Logos Network

A Highly Scalable Decentralized Transaction System

Michael Zochowski

***Abstract*—The Logos Network is a distributed, trustless transaction network designed for extreme scalability. It achieves high transaction throughput and low confirmation latency using a novel structure. Each account on the network has an individual chain that tracks all of its transactions, allowing independent transactions to be processed in parallel. A main settlement chain provides universal synchronization of nodes as well as dynamic validator sets. Sharding via *Polis* adds a second dimension of parallelism to the network and additional scalability. Transactions are validated and approved by a small number of delegates elected via a representative system, which reduces redundant operations while preserving safety. Consensus is achieved via *Axios*, a delegated PBFT-based algorithm optimized for speed. The network provides both accountable safety and plausible liveness through slashing conditions and other game theoretic incentives. Future work will additionally implement scripting and data capabilities, which will allow for rich transactions. Logos's guiding principles that marry rigorous mathematics, game theory, and practicality help it provide the best value transfer network for all transactions, from IoT microtransactions to multi-million dollar commercial payments.**

I. Introduction

The limits of traditional blockchain consensus networks are increasingly evident. As interest in cryptocurrency soars, existing networks have become bogged down by the wave of demand for limited network resources. As we move beyond the tip of mainstream adoption and look toward applications that are useful to global society, it is clear that scalability is an existential question that must be answered for distributed ledger technology to succeed.

This paper introduces Logos, a novel transaction network designed for high scalability while maintaining decentralization and trustlessness. Logos is built for applications that require cheap, efficient, intelligent, and secure transfers of economic value that range in size from microtransactions to industry-scale. Such a network is critical for the growth of peer-to-peer systems, such as the internet of things (IoT), in a truly global context.

Logos builds on the best components of existing cryptocurrency networks while introducing innovations that enable its primary goals: practical scalability, fully aligned incentive structure, and security in the presence of Byzantine faults. Its base data structure is the *chain mesh*, a directed acyclic graph (DAG) consisting of a dedicated chain of transactions for each account. Transactions are processed in parallel rather than in blocks, and consensus is achieved via *Axios*, a delegated Practical Byzantine Fault Tolerance (dPBFT) algorithm that provides both liveness and safety as long as more than ⅔ of nodes by holdings are honest. Stake slashing conditions combined with a robust reward structure align validator and overall network interests. A main settlement blockchain called the *archive* serves as a ledger for finalized transactions, delegate elections, and native network governance.

This hybrid structure allows Logos to process transactions in parallel while maintaining universal consensus. A state sharding algorithm called *Polis* adds a second dimension of parallelism that provides scaling in the number of validating nodes. Future work will implement bounded smart contracts, which will enable sophisticated structures like payment channels, data capabilities, and interoperability with other crypto networks. These developments will allow Logos to provide unbounded scaling of rich transactions without sacrificing performance or security.

The current network implementation has a model estimated capacity of 2,500 transactions/second with a confirmation latency of less than 3 seconds. As we roll out additional improvements such as *Polis*, we project a theoretical capacity in excess of 100,000 transactions/second under reasonable network assumptions. Deployment of sophisticated second layer solutions like Lightning will allow for further unbounded growth in capacity. Thus, Logos represents a substantial improvement over both centralized networks such as Visa and decentralized networks such as Bitcoin and Ethereum as a practical transaction network.

Guiding Principles: Logos is built on a rigorous mathematical approach that takes into account both theoretical and practical concerns. We place a heavy emphasis on game theoretically sound economic incentives to encourage network growth and safety. This mindset will continue to guide future development. Importantly, we recognize the limitations of a single network and ascribe to a "No Free Lunch" principle: given finite computational resources and the need for universal consensus, a single network cannot provide an arbitrarily broad and complex range of services for an arbitrarily low cost and high performance. As such, rather than trying to overpromise by simultaneously endeavoring to guarantee infinite scalability, fully Turing-complete computation, zero fees, and absolute security, Logos instead focuses on implementing the most practical and scalable transaction network.

II. Comparison to other networks

Logos offers tangible benefits over other types of transaction networks.

A. Centralized Networks

Centralized payment networks such as Visa and Paypal provide high capacity and low latency but require a trust relationship that frequently breaks down. Furthermore, these

Logos (logos.network) is developed by Logos Genesis (logosgenesis.org).
Version 1.0, January 29, 2018,

networks rely on single points-of-failure and are vulnerable to attacks that can take down the entire network. The key innovations of cryptocurrencies are the removal of the need for a trusted counterparty and network decentralization, but scaling has proved problematic. Logos provides the decentralization benefits of traditional cryptocurrencies and the scalability and speed of centralized networks.

B. Traditional Blockchain Networks

Logos processes transactions individually in parallel, providing significantly higher bandwidth and lower latency than networks that batch-process transactions in blocks, such as proof-of-work systems like Bitcoin. The chain mesh structure also obviates the possibility of most forks as the contents and order of all chains are known by construction to all honest nodes. This allows greatly simplified network logic as well as rapid finalization of transactions. At the same time, the settlement chain provides universal security guarantees of blockchains.

C. Proof-of-Stake Networks

Like proof-of-stake (PoS) networks, Logos achieves energy efficiency and transaction scalability by replacing the computational investment required in proof-of-work networks with an economic investment in the network that acts as a guarantee of honest validator behavior. However, this scheme requires a high degree of coordination between validators to approve transactions, as opposed to proof-of-work where validators mine transaction blocks independently. Unless the validator set is explicitly limited, this will result in anti-scaling as the number of nodes increase. Several PoS consensus algorithms address this problem by requiring validators have a high minimum stake, but this effectively disenfranchises the vast majority of users who lack the resources to meet that threshold. Conversely, Logos and other delegated proof-of-stake (dPoS) networks allow all users to vote on delegates to act as validators while preserving safety.

D. DAG-based Networks

DAG-based networks provide huge scalability potential, but these gains are typically at the expense of security. Logos uses a hybrid DAG and blockchain structure that gives security guarantees not offered by other DAG-based networks. For instance, RaiBlocks's consensus algorithm is not Byzantine fault tolerant, and thus neither safety nor liveness are guaranteed. Furthermore, it requires active validators but offers no incentive, which will inevitably result in a Tragedy of the Commons outcome as the number of transactions increase and thus the cost of validation. Similarly, IOTA's "tangle" structure offers only probabilistic guarantees contingent on dubious assumptions, such as that the tree periodically converges to an arbitrarily small cutset. IOTA has also struggled in practice, exhibiting high latency, dropped transactions, and the need for centralized validation to combat poor security. Claims that these problems will disappear with scale are thus far unproven.

III. Use Cases

Logos is a practical solution for all applications requiring efficient and secure transfer of value.

A. IoT Microtransactions

The Internet of Things market is projected to generate $450 billion in annual revenues by 2020 [1]. To achieve their full potential, IoT devices will require the ability of interact with and transfer value to other devices on a trustless basis. Logos provides an ideal solution to economically connect the IoT. It offers unbounded scalability, rich transactions, and minimal transaction fees, enabling frictionless and intelligent transfers. Furthermore, Logos is carefully designed to accommodate low footprint light clients that can run on limited hardware without having to continually trust a third party full node. Finally, its programmability allows for interoperability with other networks used by IoT devices, both public and private.

B. Merchant Payments

Merchants require secure payment networks that provide both high transaction throughput and low confirmation latency. Logos is designed around these qualities, and the Logos ecosystem will develop applications that enable seamless network integration with merchant systems. Logos offers additional benefits of global usability (particularly valuable for internet commerce), low fees (orders of magnitude cheaper than the 2.5% charged by credit card networks), and transaction finality (which will reduce losses due to fraud and charge backs). Logos is able to finalize transactions faster than other cryptocurrencies due to the chain mesh structure, allowing merchants to be confident that transactions will not be reversed as soon as they are approved. This drastically reduces fraud such as improper charge backs, and savings can be passed on to consumers. At the same time, Logos's cryptographically secure account system reduces the risk of theft, protecting users in the event of data breaches and removing the need for trust in online payments.

C. Peer-to-Peer Transfers

Logos is entirely open and trustless, enabling anyone anywhere in the world to join the network and transact with other users in a decentralized manner. Compared to centralized peer-to-peer payment solutions like PayPal and Venmo, users always control their funds, so they cannot be frozen or locked up arbitrarily. The Logos ecosystem will include frameworks that will empower secure and simple peer-to-peer transactions. This functionality can be extended to a diverse range of applications, including game economies, international remittances, and alternative payment systems for countries with unstable currencies.

D. International Trade and Reserves

Many countries will welcome the chance to settle corporate accounts in a currency independent from any single country or central authority. Currently, most countries need to hold

large reserves of international currency to facilitate trade that occurs in dominant currencies like USD and EUR. Countries like China (exports priced in USD) and Russia (commodities priced in USD) are deeply uncomfortable with this status quo, as it ties a substantial portion of their economy to a foreign government and central bank. A non-sovereign, trustless crypto network like Logos is far preferable as a reserve. Furthermore, Logos's highly secure and fast transaction capabilities are an attractive option for companies transacting with foreign entities, as it substantially reduces risk and settlement times while being easy to integrate.

E. Store of Value

In many ways, cryptoassets are ideal stores of value. They are open, decentralized, cryptographically secure, censorship-resistant, mostly non-inflationary, cheap to store, and increasingly easy to use. The main barrier to this use case is the high volatility of crypto prices; however, as successful cryptoassets mature and adoption increases, volatility will dampen significantly. Logos, as a scalable, highly secure, and minimally frictional payment system with interoperability with other systems, is an ideal cryptoasset for store of value. A few dominant, transaction-focused cryptoassets will likely win out as stores of value and have a high chance of displacing much of the $3 trillion of gold used for store of value [2] and the $12 trillion of national fiat currencies held as reserves [3]. Store of value is thus the highest potential source of cryptoasset value, and Logos is well positioned to capture a portion of that value.

IV. Network Objectives and Assumptions

A. Objectives

Logos aims to be the best universal network for decentralized, trustless, and rich transactions. On a practical level, this goal mandates the following network characteristics:

1) *Security*: users must be confident that the network will correctly preserve economic value and transfers
2) *Speed*: the network must provide both high throughput (transactions/second) and low latency (time to confirm transactions) under most circumstances
3) *Accessibility*: anyone anywhere in the world, whether an IoT device sending microtransactions or a large corporation settling invoices, should be able to use the network in an economically and practically efficient manner
4) *Scalability*: the network should gracefully grow to accommodate an arbitrary number of users without degradation of performance
5) *Robustness to Attacks*: the network should be persistent and resistant to censorship or single points of failure
6) *Efficiency*: validation uses a minimum of resources, such as computational power or energy, necessary to meet the other objectives so that economic deadweight loss is minimized

Formally, Logos is a deterministic replicated state machine across a set of nodes, $\mathcal{N}$, that transitions from state to state via operations requested by clients. For the service to be practically useful, it must have the following fault tolerance properties under reasonable assumptions:

1) *Safety:* the service is linearizable, *id est* it behaves like a trusted centralized service that executes operations atomically.
2) *Liveness:* clients eventually receive replies to their requests, *id est* the service does not halt despite individual failures.

The system uses a Byzantine failure model, meaning that faulty processes may deviate arbitrarily from the protocol, such as by sending contradictory messages or failing to respond. Since the service is operating on systems with different infrastructures with their own faults, such as the Internet, we desire *asynchronicity* as much as possible. That is, since it is not possible to know *a priori* if an unresponsive process is Byzantine or merely delayed, the service should accommodate all reasonable response times to a particular request without breaking or delaying independent requests [4].

The CAP Theorem states that it is impossible for a distributed system with potentially unreliable nodes to guarantee both safety and liveness [5]. As such, Logos errs on the side of safety rather than absolute liveness, and instead relies on some weak synchronicity assumptions to give plausible liveness. In the event of an extended network partition, a child network with sufficient support should continue to function properly.

To process requests, the service requires a consensus protocol that satisfies the following properties [6]:

1) *Agreement*: Every correct process must agree on the same value (safety property).
2) *Validity*: If all processes propose the same value v, then all correct processes decide v (non-triviality property).
3) *Integrity*: Every correct process decides at most one value, and if it decides some value v, then v must have been proposed by some process (safety property).
4) *Termination*: Every correct process decides some value (liveness property).

By the FLP impossibility proof, no fully asynchronous system can reach consensus in the presence of even one faulty process [7]. As with the CAP theorem, we must rely on a set of reasonable assumptions to achieve the desired properties.

We must also provision for clients entering the network, either for the first time or after an extended period offline. Logos is structured so that nodes can rely on a *weak subjectivity* assumption similar to that in some Ethereum proof-of-stake proposals [8]. Specifically, a new node can independently verify the system state at time t given (i) the protocol definition, (ii) the set of all operations and validation messages, and (iii) the system state at time t_s such that $t - t_s < F$ for some constant finalization time F. Under this paradigm, nodes reject any proposed changes to any historical state with age greater than F. In fact, Logos provides much stronger finalization guarantees through its consensus algorithm, but it also locks in validator stakes for at least F time to ensure *economic finality* in the interim. System snapshots will be widely disseminated by *potentially* trustworthy sources, such as large companies or nodes in social networks, that are *jointly* trustworthy with

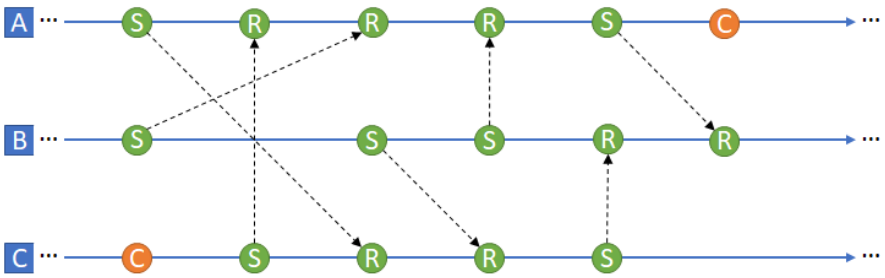

Fig. 1: The chain mesh. Each account has a separate chain of requests, which together form a directed acyclic graph structure. Note that sends are processed independently from receives.

high confidence, which will allow new nodes to bootstrap to the present.

Finally, recognizing that the cryptocurrency ecosystem is rapidly evolving, it is necessary that Logos has a native governance mechanism. This system empowers stake holders to modify system rules and parameters as needed while maintaining integrity. It will allow for seamless upgrades of many network functions, avoiding the messy governance disputes that have plagued many cryptocurrencies.

B. Assumptions

Logos makes the following assumptions:

1) Fewer than ⅓ of (weighted) validators are Byzantine: Logos's consensus algorithm requires a total of $n \geq 3f + 1$ nodes in the presence of f faults. While there are algorithms that require only $n \geq 2f + 1$, they also require strict synchronicity and are comparatively more complex. We take a more practical approach that requires only partial synchronicity, and this Byzantine assumption is a tight bound for the partially synchronous context [9]. We additionally assume that Byzantine failures are independent but allow that faulty nodes can be coordinated by an attacker.

2) Partial Synchronicity: At least ⅔ of nodes' system times are practically in sync, meaning that the maximum difference is substantially less than some large time constant L_E, the length of each distinct validation period. Furthermore, we bound the maximal delay of messages between validating nodes by L_E. However, we allow asynchronous behavior of nodes within each validation period. These assumptions are slightly stronger than the standard PBFT synchronicity assumption.

3) Non-pathological connectivity of nodes: We assume the network is relatively well connected such that there is some path that exists between any two honest validators. This, together with partial synchronicity, ensures that messages between honest nodes will be delivered in bounded time.

4) Reasonably Bounded Attacker: We assume an attacker cannot delay honest nodes indefinitely. We also assume that an attacker is computationally bounded, such that standard cryptographic assumptions hold. Logos does not currently use quantum resistant cryptography but is designed to accommodate arbitrary cryptographic schemes if the need arises.

V. System Model

The Logos Network achieves consensus using *Axios*, a delegated Proof-of-Stake (PoS) algorithm with dynamic weighting based on Practical Byzantine Fault Tolerance (PBFT). We thus define the system model consistent with the original PBFT paper for ease of confirming its correctness [4].

The network is a replicated state machine comprised of a set of replicas called *nodes*, $\mathcal{N}$, that manage a distributed database of accounts, $\mathcal{A}$. These two sets are distinct, and a node can correspond to a single, multiple, or no accounts. Accounts have a variety of associated data, including the balance of the native value token, Ł. Using the partial synchronicity assumption, we divide real time into distinct, successive time periods called *epochs*, $\mathbb{E}$, each with fixed length L_E (on the order of 12 hours), that have a definitive, agreed-upon ordering by the nodes.

Nodes interact via messages m containing public-key signatures and message digests $H(m)$, where H is a collision-resistant hash function. We denote a message m signed by node i as $\langle m \rangle_{\sigma_i}$.

Changes to account data, such as sending Ł, are accomplished via authenticated messages called *requests* (transactions) sent by that account. Accounts appoint agents called *representatives*, $\mathcal{R} \subseteq \mathcal{A}$, that are trusted to run connected nodes and act on their behalf. Requests are received by, validated, and committed to the database by a set of *delegates*, $\mathcal{D} \subseteq \mathcal{R}$, that are elected by the representatives. This relationship is expressed as a one-to-many mapping $R : \mathcal{A} \mapsto \mathcal{R}$. Note that an account can act as its own agent by designating itself as its own representative. The voting power of each representative is determined by a function of its account balance and the total account balance it represents. Both representatives and delegates are paid for successfully processing transactions and are penalized for inactivity or malicious behavior. We designate each state of the system, including all account data and messages, as a *configuration*.

A key innovation introduced by Logos is the ability to process non-dependent transactions in parallel while maintaining the safety and liveness properties of PBFT. However, since transactions are committed at specific real times and require signed prior references, there is a definitive ordering of configurations. Nodes behave deterministically, and the network begins in an initial configuration shared by all nodes. Safety is guaranteed by ensuring that all non-faulty nodes

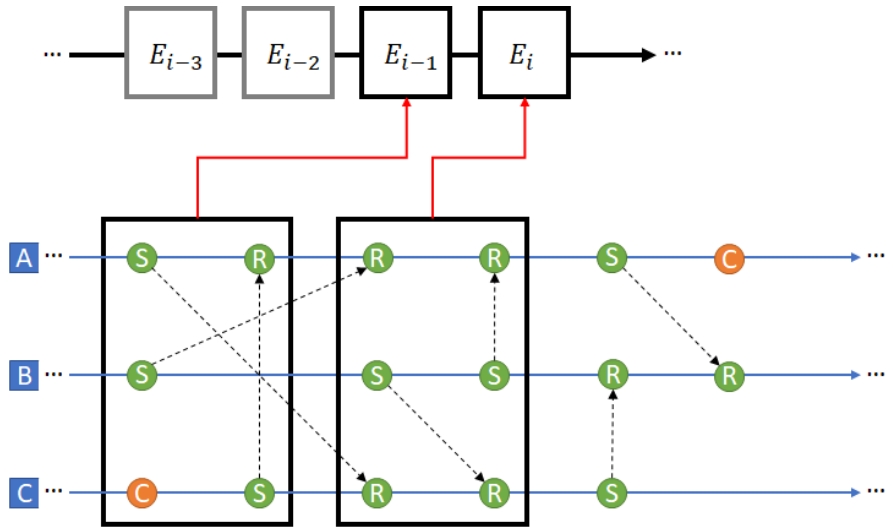

Fig. 2: The *archive*. A complete history of requests that were confirmed in each epoch is periodically memorialized into a main settlement chain. The current network governance settings can be found in the most recent block.

agree on a total order of execution of messages and their deterministic results despite failures.

These capabilities are enabled by novel data structure. Each account has a totally ordered set of committed requests that form a chain somewhat analogous to an individualized blockchain. Only an individual account is able to request changes to their account (except in the case of provable misbehavior). Since the system processes requests individually rather than in blocks, this means that, unlike traditional blockchains, chain forks and reorganizations can only occur when an account intentionally double spends. Via related requests, these blockchains form dependency relationships that, as a whole, reflect the overall system. The set of accounts plus the dependency graph constitute the *chain mesh* (Figure 1).

A main settlement chain called the *archive* consists of blocks for each epoch. These blocks contain a summary of all requests that occurred in that epoch, election results for delegates for a subsequent epoch, voting results for governance changes in future epochs, and other global information. The *archive* serves four primary purposes: (1) its blocks serve as checkpoints that force state synchronization across nodes; (2) it enables validator sets to change while the system runs a fixed validator consensus algorithm; (3) it facilitates decentralized governance; and (4) it provides a consolidated history for bootstrapping. This hybrid model ensures universal consensus while also allowing account requests to be processed individually and in parallel. The result is that Logos has the security of a traditional blockchain but can also scale transaction throughput based on validator hardware limits.

The addition of the *archive* allows for universal finality and a synchronization mechanism. Logos's Byzantine-fault-tolerant consensus protocol and incentive structure provide explicit security and game theoretic guarantees lacking in other DAG-based networks. The *chain mesh* structure is an improvement on RaiBlocks's base structure [10]. As a result, Logos is able to provide the scalability, throughput, and latency of DAG-based networks while providing the security of traditional blockchain networks.

We first detail each component of the system before turning toward the consensus algorithms. A breakdown of account and node types and their various functionalities and responsibilities are given in Figure 3 and Table I.

A. Accounts

Accounts $a_i \in \mathcal{A}$ are defined by the following tuple of values:

- *Address* (i): The public key component of a digital signature key-pair, used to identify the account. The private key component is kept secret and is used to sign requests and other messages approved by the account. Only one address may exist per account.
- *Balance* (b_i): the account's native token balance, Ł
- *Representative* (r_i): the account's designated voting representative, which can be the account itself
- *Account Chain* (c_i): the account's historical chain of transactions, beginning with an *open* request
- *Deposit* (d_i): the deposit required to open the account, with the required amount set based on the epoch

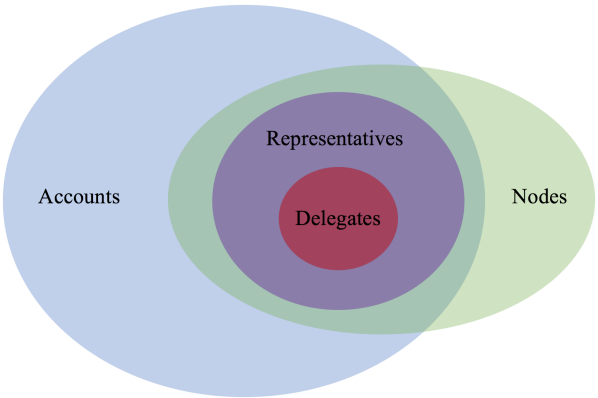

Fig. 3: Logos network participants

Network Participant Roles			
Accounts	Passive Accounts		Create requests • Open/close account • Send/receive tokens • Change representative
	Representatives	Non-Delegate Representatives	Maintain global state Elect delegates Validate delegate commitments Approve *archive* blocks Recall delegates Propose and vote on governance changes Enforce slashing conditions
		Delegates	Validate requests Propose *archive* blocks Recall delegates
Non-Account Nodes			Maintain global state Police representative and delegate actions Provide network access to passive accounts and light clients

TABLE I: Roles and functions of Logos network participants.

Each account can issue requests that modify the account state provided that they are cryptographically signed and valid. Each request is appended to the account chain in the order that the account chooses, subject to protocol verification and consensus. Because Logos does not use a UTXO structure like Bitcoin, each account chain can be extensively pruned due to the existence of the balance field and a periodic checkpoint mechanism (as discussed below).

The account representative, which votes on behalf of the account's balance, is determined by the mapping $r_i = R(a_i)$. An account can change its representative at any time via a request, but voting choices for a particular epoch are static based on the representative mapping from the end of the previous epoch.

A small deposit (specified by the network parameters set in the current epoch) is required to open an account. The deposit reflects the real cost of storing the account data across the network (on the order of $\$1$). Along with a modest proof-of-work requirement for account requests and variable transaction fees, this is one of the three primary deterrents of spam attacks. The deposit also incentivizes closing unused accounts, which enables further pruning.

B. Requests

In this section, we describe how different types of requests are created and how they interact with the Logos network. Accounts can issue the following requests:

- *Open Account*
- *Close Account*
- *Send Transaction*
- *Receive Transaction*
- *Change Representative*

The header of each request includes the *account* field, which contains the address (public key) of the account, and the *head* field, which contains the the hash of the previous request of the account chain.

1) Send Transaction: A *send* request is issued to send tokens to another account. The *destination* field contains the addresses of one or more recipients. The *balance* field contains the amount of tokens to be sent to each recipient. Once the request is confirmed by the network, the sum of the amounts are immediately and irreversibly deducted from the account's balance. Before the amount is accepted by the *destination* account with a *receive* request (as described below), the unsettled tokens in transit are tracked in a settlement pool (analagous to a mempool).

```
request:send {
    account: _
    head: _
    destination: _
    balance: _
}
```

A *send* request can send tokens to multiple destinations and also include a field for a user-defined transaction fee paid to validators (as described below). In the future, Pay to Script Hash (P2SH) functionality and data fields will enable rich, intelligent transactions. Note that a *send* request may not spend amounts stored in the account deposit.

2) Receive Transaction: To claim the tokens sent by a *send* request, the *destination* account must issue a *receive* request. Each *receive* request must present proof that the network has validated and committed the corresponding *send* request, which practically takes the form of validator signatures. The *receive* request is added to the *destination* account chain independently from the *send* request. As described previously, unsettled tokens from "unconfirmed" *send* requests will remain in the settlement pool.

```
request:receive {
    account: _
    head: _
    source: _
}
```

The Logos network derives several advantages from processing each *receive* request independently from its associated *send* request:

- Each account can control the ordering of its account chain, which prevents disputed incoming *send* requests from creating any blocking conditions. Otherwise, the *destination* account for the disputed *send* request would be unable to issue valid new requests since the previous request in the account chain (the *head*) is indeterminate.
- Allows for asynchronicity in the network when processing different unsettled *send* requests while simultaneously, ensuring that all nodes by definition agree with the account's settled balance.
- Enables aggressive pruning within the network, as lightweight nodes can keep track of only a running settled balance for accounts rather than the full account chain.

3) Open Account: To create an account on the Logos network, an *open* request must be issued. An *open* request is always the first (*genesis*) block of every account chain and is created when tokens are first sent to the account. The *source* field contains the hash of the incoming *send* request. The *representative* field contains the address of the account's designated voting representative (initially the account itself by default).

```
request:open {
    account: _
    head: null
    source: _
    representative: _
}
```

To issue an *open* request, the user must first generate a virtual account (a digital signature key-pair) that only exists locally on the user's node. The user then issues a *send* request to the virtual account from another account (for example, controlled by the user, a friend, or an exchange). An *open* request can be viewed as a variation of a *receive* request, as the request instantiates a new account chain for the account while simultaneously claiming the incoming unsettled tokens in the settlement pool. Note that the initial balance must be sufficient to cover the minimum account deposit required by the current epoch in order for the *open* request to be validated by the network. Moreover, the *transaction fee* field in this request can only be greater than 0 to the extent that the received amount is greater than the minimum deposit. If the amount in the initial *send* request is insufficient, then another *send* request that satisfies the requirements can be issued, and the initial *send* request can be claimed later via a normal *receive* request. The deposit is then locked for the lifetime of the account.

4) Close Account: A close request is a modified send request that simultaneously releases the deposit in the account. The request must send the entire account balance, including the deposit, and results in the deletion of the account from the network. The account may be reopened later, but it would initialize with a new account chain and metadata.

```
request:close {
    account: _
    head: _
    destination: _
}
```

5) Change Representative: A *change* request to update an account's designated voting representative may be issued at any time but will only go into effect in subsequent epochs. The delay ensures that the entire network has a complete, constant view of the set of representatives and their voting power at all times during each epoch. After the *change* request is committed to the account chain, the account's voting power is transferred from the old to the new representative.

```
request:change {
    account: _
    head: _
    representative: _
}
```

6) Anatomy of a Request: In addition to *account*, *head*, and the other fields described above, each request also requires:

- *A proof-of-work solution* to prevent spam. The PoW difficulty is specified in the governance conditions of the current epoch and will also vary by request type, with requests that require more network resources (*open* requests, *send* requests with a large number of outputs) having a higher difficulty.
- *A transaction fee* paid to the network for confirming the request, which rewards network participation and prevents spam.
- *A hash of a recent epoch block*, which ties the request to a specific history and acts as a defense against short- and medium-range attacks (an application of Transactions as Proof-of-Stake (TaPoS)).
- *A timestamp* (within an acceptable ϵ of network time), which helps define a global ordering of requests (along with the *head* field).
- *A signature* confirming that the request was sent by the account (the holder of the private-key).

C. Representatives

Representatives $r_i \in \mathcal{R}$ are responsible for electing delegates to validate requests, voting on proposed governance changes, and adding blocks to the *archive*. In order to qualify as a representative, an account must post a moderate deposit beyond the minimum for normal accounts and choose themselves as the representative. Representative deposits and balances are subject to slashing conditions if the representative fails to follow network rules (see below for slashing rules). Representative balances (including the deposit) are locked from when they enter the representative pool until $F+1$ epochs after they withdraw from the representative pool, where F is the number of epochs where economic finality is provided.

Each representative has a set of constituent accounts, $C_i = \{j|R(a_j) = r_i\}$. The voting power of the representative is determined by the function:

$$V(r_i) = b_{r_i} + \lambda \cdot \sum_{a_j \in C_i; j \neq i} b_{a_j} \quad (1)$$

where $\lambda \in [0, 1]$ is a scalar that penalizes represented balances since they are not directly staked (on the order of $1/4$). This disincentivizes representatives from holding most of their balance in other accounts that appoint a single account as representative to avoid slashing conditions.

By the Byzantine assumption, $|\mathcal{R}| > 3f_{\mathcal{R}} + 1$, where $f_{\mathcal{R}}$ is the number of Byzantine agents (weighted by voting power). In practice, we expect that accounts will move balances to honest representatives before the threshold is breached.

Representatives receive a portion of transaction fees proportional to their weights if they are connected to the network, while they are penalized if they are not online to vote. This incentivizes representatives to run reliable and honest nodes.

To ensure representative accountability, accounts that appoint poor representatives that either continuously fail to vote or violate network rules are subject to small penalties. Furthermore, since representatives receive fees proportional to their constituent balance, they may be willing to share some of those fees with accounts to attract additional balances. This structure incentivizes accounts to maintain quality representatives, prevents shielding funds from slashing, and encourages a large portion of network value to be actively securing the network while allowing most accounts to remain offline.

D. Delegates

Delegates $d_i \in \mathcal{D}$ validate and commit requests on each account chain and propose blocks to the *archive*. Delegates are elected for terms of a fixed number ℓ epochs, although any one delegate can serve any number of consecutive terms. Delegates are weighted by their number of votes.

If delegates are not properly fulfilling their duties, either by going offline or acting maliciously, they can be recalled by more than $2/3$ of either delegates or representatives. A successful recall closes the current epoch and starts the next one with its delegates. Multiple recalls may be necessary to flush out all bad delegates. All votes are subject to slashing conditions.

By the Byzantine assumption, $|\mathcal{D}| > 3f_{\mathcal{D}} + 1$, where $f_{\mathcal{D}}$ is the number of Byzantine delegates. Note that this is significantly weaker than the representative Byzantine assumption due to finite epoch length and recalls. Representatives directly validate delegate activity, so under most circumstances delegates can only threaten liveness but not safety.

Like normal representatives, delegate stakes are locked from when they are elected as delegates until $F + 1$ epochs from the last period the account acted as a delegate to prevent any history attacks before all nodes recognize finality.

VI. Axios Consensus Protocol

Consensus is achieved via *Axios*, a delegated version of the PBFT algorithm. Voting power is determined by a Proof-of-Stake class function, which prevents Sybil attacks and allows nodes to enter and leave the network. The consensus algorithm is inspired by modifications to PBFT proposed by ByzCoin [11] and OmniLedger [12], but it is tailored to Logos's unique structure.

A. Detailed Protocol

The Practical Byzantine Fault Tolerance algorithm, as detailed by Castro and Liskov [4], is a consensus mechanism for state machine replication in the presence of Byzantine faults. The algorithm provides both liveliness and safety, provided that there are $n \geq 3f + 1$ validation nodes (weighted by some weight function) with f faulty nodes. *Axios* is a PBFT-derived algorithm that allows for large, dynamic node sets to achieve consensus on transactions, votes, and epoch blocks. Each instance of PBFT requires a single leader called the *primary* to propose transactions and manage feedback from the other validators; primary selection for each network operation is described below.

PBFT typically requires $O(n^2)$ messages between nodes and $O(n)$ signature size, but using EC-Schnorr signatures reduces message complexity to $O(n)$ and signature complexity and verification to $O(1)$. *Axios* is based off of the EC-Schnorr PBFT algorithm described in [11] and [13]. This algorithm uses digital signatures rather than the MAC scheme used by standard PBFT. Nodes verify consensus via multisignatures. However, since EC-Schnorr multisignatures are constant size (rather than a concatenation of single signatures), and consensus only requires signatures from more than $\frac{2}{3}n$ nodes, it is necessary to keep track of which node has signed a message. This is accomplished via a bitmap B of length $|n|$ managed by the primary, where the i^{th} bit is set to 1 if validator i has signed the message and 0 otherwise.

Note that nodes are weighted by some weight function (such as represented balance or number of votes), but for simplicity we use notation for equally weighted validators.

Axios has three primary phases to reach consensus:

1) **Pre-Prepare**: the primary announces a network operation to be considered for consensus, signed by the primary.
2) **Prepare**: every node checks the validity of the operation and send a signed *prepare* message to the primary if it agrees the operation is safe and valid. The leader waits for prepare messages from more than $\frac{2}{3}n$ of nodes to construct a *post-prepare* message containing the EC-Schnorr multisignature of the agreeing nodes along with the corresponding bitmap identifying those nodes.
3) **Commit**: Upon receiving proof that more than $\frac{2}{3}n$ nodes signed prepare messages for the operation (via the post-prepare message), a node sends a signed *commit* message to the primary. The leader waits for commit messages from more than $\frac{2}{3}n$ of nodes to construct a *post-commit* message containing the EC-Schnorr multisignature of the committing nodes along with the corresponding bitmap identifying those nodes. The post-commit is then disseminated across then network as proof that consensus has been reached, and all nodes update their ledgers accordingly.

Figure 4 illustrates how messages are sent between a requesting account, delegates, and other nodes during normal operation.

Messages are relayed via a hub-and-spoke topology in which the primary sends and receives messages to all other delegates. While in a serial model, communication trees [13] or communication groups [12] result in improved bandwidth usage, in the parallel execution model this results in a higher latency without any reduction in message complexity, either per-node or overall. In the event of a primary failure after a pre-prepare, replica delegates fall back on all-to-all communication to ensures liveness. This relatively expensive fallback is mitigated since the other delegates will institute a cool-down period for the offending delegate, thus limiting the ability of a failing delegate to drag down network performance.

Logos allows for concurrent processing of network operations, in particular with account requests. In the event that dependent operations are simultaneously considered, nodes topologically sort requests and construct a dependency graph. They then process the requests in the implied order.

There are several ways of handling conflicting operations. The problem is greatly simplified since account requests referencing the same parent (e.g. double spends) are the only conflicting operations that can exist. Epoch blocks, which constitute the record of all account requests processed in that epoch plus results of votes, are deterministic since all nodes agree on both by construction. Similarly, bounty requests are not specific to an account since rewards are distributed *pro rata* with transaction fees, so they too are deterministic.

In the case of conflicting requests for a single account, the simplest solution is to reject all requests and ignore that account until the next epoch, when the conflicting requests are no longer valid. An exception is if a single request has prepare messages from more than $\frac{2}{3}$ of nodes, in which case the node commits that request and ignores others, and the consensus algorithm proceeds normally. Note that due to the account chain structure, conflicting requests from a single account can only occur intentionally, so temporarily freezing an offending account protects the network from related attack vectors. The ability for the network to punt on a fork decision without holding up normal functionality of the rest of the network is a major advantage of Logos compared to other crypto networks.

We provide proofs of the algorithm's safety and liveness guarantees in section VIII and direct readers to the original PBFT paper for full treatment of the base algorithm [4].

B. Total Ordering of Requests

A key security property of consensus is determining an ordering of requests. While some consensus schemes only achieve a partial ordering, *Axios* yields a total ordering. The primary delegate assigns a timestamp to each request that is affirmed by the other delegates. To ensure practical fairness, this timestamp must be within Δ of the account-assigned timestamp as well as the system clocks of each of the validating delegates. A total ordering is then defined by the timestamp, with conflicts resolved by request hash.

Note that this ordering methodology depends heavily on the *chain mesh*. By resolving most total ordering conflicts at the data structure level, it is simple to determine a total order by a FIFO consensus model.

C. Delegate Elections

Delegates are elected for each epoch $E_i \in \mathbb{E}$ from among the representatives in epoch E_{i-3} as follows:

1) **Candidacy** (E_{i-3}): To be eligible for election, representatives send messages declaring their candidacy to representatives distributed via a gossip protocol during epoch E_{i-3}.
2) **Voting** (E_{i-2}): Each representative votes by distributing its voting power among the candidates (Cumulative Voting) during epoch E_{i-2} and sends this vote to every other representative via a secure multicast-based broadcast algorithm, which ensures that all honest nodes see the same votes. For security, each vote message includes a hash of the previous epoch block, a timestamp, and a message signature. When a representative is found to have cast more votes than allowed by their voting power (i.e. the sum of votes cast can be no greater than $V(r_i)$), their stake is slashed. Note that although this process is not very efficient, it has minimal impact on overall network performance since elections happen infrequently.
3) **Certification** (E_{i-2}): The top K vote receivers are selected as delegates for E_i. Note, for the purposes of voting consensus, we cap the voting power of any delegate $1/6$ and renormalize. This restriction prevents edge cases where any one node can have greater than $1/3$ of the voting power and grief the network by delaying responses and avoiding slashing conditions. A merkle tree of voting results and a list of delegates are included in the block for epoch E_{i-2} on the *archive*, which certifies the election results. The delegate with largest voting power is the leader for the epoch and has the first chance to propose the block for that epoch, as described below.

Delegates serve terms of $\ell > 3$ epochs, balanced such that each election determines $1/\ell$ of the next epoch's delegates. This ensures that at least $1/3$ of delegates for each epoch are non-Byzantine and are aware of all requests that were being considered in the previous epoch, which prevents any potential double spending over different epochs. In the event of a recall, all requests are halted until the new delegates have been determined.

The voting process does not require an explicit quorum, instead relying on economic incentives to ensure an adequate portion of the network is involved either directly or indirectly. This allows the network to continue to function in the event of an extended partition. Provided non-pathological network connectivity, the most likely cause of such a partition is a large portion of representatives dropping offline within a short time period. In this scenario, there will be no fork of the network as the remaining nodes will maintain consensus.

If a representative fails to broadcast a delegate vote for several epochs, they incur a small penalty proportional to their constituent balance. This penalty incentivizes maximal voter participation, which helps provide security guarantees.

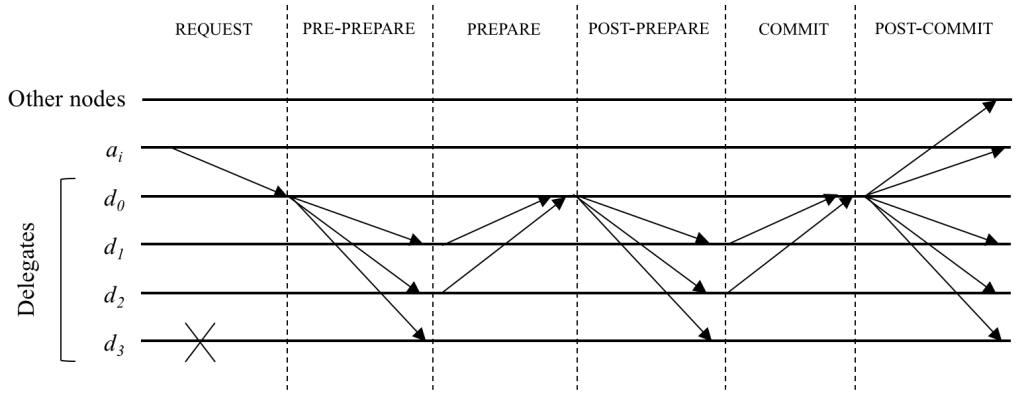

Fig. 4: Various message types sent during Logos's *Axios* consensus algorithm. Account a_i sends a request to delegate d_0, who acts as primary. The request is validated iteratively according to the algorithm, and consensus is achieved despite the failure of delegate d_3. After the request is committed, it is disseminated to all network nodes. For simplicity, this diagram assumes messages are sent via broadcast.

D. Governance Votes

Any representative can propose a governance change to an allowed parameter. The signed proposal is disseminated via the gossip protocol, and the proposal is voted upon in the same epoch by representatives according to the *Axios* consensus algorithm, with the proposer acting as leader. If successful, the governance changes are recorded in that epoch's block and are effected in a subsequent epoch, depending on the parameter changed. For example, a change to the number of delegates would need a lag of 3 epochs since delegates are chosen 2 epochs ahead of time.

Governance proposals require a high transaction fee as well as substantial proof of work compared to normal requests to prevent spam attacks, since governance votes require a large number of messages.

The following system variables can be modified by governance votes (non-comprehensive):

1) Number of delegates
2) Minimum deposit/bond for a new account
3) Minimum transaction fee
4) Minimum stake for delegate
5) Minimum stake for representative
6) Required proof-of-work for various requests
7) Distribution of transaction fees between various account types
8) Epoch time
9) Constituent balance penalty parameter
10) Pseudorandom function for load balancing

E. Request processing

Accounts send their requests to a single delegate who is designated as the primary for that request as it is processed, similar to the construct of the PBFT algorithm. If a leader proves faulty (Byzantine or offline), the account can resubmit the same request to another delegate after a short period of time L_r, where $L_p << L_r << L_E$ for expected processing time L_p and epoch time L_E. This allows an account to maintain liveness without allowing it to spam the delegates.

Note that this timeout function effectively replaces the view change mechanism of standard PBFT that is used by almost every alternative cryptocurrency. View changes for most BFT consensus schemes are very expensive and halt all ongoing requests. Logos's ability to parallelize consensus means that any view change halts only a fraction of ongoing requests, and each view change is inexpensive in that it only delays those requests rather than imposing a high overhead on the network.

In order to balance the request load amongst delegates, clients will select a delegate according to a delegate selection function, DS, that pseudorandomly selects a delegate j. That is, for request m sent by $a_i \in \mathcal{A}$

$$\mathsf{DS}(m) \equiv j \mod |\mathcal{D}| \tag{2}$$

For the purposes of implementation, we can use any pseudo-random collision resistant hash function voted on based on the governance protocol and deterministic ordering scheme (i.e. ordered by the cumulative representative weights) without needing to prove randomness. As a practical matter, this ensures that honest clients do not overload a single delegate with requests. While Byzantine accounts may collude and send all requests to a single delegate, safety and liveness is still ensured given ultimate usage of the standard PBFT three commit protocol. To identify accounts attempting to circumvent the load balancing, delegates can validate that messages from said account should be directed toward them. Each delegate will ultimately see most valid requests, but this additionally allows invalid requests to be efficiently filtered out by individual delegates before consuming resources of other delegates.

Requests received by a single primary delegate can be easily batched together for joint validation to reduce the overall number of messages without any other changes to request processing. Since each request is assigned to a specific primary, there is no duplication across batches of requests. Individual accounts can send multiple requests to a single delegate at the same time, provided that they are consistent, which can then be batched with requests from other delegates. Together with the ability to include multiple inputs in a receive

and multiple outputs in a send, this batching functionality eliminates the overhead associated with running consensus on individual transactions and for high volume accounts.

Every request includes a transaction fee. These fees are pooled together and distributed to representatives and delegates according to several factors:

- Constituent balance
- Bonus for delegates
- Bonus for the delegate that successfully proposes the epoch block

Rounding is used to ensure invariants such as total token supply are not broken. Balances confiscated for violations of slashing conditions are added to the transaction fee pool and distributed accordingly.

Since the cost of running the network is substantially less than proof-of-work systems and Logos is highly scalable, there will be little upward pressure on fees, and they should be *de minimis* for any size transaction. Nevertheless, they are critical for properly aligning incentives in the network:

1) Incentivizes nodes to actively validate requests by rewarding them in proportion to the actual cost of validation
2) Indirectly incentivizes accounts to include their balances in voting, ensuring that a large percentage of network value is securing the network
3) During periods of extreme volume (e.g. in the early stages of the network before large, professional validating nodes have been formed and node hardware can't keep up with transactions), it allows the network to maximize social utility by processing transactions from accounts that get the most utility from their transactions

Note that the network will operate at far below the capacity of even average consumer hardware for a long time due to its efficient design, so zero-fee transactions will be able to be processed. Additionally, the network will reserve a portion of the total token supply to initially subsidize transactions.

Transaction fees are claimed from a pool after an epoch block is approved by the network. Delegates who failed to validate a request and representatives that fail to vote during an epoch lose their claim to the transaction fees and penalty distributions. Unclaimed fees are added to the subsequent pool. Provided that epoch length is several orders of magnitude greater than transaction processing time (by construction), these reward requests will only represent a *de minimis* portion of overall network transactions.

F. Epoch Block Processing

Epoch blocks contain the following information:

- Merkle tree of account requests committed during that epoch
- Merkle tree of delegate election results for epoch $i + 2$ and list of delegates
- List of representatives for the epoch $i+2$ and their voting power
- Successful governance changes and a merkle tree of the vote
- Transaction fee pool

Epoch blocks are deterministic, and adding them to the blockchain is a *pro forma* process. All honest nodes have the same view of all block contents (after accounting for network latency), and they will reject any block that does not match that view exactly. If a node does not have a record for a request that is included in the block or disagrees with it, they can query the block proposer or other nodes for the transaction. This process is important for synchronizing the global state across nodes and uncovering slashing violations.

The two epoch delay for delegate elections and representative changes, as well as a minimum two epoch delay for governance changes, is necessary to ensure that all changes are known deterministically by all nodes well before the start of a new epoch.

Blocks are confirmed by the set of representatives via an *Axios*-class algorithm. The lead delegate for an epoch has the first chance to the propose the block, then delegate who received next amount of votes, and so on, until a delegate with a valid proposal has been found. Practically, this takes the form of a timeout for each proposal slot. To incentivize efficient block creation, an epoch block must be added to the *archive* before transaction fees can be claimed, and the successful proposer gets a bonus payment. Delegates who fail to propose a successful block forfeit any transaction fees.

A recall also ends with a block added to the *archive* as it closes an epoch. If delegates are unable to put together the correct block, representatives can force it through a recall.

G. Slashing Conditions

Logos uses slashing conditions similar to those proposed in [14]. Specifically, the following are slashable offenses:

1) *False Representation*: a node sends a prepare or commit message for a transaction that is invalid in some way, including referencing an uncommitted parent block, sending more than the current balance, sending frozen funds, or referencing the incorrect epoch.
2) *Double Prepare*: a node sends prepare messages for two conflicting requests in the same epoch
3) *Double Commit*: a node sends commit messages for two conflicting requests in the same epoch
4) *Conflicting Prepare/Commit*: a node sends a prepare message and a commit message for two conflicting requests in the same epoch
5) *Double Voting*: a representative casts more than one delegate or governance vote in a single epoch

Note that these are somewhat simpler than many slashing schemes, such as those in [14], while still maintaining desired properties such as accountable safety and plausible liveness. This is partially due to Logos's novel structure, where forks and reorgs can only occur intentionally. This greatly reduces the set of pathological scenarios where nodes must be slashed for the network to continue processing requests.

When balances are slashed, they go into the normal transaction fee pool and are distributed accordingly. This wide distribution is preferable to allowing a single account to claim the slashing reward, as it prevents a Byzantine account from claiming its slashed balance with an alternative account and

creates an economic incentive for all nodes with a balance to monitor for slashing conditions. An additional benefit is network efficiency: each node only requires one *receive* request for all transaction fees and slashing penalties that they are entitled to during an epoch. Thus, total request complexity is $O(n)$ for the pooled distribution, a substantial improvement over the $O(kn^2)$ complexity for direct distribution of k fees and penalties.

Functionally, slashing works as a special send to the transaction fee pool that is involuntarily added to account chain of the offending account. After the slashing condition has been discovered, any node can send this slashing request to a delegate for consideration, and delegates validate it like any other request. If the delegates are unable to commit the slashing request in the epoch which it occurred, any representative or delegate can initiate *Axios* consensus. Since the request can only take one form (that is, a send of a portion of the offender's account balance and deposit to the transaction fee pool), there are no race conditions between different nodes.

A slashing offense is highly unlikely to go unpunished if discovered by even a single honest node acting rationally, as that node has an economic incentive to publicly broadcast the offense. Since slashing offenses are only dangerous in so far as they impact honest nodes, we can be sure that these conditions offer a credible deterrent for any potential violators.

We note that slashing conditions are robust to more than $1/3$ of delegates or representatives acting Byzantine, even if valid slashes cannot be effected immediately. Staked funds remain frozen for a long period of time, which will give accounts sufficient time to redirect balances to honest representatives, who will then elect honest delegates. These honest delegates will still have time to slash the balances of the offenders before they thaw.

H. Polis State Sharding

Logos's state sharding scheme, *Polis*, is the final major component of Logos's scalability. Sharding involves breaking the network into disjoint groups that independently validate requests in parallel, thus allowing the network to scale in the number of validators. Each account is pseudo-randomly assigned to a shard, and shards process all requests from member accounts. Sharding is a well studied concept in distributed systems, but it is relatively new to crypto networks.

Logos's structure is uniquely conducive to and optimized for sharding. By maintaining a request chain for each account, the network state can easily be partitioned into groups. Other design choices, such as separating sends and receives, minimize the need for inter-shard synchronicity and communication.

Sharding increases network vulnerability to some attack vectors, so special care must be taken to ensure that guarantees are maintained with high probability. Given a sufficiently large number of representatives relative to the number of shards (achieved via a cap on representative balances) as well as a source of randomness that is impractical to game, it can be shown that subgroups of representatives violate honest node assumptions with very low probability.

Specifically, let N be the total number of equally-weighted validator nodes,[1] of which K are Byzantine, that will be split among $\mathcal{S}$ shards, each of equal size $n = \frac{N}{\mathcal{S}}$. Let X_j be a random variable for the number of Byzantine validators assigned to shard j, each of which take on values k_j such that $\sum_j k_j = K$. Then the X_j's have hypergeometric distributions with the following joint pdf:[2]

$$P(X_1 = k_1, \dots X_{\mathcal{S}} = k_{\mathcal{S}}) = \frac{1}{\binom{N}{K}} \cdot \prod_{j=1}^{\mathcal{S}} \binom{n}{k_j} \tag{3}$$

Intuitively, this can be thought of as the lining up $N - K$ non-Byzantine nodes and placing K Byzantine nodes among them. The number of possible resulting orders is equal to $\binom{N}{K}$ by the Stars and Bars combinatorial result. Then the first n nodes in the resulting line up are assigned to the first shard, the second n to the second shard, and so on. Since order does not matter within each shard, we have to divide by the number of ways to arrange k_j Byzantine nodes and $n - k_j$ non-Byzantine nodes in shard j, which is equal to $\binom{n}{k_j}$ by the same result. The probability is then the inverse of the total number of distinct combinations. It is easy to verify that this is a proper pdf that sums to 1 by the generalized version of Vandermonde's identity.

Consequently, the probability that the fraction of Byzantine nodes in any one shard is more than α is:

$$P(X_i \leq \alpha, \forall i \in \mathcal{S}) = \frac{1}{\binom{N}{K}} \overbrace{\sum_{k_1=0}^{\alpha} \dots \sum_{k_{\mathcal{S}}=0}^{\alpha}}^{\sum_i^{\mathcal{S}} k_i = K} \prod_{i=0}^{\mathcal{S}} \binom{n}{k_i} \tag{4}$$

where presumably we are testing for $\alpha \leq \lfloor \frac{N}{3S} \rfloor$.

We can sample from this distribution to estimate the probability of at least one shard violating the Byzantine validator assumption by having $n/3$ or more Byzantine validators for various levels of N and $\mathcal{S}$. These simulations show that if $N \geq 1000$, then for $\mathcal{S} \leq 10$ the network has a very low probability of a Byzantine violation occurring in a shard.

At mature state, we expect Logos to have at least $1000 - 5000$ representative nodes, which could easily accommodate 10 shards. The exact shard capacity will depend on the real distribution of tokens among accounts, with capacity maximized under a uniform distribution. Security measures such as capping the voting power of any one representative in a shard would partially mitigate the risk from large balance accounts and encourage large stakeholders to allocate to several accounts, which could improve inter-shard communication.

Logos is also careful to prevent adverse balance redistributions between shards that could be used to violate these assumptions. For example, representative balances are locked before assignment to a shard. Additionally, inter-shard communication helps police behavior and provide finality. Shards periodically share internal states globally and collectively

[1] The logic for weighted nodes is analogous but more complex. We assume equal weights in the white paper for simplicity of presentation.

[2] The distribution is *not* binomial, as is often assumed elsewhere, since we are sampling without replacement. While the binomial approximation is reasonable for the first shard, it is inaccurate for a joint distribution for all shards.

cement the overall state in the main *archive* to ensure system-wide consensus. If a node detects adverse behavior, such as a violation of a slashing condition, then proof can be broadcast to other shards to adjudicate and penalize the offender.

Representatives r_i (identified by their account) are assigned to shards by a bias-resistant public random function based on RandHound [15], that has complexity $O(c^2 \log n)$ given a group size c. To achieve this, we require a one-time setup phase that securely shards nodes into subgroups and then leverages aggregation and communication trees to generate subsequent random outputs. The random output $\hat{r}$ is unbiasable and can be verified together with corresponding challenge $\hat{c}$ as a collective Schnorr signature against the collective key. Effectively, through utilizing NIZK and PVVS, we can generate public verifiable and unbiased randomness on demand. This function is abstracted as $\mathsf{S}_{\mathcal{R}}$ that produces an output hash h_i:

$$\mathsf{S}_{\mathcal{R}}(\hat{r}) \rightarrow h_i$$

These hash outputs are then ordered, and the first $n_{\mathcal{S}} = \frac{|\mathcal{R}|}{\mathcal{S}}$ representatives are assigned to the first shard, the next $n_{\mathcal{S}}$ are assigned to the second shard, and so on.

Sharding assignments occur every M epochs. After sharding occurs, the representatives assigned to each shard elect shard delegates to validate requests assigned to that shard and propose epoch shard blocks in the same manner as in the unsharded system. There are additionally system-wide delegates that receive all request committment proofs and epoch shard blocks to ensure proper functioning. These system delegates are in charge of constructing *archive* blocks that serve as checkpoints for the entire network. M is chosen to balance the mixing frequency of shards with system overhead.

Non-representative account a_i is assigned to shard j by a deterministic function $\mathsf{S}_{\mathcal{A}}$ modulo the number of shards.

$$\mathsf{S}_{\mathcal{A}}(a_i) \equiv j \mod |\mathcal{S}| \tag{5}$$

All requests originating from an account are processed by its shard's delegates. By design, each account can function completely independently of other accounts, which makes this division simple. Since unsettled incoming balances are secure, accounts under most circumstances will be able to wait for the end of an epoch block when all shards sync to that checkpoint to issue receives for sends from outside the shard. If the funds are required immediately, the account can issue a cross-shard receive. Transaction fees will be distinct for intrashard and intershard requests, which will allow the network to appropriately process transactions in an order that maximizes social utility.

The ability for shards to efficiently synch on summary data (in the form of epoch blocks) greatly reduces the overhead imposed by cross-shard transactions. This is enabled by Logos's unique hybrid data structure and is a key improvement over alternative sharding schemes.

Our models estimate that sharding can increase transaction throughput by at least an order of magnitude to more than 100,000 transactions/second, enabling the network to exceed the capacity of centralized payment networks such as Visa. It also can reduce average confirmation time to less than 1 second and decrease transaction fees, which will further open the network to applications requiring rapid and cheap transactions.

Sharding remains an area of active research, and we anticipate subjecting further implementation details to a thorough peer-review process to ensure logical consistency.

VII. Examples of Network Functionalities

A. Life Cycle of a Request

1) Create Request Packet: A request begins when an account, a_i, prepares a request packet m that includes the request type and other request specific data (such as the destination and balance for a send request) as described in subsection V-B.

2) Delegate Selection: a_i then runs a delegate selection function DS on the request packet to determine which current delegate will be the primary D' for this request:

$$D' = \mathsf{DS}(m)$$

DS is an arbitrary function chosen by the client that should select a delegate at random from the active set of delegates, with the purpose of load balancing requests among delegates. a_i then signs the request packet to produce the message:

$$\sigma(m) = \langle \langle \text{REQUEST}, H(m), a_i, D' \rangle_{\sigma_{a_i}}, m \rangle$$

where $H(m)$ is the hash of the send packet m.

a_i then sends $\sigma(m)$ to D'. The *Create Request Packet* and *Delegate Selection* steps correspond to the *Request* step of the *Axios* consensus algorithm.

3) Pre-Prepare: Upon receiving $\sigma(m)$, D' will first confirm that the request is valid, and drops the message if it is invalid. Otherwise, D' broadcasts[3] a signed pre-prepare message to the other non-primary delegates (backups),

$$\langle \langle \text{PRE-PREPARE}, H(m), D' \rangle_{\sigma_{D'}}, \sigma(m), m \rangle$$

The pre-prepare message indicates that the request should be considered for commitment.

4) Prepare and Post-Prepare: Upon receiving a valid pre-prepare message, each backup $b \in \mathcal{D} \setminus D'$ confirms that it has not yet accepted a conflicting pre-prepare message for the same slot in account chain c_i (two requests have the same *head* field or the same block number). If a backup accepts the pre-prepare message, it sends a prepare message back to the primary,

$$\langle \text{PREPARE}, H(m), D' \rangle_{\sigma_b}$$

The primary records all incoming prepare messages until it has accumulated prepares from $\frac{2}{3}|\mathcal{D}| + 1$ delegates (weighted, including itself). Upon passing the prepare threshold, the primary constructs post-prepare proof,

$$\langle \langle \text{PREPARE}, H(m), D' \rangle_{\Sigma_B}, B \rangle$$

where B is a bitmap of delegates that sent prepare messages and Σ_B is the corresponding EC-Schnorr multisignature, and broadcasts it to the backups.

[3] In reality, delegates use communication trees rather than broadcasts to exchange messages for bandwidth efficiency, but we used broadcasts in this example for simplicity.

5) Commit and Post-Commit: Upon receiving a valid post-prepare message for the request, each backup b sends a commit message to the primary,

$$\langle \text{COMMIT}, H(m), D' \rangle_{\sigma_b}$$

Once again, the primary waits until it has received commits from $\frac{2}{3}|\mathcal{D}| + 1$ delegates (weighted, including itself). Upon passing the commit threshold, the primary constructs post-commit proof,

$$\langle \langle \text{COMMIT}, H(m), D' \rangle_{\Sigma_B}, B \rangle$$

with bitmap of committers B. This post-commit message is then broadcasted to delegates and then distributed along with the original request s to all nodes via a gossip protocol. Upon receiving a valid post-commit message, each node accepts the request as committed and updates the corresponding account chain locally.

B. Epoch Blocks

The block for epoch E_i is constructed by its delegates $d_{i,j} \in \mathcal{D}_i$, $j = \{0, \ldots, |\mathcal{D}_i|\}$, where delegates are ordered by their vote count. After the end time t_{E_i} of an epoch has passed, additional requests will no longer be considered for that epoch. An small allowance is made for differences in the clocks of various nodes. At this time, delegates can propose epoch blocks, which include voting results and a summary of all requests that were committed in that epoch. Since all of these elements are deterministic, the process confirming the validity of a proposed block is objective and unambiguous. All nodes can pre-compute the block so that they confirm proposal legitimacy by comparing hashes.

The first delegate, $d_{i,1}$ is given the first slot of length SlotTO to propose an epoch block. This entails constructing the block B and sending it to the set of representatives via a gossip protocol in the form a pre-prepare message $\langle \langle \text{PRE-PREPARE}, H(\mathsf{B}), d_{i,j} \rangle_{\sigma_{d_{i,j}}}, \mathsf{B} \rangle$. Upon receiving the pre-prepare message, representatives confirm that its hash matches their pre-computed hash. If a representative r_k recognizes the block as valid, it returns a prepare message to the proposer, $\langle \text{PREPARE}, H(\mathsf{B}), d_{i,j} \rangle_{\sigma_{r_k}}$. If it does not recognize the block as valid, it identifies requests in the block that do not match its local ledger state. It then queries other nodes for proof of that request. This ends with r_k either receiving proof and updating its ledger, finding a slashing condition violation, or rejecting the block.

As with requests, the proposer waits until it has received prepares from $\frac{2}{3}|\mathcal{R}| + 1$ by weight and sends proof to the network. The cycle repeats with commit messages until the threshold is passed, at which point proof of block commitment is sent to the network. At that point, all nodes update their local *archive* accordingly and query the network for any requests they missed due to latency or connection interruptions. This effectively forces the network to sync at periodic checkpoints.

If $d_{i,j}$ has failed to propose a valid block within the allotted slot, then the next delegate $d_{i,j+1}$ can propose a block. This continues until the correct block has been proposed. As a fallback, if all delegate proposal slots have timed out, any node can propose a block by sending a general pre-prepare message of the form $\langle \langle \text{PRE-PREPARE}, H(\mathsf{B}) \rangle_{\sigma_{\mathsf{node}}}, \mathsf{B} \rangle$.

After the epoch block has been appended to the chain, the transaction fees for that epoch (with a bonus given to the successful proposer) are unlocked as pending receive transactions for each representative and delegate account. This provides and economic incentive for nodes to promptly agree on a block.

C. Slashing

If a malicious node η has violated a slashing condition in epoch E_i, nodes can construct evidence, proof, that the condition was violated (e.g. commits for two different requests for the same account chain slot). Then at any time any network node may submit a slashing request to a current delegate D' with the form $\langle \text{SLASH}, \mathsf{proof}, \eta, D' \rangle$. D' then checks that it is indeed a slashing violation, and initiates a special send request that moves the penalty balance from η to the transaction fee pool. This SLASH request is then validated and committed by the current delegates in the same manner as normal requests.

If E_i has passed and η has not yet been slashed, then at any time any representative may propose $\langle \text{SLASH}, \mathsf{proof}, \eta \rangle$ to the set of current representatives. The proposal is then committed in the same manner as epoch blocks. This condition encourages delegates to efficiently process most slashing requests while also ensuring that slashing can still occur without delegate cooperation.

VIII. Proofs of Desired Properties

This section contains sketches of proofs of security, liveness, and protocol fairness. As sharding remains an area of active research, we present these proofs in the unsharded context.

A. Safety and Liveness

For this section, we assume an account $a \in \mathcal{A}$ sends a request m with account chain sequence number n to a primary $D' \in \mathcal{D}_v$ in epoch E_v. Requests are processed according to the consensus protocol described in section VI, which involves three phases: pre-prepare, prepare, and commit. As previously, we assume there are the maximum f faulty delegates out of $|\mathcal{D}_v| = 3f + 1$ total, with an equivalent assumption for representatives.

1) Safety: For safety to hold, we require that every non-faulty delegate and representative agree on the sequence numbers of requests for each account chain. An important consideration is dependent requests between account chains. Logos's structure has two important features that simplify this analysis. First, it separates requests involving different account chains, in particular sends and receives, into different requests. This means that each account can definitively order its own account chain without ambiguity. Second, it requires that dependent requests between chains be processed sequentially rather than in parallel by construction. For example, *receive* requests are rejected as invalid unless the corresponding *send* has already completed the commit phase. This means that if the safety property holds for a single account chain, then it

necessarily holds for all account chains. Thus, we can reduce the problem from a parallel setting to a sequential setting where the delegates only consider a single request at a time, which is equivalent to the problem presented in [16].

In the *pre-prepare* phase, D' sends a pre-prepare message to the backup delegates to propose m for commitment. We define the predicate $\mathsf{preprepared}(m, E_v, n, a, i)$ to be true if delegate i accepts m as a valid request sent in a valid pre-prepare message from the primary. This requires that the signatures are correct, the current epoch is E_v, and delegate i has not accepted a pre-prepare message for E_v and account a with sequence number n containing $m' \neq m$.

In the *prepare* phase, delegates in the set $\{i|\mathsf{preprepared}(m, E_v, n, a, i) = \text{TRUE}\}$ send signed prepare messages back to the primary, which waits for $2f + 1$ prepares (including itself) before broadcasting multisignature proof to the backups. The backups then verify the validity of the prepare proof. We define the predicate $\mathsf{prepared}(m, E_v, n, a, i)$ to be true if and only if delegate i has $\mathsf{preprepared}(m, E_v, n, a, i) = \text{TRUE}$ and has verified multisignature proof that $2f + 1$ delegates (including itself) have sent prepare messages that match the pre-prepare message.

Theorem 1. *Let H be a collision-resistant hash function. If* $\mathsf{prepared}(m, E_v, n, a, i) = \text{TRUE}$ *for some non-faulty delegate i, then* $\mathsf{prepared}(m', E_v, n, a, j) = \text{FALSE}$ *for any non-faulty delegate j and any m' such that $H(m') \neq H(m)$.*

Proof. Assume there exist two non-faulty delegates i and j such that both $\mathsf{prepared}(m, E_v, n, a, i) = \text{TRUE}$ and $\mathsf{prepared}(m', E_v, n, a, j) = \text{TRUE}$, where $H(m') \neq H(m)$. Since H is collision resistant, $m' \neq m$. Since $\mathsf{prepared}(m, E_v, n, a, i) = \text{TRUE}$, i has seen proof that $2f+1$ delegates sent a prepare for m, which means that at least $f+1$ non-faulty delegates sent a prepare for m. Similarly, since $\mathsf{prepared}(m', E_v, n, a, j) = \text{TRUE}$, j has seen proof that $2f+1$ delegates sent a prepare for m', which means that at least $f + 1$ non-faulty delegates sent a prepare for m'. Since there are $2f+1$ non-faulty delegates in total, that means at least one non-faulty delegate has sent prepare messages for both m and m'. However, this is a contradiction since a non-faulty delegate will not send out conflicting prepares by construction. □

In the *commit* phase, delegates in the set $\{i|\mathsf{prepared}(m, E_v, n, a, i) = \text{TRUE}\}$ send signed commit messages back to the primary, which waits for $2f + 1$ commits (including itself) before broadcasting multisignature proof to the backups. The backups then verify the validity of the commit proof. We define the predicate $\mathsf{committedlocal}(m, E_v, n, a, i)$ to be true if and only if delegate i has $\mathsf{prepared}(m, E_v, n, a, i) = \text{TRUE}$ and has verified multisignature proof that $2f + 1$ delegates (including itself) have sent commit messages that match the prepare message. We also define $\mathsf{committed}(m, E_v, n, a)$ to be true if and only if $\mathsf{prepared}(m, E_v, n, a, i) = \text{TRUE}$ for all i in some set of $f + 1$ non-faulty delegates.

Theorem 2. *If* $\mathsf{committedlocal}(m, E_v, n, a, i) = \text{TRUE}$ *for some non-faulty delegate i, then* $\mathsf{committedlocal}(m', E_v, n, a, j) = \text{FALSE}$ *for any non-faulty delegate j and any m' such that $H(m') \neq H(m)$.*

Proof. This proof is analogous to the proof of Theorem 1. □

Theorem 3. *If* $\mathsf{committedlocal}(m, E_v, n, a, i) = \text{TRUE}$ *for some non-faulty delegate i, then* $\mathsf{committed}(m, E_v, n, a) = \text{TRUE}$.

Proof. Since $\mathsf{committedlocal}(m, E_v, n, a, i) = \text{TRUE}$ and i is non-faulty, then i has received proof that $2f+1$ delegates have sent commit messages. Since at most f delegates are faulty, then $\mathsf{prepared}(m, E_v, n, a, j) = \text{TRUE}$ for all j in some set of at least $f + 1$ non-faulty delegates. □

Once $\mathsf{committedlocal}(m, E_v, n, a, i) = \text{TRUE}$, delegate i updates its local state to reflect the request and disseminates the commit to other network nodes. All non-faulty non-delegate nodes confirm validity of the commit and update their own local states, thus reflecting the states of the delegates.

These results mean that non-faulty nodes agree on the sequence numbers in the account chain for requests that commit locally. Once a request is committed locally on any non-faulty delegate, then $\mathsf{committed}(m, E_v, n, a) = \text{TRUE}$. A prerequisite for committing locally is having proof that $2f+1$ delegates sent commit messages, so at least $f + 1$ non-faulty delegates have committed to the request in slot n. Since these nodes will never accept another request in slot n, they will always outnumber faulty nodes that may do so. This is the desired safety property.

2) Liveness: Liveness is provided by the timeout mechanism for a request. If the requesting account does not receive a reply for its request within timeout of length T, it can then resubmit the request to a different primary delegate on a new timeout. To accommodate fluctuations in network latency, the timeout increases each time, e.g. kT for the k^{th} submission. This timeout is enforced by the delegates, who will refuse to consider repeated requests with different primaries until the timeout is breached. This is possible because the maximum difference between honest delegates' clocks is bounded by $\tau < T$ by assumption.

By the weak synchronicity and connectivity assumptions, the maximal delay of messages between validating nodes is bounded, and by the Byzantine assumption, the account will eventually select a non-faulty delegate as primary. Even if the Byzantine assumption for delegates breaks, the recall and delegate election mechanisms will eventually replace faulty delegates assuming the Byzantine assumption for representatives holds. Thus, liveness is guaranteed under the model assumptions.

3) Consensus properties: Given safety and liveness guarantees, it is easy to see that the four desired consensus properties hold for the value of a request in a given account chain slot. *Agreement* means every non-faulty node must agree on the request. If a non-faulty node has locally committed the request, that means committed is true for that request. Consequently, no other non-faulty node has locally committed another request. By the weak synchronicity assumption, any node that not yet locally committed the request will eventually receive the proof required to locally commit that request.

Validity requires that if all nodes propose the same request r, then all non-faulty nodes locally commit r. While in practice only a single node proposes a value, this property is tautologically true as it means all non-faulty delegates would send a prepare message for r and thus locally commit r, as per the safety proof.

For *integrity* to hold, every non-faulty node locally commits at most one request, and if it locally commits some request r, then r must have been proposed by some node. The first condition is true by Theorem 3. The second condition is true by construction, as a pre-prepare message must be sent for any delegate to prepare or commit.

Termination means that every non-faulty node decides some request. This can be guaranteed to be true by allowing a null value if an epoch concludes without a proposed request being committed. This request can be reconsidered in the next epoch. Once a request has been locally committed by a single node, the gossip protocol that sends the commitment proof will eventually reach all other nodes with probability 1 by the connectivity assumption, at which point they will also locally commit that request.

B. Nash Equilibrium and Fairness

By construction, Logos's protocol is a Nash equilibrium assuming that any individual representative's weighted deposit makes up less than $1/3$ of the total. Intuitively, because the protocol contains slashing conditions for malicious actions and incentives for honestly voting, representatives and accounts maximize utility (defined as maximizing account balance) by operating honestly within the confines of the network. Hence, the demonstration of fairness can be reduced to showing that all bad actions cause slashing and all good actions result in nominal payment. We briefly formalize this concept below.

Theorem 4. *If no representative has more than $1/3$ of the total voting power in any epoch, preparing and committing blocks in accordance with the protocol is a Nash equilibrium.*

Proof. Let us define S_i as the action set for representative r_i, $P_i(S_0, ..., S_n)$ as the uniform payoff utility function, TXF as the transaction fee, and SC as the slashing cost. The solution to the system $s^* \in S$ is considered to be a Nash equilibrium when

$$\forall_i, s_i \in S_i : P_i({s_i}^*, {s_{-i}}^*) \geq P_i(s_i, {s_{-i}}^*)$$

Here, for the sake of simplicity, we have assumed that any colluding group of representatives can be combined as one representative without loss of generality. As noted in Table I, representative responsibilities are limited to a set of tasks. Each task can ultimately boiled down to a three commit protocol where the payoff possibilities are either a) good: 0, bad: $-\mathsf{SC}$, b) good: $\lambda\mathsf{TXF}$, bad: 0, or c) good: $\lambda\mathsf{TXF}$, bad: $-\mathsf{SC}$. Note, we make the assumption that any malicious behavior results in the slashing behavior being reported given the clear economic incentive for reporting. Hence, assuming that greater than $2/3$ of nodes are good actors, we can see by construction that any action s_i chosen by r_i deemed to be good and within the honest intent of the protocol is always strictly greater than the bad action. If r_i is less than $1/3$, any bad action will ultimately flagged and the representative's balance will slashed by consensus. □

Additional discussion of various validator payoffs can be found in section XIII.

IX. Network Extensions

Logos has an ambitious roadmap for future developments, including smart contract capabilities and broad ecosystem expansion.

A. Bounded Smart Contracts

Logos will have native scripting and data that enable richer and smarter requests. Note that these functionalities will be purposefully limited so as not to hinder the network's primary goal of optimal value transfer. The scripting language will be sufficiently rich to enable conditional transfers, side channels, cross-chain support, basic privacy, and other accretive functionality, but will also be more constrained than Turing-complete smart contract systems to prevent the chain from becoming bogged down in complex computation.

This scripting language will enable to development of smart contracts on top of Logos. The smart contract layer will exist entirely separately from the base settlement layer, which segregates functionality to preserve modularity and flexibility. By limiting the capabilities of smart contracts, we strike a balance between preserving network throughput while enabling the most compelling use cases. Smart contracts will execute on a virtual machine, which will have compilers for many common, modern programming languages. WebAssembly is a promising option that would allow for compatibility with existing platforms and frameworks together with high performance. We place a particular emphasis on code security, testing, and formal verification, and will develop tools to empower users to easily deploy robust contracts.

B. Logos Ecosystem

Substantial work will be devoted to developing the Logos ecosystem. We will support the development of a full range of clients, from full nodes run by validators or exchanges to light nodes run by IoT devices. Clients will be available for desktop, mobile, and web platforms, and a heavy emphasis will be placed on global accessibility and usability.

The core team will develop turn key solutions for many of Logos's use cases, including payment services, point-of-sale systems, peer-to-peer payment apps, e-commerce plugins, and invoicing apps. At the same time, we want to empower the community as a whole to produce other applications, such as crowdfunding apps, accounting software, IoT integration, and payroll apps, while also producing competing solutions to core uses. As such, accessible APIs will be available in a variety of languages, enabling easy integration into web frameworks and development stacks.

User-friendly features will also be aggressively pursued. For example, an address name service will allow human-readable account addresses similar to emails. This will facilitate easy raw transactions independent of any client. Other features will

further abstract the base protocol for the average user, reducing costly errors and improving accessibility.

These improvements will be variously developed by the core team as well as the broad community. In order to incentivize community engagement, the core team will fund and promote a wide range of external projects. A primary goal in these endeavors is to promote ecosystem diversity, which reduces single point-of-failure risk and gives users flexible tool kit to attack unique problems.

C. Archive Microblocks

Intermediate epoch microblocks could be used to simplify the process of constructing a large epoch block. These microblocks would contain summaries of the transactions that occurred in a portion of the epoch, similar to the basic epoch blocks. These microblocks would periodically be issued by the delegate assigned to produce the full epoch block and validated by the other delegates via standard consensus. The full epoch block could then contain a reference to the microblock chain (as well as voting results) rather than the transaction summary data.

Microblocks would confer several benefits. Nodes would be able to periodically ensure they are fully synched at a much higher frequency than if they solely relied on the full epoch blocks. Also, failures in the priority queue for proposing the full epoch block would be identified and resolved much sooner, reducing overhead on the network in the event of a failure.

D. Key Permissioning

In order to facilitate strong security practices, Logos could include a variety of key pairs for each account that have limited functionality. For instance, a voting-only key pair would be permissioned for voting and validation but not transfers of funds. This would allow users to keep the keys that are permissioned for transferring funds offline in secure storage while also having an online node interacting cryptographically with the network.

E. Separate Receive Chain

Rather than using a single chain structure for all account activities, accounts could have a separate chain solely for receives. As the account is sent tokens, the corresponding receives are added automatically to the receive chain. Unlike debits, the order of credits do not matter, so there is no need for accounts to have a validated request to receive incoming funds. The order of the receive chain can be determined dynamically by the timestamp assigned by consensus (with hash value as a tiebreaker). Given an assumed max clock difference of Δ and local clock of time t, then all receives before time $t - 2\Delta$ have a definitive total order that can be memorialized in the database. Regardless if the *total order* of a particular receive is finalized, an account can safely use any finalized send that has a timestamp in the past for any outgoing send, which ensures liveness.

Separating receives from other requests thus allows the network to implicitly achieve consensus on receives, which can reduce network overhead by a factor of 2 in an unsharded state. While sharding will likely require explicit receives, separate receive chains simplify network logic around unsettled funds and improve data structure efficiency.

F. BLS Signatures

BLS signatures are an alternative to EC-Schnorr signatures that retain many of the same desirable complexity properties while simplifying signature aggregation [17]. EC-Schnorr multisignatures signatures require an upfront commitment of signatories to produce a challenge unique to that set. This means that the signatories need to be known *a priori*, which typically adds a round trip per multisignature, as in [18]. Any change in the signatory set requires a new challenge to be issued and thus a new round trip. While this challenge can piggy-back on the post-prepare stage of *Axios*, it results in a larger post-prepare message size since unaggregated validator prepare signatures are sent ahead of the commit. It also prohibits the use of a consensus accelerator that could reduce the total round trips from 2.5 to 1.5 (except in an optimistic setting that assumes the delegates that confirmed the last request batch will confirm the next one).

On the other hand, BLS signatures offer the same fixed-sized aggregate signature while also allowing a dynamic signatory set. The signers of the multisignature do not need to be known *a priori*, and individual signers can be deduced from the resulting multisignature. This allows efficient aggregation at each stage of consensus and the use of an accelerator. BLS aggregated signatures are an area of active research, and it remains to be seen if lower empirical performance can be improved upon.

G. Consensus Accelerator

Logos could speed up consensus in an optimistic setting by implementing an accelerator similar to Thunderella [19] or chain.com on top of *Axios* consensus. While their claims that these accelerators provide "instantaneous" confirmations of transactions are inaccurate, they can reduce the number of round trips required for consensus from 2.5 to 1.5. This improved performance does not come "free" as it relies on substantially stronger assumptions, most significantly requiring at least $3/4$ of validators to be honest.

A possible accelerator in the context of Logos works as follows:

1) Consensus is initiated as usual with an account sending a request to a primary delegate, who validates the request and sends out a pre-prepare.
2) Replica delegates for the request validate it and send back a prepare message to the primary.
3) If the primary receives prepares from at least $3/4$ of delegates, then the transaction is finalized and the primary propagates proof.
4) Otherwise, proceed with the post-prepare and commit stages of *Axios*.

Note that in the event of failure, consensus reverts back to normal PBFT-derived *Axios*. Note that unlike in the architecture proposed in Thunderella, this is a smooth transition rather

than an expensive fallback. This results in high performance even if the rather stringent assumptions of the accelerator are violated.

H. Fast Archive Traversal

When bootstrapping to the present state of the network, nodes must traverse the entire archive chain from genesis to the present to confirm that the various governing sets (representatives and delegates) and their associated signatures are correct. Once a node knows the delegates for a particular epoch, payment verification simply involves validating a multisignature, but learning of the delegates can be onerous. More efficient delegate discovery could facilitate simple payment verification (SPV) in the context of sidechains.

To facilitate fast traversal of the archive chain, Logos can employ a multiple-links approach, similar to those proposed in the proof-of-stake context [20]. Rather than each block merely linking to the previous block, each block could include links to major "milestone" blocks, such as the previous block that has an ID ending with k 0's for $k = 1, \ldots, 5$. This link table can then be used to prove connections of blocks to historical blocks and rapidly move throughout the archive chain.

Alternatively, Logos could adopt the stamping certificate approach for fast bootstrapping proposed in [21].

I. Provisional request processing

Logos could support optional *trust-but-verify* execution of requests, as described in [12]. Under this scheme, each shard contains several groups of optimistic validators, but only one core set of delegates. The optimistic validator groups would provide provisional validation that has a high chance of accuracy (but is not guaranteed). All requests would still need to go through the core validator set for final validation, but the optimistic validators can provide a rapid indicative response since they only see a subset of transactions and do not require many validators per group. This trust-but-verify model would be most appropriate for small transactions where the benefits of speed outweigh the small potential for fraud. Recipients of larger transactions would still likely wait for the core validator consensus before accepting a send as finalized.

A version of this scheme could also be used to filter out spam and garbage transactions by requiring mandatory provisional validation. While a corrupted group of provisional validators would still let spam transactions through, the majority of provisional validator groups in expectation would reject these requests and not submit them for final validation, thus preserving the resources of the core validators.

The PBFT paper proposes an alternative provisional commitment scheme (tentative execution), but this likely involves too much message overhead to be practical in an open network.

J. Robust relay network

To reduce latency and packet loss, and thus overall network capacity, Logos nodes could be run off a relay network. Multiple relay networks have been either proposed or implemented for Bitcoin and other cryptocurrencies, such as FIBRE and bloXroute [22]. These networks have the potential to substantially improve propagation of transactions and batches/blocks by removing bottlenecks present in standard Internet networks. These relay networks are thus complementary to the improvements Logos makes in consensus, data structures, and sharding.

X. Cryptography

A. Hashing Functions

Blake2b is used for producing digests of messages, requests, and blocks. It was chosen over alternatives such as Keccak due offering better software performance while also providing unbroken security [23].

However, proof-of-work uses a composition of scrypt and Balloon, which are both bandwidth hard, to provide multiple levels of ASIC-resistance to preserve the intended anti-spam property [24].

The key derivation function uses Argon2id, which won the public PHC competition for the best password hashing algorithm [25].

B. Signature Scheme

Logos uses an elliptic curve Schnorr signature algorithm (EC-Schnorr) with the ED25519 curve. This scheme has several advantages relative to other common schemes such as ECDSA, including native multisignature of constant size, non-malleability, speed, and resistance to several side channel attacks [26].

We modify classical EC-Schnorr to the PBFT context, which requires $\frac{2}{3}n + 1$ signatures rather than n for prepare and commit messages. This modification introduces a bitmap to keep track of which validators out of the entire set are included in a specific multisignature.

EC-Schnorr reduces the PBFT message complexity from $O(n^2)$ to $O(n)$ and the signature complexity from $O(n)$ to $O(1)$. Since consensus is required for each account request, these improvements are necessary to achieve high scalability that maintains safety properties.

XI. Network implementation

A. Network Parameters

Logos will initially use the following parameters (subject to change in final implementation):

- *Token Supply*: 100 billion Logos, divisible to 10 decimals (resulting in 10^{24} indivisible units)
- *Epoch Length*: 12 hours
- *Number of Representatives*: 1,000 to 5,000
- *Number of Delegates*: 50
- *Maximum Request Size*: 2048 bytes
- *Deposit Size*: on the order of $1
- *Minimum Transaction Fee*: on the order of $0.00001
- *Proof of Work Difficulty*: 10K hashes
- *Constituent Penalty* (λ): $1/4$

Besides the token supply, all parameters are modifiable via the governance mechanism to allow the network to adapt. The initial token supply will be minted in the genesis block and distributed to initial holders. Subsequent network inflation, if any, and related economics are areas of active research.

B. Performance Analysis

To validate the network structure, we estimate network throughput and confirmation latency using the above network parameters. We assume that delegates communicate through a binary tree and have an average one-directional latency of 100 ms.

The primary determinants of network throughput are validation time (that is, hardware) and node bandwidth. Table II shows throughput in transactions/second under ranges of both parameters for the unsharded network model. Under conservative expectations of 0.1 ms validation time and 100 Mbps, bandwidth is significantly more binding than hardware, limiting the network to about 2,500 tps. In reality, we expect delegates and, to a lesser extent, representatives to specialize as service providers with both above average hardware and network connectivity. This would allow the network to achieve capacity of 15,000 – 20,000 tps on the initial implementation. Sharding will improve this capacity by an order of magnitude to over 100,000 tps.

Average confirmation time under this model is 2.92 seconds, which is dominated by inter-delegate latency. Intelligent sharding can realistically reduce latency by a factor of 2 or more, resulting in an expected average confirmation time of around 1 second.

On AWS, processing a single transaction has a marginal cost of less than $\$0.0008$, which is dominated by IO costs. A dedicated server will reduce processing costs by an order of magnitude, and price should decline as hardware and bandwidth improve. However, even assuming all delegates run on AWS or similarly priced services, validators could be consistently profitable with just a $\$0.001$ transaction fee while running a top line instance with 64+ cores, 200+ GB of RAM, SSD storage, and 10+ Gbps network speeds.

Storage requirements are more difficult to estimate since they are heavily dependent on the number of accounts. Logos is structured to encourage maximal pruning, which allows most nodes to only store balance and metadata associated with each active account. This amounts to less than 2KB per account, or less than 200 GB for 100 million accounts. In reality, the number of active accounts will be orders of magnitude less than 100 million for several years at least, but the network will be accessible to full nodes running commodity hardware even for an unexpectedly high number of accounts. Other data, such as current epoch requests and votes, will be considerably less in size, and can be discarded after they are finalized. Sharding will enable even further data pruning.

XII. Structure Commentary

A. Network Scalability

Fundamentally, the network achieves greater scalability than other crypto systems primarily through three design choices: (1) separating leader selection, validator selection, and request validation, (2) decoupling independent transactions, and (3) state sharding.

Traditional blockchains merge the processes of leader selection, validator selection, and request validation. In Bitcoin, the leader and validator are one-in-the-same, and are determined by competitive proof-of-work mining, which also verifies transactions. Independent mining necessitates a block structure that bundles together transactions, as network latency as well as forking issues makes processing individual transactions infeasible. Proof-of-stake consensus algorithms typically achieve speed and energy efficiency gains by decoupling validator selection and request validation. By moving from a competitive, uncoordinated system to a coordinated system, blocks can be deterministically assigned to a validator according to simple rules rather than wasteful, arbitrary computation, and these blocks can be committed far faster. Validator selection is achieved via network rules that qualify a node as a validator, such as a minimum stake. However, this open, indirectly determined validator set results in anti-scaling with the number of validators, and the network must sacrifice some efficiency gains by allowing for a continuously dynamic set. Delegated consensus schemes directly determine locally fixed validator sets of relatively small size that can rapidly process transactions while also allowing for changes to the set over time and preserving safety conditions. Provided proper economic incentives, delegate nodes can be expected to become specialized for validation with sophisticated hardware. Any coordinated consensus system that requires x out of n validators to commit a transaction will naturally be limited by the x slowest honest validator, so encouraging specialization further improves capacity. Logos's *Axios* consensus algorithm falls into this final category and leverages decades of established research on practical distributed systems.

While Logos is one of several networks using such a consensus algorithm, its hybrid base structure is unique and critical to its scalability. Despite the many advances in blockchain technology since the introduction of Bitcoin, almost all major crypto networks use Bitcoin's same basic block structure that consolidates arbitrary unrelated requests into single units. Logos instead uses a *chain mesh* structure that decouples independent transactions, which confers several advantages. Most importantly, unrelated transactions can be processed in parallel rather than in batches. This ensures that any issues that arise with conflicting transactions do not impact any independent transactions. Contrastingly, when blockchains identify a conflicting transaction, all unrelated transactions in the affected block as well as the block's descendants can potentially be reversed in a fork.

This first benefit has several additional implications. Forks can only be produced purposefully, and their impact is localized, meaning they can be resolved definitively and simply without wasting validator time and energy. Together with the choice of consensus algorithm, this means that clients do not need to wait for extended periods of time to be sure transactions are committed. Consistency between nodes on the global state is also ensured since there is only one unambiguous request that exist in each slot of any chain, whereas blockchain network nodes can have significantly different views of the global state depending on if they have seen the (correct) latest block. Blockchains must then place limits on both block time and block size, as pushing either past the limit increases both node inconsistency and fork probability, both of which

		Validation time (ms)					
		1	0.5	0.25	0.1	0.05	0.01
Node IO (Mbps)	10	254	254	254	254	254	254
	50	833	1,272	1,272	1,272	1,272	1,272
	100	833	1,667	2,544	2,544	2,544	2,544
	500	833	1,667	3,333	8,333	12,720	12,720
	1000	833	1,667	3,333	8,333	16,667	25,441
	10000	833	1,667	3,333	8,333	16,667	83,333

TABLE II: Network throughput (tps) under various node bandwidth and verification time assumptions. Blue indicates network is hardware bound, while red indicates network is bandwidth bound.

lower overall network security and efficiency. Conversely, Logos nodes can process unrelated transactions as quickly as possible given hardware and bandwidth constraints. While individual validation of each request increases the total number of network messages, the gains from the ability to process in parallel and the obviation of block constraints (and the accompanying dangers that come along with violating them) more than make up for the additional overhead. Finally, the large number of transactions (as opposed to chunky blocks) reduces reward variance to zero and removes the need for complicated leader elections that are practically challenging to implement.

Other DAG-based crypto networks share many of these advantages, but mostly fail to provide adequate security guarantees, both practically and theoretically. Logos contains a global settlement *archive* that achieves universal consensus and finality with the same security as traditional blockchains. This hybrid structure uniquely gives Logos both practical speed and safety.

While the hybrid chain mesh-blockchain structure enables one dimension of parallelism, *Polis* adds a second dimension. As discussed in subsection VI-H, sharding will enable disjoint groups of accounts to independently process transactions. Logos's structure greatly simplifies sharding by natively assigning requests to a single account rather than a shared chain. While this remains an active area of research, we expect to incorporate sharding early in the Logos life cycle.

Overall, these design choices reflect the philosophy that Logos should focus on being the best practical transaction network. Care is taken to optimize all levels of the design, from client implementation to choice of network features. An emphasis is placed on incorporating accretive functionality such as smart contracts while ensuring they are not detrimental to the core transaction layer. This balance is continuously scrutinized and reviewed to see if it supports Logos's goal of unbounded scalability.

B. Clients and Pruning

By construction, the chain enables heavy pruning. To interact with the network, a node at a minimum needs the head of each account's chain, account summary data (including balance), and the last few blocks on the *archive*. Furthermore, system economics encourage users to close unused accounts to free up their deposits, which allows nodes to discard that account after all of its requests have been finalized.

This means that the network can support very light autonomous nodes. This opens up the system to a wide range of users that cannot support full nodes that are continuously online. Such light nodes can be used in peer-to-peer mobile apps, IoT devices, and point-of-sale applications.

For security, the network foundation and other major stakeholders will maintain full histories of the network plus periodic, publicly available snapshots. This will enable light nodes to securely bootstrap up to the present.

C. Economic Analysis

The Logos system aims to fully align economic incentives of all participants to ensure desired properties, particularly reliability, scalability, and accessibility. The network is paid for by users in proportion to the amount of system resources they consume. This contrasts with systems like Bitcoin where the entire network subsidizes transactions. At the same time, Logos is heavily optimized to minimize the overall cost to use the network. Data storage for bootstrapping is similarly incentivized: each node has an incentive to store its own transaction history and commitment metadata and can pass this along to any chain that lacks it when needed. This data can be validated against the *archive* to ensure its integrity.

The economic incentive structure is critical in protecting the network. Deposits and transaction fees deter spam and Sybil attacks. Proof-of-stake based validation in conjunction with slashing conditions police validator behavior. This is particularly an advantage over proof-of-work networks, where an attacker's loss is bounded at the cost of mining equipment. Tying validation to holding a direct stake in the network ensures that validators' incentives are aligned with users, with is of particular importance for gracefully handling governance issues. Slashing conditions are enforced by all nodes, which benefit economically from successfully penalizing misbehaving nodes.

By individually validating each transaction rather than blocks, Logos can also directly distribute transaction fees to validators pro rata to their stakes. As a consequence, there is no competition between validator nodes to support a particular block or fork. All stakeholders benefit from each validated transaction, so throughput is the top priority, as desired. If delegates are providing subpar service, then representatives are incentivized to vote them out of their role. Similarly, if representatives are electing poor delegates, accounts are incentivized to shift their voting power to better representatives.

XIII. Attack Vectors and Mitigation

A. Double / Triple / n-tuple Spends

An n-tuple spend attack occurs when n conflicting transactions are sent for validation in the hopes of diverting

funds from a legitimate recipient. Provided that the model assumptions hold, this type of attack is impossible in Logos. If one transaction has already been committed, then conflicting transactions are ignored. The *Axios* consensus algorithm in conjunction with slashing conditions ensure that no double spends can occur after a transaction is committed. If none of the transactions have been approved, then validators follow deterministic rules to choose one transaction to commit. Since conflicting transactions can only occur intentionally in the chain mesh structure, violating accounts can be frozen without affecting the liveness of the rest of the system.

B. Control Attacks

A control attack occurs if the Byzantine node assumptions are violated and ⅓ or more of nodes are controlled by an attacker. Control attacks can compromise both safety and liveness. If the assumption is violated by delegates, only liveness can be compromised, as the super majority of other nodes, including representatives, will reject any malicious commitments. These nodes will then vote for other delegates, which can restore proper network functioning. If the assumption is violated at both the delegate and representative level, then the network can be halted indefinitely. Recalls are the primary method of restoring liveness in the event of Byzantine assumption is violated.

Safety has stronger defenses due to slashing conditions. These slashing conditions ensure that transactions that were committed according to *Axios* can only be reversed if more than ⅓ of validator stakes are slashed. To enforce these conditions, validator stakes are locked for many epochs after they stop validating, which helps ensure with high probability that all honest nodes see the same commitments. This gives committed transactions economic finality by guaranteeing a substantial cost for any attack.

Logos also minimizes the impact of a successful control attack. While such an attack would result in the potential reversal of all transactions that occurred after the modified transaction in traditional blockchains, the chain mesh structure ensures that only dependent transactions are impacted.

C. History Attacks

A related attack vector is history attacks, which include both short-range and long-range revisions to the network history. Provided that Byzantine node assumptions hold, these attacks are not possible in Logos without severe economic loss to the attackers. After an epoch is finalized (plus reasonable time for dissemination), all nodes have the same view of the global state. Any revisions to this global state by old delegates will be rejected without issue. Short term revisions are subject to slashing conditions.

These attacks are most problematic for nodes that are synchronizing to the current network state. If they are syncing from malicious previous delegates that have moved their balances to other accounts (and thus cannot be slashed), the old delegates can manipulate the past commitment votes and cause syncing issues (a.k.a bootstrap poisoning). This type of attack can be avoided if our weak subjectivity model holds, which requires that new nodes can obtain a recent, correct snapshot of the global state. This will be facilitated by periodic publishing of snapshots by major stake holders, which collectively can be cross-checked and accepted with a high confidence. Even if this model fails, however, online nodes are unaffected and safety and liveness are preserved.

D. Nothing at Stake

Another related issue is the nothing at stake problems, whereby validators are incentivized to validate conflicting forks to maximize their share of transaction fees. Slashing conditions that asymmetrically penalize such behavior plus a network structure that eliminates most forks means that this is not an issue with Logos.

E. Selfish Validator Attacks

This class of attacks includes selfish miners and stake grinding. In both cases, validators attempt to increase their potential rewards at the expense of other validators. These attacks do not impact safety or liveness, but are undesirable in a fair system. The selfish mining attack, which impacts proof-of-work blockchains such as Bitcoin, allows miners with 25% or less hashing power to claim more than their fair share of rewards by selectively withholding discovered blocks. Similarly, stake grinding involves validators in proof-of-stake systems that award block slots to specific validators based on random selection manipulating blocks or computations to bias the random process in their favor.

Logos is not susceptible to common selfish validator attacks. Since Logos does not use a block paradigm, each account can choose a primary delegate to lead validation of a transaction. As a result, there is no randomness in the validation process that can be manipulated.

Logos is, however, susceptible to a fee denial or consensus-group exclusion attack, whereby a malicious delegate can purposefully exclude other delegates from the consensus group, thereby denying them their share of the fees. We expect that other nodes will penalize this behavior by voting offending delegates out of their roles.

F. Eclipse or Network Partition Attacks

An eclipse attack occurs when an attacker controls all connections in and out of a victim's node. This allows them to manipulate and censor that node's view of the global state. This could theoretically result in the victim node accepting double spends, but this is much less of an issue than in traditional blockchains. For such an attack to work in Logos, the attacker would have to eclipse the victim for a several epoch periods during which the attacker spoofs delegate votes. In contrast, an eclipse attacker on a traditional blockchain only needs to produce a double spend block, which is substantially easier.

A larger threat is an extended network partition. There are a number of defenses that preserve safety in the event of a partition, in particular the *Axios* consensus algorithm and the *archive* that is subject to the approval of the representative set.

This means that a full network partition will generally result in a loss of liveness, but some pathological network topologies could lead to two incompatible histories.

G. Spam Attacks

A spam attack involves sending a large number of garbage requests or other messages with the intent of denying capacity to legitimate transactions and increasing the cost of validation. Logos includes several anti-spam measures. Each request has a cost in the form of proof-of-work and transaction fees. The proof-of-work is specifically designed to be ASIC-resistant, and both parameters can be adjusted by the network during periods of high demand. Account deposits also prevent spam in two ways. First, they prevent a Sybil-type attack where an attacker sends spam from multiple accounts, allowing validators to easily throttle requests from high volume accounts. Second, it prevents potential storage bloat from retaining the metadata of many accounts.

The related denial-of-service (DoS) attack involves flooding a node with superfluous messages to prevent it from interacting with the rest of the network. A successful DoS attack on a sufficient number of delegates can prevent the network from validating requests, thus compromising liveness. However, assuming that a sufficient number of representatives remain available, which is reasonable in a sufficiently decentralized network, they can use the recall mechanism to substitute in delegates not under attack. Common DoS preventative measures, such as IP whitelisting or filtering, can also reduce the impact of such attacks. Note that DoS resistance is a major benefit of decentralized networks over centralized, single-point-of-failure services.

H. Quantum Attacks

Common cryptographic signature algorithms are theoretically vulnerable to a certain quantum computer attacks. However, quantum computing is still very far from practical use, and Logos will allow changes to the cryptographic scheme when the threat is more relevant.

XIV. Conclusions

Logos represents the next generation of decentralized transaction networks. Logos achieves unbounded scalability by abandoning the old blockchain paradigm and building on an optimized architecture. The *chain mesh* structure unbundles transactions from blocks, enabling parallel validation and heavy pruning to accommodate light nodes, while the *archive* chain ensures synchronization across the network. Logos's *Axios* consensus algorithm is supported by decades of secure distributed systems research and allows for rapid and efficient validation. The deployment of *Polis* state sharding and bounded smart contracts will allow for scaling in the number of validators and rich transactions. Together with other design choices, these innovations help Logos reach an estimated 2,500 tps initially and over 100,000 within several years.

Logos has an explicit mandate to be the premier global transaction network. This singular focus makes it better suited for value transfer applications than either centralized solutions or other crypto systems, most of which aim to be general purpose networks with arbitrary capabilities or are derived from such networks. By drastically reducing transaction fees, minimizing frictions, and increasing security, Logos can disrupt a wide range of existing payment-based services, unlocking tremendous consumer value and social utility. Already, Logos is developing a robust ecosystem that seamlessly integrates point-of-sale systems, peer-to-peer payments, IoT capabilities, and numerous other applications into the Logos settlement layer. Logos's foundational philosophy will continue to guide its development, with the ultimate goal of integrating the world into an internationally accessible and open value network.

Acknowledgements

We would like to thank Andrew Wang, Kevin Zhang, Barr Yaron, Christophe Van, Joseph Rafidi, Lorin Gu, Keri Findley, Dennis Lu, Hikari Senju, and David Smalling for their feedback and discussion.

References

[1] P. Bowen, A. Goel, M. Schallehn, and M. Schertler, "Choosing the right platform for the industrial iot." [Online]. Available: http://www.bain.com/publications/articles/choosing-the-right-platform-for-the-industrial-iot.aspx

[2] R. O'Connell, C. Alexander, R. Strachan, B. Alway, S. Nambiath, J. Wiebe, S. Li, and D. Aranda, "Gfms gold survey 2017," March 2017. [Online]. Available: http://thomsonreuters.ru/wp-content/uploads/2017/04/GFMS-Gold-Survey-2017.pdf

[3] IMF, "Appendix i. international reserves," 2017.

[4] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[5] S. Gilbert and N. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 2, pp. 30–36, 2012.

[6] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[7] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[8] V. Buterin, "Proof of stake faq," https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ, (Accessed on 12/28/2018).

[9] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren, "Efficient synchronous byzantine consensus," *arXiv preprint arXiv:1704.02397*, 2017.

[10] C. LeMahieu, "Raiblocks: A feeless distributed cryptocurrency network," 2017. [Online]. Available: https://raiblocks.net/media/RaiBlocks_Whitepaper__English.pdf

[11] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 2016, pp. 279–296.

[12] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," 2017.

[13] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities" honest or bust" with decentralized witness cosigning," in *Security and Privacy (SP), 2016 IEEE Symposium on*. Ieee, 2016, pp. 526–545.

[14] V. Buterin, "Minimal slashing conditions," https://medium.com/@VitalikButerin/minimal-slashing-conditions-20f0b500fc6c, (Accessed on 12/28/2018).

[15] E. S. et al., "Scalable bias-resistant distributed randomness," *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[16] M. Castro, B. Liskov *et al.*, "A correctness proof for a practical byzantine-fault-tolerant replication algorithm," Technical Memo MIT/LCS/TM-590, MIT Laboratory for Computer Science, Tech. Rep., 1999.

[17] D. Boneh, M. Drijvers, and G. Neven, "Bls multi-signatures with public-key aggregation," https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html, March 2018, (Accessed on 3/24/2018).
[18] T. Z. Team, "The zilliqa technical whitepaper." [Online]. Available: https://docs.zilliqa.com/whitepaper.pdf
[19] R. Pass and E. Shi, "Thunderella: blockchains with optimistic instant confirmation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 3–33.
[20] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," 2017.
[21] D. Leung, A. Suhl, Y. Gilad, and N. Zeldovich, "Vault: Fast bootstrapping for cryptocurrencies," Cryptology ePrint Archive, Report 2018/269, 2018, https://eprint. iacr. org/2018/269, Tech. Rep.
[22] U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sirer, "bloxroute: A scalable trustless blockchain distribution network whitepaper."
[23] VAMPIRE and E. II, "ebacs: Ecrypt benchmarking of cryptographic systems," http://bench.cr.yp.to/results-sha3.html, (Accessed on 12/28/2018).
[24] L. Ren and S. Devadas, "Bandwidth hard functions for asic resistance." *IACR Cryptology ePrint Archive*, vol. 2017, p. 225, 2017.
[25] PHC, "Password hashing competition," https://password-hashing.net/, (Accessed on 12/28/2018).
[26] Ed25519, "Ed25519: high-speed high-security signatures," http://ed25519.cr.yp.to/, (Accessed on 12/28/2018).
[27] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, Jun. 2002. [Online]. Available: http://doi.acm.org/10.1145/564585.564601
[28] A. Hesterberg, A. Lincoln, and J. Lynch, "Improved connectivity condition for byzantine fault tolerance," *arXiv preprint arXiv:1503.02774*, 2015.
[29] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[30] J.-P. Aumasson, S. Neves, Z. Wilcox-O?Hearn, and C. Winnerlein, "Blake2: simpler, smaller, fast as md5," in *International Conference on Applied Cryptography and Network Security*. Springer, 2013, pp. 119–135.
[31] R. Pass and E. Shi, "Fruitchains: A fair blockchain," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2017, pp. 315–324.
[32] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.
[33] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE, 1999, pp. 120–130.
[34] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

APPENDIX A
LEGAL NOTICE

The information contained in this paper is confidential and may constitute trade secrets and information that is otherwise protected by applicable law. This paper is only for informational purposes, and may not be copied or disclosed to anyone without the express written permission of Promethean Labs LLC ("Promethean").

The information included in this paper is based on information reasonably available to Promethean as of the date hereof, and does not purport to be complete. Promethean does not undertake any duty to update the information set forth in this paper. Furthermore, the information included in this paper has been obtained from sources that Promethean believes to be reliable. However, these sources cannot be guaranteed as to their accuracy or completeness. No representation, warranty or undertaking, express or implied, is given as to the accuracy or completeness of the information contained herein by Promethean, its members, managers, employees representatives or affiliates, and no liability is accepted by such persons for the accuracy or completeness of any such information.

Projected figures set forth in this paper are hypothetical in nature and are shown for illustrative, informational purposes only. This material is not intended to forecast or predict future events, but rather to demonstrate the anticipated business activities of Promethean.

This paper contains certain "forward-looking statements," which may be identified by the use of such words as "believe," "expect," "anticipate," "should," "planned," "estimated," "potential," "outlook," "forecast," "plan" and other similar terms. Examples of forward-looking statements include, without limitation, estimates with respect to financial condition, results of operations, and success or lack of success of Promethean projects. All are subject to various factors, including, without limitation, general and local economic conditions, changing levels of competition within certain industries and markets, changes in legislation or regulation, and other economic, competitive, governmental, regulatory and technological factors, any or all of which could cause actual results to differ materially from projected results.

This paper does not constitute an offer to sell or the solicitation of an offer to purchase any securities of Promethean or any of its affiliates. Any such offer or solicitation may be made only by means of the delivery of a confidential private placement memorandum and other definitive legal documentation, which will contain material information not included herein, and which will supersede, amend and supplement this paper in its entirety.

Direct and indirect investments in distributed ledger technologies, cryptocurrencies and other digital assets involve substantial risks due in part to the highly speculative nature of such investments, risks relating to the regulatory regimes governing such technologies and other assets, and uncertainty relating to technology. Promethean cannot anticipate every possible current or future regulation or technological development that may affect Promethean's businesses and operations. Future developments may have a significant impact on Promethean's businesses and operations causing it to lose some or all of its working capital. The information set forth in this paper does not constitute legal, tax, investment or other advice, or a recommendation to purchase or sell any particular security.