Atomcoin and Atom Nodes: The Basis of Atomchain P2P Network

Okpara Okechukwu D.
okpara.net@gmail.com
www.okpara.net
01-05-2022

Abstract:
This paper provides an architectural overview of the maiden release of atomchain, the atom big bang. Distributed computations require agreement between a set of nodes in an atom network. These nodes achieve agreement within a given timeframe via atomchain consensus protocols. The atom network can be used in creating, transferring, and trading of digital assets. The atom network presented in this paper is considered as being almost 100% Byzantine Fault Tolerant.

Keywords: web3, atomchain, atomcoin, Atom Field, Atom Network

Note:
To help you better understand this page, we recommend you read the Atomchain Wallet Whitepaper, https://okpara.net/AtomWallet.pdf

Disclosure:
The information in this paper is preliminary and is subject to change anytime in the future.

1. Introduction

Atomchain networks are scalable, customizable, secure and decentralized. They can be used as a permissioned/private and/or as a permissionless/public environment for building decentralized applications.

Classical consensus protocols rely on all-to-all communication, hence they can have low confirmation latencies and high throughput – they neither scale to large numbers of participating nodes, nor become robust when node membership changes are highly dynamic. Thus classical protocols have the tendency of making the distributed systems to be permissioned, and mostly statically deployed. On the other hand, atomchain consensus protocols, referred to as "**bonds**" are based on lightweight network sampling mechanisms. Bonds achieve low latency and high throughput without needing to agree on the precise membership of the nodes. Bonds are created when nodes repeatedly sample the atom network. Bonds of atom fields are regarded as **strong** bonds. Atomic reactor nodes sample molecule nodes repeatedly until convergence (in an atomic molecule node) is reached during normal operation.

In order to support the atom network, developers need the cryptocurrency, **Oke**, to create and run applications. **Atomcoin** is the smallest denomination of Oke, and it is used to pay for transaction fees and computational services. Atomcoin is pseudonymous in the sense that funds are not tied to real-world entities but rather to atomchain α addresses.

Atomcoin can be stored into α address and can be spent or received as part of transactions or atom generation. In other words, atomcoin is used for atomchain's coinbase α-transactions, and users can send atomcoins to other users from their wallets.

Atomcoin/Oke comes into existence by the validation of transactions or transaction events on the Atomchain platform, through an atom creation process called atomic "mining". Oke is used as the default for transaction events involving Ω addresses.
Coinbase of Ω addresses is called "**Okebase**", while that of α addresses is called "**Atomcoin Base Coins or ABCs**".

Subunits

Multiplier /Precision /Exponent	Special/Common Name	SI Name	Note
10^{-9}	atomcoin	nanooke	Smallest unit
10^{-6}	aku	microoke	
10^{-3}	akuoke	millioke	
10^{0}	oke	Oke	Default for Ω addresses
10^{3}	okeaku	Kilooke, Koke	
10^{6}	okechi	Megaoke, Moke	
10^{9}	okechiaku	Gigaoke, Goke	Largest unit

Features of atomcoin include the following:

- Atomcoin transactions are recorded and verified on an atomchain.
- There are multiple ways for an individual to obtain atomcoins.
- It can be purchased on an exchange using a fiat currency (under the symbol XAX, or any proposed symbol with prefix, "XA-" such as XATOM, XATC and so on. Note that the nomenclature used here further guarantees conformity with ISO 4217 currency standard.
- It can be transferred to you from another person or entity.
- It can be earned from atomic mining.

In order to hold atomcoin, you must have an Atomchain wallet, which can be downloaded and set up onto a computer, smartphone or other mobile device.

Each Atomchain wallet stores an individual's private key which allows the wallet owner to sign transactions that send atomcoins to other parties.

Atomcoin is a critical component to keeping the Atomchain platform growing and evolving in the digital asset environment.

2. Atom Wallet Address Codes

α addresses normally exist as pure payment objects, while Ω addresses exist as state objects which may or may not be used as payment objects. α addresses may be controlled by their own αaddress code called the **Shell Code**.

Just as in Bitcoin blockchain, Shells are unable to know and handle the transaction amount inside, therefore, one needs to sum up the amounts of all the α transactions sent to the same α-address in the atomchain in order to know the balance of the α-address in an atom wallet; and they are unaware of anything happening outside the program stack. Hence, they cannot access the atomchain data.

For security reasons, some shell codes are absent or unusable due to likelihood of vulnerabilities, or could lead to errors and out-of-control operations. As a result, Shell codes are deliberately non-turing complete. For instance, there are no loops (or loopholes) with shells.

To verify the transaction ID (TXID) or the raw transaction data of a given α transaction (raw_α_trans), one calculates the double SHA256 hash of the raw α transaction data which should match the transaction ID.

TXID == SHA256(SHA256(raw_α_trans)

A simple Python script for the implementation of the above code is shown below:

```
importhashlib
importbinascii

def doubleSha256(hex):
bin = binascii.unhexlify(hex)
   hash1 = hashlib.sha256(bin).digest()
   hash2 = hashlib.sha256(hash1).digest()
returnbinascii.hexlify(hash2)

# sampleraw α transaction data
sample_raw = 'raw_α_trans'

# Double-SHA256 hash
αhash = doubleSha256(sample_raw)

# Using big-endian hexdec
txid = binascii.hexlify(binascii.unhexlify(αhash)[::-1])
txid = str(txid,"ascii")

# Compare with α transaction ID
chkTxID = "TXID";
print("α-address of atom network with transaction exists:
"+str(txid==chkTxID))
print("Transaction_ID:\n   "+txid)
```

If aΩ address is controlled by home, it is said to be **at rest**; but if it is controlled by their ownΩ address code (called **Orbit Code**), they are said to be **in orbit**. In fact, an Ω address must either be *at rest* or *in orbit.*

A Ω address in orbit has no home. It is controlled by the logic of its orbit code. Any execution in the orbit code is triggered by transaction events or messages produced by Ω address at rest. Ω addresses in orbit cannot initiate transaction events by themselves since they do not have homes.

In Atomchain, Ω address objects (made of 20-bytes or 5 words of 32-bits) have state transitions which are direct transfer of value and information between them.

Orbit codes may be similar in functionality with “**smart contracts”** in some blockchains.

AnΩ address code normally contains four(4) fields:

1. The **event count**, a counter used to make sure each transaction can only be processed once. It is a scalar value equal to the number of transactions sent from this address or, in the case of accounts with associated orbit code, the number of orbit-creations made by this account.
2. The code’s current **atomchain balance**. It is a scalar value equal to the number of OKE owned by this Ω address.
3. The **orbit code**, if present. Orbit codes are written in (ISO/ANSI) Standard C++.
4. The object’s **storage**, which is empty by default.

Ω address state objects, whether at rest or in orbit are only added to the **State trie** once a successful transaction (that consumed energy and is inside an atom) has taken place in relation to that address. This is to guard against malicious attackers continuously creating new Ω addresses in order to bloat the state trie. A trie (or prefix tree) is a particular kind of search tree, where nodes are usually keyed by strings. (See Atomchain Storage Whitepaper, https://okpara.net/AtomStorage.pdf)

There are two types of αaddress ownership in atomchain:

1. Normal αaddresses: these are controlled by homes, and
2. Shell αaddresses: these are controlled by their **shell code**.

There are two types of Ω address objects ownership in atomchain:

3. Ω addresses at rest: these are controlled by homes, and
4. Ω addresses in orbit: these are controlled by their **orbit code**.

Sample Orbit Code is shown in Appendix A (See the atomocore.json file section)

An orbit code is a model consisting of three sets—input, output, and state of the model—and three functions—input transform function, state transition function, and output function. This model provides an improved level of performance and safety for the atomchain since it consists of codes that are in line with current industry performance and security standards.

3. Transactions on Atomchain Network

In order to prevent accidental or malicious infinite loops or other computational wastage in code, each transaction must set a limit to how many computational steps of code execution it can use. The unit of this computation is "**Atom energy**" (or simply, "**energy**"). By default, there is an incremental fee of **10 energy** for every byte in the transaction data. This fee is necessary so that an attacker will pay proportionately for each resource that they consume, including computation, bandwidth and data storage.

A transaction is a signed data package or process initiated by an Ω address at rest or by a normal α address. Transactions require energy to be consumed and must be mined (or inserted in an atom) to become valid. Each transaction has an ID, the TXID.

In messages that are to be sent from an Ω address, a submitted Ω address transaction contains the following:

- The **recipient** of the message; the receiving Ω address (if this address is at rest, the transaction will transfer value. If this address is in orbit, the transaction will execute the orbit code).
- A **signature** identifying the sender. The sender's identifier is generated when the sender's home signs the transaction and confirms the sender has authorized this transaction.
- A **value or amount**; the amount of atomcoin to transfer from the sender to the recipient - atomcoin is the smallest denomination of **OKE**.
- an optional **data** field, for including arbitrary data.
- An **energyMAX** value, representing the maximum number of computational steps the transaction execution is allowed to take. This is also the maximum units of energy that can be consumed by the transaction
- An **energyPRICE** value, representing the fee the sender pays per computational step. Usually, it is calculated average of recently successful transactions.
- A **maxCreatorEnergyFee** value, the maximum amount of energy to be included as a tip to the atom creator.
- A **maxFeePerEnergy** value, the maximum amount of energy willing to be paid for the transaction

The first three are normal fields expected in a typical cryptocurrency system. The data field has no function by default. **energyMAX** and **maxCreatorEnergyFee** determine the maximum transaction fee paid to the atom creator. Every atomchain transaction event requires payment of a fee, which is collected by the miners (or atom creators) to validate the transaction. This fee is charged in a virtual currency called **energy**, which is paid with Oke/atomcoin as part of the transaction event. The value of Oke is represented internally as an unsigned value denominated in atomcoin.

The use of both **energyMAX** and **energyPRICE** fields in Atomchain helps in solving the problem of denial of service attacks through malicious (or bugged) scripts that run forever. Energy in Atomchain is like “gas” in Ethereum blockchain.

As an example, if for a particular transaction, the energy (used by the transaction) limit was 21,000, and the energy price was 14 atomcoins, and maxCreatorEnergyFee of 7 atomcoins. Then, (14 + 7) * 21,000 will give the actual transaction fees: 441,000 atomcoins or 0.000441OKE (pronounced as, “oak”). This transaction fee goes to the atom creator (or miner), that has validated the transaction.

The transaction object may look like this:

```
{
from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
energyMAX: "21000",
maxFeePerEnergy: "300",
maxCreatorEnergyFee: "10",
nonce: "0",
value: "10000000000"
}
```

We elucidate a basic operational mechanism thus: a user creates a transaction - with an application running on atomchain VM – in a message and sends it to a validating (or an atomic creator) node participating in the consensus (or bonding) procedure. The message is then propagated out to other nodes (such as to an element node) in the network. If the user is malicious and uses a multispend (or doublespend) attack, the node randomly selects a small subset of its atomic reactor nodes and queries which of the conflicting transactions the nodes think is the valid one. If the querying element node receives a supermajority response in favour of one transaction, then the element node changes its own response to that transaction. Every molecule node in the network of that atomic (reactor) node repeats this process until that particular atom field comes to consensus on one of the conflicting transactions.
Atom Wallet is an application that let a user interact with their α addresses and Ω addresses. It lets users check their address objects’ balances and send transactions. The orbit code of an Ω address in orbit governs the state behavior of that Ω address object.An orbit code deployment transaction is an Ω address-based transaction without a 'to' address, where the data field is used for the orbit code. An orbit code execution transaction is a transaction that interacts with a deployed orbit code. In this case, 'to' address is the orbit code’s Ω address.

A transaction event is a normal transaction from one wallet to another. This includes all Payment Transactions.

Atom Wallet α-addresses start with the lowercase prefix “atom,” while Ω-addresses start with the uppercase prefix “ATOM”.

4. Inside atomcore.json State File

atomcore is the start of the atomchain, atom 0; and atomcore.json is the atomcore state file that defines it. It is the default settings/configurations for the atomchain network.

```
{
        "setup":{
                "atomID":0,
                "atomHome":0,
                "atomState": {
                "ASH":"…",
                "DNA":"…",
                "PNH":"…",
                "SIG":"0",
                "MRT":"0"
                },
                "atomDump":"…"
        },
    "atomBank": {},
    "okebase": {
            "αaddress":"balance",
            "Ωaddress":"balance"
        },

        "ΩTXevent":1,
                "timestamp":"0x0",
                "okebase":"0x00…00",
                "energyMax":"0x1ffffffffffff",
                "version":"0.1.0",
                "elevMode":"0x0",
                "atoms": { },
                "periods": {"name":"atom" },
                …
}
```

A custom atomchain.json state file

This file defines the rules for the atomchain itself. In its **setup**/configuration section it contains: **ASH** (Atom creator's node SHA Hash), **PNH** (Previous Nucleus Hash, a 256-bit hash of the entire parent/previous Nucleus. It is a pointer to the encapsulated atom, thus effectively building the **bond of atoms**. PNH of atomcore is set to zero), the **DNA** (Decentralization Number of the Atom), **SIG** (atom's Significance, this is a boolean value to handle signed/unsigned type or parity).**MRT** (Merkle Root), this is constructed using all the TXIDs of transactions in the atom. The ABCs transaction's TXID is always placed first. If an atom only has an ABC transaction, the ABC TXID is used as the Merkle root hash.

If an atom only has an ABC transaction and one other transaction, the TXIDs of those two transactions are placed in order, concatenated as 64 raw bytes, and then SHA256(SHA256()) hashed together to form the Merkle root. If an atom has three or more transactions, intermediate Merkle tree rows are formed. The TXIDs are placed in order and paired, starting with the ABC transaction's TXID. Each pair is concatenated together as 64 raw bytes and SHA256(SHA256()) hashed to form a second row of hashes. If there are an odd (non-even) number of TXIDs, the last TXID is concatenated with a copy of itself and hashed. If there are more than two hashes in the second row, the process is repeated to create a third row (and, if necessary, repeated further to create additional rows). Once a row is obtained with only two hashes, those hashes are concatenated and hashed to produce the Merkle root. TXIDs and intermediate hashes are always in internal byte order when they're concatenated, and the resulting Merkle root is also in internal byte order when it's placed in the nucleus.

- **periods** is for specific network upgrades that will occur in the future at any *atom* (the atom number where the milestone or upgrade occurred). *name* to be used will be adapted from the **Periodic Table of Elements** starting with hydrogen. **atoms** contains the initial list of atomic (reactor) nodes with elevation modes in the format such as *"node@IPaddress:Port":elevMode*
- **setup** contains all the configuration parameters/variables and thresholds that control the atom network's basic operations. The values in the variables need to match the configuration information of any other node this atomcore has to interact with.
- **AtomID** protects the network from a **replay attack**– where an authorized attacker node acts as the original sender. It acts like an offset to prevent attackers from deciphering continuous values in the **atom network**.It is a unique identifier that allow only hashed transactions that are signed.
- **okebase or Atom's Base Coins (ABCs)** is the network coinbase account; which is where all atom rewards for the network will be paid. It can be set as an arbitrary address of 160-bits to which all rewards (in atomcoin) collected from the successful mining of this atom have been transferred. Its atomcoin-the native token of Atomchain-is the sum of the atom reward and the transaction fee. The terms "okebase" and "ABCs" can be used interchangeably.Okebase of atomcore usually contains pre-funded wallet addresses, and it allows defining a list of pre-filled wallets, needed to handle the "atomcoin pre-sale" period. Note that the first transaction in an atom must be aokebase transactionwhich should collect and spend any transaction fees paid by transactions included in the atom.
- **energyMax** is the total energy limit for all transactions included in an atom. It defines how large the atom size can be, and it is represented by an hexadecimal string. This limit impacts how much atomchain VM computation can happen within an atom. In short, this is the limit of energy cost per atom.
- **Version** is the AtomVMChainware version. Atomchain codebases will be released using three numeric identifiers, labeled "v.[0-9].[0-9].[0-100]", where the first number identifies major releases, the second number identifies minor releases, and the third number identifies patches. The first public release, codenamed **Atomic Big Bang**, is v. 1.0.0.
- **timestamp** is a scalar value equal to the reasonable output of Unix time() function at the atom's inception/creation.
- **elevMode** is an applied scalar value corresponding to the "reputation or visibility" level of the node during the atomic molecule pool selection process.
- **α address** is created similar to the way Bitcoin blockchain's native segregated witness(SegWit) addresses are created. But it starts with the prefix "ac".
- **Ωaddress** is created similar to the way Ethereum blockchain addresses are created.

- **Ωnonce:** this is a public viewable nonce which is increased by one every time a transaction is made, in order to prevent the same transaction being submitted more than once (in a multi-spend attack).

5. Atomchain Network a.k.a. Atom Network

Atomchain nodes communicate with one another through the Atomchain P2P network protocol, and by doing so; they guarantee the integrity of the network. Any node that misbehaves or tries to propagate incorrect information is quickly detected by the “honest” nodes and is disconnected from the network. Nucleic reactor and atomic reactor nodes may come with wallets (that is, they have wallet addresses). Listening nodes are publicly accessible, while nonlistening nodes may be hidden nodes, for instance, a node operating behind a firewall. The Atomic nodes are the backbone of the atomchain because they are responsible for storing and distributing copies of the entire atomchain ledger – or database.

The Network ID is usually the same as the atom ID in the atomcore.json state file. It is an integer number that isolates atomchain peer-to-peer networks. Connections between atomchain nodes will occur if and only if both peers use the same Atomcore and network ID; in other words, the atom ID in the atomcore.json file must be the same for all participating nodes in the network.

Atomchain can be public/permissionless or private/permissioned. An atomchain network is a private or isolated network if the nodes are not connected to the main atomchain network. Atomchain main atom network’s atom ID is 1. If you are intending to connect to your private atom network on the internet, it is best to choose a network ID that is not in use.

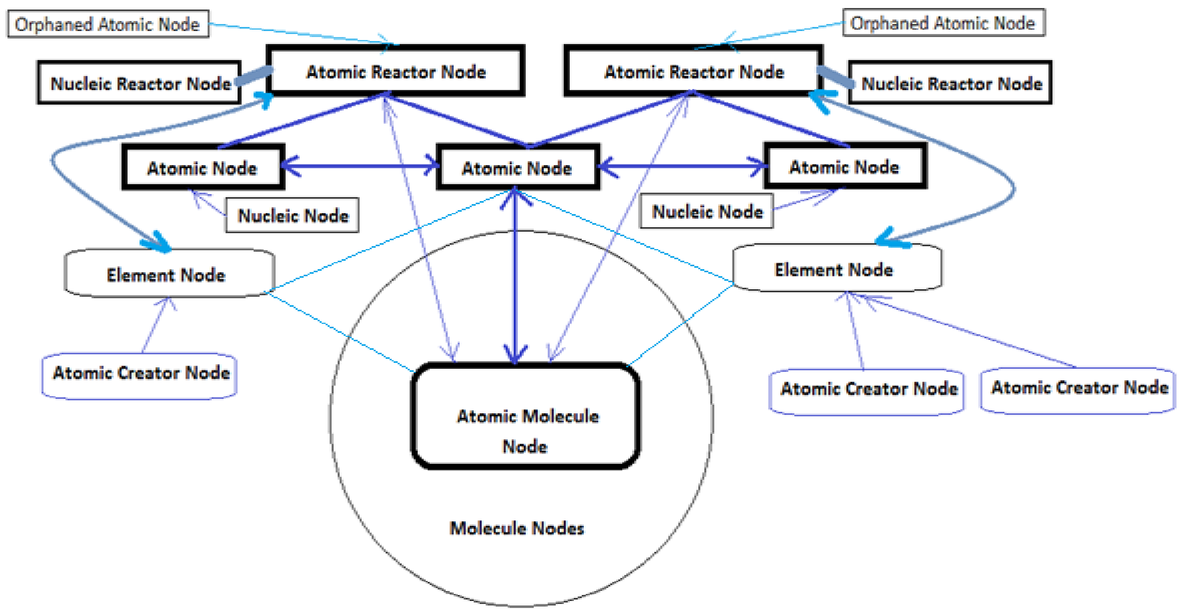

Figure 1. Atomchain Nodes

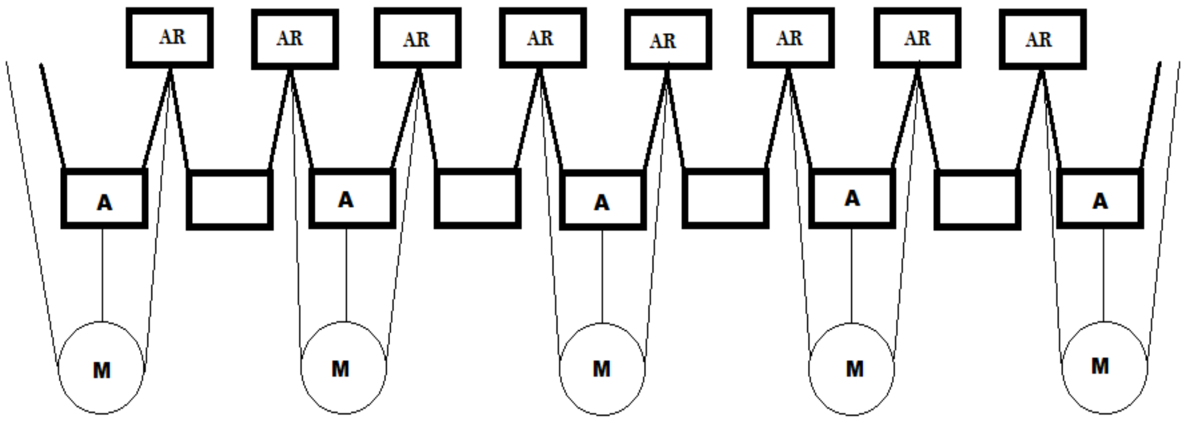

Figure 2. ‘A’ central Atomic Nodes, ‘AR’ Atomic Reactor Nodes, ‘M’ Molecule Nodes

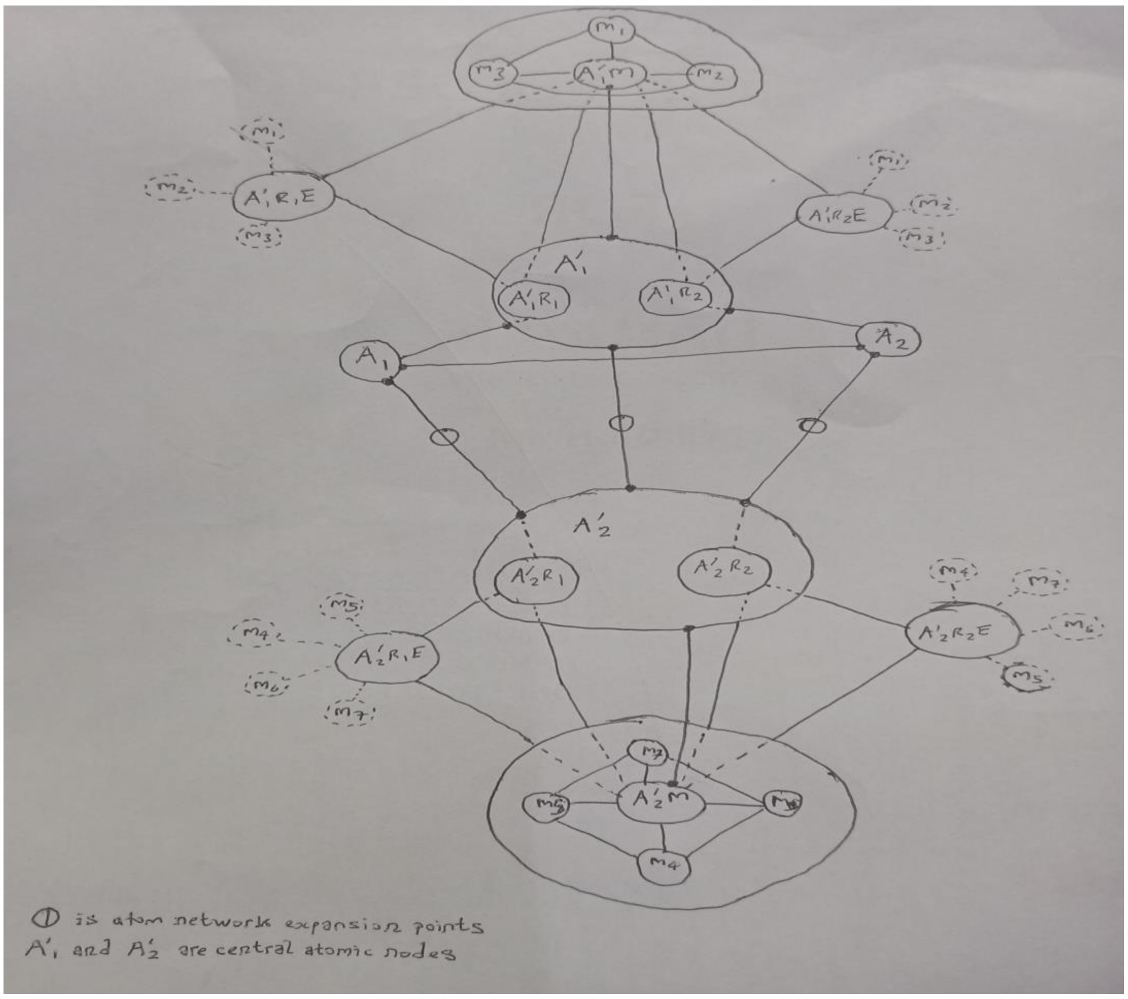

Figure 3. First Draft of AtomP2P Topology

Atomic Creator Nodes: these nodes create or "**mine**" atoms from their respective transaction pools (transpools). They are also referred to as atom "mining" nodes; similar to "miners" in blockchain.

Atomic Reactor Nodes: these atomic nodes are "**listening**," and publicly visible. It communicates and provides information to any other node that establishes a connection with it. They (including Atomic Nodes) can either mark invalid atoms (from element nodes) with the "destroy" marker before returning them, or completely discard atoms with invalid transactions. They are basically redistribution points or relays that act both as data sources and as connection bridges for the network.

Atomic Molecule Nodes: this is a node that is connected to most atomic (or atomic reactor) nodes at the point when an atom is to be added to the atomchain. Atomic (and atomic reactor) nodes actively monitor for this node. This node has a consistent distributed hash table of molecules.

Atomic Nodes: these are servers that host the entire atomchain. They are devices in the network that store and synchronize a copy of the network's entire atomchain history. They also validate atoms and maintain consensus. They are concerned with the general health of the network. Nodes **A** in Fig. 2 and **Node 10** in Fig. 3 is a central atomic node, one of the most resilient nodes of an Atom Network.

Nucleic Nodes: these nodes contain the nucleus of previous atoms. They depend on parent atomic nodes. These nodes download existing nuclei only, thereby saving users significant download times and storage spaces.

Nucleic Reactor Nodes: these are listening nucleic nodes that are publicly visible.

Element Nodes: these are indexing points for atom pools. They receive atoms from atomic creators, sends the atoms to atomic reactors for validation (that is, they query atomic reactor nodes). They also serves as points of node discoveries.

Molecule Nodes: these are candidate atomic creators, somehow placed in an elevation pool in a particular atom field.

Orphaned Atomic Nodes: these are mini-atomic nodes that save storage space for its users by "pruning or trimming off" older atoms in the atomchain starting from the atomcore. They first of all download the entire atomchain from an atomic reactor node, then they begin to delete atoms starting from the atomcore until they hold only the most recent transactions (up to a set size limit of the node operator).

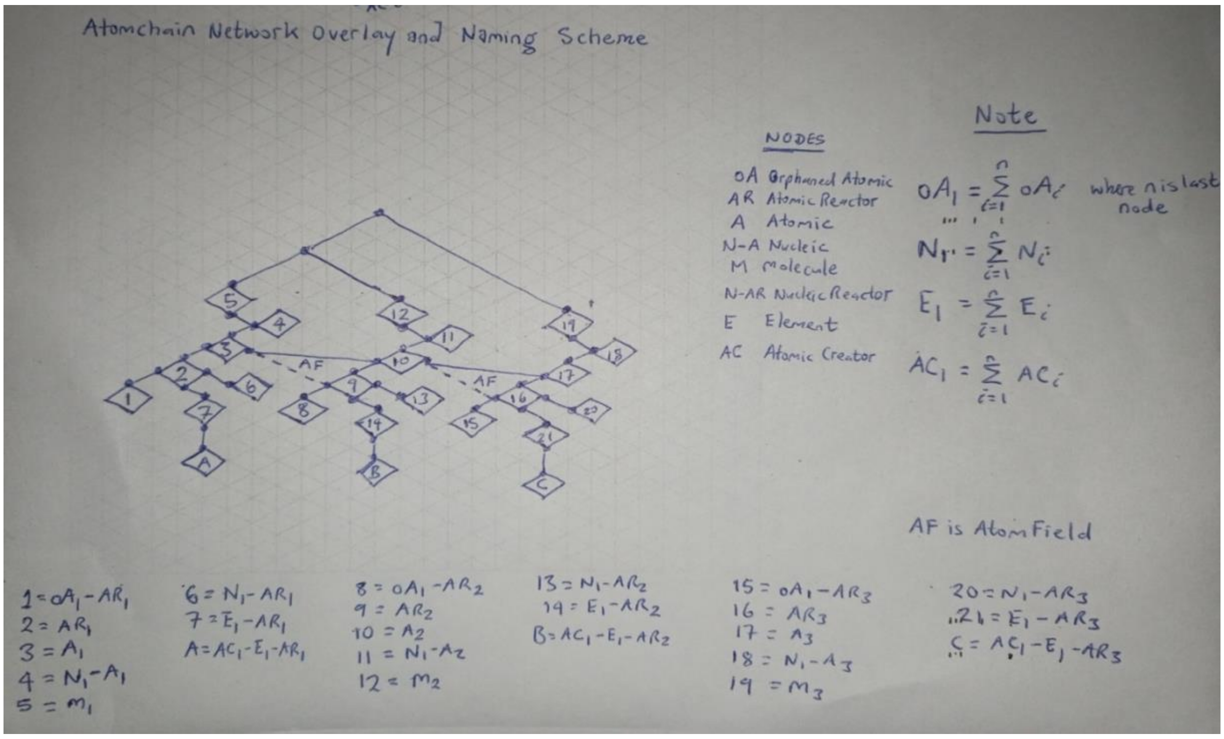

Figure 3. The Atomchain Network Overlay and Naming Scheme

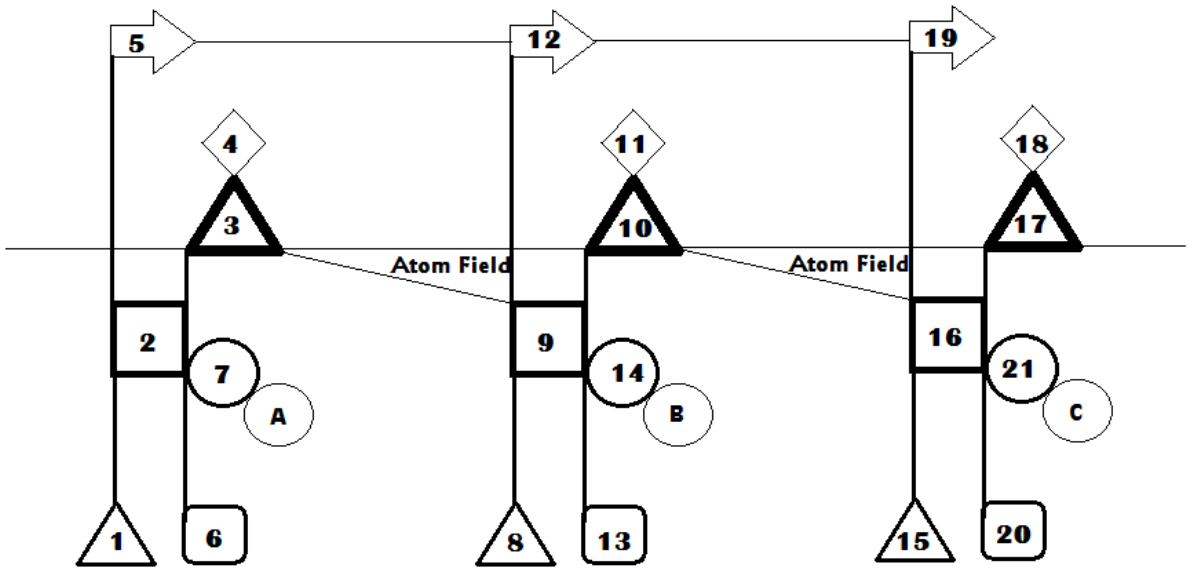

Figure 4. A simplified visualization of the Atomchain Network Overlay

At any particular point in time, an atom field (A’F) with a central atomic node (A’) having an atomic molecule node, A’M’, has two adjacent atomic nodes A’1 and A’2, and two atomic reactor nodes, A’R1 and A’R2. A’R1 has an element node A’R1E, and A’R2 has its own element node A’R2E. A’R1E and A’R2E nodes facilitate node discovery. A’M’ is in a set of A’ molecule nodes, {A’M’ (first), … , A’M’ (last) nodes}, where it is chosen as the candidate peer that will provide an atom – for the atomsphere. An A’M’ (including other atomic molecule nodes) help to keep molecule peers in sync with respect to the actual membership of their A’F – or molecule nodes pool, and also help in resolving any inconsistent routing state with other peers. An A’M’ (or any node for that matter) with elevation mode (eM) greater than zero (eM> 0), resets after it successfully adds an atom to the atomchain – that is any node that formed the atomsphere, is automatically adjusted to eM = 0.

A’R1E and A’R2E may connect to A’R1 and A’R2 respectively via a Proof-of-Stake protocol or mechanism.

Nucleic (Reactor) nodes contain only atom nucleus information without the transaction data to save bandwidth and storage space.

When an atomic node (such as A’) goes offline or leaves the atom network it becomes an orphaned atomic node (oA’) when it rejoins the network; its copy of atomchain is not recognized by the other active (atomic) nodes, A’1 and A’2. When A’ left the A’F, it is replaced with an A’M’ that is provided (or is agreed upon) by both A’R1 and A’R2. If an A’M’ is not available then A’ is replaced with an oA’ if present; if no oA’ exists, then A’R1 and A’R2 chooses the strongest and longest serving or living element node, (A’R1E or A’R2E). Detailed protocol specifications and technical documentation of Atom P2P is available to Atomchain developers on request from the author (okpara.net/?devID=NULL&key=α&hash=0x00/AtomP2P.pdf).

For a cluster with n Atomic Molecule nodes, when there are more of such nodes, a piece of (key, value) molecule node registration will only be 1/(n+1). (See Appendix B)

The process of all the nodes on an atom network agreeing that transactions are valid in the absence of a central authority is known as “achieving consensus”. Consensus refers to the rules by which an atom network operates and confirms the validity of information written in atoms.

A central atomic node is a computer in the atomchain network that stores and synchronizes a copy of the atom network’s entire atomchain history. These nodes can vote on proposed changes to the network. Central atomic nodes use Proof-of-Stake (PoS) consensus algorithm for the validation of transactions and enforcement of network rules on non-central atomic nodes in their atom fields. Non-central atomic nodes only serve to validate and record transactions.

Churn is the arrival and departure of peers to and from the overlay, which changes the peer population of the overlay. Overlay maintenance is the operation of the overlay to repair and stabilize the overlay routing state in response to churn. The overhead for overlay maintenance increases as the churn rate increases. It also increases proportional to the routing state maintained by each peer, which is in turn proportional to the size of the overlay and the degree of each peer. There are techniques to reduce churn itself, such as incentives for peers to stay connected to the overlay. In addition, newly joined nodes can be quarantined, treated as client-only nodes, either due to limited capacity or until the peer reaches a lifetime threshold. This relies on the peer lifetime distribution being heavy tailed, which has been found to occur in practice. [1]

The network's resiliency is such that it automatically checks and eliminates Byzantines – the proportion of actors in the network that can be offline or may behave outright maliciously.

The α- and Ω-operator alternating scheme [2] of the atomchain attempts to resolve or solve the problems that Casper FFG [3] and GRANDPA [4] protocols attempts to eliminate.

6. Atomchain Transpool

• When a user creates a transaction on atomchain, the unconfirmed transaction is sent to the transpool.

• All atomic (reactor) nodes have access to the transpool. They validate the unconfirmed transactions in the transpool. When a node on atom network receives a new transaction, it verifies the transaction according to the protocol's prevailing validation rules. If a transaction violates the rules (for instance, having an invalid signature or hash), the transaction is not forwarded to other peers, and the transaction is marked with the "destroy marker". On the other hand, if a transaction was successfully validated, it is then added to the transaction sets in the transpool.

• Atomic creators (or miners) select a certain number of validated transactions from the transpool and try to apply the elevation mode consensus. A collection of validated transactions forms an atom. Atom creation rewards consist of "atom reward" (which is by default 10 OKE) and the transaction costs (usually in atomcoins).

• Atomic creators choose which transactions to put in an atom from the transpool according to their fee rate, which is transaction fee divided by the transaction size.

• A transaction with a high fee rate is giving a higher priority than that with a lower fee rate. Prioritizing in this way ensures maximum profitability for the atomic creator. Maximum atom size is 1MB; but transaction sizes vary. If the size of the transpool is very large, say 100MB, it can indicate how busy the atom network is at that moment, and how long one has to wait for the confirmation of a transaction.

• Transactions continuously crisscross the atom network, creating digital footprints that require careful tracking and management to maintain the integrity and reliability of the underlying atomchain.

7. Nodes Configuration File

Peers are called nodes. All peers are supposed to be connected to the atomchain **overlay** network. If a connection or pathway exists from one peer to another, it is a part of this overlay network. The main overlay network connecting atomic reactors and the main atomic nodes is referred to as the **Atom Field**.

A node configuration file should have parameter/variable features such as:

maxtranspool – which should limit the transaction pool in megabytes. Used in reducing memory use on memory-contrained nodes.

maxconnects – which should set the maximum number of nodes from which to accept connections. Lower values implies reduced bandwidth consumption.

transindex – which should maintain an index of all transactions. A complete copy of the atomchain that allows one to programmatically retrieve any transaction by ID.

maxreceivebuffer/maxsendbuffer – which should limit per-connection memory buffer to a given multiple of 1000 bytes. Used on memory-constrained nodes.

minrelaytransfee – which should set the minimum fee rate for transactions to be relayed. This can determine which transactions are to be rejected from the transpool and not relayed.

If a new node or validator attempts to join the atomchain, it will use the atomcore.json file as the starting point in recreating the history of the atomchain in order to synchronize with the existing network (or atom fields).

The following features are expected of atomchain nodes – or **Atom Nodes** for short.

- Each molecule (or atomic creator) node must have at least one account address, if it is to receive atom rewards – from mining.
- Nodes must have an unused listening port (for instance, port 40404) associated with its IP address for communication.
- Nodes must have a data filesystem or directory/folder where the atomchain data will be located or stored.
- Nodes must have an identity according to the atomchain naming scheme.
- Nodes must have an Interprocess Communication (IPC) path, especially for any node running on a machine or on a network it has access to. This is important when attaching a “console” to a running node.

Atomic reactor nodes – or **Atomic Reactors** for short, are powerful nodes that handle all types of requests from nodes querying it. Atomic reactors are not directly linked to one another to avoid network usage payloads or bandwidth wastages. They usually interact with atomic nodes, for instance, they return the IP addresses of all neighbouring atomic nodes having a requested file to the queryingpeer (such as an Element node).

Notes:

- The initial list of atomic (reactor) nodes is configured in the atomcore.json file.
- energyMax limit is set to 1000; and the default energyPrice is set to 26 oke.

References

[1] Rüdiger Schollmeier, A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002).

[2] Okpara O. D. 2022. The Atomchain Whitepaper, https://okpara.net/AtomChain.pdf

[3] Vitalik Buterin, Virgil Griffith. Casper the friendly finality gadget.2017 https://arxiv.org/abs/1710.09437

[4] Alistair Stewart, Eleftherios Kokoris-Kogia. GRANDPA: a Byzantine Finality Gadget. 2020

APPENDIX A: Sample Orbit Code in C++ source file

```
#include <iostream>
#include <string>

class Orbit                              //Orbit Code name as
Class
{
   public:
         doublegetBalance(void);
         voidfromAddress(string fromAddress);
         Orbit();                          //class constructor
   ~Orbit();                  //class destructor
   private:
         string balance;
         string address;
         int nonce = 0;
         doubleenergyMAX;
};

Orbit::Orbit(void){
   cout<< “Accessing the Orbit ” <<endl;
}
Orbit::~Orbit(void){
cout<< “Cancelling the Orbit ” <<endl;
}
double Orbit::getBalance(void){
   return balance;
}
void Orbit::fromAddress(string fromAddress){
   address = fromAddress;
}
…
classOrbState{

   OrbState():orbAddress(nullptr){…}

   OrbState(const char* str):orbAddress(nullptr){…}

   OrbState(constOrbState& other):orbAddress(nullptr){…}

   …

private:

   char* orbAddress;

   char* orbBalance =  nullptr;

};

enum class OrbCode{world, home, hash};

auto world = …;

OrbCode hash = OrbCode::hash;

…
```

APPENDIX B: Molecule Node Registration

```
$nodes = array('ip_atomicmolecule1',' ip_atomicmolecule2',...);
$keys = array('nodeid_molecule1', 'nodeid_molecule2', 'nodeid_molecule3',
...);
//Where the added variable is modified
$replicas = 160; //Number of replications per node
$buckets = array(); //Node hash dictionaries
$maps = array(); //storage key(molecule node) mapping relationship between
and nodes
/**
 * Generate node dictionary -- distribute nodes on the circle of unit
interval [0,1]
 */
foreach( $nodes as $key) {
        //Place of modification
        for($i=1;$i<=$replicas;$i++){
        $crc = crc32($key.'.'.$i)/pow(2,32);            // CRC To move
        $buckets[] = array('index'=>$crc,'node'=>$key);
        }
}
/*
 * Sort by index
 */
sort($buckets);
/*
 * hash each key to find its position on the circle
 * Then start at this location and find the first atomic molecule node in a
clockwise direction
 */
foreach($keys as $key){
    $flag = false; //Indicates whether an atomic molecule node has been
found
    $crc = crc32($key)/pow(2,32);//calculation key(molecule node) of hash
value
    for($i = 0; $i < count($buckets); $i++){
        if($buckets[$i]['index'] > $crc){
             /*
              * Because buckets have been sorted
              * Therefore, the first node whose index is greater than the
hash value of the key is the node to be found
              */
              $maps[$key] = $buckets[$i]['node'];
              $flag = true;
              break;

        }

     }
     if(!$flag){
        //If not found, use buckets First atomic molecule node in
        $maps[$key] = $buckets[0]['node'];
     }
}
foreach($maps as $key=>$val){
    echo $key.'=>'.$val,"<br />";
}
```