair of local area network segments 308a and 308b.

The data management computers and the transaction management computers communicate between themselves over a pair of local area network segments 309a and 309b.

The transaction management executes the requested transactions on both primary and replica transaction manager computers in following manner: a) each of the transaction managers maintains data item locks in its main memory; b) each of the transaction managers obtains locks on the necessary for transaction execution data items that may conflict with any other transaction; c) the primary transaction manager requests transaction input data from one of data managers per involved data partition; d) the data manager multicasts the requested transaction input data to both transaction manager computers; e) each of the transaction managers executes a transaction program associated with the requested transaction in its main memory and then releases the locks; f) the primary transaction manager multicasts a request to make transaction outcome data durable to both data manager computers per involved data partition where the received requests are performed in a linearizable manner.

The high-availability of horizontally scaled data and high-availability of transactions with the horizontally scaled data is achieved with detection the effect of a faulty transaction manager or data manager and neutralization the effect of a faulty transaction manager. The sub-system for these comprises: A) a computer that sends transaction request messages by multicasting each message to both transaction manager computers and a computer that receives transaction response messages, where the request sending computer and the response receiving computer can be but not necessarily are the same computer; B) a primary transaction manager computer that: a) receives transaction request messages; b) sends 'read data' request messages, each message directed to one of data managers per involved data partition, by multicasting the message to the data manager and to the replica transaction manager; c) starts a timer on sending a 'read data' request message to a data manager; d) receives 'read data' result messages with from the requested data managers; e) stops a timer on receiving a 'read data' result message with from the requested data manager; f) executes each transaction in its main memory with the 'read data' result, prepares transaction response message and transaction 'write data' request; g) sends 'write data' request messages by multicasting each message to both data managers and to the replica transaction manager; h) starts a timer on sending a 'write data' request message to data managers; i) receives 'write data' result messages from the requested data managers; j) stops a timer on receiving a 'write data' result message from all requested data managers; k) sends transaction response messages by multicasting each message to the client and to the replica transaction manager; l) on recovery after a failure, rebuilds its state from the current primary transaction manager and becomes the replica transaction manager computer; m) on timeout triggers a recovery of the stop-faulty data manager; C) data manager computers that: a) receive 'read data' request messages; b) perform 'read data' operations; c) send 'read data' result messages by multicasting each message to both transaction managers; d) receive 'write data' request messages; e) perform 'write data' operations; f) send 'write data' result messages by multicasting each message to both transaction managers; D) a replica transaction manager computer that: a) uses timers to detect expiry of a timeout value specifying the maximum interval of time given to the primary transaction manager to send a message; b) starts a timer on receiving a transaction request that requires the primary transaction manager to send a 'read data' request message; c) stops the timer on receiving a 'read data' request message from the primary transaction manager; d) receives 'read data' result messages with from the requested data managers; e) executes each transaction in its main memory with the 'read data' result, prepares transaction response message and transaction 'write data' request; f) starts a timer on

readiness of a transaction 'write data' request; g) stops the timer on receiving a 'write data' request message from the primary transaction manager; h) starts a timer on receiving a 'write data' result message from a data manager; i) stops the timer on receiving a transaction response message from the primary transaction manager; j) on timeout triggers a recovery of the primary transaction manager, becomes the new primary transaction manager, continues the execution of each started but not completed transaction;

The high-availability of horizontally scaled data is achieved with recovery of a faulty data manager and making its state atomically consistent with the other data manager of same data partition. The activities performed by a transaction manager with these objectives are: a) detecting a faulty data manager; b) triggering a recovery procedure; c) detecting a recovered data manager; d) preserving in memory all group commit request messages from the detection until the recovery; e) sending to the recovered data manager all preserved group commit request messages. The activities performed by the recovered data manager are: a) switching to non-blocking atomic 'write data' operations; b) executing by the preserved group commit request messages; c) receiving and preserving all group commit request messages incoming during the execution of the non-executed group commit requests; d) continuing the execution of non-executed group commit requests and the receiving of new group commit request messages until there is no more non-executed group commit requests; e) switching back to blocking atomic 'write data' operations.

The executed by the transaction management consensus algorithm uses a fully asynchronous leaderless Byzantine consensus protocol. Implementing each node of a ledger as a cluster with resilient operations and the communication between each pair of nodes as tolerant to partitioned network links lets the design of the ledger derive a number of advantages. Firstly, the ledger can achieve consensus with a fully asynchronous protocol as the combined effect of the resilient nodes and the partition tolerant communication is a circumvention of the FLP result, which proves that consensus in asynchronous network cannot be guaranteed with one stop-faulty node. Secondly, the consensus protocol tolerance to Byzantine-faulty nodes increases from (n-1)/3 to the theoretical maximum of (n-1)/2 , where n is the number of nodes, as a result of a very realistic assumption. In a ledger with permissioned membership, no member node can afford to skip sending a message hiding behind a partitioned network or faking a stop-faulty node with an objective to prevent the termination of consensus protocol. Thirdly, once every node of the ledger knows every other node and its IP addresses, the fully asynchronous Byzantine consensus protocol can be simplified to become a fully asynchronous leaderless Byzantine consensus protocol and as a result reduce by one the number of typically required three messaging rounds.

Fig. 4 is block diagram of a cycle of a fully asynchronous leaderless Byzantine agreement protocol. Each of the primary transaction manager computer and the replica transaction manager computer of each node of a ledger executes the agreement protocol with an algorithm where a cycle of the algorithm comprises: a) on receiving a NEW_CYCLE message from all other nodes, each of both computers includes the transaction requests received during the previous cycle in a REQUESTS message and the primary computer sends it to all other nodes; b) on receiving a REQUESTS message, each of both computers invokes an asynchronous algorithm for sequential ordering that produces the same output on receiving the same set of messages regardless of their receiving order; c) on receiving the last waited REQUESTS message, the sequential ordering algorithm terminates and produces a preliminary sequential order of requests; d) on having its preliminary sequential order ready, each of both computers includes it in a LOCAL_ORDER

message and the primary computer sends it to all other nodes; e) on receiving a LOCAL_ORDER message, each of both computers invokes an asynchronous algorithm for filtering of malicious actions of up to f nodes and produces the same output on all nodes, even on different set of messages and receiving order; f) on receiving the last waited LOCAL_ORDER message, the algorithm for filtering of malicious actions terminates and produces the final sequential order of requested transactions that is the agreement order of requested transactions of the current cycle; g) on having the agreement order produced, the primary and the replica start executing the requested transactions and the primary computer sends a NEW_CYCLE message to all other nodes.

Fig. 5 is block diagram of a network with open membership between two permissioned ledgers, each one with four distributed replica nodes. Nodes 511, 512, 513, and 514 are the replica nodes of ledger 510 connected via public asynchronous network. Nodes 521, 522, 523, and 524 are the replica nodes of ledger 520 connected via public asynchronous network. Nodes 513 and 521 belong to the same business entity and may be connected via private local area network. Node 513 is connect to nodes 522, 523, and 524 via public asynchronous network. Node 521 is connected to nodes 511, 512, and 514 via public asynchronous network. Nodes 522, 523, and 524 are connect to nodes 511, 512, and 514 via public asynchronous network.

The purpose for existence of this network is to enable execution of transactions between an account of ledger 510 and an account of ledger 520. Existence in both ledgers a node that belongs to the same business entity is a precondition for an interledger transaction. The technology of this invention lets a transaction transfer the control of a digital asset from one account to another. The associated transfer of ownership and its finality from legal point of view is not a concern of the technology itself, especially having in mind that different nodes my physically exists and operate in different jurisdictions. To simplify the description, the technology of this invention will be described with transactions transferring money from under the control of one account to under the control of another account.

On the diagram, 513c represents an account in ledger 510. This account is controlled by the business entity that operates nodes 513 and 521. There is an escrow account 513a associated to account 513c. Money in account 513a is under the control of the ledger 510 via a ledger-executed contract or multiple ledger-executed contracts, each with two possible outcomes. If the specified condition for successful completion of a contract is satisfied before expiring of an associated timeout interval, the money controlled via this contract is transferred from account 513a to account 513c. Otherwise, the contact completes with returning the money back to the account where the money belonged before being placed in account 513a.

On the diagram, 521d represents an account in ledger 520. This account is controlled by the business entity that operates nodes 513 and 521. There is an escrow account 521b associated to account 521d. Money in account 521b is under the control of the ledger 520 via a ledger-executed contract or multiple ledger-executed contracts, each with two possible outcomes. If the specified condition for successful completion of a contract is satisfied before expiring of an associated timeout interval, the money controlled via this contract is transferred from account 521b to account 521d. Otherwise, the contact completes with returning the money back to the account where the money belonged before being placed in account 521b.

All accounts of ledger 510 exist on ledger's four replica nodes. Accounts 513a and 513c of ledger 510 are in addition replicated on ledger 520. The replication on ledger 520 happens in the following manner. Once the nodes of ledger 510 update the content of account 513a and its replicas, presented as account 511a on node 511, as account 512a on node 512, and as account 514a on node 514, each one of ledger 510 nodes sends a notification to each one of ledger 520 nodes to update its replicas of the updated account presented as account 521a on node 521, as account 522a on node 522, as account 523a on node 523, and as account 524a on node 524. Similarly, once the nodes of ledger 510 update the content of account 513c and its replicas, presented as account 511c on node 511, as account 512c on node 512, and as account 514c on node 514, each one of ledger 510 nodes sends a notification to each one of ledger 520 nodes to update its replicas of the updated account, presented as account 521c on node 521, as account 522c on node 522, as account 523c on node 523, and as account 524c on node 524.

All accounts of ledger 520 exist on ledger's four replica nodes. Accounts 521b and 521d of ledger 520 are in addition replicated on ledger 510. The replication on ledger 510 happens in the following manner. Once the nodes of ledger 520 update the content of account 521b and its replicas, presented as account 522b on node 522, as account 523b on node 523, and as account 524b on node 524, each one of ledger 520 nodes sends a notification to each one of ledger 510 nodes to update its replicas of the updated account presented as account 511b on node 511, as account 512b on node 512, as account 513b on node 513, and as account 514b on node 514. Similarly, once the nodes of ledger 520 update the content of account 521d and its replicas, presented as account 522d on node 522, as account 523d on node 523, and as account 524d on node 524, each one of ledger 520 nodes sends a notification to each one of ledger 510 nodes to update its replicas of the updated account, presented as account 511d on node 511, as account 512d on node 512, as account 513d on node 513, and as account 514d on node 514.

Fig. 6 is block diagram of a network with open membership between three permissioned ledgers, each one with four distributed replica nodes. Nodes 611, 612, 613, and 614 are the replica nodes of ledger 610 connected via public asynchronous network. Nodes 621, 622, 623, and 624 are the replica nodes of ledger 620 connected via public asynchronous network. Nodes 631, 632, 633, and 634 are the replica nodes of ledger 630 connected via public asynchronous network. Nodes 613 and 621 belong to the same business entity and may be connected via private local area network. Nodes 613 is connect to nodes 622, 623, and 624 via public asynchronous network. Node 621 is connected to nodes 611, 612, and 614 via public asynchronous network. Nodes 622, 623, and 624 are connect to nodes 611, 612, and 614 via public asynchronous network. Nodes 623 and 631 belong to the same business entity and may be connected via private local area network. Nodes 623 is connect to nodes 632, 633, and 634 via public asynchronous network. Node 631 is connected to nodes 621, 622, and 624 via public asynchronous network. Nodes 622, 623, and 624 are connect to nodes 631, 632, and 634 via public asynchronous network.

The accounts controlled by the business entity that operates nodes 613 and 621 and the escrow accounts associated to these accounts are replicated on nodes 611, 612, 613, and 614 of ledger 610 and on nodes 621, 622, 623, and 624 of ledger 620. Once the replica nodes of ledger 610 have updated the account controlled by the business entity that operates nodes 613 and 621 or the associated to it escrow account, each replica node of ledger 610 sends a notification message to every replica node of ledger 620. Once the replica nodes of ledger 620 have updated the account controlled by the business entity that operates nodes 613 and 621 or the associated to it escrow account, each replica node of ledger 620 sends a notification message to every replica

node of ledger 610. Similarly, the accounts controlled by the business entity that operates nodes 623 and 631 and the escrow accounts associated to these accounts are replicated on nodes 621, 622, 623, and 624 of ledger 620 and on nodes 631, 632, 633, and 634 of ledger 630. Once the replica nodes of ledger 620 have updated the account controlled by the business entity that operates nodes 623 and 631 or the associated to it escrow account, each replica node of ledger 620 sends a notification message to every replica node of ledger 630. Once the replica nodes of ledger 630 have updated the account controlled by the business entity that operates nodes 623 and 631 or the associated to it escrow account, each replica node of ledger 630 sends a notification message to every replica node of ledger 620.

Fig. 7, Fig. 8, Fig. 9, Fig. 10, and Fig. 11 describe the steps of an interledger transaction transferring an amount of digital money from an account in a ledger to an account in another ledger. Digital money is the most convenient for description, illustration, and comprehension purposes digital asset. An interledger transaction with this invention's open network of permissioned ledgers can transfer the ownership of any other digital asset. The description of the steps uses the notion of digitally signed receipt, which is a digital asset itself. When an interledger transaction transfers ownership of a non-monetary digital assets for a digitally signed receipt or executes a barter swapping ownership of one non-monetary digital asset for ownership of another non-monetary digital asset, digital money may still be necessary as a way to compensate a business entity that enables the interaction between transacting ledgers if this entity undertakes any risks associated with the execution of the interledger transaction.

Fig. 7 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - placing the money on escrow account of a business entity that has a node in Ledger1 and a node in Ledger2 and notification to Ledger2 that money was placed on escrow. On the diagram, ledger 710 is Ledger1, ledger 720 is Ledegr2, and ledger 730 is Ledger3. Each node of ledger 710 has a copy of an account presented on the diagram as 711e, 712e, 713e, and 714e. When the owner of this account requests transfer of a certain amount of money to an account on ledger 730 with a copy of each node presented on the diagram as 731f, 732f, 733f, and 734f, the system performs the following: a) ledger 710 uses its internal messaging to verify that the business entity that operates nodes 713 and 721 has available funds in excess of the amount to be transferred and agrees to participate for a stated fee; b) ledger 710 uses the messaging channels from ledger 710 to ledger 720 to request ledger 720 to perform the same; c) ledger 720 uses its internal messaging to verify that the business entity that operates nodes 723 and 731 has available funds in excess of the amount to be transferred and agrees to participate for a stated fee; d) ledger 720 uses the messaging channels from ledger 720 to ledger 730 to request ledger 730 to provide the public key of the transfer's recipient account presented on the nodes of ledger 730 as 731f, 732f, 733f, and 734f; e) ledger 730 uses the messaging channels from ledger 730 to ledger 720 to provide the requested public key; f) ledger 720 uses the messaging channels from ledger 720 to ledger 710 to inform ledger 710 about the verification outcome and to provide the recipient's public key; g) ledger 710 requests the business entity that operates nodes 713 and 721 to hold for a specified time interval the amount that is sum of the amount to be transferred plus the participation fee of business entity that operates nodes 723 and 731; h) ledger 710 requests ledger 720 to request the business entity that operates nodes 723 and 731 to hold for a specified time interval the amount to be transferred; i) on receiving confirmation from ledger 720, ledger 710 is ready to start the execution of the requested interledger transaction.

Building blocks of interledger transactions are ledger-executed contracts. A contract is a parameterized event-driven state-machine workflow instance executed on all nodes of a ledger. A workflow instance is parameterized with initialization of its control state, workflow state, and data state. The workflow is itself defined with its: a) workflow state transitions; b) pieces of execution code that enact transitions of the workflow state; c) events that trigger the transitions of the workflow state; and d) transactions in the local copy of ledger's database that are executed to reflect the transitions of the data state. Events typically modify the control state and data state, and trigger the execution of code that enact transitions of the workflow state, which may invoke execution of transactions in the local copy of ledger's database.

In the preferred embodiment of this invention, an event-driven state-machine workflow is implemented as COM class. The COM class is programmed as a contract template indicating actions that must be performed in time or in response to predetermined external events. Similarly to the way a contract template needs to indicate some more specific details in order to become a contract body, the required behavior an instance of the COM class object needs to be parameterized with initialization of its control state, workflow state, and data state. Similarly to the way a contract signed by the party making an offer legally binds the offering party to the terms and conditions of the contract until the offer expires, an account in a ledger that initiates the creation and parameterization of an event-driven state-machine workflow instance by requesting a digital asset (digital money for example) under the its control to be placed in escrow account under the control of the workflow instance is the IT equivalent of making a one-party-signed contract offer. Satisfying by the offered party the conditions of the offer before it expires is equivalent to a both parties signed and executed contract.

The requested interledger transaction will be performed in the following manner. The sender's account will initiate creation of a workflow instance and authorize transfer under the control of the workflow instance the sum of the transferable amount plus the fee for entity that operates nodes 713 and 721 plus the fee for entity that operates nodes 723 and 731 to an escrow account. If ledger 710 does not receive within the specified time a notification from ledger 720 that ledger 720 has been notified by ledger 730 that the recipient account has received the transferable amount, the workflow instance will return the amount on escrow back to the sender's account. If ledger 710 receives that notification within the specified time, the workflow instance waits the account belonging to the entity that operates nodes 713 and 721 to provide a signed receipt from the recipient and initiate execution of transaction that will transfer the escrowed amount to it. Similarly, the entity that operates nodes 713 and 721 will initiate creation of a workflow instance and authorizes transfer of the sum of the transferable amount plus the fee for entity that operates nodes 723 and 731 to a escrow account under the control of the workflow instance. If ledger 720 does not receive within the specified time a notification from ledger 730 that the recipient account has received the transferable amount, the workflow instance will return the amount on escrow back to the account belonging to the entity that operates nodes 713 and 721. Finally, the an account in ledger 730 under the control of entity that operates nodes 723 and 731 will initiate creation of a workflow instance, authorize transfer of the transferable amount to a escrow account under the control of the workflow instance, and notify the recipient account that there is an amount to for it in exchange for a digitally signed receipt. If the recipient account provides a receipt on time, it will receive the amount on escrow; otherwise the amount will be returned back to the account under the control of entity that operates nodes 723 and 731. Once the entity that operates nodes 723 and 731 has the receipt, a node in ledger 720 under the control of that entity passes the receipt to the workflow instance run on nodes of ledger 720 to

receive the amount on escrow. The same workflow instance runs on every node of ledger 720 including a node under control of entity that operates nodes 713 and 721. Once this entity has the receipt, a node in ledger 710 under its control passes the receipt to the workflow run on nodes of ledger 710 to receive the amount on escrow. The same workflow instance runs on every node of ledger 710 and the sender's account on every node registers the possession of the signed by the recipient receipt.

On Fig.7, the sender's account represented on ledger nodes as 711e, 712e, 713e, and 714e initiates creation of a workflow instance and authorizes transfer under the control of the workflow instance the sum of the transferable amount plus the fee for entity that operates nodes 713 and 721 plus the fee for entity that operates nodes 723 and 731 to the escrow account represented on ledger nodes as 711a, 712a, 713a, and 714a. Once the local transaction performing the transfer to the escrow account is completed on a node of ledger 710, the node sends a notification message to every node of ledger 720. As a result of the messages from each node of ledger 710, the content of the escrow account represented on ledger 720 nodes as 721a, 722a, 723a, and 724a is updated to reflect the received escrowed amount. Once the node 721 is aware about the funds on escrow account to be received on presenting a receipt signed by the recipient account confirming that the transferred amount was received, the ledger 720 is ready to perform the next step in the interledger transaction.

Fig. 8 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - placing the money on escrow account of a business entity that has a node in Ledger2 and a node in Ledger3 and notification to Ledger3 that money was placed on escrow. On the diagram, ledger 810 is Ledger1, ledger 820 is Ledegr2, and ledger 830 is Ledger3. Nodes 813 and 821 are operated by the same business entity. Node 821 was assured that the escrowed amount in the account presented on ledger 810 nodes as 811a, 812a, 813a, and 814a will be received by the account, presented on ledger 810 nodes as 811c, 812c, 813c, and 814c, belonging to the same business entity, on receiving by the workflow instance controlling the escrowed funds a signed receipt from the recipient. That's why, node 821 is ready to place on escrow an amount equal to the escrowed amount in the previous step less the fee to be received by the operating entity.

The account represented on ledger 820 nodes as 821d, 822d, 823d, and 824d initiates creation of a workflow instance and authorizes transfer under the control of the workflow instance the sum of the transferable amount plus the fee for entity that operates nodes 823 and 831 to the escrow account represented on ledger nodes as 821a, 822a, 823a, and 824a. Once the local transaction performing the transfer to the escrow account is completed on a node of ledger 820, the node sends a notification message to every node of ledger 830. As a result of the messages from each node of ledger 820, the content of the escrow account represented on ledger 830 nodes as 831a, 832a, 833a, and 834a is updated to reflect the received escrowed amount. Once the node 831 is aware about the funds on escrow account to be received on presenting a receipt signed by the recipient account confirming that the transferred amount was received, the ledger 830 is ready to perform the next step in the interledger transaction.

Fig. 9 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - paying the money to the recipient in exchange for a digitally signed receipt, sending notifications to Ledger2 and Ledger1 about payment to the recipient and a message to Ledger2 containing the payment receipt. On the diagram, ledger 910 is Ledger1, ledger 920 is Ledegr2, and ledger 930 is Ledger3. Nodes 923 and 931 are operated by the same business entity. Node 931 was already

assured that the escrowed amount in ledger 920 will be received by the account on ledger 920 belonging to the same business entity, on receiving by the workflow instance controlling the escrowed funds a signed receipt from the recipient. That's why, node 931 is ready to make the payment to the final recipient.

The account represented on ledger 930 nodes as 931d, 932d, 933d, and 934d initiates creation of a workflow instance, authorizes transfer under the control of the workflow instance the transferable amount to the escrow account represented as 931e, 932e, 933e, and 934e on ledger 930, and notifies node 933 holding the private key of account 933f for signing receipts that the workflow instance will transfer the escrowed amount on receiving a signed receipt. Once the transfer to the escrow account is complete and node 933 is assured that the amount is under the control of the workflow instance, node 933 generates a signed receipt and requests the escrowed amount by passing the receipt to all other nodes. The workflow instances run on ledger nodes transfer the escrowed amount from the escrow account represented as 931e, 932e, 933e, and 934e to the recipient account represented on ledger 930 nodes as 931f, 932f, 933f, and 934f. On completed transfer, each node of ledger 930 sends notification messages to every node of ledger 920 about the payment to the recipient. In turn, each node of ledger 920 sends notification messages to every node of ledger 910 about the payment to the recipient. Account presented as 931d, 932d, 933d, and 934d that made the payment belongs to the business entity that operates nodes 923 and 931. Once node 931 has the signed receipt, it sends the receipt to node 923. Sending of the receipt is presented as message 93123. On receiving the receipt, node 923 can claim the escrowed amount that is equal to the sum of paid amount and the stipulated fee.

Fig. 10 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - Ledger2 payment of the money from the escrow account to the account of business entity that has a node in Ledger2 and a node in Ledger3 and sending notifications to Ledger3 and a message to Ledger1 containing the payment receipt. On the diagram, ledger 1010 is Ledger1, ledger 1020 is Ledegr2, and ledger 1030 is Ledger3. Nodes 1023 and 1031 are operated by the same business entity. This business entity is the one that made the payment to the recipient and it must be reimbursed with the paid amount plus the stipulated fee. To make this happen, account 1023c on node 1023 of ledger 1020 initiates execution of the last step of the workflow instance that controls the amount in the escrow account presented on ledger 1020 nodes as 1021a, 1022a, 1023a, and 1024a. The execution of this step is initiated with a message containing the signed payment receipt and requesting in exchange payment of the amount on escrow. Once all nodes of ledger 1020 have received the encoded payment receipt, the receipt is verified and each workflow instance per node triggers execution of a local transaction that transfers the money from the escrow account presented on ledger 1020 nodes as 1021a, 1022a, 1023a, and 1024a to the account presented on ledger 1020 nodes as 1021c, 1022c, 1023c, and 1024c belonging to the entity that controls nodes 1023 and 1031 and has made the payment to the recipient account during the previous step.

On completion of the local transaction, every node of ledger 1020 sends a notification message to all nodes of ledger 1030 to update the content of the affected accounts. Ledger 1030 processes these messages by adjusting on the nodes of ledger 1030 the content of respective copies of the accounts on ledger 1020 affected by the completed local transaction. Before the transaction is started, all nodes of ledger 1020 already have a copy of the receipt signed by the recipient. Once node 1021 ledger 1020 receives the signed receipt, it sends a message 102113 containing the

signed receipt to node 1013 of ledger 1010. After node 1013 of ledger 1010 receives message 102113, it is ready to request reimbursement with the paid amount plus the stipulated fee.

Fig.11 is block diagram of a step of an interledger transaction between Ledger1 and Ledger3 - Ledger1 payment of the money from the escrow account to the account of business entity that has a node in Ledger1 and a node in Ledger2 and sending notifications to Ledger2. On the diagram, ledger 1110 is Ledger1, ledger 1120 is Ledegr2, and ledger 11130 is Ledger3. Nodes 1113 and 1121 are operated by the same business entity. This business entity is the one whose money held on escrow was paid during the previous step and it must be reimbursed with the paid amount plus the stipulated fee. To make this happen, account 1013c on node 1113 of ledger 1110 initiates execution of the last step of the workflow instance that controls the amount in the escrow account presented on ledger 1110 nodes as 1111a, 1112a, 1113a, and 1114a. The execution of this step is initiated with a message containing the signed payment receipt and requesting in exchange payment of the amount on escrow. Once all nodes of ledger 1110 have received the encoded payment receipt, the receipt is verified and each workflow instance per node triggers execution of local transaction that transfers the money from the escrow account presented on ledger 1110 nodes as 1111a, 1112a, 1113a, and 1114a to the account presented on ledger 1110 nodes as 1111c, 1112c, 1113c, and 1114c belonging to the entity that controls nodes 1113 and 1121 and has made the payment to reimburse the entity that made the payment to the recipient account. Before the local transaction was executed, on all nodes of ledger 1110 the account that initiated the interledger ledger transaction has the signed receipt verified and archived. On completion of the local transaction, every node of ledger 1110 sends a notification message to all nodes of ledger 1120 to update the content of the affected accounts. Ledger 1120 processes these messages by adjusting on the nodes of ledger 1120 the content of respective copies of the accounts on ledger 1110 affected by the completed local transaction.

Editorial Note

2016101976

There is one page of claims

Open Network of Permissioned Ledgers

Claims

1. A system providing delivery of messages between two computers or between a computer and plurality of computers that is tolerant to a network link partition.

2. A method of providing detection, neutralization, and recovery of a stop-faulty process on a computer of a cluster of computers.

3. A cluster of computers ensuring high-availability of horizontally scaled data by performing highly available deterministic transactions.

4. A digital ledger with known number of distributed replicas with atomically consistent data, modified with deterministic transactions executed on each replica under sequential order established with an asynchronous leaderless Byzantine agreement algorithm.

5. A multitude of digital permissioned ledgers integrated in a network with open membership for execution of interledger transactions with safety and liveness guarantees that are equivalent to the guarantees of the participating permissioned ledger with weakest guarantees.

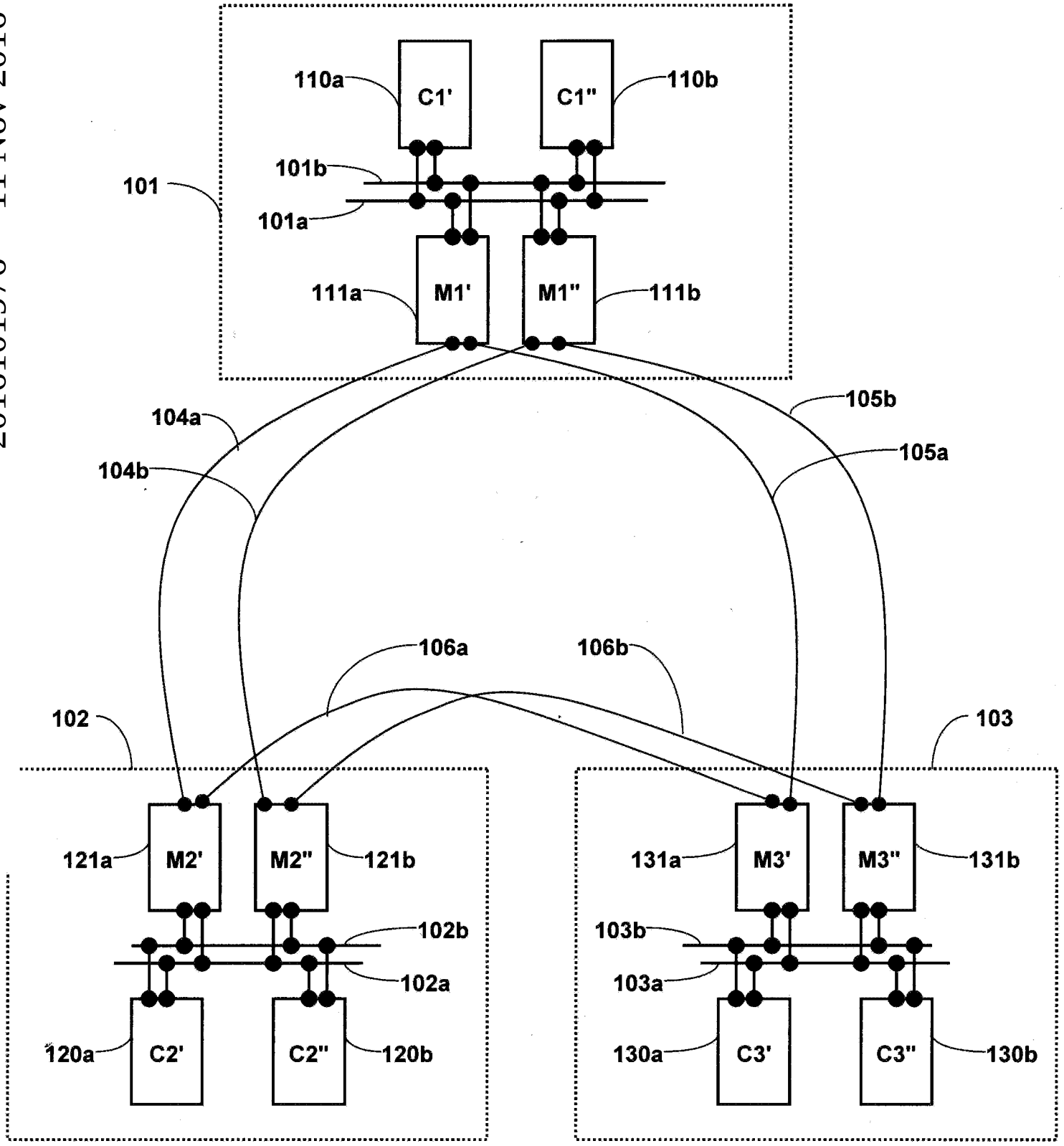
110a
C1'
C1"
110b
101b
101
101a
111a
M1'
M1"
111b
104a
105b
104b
105a
106a
106b
102
103
121a
M2'
M2"
121b
131a
M3'
M3"
131b
102b
103b
102a
103a
120a
C2'
C2"
120b
130a
C3'
C3"
130b

Fig. 1

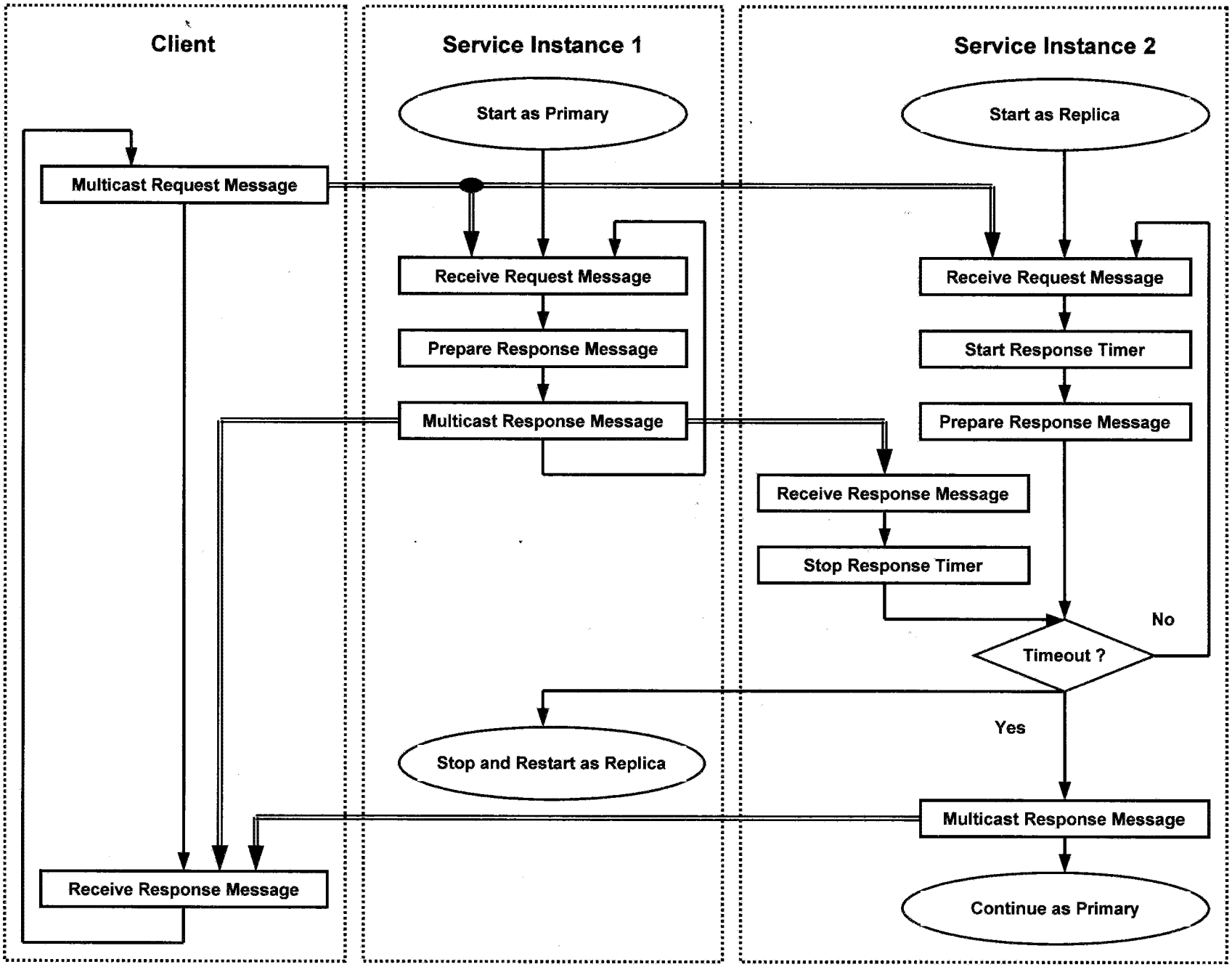
Client
Service Instance 1
Service Instance 2
Start as Primary
Start as Replica
Multicast Request Message
Receive Request Message
Receive Request Message
Prepare Response Message
Start Response Timer
Multicast Response Message
Prepare Response Message
Receive Response Message
Stop Response Timer
No
Timeout ?
Yes
Stop and Restart as Replica
Multicast Response Message
Receive Response Message
Continue as Primary

Fig. 2

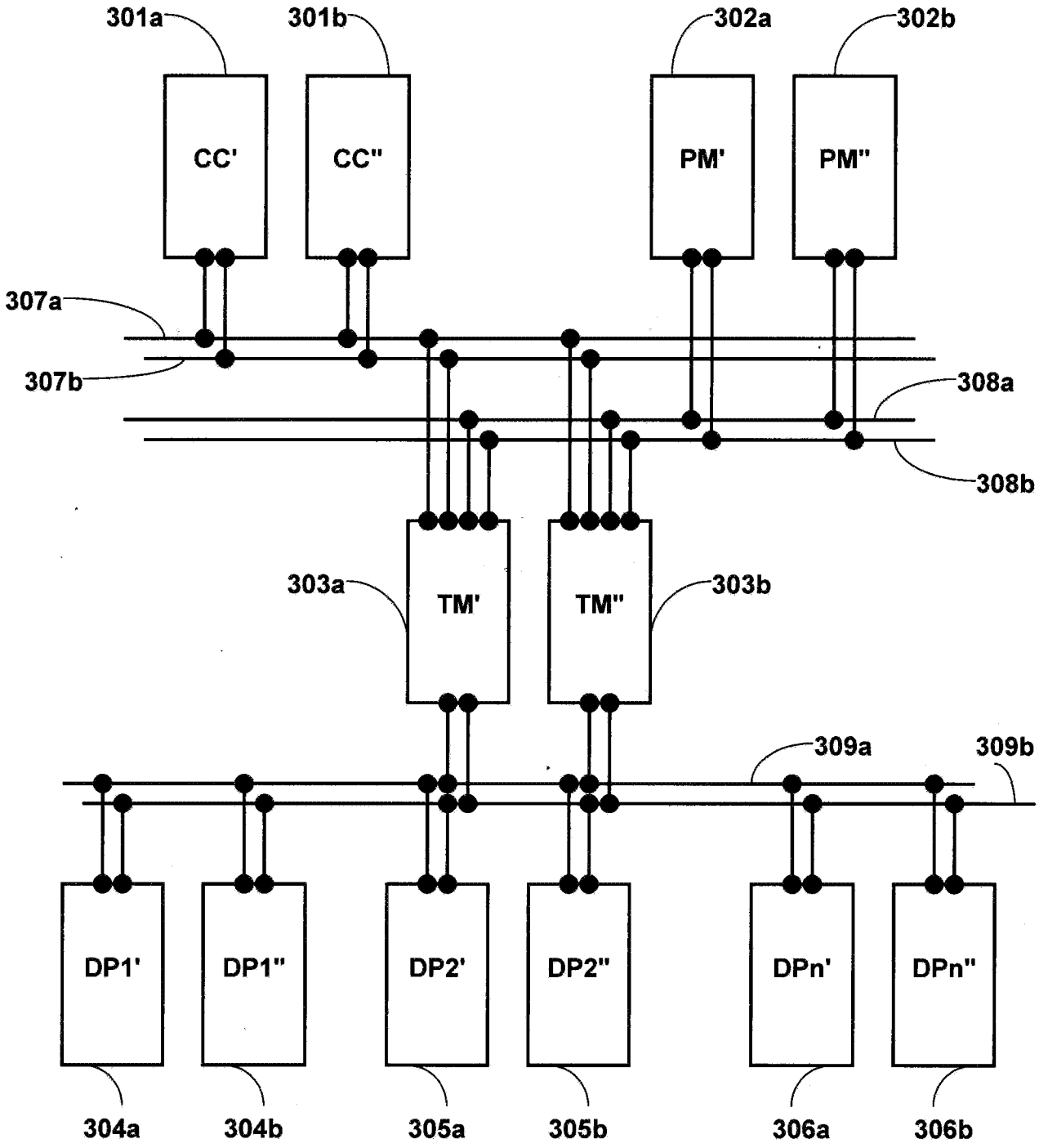
301a
301b
302a
302b
CC'
CC"
PM'
PM"
307a
307b
308a
308b
303a
TM'
TM"
303b
309a
309b
DP1'
DP1"
DP2'
DP2"
DPn'
DPn"
304a
304b
305a
305b
306a
306b

Fig. 3

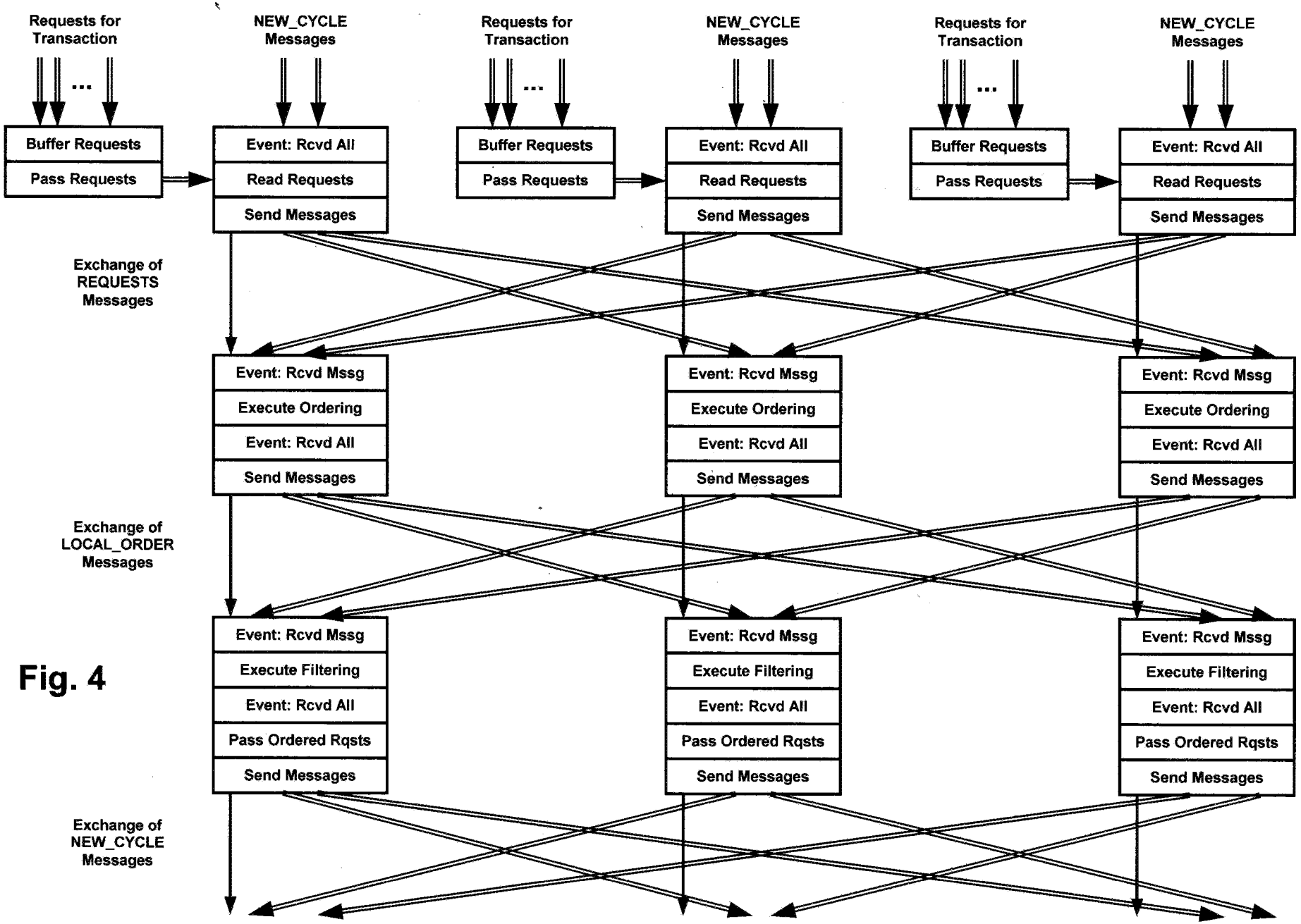
Requests for Transaction
NEW_CYCLE Messages
Requests for Transaction
NEW_CYCLE Messages
Requests for Transaction
NEW_CYCLE Messages
Buffer Requests
Pass Requests
Event: Rcvd All
Read Requests
Send Messages
Buffer Requests
Pass Requests
Event: Rcvd All
Read Requests
Send Messages
Buffer Requests
Pass Requests
Event: Rcvd All
Read Requests
Send Messages
Exchange of REQUESTS Messages
Event: Rcvd Mssg
Execute Ordering
Event: Rcvd All
Send Messages
Event: Rcvd Mssg
Execute Ordering
Event: Rcvd All
Send Messages
Event: Rcvd Mssg
Execute Ordering
Event: Rcvd All
Send Messages
Exchange of LOCAL_ORDER Messages
Event: Rcvd Mssg
Execute Filtering
Event: Rcvd All
Pass Ordered Rqsts
Send Messages
Event: Rcvd Mssg
Execute Filtering
Event: Rcvd All
Pass Ordered Rqsts
Send Messages
Event: Rcvd Mssg
Execute Filtering
Event: Rcvd All
Pass Ordered Rqsts
Send Messages
Exchange of NEW_CYCLE Messages

Fig. 4

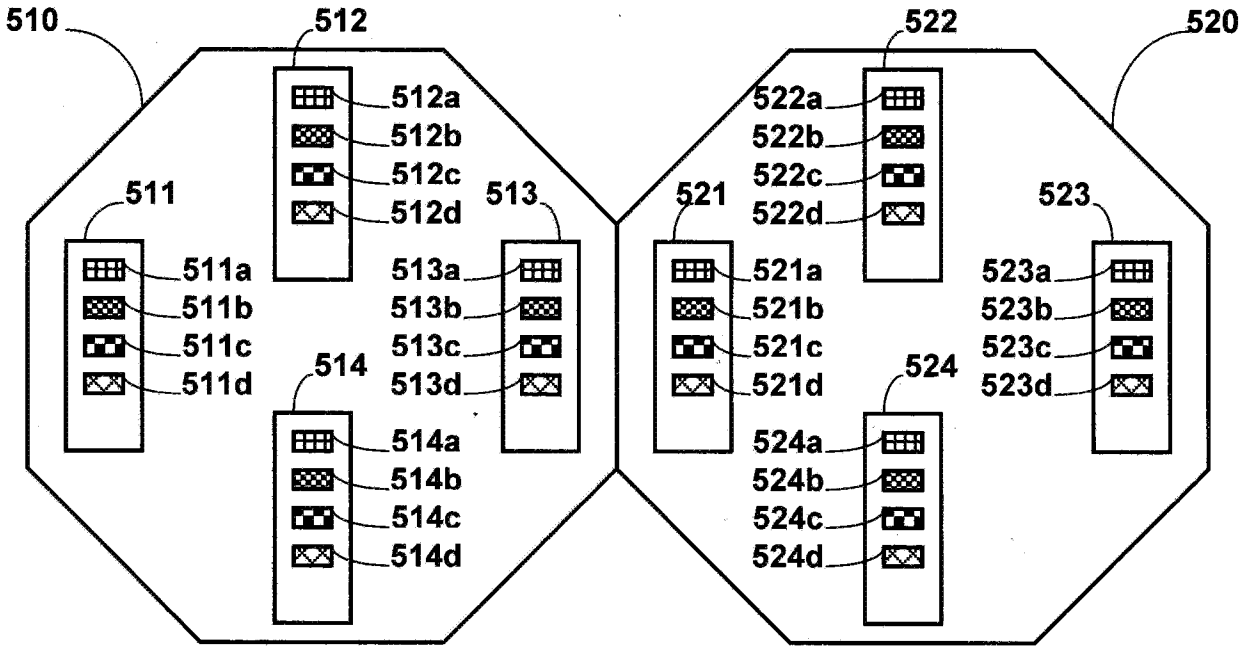
510
512
522
520
512a
512b
512c
512d
522a
522b
522c
522d
511
513
521
523
511a
511b
511c
511d
513a
513b
513c
513d
521a
521b
521c
521d
523a
523b
523c
523d
514
524
514a
514b
514c
514d
524a
524b
524c
524d

Fig. 5

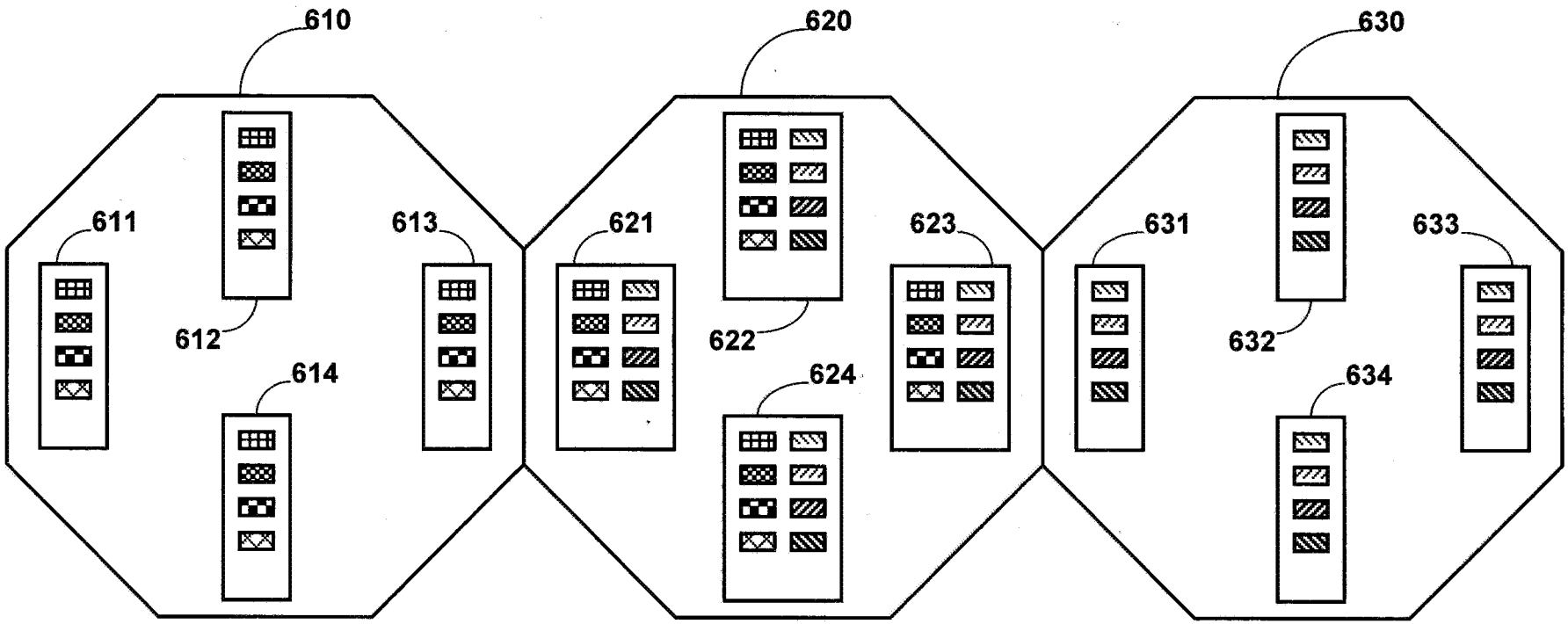

Fig. 6

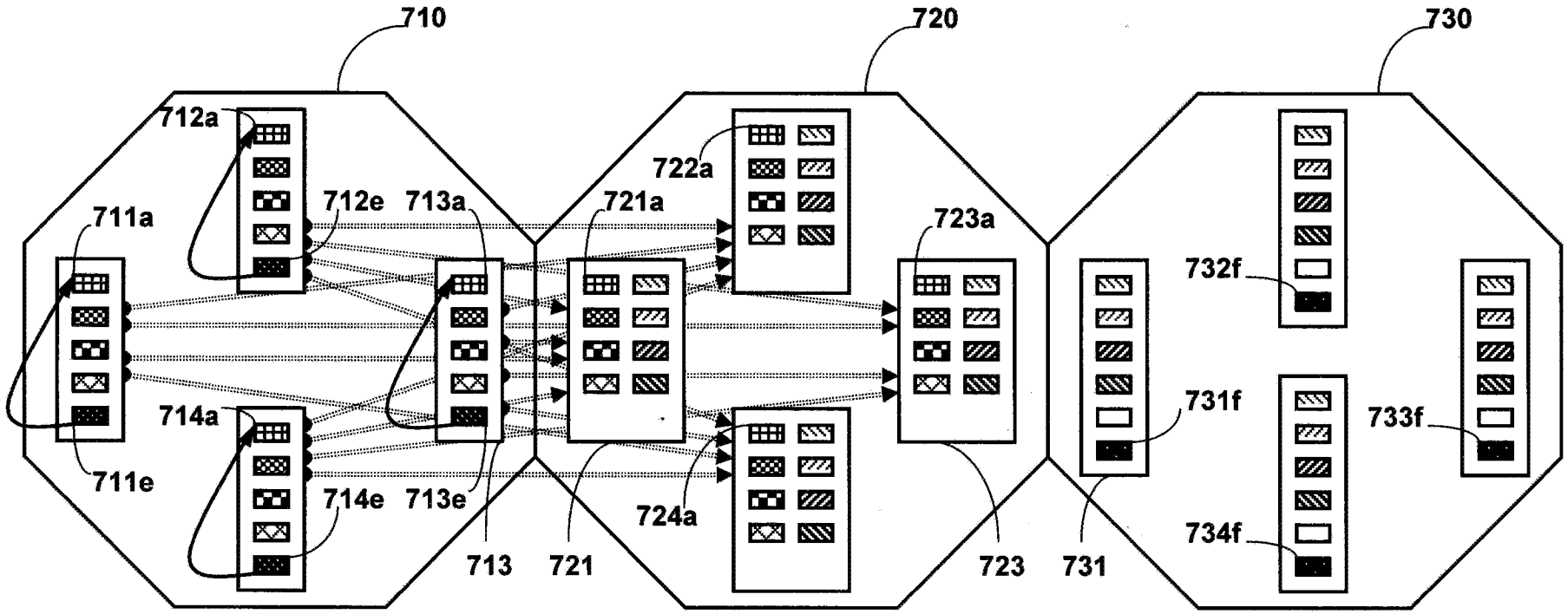
710
720
730
712a
711a
712e
713a
722a
721a
723a
732f
731f
733f
714a
711e
714e
713e
724a
734f
713
721
723
731

Fig. 7

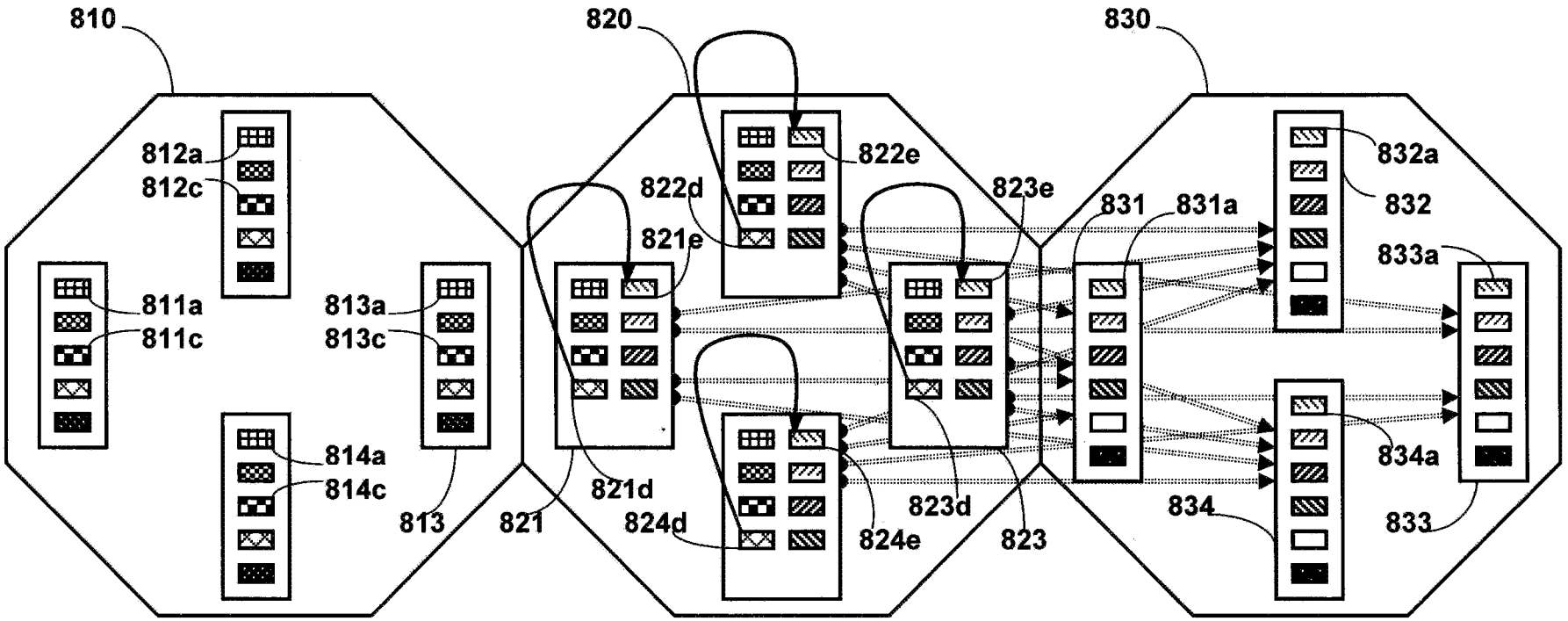
810
820
830
812a
812c
811a
811c
813a
813c
814a
814c
813
821
821d
824d
822e
822d
821e
824e
823e
823d
823
831
831a
832a
832
833a
834a
834
833

Fig. 8

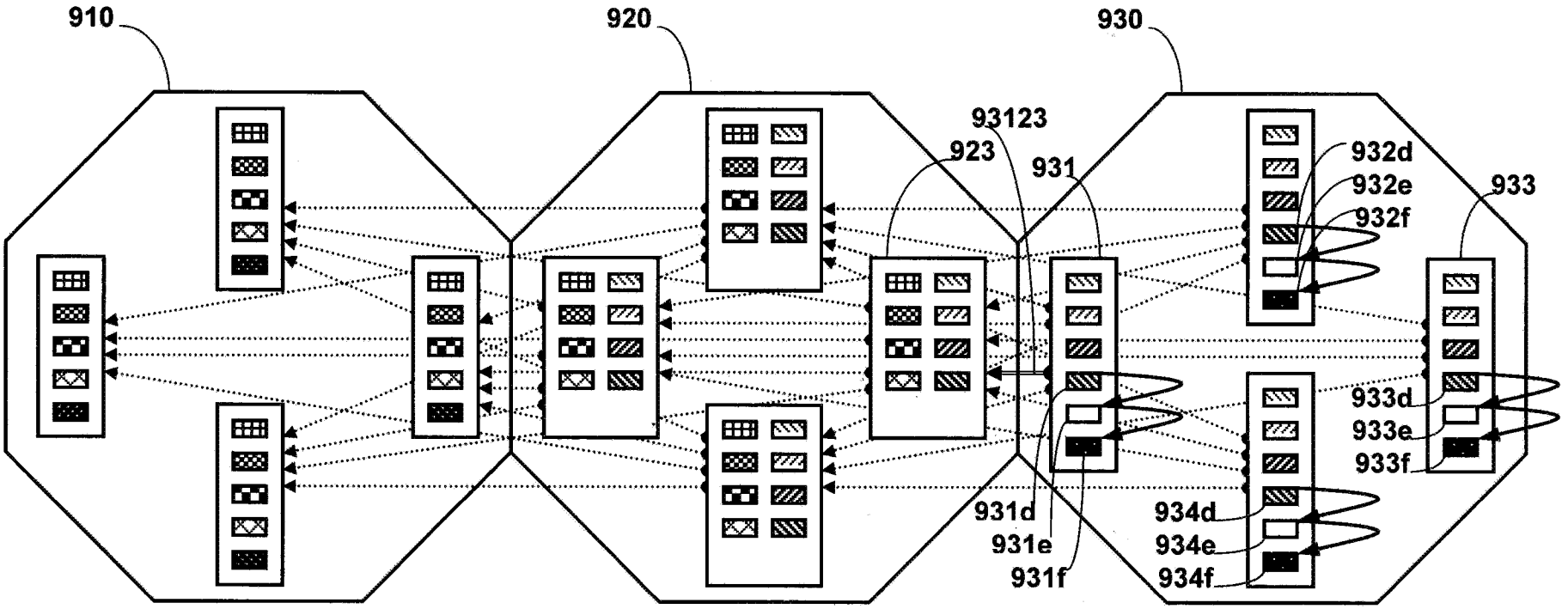
910
920
930
93123
923
931
932d
932e
932f
933
933d
933e
933f
931d
931e
931f
934d
934e
934f

Fig. 9

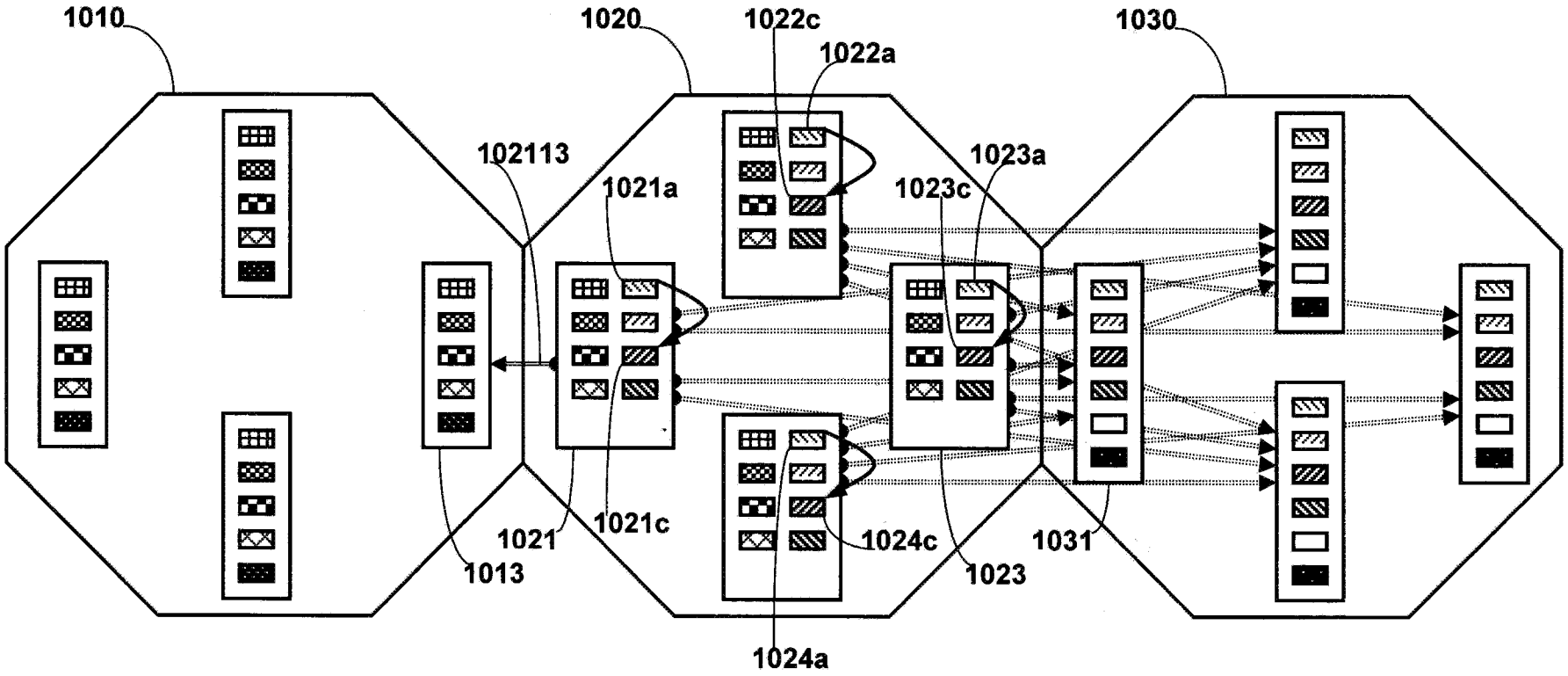
1010
1020
1022c
1022a
1030
102113
1021a
1023c
1023a
1021
1021c
1024c
1031
1013
1023
1024a

Fig. 10

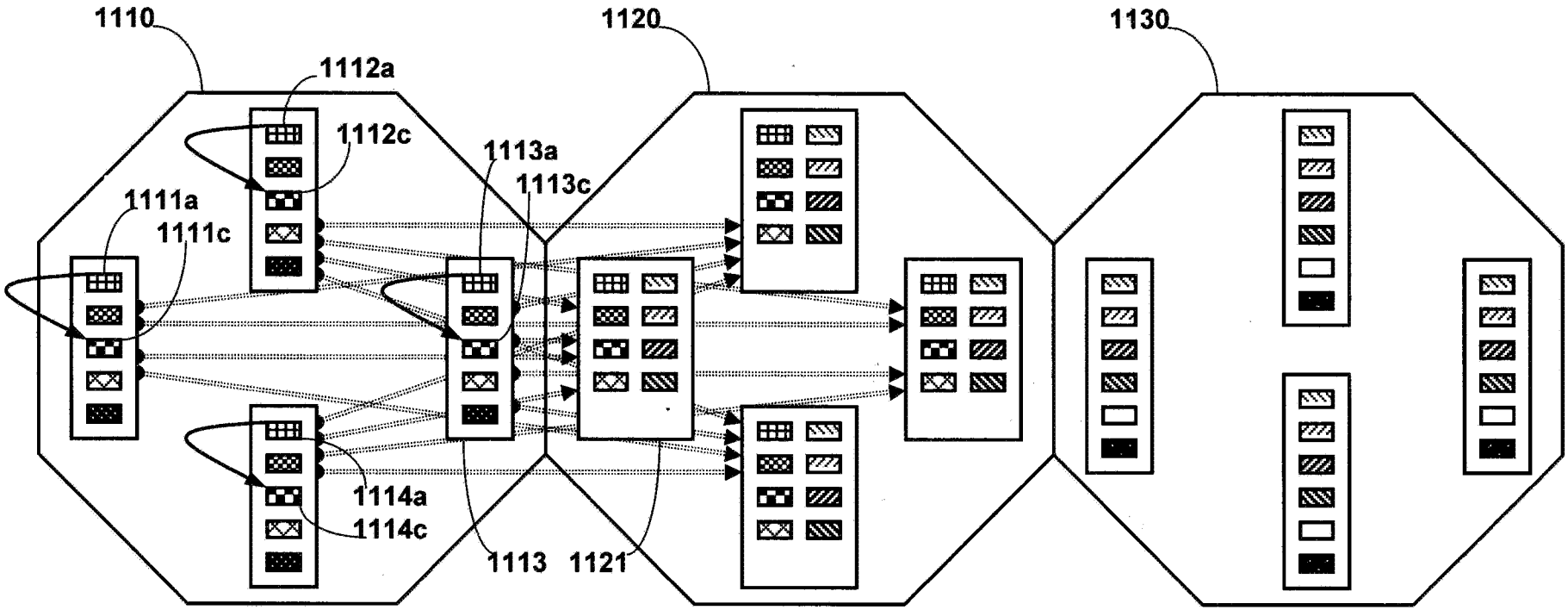
1110
1112a
1112c
1113a
1113c
1111a
1111c
1114a
1114c
1113
1121
1120
1130

Fig. 11