Helix: A Scalable and Fair Consensus Algorithm Resistant to Ordering Manipulation

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

Orbs Research (orbs.com)

***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, limiting censorship and front-running. 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 promoting 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 [1] was first to realize a permissionless membership model, mitigating Sybil attacks and reducing communication complexity by integrating the ingenious Proof-of-Work [2] (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 [3].

Variants of the original PoW-based blockchain protocol try to improve certain aspects of it, while maintaining PoW as a key factor. Byzcoin [4] offers finality in a PoW-based blockchain by introducing a PBFT [5] 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 [6], which offers an alternative to the longest chain rule whose resilience does not deteriorate when the block rate is high; and SPECTRE [7], 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 [8], 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 [5] 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 [10], 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 semantics to the transactions it arranges and is completely unaware of state changes these transactions may invoke[1]. It is therefore possible to use threshold encryption to obfuscate the transactions from the nodes. Once the position of a transaction becomes final, its content is revealed.

The threshold encryption scheme is further used to produce verifiable randomness within the protocol. Helix realizes a randomness beacon without requiring additional rounds of communication[2]. It then makes use of the randomness in two respects: first, to elect leaders (and committees, which are used to address scalability concerns, as PBFT does not scale adequately to large networks) in an unpredictable and non-manipulable manner; and second, to realize correlated sampling [13], which is used to force nodes to randomly sample their pool of pending transactions when constructing blocks. The correlated sampling scheme enables new fee models (e.g., constant fees) that differ from traditional approaches and prevent the emergence of undesired fee markets.

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 she favors and include them in the block she 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[3] 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 a protocol, executed locally by each entity, which ensures all entities output the same value. Existing agreement protocol are designed with various execution environments in mind, or possible behaviors by entities, and result in different performance characteristics.

The properties of a distributed protocol are affected by the quality of the underlying network communication. A few types of synchronous environments we refer to throughout the paper, taken from [14] 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.

[1]The execution and validation of transactions is beyond the scope of this paper. We mention though, that under the abstraction of a well-defined order (obtained by Helix), invalid transactions can simply be skipped by the execution service.

[2]We note that the problem of using multi-party computation to produce verifiable randomness has a long history both in and apart from the context of blockchains [11], [8], [12].

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

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 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 [15]. 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 [16], a modern Paxos variant, explicitly defines a timeout scheme that achieves liveness when communication is synchronous and reliable.

Fischer, Lynch and Paterson [17] showed that a deterministic agreement protocol in an asynchronous network can not guarantee liveness if one entity may crash, even when links are assumed to be reliable[4]. 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 variant of the agreement problem assumes a more involved failure model, where nodes may act maliciously in addition to crashing. A Byzantine entity does not follow the protocol's instructions and behaves arbitrarily. This problem is often called *Byzantine agreement* (BA) and was introduced by Lamport, Shostak and Pease [19]. A well known result in the field of distributed systems is that in an asynchronous network, agreement cannot be reached unless less than $1/3$ 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) of values, 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 of values. However, a version of Paxos (sometimes called Multi-Paxos or multi-decree Paxos) optimizes 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) [5] which was the first efficient protocol to implement agreement on a sequence of values in the presence of (less than a third) Byzantine entities, employing similar type of optimization.

We further formulate a primitive that satisfies *agreement on a sequence of values* written to a log, through the previous requirements, together with an additional requirement. In agreement on a sequence of values, 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 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.

A different approach to reach agreement on a sequence of values is referred to as Nakamoto consensus [1]. The fundamental data structure that underlies the consensus 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 multiple values (blocks) to exist 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, she selects the longest one as the valid chain (more accurately, she selects the branch that admits to the most amount of accumulative work according to the PoW principle [2], which we do not explain in this paper). Over time, this rule yields 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 eventual consistency of the Nakamoto consensus highly depends on the network being strongly synchronous, where

[4]When links are unreliable reaching an agreement is impossible even in a strongly synchronous network, a result known as the two generals problem [18].

new blocks propagate to the network in negligible time relative to the average block creation rate [20]. In case this assumption does not hold, the resilience of the longest chain rule may become fragile, facilitating an attacker to successfully re-organize the valid chain[5].

B. Cryptographic primitives

Threshold Cryptography

Threshold cryptography [21] refers broadly to techniques for allowing only *joint groups* of parties to use a cryptosystem, be it to compute signatures, or to decrypt data. In particular, a (t, n)-threshold encryption cryptosystem is executed by n entities, any of which can encrypt messages, while the cooperation of $t+1$ $\big($for some fixed $t \in \{1, \ldots, n-1\}\big)$ is necessary in order to decrypt messages successfully. Threshold security guarantees that any attempt by up to t of the entities to decrypt a ciphertext is bound to fail. The (t, 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 common public key PK, secret keys $S_0, \ldots, S_{n-1}$ and verification keys $V_0, \ldots, V_{n-1}$.
2) A non-malleable encryption scheme[6] [22] which uses PK for encryption.
3) A threshold decryption scheme which consists of two parts. In the first part, individual (and secret) key shares obtained by each entity are used in order to produce decrypted shares (from the ciphertext). The second part allows joining $t+1$ decrypted shares in order to retrieve the original message. A non-malleable threshold encryption scheme must include two local verification steps. First, to validate the ciphertext before producing a share, and second, to validate the shares before joining them.

We note that one appropriate choice for a such a threshold encryption cryptosystem is [23].

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 [24] as is often assumed in the literature.

[5]The GHOST [6] protocol suggests the "heaviest subtree" rule to determine the valid chain, partially mitigating this problem.

[6]A non-malleable encryption scheme guarantees that it is impossible to transform a given ciphertext, of a plaintext m, into another ciphertext which decrypts to a related plaintext, $f(m)$ (for a known function f).

Digital Signatures

Digital signature schemes enable authenticating a message by verifying it was created by a certain entity. Digital signatures usually require a secret key, denoted by sk_i, used by entity i for signing a message, and a public key, denoted by pk_i, used for verifying the signature of entity i. To prevent anyone else from signing on her behalf, an entity should not share her secret key. Any entity with knowledge of the public key can verify the signature. A digital signature scheme typically consists of three algorithms: a key generation algorithm, a signing algorithm, and a verification algorithm. In order to produce a new pair of keys, sk and pk, the generation algorithm is used. The binary string $\sigma_i(m)$ is referred to as the digital signature of entity i over a message m and is produced using the signing algorithm and sk_i.

The main properties required from a digital signature scheme are:

1) Legitimate signatures pass verification. $\sigma_i(m)$ passes the verification algorithm 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 algorithm, for a message m that was never signed by i beforehand, even to an adversary that is allowed to view signatures under sk_i (for other messages).

We denote by $\langle m \rangle_{\sigma_i}$ the signature message which contains the message m, the identity of the signer pk_i and the signature $\sigma_i(m)$.

III. Model

In this section we introduce the model assumptions that the Helix protocol relies on to achieve safety, liveness and fairness. 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

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

Nodes. The nodes participating in the protocol form a fixed set of ids (public keys) known to all. 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.

Users. 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 as a virtual fragment of the node they are connected to.

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.

Node-node connections are assumed to be strongly synchronous[7], 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[8].

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[9] $3f + 1 \leq n$. 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 she 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 she can choose different faulty nodes. In a specific term, the adversary is restricted to sign only on behalf of the nodes she controls in that term[10]. The adversary controls all the faulty nodes and coordinates their behavior. In addition, she delivers messages between the faulty nodes instantaneously. Hence, the adversary may perfectly coordinate attacks she 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[11]. 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 fully-connected Helix network with eight nodes (illustrated by black large circles), serving various numbers of users (appearing as small red circles).

D. Communication schemes

The protocol uses two high level approaches for communication. The first aims to reduce message 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 she sends messages to.

The *fast broadcast* scheme is deterministic (namely, not using random gossip), implying a fixed bound for the time a message is propagated in the network. The scheme is resilient to f Byzantine nodes that do not follow the scheme and may act arbitrarily. Let δ be the maximal time it takes for point-to-point messages to arrive at their destinations and let Δ be the maximal time it takes a message that a correct node broadcasts, to reach all correct nodes. By the scheme a node multicasts a message to $2(f + 1)$ neighbors and a message is propagated to the whole network within $\zeta = O(\log(n/f))$ hops such that $\Delta = \zeta \cdot \delta$.

When anonymity is of interest, a second approach is used. The *anonymous broadcast*, generalizes a simple ring topology and uses different heuristics to reach destined nodes while not revealing much information on the identity of the source node.

[7] 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.

[8] 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 are more strict.

[9] We further limit the adversary in the amount of resources she 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 analogous to the common restriction of Byzantine hash power in PoW chains or stake in PoS chains.

[10] 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, she cannot sign messages from an earlier term.

[11] 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.

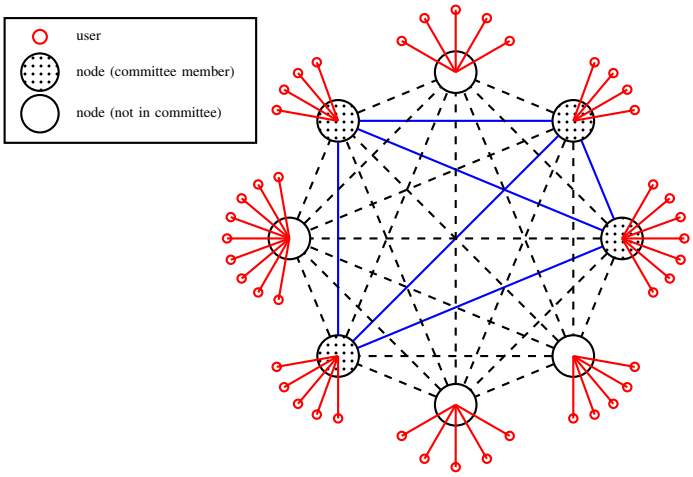

Figure 1: Illustration of a Helix fully mesh network with $n = 8$ nodes, serving various numbers of users. Among the nodes, $m = 4$ nodes (filled with dots) participate in the committee of some term r.

IV. The Helix protocol

This section is dedicated to a high-level description of the Helix protocol—we first illustrate the main data types used in Helix and then describe its operation.

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 her **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 (see Fig. 1). 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 (t, n)-threshold encryption scheme among the nodes, where $t = f$, 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 her own **Shares-block**, and later $t + 1$ (for a further explanation see Sec. IV-C) different Shares-blocks are later combined to reveal the **Dblock**, block of decrypted etxs (corresponding to the Eblock).

Each node incorporates a so-called **Encrypted-secret** into the Shares-block that she 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 appending a new Eblock to the blockchain in the next term. Intuitively, the reputation is a measure of node's behavior in the protocol.

A. The Helix data types

Transaction (tx) and **Encrypted-transaction** (etx). Transactions are the atomic pieces of information[12] that Helix communicates and orders. Users issue transactions and sign them to prove their system-identities. They then employ a threshold encryption scheme[13], where only $t + 1$ 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 fixed-length bit string sampled uniformly at random periodically by each node, and used to implement Helix's randomness beacon. As their name suggests, the Encrypted-secrets are the ciphertexts corresponding to the secrets[14]. Each ese is propagated along the network after being associated with a term number r and signed by its issuer i, denoted $\langle ese, r\rangle_{\sigma_i} = ese_i^r$.

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 have been issued by the node's users, or forwarded from other nodes (we emphasize that the encryption scheme together with the anonymous communication scheme obfuscate the issuing user and his owner node). Once an etx is included in a committed Eblock (i.e., an Eblock that is appended to the blockchain) it is removed from all Epools it appears in.

Committee-map (Cmap^r). A Committee-map is an ordered list of the nodes in the system and their respective reputations, corresponding to a specific term. The first $m = 3f + 1$ nodes in Cmap^r form the committee of term r (with the first node serving as primary), 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). The Helix 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 $t + 1$ signed (and distinct) ese^rs $\big($these come from the $t+1$ Shares-blocks that were used in order to decrypt $EB^{r-1}\big)$. The header

[12] 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 transactions.

[13] Note that the encryption scheme should be CCA secure [25] as the protocol could be interpreted as a decryption oracle.

[14] Note that the encryption scheme should be non-malleable to prevent manipulating the randomness beacon, as will be discussed later.

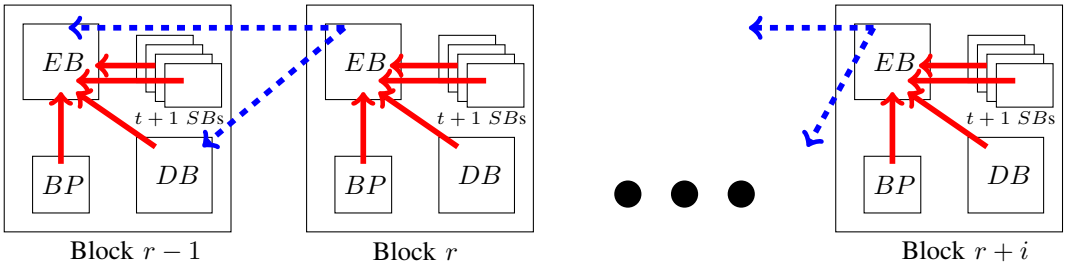

Figure 2: Illustration of the Helix blockchain. Hash pointers are used to point within blocks (in solid red) or over blocks (in dashed blue).

contains the following metadata:

- Term number, r.
- Current Committee-map[15], Cmap^r.
- Hash of the previous Eblock's header, $H(EB^{r-1})$.
- Hash of the previous Dblock's header, $H(DB^{r-1})$.
- Merkle root for the payload's Merkle tree.
- Hash of $t+1$ concatenated ese^rs.
- Composer. The composer is the node that originally constructed the current EBlock EB^r from her Epool. The composer's identity is taken into account for reputation updates.
- 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 note that the header contains a digest of the payload and that Eblocks hash-point their predecessors. To support the inherent tamper-resistance of blockchains such a cryptographic dependency structure is required. We assume that EBs are limited in *capacity*[16]. In Helix we denote by b the maximal number of etxs in an Eblock.

Block-proof (BP^r). BP^r is a list of messages and signatures, generated by the committee of term r, that acts as proof that EB^r has successfully undergone all the phases of the Helix 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 $H(EB^r)$ and the term number r. There is a slight abuse of notation here as there could be many valid different Block-proofs in term r, and different nodes might store different ones. The exact content of a Block-proof is described in Sec. IV-C.

Shares-block (SB_i^r). SB_i^r consists of a list of decryption-shares, produced by a single node i for term r; the list comprises one decryption-share per etx and ese in EB^r. According to the threshold encryption scheme, each node is capable of producing exactly one Shares-block per term. A node that received (any) $t+1$ distinct Shares-blocks for term r can verify each one's authenticity and then decrypt EB^r. The metadata of SB_i^r includes $H(EB^r)$, the term number r, a legally-generated ese_i^{r+1} and node i's signature.

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:

- Term number, r.
- $H(EB^r)$.
- Merkle root of the (decrypted) transaction tree.
- New random seed, calculated as follows: $RS^r = se_0^r \oplus \cdots \oplus se_t^r$. The choice of $t+1$ ses is dictated by the assumption on the number of Byzantine nodes ($f = t$). In Sec. V-C we formally prove the unpredictability of the random seed and explain the choice of the XOR operator.

Fig. 2 illustrates the Helix blockchain. Each block is composed of an EB, a BP, a DB, and $t+1$ SBs. A block includes internal hash pointers between components of the block as well as hash pointers to components in the previous block. Notice the Helix hash chaining structure—only unique elements (i.e., Eblocks and Dblocks) are referenced, while non-unique elements, that may differ from node to node, reference unique elements.

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

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

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

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 her role, she initiates the consensus base protocol (PBFT), proposing a new Eblock. To construct a valid Eblock, the primary chooses uniformly at random at most b etxs from her Epool. If a correct primary constructs a valid Eblock, then with overwhelming

[15]The motivation behind including Cmap in Eblocks is to allow joining nodes to sync quickly (see further details in Sec. VII).

[16]Capacity can be defined in many ways: space (such as in Bitcoin, where blocks are limited to some predetermined memory size) or "gas" (such as in Ethereum [26], where blocks are limited according to the amount of computation and storage usage they invoke).

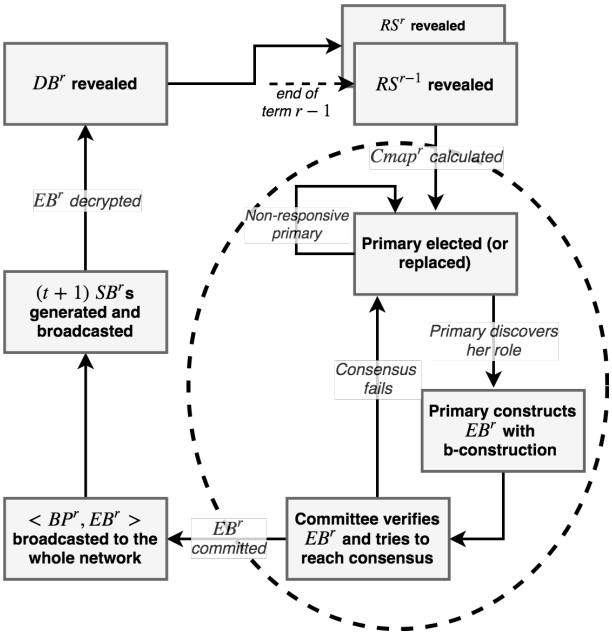

Figure 3: A high-level, system-view of the dynamics of a Helix term. Segment four and the intra-committee communication related to the base agreement protocol lies in the dotted ellipse.

probability (w.o.p.) she succeeds in committing it (see analysis in Sec. V-B) and 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 begins. Nodes that do not hear of $t+1$ 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 $t+1$ Shares-blocks, EB^r is decrypted and DB^r is revealed. Finally, a new term begins. Fig. 3 illustrates a high-level flow of a helix term.

C. The 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 etx 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; and the fourth handles reaching 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

Since Helix relies on a threshold cryptosystem, an initial interactive step for key generation is required. A distributed key generation (DKG) protocol among the nodes would be used (e.g., [23], [27], [28]).

Threshold encryption. We use the DKG to generate a common public key (PK), secret keys $(S_0, ..., S_{n-1})$ and verification keys $(V_0, ..., V_{n-1})$. The PK 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 and are used to decrypt etxs. In this article we assume static membership of nodes[17].

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 [11].

Segment 2—*etx* propagation

Transactions are issued by users; each user signs and then encrypts each transaction that he issues[18]. Each user's unique sk is used for signing, whereas the common PK is used for encryption. The sk is generated locally by each user, whereas the PK is hard-coded into the genesis block[19]. After signing and encrypting is complete, the user sends the etx to his owner node.

Communicating Encrypted-transactions. As described previously, each etx is sent by the issuing user to his 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 (a node should only certify which etxs she owns but will not publicize this until after the Eblock containing her etx is committed). This is crucial for maintaining fairness (as described in Sec. V-C). Each node maintains her own Epool by adding new etxs she receives and by removing committed etxs (i.e., etxs included in committed Eblocks). We emphasize that the progress of segment 2 is independent of the advancement of the blockchain as described in segments 3 and 4.

Segment 3—Spreading consensus

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

[17] 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 she collects enough shares to construct her key pair, in the spirit of [28]. Supporting removal of nodes can be more tricky.

[18] 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 her 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.

[19] To address cases in which the PK 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.

Process User

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

// Issuing transaction and casting it

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

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 $t+1$ SB^rs
2: $DB^r \leftarrow$DecryptEblock$(EB^r, SB^r[0, \ldots, t])$
3: AppendToBlockchain$([EB^r, SB^r[0, \ldots, t], BP^r, DB^r])$
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

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 $\langle EB^r, BP^r \rangle$, she performs a *brief-validation* and appends EB^r to her 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. In particular, a correct node would refuse to append $\langle EB^r, BP^r \rangle$ unless she had already appended the previous Eblock and can calculate the current Cmap and the current committee's composition. As a matter of fact, this step can 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 four) by the rightful committee (i.e., all signers appearing in BP^r appear in Cmap$^r[0, \ldots, m-1]$).

Shares-blocks and decryption. As briefly noted above, Shares-blocks are constructed by nodes that have committed an Eblock and have not yet received $t+1$ Shares-blocks for that Eblock (this condition is set to reduce the number of Shares-blocks propagating in the network). Normally, committee members would be first to hear of 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, she broadcasts the Shares-block to all nodes through the fast forwarding scheme. Once a node obtains $t+1$ Shares-blocks for a committed Eblock, the Eblock is decrypted and the Dblock is revealed. We note that from the nature of the threshold encryption scheme, any valid $t+1$ Shares-blocks would produce the same Dblock. The choice $t = f$ minimizes the time from committing an Eblock to decrypting it, while maintaining Helix's resilience to f Byzantine (and cooperating) nodes.

Segment 4—Base agreement protocol

Generally speaking, the base agreement protocol is a single-value variant of PBFT, re-instantiated every term, and responsible for intra-committee consensus regarding the current term's Eblock. The ellipse in Fig. 3 illustrates segment four within the Helix protocol. PBFT, as a Byzantine agreement protocol, enables Helix to reach finality, i.e., no two correct nodes ever commit different Eblocks in the same term, as long as the bound f on the number of Byzantine nodes holds (see proof in Sec. V-A). To emphasize this, Helix inherits PBFT's robust safety guarantees—safety is kept even during times the communication network is asynchronous or unreliable. We split the segment's description into two parts. First, we describe segment four as an *ideal agreement protocol* that achieves certain properties. Then, we explain how PBFT is used to implement it.

Ideal agreement protocol. The ideal agreement protocol run between m nodes $u_0, \ldots, u_{m-1}$ shall implement the following primitive.

Definition 9 (ideal agreement protocol) *An agreement protocol is said to implement the ideal agreement protocol for Helix's segment four 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 she can produce a proof p_α, such that anyone can efficiently verify that α was agreed upon by the m nodes.*

A typical Byzantine agreement (BA) protocol satisfies the first three items. Nodes that 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 ideal agreement protocol, where EB^r is the value agreed upon and BP^r serves as a proof that it was indeed agreed upon by nodes $u_0, \ldots, u_{m-1}$. The question remains though—which nodes are the rightful committee members in term r? In order to avoid forks (i.e., inconsistencies), 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 ideal agreement protocol. In Helix,

the ideal agreement protocol is implemented using a simplified version of PBFT [5] 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[20].

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

1) The view in the pre-prepare message matches her internal view number.
2) The pre-prepare message is signed by the rightful primary in that view.
3) She 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 she 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., $H(ese^r_{i_0}, \ldots, ese^r_{i_t})$, properly signed by the different nodes $i_0, \ldots, i_t$ (taken from the $t+1$ 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 exist).
6) EB^r's payload contains at most b etxs.

View-change mechanism. In PBFT, when timeouts expire (e.g., 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 ensure safety is maintained cross-views, proofs of previously committed values are submitted within the view-change mechanism. To avoid rerunning all 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 ideal agreement protocol). 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.

[20]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 [29] as proposed in Byzcoin [4] or Zilliqa [30]. In Helix, optimizations to PBFT's bandwidth consumption can be relieved as we assume $m \ll n$, where m is the size of the committee running PBFT.

Verifying consensus. The verifiability property of the ideal agreement protocol (see definition 9) is obtained by the Block-proof that stores a signed pre-prepare message, $2f$ corresponding signed prepare messages and $2f+1$ corresponding signed commit messages[21]. 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 positive 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 (see [32]) as timeouts must keep growing and cannot be adjusted to be shorter[22].

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 her 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.

D. Incentive mechanism in Helix

Processing a transaction in Helix costs money as it consumes resources such as bandwidth, storage and computation. Each node is responsible for the processing costs of transactions issued by her own users[23]. However, the cost for processing transactions for other nodes' users may be considerable, and thus requires in-protocol compensation from those nodes. If all nodes produce the same amount of transactions, compensation is settled trivially as costs balance. Though if nodes produce different amounts of transactions,

[21]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 [31].

[22]As a matter of fact, once a backup commits-locally a value in PBFT, she 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.

[23]The way a node covers the costs of her users depends on her business model, e.g., funding may come from the users themselves or from a third party.

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$_{committee}\big(\langle\langle$PRE-PREPARE, 0, r, H$(CEB)\rangle_{\sigma_{p_0}}, CEB\rangle\big)$

// Normal flow
Upon receiving a $\big\langle\langle$PRE-PREPARE, v, r, H$(EB)\rangle_{\sigma_{p_v}}, EB\big\rangle$ message
  ValidateEblock(EB) ▷ see Sec.IV-C
  $CEB \leftarrow EB$
  Multicast$_{committee}\big(\langle$PREPARE, v, r, H(CEB), $pk_i\rangle_{\sigma_i}\big)$
Upon receiving $2f$ $\langle$PREPARE, v, r, H(CEB), $pk_i\rangle_{\sigma_j}$ messages and $CEB \neq null$
  $prepared \leftarrow true$
  Multicast$_{committee}\big(\langle$COMMIT, v, r, H(CEB), $pk_i\rangle_{\sigma_i}\big)$
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$_{p_{v+1}}\big(\langle$VIEW-CHANGE, $v+1$, r, $\mathcal{P}_i^r$, $pk_i\rangle_{\sigma_i}\big)$
  $v \leftarrow v+1$

// 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()
  $\mathcal{O} \leftarrow \big\langle\langle$PRE-PREPARE, $\tilde{v}$, r, $H(CEB)\rangle_{\sigma_i}, CEB\big\rangle$
  Multicast$_{committee}\big(\langle$NEW-VIEW, $\tilde{v}$, r, $\mathcal{V}$, $\mathcal{O}\rangle\big)$ ▷ NEW-VIEW messages, after validated, are treated as ▷ PRE-PREPARE messages by recipients

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[24].

Fees, in addition to balancing costs, serve as means to incentivize nodes to follow the protocol's instructions and allow the protocol 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 her fee tariffs, where nodes with low reputation can charge lower fees for their services than their peers, possibly resulting in losses. For this reason, 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. 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

[24] 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.

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 is evaluated according to the average view number of committed Eblocks when she 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 should be punished by reducing hers reputation.
- 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 is found to construct Eblocks that significantly deviate from the Eblocks in their surrounding, she should be penalized by reducing her reputation.
- In order to encourage nodes to stay up-to-date with protocol changes, a node that constructs an Eblock with a decommissioned protocol version should be penalized by reducing her 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 contrary, 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 a 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 [5].

Claim 10 *(Helix is safe) Let $0, \ldots, n-1$ be the nodes running Helix. In term r, let EB_i^r and EB_j^r be Eblocks appended by correct nodes $i, j \in [n-1]$, 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 she is a committee member when she 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 she has been assigned to 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 her 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 the liveness property in PBFT, in a finite time, at least one of them will append a valid Eblock to her 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 $t+1$ SB^rs. A simple syncing and feedback protocol[25] 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 her 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 $t+1$ 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 are back in state 2, which 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 her Eblock before a new primary is appointed. The first of these requirements is driven by the need for practical liveness (with a guaranteed bound on progress time), whereas the second requirement 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[26] 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 τ_r, and with it, Cmap^{r+1}.

Lemma 13 *(Term with a correct first primary) If the first primary Cmap$^r[0]$ is correct, she 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 $\tau_{r-1}+x$). Since u_{r-1} follows the communication protocol, she has propagated each piece of information she has heard or produced so far (in particular EB^{r-1}, BP^{r-1} and some $t+1$ SB^{r-1}s) and thus necessarily all nodes reveal DB^{r-1} (and Cmap^r) by time $\tau_{r-1}+\Delta$. In particular, the first committee's primary (since assumed to be correct) would realize her role by time $\tau_{r-1}+\Delta$ 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 $\tau_{r-1}+\Delta+\delta$ (intra-committee communication is done point-to-point). Then, by time $\tau_{r-1}+\Delta+2\delta$ all correct committee members should receive enough prepare messages and by time $\tau_{r-1}+\Delta+3\delta$, EB^r should be committed-locally by all correct committee members. Since all correct nodes' timers expire later than $\tau_{r-1}+\Delta+3\delta$, 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

[25] 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.

[26] 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 for the Bitcoin network in [33].

committee member would append EB^r to her 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 $\tau_{r-1}+(j+2)x$, on the other hand, $\langle EB^r, BP^r\rangle$ would reach all nodes, the latest at $\tau_{r-1}+(j+1)x+2\Delta$. 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 $\tau^i_{r-1} = \tau_{r-1} + i \cdot x$. 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 $\tau^i_{r-1}+\Delta$ 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 $\tau^i_{r-1}+\Delta+\delta$. As was explained before, by $\tau^i_{r-1}+\Delta+3\delta$, EB^r would be committed-locally by all correct committee members. Since all the $(i+1)^{\text{th}}$ timers expire later than $\tau^i_{r-1}+\Delta+3\delta$, none should expire before committing-locally EB^r as required. □

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

Proof. The first property (in definition 12) is deduced directly from the two previous lemmas.

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 her 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. □

We note that if Helix is run with the doubling timeout mechanism, nonetheless 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 her resource holdings, which in our system is related to reputation[27]. *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 her probability of winning, if she 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 her holdings in the resource. We denote node i's resource holdings in term r as res^r_i and her proportionate share as $rep^r_i = \frac{res^r_i}{\sum_j res^r_j}$.

Definition 16 *(Election fairness) A blockchain protocol, where nodes are randomly elected to 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^r_i.*

Nakamoto consensus for example is not election-fair—selfish mining attacks are possible by a strong adversary with $p_a \geq 0.25$, increasing her probability to find a block over p_a as was shown in [34], [35]. Another example is Algorand [8], which falls a bit short—there, a leader knows before the network what the next random seed would be in case her block is accepted. Thus, she 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, assuming at most f nodes are Byzantine (i.e., deviate from the protocol).*

We note that in the context of Helix, election fairness is more relevant to consider in terms of committee membership rather than leader election. We proceed to prove claim 17 with the following argument: since Cmap^r is determined by RS^r, it is enough to show that RS^r serves as a randomness beacon. That is, the adversary (or any other node) cannot manipulate or foresee RS^r prior to its "global" disclosure. Put formally,

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

1) *Two correct nodes i and j with RS^r_i and RS^r_j have $RS^r_i = RS^r_j$.*

[27] 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.

2) *Prior to EB^r being committed by at least one correct node, none of the bits in RS^r can be predicted with probability greater than $1/2$ by the adversary (with non-negligible probability).*

Before the proof, we recall the fact that a valid Eblock must contain exactly $t+1$ signed (and distinct) eses (see Sec. IV-C).

Proof.

1) The first item follows directly from Helix's safety.
2) Regarding the second item, consider the adversary that wishes to predict RS^r given a candidate valid EB^r (considering a non-valid Eblock is irrelevant as it would never be committed) that has not yet been committed by even a single correct node. EB^r must contain exactly $t+1$ distinct eses and the adversary needs to predict $RS^r = se_0^r \oplus \cdots \oplus se_t^r$. We show that none of the bits in RS^r can be predicted with probability grater than $1/2$. Pick an arbitrary bit in RS^r, $z = z_0 \oplus \cdots \oplus z_t$ where z_i is the corresponding bit in se_i^r. Before EB^r being committed, the adversary has access to $ese_0^r, \ldots, ese_t^r$. From the assumption that at most $f < t+1$ nodes are controlled by the adversary, and from the security of the encryption scheme, the adversary knows at most t z_is (the adversary knows the $(t+1)$th z_i with negligible probability). Without loss of generality, be these bits $z_0, \ldots, z_{t-1}$. This implies that z_t is sampled by an honest node, i.e., in the eyes of the adversary z_t can be modeled as an unbiased Bernoulli random variable. Additionally, the encryption scheme used by Helix is non-malleable [22], which means the adversary cannot correlate her eses to ese_t such that the corresponding ses are related. This implies that (again, from the eyes of the adversary) z_t is sampled independently from any other z_i for $0 \leq i \leq t-1$. To conclude, the adversary can denote $z = w \oplus z_t$, where w is known and z_t is an unbiased Bernoulli random variable sampled independently from w. It is a well-known fact that due to the XOR operation z can also be regarded as an unbiased Bernoulli random variable, which completes 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 a non-manipulable source of randomness, we conclude that the values $\left\{H(RS^r, r+1, pk_i)\right\}_{i=0}^{n-1}$ induce a random ordering of the nodes, and the values $\{v_i^{r+1}\}_{i=0}^{n-1}$, induce a random ordering weighted by reputation. □

We draw the reader's attention to a nice benefit 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 high-level and illustrates Helix's approach. In general, 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.

The key factor in reducing nodes' ability to manipulate the blockchain is etx indistinguishability. Ideally, two etxs propagated through the network are indistinguishable—in terms of their issuer nodes, sizes, or any other criteria (of course, a node knows which etxs she issued). Thereby, using a suitable encryption scheme that hides transaction information is necessary[28]. This is realized in Helix using two 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 further 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 in an etx[29].
- 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, she can refuse to propagate it or add it to Eblocks she proposes. This problem can be mitigated quite simply by allowing users to have multiple owner nodes, forwarding etxs to all of them[30].
- 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,

[28] Note that such indistinguishability facilitates spam attacks. To mitigate this problem, we intend on using a scheme that allows a subset of the nodes to trace the issuer of any etx in question. Such a scheme can be realized via group signatures with distributed traceability [36], [37].

[29] 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.

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

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 [38]. 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 she cannot relate any semantics to, there is nothing she 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 her. Helix is order-fair in this sense when the more profitable 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 etx 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[31]. 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

Thus far we discussed Helix's mechanisms for combating transaction censorship and ensuring fair primary (and committee) election. We now turn to the more subtle procedure of constructing blocks from the pool of pending transactions.

In typical blockchain protocols, nodes (such as Bitcoin's miners) have the freedom to select which transactions to include in their blocks. Normally, when transactions come along with a fee determined by the issuer, this results in a fee market that drives fees unnecessarily high at times of high demand. While we do not elaborate on Helix's fee structure, we emphasize that Helix is designed such that fees do not play a role in etx selection[32]. This design choice makes sense when fees are not needed as the main motivation of nodes to include transactions in blocks. Indeed, our presumption in Helix is that nodes have inherent interest in processing their own transactions[33]. Under these circumstances, the main goal of this section is to describe Helix's mechanism for selecting etxs in a *fair* manner. Fairness in this regard should be reflected by the blockchain, representing each node according to the proportion of etxs she owns[34].

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. The core aspect of selection fairness is to ensure that the average *waiting time* of an etx depends solely 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 her Eblock. Correlated sampling, considered in [13], [39], [40] (see also [41] 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 her distribution while minimizing the probability that the outputs differ. In the work of [13], a MinHash strategy was used for performing correlated sampling for the case that the distributions are uniform over a subset of the domain (analogous to Epools in our setting). In [42], 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.

Instead of dictating an Eblock construction strategy that might not be aligned with primaries' selfish interests and might be difficult to validate, we suggest an Eblock validation procedure that is better aligned with nodes' interests. In the following section we describe the Eblock validation procedure applied in Helix in accordance with the correlated sampling scheme. We then show that strategies with substantial prob-

[31] 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 her own buy transaction beforehand, where she can buy the stock's shares before the price rises. Later, the node releases the user's transaction and sells her recently bought shares at a higher price.

[32] This design choice enables a variety of fee mechanisms e.g., constant fee per transaction, periodic subscription fees, and models in which fees are determined after processing.

[33] For instance, nodes run consumer applications and wish to service their end-users by processing the transactions they issue.

[34] Under such notion of fairness nodes may be encouraged to artificially produce many self-owned transactions in order to get more block real-estate. Carefully designed fee mechanisms could disincentive such behavior, e.g., paying increased fee for failing transactions.

ability of passing validation remain close to the construction dictated by the correlated sampling scheme. Throughout this section we restrict ourselves to high-load epochs by assuming that the total number of pending etxs is Γb for some $\Gamma > 1$ (where b is the maximal number of etxs in a valid Eblock).

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 tweaked with a random seed to eliminate its predictability, yielding a common, random and unpredictable serialization of the etxs. Formally, when considering the Eblock in term r, the nodes order the pending etxs according to the values $H(RS^{r-1}, etx)$, and refer to these values as the *hash values* of the etxs. We denote $H(etx)$ for brevity.

To ensure unpredictability, we introduce the notion of *locking time*. We denote by EP_i^τ node i's locked Epool at time τ, which is a snapshot of the etxs in i's Epool at time τ[35]. We denote by τ_i' the time at which node i revealed DB^{r-1}, and define the locking time of node i to be $\tau_i := \tau_i' - 2\Delta$ (recall the definition of Δ from Sec. III-D as the upper bound on the dissemination time between any pair of nodes).

In the validation procedure, nodes are asked to consider the serialized etxs in their Epool at locking time. The choice of τ_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, detracting from the randomness of the common ordering. Restricting nodes to consider only etxs which were known to them prior to time $\tau_i' - 2\Delta$ prevents this kind of attack. From now on by writing EP_i we mean $EP_i^{\tau_i}$, unless stated otherwise.

Due to network latency and inherent properties of Helix, we cannot expect EP_i and EP_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. To model this similarity, we use a probabilistic model satisfying the following property. For any two correct nodes i, j each etx in EP_i is in EP_j with probability at least α. We refer to α as the *similarity parameter* of the network.

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, she checks two conditions regarding the size of the proposed Eblock and the size of its overlap with a set of transactions calculated according to her own Epool. We define T_p be the maximal hash of an etx in EB_p, i.e., $T_p := \max\{H(etx) | etx \in EB_p\}$. Likewise, 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 T_p, and $b_i := \max\{|EB_i'|, b\}$. We further denote by b' the size of $EB_p' = \{etx \in EP_p | H(etx) \leq T_p\}$ and say that EB_p was constructed under a b'-construction. This illustrates the fact that EB_p was selected as a subset of size b among the b' lowest hashed etxs in EP_p such that the b'^{th} etx was included. The setting is illustrated in Fig. 4.

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

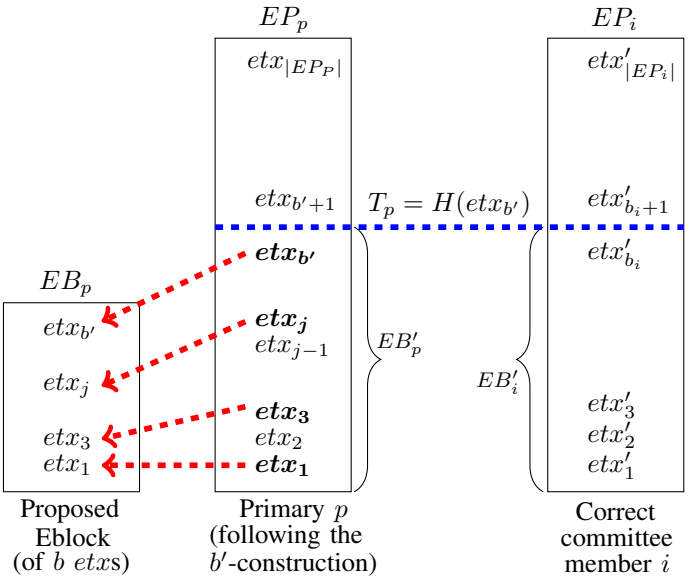

Figure 4: Illustration of the correlated sampling validation process. In each Epool, etxs are sorted based on their hash values. A threshold T_p is determined by the maximal hash value of an etx in the EBlock EB_p proposed by the primary p. A correct committee member i examines the overlap between EB_p and the set of etxs in EP_i which hash below T_p.

Under these notations, the validation checks (in the context of selection fairness) performed by correct node i upon receiving a proposed Eblock, EB_p (from primary p), are:

1) $|EB_p| = b$
2) $|EB_p \cap EB_i'| \geq \beta_\alpha(b_i)$ for $\beta_\alpha(b_i) := \alpha b_i - \sqrt{10 b_i}$

The second condition encourages primaries to construct Eblocks with low b'. The minimal value of b' is b; in the event that the value of b' is in fact b, the selection scheme is perfectly 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 large b_i and accordingly large $\beta_\alpha(b_i)$, reducing the chances of EB_p to pass validation. $\beta_\alpha(b_i)$ is the maximal value for which Eblocks constructed with $b' = b$ pass validation w.o.p., as implied by Hoeffding's bound. This is shown formally in the next section.

B. Liveness under b-construction

The extra validation process dictated by the correlated sampling scheme bears risk to the liveness of the protocol. Blocks that would have passed validation might get rejected once correlated sampling validation is enforced. We thus prove that

an Eblock compliant with the b-construction passes validation of any correct committee member w.o.p.

Claim 19 (Liveness under b-construction.) *Let EB_p be an Eblock constructed according to the b-construction, i be a correct committee member. Then, EB_p passes i's validation w.o.p. (under the assumption that α bounds from below the similarity parameter of the network)*

We use the following lemma.

Lemma 20 *For two correct nodes k, l and a general set $A \subset EP_l$, $|A \cap EP_k| > \beta_\alpha(|A|)$ with probability greater then $1 - 2\exp(-20)$, where $\beta_\alpha(x) := \alpha x - \sqrt{10x}$.*

Proof. Denote $Y := |A \cap EP_k|$. We can write $Y = \sum_{etx \in A} Y_{etx}$, where Y_{etx} are indicator random variables,

$$Y_{etx} = \begin{cases} 1 & \text{if } etx \in EP_k \\ 0 & \text{otherwise} \end{cases}$$

By definition, these random variables are i.i.d. Bernoulli variables, with success probability (at least) α. This is true from the similarity assumption. By Hoeffding's inequality, for any $\delta \geq 0$

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

The expected value of Y is bounded from below by $\alpha|A|$, thus taking $\delta = \sqrt{\frac{10}{|A|}}$ and plugging this into the left-hand side of Hoeffding's inequality, we get that with probability $> 1 - 2\exp(-20)$

$$\begin{aligned} |A \cap EP_i| = Y &> \mathbb{E}(Y) - |A|\delta \\ &\geq \alpha \cdot |A| - \sqrt{10|A|} =: \beta(|A|) \end{aligned}$$ □

We now turn to prove claim 19.

Proof. Node i's validation comprises of two checks. Check number 1 is that $|EB_p| = b$, which passes with probability 1. For clarity of the proof, we break check number 2 into two:

2.1 $|EB_p \cap EB'_i| \geq \beta_\alpha(b)$
2.2 $|EB_p \cap EB'_i| \geq \beta_\alpha(|EB'_i|)$

For 2.1, take $A = EB_p \subset EP_p$. For a correct primary, this is simply a general random set of transactions of size b. The above lemma can be used as $|EB_p \cap EB'_i| = |EB_p \cap EP_i|$, yielding that the probability for failing this condition is at most $2\exp(-20)$. The argument for condition 2.2 is similar, only this time $A = EB'_i \subset EP_i$ and we use the equality $|EB'_i \cap EB_p| = |EB'_i \cap EP_p|$. The latter equality is derived from the assumption that EB_p is a b-construction, which implies $EB_p = EB'_p = \{etx \in EP_p | H(etx) \leq T_p\}$. Now, the equality is easily derived from the definition of EB'_i. □

The claim establishes the fact that a primary following the b-construction would pass validation w.o.p., thus keeping the protocol live. In the next section we generalize the analysis above, considering Eblocks admitting b'-constructions for $b' > b$.

C. Selection fairness

Under conditions where all nodes follow the b-construction, the core aspect of selection fairness is an immediate result—an etx is included in the next Eblock with probability $1/\Gamma$. 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. The inevitable "room for error" permitted in the validation process (due to potentially non-identical Epools), prohibits such behavior from being completely eliminated. In this section we analyze the extent to which the validation scheme suggested above can guarantee selection fairness and a fair 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. The second method is through b'-constructions, for $b' > b$. We study the dependency between b' and the certainty of an Eblock to pass validation.

We start with a high-level analysis of the tailoring approach. While the locking mechanism discussed earlier eliminates the advantage of revealing the random seed prior to the rest of the network, tailoring etxs can still be a worthwhile strategy. Issuing many tailored copies of the same transaction and forwarding them to the network would increase the probability of one copy to be processed quickly. In order to avoid such behavior, suitable fee models are needed[36].

Alternatively, nodes may keep their tailored etxs hidden from the network and include them only when constructing Eblocks. the Helix Eblock validation process prevents over-exploiting this strategy. Specifically, since tailored etxs are clearly 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 $\beta_\alpha(b)$.

We continue with a formal analysis of the b'-construction. According to this strategy, the primary first looks at EB'_p which contains the lowest b' etxs in EP_p. Of those, she selects the b etxs she 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'-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 (average-case) assumption. Clearly, $|EB_p \cap EB'_i|$ is bounded from above by

[36] For example, one such fee model can be realized by a system rule that dictates an increased fee for failed transactions. In UtxO-based architectures (like Bitcoin), such an approach would be easier to enforce. In account-based architectures that support rich scripting languages (like Ethereum), circumventing such rules would be considerably impractical.

b, and we will assume that it is exactly αb, as the probability of an $etx \in EP_p$ to be in EP_i is (at least) α. Under this assumption, $Pr(A)$ becomes $Pr\left(\alpha b \geq \beta_\alpha(b_i)\right)$. Calculating for which b_i this condition holds gives rise to a quadratic inequality, whose solution yields $Pr(A) = Pr(b_i \leq M)$ for $M = \left(\frac{\sqrt{10}+\sqrt{10+4\alpha^2 b}}{2\alpha}\right)^2$. Clearly, M satisfies $\alpha b = \beta_\alpha(M)$ and $M > b$. Recall that $b_i = \max\{b, |EB'_i|\}$, which implies $Pr(b_i \leq M) = Pr(|EB'_i| \leq M)$.

We use the Chernoff inequality to bound $Pr(|EB'_i| \leq M)$ from below. For this, we use the expected value of $|EB'_i|$, which can be approximated using a simple symmetry argument (based on the even dissemination of messages in the network), yielding $\mathbb{E}[|EB'_i|] = |EB'_p| = b'$. Our goal now is to find, for any desired probability to pass validation q, the maximal b' for which $Pr(|EB'_i| \leq M) \geq q$. The Chernoff inequality for the complement event yields

$$Pr(|EB'_i| > M) = Pr\big(|EB'_i| > (1+\delta)b'\big) \leq \exp(-\delta^2 b'/3)$$

where, $\delta = \frac{M}{b'} - 1$. Chernoff requires $\delta \in [0,1]$, which implies $b' \leq M \leq 2b'$ must hold. We are thus led to compute the maximal $b' \leq M$ for which $f(b') := \exp(-\delta^2 b'/3) = 1 - q$.

Solving this equation, Table I illustrates maximal $b' = b'(q)$ values (note that for $b' \leq M$, the function f is monotonically increasing) for various passing validation probabilities q, and various Eblock sizes b, while keeping $\alpha = 0.95$ (which yields different M values). Notice that as q increases, the primary has to be more careful and b' decreases. The fact that $b'(q)/b$ is very close to 1 indicates that the freedom a primary has to act unfairly in constructing EB_p is rather limited.

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 the $b'(q)$-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

We clarify the motivation behind the analysis above. From the point of view of a primary that wants to construct an Eblock with a b'-construction, a prominent question is what b' to choose given she wants to pass validation with probability q. The above analysis bounds b' from below given b and α.

To conclude this section, we analyze to what extent a b'-construction allows a primary to promote her etxs. Given b', a primary would first include all etxs (among the first b' ones) she wishes to promote, and then complete the Eblock space randomly (this is a direct result of the indistinguishability of the encrypted transactions). 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 she issued). 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 b'-construction described above.

Lemma 21 *Let $etx \in EP_p$ and $b' \geq b$. Suppose EB_p is constructed according to a b'-construction with ξ_p, the fraction of promoted etxs among all pending etxs in EP_p. The probability that an arbitrary etx is included in EB_p is*

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

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

D. Experimental results

To examine selection fairness beyond the theoretical bounds, we conduct simulation-based experiments that reflect Helix's fairness properties in practice. We isolate specific elements within the system and inspect their effect on fairness. We simulate primaries that follow the b'-construction strategy and measure fairness as the ratio b'/b. Our open-source code is available online [43].

To obtain probabilistic results we repeat the same experiment a thousand times and give an average case analysis. In our experiments a pair of nodes, a primary p and a committee member i, construct and validate a block respectively. We model their Epools at locking time, EP_p and EP_i, by sampling random etxs, where the total number of pending etxs is $10b$ and $|EP_p \cap EP_i| = \alpha \cdot 10b$. The block size is set to $b = 1000$.

Fig. 5 illustrates our first experiment. We set $\alpha = 0.95$, and check the relation between the primary's desired probability to pass a single validation, q, and the maximal b' she can use. We compare the theoretical lower bound to the experimental results. It is interesting to note that the practical results indicate a very moderate decent from $q = 0.95$ to $q = 0.5$ where b' increases by roughly 20 etxs. This should imply that rational primaries would use $b' < 1100$ (and avoid unnecessary risks). Note that within these (less than) 100 extra etxs there is only

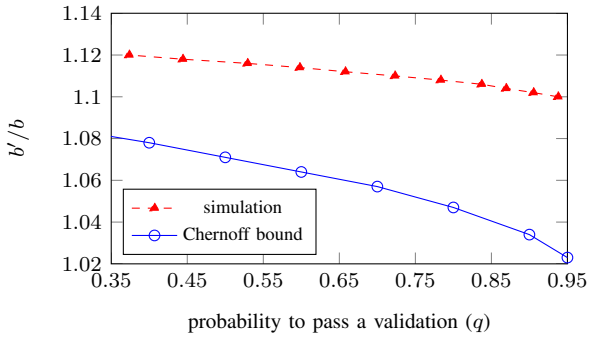

Figure 5: The ratio b'/b vs. the desired probability to pass a single validation.

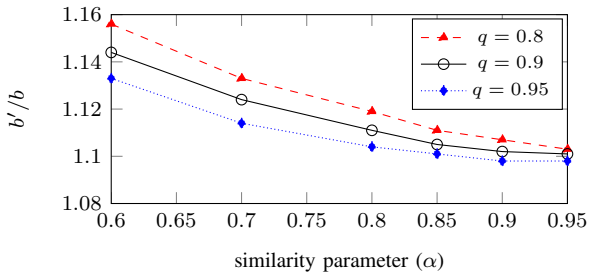

Figure 6: Impact of network similarity parameter on fairness—larger α values enhance fairness.

a fraction of promoted ones, see Fig. 7. In case we take larger bs or αs the relative amount of feasible manipulation further reduces.

In a real system, the Chernoff bound could serve as "advice" to the primary, as to what b' she can use and still pass validation with her desired probability. An upper bound analysis, that would indicate the maximal b' a primary might use and still pass a single validation with a desired probability, is left for future work.

Fig. 6 illustrates the second experiment which checks the effect of α on selection fairness. It clearly shows that larger α yields smaller b'/b ratio. Intuitively, large α values allow the validator to know more about the primary's Epool thus yielding stricter validation conditions (i.e., larger β), resulting in enhanced fairness.

Finally, Fig. 7 takes into account the analysis in Lemma 21 and the fraction of etxs the primary wishes to promote in order to give a realistic estimation of the possible manipulation in b'-constructions. We choose b' values arising from the simulations presented in Fig. 5. For non-promoted etxs, the graph shows the ratio between the probability of getting included in b'-constructions, to that in the b-construction. We plot these ratios with respect to different fractions of etxs, ξ_p, the primary wishes to promote. The fact that the observed ratios are very close to 1 (even for small q values) demonstrates that in practice, selection fairness in Helix is maintained even under a $b' > b$ manipulation.

However, the main insight from the experiment section is

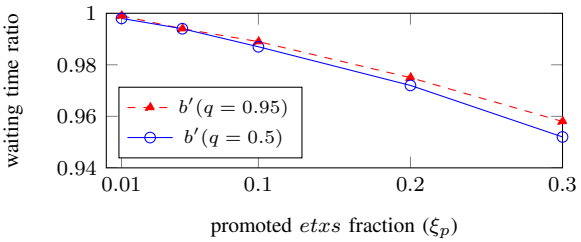

Figure 7: The ratio between the expected waiting time of a non-promoted etx in the b-construction, and that in a b'-construction. In both curves $\alpha = 0.95$.

that, surprisingly, the effectiveness of the selection fairness scheme is quite indifferent to the similarity parameter α. Between $\alpha = 0.95$ and $\alpha = 0.6$, the ratio b'/b and the expected waiting time of non-promoted etxs does not differ by much. Thus, in practice, a system that uses our selection fairness mechanism should not make a special effort to increase α.

We note however that in the experiments above, the parameter α used for validation matched the actual overlap of Epools. We believe that a loose estimation of α would enable primaries to manipulate Helix's selection fairness, e.g., by tailoring many etxs. It is left for future work to find methods to reduce the importance of estimating α accurately.

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 (i) to sync is by sending some node (j) a request to sync that contains the highest term for which she has a Dblock (DB^r). If j has higher term committed Eblocks, she sends i tuples of the form (EB^q, BP^q, SB_0^q, ..., SB_t^q) for $q > r$ in an incrementing manner. Upon receiving the $(r+1)$-term tuple, i verifies BP^{r+1} as a valid Block-proof for EB^{r+1} under the $(r+1)$-term committee (as determined by DB^r). i then appends EB^{r+1} and reveals DB^{r+1} and the committee for term $r+2$ (using $SB_0^{r+1}, \ldots, SB_t^{r+1}$). This process is performed repeatedly until j completes sending all its blockchain. If j itself is *in-sync with the network*, then once the process is terminated, i 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 i 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., $\langle EB^r, BP^r \rangle$) i validates BP^r. Particularly, in the fast sync optimization, i drops the Cmap brief-validation check as defined in Sec. IV-C. If found valid, we claim that i is safe to append EB^r to her blockchain, even though i 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 i receives segment 4 messages (i.e., committee-related messages such as pre-prepare, prepare, etc.) she may assume that it is in the r-term PBFT committee. But since i does not have the previous Dblock yet, she cannot reassure it and thus cannot participate. Thus, in this term, i still counts as faulty. In particular, if i is primary in this term, she would lose her turn.

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

Claim 22 *(Fast syncing safety) Let $0, \ldots, n-1$ 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 $i, j \in [n-1]$, 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 experience an equal level of service (transaction confirmation time, throughput, etc.). Helix is 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. Helix ensures the end-users enhanced protection from censorship and discrimination relative to other typical solutions, while requiring them to utilize a negligible amount of resources. It can therefore be seen as a fair protocol towards end-users as well.

IX. Acknowledgements

We would like to thank Rosario Gennaro, Steven Goldfeder and Idit Keidar for their fruitful discussions and helpful suggestions.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://bitcoin.org/bitcoin.pdf

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

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

[4] 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.

[5] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002.

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

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

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

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

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

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

[12] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series consensus system," January 2018. [Online]. Available: https://dfinity.org/pdf-viewer/library/dfinity-consensus.pdf

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

[14] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[15] L. Lamport, "Paxos made simple," *SIGACT News*, vol. 32, no. 4, pp. 18–25, 2001.

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

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

[18] J. Gray, "Notes on data base operating systems," in *Operating Systems, An Advanced Course*, 1978, pp. 393–481.

[19] L. Lamport, R. E. Shostak, and M. C. Pease, "The Byzantine generals problem," *ACM*, vol. 4, no. 3, pp. 382–401, 1982.

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

[21] Y. Desmedt, "Threshold cryptography," *Encyclopedia of Cryptography and Security*, pp. 1288–1293, 2011.

[22] D. Dolev, C. Dwork, and M. Naor, "Nonmalleable cryptography," *SIAM*, vol. 45, no. 4, pp. 727–784.

[23] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," *J. Cryptology*, vol. 15, no. 2, pp. 75–96, 2002.

[24] G. Bleumer, "Random oracle model," *Encyclopedia of Cryptography and Security*, pp. 1027–1028, 2011.

[25] B. Alex, *Chosen Ciphertext Attack*. Springer US, 2011, pp. 205–205.

[26] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014. [Online]. Available: http://gavwood.com/Paper.pdf

[27] 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.

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

[29] 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.

[30] The ZILLIQA team, "The ZILLIQA technical whitepaper," 2017. [Online]. Available: https://docs.zilliqa.com/whitepaper.pdf

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

[32] Y. Amir, B. A. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine replication under attack," *IEEE Trans. Dependable Sec. Comput.*, vol. 8, no. 4, pp. 564–577, 2011.

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

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

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

[36] J. Blömer, J. Juhnke, and N. Löken, "Short group signatures with distributed traceability," in *MACIS*, 2015, pp. 166–180.

[37] D. Zheng, X. Li, C. Ma, K. Chen, and J. Li, "Democratic group signatures with threshold traceability," *IACR*, vol. 2008, p. 112, 2008.

[38] L. Baird, "The Swirlds Hashgraph consensus algorithm: Fair, fast, Byzantine fault tolerance," 2016. [Online]. Available: http://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf

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

[40] J. M. Kleinberg and É. Tardos, "Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields," *J. ACM*, vol. 49, no. 5, pp. 616–639, 2002.

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

[42] R. L. Rivest, "Symmetric encryption via keyrings and ECC," 2016.

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