Network of Momentum - leaderless BFT dual ledger architecture

DRAFT v0.1, 31 March 2020

The Zenon Team
portal@zenon.network

***Abstract*—This paper introduces a new, fast and scalable decentralized ledger system called Network of Momentum that achieves high throughput transactions and enables a new form of distributed applications. The paper begins with a short history about distributed systems together with the core components of the existing architectures, alongside with their challenges - the consensus mechanism, the underlying transactional data structure, the smart contracts layer, then it presents the proposed architecture that is leaderless in its nature and is based on a dual ledger system called Network of Momentum that uses virtual voting for consensus. It also provides a brief introduction into a framework that employs unikernels to run distributed applications, but because it is beyond the scope of the paper, a more comprehensive study will follow as a yellow paper. Subsequently, the paper describes some classical attack scenarios and how to surpass them, analyzes the complexity and key protocol parameters and draws conclusions together with discussing related potential research directions.**

***Keywords*: decentralization, byzantine fault tolerance, permissionless consensus, P2P network, protocol design**

I. Introduction

Interest in decentralized systems was reignited by the rise of Bitcoin [1] born in the midst of the 2008 financial crisis and paved the way to countless research initiatives and innovative technologies in the space of computer science and beyond, with focus in areas of cryptography, distributed systems and game theory. A new concept, the blockchain, was popularized by emergent cryptocurrencies that exploited the nature of decentralization to create complex economic systems that culminated with the implementation of the first general-purpose bytecode execution platform, Ethereum [2], and enabled trusted computation among a group of mutually distrustful participants and further materialized in a myriad of decentralized applications ranging from creating self-sovereign identities, peer-to-peer energy markets, prediction markets, improving supply chain logistics to complex financial instruments. A wave of innovation was fueled by their success in the market and shaped a rich landscape of thousands of cryptocurrencies. Consequently, several consensus algorithms and new ledger data structures have emerged for decentralized systems, each of which retains interesting capabilities and unique properties as we will explore in the following sections. This paper presents a decentralized ledger system that features a leaderless, fully-local and scalable consensus algorithm based on virtual voting coupled with proof of work and proof of stake anti-sybil [6] mechanisms that reaches eventual consensus with probability one. The concept of virtual voting was known long before in the literature, starting with the pioneering paper "Byzantine-Resistant Total Ordering Algorithms" [5] by Moser and Melliar-Smith where they formulated four algorithms to establish a total ordering from network events. The peculiar concept of virtual voting that later reappeared in other papers such as Hashgraph [8], PARSEC [9] or Blockmania [51], was the ability to execute a virtual agreement protocol, as authors Moser and Melliar-Smith cleverly observed long before and exploited the fact that votes weren't explicitly contained in the messages, but were deducted from the causal relationships between them. The contributions of this paper are outlined as follows: the protocol comprises of a dual ledger architecture, a meta-DAG created by participating consensus nodes, a projection of the meta-DAG that represents the transactional ledger, a proof-of-work link between relayed transactions emitted by clients, together with the following properties and functions: a vote-weighting function based on proof of stake for participating consensus nodes, an incentivization scheme based of proof of work, a difficulty oracle and a super-quorum selector. The remainder of the paper is organized as follows: in Section II we will discuss basic notions about the Bitcoin protocol and smart contracts and in Section III we will provide some insights about various state-of-the art components that comprises a decentralized ledger system. We build our system on a dual directed acyclic graph based architecture called Network of Momentum that

employs a consensus algorithm that uses a virtual voting technique in association with proof of work (PoW) [3] and proof of stake (PoS) [4] that we present in Section V. We then show in Section VI how the network can withstand different attacks and threat models. In section VII we analyze the protocol parameters together with the complexity and outline the cryptoeconomic system. In Section VIII we conclude with a summary and discuss future research directions.

II. Background

In this section, we present the core concepts of distributed ledgers. We then explore a simplified specification of the Bitcoin protocol and present a short review about smart contracts.

A. From distributed to decentralized ledgers

A distributed ledger is a consensus of replicated and synchronized digital data structure shared between multiple nodes in a peer to peer network. Each node replicates and saves an identical copy of the ledger, updating it independently from the rest of the network. The updating process is based on a consensus algorithm where nodes vote which copy is correct; once the consensus has been reached, all the other nodes update their ledger accordingly. An important aspect of a distributed ledger system is that there is no central authority to enforce rules and therefore no single point of failure, so the integrity and security are accomplished by using a consensus algorithm and cryptographic mechanisms.

A consensus algorithm is at the core of a distributed ledger system - it ensures that the nodes agree on a unique order in which entries are appended. An essential aspect of a consensus algorithm is fault tolerance - the property that enables a system to continue operating normally in the presence of one or more faults. Therefore the consensus algorithm must be resistant to different types of faults, that can be either unreliable or malicious nodes attempting to hijack the system.

There are two main categories of failures a node may be subjected to: crash failures and Byzantine failures. Crash failures occur when nodes suddenly stop and do not resume operation. Byzantine failures are arbitrary faults presenting different symptoms to different observers - in a decentralized environment they may occur as a result of malicious activity.

The problem of designing a system that can cope with byzantine faults was formulated and presented as the Byzantine Generals Problem [7], hence a consensus protocol tolerating byzantine failures must be resilient to any possible error that can appear.

Distributed ledgers can be further categorized, depending on the nature of the environment i.e. public or private, into permissioned or permissionless: this determines the participation eligibility of nodes or users that can join the system.

In a permissionless distributed ledger, such as Bitcoin, anyone can become a node and participate in the consensus process for determining the ledger state or commit to the shared state by invoking transactions. A permissioned distributed ledger e.g. blockchain consortium, in contrast is operated by a set of entities that can identify and decide what nodes can join and update the shared state and even can control the transaction issuance. We will refer to permissionless distributed ledgers as decentralized ledgers to emphasize this distinction.

B. Bitcoin

The history of cryptocurrencies started in 2009 when an anonymous figure known by the pseudonym of ”Satoshi Nakamoto” released the first Bitcoin client and mined the first block of the Bitcoin timechain, thus successfully managing to solve the decades old problem of double-spending in a permissionless environment. The release of the first Bitcoin client marked the inception of a completely decentralized electronic cash system that facilitates pseudonymous payments without any trusted third parties.

The problem of double-spending in a decentralized network was solved by Satoshi using a ”distributed timestamp server” that consists of a proof of work mechanism and an incentivization scheme using an unstructured peer to peer network with an unknown number of participants susceptible of sybil identities together with a method of determining the ”legitimate” ledger by each participant independently.

The consensus protocol is commonly known, although informal, as Proof of Work, and is often encountered in literature as the ”Nakamoto protocol” family, implemented in a wide array of cryptocurrency networks; it virtually uses the longest chain of most accumulated proof of work selection rule to probabilistically determine the valid timechain. A de facto formalization of the Bitcoin protocol isn't broadly accepted by the academic community given the difficulty in providing a good generalized definition - that ultimately depends on the tight interaction between the various parts that make up Bitcoin.

A Bitcoin account consists of a public and private key pair; an address is the hash of the public key and is used to receive coins, while the private key of the account is used to authorize the transfer of coins. A

transaction consists of three data fields - inputs, outputs and metadata; the metadata holds generic information - the hash and size of the transaction, the number of inputs and outputs and a lock time field. The transaction fee represents the difference between the total value of the inputs and the total value of the outputs. If the validation procedure passes, the transaction is broadcasted to the network; a correct node only broadcasts it once, in case it receives the same transaction multiple times. Finally, the transaction is included into the mempool and awaits confirmations i.e. become embedded into the blockchain. Every node can participate in the consensus protocol, a process known as mining, by computing a cryptographic proof-of-work puzzle; if the node finds a solution, the newly created block containing transactions from the mempool is propagated through the peer to peer network and if valid, it is appended to the blockchain. We will discuss different Proof-of-X algorithms in Section III.

Although Bitcoin taken as a whole is fulfilling its purpose for more than a decade under real-world conditions, there have been studies of individual components of Bitcoin that point out limits or even theoretical design flaws of the protocol; for example, according to Eyal and Sirer et al. Bitcoin is incentive incompatible due to selfish mining [20].

C. Smart contracts

Another interesting topic is the programmable component of a cryptocurrency, smart contracts and decentralized applications. The basic idea is that one can run arbitrary quasi-Turing complete code in a decentralized setting, from simple smart contracts for automated payments to more complex applications.

Early blockchain networks such as Bitcoin have a simple, Turing-incomplete stack-based scripting language used as a locking mechanism for transaction outputs: "The script is actually a predicate. It's just an equation that evaluates to true or false. Predicate is a long and unfamiliar word so I called it script." [36].

The smart contract concept was pioneered by Szabo in [21]. With the expansion of the Internet it became clear that cryptographic enforcement of agreements can become a cornerstone for human cooperation in a digital world. Ethereum was the first project to successfully implement the smart contract paradigm. A smart contract is a piece of code typically written in a higher level language, for example Solidity, and compiled down to bytecode interpreted by a specialized virtual machine. In Ethereum's case, the resulting bytecode is ran inside the Ethereum Virtual Machine that is present in every node of the blockchain network and the execution of instructions is deterministic and verifiable by all participating nodes.

A desired property of smart contracts is their immutability: once a smart contract is deployed it cannot be modified by third parties. This can be a double edge sword, amplifying both the strength of censorship resistance and the weakness of poorly written code. Fortunately, there are techniques to overcome bugs by upgrading the vulnerable smart contracts code with the use of proxy contracts or by using a formal verification framework [22].

Overall, smart contracts dramatically increase the specter of use-cases for DLTs, from allowing basic conditional payments to more complex business logic. We will provide a deeper analysis of this topic and describe our proposed solution in Section V.

III. State of the art

A. Ledger types

Even though the terms distributed ledger and blockchain are often used inter-changeably in the literature, there is a subtle distinction between them which is worth headlining: a blockchain is just subset of the larger superset of distributed ledgers. One of the most important aspects when designing a new architecture is the distributed ledger component that describes how transactions are embedded.

Definition 1 *A decentralized ledger is defined as a distributed data structure with entries that are digital records of actions, in a permissionless environment.*

1) Blockchain: The most common decentralized ledger is the blockchain. One definition for the blockchain is a distributed, decentralized, public ledger in the form of a cryptographically secured linked list of blocks holding transactions, without a central authority or coordinator, managed by multiple entities participating in a peer to peer network, usually in a trust minimized context.

The digitally signed transactions are hashed and encoded into a cryptographically tamper-evident data structure known as a Merkle tree, forming a "block". Each block contains a cryptographic hash of the prior block, creating a linear list of blocks linked by tamper-evident hash pointers, thus enabling a tamper-resistant way to confirm the integrity of previous blocks, all the way back to the genesis block.

Additional integrity measures are used to combat potentially malicious, byzantine adversaries, such as the

requirement that a block hash is smaller than a given target e.g. in Nakamoto protocol family, or a multi-signature or threshold signature over a block, by the nodes participating in the blockchain network.

For example, in order for a block to be added to the Bitcoin ledger, the nodes have to participate in a lottery where their chances are proportional with the amount of computational work invested to find a solution for a cryptographic hash puzzle that allows them to link it with the previous block.

Once a valid block is appended by the miner, all the transactions from that block become finalized and immutable, however due to the independent Poisson processes in the block proposal race, more than one miner may propose to extend the blockchain using different blocks with corresponding valid proof of work solutions at roughly the same time, leading to a fork; this results in one of the competing blocks to land on a fork and subsequently be discarded given the longest chain selection rule employed by the Nakamoto protocol.

For this reason, in [19], Garay et al. presents a framework to capture the properties of liveness, validity and agreement of the Nakamoto consensus protocol by three chain based properties: common-prefix, chain growth and chain quality. With these in mind, the proof of work based Nakamoto protocol can be modeled as a probabilistic Byzantine agreement protocol.

However, what we described earlier - the proof of work Nakamoto consensus of Bitcoin, is not the only consensus algorithm for blockchains. There are now many other consensus algorithms that can power a blockchain network. We will describe them in the following section.

Even if the blockchain paradigm has many advantages such as robustness and the fact that is well studied and more understandable, it ultimately sacrifices scalability due to the limited number of transactions that fit in any given block.

2) Directed Acyclic Graph: Nonetheless, in order to boost scalability and increase transaction processing, the linear data structure has been expanded into nonlinear forms such as block graphs and trees [17], [18]. A DAG, as the name implies, is a finite directed graph with no directed cycles. For example, IOTA [23] proposed a custom DAG called Tangle. The Tangle has a genesis block, then all the transactions are linked to each other forming a DAG. The Tangle is basically a DAG where each new transaction is linked to two previous transactions, an architecture that in theory would allow the structure to be highly scalable. Other cryptocurrencies that implemented DAG structures are Byteball [25], which is similar to IOTA, and Avalanche [10], which has a more complex model. Another interesting DAG ledger approach is represented by Hashgraph, developed by the company Swirlds and used as the backbone of the Hedera cryptocurrency. The hashgraph is a special type of DAG where each record is a message that can accommodate several transactions. Furthermore blockless, nonlinear data structures are also adopted in many recent architecture designs for their potential to enhance transaction throughput.

3) Holochain: Another decentralized ledger is the Holochain [26], a concept implemented in the Holo cryptocurrency presented as a scalable agent-centric distributed computing platform. Holochain applies the ”trustless” principle of decentralized ledgers by making context specific ledgers where trust exists contextually and locally, being interoperable with other ledgers that are similarly trustful. It is a combination of multiple concepts: distributed hash tables, git and bittorent. In Holochain, each node runs its ”local source chain”, an append-only log and operate autonomously.

Rather than storing a copy of the full ledger on every node of the network and enforcing a universal consensus protocol, Holochain takes an agent-centric approach and divides the data to many different nodes and establishes access only to the data that is useful for a particular node. Nodes validate each other based on jointly relevant information and on context specific rules.

4) Block-lattice: The last ledger data structure we analyze is the block-lattice. First used by Nano cryptocurrency [27], it is designed for throughput and scalability: every user has its own autonomous account-chain, that can be updated independently from the rest. The blocks from different account-chains acknowledge each other and collectively form a mesh-like structure. Because the account-chains can grow concurrently, the throughput can be quickly scaled up. The blocklattice has many advantages - scalability, simplicity, and it can be secure provided it is implemented with an adequate consensus algorithm.

Our architecture is based on a dual ledger approach: a generic DAG, called the meta-DAG used for the consensus layer and a block-lattice data structure used to store the transactional data.

We have separated the ledger architecture in order to achieve a better complexity and faster processing times when a user wants to query nodes for transactional data. An overview presenting the advantages and disadvantages for different types of ledgers can be seen

in Table 1.

B. Consensus types

The key component of a distributed system that enables all participants to agree on a state without a central authority is the consensus algorithm.

Definition 2 *Consensus is the process of committing entries to the decentralized ledger that complies with a set of well-defined rules that are enforced by all honest network participants after an entry containing transactions is accepted.*

Different consensus algorithms have distinctive design choices that have a considerable impact on the system's performance, including its transaction throughput, scalability and fault tolerance.

Therefore consensus algorithms have trade-offs between the level of security and performance. We will list security and performance properties that are essential for a permissionless consensus algorithm designed for a decentralized ledger system.

- **Adversary resistance**: Indicates the threshold of byzantine nodes that can be tolerated by the consensus algorithm.
- **Sybil resistance**: Specifies if the consensus algorithm implements an anti-sybil mechanism. For example, the consensus algorithm should have a mechanism to prevent the generation of sybil identities in a permissionless environment.
- **Accountability & non-repudiation**: Indicates if the consensus protocol implements an identity system and cryptographic signatures.
- **Denial of Service resistance**: Specifies if the consensus algorithm implements a denial of service defense mechanism. For instance, some leader based consensus algorithms are susceptible to DoS attacks.
- **Censorship resistance**: Indicates if the consensus algorithm is censorship resistant. For example, it precludes external entities from trying to censor transactions.

From the perspective of quantifying the performance of a consensus algorithm, we will highlight the following performance indicators:

- **Throughput**: Represents the number of TPS (i.e. transactions per second) a consensus algorithm can process.
- **Scalability**: Represents the ability for a system to expand without degrading performance. Generally, the throughput of a system is directly affected by scalability.
- **Fault tolerance threshold**: Indicates an upper bound of faulty nodes that directly impacts the performance of the consensus algorithm. For example, some consensus algorithms have an optimistic regime that favors performance.
- **Latency**: Also known as finality in this context, it represents the time it takes for a transaction to become settled in the ledger.

We will review some of the most important consensus protocol families that are at the core of countless decentralized systems.

1) Proof-of-Work: Proof of work was initially designed as a spam mitigation solution [14] and involves the asymmetry in terms of resource usage between two separate entities, the prover and the verifier. The prover performs a resource-intensive task in order to obtain a result and presents it to the verifier for validation - the asymmetry comes from the fact that the validation of the proof requires only a fraction of the resources invested into its generation.

The core concept of the Proof of Work consensus algorithm is the competition of nodes in finding solutions for a cryptographic hash puzzle that satisfies a difficulty requirement based on the measurement of the total hash power in order to maintain a specified rate of puzzle solutions per time interval; once a solution is found, nodes create and cryptographically link the block with the tip of the blockchain and advertize it over the peer to peer network.

For a cryptographically secure hash function H(·) like SHA-256 in the case of Bitcoin, and a given difficulty level D(h), each single query to H(·) is an independent and identically distributed Bernoulli trial with a success probability described by the following equation:

$$Pr(y : \mathcal{H}(x\|y) \leq D(h)) = 2^{-h}$$

Different implementations of PoW algorithms require different rates at which solutions are found in a given time interval: in the case of Bitcoin this rate is one solution for every 600 seconds, and for Ethereum every 15 seconds. The corresponding time period is directly correlated with the underlying data structure: for instance, Ethereum implements GHOST [30] to optimally determine the path that has the most computation work done upon to accommodate the short block times.

Cryptocurrencies that have a PoW based consensus algorithm employ different classes of PoW, (e.g. compute-bound PoW, memory-bound PoW, chained PoW or other

TABLE I: Ledger types

Ledger type	Advantages	Disadvantages
Blockchain	Wide-scale adoption in industry Robust and well studied	Limited scalability
DAG	Can scale better than a blockchain	Increased attack surface
Block-lattice, Holochain	Account independence Asynchronous transactional model	Decentralization trade-offs
HashGraph	The consensus is derived locally from the graph	Potential delays for reaching consensus Graph bloat

custom implementations) to obtain some desired properties like ASIC-resistance, such as to avoid some forms of miner centralization.

In a decentralized model, PoW consensus assumes that a majority of hashing power is controlled by honest parties.

2) Proof-of-Stake: Proof of Stake was proposed as candidate to solve a number of potential shortcomings of the proof of work consensus such as energy consumption, miner centralization and certain types of economic attacks.

One of the first cryptocurrencies to implement PoS as a consensus algorithm in their blockchain network was Peercoin [47], released in 2012; the success sparked a wave of innovation, culminating with the Ouroboros protocol, a provably secure PoS algorithm [31] that is at the core of the Cardano cryptocurrency [24].

The core notion of the PoS consensus algorithm is the block creation process that requires a proof that the participating node owns a certain number of coins. Naive implementations of PoS may lead to unexpected problems that naturally don't occur in PoW based cryptocurrencies: the "nothing at stake" problem [49], short or long range attacks, coin-age accumulation, pre-computing attacks, stake-grinding or cartel formation attacks.

Some of the problems can be avoided by a slashing mechanism within the protocol during the block creation process. A node that wants to participate in the consensus algorithm first needs to lock a certain number of coins; this stake represents a collateral. The node that seals the stake is called a leader, forger, or minter in PoS terminology and can lose this collateral through a technique called slashing, in case it deviates from the protocol specification.

3) Delegated Proof-of-Stake: A popular variant of PoS is the delegated proof of stake consensus algorithm (dPoS), where each user can choose to delegate its coins to a node that takes part in the consensus algorithm.

The idea is similar to the committees found in classical consensus models; some cryptocurrencies have a fixed, static set of delegators, while others utilize a dynamic size of the set of delegators; as for the dPoS terminology, in a blockchain network they are called block producers. For instance EOS [41] and Lisk [44] employ a fixed number of 21 and 101 delegators respectively, while Tezos [42] takes a different approach with a technique that allows anyone to amount delegated coins such that it meets the threshold to become a baker, in exchange returning for this service a certain proportion of the block rewards back to the delegating party.

4) Proof-of-X: Proof-of-X consensus algorithms are extending the concept beyond work and stake to non-interactively prove a commitment of computational resources.

A PoX scheme should be resistant to puzzle grinding (i.e. the puzzle must meet several criteria to satisfy completeness, soundness, non-invertibility, and freshness), including aggregation or outsourcing [34] of the computational resources and manipulation of the leader election process. This leads to hybridizations such as Proof of Activity, a combination of PoW and PoS used in Decred [39] or Proof of Importance, used in NEM [40] that is based on PoS and an "importance score" calculated from the net coin transfers from an account.

For instance, PoX can also be designed to incentivize distributed storage provision like proof of capacity, proof of storage [16], proof of retrievability and proofs of space and time.

In Proof of Elapsed Time, each of the block producers has to wait a random time to create a block; an equivalent for it would be a verifiable delay function [45], suitable for the permissionless regime. PoET and similar variants use a trusted execution environment to enforce these random delays. One notable example is Hyperledger [37], but a major drawback is that it is only suitable for a permissioned environment given that the process depends on a non-standard secure hardware enclave within the processor.

5) Hybrid BFT consensus: Byzantine fault-tolerant consensus protocols are a vast topic with a long history of research and development, and became candidates for

hybridization with current blockchain consensus algorithms: for example, PoW-BFT and PoS-BFT are most widespread.

Due to the scalability constraints of the BFT protocol in terms of communication overhead, the above hybridization is intended to decouple the committee election from the actual consensus.

The primary functionality of the PoX mechanism is to simulate the leader election in the traditional BFT protocols; thus it is utilized for managing a stable consensus committee for each BFT protocol instance.

An example of PoW-BFT hybrid architecture is Ziliqa [29] that uses PoW to allow identity establishment and group assignment and multiple rounds of PBFT over the consensus committee.

As for the PoS-BFT hybrid architecture, a prominent example is the Tendermint protocol [15]; the committee formation of the block validators is made using a PoS process that involves a bond deposit. Moreover, the size of the bond stake is proportional to the voting power and the leader of the committee is designated using a round-robin strategy.

Another alternative is delegated BFT, where the initial problem of the byzantine generals is slightly adapted with representative leaders for the generals. This, however, centralizes the network in a similar way to dPoS, even if the delegates can be replaced; a notable example implementing dBFT is NEO [43].

Algorand [52] also relies on a customized hybrid PoS-BFT consensus protocol for committing transactions. The PoS mechanism is used to compute via cryptographic sortition the probability of a node to participate in a committee proportional to its stake and the total current stake in the network and a verifiable random function is used to generate a publicly verifiable BFT-committee of random nodes.

Generally, hybrid BFT protocols enhance overall network throughput and provide faster finality times in contrast with Nakamoto inspired protocols.

6) Cellular Automata: New Kind of Network [50] proposes a new consensus algorithm that is based on cellular automata and a mathematical framework developed for the Ising model. The nodes act as cells and together with a message-passing algorithm based only on sparse local neighbors and a MVCA algorithm [48] (i.e. Majority Vote Cellular Automata, an algorithm that uses majority vote as updating rules for the cells) they reach consensus.

7) Virtual voting: Virtual voting is a concept introduced by authors Moser and Melliar-Smith in 1999, where the main idea is to interpret messages as virtual processes and execute a consensus algorithm to derive a total ordering of events. An example of a cryptocurrency network that uses virtual voting to derive consensus is Hedera. They implement a modified virtual voting consensus algorithm, called gossip about gossip, where nodes gossip information not only about transactions but also about the gossip they receive. In this way the nodes will arrive at the same conclusion, knowing how votes would be casted if a voting process would happen, so they only compute a local "virtual" vote in order to achieve consensus.

Other systems that use virtual voting techniques are [51] and [35], where the communication DAG is subsequently interpreted to derive consensus. We will also present a customized implementation of virtual voting in section VI.

Our architecture will implement a virtual voting scheme based on a hybridization between proof of stake and proof of work. A summary of the consensus types can be seen in Table 2.

IV. Prerequisites

A. Definitions

We will use a few definitions needed for a better understanding of our ledger architecture and the consensus protocol.

Definition 3. *A node is a software program running on a device that participates in the NoM network and complies to the protocol specification. It can directly participate in the consensus algorithm, manage accounts, observe traffic and relay transactions.*

There are three kinds of nodes in NoM, depending on their contributions towards the health of the network, as follows:

- Trusting nodes called *Sentry nodes*. A basic type of node, lightweight in the sense that they only store the transaction ledger or a pruned version of it. A light node only monitors traffic for specific accounts allowing minimal network usage and resources.
- Trustless nodes called *Sentinel nodes*. A trustless node is similar to a Pillar node, but only acts as an observer, it doesn't participate in the consensus algorithm. It carries out the creation of PoW links for transactions and requires moderate resources to operate.
- Consensus nodes called *Pillar nodes*. They participate in the consensus protocol and have

TABLE II: Consensus types

Consensus type	Advantages	Disadvantages
PoW	Enables large scale decentralization	Doesn't scale well with a traditional approach
PoS	Power efficient in comparison with PoW	More attack vectors non-existent in PoW
DPoS	More scalable than PoS	More susceptible to centralization
PoET	Power efficient, suitable for permissioned environments	Susceptible to third party interferences e.g. in the case of hardware enclaves
BFT consensus	Well studied and understood Based on quorums	Need to be coupled with other mechanisms for permissionless networks High complexity
Cellular Automata	Good scalability	Complex Requires specific network topology
Virtual Voting	Efficiency of the voting process	Delays can happen until a transaction is accepted

information about the transactions made in the network by users. A Pillar requires additional resources as it relays network traffic from other Pillars and processes it.

Definition 4. *Pillar nodes representing more than a fraction of the locked stake in any given epoch ϵ are called supermajority, as follows:*

$$\zeta = \frac{N * 2}{3} + 1$$

Definition 5. *Representative – A sentinel node that knows about user transactions.*

Definition 6. *Transaction – A transaction can be of two types: ordinary send transactions with a corresponding receive transaction or special transactions for different circumstances: to mark the entrance in a new round of consensus, the finishing PoW for a round or smart transactions regarding zApps. An ordinary transaction contains the address of the sender and its balance, the address of the receiver, and metadata containing hashes for PoW solutions.*

Definition 6. *zApp – A distributed application that is based on an unikernel controlled by a smart contract.*

Definition 7. *Epoch - The transactions are grouped in consensus rounds called epochs. In every epoch, each of the nodes that participate in the consensus algorithm must compute a PoW with adjustable difficulty. The finish of a PoW is marked by a special transaction, which is then sent through the network via broadcast. After receiving the finishing PoW transaction from ζ, the node enters in the next epoch and marks this with a particular transaction.*

Definition 8. *Virtual voting - the concept that voting is not done with explicit messages. Instead, a node computes the state of the ledger based on the information received throughout many epochs from the network. We will show that after some epochs, if a node can reach to a conclusion regarding a transaction, all the honest nodes will reach the same conclusion.*

Definition 9. *Broadcast – the process of sending the finishing PoW and the transactions for undecided epochs to all Pillar nodes.*

B. Network Model

We consider the execution of the protocol in an open, dynamic, distributed system enabled by a message oriented transport protocol for data packets exchange, where nodes can join or leave freely. Nodes represent the core infrastructure of the network and clients are external in the sense that they are issuing transactions for nodes to agree upon. We assume an asymmetric cryptographic signature scheme that enables participants to authenticate messages. A node is considered honest if it follows the protocol as described or byzantine if it deviates arbitrarily from the protocol specification. In addition, the system is considered asynchronous i.e. there are no bounds on messages delivery.

C. Goals and assumptions

NoM allows Pillar nodes to agree on an ordered log of transactions and attains three goals with respect to the log:

- **Liveness goal** – Even if there is a number of active byzantine nodes and under additional assumptions about network conditions, the system will eventually make progress i.e. continue appending transactions to the log.
- **Safety goal** – With high probability all the honest nodes will reach to the same conclusion regarding

the order of the transactions; specifically, if an honest node accepts transaction T (i.e. it is included in the log), then any future transactions accepted by other honest nodes will appear in a log that already contains T.
- **Finality goal** – Once a transaction is included into the log and confirmed by honest nodes, it will remain confirmed in the log, despite any actions from byzantine nodes.
- **Scalability goal** – The network will keep optimal confirmation times for non-conflicting transactions, even if the number of nodes is constantly increasing.

Starting with these assumptions, the byzantine agreement consensus algorithm has to simultaneously meet the following three properties:

- **Validity**: If all correct processes propose the same value ϑ, then any correct process that decides, decides ϑ.
- **Agreement**: No two correct processes decide differently.
- **Termination**: Every correct process eventually decides.

The first two properties are safety properties, i.e., properties that state that "bad things" cannot happen and the last one is a liveness property, i.e. a property that states that "good things" must happen.

D. Important attributes

When designing a distributed system, there are some attributes any distributed system exhibits and we want to obtain a good balance between them:

- *Consistency*: when a node requests the state of the system – in our case the distributed ledger, the consistency means that we will obtain the most recent state of the system.
- *Availability*: For a request for the state of the ledger, there must be an answer, even if the answer does not reflect the latest state of the ledger.
- *Partition tolerance*: The system continues to be functional even if there are message failures in the system.

The CAP theorem [46] states that it is impossible to achieve all three properties simultaneously. However, we design the network to have partition tolerance, availability and *eventual consistency* – after a number of retries, a node will eventually find the state of the network at the time of the request. The eventual consistency is preferred over availability in many other distributed systems.

E. Theorems

- T1. **Availability**. If a user will emit a transaction to an honest node, in the absence of attacks (e.g. denial of service), all honest nodes will receive that transaction.
- T2. **Validity**. A double spend is not possible assuming a supermajority of honest nodes.
- T3. **Safety**. If there is a supermajority of honest nodes, once a node reaches to a conclusion regarding a transaction, all the honest nodes will reach the same conclusion.
- T4. **Liveness**. If the number of byzantine nodes is bounded i.e. $f < \frac{1}{3}$, the system will come to an agreement about the total ordering of the transactions.
- T5. **Scalability**. Transaction times processing will grow linearly with the number of pillar nodes.
- T6. **Finality**. If a transaction is confirmed (i.e. is part of the ledger), it will remain forever in the ledger.

Proofs for the theorems are available in Appendix A.

V. NoM Ledger and Consensus

A. NoM Ledger

Our proposed NoM ledger architecture consists of two separate ledgers – the actual ledger consisting of settled transactions structured as a block-lattice where there are stored independent individual user account chains, and a DAG called the meta-DAG that contains the transactions required by the virtual voting algorithm.

The block-lattice consists of actual transactions appearing in the network that are settled - send, receive and zApp related transactions.

Every user has an account chain that is independently updated from other account chains as the virtual voting progresses.

The flow of issuing a transaction is as follows: a user will have assigned some representative nodes, sentinel nodes that will process their transactions and that can be queried in order to pull new information regarding the account chain or the state of the ledger.

However, in order to prevent denial of service attacks, the queries can require a fee that needs to be applied in order to return a valid response: for example, a user can

use the sentinel nodes for querying the state of the ledger or sentry nodes to get updates regarding its account chain.

As we highlighted earlier in the Definitions subsection, not all network participants are also consensus nodes; only full nodes (i.e. pillar and sentinel nodes) keep both the transactional ledger and consensus ledger used for the virtual voting process. The consensus ledger is organized in virtual epochs, and the consensus is achieved per epoch.

B. PoW Links

In this subsection, we will introduce a novel anti-sybil and anti-spam mechanism called proof of work links that will enhance connectivity within the network and limit certain attacks by sharing their commitment and contributing resources for routing and efficient data delivery.

There are two goals this mechanism aims to achieve: the first one is to strengthen the ledger by adding weight into it (i.e. recording the resulting work of the PoW link) and the second is to further incentivize the sentinel nodes to safeguard the network against different attacks such as spam or distributed denial of service.

A PoW link needs to satisfy the following conditions:

- Only *Sentinel nodes* can participate in the creation of a proof of work link.
- Only the private key owner of a Sentinel node can produce valid signatures to be used the composition process a proof of work link.
- The signature attached to any transaction should be unique (i.e. only one signature will be considered for any key pair).
- A minimum overall weight for the proof of work link is required in order to be considered valid; a difficulty parameter is computed in order to obtain a *min_target_weight* for the transaction.

Users constantly issue transactions that are disseminated to a number of sentinels equal with $\log \sigma_n$, where σ_n is the total number of the sentinels that a user is aware of.

Sentinel nodes prove the receipt of the transaction by adding a small PoW computation and other additional data (e.g. digital signature, metadata), then they will relay that transaction to another sentinel node in a random manner. Basically, for each transaction, the sentinels will attach a small PoW computation to it, then they will randomly relay the transaction to other sentinels, which will continue to add PoW and further relay that transaction, constructing a PoW link; a minimum number of three hops is required by a *min_relay_dimension* constant, and an upper bound will be dynamically imposed by a difficulty parameter. The proof of work will be calculated with respect to the transaction fee paid by the user to issue the transaction. The sentinels will continue to forward the transactions to other sentinels until the proof of work meets a specific weight threshold; when the PoW link is complete, the transaction will be sent to a pseudorandomly chosen consensus node (i.e. pillar node). Finally, the PoW link will serve an additional objective in the consensus algorithm, representing an eliminatory criteria to select between two conflicting transactions in case of a double spend. An overview about the dissemination and composition of a PoW link can be seen in Algorithm 1.

Algorithm 1 PoW Link Algorithm

```
1:  procedure PoW_Link
2:      while True do
3:          t ← ReceiveTransaction();
4:          if t.Sender() in Users then
5:              t.weight += ComputePoW(t, t.fee);
6:              t.links++;
7:              s ← ChooseRandom(Sentinels);
8:              SendToSentinel(t,s);
9:          else
10:             t.weight += ComputePoW(t, t.fee);
11:             t.links++;
12:             if t.weight ≥ min_target_weight then
13:                 p ← ChooseRandom(Pillars);
14:                 SendToPillar(t,p);
15:             else
16:                 s ← ChooseRandom(Sentinels);
17:                 SendToSentinel(t,s);
18:             end if
19:         end if
20:     end while
21: end procedure
```

A visual representation of this algorithm can also be seen in Figure 1.

C. The consensus explained

In this paragraph, we will describe how the consensus is achieved in NoM.

Clients can connect to specific nodes called representatives and submit transactions for processing. For this consensus algorithm description we will also suppose that there are no malicious actors and there are no ongoing attacks (e.g. denial of service, eclipse attacks,

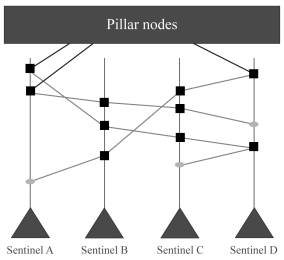

Fig. 1: PoW Link messages

etc.); we reserve for treating those particular cases in the following section.

In order to make a transaction, a user needs to inform the representatives, in this case sentinel nodes. If the user is running a sentinel node, it will further disseminate the transaction to other sentinels in order to prevent eclipse attacks; the PoW link generation starts and develops as described by the algorithm from the previous paragraph.

Let's shift the focus to what happens with the transactions when they reach a pillar node. As time progresses pillars are incorporating transactions into the consensus ledger and initially mark them as not decided. The reason is that a user can make a double spend and disseminate it to two different representatives. After a number of epochs, all the pillar nodes will detect with high probability the double spend transaction and they will vote only one to remain in the ledger. After some time, we will presumably have many transactions – send and receive pairs, that are individually held by consensus nodes.

We will further detail how the normal operation of the consensus algorithm takes place, assuming only honest participants.

First stage

At the start of the algorithm, let's suppose that all transactions are marked as being in epoch ϵ_0. So when a user issues a transaction, the pillar node will keep the transaction received from a sentinel node if valid, and marks it as belonging to epoch ϵ_0. At the same time, all the pillars will compute a proof of work with adjustable difficulty, in order to keep the epoch duration within some time bounds, for example 1 minute. After a pillar node finishes its proof of work, it will broadcast a special transaction to all other pillar nodes from the network, to announce them that it has finished the PoW for epoch ϵ_0. The special transaction includes additional information like the number of the current epoch and represents the fact that the pillar node is ready to enter into the next epoch; we will call this the "finishing PoW" transaction. After receiving the finishing PoW from ζ pillar nodes, it will proceed to the next epoch, ϵ_1.

Note that the pillar could receive a finishing PoW transaction before it finished computing its own PoW transaction and other transactions. If it hasn't completed the PoW yet, it will abandon it and broadcast a message with its transactions, then marks itself as being in epoch ϵ_1.

Second stage

The process continues with other transactions from users, but marked now as belonging to epoch ϵ_1. Again, the node will start working for the proof of work in order to enter in epoch ϵ_2; again, it will enter after receiving "finishing PoW" transactions from ζ and so on.

Notice that if a node already received messages from a supermajority of pillar nodes informing it that they finished the PoW for the current epoch, it will abort the proof of work generation and enter automatically into the next epoch. The reason for aborting the proof of work is that there will be no reward for it, because at a later epoch the nodes will compute which were the fastest nodes for that particular epoch and issue rewards accordingly.

Now, let's consider two random, independent pillars from the network in a certain moment during the consensus algorithm: between the finishing PoW transaction and entering into the next epoch. When a node sends a broadcast, it also includes all the transactions it knows about from other nodes, so in perfect network conditions after a broadcast all the nodes will know about the transactions between the start of epoch and the finishing PoW directly from it. However, they will not know about the transactions between the finishing PoW and the next epoch. After the finishing PoW transaction from epoch ϵ_1, the node will broadcast all its transactions, including those between the finishing PoW from epoch ϵ_0 and the start of epoch ϵ_1.

Third stage

At the beginning of epoch ϵ_2, every node will know about all the transactions from epoch ϵ_0 – they will receive information about the transactions between the start of epoch and the finishing PoW transaction from ζ.

This will happen because, in the meantime, all the pillars found out about those transactions at the end of epoch ϵ_2 and they also send a broadcast at epoch ϵ_2, but they will have only one copy regarding the messages from the finishing PoW and the start of epoch ϵ_1 –

only the transaction made by the pillar itself. However, at epoch ϵ_2, all the nodes will make another broadcast. Let's suppose that all pillars will have a copy of those messages between the finishing PoW in epoch ϵ_0 and the start of epoch ϵ_2.

Fourth stage

At the start of epoch ϵ_3, pillars will have messages from ζ regarding all the transactions between epoch ϵ_0 and epoch ϵ_1 – the transactions between the start of epoch ϵ_0 and the finishing PoW transaction will be discovered at the start of epoch ϵ_2, and the remaining transactions will be found at the start of epoch ϵ_3, so any pillar will apply the same ordering to them.

Later stages

However, special cases can appear where not all messages will arrive to all pillars, so even if the pillar will receive messages from ζ, not all the pillars will have all the transactions.

Let's assume that only a simple majority will have them. In that situation, the pillar will have to wait for the next epochs until all the transactions between epoch ϵ_0 and epoch ϵ_1 will be confirmed by a supermajority of nodes. For theoretical reasons, if a conclusion can't be reached after a certain number of epochs, a coin round will be needed - every honest pillar will randomly vote on transactions, in order to prevent an attacker controlling the internet traffic from deducing the votes. The node will further broadcast its vote in the next epoch.

Now, regarding the previous theorems, if a node will know the transactions between epoch ϵ_0 and epoch ϵ_1 and it will apply a deterministic ordering algorithm and in case of double spends, a deterministic tie-breaker algorithm, thus all the remaining honest nodes will arrive at the same decision. After the node will have a supermajority of messages with all the transactions between epoch ϵ_0 and ϵ_1 (as per definition 4, the supermajority is weighted with a proof of stake mechanism), it will start to virtually vote on the ordering.

The vote is not actually a real one in the sense that it doesn't involve sending additional network messages, but a set of rules that define a deterministic way to order the transactions, such as: the PoW link weight, the timestamp when they arrived at the pillar and, as tiebreakers, the hash of the transaction. After a node will order the transactions, it will know that the order is the same for all the nodes so it will mark them in the ledger and for every transaction it will put an id to know the number of the transaction. So, in optimal network conditions, a pillar will show the new ledger to a sentinel after three epochs – if a user have made a transaction at epoch ϵ_0, it will find about it at epoch ϵ_3.

In the next section we will discuss some attack scenarios and also the complexity of the consensus algorithm.

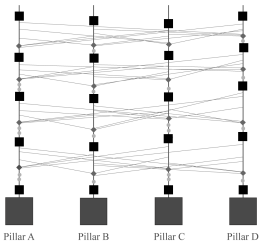

Fig. 2: Consensus algorithm visualization

The consensus mechanism can be better visualized in Figure 2. There are four pillars, A, B, C and D. Each pillar computes a proof of work during an epoch, receiving transactions supplied by sentinel nodes. At the beginning of epoch ϵ_1, A doesn't know the transactions that happen between the finishing PoW transaction for B and the starting of epoch ϵ_1 for B. At the start of epoch ϵ_2, A has received those transactions, but only from B. At the beginning of epoch ϵ_3, A has received messages from all the pillars regarding the transactions at epoch ϵ_0, including those between the finishing PoW and the start of the epoch.

The consensus is summarized in Algorithm 2.

D. Pillars PoW pools

In order for the pillars to be competitive in the process of producing the proof of work, they will have the possibility to outsource it using the mining pool concept. This will create a market efficient ecosystem that will further strengthen the network and clients committing resources for Pillar pools will be rewarded proportionally to their contribution of processing power. We are also investigating the use of a custom difficulty adjustment mechanism that will balance between ASIC-friendly and ASIC-resistant hashing algorithms in order to improve network security and obtain a higher degree of decentralization. We will defer a detailed specification for a later date.

E. Unikernels and distributed applications

The following subsection is describing the core component of our future distributed apps system, called zApps, which will be integrated into the NoM architecture. We are introducing a novel design based on

Algorithm 2 Consensus Algorithm

```
1: procedure CONSENSUS ALGORITHM
2:     Thread WaitForTransactions = new WaitThread();
3:     ComputePoW = new ComputeThread();
4:     WaitForTransactions.run();
5:     Epoch ← 0;
6:     min_consensus_delay ← 3;
7:     min_coin_round_delay ← 5;
8:     LastConsensusEpoch ← 0;
9:     while True do
10:        count ← 0;
11:        zeta ← 2/3 * Nodes + 1;
12:        while count < zeta do
13:            count+ = AcceptedBroadcast();
14:            if Epoch < CurrentEpoch() then
15:                Epoch ← CurrentEpoch();
16:            end if
17:            if ComputePow.finish() then
18:                BroadcastPoWandTxs();
19:            end if
20:        end while
21:        if !ComputePoW.ended() then
22:            ComputePoW.abort();
23:            BroadcastTxs();
24:        end if
25:        Epoch ← Epoch + 1
26:        if Epoch ≥ min_consensus_delay then
27:            for t in UnresolvedTransactions do
28:                if t.Epoch() ≤ Epoch − min_consensus_delay then
29:                    if countVotes(t) > zeta then
30:                        TxEpochs[Epoch].add(t);
31:                        counter[Epoch]++;
32:                    end if
33:                end if
34:                if t.Epoch() ≤ Epoch − min_consensus_delay − min_coin_round_delay then
35:                    if t.HasConflict() then
36:                        tc ← t.GetConflict();
37:                        coin ← random(0, 1);
38:                        if coin = 0 then
39:                            remove(t);
40:                        else
41:                            remove(tc);
42:                        end if
43:                    end if
44:                end if
45:            end for
```

Algorithm 2 Consensus algorithm (continued)

```
46:             for i ← Epoch − min_consensus_delay; i >= LastConsensusEpoch; i −= 1 do
47:                 HaveToOrder = [];
48:                 if counter[i] = TotalTransactions[i] then
49:                     LastConsensusEpoch = min(LastConsensusEpoch, i);
50:                     for t in TxEpochs[Epoch] do
51:                         HaveToOrder.add(t);
52:                         UnresolvedTransactions.remove(t);
53:                     end for
54:                 end if
55:             end for
56:             if HaveToOrder.size() > 0 then
57:                 sort(HaveToOrder);
58:                 for t in HaveToOrder do Ledger.add(t);
59:                 end for
60:             end if
61:         end if
62:     end while
63: end procedure
```

unikernels [54] to expand the limits of smart contracts and enable complex computational tasks. An ideal version of a zApps platform should exhibit the following characteristics:

- *Security*: The environment for the applications should be sandboxed with granular permission policies.
- *Immutability*: The zApps should be immutable in the sense that they cannot be modified or tampered with. Running zApps on untrusted hardware should be trustless and deterministic and one should expect consistent results.
- *Privacy*: To provide means that protect the privacy of participants, both internal between them and external from third parties, based on secure multi-party computation protocols.

With the rise of the unikernel (i.e. minimal stand alone virtual machine), these properties can be achieved using a smart contract layer to create a hybrid system suitable for complex workloads. The most important advantages in using an unikernel based approach are in terms of security - they are completely isolated from the host and performance - they are lightweight and run at native speeds.

Regarding security, unikernels are systems designed with a single process and a limited number of system calls, further reducing the attack surface in terms of remote code execution, shellcode attacks, etc. They further limit potential attacks by lacking a user based system: the configuration and management are carried out by smart contracts that handle certain aspects of the applications' life cycle such as compilation or deployment. By using this approach, errors like accidental configuration alterations can be prevented and the exploitation can exclusively be carried only on the end application. Performance is another issue for many systems; unikernels have several benefits such as fast booting times (e.g. can boot several orders of magnitude faster than normal virtual machines), avoiding context switching and using minimum system resources.

The infrastructure to run zApps will consist of special nodes that will have specific requirements (e.g. minimum resources in terms of connectivity, hardware specifications, collateral, etc.). The idea is similar to a decentralized infrastructure-as-a-service model where users can have access to an instant computing infrastructure, managed and provisioned within the NoM network.

The unikernel design ensures both internal and external protection for the underlying infrastructure that performs the execution. Furthermore, we are analyzing several economical models to implement in order to ensure that an application will reach the end of an execution without issues, including providing a way for the user to hire several other nodes to verify certain checkpoints for example.

Periodically, the users will need to pay for the zApps usage; this system will be designed in a similar way gas [53] is implemented for smart contracts as a fees mech-

anism that prevents abuse and circumvent the Turing completeness property (e.g. infinite loops). This process will be automatized through a series of smart contracts that will be used to manage the zApps operation and the transfer of gas. The user will have the possibility to cancel at any time the execution of the app, by either explicitly sending an abort command or by not paying the corresponding gas to the node.

In Figure 3 we describe how unikernels can be integrated into the system.

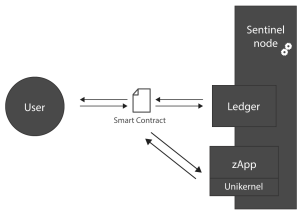

Fig. 3: Unikernels and zApps

VI. Possible attacks

Since every distributed network is designed to withstand byzantine activity, it is necessary to highlight the most important related attack vectors for a decentralized ledger system.

[55] gives us a detailed list of attacks targeted at the Bitcoin network. We will analyze the most important problems and how the network confronts them.

An in-depth analysis of these attack vectors and mitigation solutions are presented in the following subsections.

A. Double-spending

One of the first classes of attacks and one of the most important problem when designing a decentralized ledger system is the double-spending problem. Bitcoin emerged as the most influential cryptocurrency network because it was the first to solve the double-spending problem in a decentralized environment. The integrity property of the consensus algorithm states that for any honest node, accepted transactions are consistent among honest nodes (i.e. double-spending cannot occur).

Any user can initiate a double spend if it distributes two transactions with the same parent to different pillars; however, after a number of epochs all of the pillars will know about both transactions and all honest pillars will decide on the same transaction to be retained in the ledger as confirmed, using the predefined rules we presented earlier and discarding the double spend.

If a pillar node gets corrupted and is acting maliciously, it can accept a double-spending transaction and send contradictory information to other pillars. This attack scenario is improbable but entirely possible; pillars unlike miners have no economic incentive to attack the network; nevertheless, it is an irrational attack because attempting to violate the ledger would be destructive to the network as a whole, which in turn would undermine the validity of their investment. Byzantine pillars are accounted for and as long as there is only a malicious minority, after a number of epochs all honest pillars will eventually detect the double spend and discard it. We are also considering a penalizing algorithm to punish such behavior for corrupted pillars.

B. Forking

A forking attack can happen at any stage, including from the start of the ledger - the attack is also known as ledger cloning and all pure proof of stake solutions are vulnerable to it.

However, NoM is not affected because we employ two proof of work mechanisms: virtuous transactions need a PoW threshold satisfied obtained from the PoW link algorithm to get into the ledger, together with the proof of work mechanism used by the pillars.

For each pillar, a proof of work translated into computational power is required to complete each epoch and an attacker needs to outrun all honest pillars in order to obtain a heavier ledger in terms of accumulated PoW. Also, we take into account that a user cannot be tricked to land on a forked ledger. A malicious adversary can only try to convince new nodes that his fork it the real ledger, but there are certain ways to deter this: a node will connect to several pillars to get the ledger. Upon successful synchronization it will observe which is the heaviest one before legitimizing it. Therefore forking the ledger at any time is an irrational attack.

C. DNS Attacks

A DNS attack may occur when a new user wants to join the network and connects to a list of peers obtained from a DNS query; this is a common network discovery mechanism used by many major blockchain networks. If the attacker manages to inject his IP addresses instead of the original ones, the new user can be compromised.

This attack can be part of a chain of attacks; there is research regarding DNS attacks and there are some solutions to this kind of attack [56] [57]. This attack

can't be totally neglected and is still a valid attack for any type of distributed network, but there are many viable solutions that are already implemented in real-world systems.

D. Eclipse attacks

An eclipse attack means that an attacker manages to isolate a user from the rest of the network. Even if an eclipse attack is not possible for a pillar, because it has access to all the other ones and it is very unlikely to replace pillar identities with fake ones, an eclipse attack can occur for a single user which connects to a percentage of the pillars, as discussed earlier.

After an eclipse attack, the user will only see what the attacker wants, and this can have bad consequences, like a double spend. However, this attack is common to other decentralized systems and there are some strategies, for example random connection at the nodes in the beginning, making very unlikely for an attacker to accomplish this attack.

E. Sybil attacks

Sybil attacks are among the most destructive for a decentralized network because if an entity is able to create a large number nodes on a machine in order to gain control over the network. However, because the voting is weighted based on the pillar's stake, adding more nodes will not gain the attacker extra power in the consensus algorithm. Therefore there are no advantages to be attained with a sybil attack.

F. DoS attacks

The denial of service attack can occur if a malicious user sends a lot of transactions to the sentinels. We made this attack harder by adding a transaction fee, which means that the attacker will make the sentinels unavailable at the cost of investing resources in the system, which is a positive aspect for the network and a negative one for the attacker, taking into account that the consensus is unaffected.

G. Consensus delay

A consensus delay can happen if the attacker can interfere with network traffic among pillar nodes. This attack is unlikely to cause damage if there is a sufficient number of active pillars in the network; an attacker could interfere with messages between a certain number of nodes – for example, by initiating a DDoS attack. In this case, the consensus may be delayed resulting in unconfirmed transactions; still, the probability of reaching a supermajority is one as the number of epochs increases.

The worst case scenario is an attacker taking down some pillar nodes, but it will have a limited impact on the network that will continue to confirm transactions with lag. There are some mechanisms to prevent this like a detection of the consensus delay mechanism and a special coin round.

H. Majority attack

This is one of the most destructive attacks that can occur in a decentralized network, however, it is highly improbable due to the incentive mechanisms of the system.

If an attacker can somehow obtain pillars that have a cumulated stake of 51%, it can add or alter new transactions. It can't modify transactions that happened in the past, but nonetheless, the network is compromised in this case. In order to avoid this attack, the honest majority assumption should hold at all times:

$$\texttt{Honest nodes} > \texttt{Malicious nodes}$$

Even with a stake of $\frac{\zeta}{2}+1$ the attacker can overpower the network - because there will be no honest supermajority.

This is worth mentioning as a hard limit for the network. So, in order to function properly, a vital condition for the network is:

$$\texttt{Honest nodes} >= 2 * \texttt{Malicious nodes} + 1$$

VII. Parameters and complexity

A. Complexity analysis

We will now discuss the complexity of the algorithms. We can express the complexity regarding the number of messages and time. As we have seen earlier, during an epoch, users make transactions, that are first distributed at a small number of sentinels and that are further forwarded to other sentinels – we can consider this $\mathcal{O}(log(S))$ in terms of messages, where S is the number of the sentinels.

The most consuming time happens in the consensus algorithm, where all the pillar nodes send broadcasts to all other pillar nodes, so during an epoch the total number of messages is $\mathcal{O}(N^2)$. However, if we assume good network conditions and if we consider the calculations per pillar, during an epoch, we will obtain that a pillar has to send a message to every other pillar node and receive a message from every other pillar node. In conclusion, we will obtain $\mathcal{O}(N)$ number of messages per pillar, and $\mathcal{O}(N)$ time complexity. Due to the fact that we assumed good network conditions, the total time spent for a broadcast is small enough in order for the

network to support thousands of messages per second during the consensus epochs.

Because the network has a representative system, if there are N users which send M messages per epoch each and we have K number of pillars and we suppose a user sends a transaction to $\log(K)$ pillars ($\log(K)$ sentinels which further send to a pillar), that means we will have τ messages, where

$$\tau = log(K) * M$$

A pillar will support θ messages during an epoch, so for every $\log(K)$ messages a pillar will receive only one, with

$$\theta = \frac{M * log(K)}{K}$$

During optimal network conditions, with speeds of 100 Mb/s, and for a packet of 100 kb, a pillar can support 1000 messages per second. For a number of 1024 pillars, however, each 10 pillars will have the same messages, but there will be 102 groups of 10 pillars with different messages, so the total number of messages per second in the network can top at 100000 TPS. However, this is from a purely theoretical perspective, but it gives as an upper bound for the calculations. The real speed will decrease due to the cost of the broadcasts. A pillar receives θ messages during an epoch and sends only K messages, but those messages will be bigger and will take a larger amount of time to be propagated throughout the network.

In the future, we plan to make some experiments to quantify the supported number of transactions per second. Another research direction that we will tackle is to see how the traditional broadcast will compare to more scalable alternatives. In general, there is a trade-off between latency and the number of messages. If the bandwidth is good enough and the number of pillars is reasonable, the traditional broadcast method has the advantage of having $\mathcal{O}(1)$ latency.

A scalable type of broadcast can be made, for example, by sending the broadcast in rounds - the user will send a transaction to $\log(K)$ pillars, then they will further transmit the information, each of them to other $\log(K)$ pillars and so on.

The number of the messages will be much lower, $\mathcal{O}$(log K), but there will be some latency involved - for example, in [58] they have a latency of $\mathcal{O}\ \frac{logK}{log(logK)}$.

B. Finding a representative

The problem of finding a representative is important because there can be some attacks, like the eclipse attack, if a user connects to malicious nodes. This is the reason why each user has to send his transactions to $\log(S)$, where S is the number of the sentinels. Because sentinel nodes are interested in maintaining a healthy network by computing PoW links and consuming fees, we can assume that most of them will be honest. Coupled with a random selection algorithm for choosing the sentinels, even for 40 sentinels, if 12 are corrupt sentinels (33%), the probability of choosing all sentinels being malicious is under 0.1%. The same logic can be applied for pillars. Another problem is how to choose the initial bootstrapping nodes. The user will connect to a number of nodes and will choose randomly among them, and they will send a list of sentinels from the network to it. This way, the chance for an attack is very small.

C. Cryptoeconomic system

In order for the network to function properly, a cryptoeconomic layer will be put in place for all the network participants. The sentinel nodes will benefit from the fees by consuming them in order to compute the PoW links. Also, the sentinels can enable a separate fee system for user queries that retrieve information about the state of the ledger. The pillars will be incentivized for computing the proof of work for the current epoch. If a pillar receives a supermajority of messages from the next epoch before finishing its PoW, it will no longer be rewarded. This is designed to ensure a network wide competitiveness: the pillars can outsource the proof of work, acting as mining pools to amass resources and rewarding accordingly the clients that supply them with computational power. The last type of incentivization is for the zApp platform, where a gas like system will be implemented in order to reward nodes that support this feature.

D. Managing epochs

In order for the transactions to be confirmed as fast as possible, there are two important factors that need to be accounted for – the proof of work should be completed in a decent time frame, according to a desired difficulty for an epoch and the messages should have a high delivery success rate. The second condition is harder to accomplish, but in general, we can safely assume that a negligible amount of messages will be lost due to network connectivity issues. Regarding the the proof of work, in order to maintain an adequate time frame we employ a difficulty mechanism and an incentivization scheme that was described earlier.

Thus, if non-competitive pillars that are overrun during an epoch by a supermajority of pillars will not

receive rewards for their work. This competition will ensure that the epochs will have similar periods and that transactions are approved as fast as possible and included in the ledger. Also, after receiving ζ messages from the network, an honest pillar must abandon its proof of work and automatically enter into the next epoch. If there is not clear which are the winner nodes, each pillar will compute the faster winner, then the runner-up and so on. Even if a pillar tries to cheat by saying it belongs to the list and sends its proof of work later, the other honest pillars will know that the faulty pillar tried to mislead, as they will see in the ledger that the other ones have received its PoW later.

Just for theoretical reasons, if there is an attacker who can control the internet traffic and messages between pillars, we have introduced a shared coin epoch: if there are more than four consecutive epochs that don't end up with a conclusion (i.e. either it is a tie or the majority criteria isn't met), all honest pillars will vote randomly. This way, the attacker will have only half chances to guess what is the decision of honest pillars, and after a number of epochs the probability of reaching consensus will be 1. However, this technique is implemented only for theoretical completeness, in a real-world system the probability for a coin round is insignificant.

The expected time to finish is $\mathcal{O}(1)$, given this round, and the probability of finishing is

$$P = 1 - \prod_{r=1}^{\infty} \frac{1}{2} = 1$$

Regarding the minimum number of epochs ν needed for a transaction in order to be included in the ledger, we have the following equation:

$$3 <= \nu < \infty$$

.

E. Adjusting the difficulty of PoW

As we have previously stated, the idea behind the proof of work mechanism has multiple advantages – prevents ledger cloning, acts as an additional anti-sybil layer and provides a fair incentivization scheme for the consensus nodes. The PoW will be adjusted in order to keep an epoch at a constant time, for example 1 minute. The algorithm will check at every epoch the time needed for every pillar to solve the proof of work, will remove the outliers and then will compute a median time. We plan to release a self-balancing difficulty algorithm that will use both ASIC-friendly and ASIC-resistant hashing algorithms; if the difficulty is below a threshold an ASIC-friendly hashing algorithm will be activated, and also if the difficulty is above a threshold, an ASIC-resistant will come into effect.

The time of one epoch is responsible for the minimum time after a transaction is confirmed – a transaction is confirmed after three or four epochs, in the best case, so for a one minute epoch it will least at least three minutes in order for the transaction to be confirmed.

If we note with Δt_i the difference between the timestamps for two consecutive finishing PoW transactions, the mean time will be

$$Avg = \frac{\sum_{i=1}^{n} \delta t_i}{n} \quad , \quad \Delta t_i \notin \texttt{outliers}$$

F. Replacing regular quorums with proof of stake

The consensus timeline is divided into virtual epochs. In a centralized, non-malicious environment classical distributed consensus algorithms use a quorum for the voting process: every node has an equally $\frac{1}{N}$, N being the total number of nodes. In our case, a decentralized, byzantine environment this approach is vulnerable to sybil attacks where a malicious entity can gain an unfair advantage and manipulate the voting process. That's why nodes can lock a certain amount of stake in order to obtain different roles in the network, e.g. to become sentinel and pillar nodes. At the start of each epoch, all nodes determine the stake weight of all the nodes in the network. In the case of pillar nodes, network participants can directly delegate stake.

The virtual voting process is determined on the basis of the total stake during a virtual epoch. Pillar nodes with stake make the decisions within the consensus algorithm to finalize transactions. Nodes can freely unlock the stake at any moment; however, consensus nodes have to wait for a period of time known as "unstaking period". Upon deciding transactions during the next epoch, nodes process all transactions relating to locking, delegating and unlocking stake, and update the staking stats of the nodes for the next epoch.

These mechanisms aim to keep a healthy system, by involving all network participants to collaborate towards a common good.

VIII. Conclusions and future work

This work presents a new decentralized system architecture, namely a new decentralized ledger that employs a virtual voting-based consensus. We first presented the most important works regarding ledger types, consensus algorithms, and smart contracts. We continued by making some definitions and assumptions,

stating some properties and theorems, then we described the core of the architecture - the dual ledger system and the consensus algorithm. We analyzed frequent attack scenarios, the complexity and how to choose different protocol parameters for optimal performance. The Network of Momentum has a continuous cycle of research and is still under active research; as a result, some parts will require further clarification or revision. We also plan to release a technical yellow paper dedicated to a detailed presentation of the zApps component and other improvements.

ACKNOWLEDGMENT

We want to thank Professor Z for his support and for the continuous research and development program.

REFERENCES

[1] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. [Online]. Available: https://bitcoin.org/bitcoin.pdf. 2008
[2] Ethereum Foundation, Ethereum wire protocol. [Online]. Available: https://github.com/ethereum/wiki/wiki/Ethereum-Wire-Protocol
[3] Juels, Ari; Brainard, John. Client puzzles: A cryptographic defense against connection depletion attacks. NDSS. 1999.
[4] King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake (2012)
[5] Moser, L. E. and Melliar-Smith. Byzantine-resistant total ordering algorithms. Information and Computation. 1999
[6] J. R. Douceur, The sybil attack, in Peer-to-Peer Systems, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260.
[7] The Byzantine Generals Problem, LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE. SRI International.
[8] Dr. Leemon Baird, Mance Harmon, and Paul Madsen. [Online]. Available: https://www.hedera.com. 2019
[9] Pierre Chevalier, Bart lomiej Kami´nski, Fraser Hutchison, Qi Ma, and Spandan Sharma. Protocol for asynchronous, reliable, secure and efficient consensus (PARSEC). [Online]. Available: http://docs.maidsafe.net/ Whitepapers/pdf/PARSEC.pdf, Jun 2018.
[10] Team Rocket, Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies, 2018. [Online]. Available: https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV
[11] Driscoll, K.; Hall, B.; Paulitsch, M.; Zumsteg, P.; Sivencrona, H. The Real Byzantine Generals. The 23rd Digital Avionics Systems Conference. 2004
[12] Lamport et al. L. Lamport, R. Shostak, M. Pease. The Byzantine generals problem. ACM Trans.on Programming. 1982
[13] L. Lamport. The part-time parliament. ACM TOCS 16, 2 (1998), 133–169.
[14] Dwork, Cynthia and Naor, Moni. Pricing via processing or combatting junk mail Annual International Cryptology Conference. 1992
[15] J. Kwon, Tendermint: Consensus without mining (draft), Self-published Paper, fall 2014. [Online]. Available: https://tendermint.com/static/docs/tendermint.pdf/
[16] Filecoin: A decentralized storage network, Protocol Labs.
[17] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, Spectre: A fast and scalable cryptocurrency protocol, IACR Cryptology ePrint Archive, Report 2016/1159, 2016, https://eprint.iacr.org/2016/1159/
[18] A. Kiayias and G. Panagiotakos, On trees, chains and fast transactions in the blockchain, IACR Cryptology ePrint Archive, Report 2016/545, 2016, https://eprint.iacr.org/2016/545/
[19] J. Garay, A. Kiayias, and N. Leonardos, The bitcoin backbone protocol: Analysis and applications, in Advances in Cryptology - EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Part II, Sofia, Bulgaria, Apr. 2015, pp. 281–310.
[20] I. Eyal and E. G. Sirer, Majority is not enough: Bitcoin mining is vulnerable, in Financial Cryptography and Data Security: 18th International Conference, Christ Church, Barbados, Mar. 2014, pp. 436–454.
[21] Nick Szabo. Formalizing and securing relationships on public networks. First Monday. 1997.
[22] Bruno Bernardo et al. Mi-Cho-Coq, a framework for certifying Tezos Smart Contracts. arxiv:1909.08671 [Online]. 2019
[23] S. Popov, The tangle, cit. on, p. 131, 2016.
[24] Cardano Platform. [Online] Available: https://www.cardanohub.org/en/home/
[25] A. Churyumov, Byteball: A decentralized system for storage and transfer of value. [Online]. Available: https://byteball.org/Byteball.pdf. 2016
[26] A. Brock et al. Holo Green Paper. [Online]. Available: https://files.holo.host/2018/03/Holo-Green-Paper.pdf. 2018
[27] Colin LeMahieu. 2018. Nano: A Feeless Distributed Cryptocurrency Network. [Online]. Available: https://nano.org/en/whitepaper/
[28] M. Castro, B. Liskov. Practical Byzantine Fault Tolerance. 3rd OSDI. 1999.
[29] The Ziliqa Team. The ziliqa technical whitepaper. [Online]. Available: https://docs.zilliqa.com/whitepaper.pdf. 2017
[30] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. Financial Cryptography and Data Security, 2015.
[31] Kiayias,A.,Russell,A.,David,B.,andOliynykov,R. Ouroboros: A provably secure proof-of-stake blockchain protocol. Annual International Cryptology Conference pp. 357-388. August, 2017.
[32] Steemit article. [Online]. Available: https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper/
[33] NXT Whitepaper. [Online]. Available: www.nxtdocs.jelurida.com/Nxt_Whitepaper
[34] A. Miller, A. Kosba, J. Katz, and E. Shi, Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions, in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '15. Denver, CO: ACM, Oct. 2015, pp. 680–691.
[35] Aleph: Efficient Atomic Broadcast in Asynchronous Networks with Byzantine Nodes. [Online]. Available: https://arxiv.org/pdf/1908.05156.pdf
[36] S. Nakamoto. Bitcoin Talk Forum. [Online]. Available: https://bitcointalk.org/index.php?topic=195.msg1611
[37] Tamas Blummer et al. An introduction to Hyperledger. [Online]. Available: An Introduction to Hyperledger. 2018.
[38] Chain whitepaper. [Online]. Available: https://crypto.com/images/chain_whitepaper.pdf. 2019
[39] Decred (DCR) – Whitepaper. [Online]. Available: https://decred.org/research/buterin2014.pdf. 2014
[40] NEM - Whitepaper. [Online]. Available: https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf. 2018
[41] EOS Platform. [Online] Available: https://eos.io/
[42] Tezos Platform. [Online] Available: https://tezos.com/
[43] NEO Platform. [Online] Available: https://neo.org/

[44] Lisk Whitepaper. [Online] Available: https://github.com/slasheks/lisk- whitepaper/blob/development/LiskWhitepaper.md
[45] Verifiable Delay Functions. Dan Boneh, Joseph Bonneau, Benedikt Bunz, and Ben Fisch. [Online] Available: https://eprint.iacr.org/2018/601.pdf
[46] Perspectives on the CAP Theorem. [Online]. Available: https://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf
[47] Peercoin discussion forum, discussion 2524. [Online]. Available: https://talk.peercoin.net/t/the-complete-guide-to-minting/
[48] Majority-Vote Cellular Automata, Ising Dynamics, and P-Completeness Cristopher Moore. [Online] Available: https://arxiv.org/pdf/cond-mat/9701118.pdf
[49] Proof of Stake versus Proof of Work. [Online]. Available: http://bitfury.com/content/5- white- papers-research/pos-vs-pow-1.0.2.pdf
[50] NKN Lab. NKN: a Scalable Self-Evolving and Self-Incentivized Decentralized Network. [Online]. Available: https://www.nkn.org/doc/NKN_Whitepaper.pdf. 2018
[51] George Danezis, David Hrycyszyn. Blockmania: from Block DAGs to Consensus. arXiv:1809.01620. 2018
[52] Jing Chen, Silvio Micali. Alogrand. arxiv:1607.01341v9. 2017
[53] Empirically Analyzing Ethereum's Gas Mechanism. Renlord Yang, Toby Murray, Paul Rimba, Udaya Parampalli. [Online] Available: https://arxiv.org/pdf/1905.00553.pdf
[54] Unikernels: Library Operating Systems for the Cloud, Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand and Jon Crowcroft. [Online] Available: http://mort.io/publications/pdf/asplos13-unikernels.pdf
[55] Muhamaad Saad et al. Exploring the Attack Surface of Blockchain: A Systematic Overview. arxiv:1904.03487. 2019
[56] P. Silva. Dnssec: The antidote to DNS cache poisoning and other dns attacks, A F5 Networks, Inc. Technical Brief. 2009.
[57] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems,. ACM Computing Surveys (CSUR), vol. 39, no. 1, p. 3. 2007.
[58] Scalable Byzantine Reliable Broadcast. Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi

APPENDIX

A. Proof of theorems

Proof of Theorem 1

Proof. If a node emits a transaction and it is received by its representatives, the representatives will send the information about the transaction to the pillars that will further broadcast it, and every honest pillar node will know about the transaction. For maximizing the chance of receiving the transaction, a node will have not just one representative, but a logarithmic size of the total number of sentinel nodes. After three epochs, if the transaction is not seen throughout the network upon a request, it means with high probability that the initial transaction wasn't received. However, in the absence of a DoS, the transaction will eventually be seen by the entire network. □

Proof of Theorem 2

Proof. Suppose that we have a double spend and we also assume that the rest of the pillars are malicious i.e. K - ζ. After they all broadcast them and all have them both, one will be chosen based on the rules and the other discarded. If the majority vote for transaction A and the minority instead keep B, a fork will be created, but no double spend. □

Proof of Theorem 3

Proof. When pillars are in epoch ϵ_k, all of them have received all transactions from epoch ϵ_{k-3}. The pillars from the majority will choose one transaction based on the rules and the minority will choose the other, not reaching the total number of votes required and it will not be integrated. If they still decide to accept it, a fork will be created. □

Proof of Theorem 4

Proof. After the first pillar finishes the proof of work and sends it along with the transaction, the rest of the honest nodes will follow and the vote count for this transaction will reach majority, so it will be integrated into the ledger, even if the minority of malicious pillars will decide not to relay it. □

Proof of Theorem 5

Proof. The complexity of the messages is M * $\log(K)$ per round so when a new pillar joins the network, it will become M * $\log(K+1)$. An increase of the number of pillars is almost unnoticed in the complexity. □

Proof of Theorem 6

Proof. Transaction times processing will grow sublogarithmically with the number of pillar nodes. □