AVES - a first-class child of Ethereum brought to life by Ibro Fazlić MCS (2022)

AVES is a hard fork of the Ethereum blockchain, with numerous differences:

1. The mainnet starts with a fresh Genesis block and this allows the DAG file to be small enough to allow anyone to mine coins, regardless of hardware.

2. The mining fee has been reduced by more than 50% compared to Ethereum, so we expect to see a significant reduction in the cost of using smart contracts.

3. The snapshot sync option has been disabled as our blockchain is very small at the moment and we do not want nodes to take over the chain from peers instead of nodes. The pros and cons were discussed and it was determined that the pros outweigh the cons.

4. There are no plans to upgrade POS in the (near) future as this blockchain offers many possibilities. We are also investigating the implementation of a sub-protocol, also called the green protocol, where the mining software requires dedicated hardware from the miners but the resource consumption is up to 40% lower.

5. The code used in our blockchain stores 5% of each block to a publicly announced address from which funds are allocated to alternative energy innovators. This is done through community surveys and publicised in the media.

Accounts

An account is like a bank account, except it is not for money, but for AVS, where it can be held and transferred to different accounts, and can also be used to run smart contracts. An account is an entity composed of an address and a private key. The first 20 bytes of the SHA3-encrypted public key are the address.

There are two types of accounts:

1. Externally Owned Account: This is the basic type of an Aves account, it works similarly to a Bitcoin account. A private key controls the address for EOAs. (Externally Owned Account). Anyone can open as many EOAs as they need. The account is created each time a wallet is created, with a private key needed to access the EOA, check the balance, send and receive transactions and create smart contracts.

Advantages:

Transactions from an external account to a contract account can trigger code that can perform many different specific actions, such as transferring tokens or even creating a whole new contract.

External accounts cannot list incoming transactions.

2. Contract-based account: Contract-based accounts have all the features of an Externally Owned Account, but unlike EOAs, they are formed when a contract code is deployed and are controlled by contract codes accessed via a unique address. When a party accepts a contract, a completely unique account is created that contains all charges associated with that

contract. Each contract is given a unique serial number called a contract account.

Advantages:

A contract account can list incoming transactions.

Contract accounts can be set up as multisig accounts.

A multisig account can be set up to have a daily limit set by the account holder. Only if this limit is exceeded are multiple signatures required.

Disadvantage:

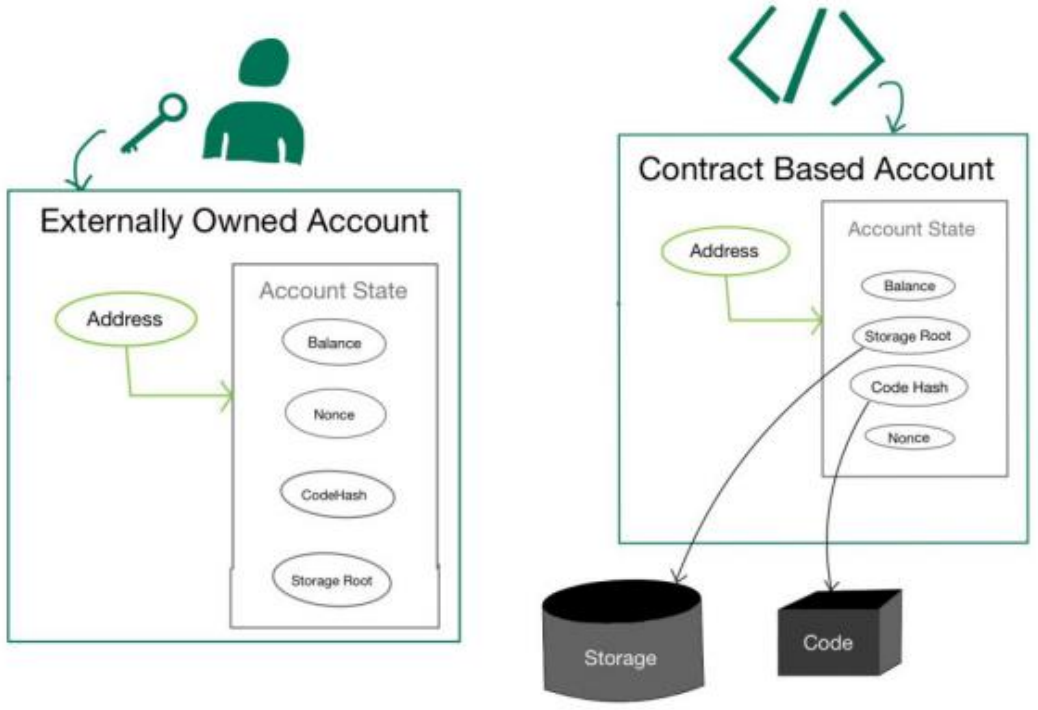
Creating contract accounts costs energy, as it takes up valuable computing and storage resources of the network. The energy needed is provided by fossil fuels such as natural gas and oil. Contract accounts cannot initiate new transactions on their own. Instead, contract accounts can only send transactions in response to other transactions they receive from either an Externally Owned Account or another account.

Types of contract accounts

Simple Account: The account is set up and maintained by a single account holder.

Multisig (multisignature) account: A multisig wallet contains several bearer accounts, one of which is also the author in each case.

Externally owned accounts vs. contract-based accounts



Explanations of fields within accounts

Nonce: The nonce in an account indicates the number of transactions sent from that account. This ensures that each transaction is unique by counting for each transaction.

Avs Balance: The balance in an account indicates the amount of Avs held in an Avs shop in the current Avs account.

Contract Code: This is optional as not all accounts have a contract code. Note, however, that it cannot be changed after execution.

Code hash: The value of the code hash for contract accounts is a hash that refers to the code of the account. Since there is no code associated with an external account, the code hash is an empty string.

Externally owned accounts and key pairs

An account is a private-public key pair that can be associated with a blockchain address.

It is an "owned" or "external" account if the private key is known and controlled by someone. Otherwise, it is a "smart contract" account if the private key is unknown and an address exists. Contract accounts are not associated with a private key, even if externally owned accounts have one.

Control and access to assets and contracts are granted via the EOA private key. The user is responsible for the security of the private key.

The account's public key, on the other hand, is, as the name implies, public. This key serves as the identity of the account. A one-way cryptographic function is used to generate a public key from the private key.

For example, when you create an account, you keep a private key for yourself while sharing the public key. Transactions between accounts are carried out with public keys.

Smart contract accounts: Like EOAs (Externally Owned Accounts), each smart contract account has a unique public address, and it is impossible to distinguish them from EOAs based on their address. Smart Contract accounts can receive and execute transactions just like EOAs. The main difference is that no single private key is used to verify transactions. Instead, the logic by which the account processes transactions is defined in the smart contract code. Smart contracts are programmes that run on the Aves blockchain and are executed when very specific conditions are met.

This feature of smart contract accounts, unlike EOAs, allows these accounts to implement access rights that define by whom, how and under what conditions transactions can be executed, as well as more complex logical tasks.

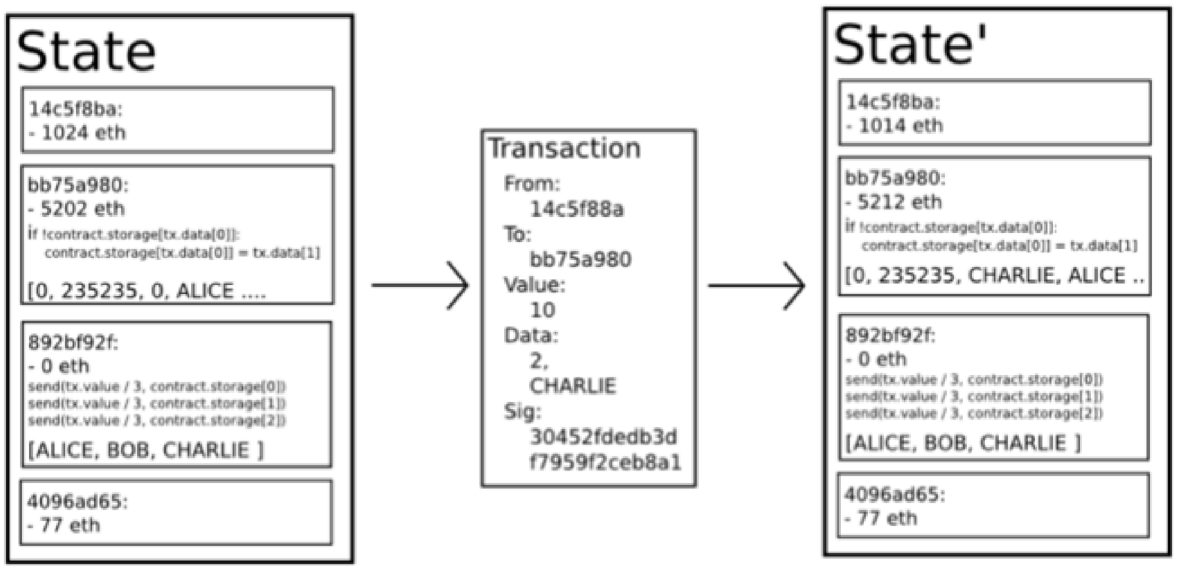
Messages and transactions

"Messages" in Aves are similar to "transactions" in Bitcoin, but with 3 key differences. First, an Aves message can be created by either an external entity or contract, whereas a Bitcoin transaction can only be created externally. Second, Aves messages can contain data. Third, the recipient of an Aves message, if it is a contract account, has the option to send a response. This allows Aves messages to take on the role of functions.

The term "transaction" is used in Aves to refer to the signed data packet that stores a message to be sent from an external account. Transactions contain the recipient of the message, a signature identifying the sender, the amount of Avs and data being sent, and two values called STARTGAS and GASPRICE. To prevent exponential bloat and infinite loops in the code, each transaction must set a limit on how many computational steps of code execution it can trigger, including the initial message and any additional messages that are triggered during execution. STARTGAS is this restriction (limit), and GASPRICE is the fee to be paid to the miner per computational step. If the transaction execution "runs out of gas", all state changes are reversed - except for the payment of the fees. If the transaction execution ends with some gas remaining, the remaining portion of the fees is returned to the sender. For the creation of a contract, there is a separate transaction type and a corresponding message type. The address of a contract is calculated based on the hash value of the account nonce and the transaction data.

An important consequence of the messaging system is the "first class citizen" property of Aves - the idea that contracts have the same powers as external accounts, including the ability to send messages and create other contracts. This allows contracts to perform many functions simultaneously: For example, a member of a decentralised organisation (one contract) can be an escrow account (another contract) between a paranoid individual using bespoke quantum-safe Lamport signatures (a third contract) and a co-signing entity, which in turn uses a five-key account for security (a fourth contract). The strength of the Aves platform is that the decentralised organisation and the trust contract do not have to worry about what form of account each party uses.

Aves state transition function


State
14c5f8ba:
- 1024 eth
bb75a980:
- 5202 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, 0, ALICE ....
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65:
- 77 eth
Transaction
From:
14c5f88a
To:
bb75a980
Value:
10
Data:
2,
CHARLIE
Sig:
30452fdedb3d
f7959f2ceb8a1
State'
14c5f8ba:
- 1014 eth
bb75a980:
- 5212 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, CHARLIE, ALICE ..
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65:
- 77 eth

The state transition function of Aves, APPLY (S, TX) - > S' can be defined as follows:

1. Check that the transaction is well-formed (i.e. has the correct number of values), the signature is valid and the nonce matches the nonce in the sender's account. If not, return an error.

2. Calculate the transaction fee as STARTGAS * GASPRICE and determine the sender address from the signature. Subtract the fee from the sender's account balance and increase the sender's nonce. If the balance is not sufficient to spend the fee, return an error.

3. Initialise GAS = STARTGAS and take off a certain amount of gas per byte to pay for the bytes of the transaction.

4. Transfer the transaction value from the sender's account to the receiver's account. If the receiver account does not yet exist, create it. If the recipient account is a contract, execute the contract code either until completion or until the execution runs out of gas.

5. If the transfer of value failed because the sender did not have enough money or the code execution ran out of gas, reverse all state changes except the payment of fees and credit the fees to the miner's account.

6. Otherwise, refund the fees for the remaining gas to the sender and transfer the fees paid for the used gas to the miner.

For example, let us assume that the code of the contract is as follows

```
if !contract.storage[msg.data [0]]:

contract.storage[msg.data [0]] = msg.data [1]
```

Note that the contract code is actually written in low-level EVM code. This example is written in Serpent, our high-level language, for clarity and can be compiled down to EVM code.

Let us assume the contract shop is empty at the beginning and a transaction is sent with 10 Avs value, 2000 gas, 0.001 Avs gas price and two data fields: [2, 'CHARLIE'][3]. The process for the state transition function in this case is as follows:

1. Check that the transaction is valid and correctly constructed.

2. Check that the sender of the transaction has at least 2000 * 0.001 = 2 Avs. If so, deduct 2 Avs from the sender's account.

3. Initialise gas = 2000; assuming the transaction is 170 bytes long and the byte charge is 5, deduct 850, leaving 1150 gas.

4. Subtract another 10 Avs from the sender's account and add it to the contractor's account.

5. Execute the code. In this case it is quite simple: it checks whether the contract's storage at index 2 is used up, finds that it is not, and sets the storage at index 2 to the value CHARLIE. Let us say 187 gas is consumed, so the remaining gas is 1150 - 187 = 963. 6. Add 963 * 0.001 = 0.963 Avs to the sender's account and return the resulting status.

If there were no contract on the receiving side of the transaction, then the total transaction charge would simply be equal to the specified GASPRICE multiplied by the length of the transaction in bytes, and the data sent with the transaction would be irrelevant. Also note that contract-initiated messages may assign a gas limit to the calculation they initiate. If the sub-calculation runs out of gas, it will only be reset to the point where the message was invoked. So, just like transactions, contracts can protect their limited computing resources by setting strict limits on the sub-calculations they trigger.

Code execution

The code in Aves contracts is written in a low-level, stack-based bytecode language known as

"Ethereum Virtual Machine Code" or "EVM Code". The code consists of a series of bytes, with each byte representing an operation. Generally, code execution is an infinite loop that consists of repeatedly executing the operation at the current programme counter (which starts at zero) and then incrementing the programme counter by one until the end of the code is reached or an error or STOP or RETURN instruction is detected. Operations have access to three types of memory where they can shop data:

● The stack, a last-in-first-out container into which 32-byte values can be pushed and popped

● Memory, an infinitely expandable byte array

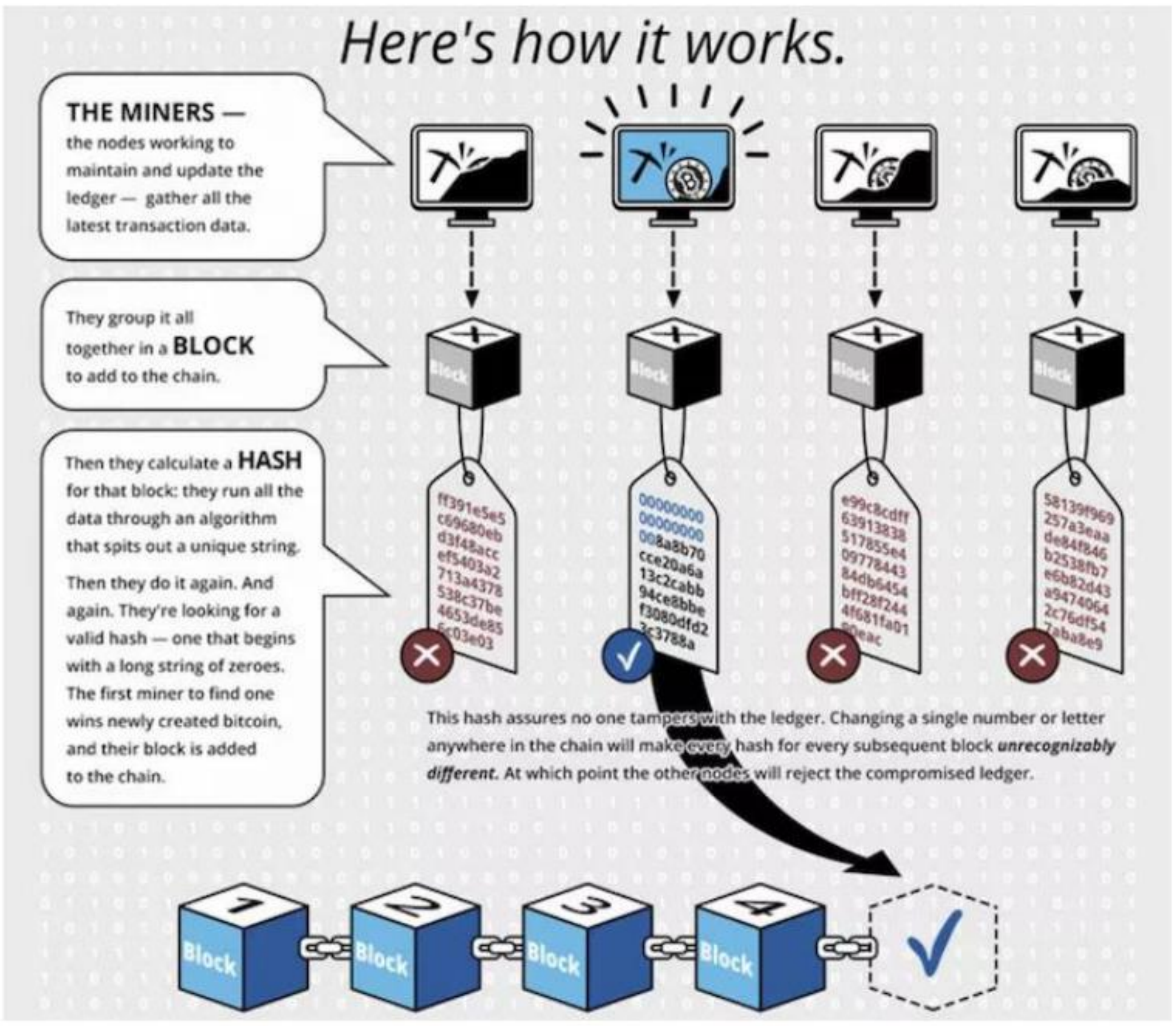
● The contract's long-term memory, a key/value shop in which keys and values are each 32 bytes in size. Unlike the stack and memory, which are reset after the end of the calculation, the memory remains for the long term.

The code can also access the value, sender and data of the incoming message, as well as the block header data, and the code can also return a byte array of data as output.

The formal execution model of the EVM code is surprisingly simple. While the Aves virtual machine is running, its complete computational state can be described by the tuple (block_state, transaction, message, code, memory, stack, pc, gas), where block_state is the global state containing all accounts and includes balances and memory. In each round of execution, the current statement is found by taking the pc-th byte of code, and each statement has its own unique definition in terms of how it affects the tuple. For example: ADD fetches two items from the stack and adds them, reduces gas by 1 and increases pc by 1. SSTORE fetches the top two items from the stack and inserts the second item into the contract's memory at the index specified by the first item, reduces gas by up to 200 and increases pc by 1. Although there are many ways to optimise

Aves through just-in-time compilation, a simple implementation of Aves can be completed in a few hundred lines of code.

Blockchain and Mining


Here's how it works.
THE MINERS — the nodes working to maintain and update the ledger — gather all the latest transaction data.
They group it all together in a BLOCK to add to the chain.
Then they calculate a HASH for that block: they run all the data through an algorithm that spits out a unique string.
Then they do it again. And again. They're looking for a valid hash — one that begins with a long string of zeroes. The first miner to find one wins newly created bitcoin, and their block is added to the chain.
This hash assures no one tampers with the ledger. Changing a single number or letter anywhere in the chain will make every hash for every subsequent block unrecognizably different. At which point the other nodes will reject the compromised ledger.
Block
1
2
3
4

The Aves blockchain is similar to the Bitcoin blockchain in many ways, although there are some differences. The main difference between Aves and Bitcoin in terms of blockchain architecture is that, unlike Bitcoin, Aves blocks contain a duplicate of both the transaction list and the last state. In addition, two different values, the block number and the difficulty, are stored in the block.

The algorithm for block validation in Aves is as follows:

1. Check if the previous block referenced exists and is valid.

2. Check that the timestamp of the block is greater than that of the referenced previous block and less than 15 minutes into the future 3. Check that the block number, difficulty, transaction root, uncle root and gas limit (various Aves-specific low-level concepts) are valid.

4. Check that the proof of work is valid for the block.

5. Let S[0] be the STATE _ROOT of the previous block.

6. TX let be the transaction list of the block, with n transactions. For all in in 0...n-1, setS[i+1] =

APPLY(S[i], TX [i]). If any of the applications return an error, or if the total gas consumption of the block exceeds the GASLIMIT up to that point, return an error.

7. Let S_ FINAL be S[n], but adding the block reward paid to the prospector.

8. Check if S_ FINAL is the same as STATE _ROOT. If it is, the block is valid, otherwise it is invalid.

The approach may seem very inefficient at first glance, as the entire status has to be stored for each block, but in reality the efficiency should be comparable to that of Bitcoin. The reason for this is that the status is stored in the tree structure and only a small part of the tree needs to be changed after each block. For this reason, the majority of the tree should be the same between two adjacent blocks, so that the data can be stored once and referenced twice using pointers (i.e. hashes of subtrees). This is done using a special type of tree called a "Patricia tree", which is a variation of the Merkle tree concept and allows nodes to be inserted and deleted efficiently, not just changed. Furthermore, since all state information is part of the last block, there is no need to shop the entire blockchain history - a method that, if it could be applied to Bitcoin, would mathematically offer a 5-20-fold space saving.

Applications

In general, there are three types of applications that are built on top of Aves. The first category is financial applications that provide users with more powerful ways to manage and contract their money. These include sub-currencies, financial derivatives, hedging contracts, savings, wills and finally even some classes of full-scale employment contracts. The second category is semi-financial applications that involve money but also have a strong non-monetary side; a perfect example is self-enforcing rewards for solving math problems. Finally, there are applications such as online voting and decentralised governance that have no financial character at all.

Token System

On-blockchain token systems include everything from sub-currencies representing assets such as US dollars and gold, to corporate stocks, individual tokens representing smart properties, secure anti-counterfeit coupons, and even traditional value respects. It has a wide variety of uses, all the way up to unconventional token systems. incentive point system. A token system is surprisingly easy to implement in Aves. The most important thing to understand is that any currency or token system is basically a database with one operation. Subtract X units from A, giving B

X units. However, (1) X occurs before X units are traded for at least X units. (2) the transaction is authorized by A; Implementing this logic in a contract is all that is required to implement a token system. The basic code for implementing the token system in

Serpent is:from = msg.sender to = msg.data[0] value = msg.data[1]

```
if contract.storage[from] >= values :


contract.storage[from] = contract.storage[from] value

contract.storage[to] = contract.storage[to] + value
```

This is essentially a literal implementation of the state transition function "banking system".

Added a few lines of code to allow for the first step of distributing currency units first and a few other edge cases. Ideally, add a function that allows other contracts to query the balance of the address. But that's it. Theoretically, an Aves-based

token scheme acting as a sub-currency could include another important feature not found in Bitcoin-based on-chain meta-currencies. It is the ability to pay transaction fees directly in that currency. This is implemented such that the contract maintains an Avs balance, uses this to refund the Avs used to pay fees to the sender, and accumulates internal currency units received as fees to replenish this balance. , is implemented to resell them. With always ongoing auctions. So the user has to "activate" their account with Avs, but with Avs it's reusable because the contract refunds every time.

Financial Derivatives and Currencies of Stable Value

Financial derivatives are the most common application of "smart contracts" and one of the easiest to implement in code. A major challenge in implementing financial contracts is that most of them require a reference to an external price ticker. For example, a highly desirable application would be a smart contract to hedge the volatility of his AVS (or any other cryptocurrency) against the US dollar, but that would require the contract to know the value of his AVS/USD. The easiest way to do this is through a "data feed" contract that is controlled by a specific party (such as NASDAQ) and designed to allow that party to renew the contract as needed.

This allows other contracts to send messages to this contract and receive responses with prices. Considering this important factor, the hedging contract looks like this:

1. Wait for Party A to enter her 1000 Avs.

2. Wait for Party B to enter her 1000 Avs.

3. Record in memory the USD value of 1000 Avs calculated by querying the data feed contract. Suppose this is $x.

4. After 30 days, A or B "pings" the contract and sends x$ worth of Av (calculated by rescanning the data feed contract to get the new price) to A and the rest to B allow it to be sent. This type of contract could have great potential in crypto commerce. One of the main problems mentioned with cryptocurrencies is the fact that they are unstable. Many users and traders want security and convenience when working with crypto assets, but they may not want to face the possibility of losing 23% of the value of their funds in a single day. To date, the most commonly proposed solution is issuer-backed assets. The idea is to create a sub-currency where the issuer has the right to issue and withdraw units, and who provides (offline) units of a particular underlying asset (e.g. gold), which is one unit provided by the currency. But it's about creating USD.). The issuer then promises to provide her 1 unit of the underlying asset to the person returning her 1 unit of cryptocurrency. This mechanism

allows non-crypto assets to be “lifted” to crypto if the issuer is trusted.

In practice, however, issuers are not always trustworthy, and in some cases, banks' infrastructure is too weak or too hostile for such services to exist.Financial derivatives are an alternative. provide the means. A decentralized marketplace of speculators betting on crypto benchmarks rising in price instead of a single issuer providing the funds to back the asset fills this role. Unlike the issuer, the speculator has no chance of default on the part of the trade as the hedge contract holds the funds in trust. Note that this approach is not fully decentralized as it requires a trusted source to provide price tickers. It could probably be classified as free speech), reducing the chance of fraud.

Identity and Reputation System

Namecoin, the first of all alternative cryptocurrencies, applies a Bitcoin-like blockchain to allow users to register their name in a public database along with various data. We tried to provide a name registration system that could The most commonly cited use case is a DNS system that maps domain names like "bitcoin.org" (or "bitcoin.bit" for Namecoin) to her IP address. Other use cases might include email authentication or more advanced reputation systems. The basic contract for

Aves to provide a Namecoin-like name registration system is:

```
if !contract.storage[tx.data[0]]:
```

Decentralized file storage

Over the past decade, there have emerged a number of popular online file storage startups, the most prominent being Dropbox, allowing users to upload a backup of their hard drive and have the service store the backup and allow the user to access it inexchange for a monthly fee However, at this point the file storage market is at times relatively inefficient; a cursory look at various existing solutions shows that, particularly at the "uncanny valley" 20-200 GB level at which neither free quotas nor enterprise-level discounts kick in, monthly prices for mainstream file storage costs are such that you are paying for more than the cost of the entire hard drive in a single month. Aves contracts can allow for the development of a decentralized file storage ecosystem, where individual users can earn small quantities of money by renting out their own hard drives and unused space can be used to further drive down the costs of file storage.

The key underpinning piece of such a device would be what we have termed the "decentralized Dropbox contract". This contract works as follows. First, one splits the desired data up into blocks, encrypting each block for privacy, and builds a Merkle tree out of it. One then makes a contract with the rule that, every N blocks, the contract would pick a random index in the Merkle tree (using the previous block hash, accessible from contract code, as a source of randomness), and give X Avs to the first entity to supply a transaction with a simplified payment verification-like proof of ownership of the block at that particular index in the tree. When a user wants to re-download their file, they can use a micropayment channel protocol (eg. pay 1 szabo per 32 kilobytes) to recover the file; the most fee-efficient approach is for the payer not to publish the transaction until the end, instead replacing the transaction with a slightly more lucrative one with the same nonce after every 32 kilobytes.

A crucial function of the protocol is that, although it may seem like one is trusting many random nodes not to decide to forget the file, one can reduce that risk down to near-zero by splitting the file into many pieces via secret sharing, and watching the contracts to see each piece is still in some node's possession. If a contract is still paying out money, that provides a cryptographic proof that someone out there is still storing the file.

Decentralized Autonomous Organization

The general concept of a 'decentralized organization' is that a certain number of members or shareholders, perhaps in a majority of 67%, have the right to use and change the entity's funds. The concept of a virtual entity with groups. code. Members collectively decide how the organization should allocate its funds. Methods of allocating DAO funds can range from bonuses, salaries, to even more exotic mechanisms like internal currency to reward work. It essentially replicates the legal specifications of a traditional business or non-profit organization, but uses only cryptographic blockchain technology for enforcement. has revolved around a “capitalist” model of a “decentralized autonomous society” (DAC) with shareholders paying An alternative, perhaps called a “decentralized autonomous community,” would require all members to share decision-making equally, and her 67% of existing members to approve adding or removing members.

Here's a general overview of how DO is encoded: The simplest design is self-modifying code that changes when her two-thirds of the members agree to the change. Your code is theoretically immutable, but you can easily get around this by placing parts of your code in separate contracts and storing the addresses of the contracts you call in mutable memory, effectively reducing mutability. can have A naive implementation of such a DAO contract has three transaction types distinguished by the data provided in the transaction. Storage index K of value V

- [0,i] Register approval for proposal i
- [2,i] Close proposal i if enough votes are cast

The contract has these There are respective terms of A record is kept of all open save changes and a list of who voted for them. There is also a list of all members. When a change in storage reaches her two-thirds of the members who voted for it, the final transaction can execute the change. A more sophisticated skeleton would also have built-in voting functionality for functions such as submitting transactions, adding and removing members, and potentially enabling Liquid Democracy-style voting delegation (i.e. anyone can be assigned to vote for , and the assignment is transitive). , that is, if A assigns B and B assigns C, then C determines A's vote). This design allows the DO to grow organically as a decentralized community, eventually delegating the task of excluding people who are members to experts, but what the "current system" is In contrast, experts easily grow over time and individual community members change their attributes.

Another model is a decentralized enterprise, where each account can have 0 or more shares, and two-thirds of the shares are required for decision making. A complete backbone includes asset management functionality, the ability to create offers to buy or sell stocks, and the ability to accept offers (preferably using an order matching mechanism within a contract). Delegation also exists in the style of liquid democracy, which generalizes the concept of a "board of directors." In the future, more sophisticated governance mechanisms may be implemented. At this point, the decentralized organization (DO) can be called a distributed autonomous organization (DAO). The distinction between DO and DAO is fuzzy, but the general line of demarcation is Avs.

Governance is typically done through policy-like or “automated” processes. A good intuitive test is the "no common language" criterion. Can the organization continue to function without her having two members who speak the same language? Of course, a simple, traditional shareholder-style company would fail, but something like the Bitcoin protocol would be far more likely to succeed. Robin Hanson's Futarchy, an organized control mechanism with prediction markets, is a good example of what truly "autonomous" control looks like. Note that you shouldn't necessarily assume that all DAOs are better than all DOs. Automation is just a paradigm with very strong benefits in certain places and likely impractical elsewhere, and there can be many semi-DAOs as well.

Other applications

1. savings wallet. Ana wants to keep her funds safe, but she's worried she might lose her private key or she might be hacked by someone. Suppose there is her AVS contracts with bank Brent as follows.

● Brent is single and he can withdraw up to 1% of her balance per day, but Ana can use the key to trade, which she disables.

● Ana and Brent can pull anything together.

Ana 1% a day is usually enough. If Ana wants to withdraw more money, she can ask Brent for help. When Ana's keys were hacked, she ran to Brent to transfer her money to her new contract. Brent will eventually withdraw the money if she loses her key, and if Brent turns out to be evil, she can disable his retreat ability.

2. Crop insurance. You can easily enter into financial derivative contracts, but use a weather data feed instead of a price index. If a farmer in Iowa bought a derivative that was inversely proportional to the amount of rainfall in Iowa, when a drought hit, the farmer would automatically make money, and if there was enough rain, the crop would do well, so the farmer would become happy.

3. Distributed data feed. Financial contracts could actually decentralize their data feeds via a protocol called SchellingCoin. Basically, SchellingCoin works like this: N parties each enter a specific date value (e.g. AVS/USD price) into the system, and the values are sorted and placed between the 25th and 75th percentiles. will get his 1 token as a reward. Everyone has an incentive to give an answer that everyone else will answer, and the only value that a large number of players can realistically agree on is truth, which is an unambiguous criterion. This creates a decentralized log that can theoretically provide any number of values, such as AVS/USD prices, Berlin temperatures, or the results of certain hard calculations.

4. Smart multi-signature escrow. Bitcoin allows for multi-signature transaction contracts where, for example, 3 out of 5 keys can issue money. Aves is more granular. For example, 4 out of 5 can use everything, 3 out of 5 he can use up to 10% a day, 2 out of 5 can use up to 0%.

Additionally, Aves multisig is asynchronous - two parties can register their signatures on the blockchain at different times and the last signature will automatically send the transaction.

5. Cloud Computing. EVM technology can also be used to create verifiable computing environments. This allows users to have other users perform computations and optionally request proof that the computations were performed correctly at certain randomly chosen checkpoints. It enables the creation of a cloud computing marketplace where all users can participate with desktops, laptops or dedicated servers, using security deposit sampling to ensure systems are trustworthy. (i.e. no node can cheat for profit). Although such systems are not suitable for all tasks. For example, tasks that require high-level communication between processes simply cannot run on a large cloud of nodes. However, other tasks are much easier to parallelize. Projects such as SETI@home, Folding@home, and genetic algorithms can be easily implemented on such platforms.

6. Peer-to-Peer Gambling. Any number of peer-to-peer gaming protocols can be implemented on the Aves blockchain. Cyberdice by B. Frank Stajano and Richard Clayton. The simplest gambling protocols are really just contracts for the difference of the next block hash, from which more advanced protocols can be built to create non-cheat, near-zero fee gambling services.

7. Prediction Markets. Prediction markets can also be easily implemented with an oracle or Schering coin,

A prediction market using SchellingCoin could prove to be the first mainstream application of Futarchy as a governance protocol for decentralized organizations.

8. An on-chain decentralized marketplace based on identity and reputation systems.

NOTES

Modified His GHOST Implementation

The Greedy Heaviest Observed Subtree (GHOST) protocol was first introduced by Yonatan Sompolinsky and Aviv Zohar in December 2013 for transactions processed without committing innovation. It is an end-to-end cryptographic protocol that provides authentication without relying on a centralized trust authority. It can be symmetrical or asymmetrical depending on how you use it. The principle of GHOST is that the sender simply sends ghost-her packets (or dummy packets) to the receiver, and the receiver can respond with as many packets as they want.

The sender creates a digital signature by encrypting the packet with the recipient's public key.

Recipient decrypts with private key (public key is used for encryption). If the decryption is successful, the sender is authenticated and the transaction is accepted. You can also use the same method to send this ghost-her packet to other recipients (i.e. the transaction is broadcast). Because there can be multiple receivers, the protocol is called "GHOST", short for "Greedy Heaviest Observed Sub-Tree", and in addition to the direct route to and from the sender, other nodes Used as a reference to how to route packets through Receiving machine.

Need For GHOST Protocol

Transactions in the blockchain can be published anywhere. In PoW blockchains such as Bitcoin, Aves, the random nature of hashes allows two miners to work on the same transaction creating two blocks of hers.

Only one of these transactions can be added to the main blockchain.

This means that all the work the second miner has done to validate her second block is lost (orphaned).

Miners are not rewarded. These blocks are called uncle blocks in Aves. The GHOST protocol is a chain selection rule that takes previously orphaned blocks and adds them to the main blockchain, partially rewarding miners as well. This makes attacks on the network more difficult. Because not only winning miners have computing power. More nodes hold power, discouraging the need for centralized mining pools with larger chains.

GHOST Protocol Implementation Bitcore, the developer of

Bitcoin, has implemented the GHOST protocol. It is also the first public implementation of the GHOST protocol.

Avs can use them in different ways to maximize their effectiveness.

For example, GHOST channels can be used to exchange coins or various digital assets that do not take advantage of Bitcoin's block verification time and consensus mechanism (e.g., coins that require reliable processing, such as stablecoins).

How does the GHOST protocol work?

GHOST works by sending dummy/empty packets or "ghosts" to the recipient.

Sender sends ghost his packet with header and encrypted payload, but no block reward (i.e. no transaction). Wait for an empty packet from the receiver. If an empty packet is received, it means the receiver has received a ghost, so it can send up to 2*pendingtxns to the network without sending. If multiple nodes have transactions pending in the queue, some kind of protocol must be put in place to decide which node will send that block (i.e. which node wins)

GHOST Protocol Pros

Built with scalability and security in mind, it can easily handle thousands.

Easy Transactions: In a world where cryptocurrency transactions can be completed in seconds from anywhere in the world, the GHOST protocol will allow individuals to make transactions easily with efficient use of computing power.

Developer Freedom: If a developer doesn't want to be held responsible for maintaining his own infrastructure, he can instead use his GHOST-based smart his contracts running on top of it.

Effort and Time: It's their time and effort. Smart contracts are much faster and easier than creating packages from scratch. This will allow more people to join his dApp space. This is very good for brand new developers and entrepreneurs to join.

Superior Transparency: Provides greater transparency than Aves' ERC20 standard (which platforms such as MyEthWallet and MetaMask still use).

It allows developers to accept completely anonymous payment. Non-anonymous or pseudonymous payment systems are highly preferred to prevent hackers and online his phishers from attacking you and stealing your funds.

Denial of GHOST Protocol

Hinder Increased Recruitment: Hinder Increased Recruitment.

Too complicated when not in use: If no one wants to use the GHOST protocol, the means of paying customers in tokens or Avs is still too complicated.

Impractical Option: This is not a practical option on the particular platform. Blockchain-based games come to mind first.

Make dApps Expensive: Make dApps more expensive.

gas cost for every transaction: dApps using this protocol must pay the gas cost for every transaction, even those that don't contain a transaction.

Fee

Gas Fee is the amount of Avs (AVS) required for a user of the Aves blockchain network to complete a transaction on the network.

gas fees are used by Aves miners to validate transactions and compensate for their work in securing the network. Gas tariffs also help prevent the network from being blocked by malicious users spamming the network with transactions. Aves gas rates fluctuate because the formulas used to calculate them are dynamic. High fees and relatively slow speeds are common criticisms of the Aves network. Gas bills are paid in Avs and denominated in Gwei or Gigawei. . Each gwei is equal to 0.000000001 AVS. The gas fee is dependent on two factors. Gas units and gas price.

Gas fee= gas units X gas price

Gas units is a number that depends on the amount of computation required for a transaction. For e.g., if you send some Ave to someone, it requires 11,000 gas units. It’s the minimum number of units required for any transaction. On the other hand, Gas price is determined by the demand for making transactions. The more traffic, the higher the price. This is why you don’t pay the same gas fee each time you transact. In our case, we have a lower gas fee than Ethereum, so we don't expect too high prices in case of huge demands for smart contracts

Our minimal gas fee is:

```
TxGas                  uint64 = 11000
TxGasContractCreation uint64 = 23000
```

Computation And Turing-Completeness

What is Turing completeness?

Turing-completeness is a phrase defined by Alan Turing which describes the idea that some computing machines are capable of performing any task a computer can perform.

The concept of Turing-completeness is one at the heart of software and application development, where it allows code to be written without having to check beforehand if it will work or not.

In other words, one can write your program without worrying about what else is allowed for it to do.This is essential in determining usability as well as many other aspects of the software. It is also important to know what “Turing-complete” means and how it relates to Aves. In Turing’s paper, the concept of Turing-complete machines is used to disprove the possibility of true artificial intelligence. For instance, a machine can eventually imitate the behaviours of a human. In practical terms, this means that “Turing-complete” allows programmers to write code that can be used by any computer to achieve any result. It is necessary for introducing new techniques and ideas into software programming such as functional programming or even for understanding ideas about universal computation with regard to general computing.

One of the main obstacles that cryptocurrency runs into is reliance on a third party, typically an entity such as a bank. These companies are responsible for ensuring that a cryptocurrency can be used in everyday transactions because it must be compatible with traditional banking services. Turing completeness is a characteristic of a programming language. A language is Turing-complete if it can be used to simulate a Turing machine, which means that an appropriately designed program can solve any problem that a Universal Turing Machine (UTM) can solve. In order for this to be feasible, programs must be free from restrictions, such as halting and infinite loops. Theoretically, Turing’s completeness enables the development of highly advanced programs in one language and allows other projects or companies to create highly advanced applications using the same tools.

Key points

Aves can be built on the blockchain that has been built right now with no need to upgrade. A cheap and scalable Blockchain offering storage and processing power that is almost limitless and at the same time evolving. Aves will change the whole

Blockchain ecosystem by making it possible to do many more things on it. All thanks to its Turing-complete language Solidity.

The whole concept of cryptocurrencies and smart contracts is based on the idea of Turing-complete languages. Smart contracts are being used for all kinds of applications including common transactions like payments, buying a car, or even licensing music or software. Smart contracts can be used in an efficient way to run state channels between users and to make payments transparently.

ERC20 is the standard document that provides a structure for tokens (works on Aves) and it is compatible with most tokens projects like City Coin, Request Network, and others.

Initial coin offering (ICO) which is a new way to raise money for a project – uses ERC20 as the monetary unit without issuing tradable ERC20 tokens before the ICO launch.

Aves as Turing-Completeness.

Since it relies on programmable smart contracts, Aves is not reliant on third-party services to function. This means that, theoretically, one could buy a house or make other major purchases on the Aves blockchain through the use of a smart contract. However, there are concerns regarding whether or not this is feasible due to the high costs associated with Turing complete systems and their ability to run continuously without human intervention. Theoretically, there are several challenges associated with Turing complete cryptocurrencies. As the cryptocurrency industry continues to grow and expand, new ideas are being brought forth constantly — many of which would not have been possible without Turing completeness.

Does a System Have To Be Turing Complete To Be Useful in Blockchain?

A system has to be Turing complete in order to be useful in the blockchain, but it can have all the other desirable properties of a blockchain, such as decentralization and trustless transactions.

A system is Turing complete if it can simulate an arbitrary computer program. Turing complete systems have to be able to run any possible computation, which includes the most complex types of computation such as those found in blockchain. This type of system also often has better performance than other systems because it can use a set of rules which are more efficient when solving problems with many steps.

In addition to being Turing complete, a system must also be decentralized and allow trustless transactions according to consensus in order for it to be useful in the blockchain. These properties are necessary for the security and consistency that a blockchain needs in order for its data records or “blocks” to have value and meaning.

Aves is a very new technology with many possibilities that can disrupt our lives in the future. In short, it is a complex network of computers that allows one to create own currency and it is totally decentralized and free to use.

Mining Centralization

The Bitcoin mining algorithm essentially works by having miners compute SHA256 on slightly modified versions of the block header hundreds of thousands of times over and over again until eventually, one node comes up with a version whose hash is much less than the target (currently around 2^190). however, this mining algorithm is susceptible to two forms of centralization. First, the mining ecosystem has come to be dominated by ASICs (application-specific integrated circuits), computer chips designed for, and therefore thousands of times more efficient at, the specific task of Bitcoin mining. This means that Bitcoin mining is no longer a highly decentralized and egalitarian pursuit, requiring millions of dollars of capital to effectively participate in. Second, most Bitcoin miners do not actually perform block validation locally; instead, they rely on a centralized mining pool to provide the block headers. This problem is arguably worse: as of the time of this writing, the top two mining pools indirectly control roughly 50% of processing power in the Bitcoin network,

although this is mitigated by the fact that miners can switch to other mining pools if a pool or coalition attempts a 51% attack.

The current intent at Aves is to use a mining algorithm based on randomly generating a unique hash function for every 1000 nonces, using a sufficiently broad range of computation to remove the benefit of specialized hardware. Such a strategy will certainly not reduce the gain of centralization to zero, but it does not need to. Note that each individual user, on their private laptop or desktop, can perform a certain quantity of mining activity almost for free, paying only electricity costs, but after the point of 100% CPU utilization of their computer additional mining will require them to pay for both electricity and hardware. ASIC mining companies need to pay for electricity and hardware starting from the first hash. Hence, if the centralization gain can be kept to below this ratio, (E + H) / E, then even if ASICs are made there will still be room for ordinary miners. Additionally, we intend to design the mining algorithm so that mining requires access to the entire blockchain, forcing miners to store the entire blockchain and at least be capable of verifying every transaction. This removes the need for centralized mining pools; although mining pools can still serve the legitimate role of evening out the randomness of reward distribution, this function can be served equally well by peer-to-peer pools with no central control. It additionally helps fight centralization, by increasing the number of full nodes in the network so that the network remains reasonably decentralized even if most ordinary users prefer light clients.

Scalability

One common concern about Aves is the issue of scalability. Like Bitcoin, Aves suffers from the flaw

that every transaction needs to be processed by every node in the network. With Bitcoin, the size of the current blockchain rests at about 437 GB, growing by about 1 MB per hour. If the Bitcoin network were to process Visa's 2000 transactions per second, it would grow by 1 MB per three seconds (1 GB per hour, 8 TB per year). Aves is likely to suffer a similar growth pattern, worsened by the fact that there will be many applications on top of the Aves blockchain instead of just a currency as is the case with Bitcoin, but ameliorated by the fact that Aveses full nodes need to store just the state instead of the entire blockchain history. The problem with such a large blockchain size is centralization risk. If the blockchain size increases to, say, 100 TB, then the likely scenario would be that only a very small number of large businesses would run full nodes, with all regular users using light SPV nodes. In such a situation, there arises the potential concern that the full nodes could band and all agree to cheat in some profitable fashion (eg. change the block reward, give themselves BTC). Light nodes would have no way of detecting this immediately. Of course, at least one honest full node would likely exist, and after a few hours information about the fraud would trickle out through channels like Reddit, but at that point it would be too late: it would be up to the ordinary users to organize an effort to blacklist the given blocks, a massive and likely infeasible coordination problem on a similar scale as that of pulling off a successful 51% attack. In the case of Bitcoin, this is currently a problem, but there exists a blockchain modification suggested by Peter Todd which will alleviate this issue. In the near term, Aves will use two additional strategies to cope with this problem. First, because of the blockchain-based mining algorithms, at least every miner will be forced to be a full node, creating a lower bound on the number of full nodes. Second and more importantly, however, we will include an intermediate state tree root in the blockchain after processing each transaction. Even if block validation is centralized, as long as one honest verifying node exists, the centralization problem can be circumvented via a verification

protocol. If a miner publishes an invalid block, that block must either be badly formatted, or the state S[n] is incorrect. Since S[0] is known to be correct, there must be some first state S[i] that is incorrect where S[i-1] is correct. The verifying node would provide the index i, along with a "proof of invalidity" consisting of the subset of Patricia tree nodes needing to process APPLY(S[i-1],TX[i]) -> S[i]. Nodes would be able to use those nodes to run that part of the computation, and see that the S[i] generated does not match the S[i] provided. Another, more sophisticated, attack would involve the malicious miners publishing incomplete blocks, so the full information does not even exist to determine or not blocks are valid. The solution to this is a challenge-response protocol: verification nodes issue "challenges" in the form of target transaction indices, and upon receiving a node a light node treats the block as untrusted until another node, the miner or another verifier, provides a subset of Patricia nodes as a proof of validity..

Decentralized Applications

The contract mechanism described above allows anyone to build what is essentially a command line application run on a virtual machine that is executed by consensus across the entire network, allowing it to modify a globally accessible state as its “hard drive”. However, for most people, the command line interface that is the transaction sending mechanism is not sufficiently user-friendly to make decentralization an attractive mainstream alternative. To this end, a complete “decentralized application” should consist of both low-level business-logic components, whether implemented entirely on Aves, using a combination of Aves and other systems or other systems entirely, and high-level graphical user interface components. The Aves client’s design is to serve as a web browser, but include support for a “Ave” Javascript API object, which specialized web pages viewed in the client will be able to use to interact with the Aves blockchain. From the point of view of the “traditional” web, these web pages are entirely static content, since the blockchain and other decentralized protocols will serve as a complete replacement for the server for the purpose of handling user-initiated requests. Eventually, decentralized protocols, hopefully themselves in some fashion using Aves, may be used to store the web pages themselves.

Conclusion

The Aves protocol was originally conceived as an upgraded version of a cryptocurrency, providing advanced features such as on-blockchain escrow, withdrawal limits and financial contracts, gambling markets and the like via a highly generalized programming language. The Aves protocol would not "support" any of the applications directly, but the existence of a Turing-complete programming language means that arbitrary contracts can theoretically be created for any transaction type or application. What is more interesting about Aves, however, is that the Aves protocol moves far beyond just currency. Protocols and decentralized applications around decentralized file storage, decentralized computation and decentralized prediction markets, among dozens of other such concepts, have the potential to substantially increase the efficiency of the computational industry, and provide a massive boost to other peer-to-peer protocols by adding for the first time an economic layer. Finally, there is also a substantial array of applications that have nothing to do with money at all. The concept of an arbitrary state transition function as implemented by the Aves protocol provides for a platform with unique potential; rather than being a closed-ended, single-purpose protocol intended for a specific array of applications in data storage, gambling or finance, Aves is open-ended by design, and we believe that it is extremely well-suited to serving as a foundational layer for a very large number of both financial and non-financial protocols in the years to come.

References

Ethereum White paper by Vitalik Buterin 2014
https://ethereum.org/en/whitepaper