Helix: A Scalable and Fair Consensus Algorithm

Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz
Ori Rottenstreich, Ronen Tamari and David Yakira

Orbs Research (orbs.com)

V.1.1

***Abstract*—We present *Helix*, a Byzantine fault tolerant and scalable consensus algorithm for fair ordering of transactions among nodes in a distributed network. In Helix, one among the network nodes proposes a potential set of successive transactions (block). The known PBFT protocol is then run within a bounded-size committee in order to achieve agreement and commit the block to the blockchain indefinitely. In Helix, transactions are encrypted via a threshold encryption scheme in order to hide information from the ordering nodes. The encryption is further used to realize a verifiable source of randomness, which in turn is used to elect the committees in an unpredictable way, as well as to introduce a correlated sampling scheme of transactions included in a proposed block. The correlated sampling scheme restricts nodes from prioritizing their own transactions over those of others. Nodes are elected to participate in committees in proportion to their relative reputation. Reputation, attributed to each node, serves as a measure of obedience to the protocol's instructions. Committees are thus chosen in a way that is beneficial to the protocol.**

I. Introduction

As the blockchain space matures, different variants of the technology emerge, each with its different strengths and weaknesses. The Nakamoto consensus [29] was first to realize a permissionless membership model, mitigating Sybil attacks and reducing communication complexity by integrating the ingenious Proof-of-Work [16] (PoW) mechanism. While the incentive structure of PoW-based blockchains proved successful in bringing many competitors to the mining game, thereby increasing the security and resilience of the system, it created a paradigm that is extremely slow and expensive. Its exceptionally open nature coincides with its ability to facilitate a large number of miners, but restricts its transaction throughput. Transaction confirmation is slow and finality is asymptotically approached but never really obtained. Moreover, miners' interests are only partially aligned with those of the users—they first and foremost wish to solve PoW puzzles rather than execute and validate smart contracts [26].

Variants of the original PoW-based blockchain protocol try to improve certain aspects of it, while maintaining PoW as a key factor. Byzcoin [23] offers finality in a PoW-based blockchain by introducing a PBFT [11] layer among a committee of recent block finders. Technologies based on block-DAGs are good examples for recently suggested solutions to some scalability and speed issues with PoW. These include GHOST [35], which offers an alternative to the longest chain rule whose resilience does not deteriorate when the block rate is high; and SPECTRE [34], which gives up the concept of a chain of blocks altogether and suggests a virtual voting algorithm to determine the order of blocks in a block-DAG.

Parallel to Nakamoto consensus variants, we see proliferation of additional protocols and blockchains, inspired by ideas from the domain of distributed systems. These try to mitigate some of the fundamental problems with PoW, and develop cheap and fast consensus without the excess work associated with the latter. An interesting protocol in this regard is Algorand [27], which cleverly introduces a cheap way to incorporate common randomness and use it to select a block leader, emulating the native randomness incorporated in PoW. Another protocol that alternates block leaders is Tendermint [9], which introduces a procedure for continuous primary rotation over a PBFT [11] variant. Other attempts to design consensus protocols for a business-oriented environment focus on assuring a degree of fairness among the participating nodes by being resilient to transaction censorship. A good example is HoneyBadgerBFT [28], where initially encrypted transactions are included in blocks, and only after their order is finalized, the transactions are revealed.

Helix borrows from these ideas to achieve a fast, scalable, and fair consensus protocol that is highly suited to the business requirements of modern applications. Helix relies on PBFT for Byzantine fault tolerance. In fact, Helix can be interpreted as a blockchain construction on top of PBFT, based on randomly-elected committees, which enables it to run many individual instances of PBFT sequentially, each responsible for a single value (block). It inherits PBFT's finality, eliminating the possibility of natural forks in the blockchain. And it restricts the agreement process to a bounded-size committee, selected from the larger set of all consensus nodes, in order to mitigate PBFT's inherent difficulty to scale to large networks. The committee size is determined according to some conservative upper bound on the number of faulty nodes, f. In PBFT, whenever the number of participating nodes n satisfies $n > 3f + 1$, performance is sacrificed for redundant security. In Helix, performance is optimized even when n is large, by setting the elected committee's size to be $m = 3f+1$, possibly satisfying $m < n$, which is the smallest committee size that retains PBFT's properties. To provide resilience against DoS attacks, the re-election of these committees is frequent and unpredictable.

Helix explicitly defines the interests driving its consensus nodes and emphasizes fairness among them in regard to these

definitions. Specifically, it focuses on ensuring fair distribution of power among the nodes and on guaranteeing that the order of transactions on the blockchain cannot be manipulated by a single node (or a small coalition). Helix also takes into account the end-users of the protocol and aims to protect them from being censored.

Helix concentrates on the *ordering* of transactions. It does not attribute meaning to the transactions it arranges and is completely unaware of state changes these transactions may invoke[1]. It is therefore possible to have nodes completely blind to the content of the transactions, using threshold encryption that obfuscates them. Only when the order of transactions becomes final is the content of the transactions revealed. The threshold encryption scheme is further used to produce verifiable randomness within the protocol. Helix incorporates a randomness generation scheme into the protocol without requiring any additional rounds of communication. It then makes use of randomness in two places: first, to elect the committees in a non-predictable and non-manipulable manner; and second, to realize correlated sampling, which is used to choose pending transactions in a fair and verifiable way. The problem of using multiparty computation to produce verifiable randomness has a long history both in and apart from the context of cryptocurrencies [10], [27], [20].

Helix assumes a known list of consensus nodes responsible for constructing and validating blocks. Such a list can be defined and modified frequently with various governance schemes—permissioned, based on an authority that explicitly determines the composition of the list; or permissionless, based on stake in the system, PoW or other criteria. For simplicity, through the course of this paper we assume that a set of consensus nodes is given. The fact that consensus nodes are identified allows Helix to introduce a reputation measure, which estimates a node's compatibility with the protocol's instructions. The reputation measure is used to compensate or punish a node, incentivizing correct behavior.

We explicitly mention our three main contributions. First, Helix achieves consensus over a **fair** ordering of transactions, where each block includes an unbiased set of transactions. Put differently, a node cannot prioritize transactions it favors and include them in the block it constructs. Such a restriction enables scenarios where transactions fees cover only the processing costs of a transaction but bare no revenue. In such scenarios nodes prioritize transactions according to other criteria[2] and such type of fairness becomes relevant. Second, in Helix **end-users**, which do not take active part in the consensus protocol, enjoy protection from being censored or discriminated against by the nodes that connect them to the network. This is achieved by having the users encrypt their transactions prior to transferring them to their nodes. Finally, while many consensus protocols rely on a stable leader to progress rapidly and suffer great latencies from leader substitution, Helix, under normal flow, efficiently rotates leaders (and committee members). The leader rotation comes only with negligible communication overhead and, unlike other protocols that use a round-robin scheme, is unpredictable and can be weighted non-equally between the nodes.

The remainder of the paper is organized as follows. We begin with some short background in Sec. II, covering basic concepts in distributed systems and a few cryptography primitives used by Helix. Then, in Sec. III we describe our system model. In Sec. IV we give a detailed description of the Helix protocol. Sec. V is dedicated to prove basic properties Helix satisfies. In Sec. VI we focus on epochs with high transaction rates and suggest a sampling scheme to construct blocks in a fair manner. Finally, in Sec. VII we propose a method to synchronize nodes rapidly.

II. Background

A. Distributed systems

In a distributed system, independent entities run local computations and exchange information in order to complete a global task. A fundamental problem in the field is reaching agreement on a common output value. In this problem we consider n entities, each associated with an input value, and the goal is to design an algorithm, executed locally by each entity, which ensures all entities output the same value. Existing agreement algorithms are designed with various execution environments in mind, or possible behaviors by entities, and result in different performance characteristics.

The performance of a protocol is affected by the quality of the network communication. A few types of synchronous environments we refer to throughout the paper, taken from [15] are given hereafter:

Definition 1 (Strong synchronous network) *A network is said to be strongly synchronous if there exists a known fixed bound, δ, such that every message delays at most δ time when sent from one point in the network to another.*

Definition 2 (Partial synchronous network) *A network is said to be partially synchronous if there exists a fixed bound, δ, on a message's traversal delay and one of the following holds:*

1) δ always holds, but is unknown.
2) δ is known, but only holds starting at some unknown time.

A particular execution environment, considered to be general and least constraining with regards to the communication synchrony is the asynchronous network.

Definition 3 (Asynchronous network) *A network is said to be asynchronous if there is no upper bound on a message's traversal delay.*

In the settings presented above, network links can either be *reliable* or *unreliable*. Reliable links are guaranteed to deliver all sent messages, while with unreliable links, messages may be lost in route. In both cases, and in all the environments

[1]The execution and validation of transactions is beyond the scope of this paper.

[2]For example, we can think of nodes as application developers that prioritize transactions made by their own users.

described above, dispatched messages can be delivered out of order.

We formulate a primitive that captures the essence of agreement by the three following requirements:

Definition 4 (Non-triviality) *If a correct entity outputs a value v then some entity proposed v.*

Definition 5 (Agreement) *If a correct entity outputs a value v then all correct entities output the value v.*

Definition 6 (Liveness) *If all correct entities initiated the protocol then, eventually, all correct entities output some value.*

An early protocol to propose a solution to the agreement problem was Paxos [24]. In Paxos, entities either follow the protocol's prescriptions, or have crashed and thus do not participate at all. Paxos is non-trivial and maintains the agreement property under all circumstances, but is guaranteed to be live only when the network is synchronous and less than half of the entities have crashed. Raft [30], a modern Paxos variant, explicitly defines a timeout scheme that helps achieve liveness when communication is synchronous and reliable.

Fischer, Lynch and Paterson [18] showed that a deterministic agreement protocol in an asynchronous network can not guarantee liveness if one process may crash, even when links are assumed to be reliable[3]. A key idea there is that in an asynchronous system one cannot distinguish between a crashed node and a correct one. Hence, deciding the full network's state and deducing from it an agreed-upon output is impossible.

A complication of the agreement problem assumes a more involved failure model, where nodes may act maliciously in addition to crashing. A malicious entity does not follow the protocol's instructions and behaves arbitrarily. This problem is often called *Byzantine agreement* (BA) and was introduced by Pease, Shostak and Lamport [25]. A well known result in the field of distributed systems is that in an asynchronous network, agreement cannot be reached unless less than $\frac{1}{3}$ of the participants are Byzantine.

A natural extension of the single-value agreement problem is to agree on a set of values. In agreement on a sequence (log), the agreement needs to cover both, the output values and their order. Given a solution for a single-value agreement, it can be used as a black box to achieve agreement on a sequence. However, a version of Paxos (sometimes called Multi-Paxos or multi-decree Paxos) optimize it by dividing the consensus protocol to a fast "normal mode" and a slow "recovery mode". The recovery mode, (which is also run initially when the protocol starts), selects a leader with a unique (monotonically increasing) ballot number. In the normal mode, the leader coordinates agreement on a sequence of values using an expedited protocol. If the leader fails or is suspected to have failed, the slow recovery mode is run to elect a new one. Similarly, Castro and Liskov proposed Practical Byzantine Fault Tolerance (PBFT) [11] which was the first efficient protocol to implement agreement on a sequence in the presence of (less than a third) Byzantine entities, employing the same type of optimization. We further formulate a primitive that satisfies *agreement on a set of values*, written to a log, through the previous requirements, together with an additional requirement. In agreement on a sequence, values are written to a specific slot or index in the log.

Definition 7 (Strong consistency) *For any pair of correct entities i, j with corresponding logs $(v_0^i, v_1^i, \ldots, v_l^i)$ and $(v_0^j, v_1^j, \ldots, v_{l'}^j)$ where $l \leq l'$, it holds that $v_k^i = v_k^j$ for every $k \leq l$.*

A protocol that satisfies agreement and strong consistency is said to be *safe*. Intuitively, this means that for every position in the log, or index, only one value can be agreed upon and once a value was agreed upon in a certain position, it will stay there indefinitely.

We now move forward to a different approach to reach agreement, referred to as Nakamoto consensus [29]. The fundamental data structure that enables the consensus there is the famous *blockchain*. A blockchain, similarly to a linked list, is a sequential data structure composed of blocks, each pointing to the previous one by storing its hash. This yields a very strong property where any modification to the data included in a block utterly changes the output of its hash. Hence, modification of one block in the blockchain causes any succeeding block to change. Therefore, securing the hash of the current block guarantees that any previous block was not tampered and consensus on the history can be kept.

Reaching consensus on the current block remains the main problem. The Nakamoto consensus allows for multiple values (blocks) in the same index (height), resulting in a data structure that is better interpreted as a tree of blocks rather than a chain. Of the branches within this tree an entity is aware of, it selects the longest one as the valid chain (more accurately, it selects the branch that admits to the most amount of accumulative work according to the PoW principle [16], which we do not explain in this paper). Over time, this rule should yield a single prefix-chain that is agreed upon by all. Nakamoto consensus is thus said to satisfy *eventual consistency*, rather than strong consistency or *finality*.

The consistency of the Nakamoto consensus highly depends on the network being strongly synchronous, propagating new blocks in negligible time relative to the average block creation rate [31]. In case this assumption does not hold, the resilience of the longest chain rule may become fragile, requiring an attacker to obtain fewer resources in order to successfully reorganize the valid chain[4]. In case the assumption does hold, the Nakamoto consensus satisfies only eventual consistency.

[3] When links are unreliable, it is well known that agreement is impossible even in a strongly synchronous network, a result known as the two generals problem [19].

[4] The GHOST [35] protocol suggests the "heaviest subtree" rule to determine the valid chain, partially mitigating this problem.

B. Cryptographic primitives

Threshold Cryptography

Threshold cryptography [14] refers broadly to techniques for allowing only *joint groups* of people to use a cryptographic system, be it to compute or verify signatures, or to encrypt or decrypt data. In particular, a (k, n)-threshold cryptosystem is executed by n entities, any k of which (for some fixed $k \in [1, n]$) are able to decrypt messages successfully. Threshold security guarantees that any attempt by up to $k-1$ of the share holders to decrypt the original message is bound to fail, whenever an assumption on the hardness of the security requirement of the underlying scheme holds. The (k, n)-threshold cryptosystem we refer to in this paper consists of the following components:

1) A distributed key generation scheme executed once to set up a master public key MPK, secret keys $(S_0, \ldots, S_{n-1})$ and verification keys $(V_0, \ldots, V_{n-1})$.
2) An encryption scheme which uses the master public key.
3) A threshold decryption scheme which consists of two parts. The first is an individual secret key decryption of the cyphertext by each entity in order to obtain its decrypted share. The second part allows joining k decrypted shares in order to retrieve the original message. A verification procedure is utilized in both parts in order to validate the cyphertext before producing a share, and the validity of the shares before joining them. The verification procedure is important for avoiding spam attacks.

Hash Functions

We rely on an efficiently computable cryptographic hash function, H, that maps arbitrarily long strings to binary strings of fixed length. One property we will explicitly assume our hash function admits to is *collision-resistance*:

Definition 8 *A hash function H is said to be* collision resistant *if it is infeasible to find two different strings x and y such that $H(x) = H(y)$.*

Additionally, we model H as a random oracle [6] as is often done.

Digital Signatures

Digital signatures enable entities to verify that a message was sent by a certain entity. Digital signatures usually require a secret key, denoted by sk_i, used by entity i for signing the message, and a public key, denoted by pk_i, used for verifying the signature of entity i. To avoid anyone else from signing on its behalf, an entity should not share its secret key. Any entity with a knowledge of the public key can verify the signature. A digital signature scheme typically consists of three schemes: a key generation scheme, a signing scheme, and a verification scheme.

In order to produce a new pair of keys, sk and pk, the generation scheme is used. The binary string $sig_i(m)$ is referred to as the digital signature of entity i over a message m and is produced using the signing scheme and sk_i.

The main properties required from the signature scheme are:

1) Legitimate signatures pass verification: $sig_i(m)$ passes the verification scheme under i's public key and the message m.
2) Digital signatures are secure: Without knowledge of sk_i it is infeasible to find a string that passes the verification scheme, for a message m that was never signed by i beforehand.

III. Model

In this section we introduce the model assumptions that the Helix protocol relies on to achieve safety, liveness and other properties. We assume a strongly synchronous distributed system where participants are connected by a network over which they exchange messages in order to achieve consensus. The participants are presented first, then the network topology and finally the adversarial model. We also mention two communication schemes used in our protocol.

A. Participants

There are two types of participants—users and nodes. Both can locally generate cryptographic key pairs which serve as unique identifiers and help to ensure message integrity (as described in Sec. II-B).

Node: Nodes are assumed to form a fixed and known in advance set of public keys (and their corresponding private keys). Nodes are run by strong machines with practically non-limited storage and abundant computation power (partially justified by parallelism capabilities). This assumption implies that reasonable local computations are done instantaneously, e.g., signature verification, hash function computation or decryption of encrypted messages. Conversely, we assume nodes cannot break the cryptographic primitives used in the protocol, e.g., forging digital signatures or finding hash collisions. We further assume that each node owns an internal clock, and that all clocks tick at the same speed; we do not, however, require that the clocks be synchronized across nodes.

User: Users form an open set of public keys and can join or leave the network as they please. They do not actively engage in the consensus process and can be seen simply as a virtual fragment of the node they are connected to. We assume users' resources are limited in capacity.

B. Network topology

There are two types of network connections: *node-node* connections and *user-node* connections. A node-node connection exists between each pair of nodes, such that the nodes are characterized by a clique topology. User-node connections are organized differently, where each user is connected to exactly one node. We assume there are many more users than nodes.

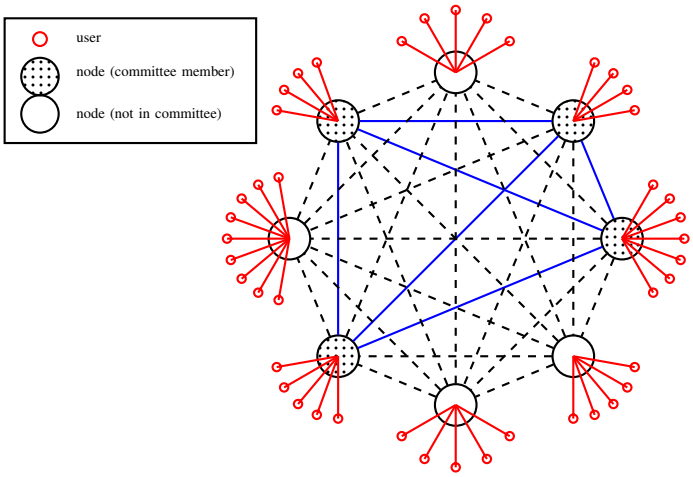


Figure 1: Illustration of a Helix network with $n = 8$ nodes, serving various numbers of users. Among the nodes, $m = 4$ nodes (filled with dots) participate in the committee of some term r.

Node-node connections are assumed to be strongly *synchronous*[5], i.e., messages transmitted over them, are guaranteed to be delivered within a known time period δ. This is a reasonable assumption under our circumstances, where the protocol assures limited-sized messages, and node participation is conditioned. In the design of our protocol, we would like both connection types to consume a limited amount of bandwidth[6].

C. Adversary

We distinguish between *correct* nodes, defined as nodes that follow the protocol's instructions precisely, and *faulty* nodes, which do not. Among the faulty nodes, we further distinguish between *Byzantine* faulty nodes, defined as nodes that act arbitrarily, and *benign* faulty (or sleepy) nodes, which are unresponsive due to a crash or lack of information. We denote the total number of nodes in the network as n, the number of Byzantine nodes among them as f_{Byz}, and the number of sleepy nodes as f_{Slp}. Thus, the number of faulty nodes is $f = f_{\text{Byz}} + f_{\text{Slp}}$. We restrict the adversary to control at most f faulty nodes, where[7] $3f + 1 \leq n$. Note that to achieve safety, it is enough to bound f_{Byz} rather than $f = f_{\text{Byz}} + f_{\text{Slp}}$.

We assume a single powerful *adversary* that determines which nodes are correct and which nodes are faulty. Generally speaking, the protocol is divided into *terms*, where in each term a single block is added to the blockchain. The adversary is *static* in the sense that it must pre-determine which nodes will be faulty during a given term r, before the term starts. The adversary is *adaptive* in the sense that in different terms, it can choose different faulty nodes. In a specific term, the adversary is restricted to sign only on behalf of the nodes it controls in that term[8]. The adversary controls all the faulty nodes and coordinates their behavior. In addition, it delivers messages between the faulty nodes instantaneously. Hence, the adversary may perfectly coordinate attacks it wishes using all the corrupted nodes.

We maintain that in the context of our environment, assuming network synchronization and a small number of faulty nodes is realistic. First, strong synchronization can be justified by constructing a highly connected network and requiring nodes to obtain high speed connections[9]. Second, a reputation mechanism maintained in our system incentivizes correct participation and increases nodes reliability (see Sec. IV-D for details). The assumed bound on the number of faulty nodes is implied by the chances of each node to be faulty. We note that Helix's performance increases as this bound can be tuned smaller.

Fig. 1 shows an example of a Helix network with eight nodes (illustrated by black large circles), serving various numbers of users (appearing as small red circles). A subset of four nodes (shown as nodes filled with dots) is selected for some round r of the protocol as a committee, taking part in the selection of the next block of Encrypted-transactions.

D. Communication schemes

The protocol uses two high level approaches for communication. The first aims to reduce dissemination time while the second aims to maximize originator anonymity. In both approaches, nodes' bandwidth consumption is limited, i.e., a node is restricted in the number of nodes it sends messages to.

According to the *fast broadcast*, a message is multicast to at most $2(f + 1)$ by each node and is propagated to the whole network within $T = O(\log(n/f))$ hops. If we denote by Δ the bound over propagation time in fast broadcast, it is clear that $\Delta = T \cdot \delta$ (under the assumption that f is small enough). When anonymity is of interest, a second approach

[5]We emphasize that if our network violates the synchrony assumption, i.e., it is asynchronous, our protocol nonetheless satisfies the safety property as shown in Sec. V-A.

[6]While many existing BFT protocols assume to be run over a LAN (local-area network) where bandwidth is not the bottleneck, we consider a WAN (wide-area network) where bandwidth constraints can be more strict.

[7]We further limit the adversary in the amount of resources it controls, which in our protocol is attributed to reputation. We assume the adversary controls at most f/n of the accumulative reputation of all nodes. This assumption is analogue to the common restriction of Byzantine hash power in PoW chains or stake in PoS chains.

[8]One practical way to achieve such a restriction is to utilize ephemeral keys for signing messages. In general, a node creates a pairwise ephemeral key for each term, composed of a secret key and a corresponding public key, which is uniquely associated both to the identity of the node and to some term number. The node publishes in advance (e.g., for the next thousand rounds) a commitment for these exclusive keys (one way of creating such a commitment is by using a Merkle tree and publishing the root). Every term the node signs messages with the term's ephemeral secret key and destroys it when the term ends. Hence, if the adversary takes control over the node in a subsequent term, it cannot sign messages from an earlier term. For conciseness reasons, we leave further implementation specific details for a later time.

[9]Centralized relay services such as Bitcoin's FIBRE network (see: http://bitcoinfibre.org/) serve as a good example for a fast, synchronous and reliable network.

is used. This approach, which we shall refer to as *anonymous broadcast* generalizes a simple ring topology and uses different heuristics.

We emphasize that the fast scheme is deterministic rather than using random gossip. This implies a deterministic bound for the time a message may propagate in the network. Moreover, it is resilient to f Byzantine nodes that refuse to follow instructions and act arbitrarily. The fast scheme is described in detail in App. A, together with general methods to increase anonymity in broadcast schemes.

IV. The Helix protocol

In this section we first illustrate the main data types used in the Helix protocol and later describe the protocol in detail.

In Helix, users issue **Encrypted-transactions** (etxs), which they send to the node they are connected to via a user-node connection. We refer to the receiving node as the *owner* node of the etx. etxs are stored locally by every node in its **Epool**, pool of Encrypted-transactions. The protocol's goal is to order the etxs, such that the order is agreed upon by all correct nodes. The etxs are grouped into **Eblocks**, blocks of Encrypted-transactions, which will ultimately be incorporated into the blockchain. The protocol progresses in iterations, or **terms**, where in each term exactly one Eblock is appended to the blockchain.

Agreement on the next Eblock in the blockchain is achieved using a Byzantine agreement protocol, run within a **committee** comprising a subset of the nodes. A special feature of Helix is that the committee members constructing an Eblock do not know the content of the transactions or their owner nodes. This property is achieved by encrypting transactions using a threshold encryption scheme (see Sec. II-B) and broadcasting them anonymously. After a committee reaches an agreement on a new Eblock, a proof for this agreement, referred to as a **Block-proof**, is constructed and broadcasted to all nodes. Only then is the Eblock decrypted, in a process where, first each individual node produces its own **Shares-block**, and $k = f+1$ (for a further explanation see Sec. IV-C) different Shares-blocks are later combined to generate the **Dblock**, block of decrypted etxs, corresponding to the Eblock.

Each node incorporates a so-called **Encrypted-secret** into the Shares-block that it generates. When the Dblock is revealed (i.e., decrypted from the corresponding Eblock), the Encrypted-secrets are also decrypted, and combined together to generate an unpredictable **random seed**. This random seed is used, together with the **reputations** of the nodes, to determine the new committee members responsible for choosing the Eblock in the next term. Intuitively, the reputation is a measure of node's behavior in the protocol.

A. The Helix data types

Next, we overview the data types involved in the protocol.

Transaction (tx) and **Encrypted-transaction** (etx). Transactions are the atomic pieces of information[10] that Helix communicates and orders. Users issue transactions and sign them to prove their system-identities. They then employ a threshold encryption scheme, where only k or more cooperating nodes can decrypt. By ordering encrypted transactions prior to their disclosure, Helix enhances the fairness of the system (as discussed in Sec. V-C).

Secret (se) and **Encrypted-secret** (ese). A Secret is a high-entropy random bit string generated periodically by each node, used to implement a common random function. The Encrypted-secrets are the secrets encrypted according to the threshold encryption scheme. Each ese is propagated along the network after being signed by its issuer and associated with a term number. For issuer node i, we denote by $\langle ese, r\rangle_{\sigma_i} = ese_i^r$ this signed information in term r.

Epool. The Epool is a collection of etxs stored and maintained locally by each node. The etxs contained in a given node's Epool may either have been issued by the node's users, or forwarded from other nodes (we emphasize that the threshold encryption scheme, together with Helix's communication model, obfuscate the issuing user and its owner node). Once an etx is included in a committed Eblock (i.e., an Eblock that is indefinitely appended into the blockchain) it is removed from all Epools where it appears.

Committee-map (Cmap^r). A Committee-map is a (dynamic) ordered list of the nodes in the system and their respective reputations, corresponding to a specific term. The first $m = 3f + 1$ (see Sec. V-A) nodes in Cmap^r form the committee of term r, which determines EB^r. A Committee-map for term r is generated after the committed Eblock for the previous term EB^{r-1} is decrypted. The calculation of Cmap^r is done locally and deterministically by each node, and depends on the random seed RS^{r-1} and the nodes' reputations.

Eblock (EB^r). Helix's blockchain is made up of Eblocks, each comprising two parts: payload and header. The payload contains a Merkle tree of etxs and a list of signed eses. The header contains the following metadata:

- Term number r.
- Current Committee-map Cmap^r.
- Hash of the previous Eblock's header, denoted $H(EB^{r-1})$.
- Hash of the previous Decrypted-block's header, denoted $H(DB^{r-1})$.
- Merkle root for the Merkle tree of the etxs (included in the Eblock).
- Hash of the concatenated eses (included in the Eblock).
- Composer. The composer is the node that originally constructed the current EBlock EB^r from its Epool. The composer's identity is taken into account for reputation updates.

[10] In many cases it is useful to think of transactions as inputs to some transition function that updates a state. However, for the sake of generality, in this article we avoid attributing any particular semantics or functionality to them.

- View. The view in which EB^r was committed (see Sec. IV-C for details). The view is also taken into account for reputation updates.

We assume that EBs are limited in *capacity*, where capacity can be defined in many ways: space (such as in Bitcoin, where blocks are limited to some predetermined memory size) and "gas" (such as in Ethereum, where blocks are limited according to the amount of computation and storage usage they invoke) are some known examples. We focus on space and denote by b the (maximal) number of etxs in an EB.

Block-proof (BP^r). BP^r is a list of messages and signatures, collected by the committee of term r, that acts as proof that EB^r has successfully undergone all the phases of Helix's base agreement protocol (see Sec. IV-C). It manifests that EB^r is valid and safe to append to the blockchain. BP^r metadata includes $H(EB^{r-1})$ and the term number r. We emphasize that the Block-proofs generated by different nodes in a given committee might differ from one another. The exact content of a Block-proof is described in Sec. IV-C.

Shares-block (SB_i^r). SB^r consists of a list of shares, produced by a single node i for term r; the list comprises one share for each etx in EB^r and one share for each ese^r in EB^r. Each node is capable of producing exactly one Shares-block per term using its threshold encryption secret key. We denote by SB_i^r the Shares-block of node i in term r. A node that has received k Shares-blocks for term r can then decrypt EB^r. Upon receiving a Shares-block from a different node, a node can verify the Shares-block's authenticity using its threshold encryption verification key. The metadata of a given Shares-block SB_i^r includes $H(EB^r)$, the term number r, a legally-generated ese_i^{r+1} and the signature of node i.

Dblock (DB^r). DB^r is the decrypted version of EB^r. Similarly to an Eblock, the Dblock contains two parts: header and payload. The payload contains a Merkle tree of txs (decrypted transactions) and a list of ses (decrypted eses), whose order is dictated by their order in EB^r. The header includes the following metadata:

- Term number r.
- $H(EB^r)$.
- Merkle root of the (decrypted) transaction tree.
- New random seed. The new random seed is calculated as follows: $I^r = se_1 \oplus se_2 \oplus \cdots \oplus se_k$ and $RS^r = H(I^r)$. In Sec. V-C we show that RS^r cannot be predicted by any of the nodes and thus serves as a common random function.

Fig. 2 illustrates the Helix blockchain. Each block is composed of an EB, a BP, and a DB, as well as k SBs. A block includes internal hash pointers between components of the block as well as hash pointers to components in the previous block. These two types of pointers are illustrated in solid red and dashed blue, respectively.

B. Happy flow: An overview of Helix's normal operation

We describe the normal flow of the protocol in high level. Node $i \in [0, n-1]$ reveals the Dblock of the previous term DB^{r-1} and obtains the new random seed RS^{r-1}. It then calculates for $j \in [0, n-1]$ the values.

$$v_j^r = \frac{H\left(RS^{r-1},\ r,\ pk_j\right)}{rep_j^r}. \tag{1}$$

Here, rep_j^r is the reputation of node j in term r. Cmap^r is ordered according to $\{v_i^r\}_{i=0}^{n-1}$, where the m nodes with the minimal values are considered the committee for term r and the first node among them is the *primary*.

Once the primary finds out its role, it initiates the consensus base protocol (PBFT), suggesting a new Eblock as the value. In order to validly construct an Eblock the primary chooses uniformly at random at most b etxs from its Epool. If a correct primary constructs a valid block, then with overwhelming probability (see analysis in Sec. V-B), it succeeds in committing it and thus a Block-proof for it is created. Committee members that complete assembling BP^r, then add EB^r to their blockchain and broadcast the pair $\langle EB^r,\ BP^r\rangle$ to all nodes (in particular to non-committee members). Now, the decryption process starts. Nodes that do not hear of k corresponding Shares-blocks SBs prior to receiving the pair $\langle EB^r,\ BP^r\rangle$, produce their version of SB^r and broadcast it to all nodes. Upon receiving k Shares-blocks, EB^r is decrypted and DB^r is revealed. Finally, a new term begins.

C. The four segments of the Helix protocol

For clarity of presentation, we divide Helix's operation into four distinct *segments*, where each segment is responsible for a specific logic within the protocol. The first handles an initial setup phase; the second is responsible for etxs propagation; the third handles dissemination of information on the consensus result to all nodes, with the purpose of concluding the current term and initiating the following one; the fourth handles achievement of consensus among committee members. Helix consists of several processes that run concurrently and invoke each other frequently. We proceed to present the different segments of the protocol and pseudo-codes for the various processes.

Segment 1—Initial setup

For simplicity, we assume that nodes interact with a trusted dealer during an initial setup phase for key distribution and randomness generation. Note that in a real deployment, if an actual trusted party is unavailable, then a distributed protocol among the nodes could be used instead [7], [12].

Threshold encryption. We use the trusted dealer to generate a master public key (MPK), secret keys $(S_0, ..., S_{n-1})$ and verification keys $(V_0, ..., V_{n-1})$. The MPK is known to all entities in the system and is used to encrypt transactions. The secret and verification keys are kept privately by the nodes

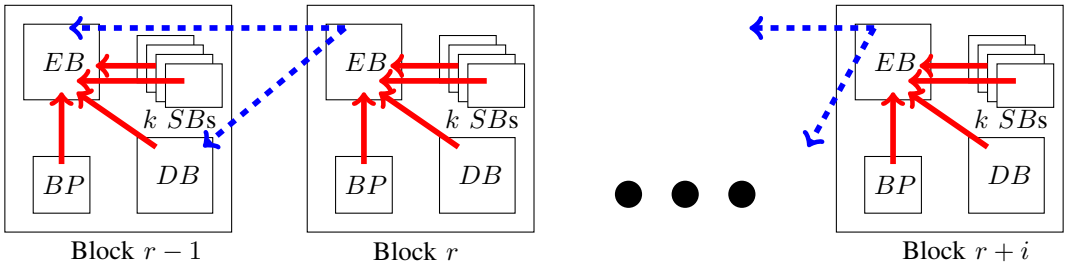


Figure 2: Illustration of the Helix blockchain. Hash pointers are used to point within blocks (in solid red) or over blocks (in dashed blue).

and are used to decrypt etxs. In this article we assume static membership of nodes[11].

Random seed. In every term, a random seed is generated as part of the decryption of the committed Eblock. The initial random seed RS^0 is hard-coded into the genesis block. The initial seed will be used to determine the first Cmap and in particular the first committee. Instead of using a dealer, a (Public)-VSS Coin Tossing scheme can be used to generate RS^0 [10].

Segment 2—*etx* propagation

Transactions are issued by users; each user signs and then encrypts each transaction that it issues[12]. Each user's unique sk is used for signing, whereas the common MPK is used for encryption. The sk is generated locally by each user, whereas the MPK is hard-coded into the genesis block[13]. After signing and encrypting is complete, the user sends the etx to its owner node.

Communicating Encrypted-transactions. As described previously, each etx is sent by the issuing user to its owner node. The owner node then forwards the etx to all the nodes in the network. A key objective of the forwarding scheme is to reduce the ability of other nodes to identify the original sender node. This is crucial for maintaining fairness (as described in Sec. V-C), which is highly dependent on nodes being incapable of associating an etx to its owner node (a node should only certify which etxs it owns). Each node maintains its own Epool by adding new etxs it receives and removing committed etxs (i.e., etxs included in committed Eblocks).

[11]An interesting topic for future investigation is removing the need to redo the setup process every time a node joins or leaves. A possible approach is to generate extra key pairs and distribute them among the nodes according to some secret sharing scheme. When a new node joins it collects enough shares to construct its key pair, in the spirit of [12]. Supporting removal of nodes can be more tricky.

[12]The choice to have users sign and then encrypt transactions favors user's anonymity, but exposes the network to spam attacks. Under our model assumptions, these attacks are mitigated by the nodes, where each node is responsible for its users. As a matter of fact, the bare minimum required is a mechanism that reveals the owner nodes of committed transactions. If this cannot be achieved in a real deployment system, user anonymity can be given up.

[13]To address cases in which the MPK changes due to a reconfiguration of the participating nodes, it is necessary to have a safe means for updating the users. We leave this discussion for future work.

We emphasize that the progress of segment 2 is independent of the advancement of the blockchain as described in segments 3 and 4.

Process User

Let sk be the user's secret key and MPK the network's master public key.

// Issuing transaction and casting it

1: $tx \leftarrow$ IssueNewTransaction ()
2: $\sigma \leftarrow$ SignTransaction (tx, sk)
3: $etx \leftarrow$ EncryptTransaction $(\langle tx \rangle_\sigma, MPK)$
4: Unicast(etx) ▷ to owner node

Segment 3—Information dissemination

The main purpose of segment 3 is to update all non-committee members as to the consensus result from the previous term and to kickstart the next one.

Eblocks and Block-proofs. In a certain term r, once the consensus procedure for EB^r has concluded and some committee member has built a valid BP^r, the pair $\langle EB^r, BP^r \rangle$ should be propagated to the whole network as rapidly as possible. Once a non-committee member receives the pair $\langle EB^r, BP^r \rangle$, it performs a *brief-validation* and appends EB^r to its blockchain. A brief-validation of a pair $\langle EB^r, BP^r \rangle$ includes the following checks:

1) The Cmap appearing in EB^r's header matches the one calculated locally by the node. This means that a correct node will not append $\langle EB^r, BP^r \rangle$ unless it had already appended the previous Eblock and can calculate the current Cmap and the composition of the current committee. This step can actually be dropped as we explain in Sec. VII.
2) BP^r serves as a valid proof that the base agreement protocol has been carried out successfully (as described in segment 4) by the rightful committee (i.e., all the signers appearing in BP^r appear in Cmap$[0, \ldots, m-1]$).

Shares-blocks and decryption. As noted briefly above, Shares-blocks are constructed by nodes that have committed an Eblock and have not yet received k Shares-blocks (the latter condition is set to reduce the number of Shares-blocks propagating in the network) for that Eblock. Normally, committee members would be first to hear about new committed

Eblocks and would thus be the ones producing Shares-blocks (this however strongly depends on message propagation in the network). After a node assembles a Shares-block, it broadcasts the Shares-block to all nodes through the fast forwarding scheme. Once a node obtains k Shares-blocks for a committed Eblock, the Eblock is decrypted and the Dblock is revealed. We emphasize that the time from committing an Eblock to decrypting it is wasted, and we reduce it by tuning k to be as small as possible. In order for the protocol to be resilient against f Byzantine nodes, we set $k = f + 1$.

Process Node (for node i)

Let r be the term number and $Cmap^r$ the Cmap of term r.

1: Upon receiving a briefly validated $\langle EB^r, BP^r \rangle$ and k SB^rs
2: $DB^r \leftarrow$DecryptEblock(EB^r, $SB^r[0, \ldots, k-1]$)
3: AppendToBlockchain$\big([EB^r, SB^r[0, \ldots, k-1], BP^r, DB^r]\big)$
4: $Cmap^{r+1} \leftarrow$UpdateCmap(DB^r, $Cmap^r$)
5: $r \leftarrow r + 1$
6: Broadcast$\langle EB^r, BP^r \rangle$
7: If $i \in Cmap^r[0, \ldots, m-1]$
8: Trigger Process PBFT

Segment 4—Base agreement protocol

Segment 4 handles inter-committee messages related to the base agreement protocol—a simplified version of PBFT. The use of a Byzantine agreement protocol enables Helix to reach finality, i.e., no two correct nodes can commit two different Eblocks in the same term (as proven formally in the next section). We split the segment's description into two parts. First, we describe segment 4 as a black-box primitive that achieves certain properties. Then, we explain how PBFT is used to implement it.

Black box primitive. The basic agreement protocol run between m nodes $u_1, \ldots, u_m$ shall implement the following primitive.

Definition 9 (Black-box primitive) *An agreement protocol is said to implement the black-box primitive for Helix's segment 4 if it satisfies the following properties:*

- *Validity. If a correct node commits a value α then some node proposed α.*
- *Liveness. If all correct nodes initiated the protocol then all correct nodes commit a value.*
- *Safety. If a correct node commits a value α then every correct node commits α.*
- *Verifiability. If a correct node commits a value α, then it can produce a proof p_α such that anyone can verify that α was agreed upon by the m nodes.*

A typical Byzantine agreement (BA) protocol satisfies the first three items. Nodes that for any reason have not participated actively in the BA and wish to commit the agreed-upon value after the fact can do so (safely and efficiently) using the verifiability proof. Helix's base agreement protocol implements the black-box primitive, where EB^r is the value agreed upon and BP^r serves as a proof that it was indeed agreed upon by nodes $u_1, \ldots, u_m$. The question remains though—which nodes are the rightful committee members in term r? In order to avoid forks, Helix must figure out a way for all the participating nodes to come to an agreement as to a single committee. This is done using EB^{r-1} which contains a commitment to the seed that determines the r-term committee. Using an inductive argument, if there are no forks in term $r-1$, then only a single committee is possible in term r and accordingly there can neither be forks in term r. Finally, since the first committee is hard-coded into the genesis block and agreed upon by all nodes, Helix enjoys no forks.

Simplified PBFT as the black-box primitive. In Helix, the black-box primitive is implemented using a simplified version of PBFT [11] that reaches agreement over a single value. We assume a familiarity with PBFT's logic and terminology and describe Helix's adjustments and modifications to PBFT[14].

Accepting an Eblock. In PBFT, a node *accepts* a pre-prepare message (and the value it proposes) by broadcasting a corresponding prepare message after validating the following:

1) The view in the pre-prepare message matches its internal view number.
2) The pre-prepare message is signed by the rightful primary in that view.
3) It has not accepted a pre-prepare message in the same view containing a different Eblock.

In Helix, when accepting EB^r, a correct committee member further validates that:

1) EB^r's header contains Cmap^r as locally computed with DB^{r-1} and EB^{r-1}. We emphasize that a correct committee member does not depend on EB^r's header in order to figure out whether it is in the committee or not. This implies that in order to participate in a committee a correct node must obtain DB^{r-1} and EB^{r-1}.
2) EB^r's header contains a valid hash pointer to EB^{r-1}.
3) EB^r's header contains a valid hash pointer to DB^{r-1}.
4) EB^r's header contains a valid hash of concatenated eses, i.e., $H(ese^r_{i_1}, \ldots, ese^r_{i_k})$, properly signed by the different nodes $i_1, \ldots, i_k$ (taken from the k SB^{r-1}s the composer node used in order to decrypt EB^{r-1}).
5) EB^r's header contains the correct composer and view (this can be validated with the new-view messages, if exists).
6) EB^r's payload contains at most b etxs.

View-change mechanism. In PBFT, when timeouts expire, due to a faulty primary or an unexpected network slowdown, view-change messages are broadcasted with the aim of replacing the primary. In order to reassure safety is maintained cross-views, proofs of previously committed values are submitted within the view-change mechanism. To avoid rerunning all

[14]To keep our presentation concise, we leave PBFT's original point-to-point communication intact and do not adopt a more bandwidth-economical protocol such as CoSi [36] as proposed in Byzcoin [23] or Zilliqa [37]. In Helix, optimizing PBFT's bandwidth consumption is less critical as we assume $m \ll n$, where m is the size of the committee running PBFT.

the committed values from the beginning of time, a checkpoint mechanism is introduced where proofs are sent only for values that were committed after the last stable checkpoint. Helix's blockchain construction enables invoking a new instance of a single-value PBFT for each term (this single-value PBFT implements the black-box primitive). Thus, the checkpoint and truncating mechanism can be given up altogether (while primaries keep being replaced seamlessly). In addition, new-view messages contain only one pre-prepare message which never contains the null digest, d^{null}. Only in case a new Eblock is proposed, the Eblock itself is passed as regularly done with pre-prepare messages.

Verifying consensus. The proof that implements the verifiability property of the black-box primitive (see definition 9) is given by the Block-proof that stores a signed pre-prepare message, $2f$ corresponding signed prepare messages and $2f+1$ corresponding signed commit messages[15]. On its own, a Block-proof is not enough as a node needs to know the composition of the term's committee and make sure all the signers are indeed committee members. As mentioned before, this can be deduced from the previous term's block. We show in Sec. V-A that BP^r indeed serves as an unforgeable proof a EB^r was committed-locally by some correct committee member in term r. Thus, the pair $\langle EB^r, BP^r \rangle$, after validation, is safe to append to one's blockchain.

Timeout mechanism. We consider two timeout mechanisms for Helix. In PBFT's original approach, timeouts are deterministic and double every time the view increases, i.e., the timer in view v is set to be $2^{v-1}x$ for some value x. This ensures that in a timely network, all backups will eventually converge to a specific view (that of the current most advanced backup or one view later), even if the network performs unreliably for a restricted amount of time. This technique, although it works in theory, is very problematic [3], as timeouts must keep growing and cannot be adjusted to be shorter[16].

We suggest a different possible timeout mechanism, in which timeouts remain constant and thus highly relies on the network being reliable and predictable in latency. Every time an instance of PBFT is initiated, or when a view-change message is sent, a node's timer is reset to the same value, $x = \max\{2\Delta,\ \Delta + 3\delta\}$. In a nutshell, this value is set so that it expires only after a committee member is certain a correct primary had enough time to get its Eblock committed. In the next section, we prove formally that Helix is indeed live under the condition any piece of information a node broadcasts (according to the fast forwarding scheme) is received by all correct nodes within a known bound Δ (see App. A that discusses communication schemes).

[15]We note that in order to reduce the size of such a proof, aggregation of signatures or a threshold signature scheme may be deployed, as in [2].

[16]As a matter of fact, once a backup commits-locally a value in PBFT, it could potentially start over with timer set to x in the next view, leaving in previous views at most f backups. These would eventually be synced with the checkpoint mechanism.

D. Incentive mechanism in Helix

Processing a transaction in Helix consumes resources - bandwidth, storage and computation - and costs money. We assume each node can be compensated for the processing costs of transactions that serve its own users. However, the costs for processing transactions for other nodes' users may be considerable. If all nodes produce the same amount of transactions, compensation is settled trivially as costs balance. If nodes produce different amounts of transactions, a fee mechanism that offsets costs among nodes is required. Fees would basically be paid by nodes with a high transaction rate to those with a low transaction rate[17].

Fees, in addition to balancing costs, serve as means to incentivize nodes to follow the protocol's instructions and allow it to reach optimal performance. This is achieved by a reputation score that is attributed to each node and is updated frequently. The reputation strives to serve as a reliable measure to a node's quality of service. A node's reputation is taken into account when calculating its fee tariffs, where nodes with low reputation can charge lower fees for their services than their peers, possibly resulting in losses. Because of that, it is important that both the measurement of the resource usage and of the quality of service are accurate.

Measuring a node's usage is straightforward after revealing the owner nodes of all etxs in the decryption process[18]. Making sure the reputation is consistent with Helix's requirements and instructions is more complicated. We suggest a few practical criteria that the reputation may examine, but a more elaborate discussion as to its exact specification is due at a later time. We further note that with the evolution of the system, the reputation measure may be enhanced to mitigate new found vulnerabilities in the protocol or to encourage new desired behaviors. The discussion above makes clear that nodes are incentivized to increase their reputation, but also wish to include their own etxs in Eblocks as fast as possible. These two distinct preferences may be conflicting, requiring nodes to prioritize.

We shall now illustrate a naive reputation measure. It is updated frequently (e.g., every 1000 blocks or every month), locally and deterministically by each node, according to the blockchain. A few concrete examples:

- In order to make sure that nodes maintain high uptime and stay available, every node will be evaluated according to the average view number of committed Eblocks when it was a committee member. The higher the average view, the lower the reputation should be.
- In order to make sure that nodes maintain the recent blockchain history, if an Eblock consists of a previously included etx, its composer will be punished by reducing its reputation.

[17]At this point we do not elaborate as to the implementation of Helix's fee mechanism, but assume that the fee is proportional to a node's transaction rate.

[18]The exact cryptographic mechanism that allows this is interesting on its own right and is left for a later work.

Process PBFT (for committee member i)

Let r be the term's number, Cmap^r the Cmap of term r, $v \in \mathbb{N}$ the view, and CEB a candidate Eblock.
Let $j \in$Cmap$^r[0, \ldots, m-1]$ be a committee member and $p_v =$Cmap$^r[v \bmod m]$ be v's primary.

Init: $v \leftarrow 0$, $CEB \leftarrow null$, $prepared \leftarrow false$, $committed\_locally \leftarrow false$, $\mathcal{P}_i^r \leftarrow null$.
Timer: Set timer to $x_0 = x$ and start countdown.

// First primary
If $i = p_0$,
  $CEB \leftarrow$ ConstructEblockFromEpool()
  Multicast$_{committee}$($\langle$PRE-PREPARE, 0, r, H$(CEB)\rangle_{\sigma_{p_0}}$, $CEB\rangle$)

// Normal flow
Upon receiving a $\langle$PRE-PREPARE, v, r, H$(EB)\rangle_{\sigma_{p_v}}$, $EB\rangle$ message
  ValidateEblock(EB) ▷ see Sec.IV-C
  $CEB \leftarrow EB$
  Multicast$_{committee}$($\langle$PREPARE, v, r, H(CEB), $pk_i\rangle_{\sigma_i}$)
Upon receiving $2f$ $\langle$PREPARE, v, r, H(CEB), $pk_i\rangle_{\sigma_j}$ messages and $CEB \neq null$
  $prepared \leftarrow true$
  Multicast$_{committee}$($\langle$COMMIT, v, r, H(CEB), $pk_i\rangle_{\sigma_i}$)
Upon receiving $2f+1$ $\langle$COMMIT, v, r, H(CEB), $pk_i\rangle_{\sigma_j}$ messages and $prepared = true$
  $committed\_locally \leftarrow true$
  $BP \leftarrow$ ConstructBP()
  PropagateEB(CEB, BP)

// Timeout: primary change proposal
Upon timer expiry, reset timer to $x_{v+1} = 2^{v+1}x_0$
  $\mathcal{P}_i^r = null$
  If $prepared = true$ then,
    $\mathcal{P}_i^r = \begin{cases} \mathcal{P}_i^r.EB \leftarrow CEB \\ \mathcal{P}_i^r.msgs_v \leftarrow 2f+1 \text{ PREPARE messages with view } v, \text{ received for } CEB \end{cases}$
  Unicast$_{p_{v+1}}$($\langle$VIEW-CHANGE, $v+1$, r, $\mathcal{P}_i^r$, $pk_i\rangle_{\sigma_i}$)
  $v \leftarrow v+1$

// Primary change takeover, for $p_{\tilde{v}}$
Upon receiving $2f+1$ $\langle$VIEW-CHANGE, $\tilde{v}$, r, $\mathcal{P}_j^r$, $pk_j\rangle_{\sigma_j}$ messages, denote them by $\mathcal{V}$
  If there exists a j s.t. $\mathcal{P}_j^r \neq null$ then,
    $j \leftarrow$ the node with the maximal v among $\mathcal{P}_j^r.msgs_v$
    $CEB \leftarrow \mathcal{P}_j^r.EB$
  Otherwise,
    $CEB \leftarrow$ ConstructEblockFromEpool()
  $\mathcal{O} \leftarrow \langle$PRE-PREPARE, $\tilde{v}$, r, $H(CEB)\rangle_{\sigma_i}$, $CEB\rangle$
  Multicast$_{committee}$($\langle$NEW-VIEW, $\tilde{v}$, r, $\mathcal{V}$, $\mathcal{O}\rangle$) ▷ NEW-VIEW messages, after validated, are treated as ▷ PRE-PREPARE messages by recipients

- In order to make sure that nodes select etxs randomly (see Sec. VI for more details), adjacent Eblocks should represent similar distributions (in terms of the owner nodes of the etxs they include). If a specific node was found to construct Eblocks that significantly deviate from the Eblocks in their surrounding, it will be penalized by reducing its reputation.
- In order to encourage nodes to stay up-to-date with protocol changes, a node that constructed an Eblock with a decommissioned protocol version will be penalized by reducing its reputation.

We emphasize that the reputation update rules that do not depend on Dblocks may be used as validation checks in the agreement protocol. On the other hand, some of the validation checks in Sec. IV-C may be relaxed and become a part of the reputation mechanism. An interesting possibility is to relate reputation to the node's stake in the system. We also note that the reputation measure is shared across other services in the system, such as the execution service.

V. Basic properties and proofs

Helix achieves three desired properties: safety, liveness and fairness. First and foremost, Helix is *safe*. As long as the bound f on the number of Byzantine (rather than sleepy) nodes

holds, even without any assumptions on network synchrony, forks cannot happen (i.e., there will not be a situation in which different nodes commit different blocks in the same term). Second, Helix is *live*. We refer to two types of liveness. First, under a weakly synchronous network, Helix achieves (eventual) liveness, meaning that new blocks are (eventually) added to the blockchain in some finite time. Second, we show that, under a strongly synchronous network, Helix achieves a stronger property, referred to as *strong liveness*, in which new blocks are added within a known bounded period of time. Finally, Helix is *fair* with regard to three types of fairness. The first two refer to fairness among nodes: election fairness (of committee members) and selection fairness (of transactions in Eblocks). The third type refers to fairness towards users. In this section, we elaborate on each of these three properties and prove that the Helix protocol fulfills them.

A. Safety

Helix is designed to provide safety even in the case of a slow and unreliable communication network. Safety implies finality, meaning that once an Eblock has been appended to the blockchain in the eyes of any (even one) correct node, then no correct node ever appends a different Eblock for that term. The safety of Helix relies on the safety of its underlying protocol, PBFT [11].

Claim 10 *(Helix is safe) Let $u_1, \ldots, u_n$ be the nodes running Helix. In term r, let EB_i^r and EB_j^r be Eblocks appended by correct nodes u_i and u_j, respectively, to their local blockchains. Then, $EB_i^r = EB_j^r$.*

Proof. For PBFT's proof to be applicable here, we need to show that the committee in term r is a well-defined and agreed-upon set of m nodes. Put differently, we need to show that no correct node can be "tricked" into believing it is a committee member when it is actually not, submitting signatures for invalid Eblocks. This is easily prevented by relying on the previous term's Eblock (or more accurately Dblock) to determine the next term's committee. A simple inductive argument can convince that there can be only a single version of DB^{r-1} and thus the composition of the r-term committee is well defined. Since all nodes compute the r-term committee from their r-term Dblock, all correct nodes will agree as to its composition. This concludes the proof using PBFT's safety proof. □

B. Liveness

In its most general sense, liveness guarantees that new blocks are added to the blockchain in some finite time. We consider two types of liveness which correspond to different synchrony assumptions and timeout mechanisms.

General liveness

For Helix to be live in the general sense, we need each PBFT instance to be live and for each member of the subsequent committee to (eventually) realize that it has been assigned this role. For this to hold, we need the same synchrony assumptions as in PBFT, namely, a weakly synchronous network, where message delay is arbitrary up to a certain point but is eventually bounded by some unknown bound.

Claim 11 *(Helix is live) Among n nodes, Helix (with the doubling timeout mechanism) continues to make progress, meaning that regardless of the internal state of the nodes, within some finite time, some correct node will commit a new Eblock to its blockchain.*

Proof. The proof follows from PBFT's liveness and the fact that committee members eventually become aware of their committee membership. Assume that the last Eblock to be committed by a correct node was in term $r-1$. We identify three possible system states:

- State 1: At least $2f+1$ correct nodes are aware of their participation in term r's PBFT instance concurrently. In this case, by liveness in PBFT, in a finite time, at least one of them will append a valid Eblock to its blockchain.
- State 2: Fewer than $2f+1$ correct committee members are aware of their committee membership, but at least one correct node knows the identities of the committee members for term r. This node has the ability (and duty, so to speak) to update all the committee members for term r (safety guarantees that this will not contradict any other correct node's view of term r's committee members) by sending them EB^r, BP^r and k SB^rs. A simple syncing and feedback protocol[19] run between the nodes ensures that this information is eventually propagated to all correct nodes. Accordingly, after a finite period of time, state 1 takes place.
- State 3: No correct node is aware of the composition of term r's committee. According to the protocol, the correct node that appended $(r-1)$'s Eblock to its blockchain is in the process of decrypting it (again, safety guarantees no forks in the term $r-1$). The decryption process duration depends only on the time of message dispersal, i.e., propagation of the committed Eblock, its Block-proof and k Shares-blocks. Thus, the process of decrypting term $(r-1)$'s Eblock is finite. Once at least one correct node decrypts the committed Eblock, we obtain state 2. This concludes the proof.

□

Strong liveness

In Helix, we wish to obtain a stronger notion of liveness that relies on two additional requirements. First, we require a time frame in which new blocks are committed. Second, we aim for a correct primary to succeed in committing its Eblock before a new primary is appointed. The first of these two requirements is driven by the need for practical liveness (with

[19] We do not get into the details of this protocol, but it is clear that as long as messages are eventually delivered, this information will propagate.

a guaranteed bound on progress time), whereas the second one is related to fairness (discussed in Sec. V-C). We refer to this stricter notion of liveness as *strong liveness*. For strong liveness to hold in Helix, we require the network to be strongly synchronous, namely, message delay is assumed to be bounded by some known constant (as explained in detail in Sec. III). To obtain strong liveness in Helix we use the constant timeout mechanism, in which $x = \max\{2\Delta,\ \Delta + 3\delta\}$ is set as the timeout, as mentioned in Sec. IV-C.

Definition 12 *(Strong liveness) A blockchain protocol (that enjoys finality), run between n participants is said to be strongly live if it satisfies the following properties:*

1) *A primary that proposes a valid block, immediately after its construction, gets it committed.*
2) *A correct node that appended the $(r-1)$-term block, appends the r-term block within some known bounded period of time.*

Before proceeding, recall the network model assumptions. Δ is the maximal time it takes for a message that a correct node broadcasts, according to the fast forwarding scheme, to reach all correct nodes, and δ is the maximal time it takes for point-to-point messages to arrive at their destinations. The relationship between δ and Δ depends on the communication protocol[20] as discussed in Sec. III.

We now prove that Helix (with the constant timeout mechanism) is strongly live if the network is strongly synchronous. We distinguish between two cases based on the status of the first primary. First, we denote by u_r the first correct node to reveal DB^r at time t_r, and with it, Cmap^{r+1}.

Lemma 13 *(Term with a correct first primary) If the first primary Cmap$^r[0]$ is correct, it will get EB^r committed prior to any correct member's timeout expiring.*

Proof. In term r, u_{r-1}'s initial timeout will be the first among the correct nodes to expire (at time $t_{r-1}+x$). Since u_{r-1} follows the communication protocol, it has propagated each piece of information it has heard or produced so far (in particular EB^{r-1}, BP^{r-1} and some k SB^{r-1}s) and thus necessarily all nodes reveal DB^{r-1} (and Cmap^r) by time $t_{r-1}+\Delta$. In particular, the first committee's primary (since assumed to be correct) would realize its role by time $t_{r-1}+\Delta$ and would multicast a pre-prepared message with EB^r to all committee members of term r. This message would reach all correct committee members by time $t_{r-1}+\Delta+\delta$ (intra-committee communication is done point-to-point). Then, by $t_{r-1}+\Delta+2\delta$ all correct committee members should receive enough prepare messages and by $t_{r-1}+\Delta+3\delta$, EB^r should be committed-locally by all correct committee members. Since all correct nodes' timers expire later than $t_{r-1}+\Delta+3\delta$, no timeouts expire before EB^r is committed-locally by all as required. □

Lemma 14 *(Term with a faulty first primary) Assume Cmap$^r[0]$,...,Cmap$^r[i-1]$ are all faulty nodes and Cmap$^r[i]$ is a correct node, where $i \in \{1,\ldots,f\}$. Then, EB^r will be committed-locally by all correct committee members in a view $v \in \{1,\ldots,i\}$.*

Proof. If EB^r is committed-locally, by even a single correct member, in view $j < i$, we are done because a correct committed-locally member would propagate $\langle EB^r, BP^r\rangle$ to all correct nodes within Δ time. This ensures that any correct committee member would append EB^r to its blockchain prior to sending a view-change message for the $(j+2)^{\text{th}}$ time (i.e., within its $j+1$ view). The $(j+2)^{\text{th}}$ view-change message can be sent, the earliest at $t_{r-1}+(j+2)x$, on the other hand, $\langle EB^r, BP^r\rangle$ would reach all nodes, the latest at $t_{r-1}+(j+1)x+2\Delta$. Indeed, we have $x \geq 2\Delta$.

Otherwise, we show that all correct members commit-locally EB^r in view i. Denote the first correct committee member to send Cmap$^r[i]$ a view-change message by u. The earliest time u can send this message is $t^i_{r-1} = t_{r-1} + i \cdot x$. Since no other correct node had already committed-locally, all correct nodes keep sending view-change messages every x time. Thus, if two correct members enter term r in some time difference, then they keep sending view-change messages with the same time difference. As was shown in the first case, this difference is bounded by Δ. Thus, by $t^i_{r-1}+\Delta$ all correct nodes would send Cmap$^r[i]$ a view-change message and Cmap$^r[i]$, being correct, would reply with a new-view message containing a valid Eblock EB^r, that would reach all correct committee members by time $t^i_{r-1}+\Delta+\delta$. As was explained before, by $t^i_{r-1}+\Delta+3\delta$, EB^r would be committed-locally by all correct committee members. Since all the $(i+1)^{\text{th}}$ timers expire later than $t^i_{r-1}+\Delta+3\delta$, none should expire before committing-locally EB^r as required. □

Corollary 15 *(Helix is strongly live in a strong synchronous network) If Helix is run over a strong synchronous network (with the constant timeout mechanism), it is strongly live.*

Proof. The first property is deduced directly from the two previous lemmas.

Second, in order to conclude the second property in the definition, we look at a correct node that appended EB^{r-1}. Since the network is strongly synchronous, then from the fast forwarding scheme, EB^{r-1} would propagate to all correct nodes within Δ time and decrypted within another Δ time. Once DB^{r-1} is revealed, the two lemmas above become relevant—the more view-changes, the longer it takes to reach agreement as to the r-term Eblock. Since there could be at most f faulty nodes within the term-r committee, we saw that there can be at most f view-changes after which it is certain a correct primary is appointed. A correct primary gets its Eblock committed in another 3δ time. Since every view lasts exactly x time, we conclude that a node that appended EB^{r-1} would append EB^r in (at most) $2\Delta + f \cdot x + 3\delta$ time. □

[20] In general gossip communication, where nodes send a new piece of information to some random group of neighbor nodes, a bound like Δ usually refers to the time it takes to reach a certain percentage of the network, say 90%. This is because the last nodes are hard to reach in such probabilistic protocols as was shown specifically in the Bitcoin network in [13].

We note that even if run with the doubling timeout mechanism, Helix is strongly live, only with a longer bound.

C. Fairness

Helix achieves *fairness* in three aspects. *Election fairness* relates to the amount of terms in which a node participates as committee member. Fairness in this sense implies that a node's participation level is proportional to its resource holdings, which in our system is related to reputation[21]. *Fairness towards users* relates to mechanisms that limit nodes' capabilities in discriminating or censoring users' transactions. In this section we discuss election fairness and fairness towards users in Helix. In the following section we analyze the third type of fairness, *selection fairness*, which thrives to achieve a blockchain in which the representation of nodes (according to their etxs) is proportional to their *transaction rate*.

Election fairness

In blockchain protocols, nodes (sometimes referred to as miners) compete in a lottery to find blocks, where a node can increase its probability of winning, if it has more *resources*. We consider this lottery as *fair* if a node's probability of winning a block (or being given the right to propose a block) is proportional to its holdings in the resource. We denote node i's resource holdings in term r as res_i^r and its proportionate share as $rep_i^r = \frac{res_i^r}{\sum_j res_j^r}$.

Definition 16 *(Election-fairness) A blockchain protocol, where nodes are randomly elected to find (or propose) blocks according to a resource holdings rep, is said to be* election-fair *if the probability of node i to be elected as the composer of the block in term r is rep_i^r.*

Nakamoto consensus for example is not election-fair—selfish mining attacks are possible by a strong adversary with $p_a \geq 0.25$, increasing its probability to find a block over p_a as was shown in [17],[33]. Another example is Algorand [27], which falls a bit short—there, a leader knows before the network what the next random seed would be in case its block is accepted. Thus, it can choose between two options—either to propose a block or not to. This introduces an advantage to the leader, even if small. The main goal in this section is to prove the following claim.

Claim 17 *(Election fairness) Helix is election-fair with respect to reputation as the resource.*

We note that in the context of Helix, election fairness is more relevant to consider in terms of committee membership rather composer election. Before proceeding to prove this claim, we need the following definition.

[21] In PoW systems, the resource is proportional to the computational ability to solve partial hash-collision puzzles. In PoS systems, the resource is proportional to stake in the system. In Helix, reputation is a concept that reflects how aligned a node is with the protocol's instructions.

The main hurdle in proving claim 17 is showing that the input that produces RS^r, namely I^r, cannot be guessed ahead of time and that it is common among all correct nodes. From our assumptions on H, this concludes that RS^r is a random oracle.

Lemma 18 *n nodes running Helix can use I^r as a common and unpredictable string of bits. Specifically, for every r:*

1) *Two correct nodes u and w with I_u^r and I_w^r have $I_u^r = I_w^r$.*
2) *Prior to EB^r being committed-locally within at least one correct node, none of the bits in I^r can be predicted with probability greater than $\frac{1}{2}$ by any node or by the adversary (with non-negligible probability).*

Before the proof, we recall the fact that a committed Eblock must contain exactly k signed eses (see Sec. IV-C).

Proof.

1) The first item simply follows from Helix's safety and the uniqueness of the decryption process.
2) Regarding the second item, from the decryption mechanism in Helix and under the assumption that at most f Byzantine nodes may cooperate by sharing their SB^rs before committing EB^r, we know that no node has the content of DB^r prior to EB^r being committed-locally by at least one correct node. Thus, a party that wishes to predict I^r given a candidate *valid* EB^r (considering a non-valid Eblock is irrelevant as it would never be committed) needs to choose k committee members (denoted $u_1, \ldots, u_k$), and include their signed eses in EB^r. It then needs to predict the string $I^r = se_1 \oplus se_2 \oplus \cdots \oplus se_k$. We recall that $\langle ese_j \rangle_{\sigma_j}$ is the encrypted and signed version of se_j generated by u_j. Since there are at most f nodes controlled by the adversary, at least one of these eses was generated legally and without the access of the adversary. We recall that a legally-generated ese is the encryption (according to the threshold encryption scheme) of a locally (and secretly) generated string of bits, se, of which no bits can be predicted with probability higher than $\frac{1}{2}$ by any other node. In addition, assuming a secure threshold encryption scheme, se cannot be predicted from its ese. Finally, the unpredictability property of the XOR operation, in the calculation of I^r, with at least one unpredictable se concludes the proof.

□

We are now ready to prove claim 17.

Proof. Recall that node i's value in Cmap^r is defined as follows:

$$v_i^{r+1} := \frac{H(RS^r, r+1, pk_i)}{rep_i^{r+1}}$$

and that the m nodes with the minimal values are the committee members in term $r+1$. Since RS^r serves as random oracle, we conclude that the values $\{v_i^{r+1} \cdot rep_i^{r+1}\}_{i=1}^n$ induce a random ordering on the nodes. □

We draw the reader's attention to a nice advantage of the random seed in Helix, namely that RS^r is independent of RS^{r-1}. This is rather surprising as RS^r is constructed iteratively (or rather we are exposed to its values iteratively).

Fairness towards users

This section describes the problem of fairness towards users in a high-level and illustrates Helix's approach.

Nodes and users are very different entities in blockchain protocols. Users are the entities issuing the transactions but they cannot directly append them to the blockchain. Users depend on their nodes for *write* operations. Nodes proposing blocks can use this power in malicious ways—they can order transactions in a block according to their will, censor certain transactions, etc. Helix reduces nodes' power to manipulate the blockchain (to its own benefits) by minimizing the knowledge they have as to the semantics of the transactions they order. This can be obtained by separating the ordering service from the execution service. Indeed, Helix is only an ordering service, and as such is not concerned with the validity of the transactions it orders. Thus, all the information encoded in a transaction could be obfuscated from the nodes[22]. This is realized in Helix using two main mechanisms:

- A threshold encryption scheme that takes place on the user end, concealing the actual content of transactions from nodes. The content is revealed only after the inclusion of the transaction in a block is finalized and its location in the blockchain can no longer be altered.
- etxs are forwarded in Helix in a manner that maximizes the sender's anonymity such that nodes are not aware who is the owner node of a transaction they receive.

We now illustrate how reducing nodes' information regarding transactions plays a role in Helix's resilience to a few common attacks against users:

- Payment censorship. In this attack, any transaction aimed to "pay" a certain user is denied and not processed. This attack is impossible in Helix as the nodes have no clue who is the payee/s in an Encypted-transaction[23].
- Service censorship. In this attack, any transaction issued by a certain user is denied service and never added to the blockchain. In case an owner node can identify an etx issuer, it can refuse to propagate it or add it to Eblocks it proposes. This problem can be mitigated quite simply by allowing users to have multiple owner nodes, forwarding etxs to all of them[24].
- Ordering manipulation. In this attack, later transactions are processed before earlier ones. This can be considered an attack only if the system should be *order-fair* in some sense. Bitcoin for example is not order-fair in any sense, and as a matter of fact, this is considered one of Bitcoin's strengths—a high-priority later transaction can "bypass" earlier low-priority transactions by offering a higher fee. In a distributed system as Helix, the seemingly simple notion of tx_1 *is earlier then* tx_2 (denoted $tx_1 < tx_2$) is not easy to define. A proper example of such a definition is given in Hashgraph [4]. Hashgraph is shown to be order-fair in the sense that the ledger's order reflects the order by which transactions reached a certain fraction of the entire network. Helix is not fair in this sense, but suggests another concept of fairness in terms of ordering transactions. Intuitively, if an entity orders transactions it cannot relate any semantics to, there is nothing it can do other then ordering them randomly. In this spirit, we refer to an ordering of two transactions tx_1 and tx_2 as fair, if at the time of ordering, the ordering entity could not have determined which order is more *profitable* for it. Helix is order-fair in this sense when the more profitable for order - $tx_1 < tx_2$ or $tx_2 < tx_1$ - depends on the content of both tx_1 and tx_2. Assuming at least one of these transactions was issued by a user, the ordering node is familiar only with the corresponding Encrypted-transaction and cannot extract any useful information from it. Thus, in this scenario, we conclude that the order of transactions Helix produces is fair. The best example to illustrate Helix's resilience to order manipulation in this regard is front-running[25]. In the next section we show that Helix is also fair in regards to another ordering manipulation where nodes service etxs owned by them prior to other nodes' etxs. This ordering manipulation does not depend on the content of transactions, but only on the ability to classify a transaction as owned by a specific node. It is crucial (and non-trivial) for Helix to be resilient to such manipulations under the incentive structure Helix assumes. We refer to this as *selection fairness*.

VI. Selection fairness via correlated sampling

In typical blockchain protocols, nodes (such as Bitcoin's miners) have the freedom of choosing which transactions to include in a block. Normally, transactions come along with a fee and nodes choose the highest paying transactions available. In Helix, fees do not play a role in etx selection, as they are obfuscated. Instead, as described in Sec. IV-D, nodes may prefer to include certain etxs over others (e.g., etxs owned by them). The main goal of this section is to introduce to Helix a mechanism for selecting etxs in a *fair* manner. Fairness in

[22]This is almost accurate—information regarding who is accountable for a transaction is needed. This will be discussed in a later work.

[23]During this article we tried to disregard any semantics attributed to transactions, this item is exceptional in that sense and assumes transactions have two sides—payers and payees.

[24]An alternative solution is to have another dedicated service: a transaction-forwarding service. For the time being, Helix handles, both, the forwarding and ordering services, although this might change in the future, helping align subtle issues with the incentive structure and the mentioned attack.

[25]Front-running is a form of market manipulation practice where an entity with non-public information exploits it to produce an unfair profit. Consider the following example, a user sends a transaction with an order to buy a large amount of shares of some stock. Since the buy order will drive the price of the stock up, the node receiving the transaction can push its own buy transaction beforehand, where it can buy the stock's shares before the price rises. Later, the node releases the user's transaction and sells his recently bought shares at a higher price.

this regard should be reflected by the blockchain, representing each node according to the proportion of etxs it owns[26].

An important aspect in the context of fair sampling is the relation between the rate at which etxs are issued and the rate at which they are appended to the blockchain. An epoch at which the *issuance rate* is smaller than the maximal *append rate* allowed by the system is relatively easy to analyze—all transactions are processed within a small time frame. A more involved case is when the issuance rate is (temporarily) greater than the append rate. During such epochs, there are many etxs pending to be added to the blockchain, and the decision of which etxs to include in Eblocks is subject to manipulation. A core aspect of selection fairness is to ensure that the average *waiting time* of an etx depends as much as possible on the number of pending transactions, rather than on its owner node or on any other characteristic.

In high-load epochs, we adapt a strategy of *correlated sampling* that restricts the freedom a primary has in selecting which etxs to include in its Eblock. Correlated sampling, considered in [8], [21], [22] (see also [5] for a review and optimality result) refers to the problem in which there are two players having different distributions over the same domain (in our case the set of issued etxs) and access to shared randomness (in our case the random seed). Each player wishes to output a single element, sampled according to its distribution while minimizing the probability that the outputs differ. In the work of [8] a MinHash strategy was used for performing correlated sampling for the case that the distributions are uniform over a subset of the domain (analogue to Epools in our setting). In [32] Rivest applied the MinHash strategy sequentially in order to sample many elements from the domain, possibly with repetitions. However, to the best of our knowledge the scenario of correlated sampling of many elements without repetitions has not been analyzed before.

In this section we describe the Eblock validation procedure applied in Helix in accordance to the correlated sampling scheme. We consider a few natural Eblock constructions and give a probabilistic analysis as to these constructions' probabilities to pass validation. We show that any strategy with substantial probability of passing validation remains close to the construction dictated by the correlated sampling scheme, and is thus restricted in its ability to manipulate the selected etxs. Throughout this section we restrict ourselves to high-load epochs by assuming that the total number of pending etxs is exactly Γb for some $\Gamma \geq 1$ (where b is the maximal number of etxs in an Eblock). Also, we focus on a specific term number r and analyze the construction and validation process of EB^r.

A. Eblock validation

The Helix correlated sampling scheme uses a hash function in order to serialize candidate etxs for the next Eblock. The hash function is initialized with a random seed to eliminate its predictability, yielding a common, random and unpredictable serialization of the etxs. Formally, we order the etxs according to the values $H(RS^{r-1}, etx)$, and refer to these value as the *hash values* of the etxs. We usually denote it by $H(etx)$ for brevity.

To assure unpredictability, we introduce the notion of *locking time*. We denote by EP_i^t node i's locked Epool at time t, which is a snapshot of the etxs in i's Epool at time t[27]. We denote by t'_i the time at which node i received DB^{r-1}, the Dblock at term $r-1$, and define the locking time of node i to be $t_i := t'_i - 2\Delta$.

In the validation procedure, nodes are asked to consider the serialized etxs in their Epool at locking time. The choice of t_i was made such that nodes cannot quickly tailor etxs with low hash values *after* revealing the random seed, but *before* the rest of the network does. Indeed, due to network latency it could be the case that RS^{r-1} is revealed to some node before the rest of the network—by at most 2Δ time. Without the notion of locking time nodes could have exploited this time to generate and forward tailored etxs, employing a bias to the sampling scheme. Restricting nodes to consider only etxs which were known to them prior to time $t'_i - 2\Delta$ prevents this kind of attack. From now on by writing EP_i we mean $EP_i^{t_i}$, unless stated otherwise.

Due to network latency and inherent properties of Helix, we cannot expect $EP_i^{t_i}$ and $EP_j^{t_j}$ to be identical, even if i and j are correct nodes. However, as locking times are similar, we may assume a measure of similarity between two correct Epools at locking times[28]. To model this similarity, we use a simple probabilistic model in which each etx in EP_i is in EP_j with probability at least α, for any two correct nodes i, j[29].

Upon receiving a proposed Eblock, EB_p, from primary p, correct committee member i validates it by performing the validation procedure presented in Sec. IV-C. In addition, it checks two conditions regarding the size of the proposed Eblock and the size of its overlap with a set of transactions calculated according to its own Epool.

We define $T_p := \max\{H(etx) | etx \in EB_p\}$ as the maximal hash of an etx in the proposed Eblock EB_p. Furthermore, we denote by $EB'_i := \{etx \in EP_i | H(etx) \leq T_p\}$ the set of etxs in EP_i with hash values lower than the bound T_p, and $b_i := \max\{|EB'_i|, b\}$.

We further denote by b' the size of $EB'_p := \{etx \in EP_p | H(etx) \leq T_p\}$. We say that EB_p was constructed as a (b, b')-construction. This illustrates the fact that EB_p was

[26] Under such a notion of fairness nodes may be encouraged to artificially produce many self-owned transactions in order to get more block real estate. Any fee-per-transaction mechanism would easily mitigate this kind of attack.

[27] This can be implemented by attaching a timestamp to each etx at the time it was received.

[28] Entities with an ability to predict network timing may conduct an attack that results in correct Epools being largely dissimilar. To mitigate such an attack, t_i can be modified to $t_i := t'_i - 2\Delta - \eta_i$, where η_i is locally sampled by i in the range $[0, \Delta]$. This would decouple the predictability of the network from the locking-time procedure.

[29] Observe that in practice it is hard to estimate α accurately. In order for the validation condition to be effective, α should be quite close to 1. This should be supported by the network.

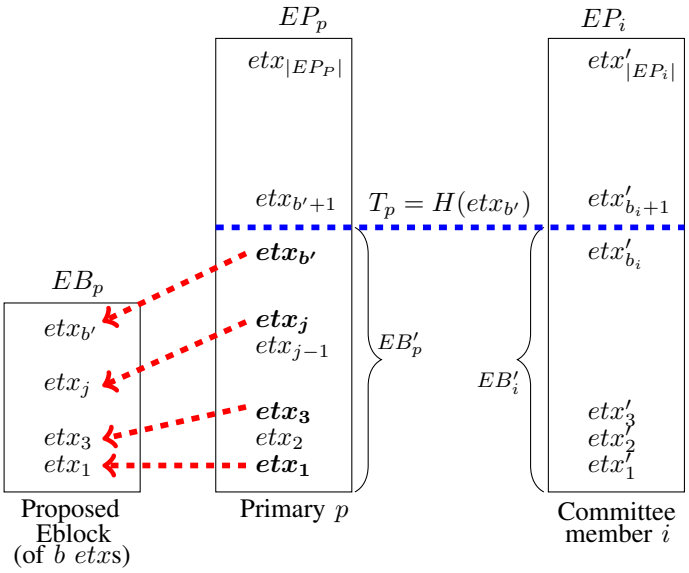


Figure 3: Illustration of the correlated sampling validation process. In each Epool, etxs are sorted based on their hash values. A threshold T_p is determined by the maximal hash value of an etx in the proposed EBlock EB_p as proposed by the primary p. A committee member examines the overlap between EB_p and the set of etxs in EP_i which hash below T_p.

selected as a subset of size b among the b' lowest hashed etxs in EP_p such that it includes the b'^{th} etx. The complete setting is illustrated in figure 3.

The examined conditions performed by a node in the process of validating EB_p are:

1) $|EB_p| = b$,
2) $|EB_p \cap EB'_i| \geq \beta(b_i)$ for $\beta(b_i) = \alpha b_i - \sqrt{10b_i}$.

The second validation condition encourages primaries to construct Eblocks with low b'. The minimal value of b' is b, which leads to a protocol that is perfectly selection fair. Intuitively, a larger b' allows the primary more freedom in the selection of EB_p (rather than selecting it as the b minimal etxs). However, since we can expect $|EB'_p| \approx |EB'_i|$, large b' yields a large b_i, reducing the chances of EB_p to pass validation. $\beta(b_i)$ is the maximal value for which Eblocks satisfying $b' = b$ pass validation with overwhelming probability as implied by Hoeffding's bound. This is shown formally in the next section.

B. Liveness under (b, b)-construction

We prove that an Eblock compliant with the (b, b)-construction passes validation of a correct committee member w.o.p..

Claim 19 (Liveness under (b, b)-construction) *Let EB_p be an Eblock constructed according to the (b, b)-construction, and let i be a correct committee member. Then $|EB_p \cap EB'_i| \geq \alpha b_i - \sqrt{10b_i}$ w.o.p..*

Proof. There are two cases to consider, depending on whether $|EB'_i| \leq b$ or vice versa. We start with the first case, in which $b_i = b = |EB_p|$. We need to show that if EB_p was constructed using the (b, b)-construction, then $|EB_p \cap EB'_i| > \alpha b - \sqrt{10b}$ holds w.o.p..

For each $etx \in EB_p$, we define Y_{etx} to be the indicator random variable stating whether $etx \in EP_i$, namely

$$Y_{etx} = \begin{cases} 1 & \text{if } etx \in EP_i \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

We further define $Y := \sum_{etx \in EB_p} Y_{etx}$. Notice two things: first, by definition of α, these random variables are i.i.d. Bernoulli variables, with success probability of at least α. This is true as each $etx \in EB_p$ lies in EP_i with probability (at least) α. Second, $Y = |EB_p \cap EP_i| = |EB_p \cap EB'_i|$. The second equality holds because any $etx \in EB_p$ satisfies $H(etx) \leq T_p$, and $EB'_i = \{etx \in EP_i | H(etx) \leq T_p\}$. Using Hoeffding's inequality, it satisfies for $\delta \geq 0$

$$Pr\left(\frac{1}{b}|Y - \mathbb{E}(Y)| > \delta\right) \leq 2\exp(-2b\delta^2).$$

Consider an overwhelming probability of at least $1 - 2exp(-20)$. Bounding the above probability by $2exp(-20)$, we derive an inequality $2b\delta^2 \geq 20 \iff \delta \geq \sqrt{\frac{10}{b}}$. The expected value of Y is bounded from below by αb. Taking $\delta = \sqrt{\frac{10}{b}}$ and plugging this into the left-hand side of Hoeffding's inequality, we get that w.o.p.

$$\begin{aligned} |EB_p \cap EB'_i| = Y &> \mathbb{E}(Y) - b\delta \\ &\geq \alpha \cdot b - \sqrt{10b} = \beta(b). \end{aligned} \tag{3}$$

In the second case $|EB'_i| > b$, and we have to show $|EB_p \cap EB'_i| > \beta(|EB'_i|)$. From the assumption that EB_p is a (b, b)-construction, $EB_p \cap EB'_i = EP_p \cap EB'_i$. With this observation at hand, one can present a similar proof, only this time define Y_{etx} for all $etx \in EB'_i$ to be the random variable indicating whether $etx \in EP_p$ or not. Then, $Y = \sum_{etx} Y_{etx} = |EB_p \cap EB'_i|$ is again the sum of i.i.d. Bernoulli variables. This time there are $|EB'_i|$ variables, all with parameter α. Using Hoeffding's inequality as before completes the proof. □

The claim establishes the fact that a primary following the (b, b)-construction would pass validation w.o.p., thus keeping the protocol strongly live.

In the next section we show to what extent is Helix selection fair. Generalizing the analysis above, we consider Eblocks admitting (b, b')-constructions for $b' > b$, and analyze their probability of passing validation. We then evaluate how these constructions effect fairness in terms of the probability an etx has to be included in EB_p.

C. Selection fairness

In case all nodes follow the (b, b)-construction (namely Eblocks include the b etxs with minimal hash values), selection fairness is an immediate result—an etx is included in the

next Eblock with probability $\frac{b}{\Gamma b} = \frac{1}{\Gamma}$. However, as nodes are assumed to prioritize their own etxs and act to shorten their average waiting time, they might choose to follow a different strategy. Due to the inevitable "room for error" permitted in the validation process (due to potentially non-identical Epools), such behavior cannot be completely eliminated. In this section we analyze the extent to which the validation scheme suggested above can guarantee selection fairness and a uniform average waiting time.

We consider two distinct methods to manipulate the selection process. The first is by *tailoring* etxs in order to hash them to low values. Precisely, once the random seed was revealed, the primary tailors prioritized etxs such that they have low hash values, keeping the maximal hash in the Eblock, T_p, low. The second method is through a (b, b')-construction, for some $b' > b$. We study the dependence between b' and an Eblock's certainty to pass validation. We deduce analytical as well as experimental results as to the effect these constructions have on the probability of an etx to get included in EB_p.

We start with a high-level analysis of the tailoring approach[30]. A primary either tailors an etx that was already forwarded to the network or an etx that it is hiding from the network. In the first case, any fee-per-transaction mechanism would significantly reduce the profitability of such behavior, even if successful, as it would imply that the same transaction is included in the blockchain twice (under two slightly different forms) and thus its cost is doubled. In the second case, the cost is not doubled, but the probability of a hidden etx to be included in the next Eblock is lower than that of a non-tailored etx which was forwarded to the network. The former probability is $\frac{1}{rep_p}$ (which is the probability of the owner node to be elected primary), whereas the latter probability is $\frac{b}{|EP_p|}$ (which is approximately the probability of forwarded etxs to be included in the next Eblock, as we show later). We deduce that hiding etxs is not a worthwhile strategy.

The Helix Eblock validation process in high-load epochs prevents over exploiting tailored etxs. Specifically, since tailored etxs are not in EP_i, including many of them reduces $|EB_p \cap EB'_i|$, while in order to pass validation, the size of this intersection must be at least $\beta(b)$. The exact analysis is simple and trivially allows at most $b - \beta(b)$ etxs to play with, in order to still have non-zero probability to pass. In practice, in order to have substantial probability of getting its Eblock committed, a primary has to include a smaller amount of tweaked etxs.

We continue with a formal analysis of the (b, b')-construction. According to this strategy, the primary first looks at EB'_p which contains the lowest b' etxs in EP_p. Of those, it selects the b etxs it prefers. Our analysis ignores this choice and is applicable to any subset of b etxs from EB'_p (that includes the b'^{th} etx). Typically, when $b' \geq b$ the validation condition becomes $|EB_p \cap EB'_i| \geq \beta(|EB'_i|)$. Both sides of this inequality grow with b', but while the left-hand side grows slowly, and is bounded by b, the right-hand side grows faster and is effectively unbounded. Thus, a large b' decreases the probability of EB_p to pass validation. A quantitative analysis is given hereafter.

We fix some $b' > b$ and consider EB_p constructed via the (b, b')-construction. We calculate the probability $q = q(b')$ in which EB_p passes a single validation. Denote this event by $A = A(b')$.

Before continuing, we make a simplifying assumption. Clearly, $|EB_p \cap EB'_i|$ is bounded from above by b. We assume that it is exactly λb, where $0 \leq \lambda \leq 1$. Realistically, λ is close to α as the probability of an $etx \in EP_p$ to be in EP_i is α. Thus, taking $\lambda = \alpha$ yields an average case analysis. Taking $\lambda = 1$, on the other hand, results in an analysis that facilitates the validation process (for the primary) and can be seen as a worst case analysis in terms of selection fairness. Under this assumption, $Pr(A)$ becomes $Pr(\lambda b \geq \beta(b_i))$. Calculating for which b_i this condition holds gives rise to a quadratic inequality, whose solution yields $Pr(A) = Pr(b_i \leq M)$ where $M := \left(\frac{\sqrt{10}+\sqrt{10+4\alpha\lambda b}}{2\alpha}\right)^2$. Clearly, this M satisfies $\lambda b = \beta(M)$.

We now turn to calculate $Pr(b_i \leq M)$ under the assumption that $\lambda = \alpha$. Under this assumption $M > b$, and since, by definition, $b_i = max\{b, |EB'_i|\}$, we have $Pr(b_i \leq M) = Pr(|EB'_i| \leq M)$.

Since it is unfeasible to compute $Pr(|EB'_i| \leq M)$, we use the Chernoff inequality to bound it from below. For this, we will use the expected value of $|EB'_i|$, which can be deduced using a simple symmetry argument, yielding $|EB'_i| = |EB'_p| = b'$. Our goal now is to find, for any desired probability to pass validation q[31], the maximal b' for which we can guarantee $Pr(|EB'_i| \leq M) \geq q$. The Chernoff inequality (for the complement event) yields

$$Pr(|EB'_i| > M) = Pr\big(|EB'_i| > (1+\delta)b'\big) \leq \exp\left(-\frac{\delta^2 b'}{3}\right),$$

where $\delta = \frac{M}{b'} - 1$. Chernoff requires $\delta > 0$, which holds when $b' \leq M$. Indeed, the range we are interested in is $b \leq b' < M$. We are thus led to compute the maximal $b' \leq M$ for which $f(b') := \exp(-\frac{\delta^2 b'}{3}) = 1 - q$ (in the range $b' \leq M$, the function f is monotonically increasing).

Table I illustrates maximal b' values for various passing validation probabilities q and various Eblock sizes b, while keeping $\alpha = 0.95$ and $\lambda = \alpha$ fixed. Notice that as q increases, the primary has to be more careful and b' decreases to ensure passing the validation with probability q. The fact that $b'(q)/b$ is very close to 1 indicates that the freedom a primary has to avoid fairness in constructing EB_p is rather limited.

To conclude this section, we analyze to what extent this extra freedom allows a primary to promote its prioritized etxs.

[30] We note that on a practical level this strategy is problematic. First, tweaking etxs costs money and takes time—the primary might lose its PBFT view before completing the task. Second, most etxs that are of value to an owner node are encrypted by its users and thus the owner node itself cannot tweak them.

[31] We emphasize that q is the probability of passing a *single* validation. For an Eblock to be accepted, several validations are required, where the event of passing node i's validation is independent of that of node j.

Table I: For Eblock size b and a desired probability to pass a single validation q, we find the maximal $b'(q)$ for which we can guarantee that a $(b, b'(q))$-construction would pass validation with probability at least q. The fact that the $b'(q)/b$ values are close to 1 suggests fairness in etx selection.

b	M	q	$b'(q)$	$b'(q)/b$
1000	1111	0.95	1016	1.0154
1000	1111	0.75	1046	1.0450
1000	1111	0.5	1064	1.0639
1000	1111	0.1	1093	1.0924
2500	2673	0.95	2523	1.0089
2500	2673	0.75	2570	1.0278
2500	2673	0.5	2600	1.0397
2500	2673	0.1	2645	1.0576
5000	5241	0.95	5029	1.0056
5000	5241	0.75	5096	1.0190
5000	5241	0.5	5138	1.0275
5000	5241	0.1	5201	1.0400

Given b', a primary would first include all etxs (among the first b' ones) it wishes to promote, and then complete the Eblock space. The deviation of this construction from simply taking the minimal b etxs depends on the fraction of etxs a node chooses to promote (e.g., the fraction of transactions it owns). We denote this fraction by ξ_p, and turn to compute the probability of an etx to be included in an Eblock constructed according to the $\big(b, b'(q)\big)$-construction described above.

Lemma 20 *Let $0 < q < 1$ be some probability, and let $etx \in EP_p$. Suppose EB_p is constructed according the $\big(b, b'(q)\big)$-construction with ξ_p, the fraction of promoted etxs. The probability that an arbitrary etx is included in EB_p is*

$$Pr(etx \in EB_p) = \begin{cases} \frac{b'(q)}{|EP_p|} & \text{if } etx \text{ is preferred} \\ \frac{b-\xi_p \cdot b'(q)}{|EP_p| \cdot (1-\xi_p)} & \text{otherwise} \end{cases}.$$

Proof. First, all preferred etxs which are among the lowest $b'(q)$ etxs are included in EB_p. The probability of an etx to admit this condition is $\frac{b'(q)}{|EP_p|}$, which explains the first case. In case the transaction is not preferred by the primary, it would be included iff both following conditions hold: it is hashed among the lowest $b'(q)$ etxs; and it is selected by the primary among the remaining etxs. The probability to meet the first condition is again $\frac{b'(q)}{|EP_p|}$, whereas for the second condition we can only give an average case estimation. On average, there are $\xi_p \cdot b'(q)$ preferred etxs among the lowest $b'(q)$. We conclude that there are $b - \xi_p b'(q)$ slots left in EB_p. The probability to get selected for these slots is $\frac{b-\xi_p b'(q)}{b'(q)-\xi_p b'(q)}$. Multiplying these probabilities yields the result. □

Let's examine the numbers with a concrete example. Based on Table I, taking $b' = 1.1b$ seems more than reasonable. Plugging this in the formula from the lemma gives

$$Pr(etx \in EB_p | etx \text{ is not preferred}) \geq \frac{(1 - \xi_p \cdot 1.1)}{(1 - \xi_p)} \cdot \frac{b}{|EP_p|}.$$

As a result, even if the fraction of etxs the primary wishes to promote is quite large, say a fraction of $\xi_p = 0.3$[32], then $\frac{(1-0.3 \cdot 1.1)}{(1-0.3)} > 0.95$ and so

$$Pr(etx \in EB_p | etx \text{ is not preferred}) > 0.95 \cdot \frac{b}{|EP_p|}.$$

Recall that for the fair (b, b)-construction, $Pr(etx \in EB_p) = \frac{b}{|EP_p|}$.

We conclude that the validation process derived from the correlated sampling scheme can, to a large extent, guarantee fairness in the selection process of etxs to Eblocks. In the next section we present experimental results that reassure the theoretical results.

D. Experimental results

To examine the fairness qualities of Helix, we conduct experiments for various system parameters. Our open-source code is made available online [1]. Our experiments focus on the validation procedure in high-load epochs. In particular, we ignore users altogether and simply assume each node owns a random number of etxs such that the total number of distinct etxs in the network is $10b$ where b is the block size. Epools are constructed so that $|EP_p \cap EP_i| = \alpha \cdot 10b$, and a random portion ξ_p of etxs are assumed to be promoted by the primary. We focus on block size $b = 1000$ and examine the impact of the similarity parameter α.

Each Epool is ordered according to the SHA256 hash values of the etx (we do not use any random seed as the unpredictability of the ordering is redundant in this context). One of the nodes is arbitrarily elected to be primary, and for a given b' it constructs an Eblock using the (b, b')-construction. We examine its validation by an arbitrarily selected node following the validation process specified in Sec. VI-A. The results presented here are the average of 1000 experiments.

Table I above illustrated bounds $b'(q)$ such that a $(b, b'(q))$-construction is guaranteed to pass validation with probability q. Our experiments show that in practice, the b' values for which a (b, b')-construction passes a q-fraction of the validations are a bit higher than $b'(q)$. Fig. 4 compares these values with $b'(q)$. The fact that the practical b' for a given q is larger than the bound $b'(q)$ is not surprising, and is mainly attributed to the Chernoff inequality. Note that b' increases only slightly when smaller qs are considered, which implies that the preferred strategy would be to take b' corresponding to q very close to 1 (in the above setting taking $b' = 1100$).

Fig. 5 illustrates the effect of α on fairness. It clearly shows that larger α yields smaller $b'(q)$. Intuitively, large α values allow the validator to know more about the primary's Epool, yielding a strict validation condition (larger β), which results in enhanced fairness.

Fig. 6 illustrates the effect a (b, b')-construction has on an $etsx$'s average waiting time. It presents the ratio between

[32] A reasonable assumption is that ξ_p is the fraction of etxs that node p owns. In a network of several dozen nodes, we do not expect this fraction to exceed 0.1.

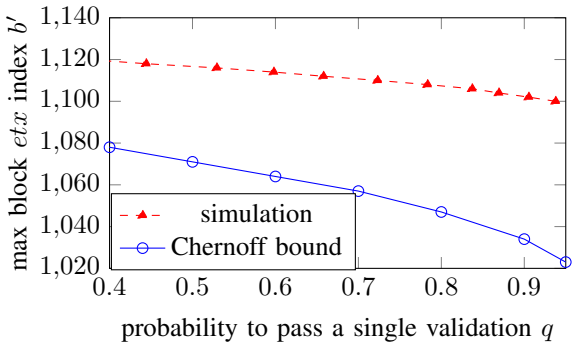


Figure 4: The maximum index b' of an etx included in the primary's proposed block vs. the required validation success probability.

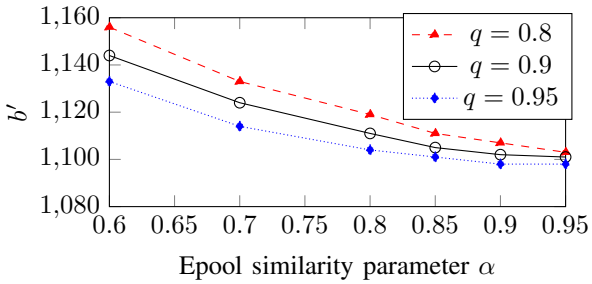


Figure 5: Impact of network similarity parameter on fairness: larger values of α enhance fairness.

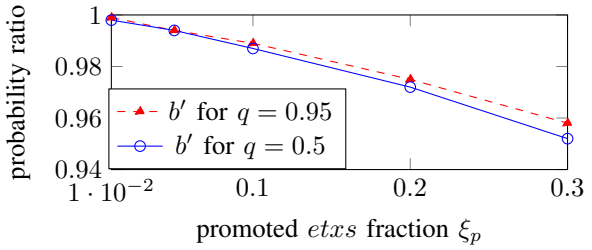


Figure 6: The ratio between the probability of a non-promoted etx to get included in a block constructed in a fair (b, b')-construction, and the probability of such an etx to get included in the fair (b, b)-construction. The curves refer to $\alpha = 0.95$ with various $b'(q)$ values.

the probability of a non-promoted etx to get included in a (b, b')-constructed Eblock, and that of an arbitrary etx to be included in a (b, b)-construction (these results rely on the formula obtained in Lemma 20). The fact that the ratios are close to 1 (even for low q values) demonstrates that in practice, Helix's selection fairness is maintained even under a $b' > b$ strategy.

VII. Fast Syncs

The following section illustrates an algorithm for syncing a node to the present *state of the system* (loosely speaking, the state of the system relates to the blockchain up to the term number of the most advanced correct node). It is desired to perform the syncing procedure as fast as possible, without sacrificing Helix's safety. Intuitively, having a quick syncing mechanism, reduces the effective time a node is considered faulty (sleepy) and thus, helps in reducing f. Reducing the number of faulty nodes enhances the performance of the system.

A straightforward way for a node (u) to sync is by sending some node (w) a request to sync that contains the highest term for which it has a Dblock (DB^r). If w has higher term committed Eblocks, it sends u tuples of the form (EB^q, BP^q, SB_1^q, ..., SB_k^q) for $q > r$ in an incrementing manner. Upon receiving the $(r+1)$-term tuple, u verifies BP^{r+1} serves as a valid Block-proof for EB^{r+1} under the $(r+1)$-term committee (as determined by DB^r). It then appends EB^{r+1} and reveals DB^{r+1} and the committee for term $r+2$ (using $SB_1^{r+1}, \ldots, SB_k^{r+1}$). This process is performed repeatedly until w completes sending all its blockchain. If w itself is *in-sync with the network*, then, when terminating, u is successfully synced.The downside of this method is that it might take a node a long time to sync (especially when syncing after a long downtime) during which the node is considered faulty and cannot participate in committees.

The Helix fast sync algorithm enables a node to actively participate while the previously described syncing procedure is taking place. A node u constantly listens to the network and learns the current term number, r, from the received messages. Upon receiving a message that contains a Block-proof and a matching Eblock (i.e., $\langle EB^r, BP^r \rangle$) u validates BP^r. Particularly, in the fast sync optimization, u drops the Cmap brief-validation check as defined in Sec. IV-C. If found valid, we claim that u is safe to append EB^r to its blockchain, even though it does not have DB^{r-1} and can not calculate locally Cmap^r (or the committee in term r) and relies on EB^r's header for it.

We emphasize that if u receives segment 4 messages (i.e., committee-related messages such as pre-prepare, prepare, etc.) it may assume that it is in the r-term PBFT committee but since it does not have the previous Dblock yet, it cannot reassure it and thus cannot participate. Thus, in this term, u still counts as faulty. In particular, if it is primary in this term, it would lose its turn.

We now turn to prove that Helix is safe when nodes apply the fast sync optimization.

Claim 21 *(Fast syncing safety) Let $u_1, \ldots, u_n$ be the nodes running Helix with the fast syncing optimization. In term r, let EB_i^r and EB_j^r be Eblocks appended by correct nodes u_i and u_j, respectively, to their local blockchains. Then, $EB_i^r = EB_j^r$.*

Proof. The proof is very similar to the original safety proof given in Sec. V-A. It relies on PBFT's safety which suffices as long as the committee in term r is a well-defined set of m nodes, which is agreed upon by all nodes. Let C^r be the committee that arises from DB^{r-1}. As in the original

safety proof, an inductive argument convinces that DB^{r-1} is agreed-upon and thus C^r is indeed well-defined. This is not enough though, as the fast sync optimization enables nodes to deduce the committee of term r from EB^r (rather than learn it locally from DB^{r-1}). However, for a correct node to deduce the committee of term r from EB^r, the latter must be accompanied by a valid Block-proof. Since correct nodes would contribute their signatures to a Block-proof only if they can calculate C^r locally and validate that it matches EB^r's Cmap, no Block-proof for a term-r Eblock that contradicts C^r can be generated. This concludes that C^r is indeed well-defined and PBFT's safety proof is now applicable. □

VIII. Conclusion

We presented Helix, a Byzantine fault tolerant consensus protocol for ordering transactions that ensures the resulting order was *fairly* determined. We believe that our work is highly suitable for ledger implementations in which transaction fees are not the sole motivation for processing transactions, such as decentralized ledgers whose nodes are operated by companies, each wishing to service its own users. Indeed, such use-case requires the protocol to achieve a fair ordering of transactions, where all participants enjoy an equal level of service (transaction confirmation time, throughput, etc.). Helixis providing this property while keeping the decentralized control of the ledger in the hands of the participating nodes; the capability to scale the transaction throughput; and a Byzantine fault tolerant engine. Helixassures its users enhanced protection from censorship and discrimination relative to other typical solutions, but only requires them to utilize a negligible amount of resources. It can therefore be seen as a fair protocol towards users.

IX. Acknowledgements

We would like to thanks Idit Keidar and Steven Goldfeder for their fruitful discussions and helpful suggestions.

REFERENCES

[1] Helix open-source implementation. https://github.com/orbs-network/consensus-fairness-simulation/, 2018.

[2] I. Abraham, G. Gueta, and D. Malkhi. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR*, abs/1803.05069, 2018.

[3] Y. Amir, B. A. Coan, J. Kirsch, and J. Lane. Prime: Byzantine replication under attack. *IEEE Trans. Dependable Sec. Comput.*, 8(4):564–577, 2011.

[4] L. Baird. The Swirlds Hashgraph consensus algorithm: Fair, fast, Byzantine fault tolerance. 2016.

[5] M. Bavarian, B. Ghazi, E. Haramaty, P. Kamath, R. L. Rivest, and M. Sudan. The optimality of correlated sampling. *CoRR*, abs/1612.01041, 2016.

[6] G. Bleumer. Random oracle model. *Encyclopedia of Cryptography and Security*, pages 1027–1028, 2011.

[7] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-group signature scheme. In *Workshop on Theory and Practice in Public Key Cryptography (PKC)*, 2003.

[8] A. Z. Broder. On the resemblance and containment of documents. IEEE Compression and Complexity of Sequences, 1997.

[9] E. Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains, Jun 2016.

[10] I. Cascudo and B. David. SCRAPE: Scalable randomness attested by public entities. In *Springer ACNS*, 2017.

[11] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[12] I. Damgård and M. Koprowski. Practical threshold RSA signatures without a trusted dealer. In *EUROCRYPT*, 2001.

[13] C. Decker and R. Wattenhofer. Information propagation in the Bitcoin network. In *IEEE Peer-to-Peer Computing*, 2013.

[14] Y. Desmedt. Threshold cryptography. *Encyclopedia of Cryptography and Security*, pages 1288–1293, 2011.

[15] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[16] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Springer CRYPTO*, 1992.

[17] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, 2014.

[18] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. In *ACM SIGACT-SIGMOD*, 1983.

[19] J. Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481, 1978.

[20] T. Hanke, M. Movahedi, and D. Williams. Dfinity technology overview series consensus system, January 2018.

[21] T. Holenstein. Parallel repetition: Simplifications and the no-signaling case. In *ACM Symposium on Theory of Computing*, 2007.

[22] J. M. Kleinberg and É. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *J. ACM*, 49(5):616–639, 2002.

[23] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *USENIX Security*, 2016.

[24] L. Lamport. Paxos made simple. *SIGACT News*, 32(4):18–25, 2001.

[25] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM*, 4(3):382–401, 1982.

[26] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena. Demystifying incentives in the consensus computer. *IACR Cryptology ePrint Archive*, 2015:702, 2015.

[27] S. Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[28] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In *ACM SIGSAC*, 2016.

[29] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[30] D. Ongaro and J. K. Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, 2014.

[31] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*, 2017.

[32] R. L. Rivest. Symmetric encryption via keyrings and ECC. 2016.

[33] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in Bitcoin. In *Springer Financial Cryptography and Data Security*, 2016.

[34] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.

[35] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in Bitcoin. In *Springer Financial Cryptography and Data Security*, 2015.

[36] 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 *IEEE Symposium on Security and Privacy*, 2016.

[37] The ZILLIQA team. The ZILLIQA technical whitepaper, 2017.

[38] S. B. Venkatakrishnan, G. C. Fanti, and P. Viswanath. Dandelion: Redesigning the Bitcoin network for anonymity. *POMACS*, 1(1):22:1–22:34, 2017.

APPENDIX A
COMMUNICATION

This section outlines broadcast communication abstractions, utilized in Helix, under the unique environment of a fully connected network. While by Section III nodes in the network are assumed to be fully connected, a forwarding scheme determines which of the links to use to optimize a relevant metric. We refer to the node who starts disseminating a message as the message originator. We describe two high level approaches for communication. The first aims to reduce dissemination time, while the second aims to maximize originator anonymity. We start with some notations and intuition.

A. Background

The design of a forwarding scheme can influence several performance metrics such as:

1) The dissemination time of a message.
2) The required fan-out of a node number, describing the maximal amount of message copies required to be sent by a node.
3) Resistance to faulty nodes which, for instance, unexpectedly do not forward messages.
4) Various privacy aspects, e.g., the anonymity of the message source.

Often, there is a tradeoff between these metrics, such that one can be improved only at the expense of some others.

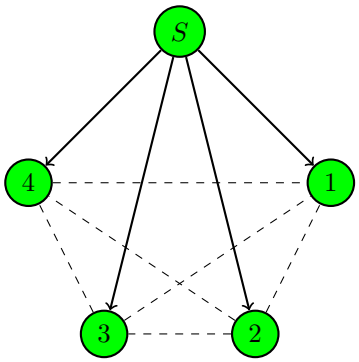
Consider for instance the basic forwarding schemes illustrated in Fig. 7 and assume all links share the same propagation delay. First, in Fig. 7(a), a message is sent from its source directly to all other nodes along their connecting link. Accordingly, all nodes receive the message within a single hop time, achieving an ideal dissemination time. On the other hand, when a node receives a message, it immediately knows the identity of its originator. This is the node from which the message was received. This implies a complete avoidance of originator anonymity.

In the scheme of Fig. 7(b), a message is forwarded as in a ring such that each node transmits messages only to a single node, the following node in some predetermined order of the nodes. Here, it might take a node several hops to receive a message where the exact time it takes is determined by its location with regards to the source location such that the maximal delay is $n-1$ hops for n connected nodes. Likewise, the scheme of Fig. 7(c) tries to balance these two metrics. A message is first sent to a common central node that forwards it to all other nodes. A message has a fixed delay of two hops and a node always receives a message from the central node without revealing an information regarding the originator. Note that here, the central node does know the originator identity. Moreover, operation highly relies on the assumption that the central node follows its expected behavior.

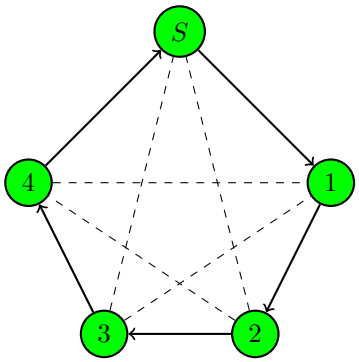
In the analysis of the described forwarding schemes we make use of parameters such as n, f and k. n describes the total number of network nodes forwarding and receiving messages. Among these at most f are Byzantine, and no node transmits a message to more than k of its neighbors. From bandwidth considerations we want to keep k small, while it is quite clear $k < f$ cannot allow dealing with f Byzantine nodes.

B. Minimizing worst-case dissemination time

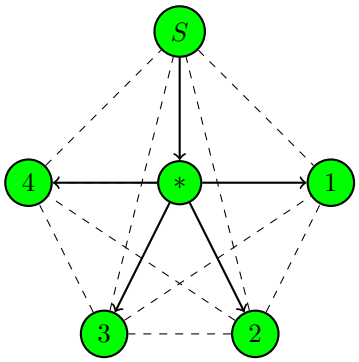
We focus here on the design of a forwarding scheme with *a small worst-case delay*. In that, we refer to the maximal time it takes to send a message between any pair of source and destination nodes. We maintain the settings of n nodes



(a) Qlique forwarding



(b) Ring forwarding



(c) Star forwarding

Figure 7: Illustration of various forwarding schemes. A message is transmitted from a source node S to all other nodes, either directly or through intermediate nodes.

with f Byzantines. Denote by T the worst-case message delay, as a function of the number of hops. Schemes are characterized by two parameters: the fan-out bound k and the obtained delay T. Intuitively, increasing the fan-out enables faster message dissemination. We describe forwarding schemes which illustrate this tradeoff. In the first scheme, the fan-out $k = O(\sqrt{n \cdot f})$ and the achieved delay is constant $T = 2$. In the second scheme $k = 2 \cdot (f+1)$ and the achieved delay is $T = O(\log(n/f))$.

Two-hop dissemination

We first describe a simple scheme with $T = 2$: In the first hop, the message is forwarded from the source node to $k \geq f+1$ nodes. Then, each of these nodes forwards the message to k nodes. We describe this forwarding through a binary matrix A of k rows and $n-k-1$ columns such that $A_{i,j} = 1$ only if a node i forwards a message to a node j among the $n-k-1$ other nodes. We can express the constraints based on values of the matrix A. This would allow us to determine the values of the matrix, the mutual required relationship of k, n and f and accordingly to derive a possible forwarding scheme. The demands from the matrix are expressed as follows.

Property 22 *The binary traffic matrix A has to satisfy the following constraints:*

1) *The sum of each line equals at most k, namely $\forall i \in [1,k], \sum_{j \in [1,n-k-1]} A_{i,j} \leq k$*
2) *The sum of each column equals at least $f+1$, namely $\forall j \in [1, n-k-1], \sum_{i \in [1,k]} A_{i,j} \geq f+1$*

While the first constraint follows the bound on the messages sent by a node, the second guarantees each nodes hears the message whenever at most f nodes are Byzantine. A possible selection of A has the following form illustrated for $f = 1$:

$$A = \begin{pmatrix} 1 & \dots & 1 & 0 & \dots & 0 & \dots & 0 & \dots & 0 \\ 1 & \dots & 1 & 0 & \dots & 0 & \dots & 0 & \dots & 0 \\ 0 & \dots & 0 & 1 & \dots & 1 & \dots & 0 & \dots & 0 \\ 0 & \dots & 0 & 1 & \dots & 1 & \dots & 0 & \dots & 0 \\ & & & & \dots\dots\dots\dots & & & & & \\ 0 & \dots & 0 & 0 & \dots & 0 & \dots & 1 & \dots & 1 \\ 0 & \dots & 0 & 0 & \dots & 0 & \dots & 1 & \dots & 1 \end{pmatrix}$$

The matrix A has $f+1$ copies of each distinct line. The first $f+1$ simply forwards traffic to the first k among the $n-k-1$ receiving nodes. The next $f+1$ to the following k nodes among the $n-k-1$ and so on. The constraint on the number of sent messages has to be satisfied for each node and in particular the source. This implies each of the $n-k-1$ nodes is guaranteed to receive the message when the number of lines k allow having at least $n-k-1/k$ groups, each of $f+1$ similar lines. Namely, the parameters satisfy $\lfloor \frac{k}{f+1} \rfloor \cdot k \geq n-k-1$, namely $k \geq (n-k)/\lfloor \frac{k}{f+1} \rfloor$.

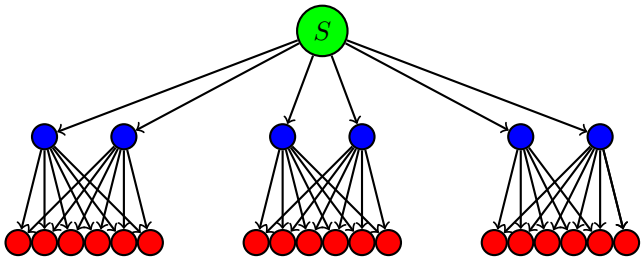


Figure 8: Illustration of the forwarding scheme optimizing the worst-case propagation delay for a network of $n = 25$ nodes with at most $f = 1$ Byzantine nodes and a maximal of $d = 6$ messages sent by a node and $k = d = 6$ nodes that serve as intermediate in the message dissemination process.

Fast Reliable Forwarding

We next generalize the above notion, specifically, we do not assume the node out degree k suffices to reach all network nodes in only two hops. We describe a forwarding scheme with a fixed out degree of $k = 2$ for a node that guarantees a dissemination time of $T = \log(n)$ where there are no Byzantine nodes. Following some number assignment for the nodes $0, \dots, n-1$ we let a node i forward traffic to exactly two nodes $2 \cdot i$ and $2 \cdot i + 1$. Along this subsection, all calculations of nodes indices are performed modulo n, we avoid repeating that for conciseness. A forwarding binary matrix (of size $n \times n$) can be derived accordingly with $2 \cdot n$ positive values. For a node i, we denote by $C_\ell(i)$, the set of nodes reached by node i after at most ℓ hops.

For the sake of simplicity, we assume the number of the network nodes n can be described as $n = 2^h$. We show that a message sent by an arbitrary node $i \in [0, n-1]$ would reach all n nodes $0, \dots n-1$ within $h = \log(n)$ hops.

Claim 23 *Every node $i \in [0, n-1]$ satisfies $C_h(i) = [0, n-1]$.*

Proof. The proof is by induction on the number of hops $j \in [0, h]$ that the set of reachable nodes after (at most) j hops $C_j(i)$ satisfies $S_{i,j} \subseteq C_j(i)$ for $S_{i,j} = \{(i \% 2^{h-j}) \cdot 2^j + z | z \in [0, 2^j - 1]\}$ of size $|S_{i,j}| = 2^j$, where $\%$ denotes the modulo operation. Then, the claim follows for the correctness for $j = h$, since the only set of nodes of size 2^h is $[0, n-1]$. As the induction basis, we have for $j = 0$ that $C_j(i) = \{i\}$ and $S_{i,0} = \{i\}$ is of size $2^j = 1$ and satisfies $S_{i,0} \subseteq C_j(i)$. As the induction step for $j \geq 1$, we show that every node in $S_{i,j}$ is reachable within one hop from some node in $S_{i,j-1}$. Let $x = (i \% 2^{h-j}) \cdot 2^j + z_x$ be a node in $S_{i,j}$. We first assume that x is even. It immediately follows that z_x is even. Then, for $y = (i \% 2^{h-(j-1)}) \cdot 2^{j-1} + z_x/2$ it satisfies $2 \cdot y = 2 \cdot ((i \% 2^{h-(j-1)}) \cdot 2^{j-1} + z_x/2) = 2 \cdot (i \% 2^{h-(j-1)}) \cdot 2^{j-1} + z_x = 2 \cdot (i \% 2^{h-j} + a \cdot 2^{h-j}) \cdot 2^{j-1} + z_x$, where the last equality holds

for some $a \in \{0,1\}$. Then, again, with calculations performed modulo n (the number of nodes), we have $2 \cdot y = 2 \cdot (i \% 2^{h-j} + a \cdot 2^{h-j}) \cdot 2^{j-1} + z_x = 2 \cdot (i \% 2^{h-j}) \cdot 2^{j-1} + 2 \cdot a \cdot 2^{h-j} \cdot 2^{j-1} + z_x = 2 \cdot (i \% 2^{h-j}) \cdot 2^{j-1} + a \cdot 2^h + z_x = (i \% 2^{h-j}) \cdot 2^j + z_x = x$. Thus x is reachable within one hop from $y \in S_{i,j-1}$. Similarly, if x is odd then $x-1$ is even and in $S_{i,j}$ (since, z_x is odd). Let $y \in S_{i,j-1}$ be a node from which $x-1$ is reachable. By the forwarding properties, x is also reachable within one hop from y. □

To withstand f Byzantine nodes, we add an abstraction on top of the described scheme. We assume here that a Byzantine node can only avoid forwarding messages (rather than modifying them). Let $f' = 2^{\lceil log_2(f+1) \rceil}$ be the minimal power of two that equals at least $f+1$. We divide the n nodes into groups of f' nodes, mapping each group to a vertex in the tree.

When $n = 2^h$ we have a number of $2^{h-f'}$ groups. We apply the forwarding scheme in terms of the groups. Namely, a node forwards the message to all $2 \cdot f'$ nodes, belonging to the two groups it should forward to, following the forwarding scheme.

We state this forwarding scheme holds the following property:

Claim 24 *If a correct node sends a message according to the above scheme having a node out degree of $k = 2 \cdot f'$, all nodes in the network receive the message in at most $T = log(n/(f')) \leq log(n/(f))$ hops.*

Proof. We explain why even with the existence of up to f Byzantine, a message sent by an arbitrary node is received by all groups (including the group for which the node belongs). Consider the group to which the source node belongs. By properties of the scheme, all groups appear in a distance of $\log(n/(f))$ from this group, similarly to the proof of Claim 23. Consider for $i \in [1, \log(n/(f))]$ the 2^i groups that appear in a distance of i following the scheme. We refer to them as groups of level i (A group can appear in more than a single level). By a simple induction, for all i, these 2^i groups receive the message. A group in some level does not receive a message only if it was not forwarded by all f' nodes in the group it should have receive the message from based on the scheme. Since $f' > f$, this cannot happen and all groups receive the message and accordingly all n nodes. The number of hops is given as the logarithm of the number of groups. □

This approach involves a potential large amount of redundancy in the number of sent messages. A message can be sent f' times between two groups. One way to avoid that is to limit members of a group to send messages incrementally. Following some arbitrary order of nodes in a group, a node sends a message only when it is clear that the message sent by a previous node was not received by at least one of the next group members.

C. Maximizing originator anonymity

We examine here, the challenges in designing a communication protocol with strong originator anonymity in a network composed of well-identified nodes.

Originator anonymity can be measured in various ways. A typical metric is the *anonymity set size*, denoting amongst how many nodes the message originator is hidden, namely the number of nodes that can potentially be the originator of a message. These are the nodes that cannot be eliminated from being the originator following information held by the receiver of the message. An alternative metric is the *originator entropy* considering the probability for a node in the anonymity set to be the originator. The metric examines the level of uncertainty for the originator identity in terms of Information theory. Intuitively, for a set of possible originators, the level of uncertainty is larger, when their probabilities to be the originator are uniform, than in the case in which some nodes have larger probabilities than others.

We say that a forwarding scheme maintains originator anonymity, if for any originator the *anonymity set*, calculated by some destination node, is the set of $n-1$ other nodes. A ring based scheme, maintains originator anonymity when all nodes are correct. In a ring forwarding, a node forwards a message to the successive node, according to some order of the nodes. It is illustrated in Fig. 7(b). For every node, there is a single node it receives messages from. Accordingly, no information regarding the source of the message is revealed to a node from the identity of the node it got a message from. Using the ring forwarding is not survivable to the existence of Byzantine nodes. The existence of even a single Byzantine node can eliminate the opportunity of some nodes to receive a message.

In a realistic adversarial model, we have to account for the presence of colluding nodes. These *spy* nodes, disguised as honest, can exploit information about the message propagation - i.e., timestamps, sending neighbors - with the used topology, to break originator anonymity. An example of a simple de-anonymization attack, called *first-spy*, outputs, as the originator, the first honest node to send the message to any of the adversarial nodes. In a structured ring scheme, the colluding nodes can deduce a set of potential originators, of size smaller than $n-1$. If, for example, the f Byzantine nodes are distributed evenly along the ring, the expected anonymity set size is $\frac{n}{f} - 1$.

De-anonymization attacks, exploit available information such as the network topology or the network protocol and even side channels traffic. An intuitive strategy, would therefore try to break persistent patterns of information, which eventually harm the system's privacy qualities The use of a dynamic topology, changed periodically, might then account as a useful heuristic towards obtaining stronger anonymity merits. Simply changing the topology, globally, while informing all participating nodes, should not be considered as a good strategy, since the adversary is actively taking part in the system. To achieve on the fly dynamic topology, we could base our solution

on a technique for anonymous communication called *Onion routing*. The idea, is to encapsulate a message in layers of encryption. Each node, after decrypting its own layer, forwards the message, according to the next encrypted layer's public key. Similarly, in our approach, the originator of the message could devise a dynamic topology, locally, where the routing path is induced by the successive layers of encryption.

Another powerful de-anonymization attack the adversary might exploit, is side channels; providing useful information about participants, such as traffic patterns. This potential anonymity breach, is even more prevalent in an environment where the participants' identities are known to each other. A client's whose mass users are located in North America, will manifest different network traffic, than a client's whose users are mainly located in Asia. Generally, the adversary could construct a typical user profile for each known participant. This information, in turn could increase or decrease the likelihood of being the originator of a message. A common countermeasure against such attack, is the use of dummy traffic. However, trading throughput for privacy, might not be a reasonable solution in our network.

Finally, we address scale in terms of latency. Some anonymity schemes might introduce prohibitive inconsistency amongst nodes, and thus, will not scale well. An interesting approach to overcome this difficulty, suggested in [38], is to break the forwarding scheme to two phases: a short anonymity phase, followed by a fast broadcasting, such as diffusion; Where the originator multicasts a message to a group, in varying random delays, to increase the uncertainty in the adversary's timing estimates. This solution while providing for good originator anonymity also maintains, reasonable propagation time, steering clear of the aforementioned problem.