Bracing A Transaction DAG with A Backbone Chain

Shuyang Tang[1,3], Qingzhao Zhang[2], Zhengfeng Gao[1], Jilai Zheng[1], and Dawu Gu[1]

[1] Shanghai Jiao Tong University, Shanghai, China
{htftsy, zfgao1994, zhengjilai, dwgu}@sjtu.edu.cn
[2] University of Michigan, MI, U.S.A.
qzzhang@umich.edu
[3] Cryptape Information Technology Co., Ltd, Hangzhou, Zhejiang Province, China
tangshuyang@cryptape.com

Abstract. Directed Acyclic Graph (DAG) is becoming an intriguing direction for distributed ledger structure due to its great potential in improving the scalability of distributed ledger systems. Among existing DAG-based ledgers, one promising category is transaction DAG, namely, treating each transaction as a graph vertex. In this paper, we propose Haootia, a novel two-layer framework of consensus, with a ledger in the form of a transaction DAG built on top of a delicately designed PoW-based backbone chain. By elaborately devising the principle of transaction linearizations, we achieve a secure and scalable DAG-based consensus. By implementing Haootia, we conclude that, with a rotating committee of size 46 and a confirmation latency around 20 seconds, Haootia achieves a throughput around 7500 TPS which is overwhelming compared with all formally analyzed DAG-based consensus schemes to date and all existing non-DAG-based ones to our knowledge.

Keywords: Blockchain · Consensus · Directed Acyclic Graph · Proof-of-Work

1 Introduction

Since the emergence of Bitcoin [1] in 2008, various decentralized consensus schemes have been brought about to realize a decentralized ledger in the permissionless environment. Most existing schemes adopt a blockchain, which is the fundamental building block of the consensus of Bitcoin (i.e. *Nakamoto Consensus*), to store and extend the ledger. However, blockchain is heavily limited in its scalability. Specifically, its throughput of transactions is far from satisfactory and transactions are confirmed at a slow rate due to possible chain forks. In recent years, it has been proposed that alternative consensus schemes based on *Direct Acyclic Graphs* (DAGs) have the potential of replacing blockchains for a significantly improved scalability. Existing DAG-based consensus schemes can be classified into three categories according to their graph structures or two categories

accordingly to their vertex structures. Some DAG-based consensus schemes are with ledgers in the form of *a DAG with a main chain* like *GHOST* [2], *Inclusive Blockchain* [3], *Conflux* [4], and Byteball [5]. Some are with ledgers of *a naive DAG* (where all nodes of the graph have no predetermined order) like *IOTA* [6], *Spectre* [7], *Phantom* [8], *Meshcash* [9], and *Graphchain* [10]. Moreover, some are with ledgers of *parallel chains* like *Hashgraph* [11], *Nano* [12], and *Dexon* [13].

Most of them are based on a DAG of blocks where each vertex of the graph is a block of transactions assembled by miners like the GHOST (with a throughput of 275), Conflux (with a throughput of 3200), or Nano (with a throughput of 306). However, from our perspective, this approach is not fully merited from the concurrency nature of DAGs and its throughput is limited due to duplicate transactions. Few existing schemes adopt a graph of transactions where each vertex corresponds to one transaction like IOTA. However, their security remains unclear due to their overcomplicated DAG structure and are challenged for possible attacks [14].

We aim to propose a consensus protocol that (1) is based on a DAG of transactions (instead of blocks) which further leverages the concurrency nature of DAGs and avoids duplicate transactions, (2) challenges the limited transaction throughput as well as providing an instant confirmation for all transactions, (3) and can be shown secure through a formal proof. To our knowledge, no existing DAG-based consensus has fulfilled all three goals above. Targeted at this purpose, we come to the following insights, forming the roadmap to the Haootia protocol we are about to propose.

The goal of consensus is a hard-to-tamper linearly ordered log. The ultimate goal of any distributed ledger has been controversial. Some believe it is the verification and confirmation of transactions, while some believe that it is the linearization of all transactions (known as *state machine republication* in the distributed system literature). We adopt the second one in this paper, since in most applications of any distributed consensus, as long as a linearly ordered list of transactions is determined, it is feasible to finish transaction verification in the application level via a deterministic rule. For example, illegal transactions are taken out of consideration and within a set of conflict transactions, only the one first appeared in the linear log is considered valid. The security of the ledger is thereby essentially the hard-to-tamper property of the append-only log of transactions. Likewise, the liveness of the consensus is in actual the property that any newly appeared transaction is always comprised into such a transaction log in time.

Instant confirmation can be provided by a permissioned consensus protocol among a committee. In the cryptocurrency literature, multiple consensus schemes with a dynamic committee elected among all miners in a decentralized manner have been proposed and carefully analyzed in the past few years [15–17]. Specifically, a dynamically rotated committee, usually consisting of proposers of a certain interval of blocks in a blockchain, performs all transaction verifications and linearizations via a permissioned consensus among the committee. In this way, a

permissioned consensus (like PBFT) can be also applicable to the permissionless environment, and hence the scalability is greatly improved due to the lightweight nature of permissioned consensus. We also adopt this methodology. However, different from existing committee-based consensus schemes, we aim to bring about a total order to a DAG of transactions. To meet this end, we have the committee reach consensus only on a linear chain of *key nodes*. Each key node u newly confirms a tree of nodes (we refer to such a tree as *the increment tree* of u). In this way, by concatenating the reversed breath-first traverse sequence of the increment trees of all these key nodes, an append-only total ordering of nodes is attained.

A considerable committee size should be supported. The precondition of the safety and liveness of permissioned consensus schemes like PBFT is a 2/3 honest rate among consensus participants. This requires committee-based consensus schemes to guarantee such a 2/3 honest rate for every committee to be elected for all the time. Thereby, the size of the committee should be considerably great to reach satisfactory security. However, in most existing committee-based permissionless consensus schemes, the committee (of size k) is determined as the miners of k newest confirmed blocks of a blockchain which limits the committee size. Since the block generation rate is limited, to make the committee size great, there has to be a long "term of office" for each lucky miner to serve as a committee member, during which period they are exposed to adaptive attacks. To face this issue, we adopt a new diagram of a generalize proof-of-work for the committee election.

1.1 Our Contributions

We propose Haootia, a DAG-based consensus with a fast convergence speed and considerable throughput provided by a dynamic committee with a considerable size maintained by a backbone chain based on our generalized proof-of-work. The basic security properties of Haootia are proved formally. By fully implementing our proposed Haootia, we conclude that, with a rotating committee of size 46 and a confirmation latency around 20 seconds, the throughput of Haootia fluctuates around 7500 TPS which is overwhelming compared with either existing DAG-based consensus schemes with formal security analyses or existing non-DAG-based ones to our knowledge.

1.2 Background and Related Works

Starting from the emergence of Bitcoin [1], recent research of consensus mechanism has evolved from a permissioned environment (see [18–21]) to a permissionless environment. Bitcoin's *Proof-of-Work* (PoW) consensus mechanism effectively defends against *Sybil attacks* [22] and double-spending [1]. However, its underlying Nakamoto consensus is notorious for its limited scalability. To more fundamentally improve the scalability, committee-based consensus schemes

(see [15,16,23–26]) are introduced to separate leader election from ledger extension and have one or few (via *sharding*) dynamically rotating committees verify and linearize transactions through a permissioned protocol like PBFT of [21].

To further overcome the drawbacks of classical blockchains [27,28] and, most importantly, improve the throughput, DAG-based consensus schemes are proposed by both commercial and academic sides. To our knowledge, existing DAG-based consensus schemes can be classified into three categories. Namely, consensus with ledgers in the form of *a DAG with a main chain* like [2–5], *an arbitrary DAG* like [6–10], or *parallel chains* like [11,12]. These schemes are variant in their design principles. Some of them are proposed in academic venues while others are proposed by commercial powers. Some of their design components lack a rigorous security analysis [14]. *Prism* [29,30] provides an extremely high throughput but is controversial for possible attacks due to its complicated block structures. In particular, the fairness and efficiency of existing DAG-based consensus are rigorously analyzed by Birmpas et al. [31].

2 Overview of Our Protocol

The protocol is executed in epochs, each corresponding to a committee of leaders elected from miners.

The Dynamic Committee. To elect the dynamic committee of considerable size (of k) without bringing about a long exposure of adaptive attacks, we adopt the new diagram of a generalized proof-of-work (GPoW) by introducing a *backbone chain*. The backbone chain differs from the Nakamoto blockchain in the sense that

1. Each block contains multiple hash solutions (λk in expectation).
2. It does not directly confirm transactions, thereby, its rate of growth is no longer a bottleneck of the throughput of transactions. As a result, it is allowed to grow relatively slowly to decrease fairness loss from hash solution losses (see later) brought about by latency.
3. To still assure the hard-to-tamper property, all nonce solutions regarding the previous epoch are recorded in the block. Among these solutions, according to a certain securely generated randomness, k of them are uniformly randomly picked up as lucky solutions.
4. Each block corresponds to an epoch, designating the committee of the epoch. Namely, proposers of lucky solutions in the block form the committee of the corresponding epoch.
5. The block assembly phase for the next epoch as well as the nonce solution reception is performed by the current committee via a permissioned PBFT that guarantees our security requirements as long as a $2/3$ honest rate among the committee is assured.

We denote the expected time length of an *epoch* as T_e (a predetermined parameter). In GPoW, the difficulty of the hash puzzle is lowered down such that in

the expectation it takes T_e time for global miners to discover λk valid hash solutions. During the epoch of a committee, committee members reach consensus on a list of received valid hash solutions. With a delicately designed scheme, they terminate the current epoch, negotiate on the next block (hence the next committee) and finish the committee switchover as long as they receive λk valid nonce solutions. With such a technique, as one fundamental building block, a dynamically rotating committee is attained securely. However, it is far from easy to actually implement any step above without compromising security or even liveness. Therefore, we have described the detailed protocol in Appendix. A for readers interested in implementation details.

Intra-Committee Consensus. In this work, we adopt *practical Byzantine fault-tolerance* (PBFT, [21]) as the intra-committee consensus that we are about to utilize in the permissioned internal consensus of a committee. Since PBFT is a mature tool implemented and testified already in many existing systems, we sometimes take it as a black box assuring both safety and liveness, namely, they always terminate in time and reach an honest agreement, as long as over 2/3 consensus participants are honest. Due to the permissioned nature of PBFT, the committee is allowed to improve the network routing in the peer-to-peer network by modifying their forwarding tables to speed up the intra-committee consensus. Due to the properties above, in certain scenarios, we are allowed to treat the committee as an entirety to avoid redundancy. For instance, an outcome s of a PBFT among committee members can be described as "the committee outputs s" rather than "having all honest committee members return to us s after they execute the PBFT which can be testified by having over 2/3 valid signatures attached to the outcome".

Transaction Linearization. It is not trivial to achieve an append-only total-ordering on transaction units of the growing DAG. In Haootia, motivated by awards and fees, the committee outputs a chain of key nodes to finish the ultimate linearization of all transactions.

1. The committee always outputs a key node appended to the key node chain every certain time interval, no matter how its committee members are internally switched over. Each key node confirms a sequence of transactions. The total order on the DAG is the concatenation of the sequences of transactions confirmed by each key node.
2. The sequence of transactions confirmed by a key node is the reversed breath-first traverse sequence of the *increment tree* of it.
3. The increment tree of a key node is a tree of transactions nodes.[4] The tree is defined to be all transaction nodes the key node connects to (i.e. there is a path from the key node to it), but not yet connected to by previous key nodes. The formal definition of the increment tree is the *recursion tree* of the

[4] We call it a tree, but it is actually a DAG which can be considered as a tree in practice, see details in Sec. 3.

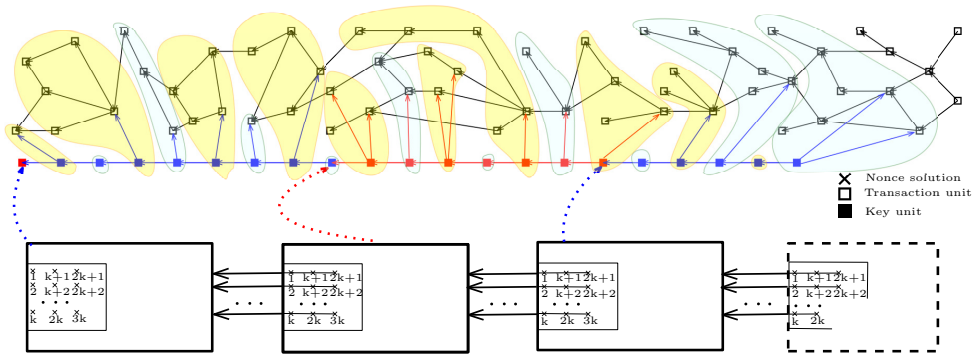


Fig. 1: A Full Ledger with $\lambda = 3$ (The DAG contains a linear chain of key nodes in red and blue outputed by committees corresponding to each block of the backbone chain. All key nodes newly confirm a distinct set of new transaction nodes marked by a colored zone. Each assembled backbone block contains $3k$ nonce solutions.)

key node subtracted by the recursion tree of the previous key node, which is shown in Sec. 3.

3 The Haootia Consensus

We firstly introduce notations and assumptions and thereafter describe the consensus by three parts with the help of Fig. 1. Namely, the PoW-based committee election protocol based on a backbone chain shown in Fig. 1 as three (with an unfinished one) blocks in the lower part, the intra-committee consensus outputting the key node chain (solid units in red and blue in the figure), and the transaction linearization protocol that *confirms* transaction nodes and brings about a total-ordering to them.

3.1 Notations and Assumptions

$[N]$ stands for the set $\{1, 2, \ldots, N\}$. A direct graph is described as a pair $G = (V, E)$ where V is the vertex set and E is the edge set where each edge is described as the pair of two vertices. For simplicity, we occasionally denote $G.V$ as the vertex set of graph G and $G.E$ as the edge set of it. For a graph $G = (V, E)$, E^+ stands for the transitive closure of E, i.e., for any u, v of V,

$$(u, v) \in E^+ \iff \exists \ell \in \mathbb{N}^+.\ \exists (v_0 = u, v_1, v_2, \ldots, v_\ell = v) \in V^{\ell+1}.\big(\forall i \in [\ell].\ (v_{i-1}, v_i) \in E\big).$$

If $(u, v) \in E^+$, we say that v is *reachable* from u in the graph. The committee of an epoch $\mathcal{T}$ is notated as $\mathsf{com}_{\mathcal{T}}$. Each committee is of size k. As our protocol is executed in epochs, we denote the previous (the next) epoch of an epoch $\mathcal{T}$ as

$\mathcal{T}_{pre}$ ($\mathcal{T}_{nxt}$) to facilitate protocol descriptions. We adopt terminologies of *nodes*, *vertices* and *units* interchangeably but they stand up for the same concept in this paper. To face the data availability issue, we assume a partially synchronized network with delays bounded by δ[5].

The Directed Acyclic Graph In this part, we propose the outline of the DAG-based ledger of transactions and show the linearization of transactions via key nodes. The DAG consists of two parts of nodes: *normal nodes* (or *transaction nodes*) proposed by each transaction proposers and *key nodes* proposed by the committee. Motivated by potential rewards, each proposed node "refers to" certain existing valid nodes on the DAG to take part in their "confirmation". This manifests in the DAG as edges pointing from the proposed node to nodes to refer. However, a node is finally confirmed only when there exists a path from a key node to it. Such an admissible directed acyclic graph describes the view of the whole ledger. We not only require that the ledger is a DAG consisting of properly linked key nodes and normal nodes but also ask the DAG to be *succinct* to reduce redundant node confirmations. Formally, admissible directed acyclic graphs are defined as follows.

Definition 1 (Admissible Directed Acyclic Graph). *In this article, a directed graph $G = (V, E)$ is an admissible directed acyclic graph iff the vertex set $V = V_{tx} \cup V_{bone}$ ($V_{tx} \cap V_{bone} = \emptyset$) and the edge set $E \subseteq V \times V$ satisfy the following properties.*

- ***Acyclic Graph.*** *For all vertex $u \in V$, $(u, u) \notin E^+$.*
- ***Succinctness.*** *For all vertex $(u, v) \in E$, $(u, v) \notin (E \setminus (u, v))^+$.*
- ***Ordered Key Units.*** *All backbone nodes (nodes of V_{bone}) are linearly ordered, formally, $V_{bone} = \{u_1, u_2, \ldots, u_{|V_{bone}|}\}$ satisfies that*

$$\Big(\forall i \in [|V_{bone}| - 1]\,.\ (u_{i+1}, u_i) \in E\Big) \wedge \Big(\forall i, j \in [|V_{bone}|].\ i \neq j + 1 \Rightarrow (u_i, u_j) \notin E\Big).$$

Moreover, to facilitate later proofs, we introduce the notion of *fully admissible DAG*. Namely, a direct graph which is not only an admissible DAG according the definition of Def. 1, but also having all its vertices reachable by at least one key node of it.

Definition 2 (Fully Admissible DAG (FAD)). *A direct graph $G = (V, E)$ with $V = V_{tx} \cup V_{bone}$ is a fully admissible DAG with the key node set V_{bone} if G satisfies the following two properties.*

- *G is an admissible DAG with the key node set V_{bone}.*
- *For all $v \in V$, there exists a $u \in V_{bone}$ that $(u, v) \in E^+$.*

We introduce two predicates. $\mathsf{AD}(G)$ denotes whether graph G is an admissible DAG and $\mathsf{FAD}(G)$ denotes whether graph G is a fully admissible DAG.

[5] Actually, any consensus protocols applicable to the permissionless network with an uncertain number of participants should know the upper bound of network delays as shown in [32].

3.2 Epochs and The Rotating Committee from Mining

As is shown in Fig. 1, the Haootia protocol consists of two layers: the upper DAG-based ledger layer holding ordinary nodes (also referred to as transactions) issued by transaction proposers as well as key nodes proposed by the underlying committee; and the lower backbone chain layer designating the committee, which is responsible for outputting key nodes to both linearize transactions and confirm nonce solutions generated by miners.

The Haootia protocol executes in a succession of epochs, each with a committee assigned according to one backbone block. There are three main roles during the epoch: transaction proposers, miners, and committee members. Transaction proposers are ordinary users who send transactions in the DAG; miners are those who participant in solving the hash puzzle regarding $\boldsymbol{B}_{\mathcal{T}}$ to earn themselves the opportunity to be members of the next committee; $\mathsf{com}_{\mathcal{T}}$ on the other hand, consists of the ones who successfully solve the puzzle in the last epoch and are lucky to be chosen (uniformly randomly) as members of the current committee.

Each time when a new $\boldsymbol{B}_{\mathcal{T}}$ is generated by $\mathsf{com}_{\mathcal{T}_{pre}}$, epoch $\mathcal{T}$ begins. After a nonce solution set $\mathsf{sol}_{\mathcal{T}}$ of size λk is formed, k solutions are uniformly randomly selected from it, the next committee is determiend as proposers of these solutions, and the current epoch terminates by having the committee execute Π_{switch}.

Our protocol asks each miner to work on a hash puzzle regarding the block $\boldsymbol{B}_{\mathcal{T}}$ that determined the current committee. All solutions are broadcast to the network and reached consensus on by $\mathsf{com}_{\mathcal{T}}$. During $\mathcal{T}$, one of the core work for $\mathsf{com}_{\mathcal{T}}$ is to continuously output key nodes in order for transaction linearization in the DAG based ledger, which is realized through internal consensus via PBFT. Meanwhile, $\mathsf{com}_{\mathcal{T}}$ also confirms $\langle \mathtt{solution}, \mathcal{T}, \mathsf{pk}, \mathsf{nc} \rangle$ tuples broadcast by miners by containing then into key nodes, forming a solution set $\mathsf{sol}_{\mathcal{T}}$ at the end of the $\mathcal{T}$, and then accordingly generate the next committee $\mathsf{com}_{\mathcal{T}_{nxt}}$ enclosed in $\boldsymbol{B}_{\mathcal{T}_{nxt}}$. After that, epoch $\mathcal{T}_{nxt}$ is triggered. The whole picture of switchover from epoch $\mathcal{T}$ to $\mathcal{T}_{nxt}$ proceeds as follows.

1. During $\mathcal{T}$, each miner P enumerates a nonce value nc. In case of satisfying $H(\boldsymbol{B}_{\mathcal{T}} || \mathsf{pk}_P || \mathsf{nc}) \in \mathsf{target}$, P broadcasts $\langle \mathtt{solution}, \mathcal{T}, \mathsf{pk}_P, \mathsf{nc} \rangle$ to $\mathcal{Z}$ which is then delivered to committee members.
2. $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{epoch}}$ is executed by $\mathsf{com}_{\mathcal{T}}$ (with common input $\boldsymbol{B}_{\mathcal{T}}$) and terminates with a set of nonce solutions $\mathsf{sol}_{\mathcal{T}} = \{\langle \mathtt{solution}, \mathcal{T}, \mathsf{pk}_{P_i}, \mathsf{nc}_i \rangle\}$ received by the committee $\mathsf{com}_{\mathcal{T}}$.
3. Afterwards, $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{switch}}$ is executed by $\mathsf{com}_{\mathcal{T}}$ with common input $\mathsf{sol}_{\mathcal{T}}$ and outputs the next block $\boldsymbol{B}_{\mathcal{T}_{nxt}}$. As a component of $\boldsymbol{B}_{\mathcal{T}_{nxt}}$, the next committee $\mathsf{com}_{\mathcal{T}_{nxt}}$ is determined as proposers of k uniformly randomly selected items of $\mathsf{sol}_{\mathcal{T}}$.

To help with understanding, we present Fig. 2 as an outline of the execution of the protocol in each terminal. Most of the time, each miner executes Π_{mine} and tries to solve the hash puzzle without taking part in the consensus work (like in $\mathcal{T}_{pre}$ and $\mathcal{T}_{nxt}$). When it luckily becomes the committee member of an epoch (like $\mathcal{T}$), it joins the intra-committee consensus $\Pi_{\mathsf{consensus}}$ (which is a succession of Π_{PBFT} instances, illustrated in Sec. 3.3) to collaboratively grow the key node

chain and make agreements on received nonce solutions. When λk nonce solutions pass through the intra-committee consensus, Π_{epoch} triggers the switchover protocol Π_{switch} and has the epoch terminate. Detailed protocols of each part of Fig. 2 are presented in Appendix. A.

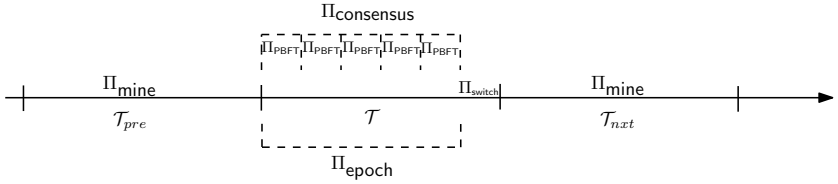


Fig. 2: Overview of The Full Protocol

3.3 Key Node Chain from Intra-Committee Consensus

The intra-committee consensus is a permissioned consensus treated as a black box in the other contexts of this section. We conservatively choose PBFT to realize the black box since its real-world security has been testified by various applications for decades. Due to the nature of permissoned consensus, every two nodes of the committee are allowed to modify forwarding tables and make direct connections. This approach significantly improves the scalability of the intra-committee consensus. Unfortunately, this brings about certain vulnerability in the face of adaptive DoS attacks. However, the risk is limited since the interval of each epoch is short for such an attack.

Each proposed key node should contain two fields. One is a list of tip units (vertices with zero in-degrees) that this key node points to in the DAG. The other one is a set of (most likely one or zero) nonce solutions received and not yet included by previous key nodes. By PBFT, such a key node is proposed and agreed by the committee (if valid), appended to the key node chain (which essentially linearizes the DAG as shown in Sec. 3.4), and thereafter broadcast to the network. The security of PBFT guarantees correctness and liveness if over 2/3 members are honest.

When realizing PBFT, we ask members of $\mathsf{com}_{\mathcal{T}}$ to take turns to serve as the primary to startup a PBFT instance by proposing a key node. In this way, fairness loss caused by the possible censorship of nonce solutions is limited for the following reasons. Firstly, the interval between key node proposals is small and each nonce solution (except for those proposed very close to the termination of the epoch) is always comprehended into a key unit by an honest primary. Secondly, each nonce solution brings about around a $\frac{1}{\lambda}$ probability of winning a committee slot, this quantity is small when λ is large.

3.4 Transaction Linearization on The DAG

In this part, we show how the key node chain uniquely determines an append-only log of transaction units in an admissible DAG. Few terminologies are introduced to facilitate descriptions. To begin with, each key node essentially confirms all units confirmed by previous key nodes (via an edge pointing to the previous key node) and a newly confirmed set of units (Fig. 5). We refer to all vertices and edges within both sets (also, the key node itself) as a *recursion tree*. Note that though we refer to it as a "tree", it may contain nodes with more than one parent node. However, this is in practice easy to solve by erasing other parental links except for the one with the smallest lexicographic order (same to increment trees). In our theoretical model, all definitions are still sound without erasing them.

Definition 3 (Recursion Tree). *The recursion tree* $\mathsf{Rec}(p) = (V', E')$ *of an admissible directed acyclic graph* $G = (V, E)$ *(*p *is a key node of* G*) is defined as the subgraph of* G *with*

$$\begin{cases} V' = \{p\} \cup \left\{ u \in V \middle| (p,u) \in E^+ \right\} \\ E' = \left\{ (u,v) \in E \middle| \ u \in V' \right\}. \end{cases}$$

Increment tree, which can be regarded as the recursion tree "subtracted by" vertices and edges already comprehended by previous recursion trees.

Definition 4 (Increment Tree). *For an admissible DAG* $G = (V, E)$ *with backbone units* V_{bone}*, an increment tree for a backbone unit* $p \in V_{bone}$ *is* $\mathsf{Inc}(p) = (\widetilde{V}, \widetilde{E})$ *with*

$$\begin{cases} \widetilde{V} = \mathsf{Rec}(p).V \setminus \mathsf{Rec}(p_{prev}).V \\ \widetilde{E} = \{(u,v) \in \mathsf{Rec}(p).E \big| v \notin \mathsf{Rec}(p_{prev}).V\}, \end{cases}$$

where $p_{prev} \in V_{bone}$ *is the unique backbone unit that* $(p, p_{prev}) \in E$*.*

Nodes of the increment tree of a key node are essentially the set of all its newly confirmed transactions. Thereby, the newly confirmed linear log of transactions is the reversed breadth-first traverse sequence of the increment tree.

Definition 5 (Reversed Breadth-First Traverse Sequence). *The reversed breadth-first traverse (RBF) sequence* $\mathsf{RBF}(G)$ *of a DAG* $G = (V, E)$ *(or* $\mathsf{RBF}(V, E)$*) is defined recursively.*

$$\begin{cases} \mathsf{RBF}(\emptyset, \emptyset) = \epsilon \\ \mathsf{RBF}(V, E) = \mathsf{RBF}\big(V \setminus \mathsf{Tip}(V,E), \{(u,v) \in E \big| u \notin \mathsf{Tip}(V,E)\}\big) \ || \ \mathsf{Lex}\big(\mathsf{Tip}(V,E)\big), \end{cases}$$

where ϵ *stands for the empty sequence,* $\mathsf{Tip}(V, E)$ *for the set of all vertices of the DAG* (V, E) *with no in-degree,* $\mathsf{Lex}(S)$ *for the sorted sequence (in the lexicographic order) of elements of the set* S*.*

Clearly, the RBF of any increment tree is well-defined. Finally, we come to the total ordering of all transaction nodes in the DAG, which is the concatenation of all confirmed linear logs of all key nodes.

Definition 6 (Total Order in An Admissible DAG). *We assume an admissible DAG $G = (V, E)$ with backbone units $V_{bone} = \{u_1, u_2, \ldots, u_\ell\}$ that $(u_{i+1}, u_i) \in E$ holds for each integer $i < \ell$. The total order of each vertex is defined as its position in the sequence*

$$\mathsf{Total}(G) = \mathsf{RBF}(\mathsf{Inc}(u_1))||\mathsf{RBF}(\mathsf{Inc}(u_2))||\ldots||\mathsf{RBF}(\mathsf{Inc}(u_\ell))$$

if it is included in $\mathsf{Inc}(u_i)$ for any $i \in [\ell]$, or infinity in the other case.

To facilitate further descriptions and analyses, we define several operations and relations on graphs. Let A and B be two graphs (not necessarily DAGs),

$$A \sqsubseteq B \iff A.V \subseteq B.V \bigwedge A.E \subseteq B.E \bigwedge (\forall u.\forall v.u \in A.V \Rightarrow (u,v) \in B.E \Rightarrow (u,v) \in A.E)\,.$$

Let $A = (A.V, A.E)$ and $B = (B.V, B.E)$ be two pairs of a vertex set and an edge set, $A \sqcup B := (A.V \cup B.V, A.E \cup B.E)$. It is easy to observe that "$\sqsubseteq$" is a partial order that satisfies transitivity. In later proofs, we will utilize a few lemmas in Appendix. B.

4 Security Analysis

4.1 Execution Model

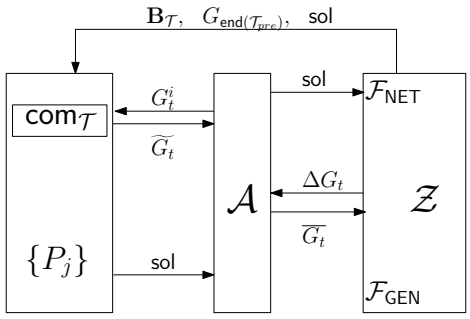


Fig. 3: The Execution Model

We provide an abstraction of the execution of our protocols. Recall that we have assumed a network with delays bounded by δ time steps. Since the intra-committee routing can be improved, we assume that the execution interval of one PBFT is bounded by δ_c ($\delta_c < \delta$). We ideally regard the adversary as a single party $\mathcal{A}$ capable of participating in the mining with an α fraction of

total hash power in an attempt of controlling a $\frac{1}{3}$ fraction of committee slots and delaying message transfers in the whole network. The graph view of $\mathcal{A}$ is notated as G_t. We suppose that the environment $\mathcal{Z}(1^\kappa)$ directs a system genesis functionality $\mathcal{F}_{\mathsf{GEN}}$ and an ideal peer-to-peer network functionality $\mathcal{F}_{\mathsf{NET}}$ that delivers all broadcasted messages instantly (we blame $\mathcal{A}$ for all network delays). In particular, $\mathcal{Z}$ simulates the behavior of common nodes that add into the DAG new transaction nodes ΔG_t over time and watch the ledger via $\overline{G_t}$ from $\mathcal{F}_{\mathsf{NET}}$. To formalize the mining, we assume a random oracle $\mathcal{H}$. In this part, we assume a dependable public-key infrastructure and hence the dependability of all possible signature schemes and put certifications out of concern. Thereby the notation sol in this part can be treated merely as a pair of a nonce value and pseudo-identity. We assume that there exists a protocol $\Pi_{\mathsf{rand}}^{\mathsf{com}_\mathcal{T}}$ during which a committee with a 2/3 honest rate can negotiate a random number. To formalize network delays, we regard that (1) for each participant in the current committee $\mathsf{com}_\mathcal{T}$, the local view of the DAG is delivered from $\mathcal{A}$; (2) all updates to the DAG are firstly sent to $\mathcal{A}$ and then delivered to the committee and $\mathcal{Z}$ within at most δ steps (recall that δ is the upper bound of delay); (3) all solutions provided by miners in $\{P_j\}$ are firstly sent to $\mathcal{A}$ and delivered to all participants by $\mathcal{F}_{\mathsf{NET}}$ in δ steps. To formally pose constraints on $\mathcal{A}$, we define $(\mathcal{A}, \mathcal{Z})$-complaint execution (Fig. 3) as below.

Definition 7 ($(\mathcal{A}, \mathcal{Z})$-Complaint Execution). *In an $(\mathcal{A}, \mathcal{Z})$-complaint execution model of our system, we consider following behaviors of $\mathcal{A}$ and $\mathcal{Z}$.*

Initial Stage. If the bootstrapping is safe (notated as $\mathbb{I}_{GEN}$), $\mathcal{F}_{GEN}$ outputs the initial committee $\mathsf{com}_{\mathcal{T}_0}$ for the initial epoch $\mathcal{T}_0$. Let $\mathsf{solpool} := \emptyset$, $G_0 := (\emptyset, \emptyset)$.

Epoch. For each epoch $\mathcal{T}$, the epoch block $\boldsymbol{B}_\mathcal{T}$ and the DAG $G_{\mathsf{begin}(\mathcal{T})} := G_{\mathsf{end}(\mathcal{T}_{pre})}$ (for $\mathcal{T} = \mathcal{T}_0$, $G_{\mathsf{begin}(\mathcal{T})} := 0$) are at first sent to $\{P_j\}$ by $\mathcal{F}_{NET}$. Afterwards, $\mathcal{A}$ and $\mathcal{Z}$ act as follows for each time $t \in [\mathsf{begin}(\mathcal{T}), \mathsf{end}(\mathcal{T})]$.

1. *Let $\mathsf{deliversol} \subseteq \mathsf{solpool}$ be the set of solutions of $\mathcal{A}$ to be delivered to the network $\mathcal{F}_{NET}$ such that*

$$\forall(\mathsf{sol}', t') \in (\mathsf{solpool} \setminus \mathsf{deliversol}).\ \ t - t' < \delta.$$

   Pended solutions $\{\mathsf{sol} | (\mathsf{sol}, t) \in \mathsf{deliversol}\}$ are sent to $\mathcal{F}_{NET}$ and broadcast to $\{P_j\}$ by $\mathcal{F}_{NET}$ if it is not empty. Then, if $\mathcal{A}$ receives a new valid solution to the hash puzzle sol, let $\mathsf{solpool} := \mathsf{solpool} \cup (\mathsf{sol}, t)$.
2. *$\mathcal{Z}$ sends to $\mathcal{A}$ a set of newly added transaction nodes $\Delta G_t = (\Delta V_t, \Delta E_t)$, where $\Delta E_t \subseteq \Delta V_t \times (V_{t-1} \cup \Delta V_t)$ ($\Delta E_t = \emptyset$ if $t = 0$). Let $V_t := V_{t-1} \cup \Delta V_t$ ($V_0 := \Delta V_0$), the view of $\mathcal{A}$ is updated $G_t := G_{t-1} \sqcup \Delta G_t$ ($G_0 := \Delta G_0$). Then, $\mathcal{A}$ sets $G_t^i \sqsubseteq G_t$ for each $P_i \in \mathsf{com}_\mathcal{T}$ such that*

$$\left(t > 0 \Rightarrow G_{t-1}^i \sqsubseteq G_t^i\right) \bigwedge \left(t \geq \delta \Rightarrow \Delta G_{t-\delta} \sqsubseteq G_t^i\right) \bigwedge \left(G_t^i \sqsubseteq \sqcup_{k=0}^t \Delta G_k\right).$$

   Afterwards, G_t^i is sent to P_i for each $P_i \in \mathsf{com}_\mathcal{T}$.

3. *There is a $\widetilde{G_t}$ returned from $\mathsf{com}_{\mathcal{T}}$ updating the graph of verified transactions. If over $2/3$ committee members are honest, then they share a common view of the DAG before time of δ (i.e. $G^i_{t-\delta} \sqsubseteq G^j_t$ for all members $P_i, P_j \in \mathsf{com}_{\mathcal{T}}$), it guarantees that there is a $w \leq \delta_c$ that $\widetilde{G_{t+w}}$ is returned from $\mathsf{com}_{\mathcal{T}}$ at time $t+w$ that extends $\widetilde{G_t}$ ($\widetilde{G_{t+w}} \neq \widetilde{G_t} \ \wedge \ |\widetilde{G_{t+w}}.V_{bone}| = |\widetilde{G_t}.V_{bone}| + 1$). Moreover, $G^i_{t-\delta-\delta_c} \sqsubseteq \widetilde{G_t}$. This corresponds to the protocol that the proposal of a newly added key node is asked to include all orphan nodes in its view δ time ago. Moreover, $\widetilde{G_t}$ should be in the form of*

$$\widetilde{G_t} = \widetilde{G_{t-1}} \sqcup (\{u, v_1, v_2, \ldots, v_\ell\}, \{(u, \hat{v}_1), (u, \hat{v}_2), \ldots, (u, \hat{v}_{\hat{\ell}})\}) \tag{1}$$

that u is the newly added key unit, $v_i \notin \widetilde{G_{t-1}}.V$ for all $i \in [\ell]$ and $v \in \{v_i\}_{i=1}^{\ell}$ for all $v \in \{\hat{v}_i\}_{i=1}^{\hat{\ell}}$. Each vertex $v \in \{v_i\}_{i=1}^{\ell}$ should be reachable from u via vertices of $\{v_i\}_{i=1}^{\ell}$.
4. *If $\mathcal{A}$ receives from $\mathsf{com}_{\mathcal{T}}$ an updated graph $\widetilde{G_t}$, it updates $G_t := G_{t-1} \sqcup \widetilde{G_t}$. Then, $\mathcal{A}$ selects a directed graph $\overline{G}_t \sqsubseteq G_t$ that*

$$\mathsf{FAD}(\overline{G}_t) \bigwedge (t > 0 \Rightarrow \overline{G}_{t-1} \sqsubseteq \overline{G}_t) \bigwedge (t \geq \delta \Rightarrow \widetilde{G_{t-\delta}} \sqsubseteq \overline{G}_t) \tag{2}$$

and sends it to $\mathcal{Z}$. Due to the significant interval from the committee switchover, for $t = \mathsf{begin}(\mathcal{T})$, we ask $\overline{G}_t := G_{t-1}$.
5. *$\mathcal{A}$ queries $\mathcal{H}$ with its spawned identity for $\alpha\Omega$ times (with $\alpha < \frac{1}{3} - \epsilon$ for a marginal ϵ) and packs each one smaller than the target into a solution sol and send it to $\mathcal{F}_{\mathsf{NET}}$.*

Switchover. The switchover protocol starts by $\mathsf{com}_{\mathcal{T}}$ instantaneously by the ℓ^{th} key node proposal. At this point, $\mathcal{A}$ and $\mathcal{Z}$ act as follows.

1. *$\mathcal{A}$ sends the full graph $G_{\mathsf{end}_{\mathcal{T}}}$ to $\mathcal{Z}$.*
2. *Upon receiving the next epoch block $\boldsymbol{B}_{\mathcal{T}_{nxt}}$ (if $\mathsf{com}_{\mathcal{T}}$ has a $2/3$ honest rate, $\boldsymbol{B}_{\mathcal{T}_{nxt}}$ includes λk nonces and determines the next committee as proposers of k random nonces among them) generated by $\mathsf{com}_{\mathcal{T}}$, it sends $\boldsymbol{B}_{\mathcal{T}_{nxt}}$ to $\mathcal{F}_{\mathsf{NET}}$ and so forth broadcast to all participants. We consider two steps above as atomic actions without delay. The same effect can be emulated by practice by having all honest participants pend for δ.*
3. *Let $\mathsf{begin}(\mathcal{T}_{nxt}) := t+\delta$. To guarantee consistency, set $G_t := G_{\mathsf{end}(\mathcal{T})}$ for each t that $\mathsf{end}(\mathcal{T}) < t < \mathsf{begin}(\mathcal{T}_{nxt})$. The new committee determined by $\boldsymbol{B}_{\mathcal{T}_{nxt}}$ receives from $\mathcal{A}$ $\mathsf{begin}(\mathcal{T}_{nxt})$ and starts consensus protocols by $\mathsf{begin}(\mathcal{T}_{nxt})$. Set $\mathcal{T} \leftarrow \mathcal{T}_{nxt}$.*

Note that ΔG_t, G^i_t are essentially set-theoretic graph segments instead of actual graphs because some edges may connect vertices out of the vertex set. In the model, $\mathcal{Z}$ simulates the view of all network nodes, the bootstrapping, and the partially synchronized network together with $\mathcal{A}$. $\mathcal{A}$ simulates all network delays and an irrational adversary that attempts to take over $1/3$ slots of the next committee. The view of all participants is regarded as a DAG since all vertices with outward edges pointing to unreceived nodes should be stored in a pool instead of entering the view.

4.2 Security Goals

The best way of showing the security of our protocols is to prove the universally composable (UC) security, however, this is unpractical due to the complication of our protocols. Thereby, our proof focuses on two abstracted key properties, namely, consistency and liveness[6] in the abstracted execution model.

– *Consistency.* The consistency property asks that for any large polynomial $\mathsf{poly}(\cdot)$,

$$\mathsf{Total}(\overline{G}_s) \preccurlyeq \mathsf{Total}(\overline{G}_t)$$

holds for all $0 < s \leq t \leq \mathsf{end}(\mathcal{T}_{\mathsf{poly}(\kappa)})$ except for a probability negligible in κ, we formalize this by

$$\Pr\left[\mathsf{view} \xleftarrow{\$} \mathsf{EXEC}^{\Pi}(1^{\kappa}, \mathcal{A}, \mathcal{Z}) : \mathbb{I}_{\mathsf{consistency}}(\mathsf{view}, 1^{\kappa}, \mathcal{A}, \mathcal{Z})\right] = 1 - \mathsf{negl}(\kappa).$$

– *Liveness.* Likewise, the liveness property asks that for any large polynomial $\mathsf{poly}(\cdot)$,

$$\Delta G_t.V \subseteq \{\mathsf{Total}(\overline{G}_{t+3\delta+\delta_c})\}$$

holds for all $0 < t \leq \mathsf{end}(\mathcal{T}_{\mathsf{poly}(\kappa)})$ except for a probability negligible in κ, we formalize this by

$$\Pr\left[\mathsf{view} \xleftarrow{\$} \mathsf{EXEC}^{\Pi}(1^{\kappa}, \mathcal{A}, \mathcal{Z}) : \mathbb{I}_{\mathsf{liveness}}(\mathsf{view}, 1^{\kappa}, \mathcal{A}, \mathcal{Z})\right] = 1 - \mathsf{negl}(\kappa).$$

For simplicity, we omit all parameters of $\mathbb{I}_{\mathsf{consistency}}(\mathsf{view}, 1^{\kappa}, \mathcal{A}, \mathcal{Z})$ and $\mathbb{I}_{\mathsf{liveness}}(\mathsf{view}, 1^{\kappa}, \mathcal{A}, \mathcal{Z})$ in case of no ambiguity.

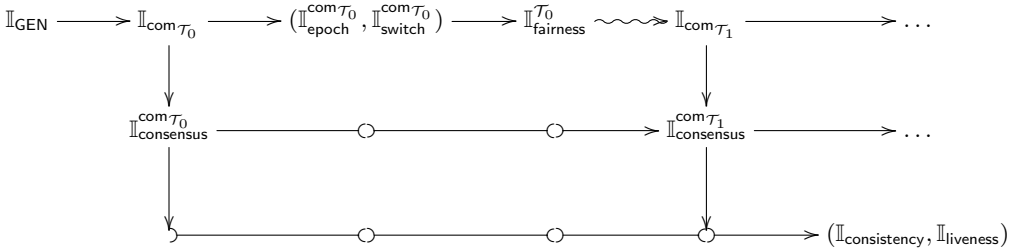


Table 1: Roadmap of Our Proof (Lemma.4-10)

[6] Note that though we have leveraged plenty UC-liked notations to facilitate descriptions, our proof is not done under the UC model.

4.3 The Proof

We break down the ultimate security goals to prove into subgoals defined as follows (for any epoch $\mathcal{T}$).

$\mathbb{I}_{\mathsf{com}_\mathcal{T}}$. At most $\llcorner k/3 \lrcorner$ slots of $\mathsf{com}_\mathcal{T}$ are controlled by $\mathcal{A}$.
$\mathbb{I}^{\mathsf{com}_\mathcal{T}}_{\mathsf{epoch}}$. Epoch $\mathcal{T}$ terminates at the end of $\mathcal{T}$ and outputs the set of all valid solutions broadcast by all participants of this epoch $\mathsf{sol}_\mathcal{T}$.
$\mathbb{I}^{\mathsf{com}_\mathcal{T}}_{\mathsf{switch}}$. Given a common input of a set of solutions $\mathsf{sol}_\mathcal{T}$, the committee outputs the next committee $\mathsf{com}_{\mathcal{T}_{nxt}}$ randomly selected from participants according to their numbers of proposed solutions in $\mathsf{sol}_\mathcal{T}$.
$\mathbb{I}^{\mathcal{T}}_{\mathsf{fairness}}$. With a current committee $\mathsf{com}_\mathcal{T}$, we say that the fairness of a GPoW scheme is achieved iff it holds that for each party contributing γ fraction of total hash rate within epoch $\mathcal{T}$, the next committee $\mathsf{com}_{\mathcal{T}_{nxt}}$ includes $(\gamma \pm o(1)) \cdot k$ members of its in expectation.
$\mathbb{I}^{\mathsf{com}_\mathcal{T}}_{\mathsf{consensus}}$. The following properties hold for every time point $t \in [\mathsf{begin}(\mathcal{T}), \mathsf{end}(\mathcal{T})]$.
- There is a $w \leq \delta_c$ that $\widetilde{G_{t+w}}$ is returned from $\mathsf{com}_\mathcal{T}$ at time $t+w$ that extends $\widetilde{G_t}$ ($\widetilde{G_{t+w}} \neq \widetilde{G_t}$).
- For any $t' < t - 2\delta - \delta_c$, $\Delta G_{t'} \sqsubseteq G_t$.
- $\widetilde{G_{t-1}} \sqsubseteq \widetilde{G_t}$.
- $\widetilde{G_t}$ is a fully admissible DAG.

Proofs of all above subgoals of each epoch are moved to Appendix. B. Tab. 1 provides a roadmap showing the implication relation between these subgoals of each epoch. Some of these subgoals can be mutually derived deterministically as shown by normal arrows. The entailment from the fairness of PoW in an epoch to the safety of the next committee is probabilistic, with an overwhelming probability. Therefore, this step of implication is marked by a curved arrow. Following this roadmap, we have finally proved the consistency and the liveness of Haootia as below (with proofs in Appendix. B).

Theorem 1. *$(\mathcal{A}, \mathcal{Z})$-compliant execution guarantees both consistency and liveness with a safe bootstrapping.*

5 Performance Evaluation

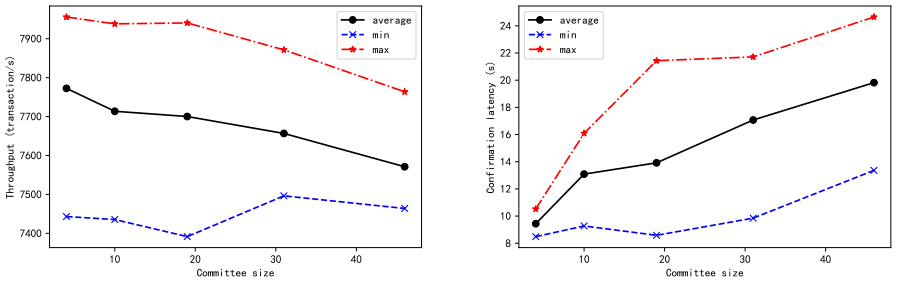
To evaluate the performance of Haootia consensus protocol, we fully implement Haootia with `Golang` (version 1.10). Without regard to unit tests and Haootia clients, our implementation consists of total 33349 lines of codes, where 4746 lines are responsible for the committee's PBFT module. Other important modules related to the protocol itself include P2P network, committee election, ledger verification, and total-ordering, etc.

During the experiment for performance, we deploy Haootia on remote servers which locates at various places across the world. All servers are AWS EC2 t2.xlarge instances, where each of them is equipped with 4vCPU and 16GB RAM. In every single experiment, all committee nodes start simultaneously, set

up a connected network, process a large number of transactions, and meanwhile execute the consensus algorithm.

As we focus on the evaluation of blockchain consensus, we conduct multiple modifications on the real-world implementation to get rid of unnecessary computation and network communication. Specifically, since the bottleneck of Haootia's performance lies in the committee's election process (GPoW-based election) and consensus process (PBFT), we set up only committee nodes to participate in consensus-related procedures, while nodes of individual miners and transaction producers are not included. Besides, to effectively simulate the large volume of transactions, the transactions used in the experiments are prepared in advance on servers. The technique of prepackaged transactions can bypass the limitation of network bandwidth on AWS servers. Finally, we also disable the signature verification module, which generally occupies CPU cycles a lot while it is irrelevant to the consensus process in essence.

We adopt two metrics throughout our experiments, throughput, and confirmation time, which are the most critical and universal indicators adopted to assess the liveness and scalability of a blockchain system. Notice that since committee size $|\mathsf{com}_{\mathcal{T}}|$ may affect system behaviours, we range committee size $|\mathsf{com}_{\mathcal{T}}|$ in $\{4, 10, 19, 31, 46\}$ and observe its effect on the overall performance of Haootia. For each choice of committee size, the experiment is repeated for 10 times to ensure the credibility of the statistic. The experiment results are listed as follows.



(a) Throughput (b) Confirmation Time

Fig. 4: Throughput and confirmation time of Haootia under different committee sizes

Throughput & Comparisons. Throughput denotes the maximum capability for the system to consume transactions. In the experiment for Haootia, the throughput is counted by the volume of confirmed units (transactions) in a period when the consensus is stably running. The boxplot in Fig. 4(a) demonstrates the average, minimum, and maximum throughput of Haootia under different com-

mittee sizes. We could observe that throughput of Haootia is around 7500 TPS (transaction per second) with a confirmation latency around 20 seconds by a committee of size 46, and its fluctuation not strongly related to the committee size. Compared with other common DAG-based consensus protocols such as GHOST (about 275 TPS [4]) and Nano (about 306 TPS [33]), the scalability of Haootia improves by one order of magnitude. Also, compared with existing non-DAG-based protocols like Byzcoin (about one thousand TPS [16]) or Elastico (about 2500 TPS [23]), our scheme has an overwhelming throughput.

Confirmation Time. Confirmation time, namely confirmation latency, denotes the client-side latency of processing a transaction. In the experiment, confirmation time is the interval from the committee receiving a transaction to a key unit confirming the transaction. Similarly, the boxplot in Fig. 4(b) illustrates the average, minimum, and maximum confirmation time of Haootia under different committee sizes. The key observation of confirmation latency is that it is positively correlated with by committee size, ranging from about $10s$ to $25s$, which we ascribe to the increased BFT burden among committee members. Such confirmation performance can be regarded as instant confirmation, which implies the strong liveness of Haootia.

6 Conclusion

In this work, we have proposed a novel consensus scheme for a DAG-based distributed ledger which achieves both consistency and liveness within any polynomially large number of epochs with an overwhelming probability. In the future, we look forward to having this scheme applied to multiple scenarios of our real world. Also, we hope to have our scheme proved fully under the UC model where the fact that our ideal world has successfully simulated the real world is formally entailed. Appendix. C provides a rough explanation of the incentive-compatibility of our scheme. We also expect to have the incentives of this scheme rigorously and thoroughly analyzed.

References

1. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
2. Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.
3. Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.
4. Chenxing Li, Peilun Li, Wei Xu, Fan Long, and Andrew Chi-chih Yao. Scaling nakamoto consensus to thousands of transactions per second. *arXiv preprint arXiv:1805.03870*, 2018.
5. Anton Churyumov. Byteball: A decentralized system for storage and transfer of value, 2016. https://byteball.org/Byteball.pdf.
6. Serguei Popov. The tangle. https://iota.org/IOTA_Whitepaper.pdf.

7. Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.
8. Yonatan Sompolinsky and Aviv Zohar. PHANTOM: A scalable blockdag protocol. *IACR Cryptology ePrint Archive*, 2018:104, 2018.
9. Iddo Bentov, Pavel Hubácek, Tal Moran, and Asaf Nadler. Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies. *IACR Cryptology ePrint Archive*, 2017:300, 2017.
10. Xavier Boyen, Christopher Carr, and Thomas Haines. Graphchain: a blockchain-free scalable decentralised ledger. In *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts, BCC@AsiaCCS 2018, Incheon, Republic of Korea, June 4, 2018*, pages 21–33, 2018.
11. Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds, Inc. Technical Report SWIRLDS-TR-2016*, 1, 2016.
12. Colin LeMahieu. Nano: A feeless distributed cryptocurrency network. https://nano.org/en/whitepaper.
13. Tai-Yuan Chen, Wei-Ning Huang, Po-Chun Kuo, Hao Chung, and Tzu-Wei Chao. Dexon: A highly scalable, decentralized dag-based consensus algorithm. *arXiv preprint arXiv:1811.07525*, 2018.
14. Ethan Heilman, Neha Narula, Garrett Tanzer, James Lovejoy, Michael Colavita, Madars Virza, and Tadge Dryja. Cryptanalysis of curl-P and other attacks on the IOTA cryptocurrency. *IACR Cryptology ePrint Archive*, 2019:344, 2019.
15. Marko Vukolic. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*, pages 112–125, 2015.
16. Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, 2016.
17. Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 39:1–39:16, 2017.
18. James N Gray. Notes on data base operating systems. In *Operating Systems*, pages 393–481. Springer, 1978.
19. Juan Garay and Aggelos Kiayias. SoK: a consensus taxonomy in the blockchain era. Technical report, Cryptology ePrint Archive, Report 2018/754, 2018.
20. Flavio P. Junqueira, Benjamin C. Reed, and Marco Serafini. Zab: High-performance broadcast for primary-backup systems. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 245–256. IEEE, 2011.
21. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In Margo I. Seltzer and Paul J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999.
22. John R. Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.
23. Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

24. Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.
25. Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 931–948, 2018.
26. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 3–33, 2018.
27. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.
28. Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 515–532. Springer, 2016.
29. Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 585–602. ACM, 2019.
30. Lei Yang, Vivek Kumar Bagaria, Gerui Wang, Mohammad Alizadeh, David Tse, Giulia C. Fanti, and Pramod Viswanath. Prism: Scaling bitcoin by 10, 000x. *CoRR*, abs/1909.11261, 2019.
31. Georgios Birmpas, Elias Koutsoupias, Philip Lazos, and Francisco J. Marmolejo Cossío. Fairness and efficiency in dag-based cryptocurrencies. *CoRR*, abs/1910.02059, 2019.
32. Rafael Pass and Elaine Shi. Rethinking large-scale consensus. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 115–129, 2017.
33. Federico Matteo Bencic and Ivana Podnar Zarko. Distributed ledger technology: Blockchain compared to directed acyclic graph. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*, pages 1569–1570, 2018.
34. Miguel Castro, Barbara Liskov, et al. A correctness proof for a practical byzantine-fault-tolerant replication algorithm. Technical report, Technical Memo MIT/LCS/TM-590, MIT Laboratory for Computer Science, 1999.
35. Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 427–437, 1987.

A Detailed Protocols

A.1 Design Components

Byzantine Fault-tolerance as a Subroutine. Haootia adopts PBFT [21, 34] protocol to reach consensus among $\mathsf{com}_{\mathcal{T}}$. The PBFT protocol executes in

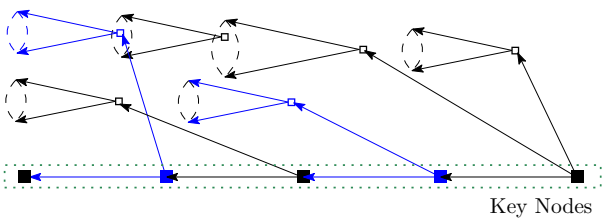


Fig. 5: The Recursion Tree

the partially synchronous network and guarantees safety and liveness provided at most $\lfloor \frac{n-1}{3} \rfloor$ out of total n replicas are simultaneously faulty. For practical considerations, we modify the client-server model which requires all the replicas to reply to the client by asking only the primary to broadcast the final result containing $2f + 1$ valid signatures from $\mathsf{com}_{\mathcal{T}}$ to $\mathcal{Z}$ in the representation of the whole committee.

For the purpose of protocol description, we assume $k = 3f + 1$, where k is committee size $|\mathsf{com}_{\mathcal{T}}|$ and f is the maximum number of faulty members that PBFT can tolerate. All members are ordered according to their position in $\mathsf{com}_{\mathcal{T}}$ and identified by a unique integer in $[k]$. There is a succession of configurations called views, the PBFT primary for view v is p such that $\mathsf{p} = \mathsf{v} \mod k$, and all others are backups. We call the whole committee $\mathsf{com}_{\mathcal{T}}$ as replicas under PBFT setting.

Generally, each execution of PBFT protocol is initiated by the primary p by multicasting to all backups a message to reach consensus on. Under normal circumstances, after a three-phase interaction, the $\mathsf{com}_{\mathcal{T}}$ will reach agreement and output a final result containing $2f+1$ valid signatures from $\mathsf{com}_{\mathcal{T}}$ indicating its validity (shown in Algo. 1). The liveness of PBFT is guaranteed by the view-change protocol in case of a faulty primary, which is triggered by timeouts or invalid messages are received from the primary by backups. In this case, backups will start a view change process (depicted in Algo. 2) for reconfiguration until a new primary is elected.

As is a mature tool, we will regard PBFT and its view change subroutine as a black box assuring both safety and liveness. Each time PBFT is called, it's given inputs, which are later used by primary to propose messages and backups to check the validity of the message. After its execution, a final result of the specified form will be broadcast to $\mathcal{Z}$. To resolve the censorship issue, our protocol asks them to take turns to serve as the primary.

For a tuple $\langle m_0, m_1, \ldots \rangle$, $\langle m_0, m_1, \ldots \rangle_{\sigma_{\mathsf{P}}}$ denotes the tuple appended with a signature on the tuple from P. Due to the nontriviality, for (and only for) the descriptions of the protocols related to PBFT, we explicitly describe signatures appended to each message tuple.

Random Number Negotiation. We assume an epoch randomness negotiation protocol $\Pi_{\mathsf{rand}}^{\mathsf{com}_{\mathcal{T}}}$ that allows each committee to securely generate a randomness.

Algorithm 1 The PBFT Protocol

Protocol $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{PBFT}}$

The PBFT protocol is executed by all k members of $\mathsf{com}_{\mathcal{T}}$, with each identified by a unique integer in $\{0, \cdots, k-1\}$. There are three phases *pre-prepare, prepare* and *commit*. It proceeds as follows.

1. *Pre-prepare*. Primary p:
   (a) Multicasts to all backups a $\left\langle \langle \texttt{pre-prepare}, \mathcal{T}, \mathsf{v}, \mathsf{d} \rangle_{\sigma_\mathsf{p}}, \mathsf{msg} \right\rangle$ message. Here v indicates the current view, msg is the message to reach a consensus on, and d is msg's digest

   Backup i:
   (a) When receives the $\left\langle \langle \texttt{pre-prepare}, \mathcal{T}, \mathsf{v}, \mathsf{d} \rangle_{\sigma_\mathsf{p}}, \mathsf{msg} \right\rangle$ message, it checks whether the following conditions hold:
   – The signature is correct and d is the digest for msg
   – It is in view v
   – msg proposed by p is valid
   (b) If any of the above validation fails:
   – Executes $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{view\text{-}change}}$ protocol with parameter v
   (c) Else:
   – Multicasts a $\left\langle \texttt{prepare}, \mathcal{T}, \mathsf{v}, \sigma^{\mathsf{d}}_{\mathsf{i}}, \mathsf{i} \right\rangle_{\sigma_\mathsf{i}}$ message to all other replicas. Here $\sigma^{\mathsf{d}}_{\mathsf{i}}$ is its signature on d
   – Starts a timer T
2. *Prepare*. Replica i:
   (a) For each $\left\langle \texttt{prepare}, \mathcal{T}, \mathsf{v}, \sigma^{\mathsf{d}}_{\mathsf{i}}, \mathsf{i} \right\rangle_{\sigma_\mathsf{i}}$ message received from other replicas:
   – Checks the validity of it, i.e., whether the signature is valid and has the same view and digest as the pre-prepare message
   (b) If it has not accepted $2f+1$ valid prepares (including its own) from different replicas until T expires:
   – Executes $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{view\text{-}change}}$ protocol with parameter v
   (c) Else:
   – Stops the timer T
   – Multicasts a $\left\langle \texttt{commit}, \mathcal{T}, \mathsf{v}, \sigma^{\mathsf{d}}_{\mathsf{i}}, \mathsf{i} \right\rangle_{\sigma_\mathsf{i}}$ message to all other replicas
3. *Commit*. Primary p:
   (a) Signature set $\sigma_{\mathsf{com}_{\mathcal{T}}} \leftarrow \emptyset$
   (b) Adds its own signature $\sigma^{\mathsf{d}}_{\mathsf{p}}$ on d to $\sigma_{\mathsf{com}_{\mathcal{T}}}$
   (c) For each $\left\langle \texttt{commit}, \mathcal{T}, \mathsf{v}, \sigma^{\mathsf{d}}_{\mathsf{i}}, \mathsf{i} \right\rangle_{\sigma_\mathsf{i}}$ message received, checks the signature and digest, and adds $\sigma^{\mathsf{d}}_{\mathsf{i}}$ to $\sigma_{\mathcal{T}}$ if it is valid
   (d) If it has received $2f+1$ valid commit messages (including its own) from different replicas, broadcasts a $\langle \texttt{end\_PBFT}, \mathcal{T}, \mathsf{v}, \mathsf{msg}, \sigma_{\mathsf{com}_{\mathcal{T}}} \rangle_{\sigma_\mathsf{p}}$ message to the network

   Backup i:
   (a) If it has received $2f+1$ commit messages (including its own) that match the prepare message, i.e., they have the same view and digest, starts a timer T
   (b) If it has not received a valid $\langle \texttt{end\_PBFT}, \mathcal{T}, \mathsf{v}, \mathsf{msg}, \sigma_{\mathsf{com}_{\mathcal{T}}} \rangle_{\sigma_\mathsf{p}}$ message until T expires, executes $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{view\text{-}change}}$ protocol with parameter v

Algorithm 2 The View Change Protocol

Protocol $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\text{view-change}}$

The view-change protocol takes as input the current view v and returns a new v'. For replica i, the following steps are iterated each time $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\text{view-change}}$ is executed.

1. $\mathsf{v} \leftarrow \mathsf{v}+1$, $\mathsf{p} \leftarrow \mathsf{v} \mod k$
2. If $\mathsf{i} \neq \mathsf{p}$:
   (a) Multicasts a $\langle \texttt{view-change}, \mathcal{T}, \mathsf{v}, \mathsf{i} \rangle_{\sigma_\mathsf{i}}$ message to all replicas
3. If $\mathsf{i} = \mathsf{p}$, i.e., it is the primary for v:
   (a) View change set $\mathcal{V}_{\mathsf{com}_{\mathcal{T}}} \leftarrow \emptyset$
   (b) Adds its own view change message $\langle \texttt{view-change}, \mathcal{T}, \mathsf{v}, \mathsf{i} \rangle_{\sigma_\mathsf{p}}$ to $\mathcal{V}_{\mathsf{com}_{\mathcal{T}}}$
   (c) For each $\langle \texttt{view-change}, \mathcal{T}, \mathsf{v}, \mathsf{i} \rangle_{\sigma_\mathsf{i}}$ tuple received, it checks the view and signature, and adds to $\mathcal{V}_{\mathsf{com}_{\mathcal{T}}}$ if it is valid
   (d) Once receiving $2f+1$ valid view-change messages (including its own) from different replicas, it multicasts a $\langle \texttt{new-view}, \mathcal{T}, \mathsf{v}, \mathcal{V}_{\mathsf{com}_{\mathcal{T}}} \rangle_{\sigma_\mathsf{p}}$ message to all replicas
   (e) Returns v
4. If $\mathsf{i} \neq \mathsf{p}$:
   (a) When it has accepted $2f+1$ view-change tuples (including its own) from different replicas for v, it starts a timer T
   (b) It accepts the $\langle \texttt{new-view}, \mathcal{T}, \mathsf{v}, \mathcal{V}_{\mathsf{com}_{\mathcal{T}}} \rangle_{\sigma_\mathsf{p}}$ message for v provided:
      – It is sent by p
      – the signature is valid
      – all $2f+1$ view-change tuples are valid for v
   (c) Returns v if all of the above hold. If T expires before any valid new-view message is received, continues the loop.

This protocol can be realized by a distributed random generation (DRG) based on verifiable secret sharing (VSS) of [35].

Nextly, we will dive step-by-step into the Full protocol depicted in Fig. 2.

A.2 Haootia Consensus

Mining Protocol. Once receiving $\boldsymbol{B}_{\mathcal{T}}$ from $\mathcal{Z}$, miners start executing mining protocol $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{mine}}$, trying to find nonce solutions for the hash puzzle determined by $\boldsymbol{B}_{\mathcal{T}}$ in an effort to earn themselves a chance to enter $\mathsf{com}_{\mathcal{T}_{nxt}}$. During $\mathcal{T}$, each miner P enumerates a random value nc and check whether $H(\boldsymbol{B}_{\mathcal{T}}||\mathsf{pk}_P||\mathsf{nc}) \in \mathsf{target}$ holds. In case of satisfying, it broadcasts $\langle \texttt{solution}, \mathcal{T}, \mathsf{pk}_P, \mathsf{nc} \rangle$ to $\mathcal{Z}$ and continues. The detailed protocol is shown in Algo. 3.

Consensus Protocol. On receiving $\boldsymbol{B}_{\mathcal{T}}$ from $\mathcal{Z}$, participants of the network decode it and fetch current committee $\mathsf{com}_{\mathcal{T}}$, then $\mathsf{com}_{\mathcal{T}}$ as a whole execute consensus protocol $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{consensus}}$ and serves as an authority in the network during $\mathcal{T}$. It observes $\mathcal{Z}$ for newly coming transaction nodes in the DAG ledger and nonce solutions broadcast by miners, and continuously executes PBFT protocol $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{PBFT}}$ to generate key nodes referencing new tip nodes in the DAG as well as confirming new nonce solutions.

Intra the committee, members of $\mathsf{com}_{\mathcal{T}}$ take turns to serve as the primary and launch the PBFT process by proposing a key node referencing all the tip

Algorithm 3 The Mining Protocol

Protocol $\Pi_{\mathsf{mine}}^{\mathsf{com}_{\mathcal{T}}}$

The mining protocol is executed by all miners participating in the protocol, each of pseudo identity P proceeds as follows.

1. $x \leftarrow 0$
2. Check whether $H(\boldsymbol{B}_{\mathcal{T}}||\mathsf{pk}_P||x) \in \mathsf{target}$
   - If it holds, broadcast $\langle \texttt{solution}, \mathcal{T}, \mathsf{pk}_P, \mathsf{nc} = x \rangle$ to $\mathcal{Z}$
3. $x \leftarrow x + 1$, goto 2)

nodes in the DAG ledger δ ago, also this key node has a special field holding possibly new nonce solutions from miners. δ is the upper bound on the network delay, thereby all other backups can check the validity of this reference relations as they have the same view of the DAG ledger δ ago. Note here that the nonce solutions to enclose don't necessarily need to be δ ago, because each backup can check the validity of a solution directly from its basic information.

Each time $\Pi_{\mathsf{PBFT}}^{\mathsf{com}_{\mathcal{T}}}$ is invoked by $\Pi_{\mathsf{consensus}}^{\mathsf{com}_{\mathcal{T}}}$, a new key node $\langle \texttt{keynode}, \mathcal{T}, \mathsf{v}', \Delta G, \Delta S, \sigma_{\mathsf{com}_{\mathcal{T}}} \rangle$ is generated under the consensus of $\mathsf{com}_{\mathcal{T}}$. Here $\mathcal{T}$ denotes the current epoch, v' denotes the view number after the execution of $\Pi_{\mathsf{PBFT}}^{\mathsf{com}_{\mathcal{T}}}$ (note it might might not be the same with the initial one after each execution of $\Pi_{\mathsf{PBFT}}^{\mathsf{com}_{\mathcal{T}}}$ due to possible view changes), ΔG denotes new reference edges and the key node vertice itself, ΔS denotes newly confirmed nonce solutions, and $\sigma_{\mathsf{com}_{\mathcal{T}}}$ denotes the signature set from $\mathsf{com}_{\mathcal{T}}$ indicating this key node's validity.

From the upper DAG ledger point of view, this key node functions both for the convergence of the ledger and transaction linearization. Immediately when a transaction node is referenced directly or indirectly by a key node (according to ΔG), its total order in G is uniquely determined (see Def. 6). In the case of double-spending, if there is partial order between them, the later one is safely rejected from entering into G. For those with no partial order, the one who comes earlier in the total order is considered valid. Since PBFT is a determined consensus protocol, once a transaction node is referenced (directly or indirectly) by a key node, it reaches its finality, i.e., if it is valid now, it will never be double-spent in the future. Moreover, as the consensus on a key node can take place very fast among $\mathsf{com}_{\mathcal{T}}$, transaction confirmation in Haootia can be achieved within seconds.

Apart from transaction linearization in the DAG ledger, a key node also confirms newly generated nonce solutions (ΔS) by miners. As we will see later, when enough solutions are received by $\mathsf{com}_{\mathcal{T}}$, a switchover will take place and a new block indicating the next epoch is generated by $\mathsf{com}_{\mathcal{T}}$ accordingly. The detailed protocol execution is illustrated in Algo. 4.

A.3 Committee Switchover

Epoch protocol. At the beginning of epoch $\mathcal{T}$, an epoch protocol $\Pi_{\mathsf{epoch}}^{\mathsf{com}_{\mathcal{T}}}$ is launched along with the consensus protocol $\Pi_{\mathsf{consensus}}^{\mathsf{com}_{\mathcal{T}}}$ by each committee member. $\mathsf{com}_{\mathcal{T}}$ as

Algorithm 4 The Consensus Protocol

Protocol $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{consensus}}$

The consensus protocol of epoch $\mathcal{T}$ is executed by all k identities of $\mathsf{com}_{\mathcal{T}}$. Each identity proceeds as follows.

1. View number $\mathsf{v} \leftarrow 0$
2. Update its local DAG state G and solution set S from $\mathcal{Z}$
3. Iterate the following steps until this protocol is halted by $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{epoch}}$
   (a) Execute $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{PBFT}}$ with inputs v, G, and S, receive $\langle \mathtt{keynode}, \mathcal{T}, \mathsf{v}', \Delta G, \Delta S, \sigma_{\mathsf{com}_{\mathcal{T}}} \rangle$ from $\mathcal{Z}$ after its termination
   (b) $\mathsf{v} \leftarrow \mathsf{v}' + 1$
   (c) Update G and S from $\mathcal{Z}$

an entirety executes $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{epoch}}$ to collect nonce soluitons contained in $\langle \mathtt{keynode}, \mathcal{T}, \mathsf{v}', \Delta G, \Delta S, \sigma_{\mathsf{com}_{\mathcal{T}}} \rangle$ from $\mathcal{Z}$, and maintains a solution set $\mathsf{sol}_{\mathcal{T}}$ accoringly. Once λk solutions are received, all instances of $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{mine}}$ and $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{consensus}}$ are halted, and a switchover protocol $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{switch}}$ is invoked to generate a new block $\boldsymbol{B}_{\mathcal{T}_{nxt}}$ and the next committee $\mathsf{com}_{\mathcal{T}_{nxt}}$. The detailed protocol is shown in Algo. 5.

Algorithm 5 The Epoch Protocol

Protocol $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{epoch}}$

The epoch protocol is executed by all k identities of $\mathsf{com}_{\mathcal{T}}$. Each identity proceeds as follows.

1. Counter of nonce solutions $\mathsf{ct} \leftarrow 0$, solution set $\mathsf{sol}_{\mathcal{T}} \leftarrow \emptyset$
2. When receiving a tuple $\langle \mathtt{keynode}, \mathcal{T}, \mathsf{v}', \Delta G, \Delta S, \sigma_{\mathsf{com}_{\mathcal{T}}} \rangle$ from $\mathcal{Z}$ each time, iterate the following steps:
   (a) $\mathsf{sol}_{\mathcal{T}} \leftarrow \mathsf{sol}_{\mathcal{T}} || \Delta S$
   (b) $\mathsf{ct} \leftarrow \mathsf{ct} + |\Delta S|$
   (c) if $\mathsf{ct} \geq \lambda k$, end the loop
3. Halt $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{mine}}$, and get the latest v' before halting $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{consensus}}$. Then launch $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{switch}}$ with input $\mathsf{sol}_{\mathcal{T}}$ and v'

Switch Protocol. After a solution set $\mathsf{sol}_{\mathcal{T}}$ of size λk are received and confirmed by $\mathsf{com}_{\mathcal{T}}$, one last thing for $\mathsf{com}_{\mathcal{T}}$ is to reach agreement on $\mathsf{com}_{\mathcal{T}_{nxt}}$ and generate $\boldsymbol{B}_{\mathcal{T}_{nxt}}$ accordingly. The key idea here is that k lucky solutions are chosen uniformly randomly from $\mathsf{sol}_{\mathcal{T}}$, and then the next committee $\mathsf{com}_{\mathcal{T}_{nxt}}$ is determined as the proposers of these lucky solutions.

To begin with, a random number r is negotiated using a $\Pi^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{rand}}$ protocol. With this random number, k positions in $\mathsf{sol}_{\mathcal{T}}$ can be uniquely determined via a hash function, and the corresponding miners who proposed these solutions are chosen as committee members of $\mathsf{com}_{\mathcal{T}_{nxt}}$. Specifically, we use $Hash(r||i) \mod |\mathsf{sol}_{\mathcal{T}}|$

to locate solutions for each i in $[k]$, where the hash function is assumed to be uniform random. As can be seen, there are chances that one specific solution is chosen twice or more times, this is allowed and the corresponding miner (pk) is regarded to have different identities in $\mathsf{com}_{\mathcal{T}_{nxt}}$ identified by its unique $\langle \mathsf{pk}, loc \rangle$, where pk is miner's public key and loc is its position in $\mathsf{com}_{\mathcal{T}_{nxt}}$.

After that, $\Pi_{\mathsf{PBFT}}^{\mathsf{com}_{\mathcal{T}}}$ is called by $\Pi_{\mathsf{switch}}^{\mathsf{com}_{\mathcal{T}}}$ for the $\mathsf{com}_{\mathcal{T}}$ to reach agreement on on a new block $\boldsymbol{B}_{\mathcal{T}_{nxt}}$ in the form of $\langle \boldsymbol{B}, \mathcal{T}, \mathsf{v}', \mathsf{sol}_{\mathcal{T}}, \mathsf{com}_{\mathcal{T}_{nxt}}, \sigma_{\mathsf{com}_{\mathcal{T}}} \rangle$. Here $\mathcal{T}$ denotes the current epoch, v' denotes the view number after the execution of $\Pi_{\mathsf{PBFT}}^{\mathsf{com}_{\mathcal{T}}}$, $\mathsf{sol}_{\mathcal{T}}$ denotes the solution set formed in epoch $\mathcal{T}$, $\mathsf{com}_{\mathcal{T}_{nxt}}$ denotes the committee members of the next committee, so that participants in epoch $\mathcal{T}_{nxt}$ can fetch $\mathsf{com}_{\mathcal{T}_{nxt}}$ by decoding $\boldsymbol{B}_{\mathcal{T}_{nxt}}$, and $\sigma_{\mathsf{com}_{\mathcal{T}}}$ denotes the signature set from $\mathsf{com}_{\mathcal{T}}$ indicating this key node's validity. For the integrity of GPoW blockchain, the hash of previous block as well as some other metadata should also be included into a block, but for simplicity they are omitted here.

As long as $\boldsymbol{B}_{\mathcal{T}_{nxt}}$ is generated under the consensus of $\mathsf{com}_{\mathcal{T}}$, participants can see this update from $\mathcal{Z}$, and then a new epoch $\mathcal{T}_{nxt}$ begins. The detailed protocol is depicted in Algo. 6.

Algorithm 6 The Switchover Protocol

Protocol $\Pi_{\mathsf{switch}}^{\mathsf{com}_{\mathcal{T}}}$

The switchover protocol is executed by all k identities of $\mathsf{com}_{\mathcal{T}}$. A common input solution set $\mathsf{sol}_{\mathcal{T}}$ and view v derived from $\Pi_{\mathsf{epoch}}^{\mathsf{com}_{\mathcal{T}}}$ is shared by all k identities. Each identity proceeds as follows.

1. A random number r is negotiated via protocol $\Pi_{\mathsf{rand}}^{\mathsf{com}_{\mathcal{T}}}$
2. For each $i \in [k]$: $\mathsf{position}_i \leftarrow H(r||i) \mod |\mathsf{sol}_{\mathcal{T}}|$
3. For each $i \in [k]$: supposing $\mathsf{sol}_{\mathcal{T}}[\mathsf{position}_i] = \langle \mathtt{solution}, \mathcal{T}, \mathsf{pk}', \mathsf{nc} \rangle$, let $\mathsf{com}_{\mathcal{T}_{nxt}}[i] \leftarrow \mathsf{pk}'$
4. $\mathsf{v} \leftarrow \mathsf{v} + 1$
5. Executes $\Pi_{\mathsf{PBFT}}^{\mathsf{com}_{\mathcal{T}}}$ with input v, $\mathtt{sol}_{\mathcal{T}}$, $\mathsf{com}_{\mathcal{T}_{nxt}}$, waits till receiving $\langle \boldsymbol{B}, \mathcal{T}, \mathsf{v}', \mathsf{sol}_{\mathcal{T}}, \mathsf{com}_{\mathcal{T}_{nxt}}, \sigma_{\mathsf{com}_{\mathcal{T}}} \rangle$ from $\mathcal{Z}$

A.4 The Main Protocol

With all the components above, we are allowed to come into the final main protocol by arranging them together. As is shown in Algo. 7, at the beginning of each epoch $\mathcal{T}$, each participant of the network fetches $\boldsymbol{B}_{\mathcal{T}}$ and starts mining based on it (in case of an ordinary miner performing mining last epoch, it must stop the old mining instance and start a new one). If this participant happens to be one of the committee members at $\mathcal{T}$, it will also fork a protocol instance for both $\Pi_{\mathsf{consensus}}^{\mathsf{com}_{\mathcal{T}}}$ and $\Pi_{\mathsf{epoch}}^{\mathsf{com}_{\mathcal{T}}}$ if pk', and then play the role of committee members during $\mathcal{T}$ as mentioned above.

Algorithm 7 The Main Protocol

Protocol Π_{full}

The following steps are iterated each epoch $\mathcal{T}$ for each participant pk.

1. Receive from $\mathcal{Z}$ $\boldsymbol{B}_{\mathcal{T}}$, decode it and fetch the current committee list $\mathsf{com}_{\mathcal{T}}$
2. If any $\Pi_{\mathsf{mine}}^{\mathsf{com}_{\mathcal{T}}}$ instance is running, stop it and launch a new $\Pi_{\mathsf{mine}}^{\mathsf{com}_{\mathcal{T}}}$ based on $\boldsymbol{B}_{\mathcal{T}}$
3. For each identity $\mathsf{pk}' \in \mathsf{com}_{\mathcal{T}}$, fork a protocol instance for both $\Pi_{\mathsf{consensus}}^{\mathsf{com}_{\mathcal{T}}}$ and $\Pi_{\mathsf{epoch}}^{\mathsf{com}_{\mathcal{T}}}$ if pk' is a public key of itself

B More Proofs

Few lemmas on the DAG are useful to our security proofs. They are not proved since their proofs are obvious inductions. However, it is a necessity to list them here not only for rigorousness but also to facilitate the understanding of our proofs.

Lemma 1 $(\forall G.\forall H.\ \mathbf{FAD}(H) \Rightarrow (G \sqsubseteq H) \Rightarrow \mathbf{AD}(G))$**.** *For any fully admissible DAG H, all directed graphs G that $G \sqsubseteq H$ should be an admissible DAG.*

Lemma 2 $(\forall G.\forall H.\ (G \sqsubseteq H) \Rightarrow (\mathbf{Total}(G) \preccurlyeq \mathbf{Total}(H)))$**.** *For any two DAGs G and H that $G \sqsubseteq H$, the totally ordered list of vertices in G should be a prefix of the totally ordered list of vertices in H.*

We introduce notation $\{s\}$ to convert a sequence of elements s to a set. Obviously, with the nature of fully admissible DAGs where all vertices are reachable from key nodes, the total order sequence attained should cover all vertices in the graph.

Lemma 3 $(\forall G.\ \mathbf{FAD}(G) \Rightarrow \{\mathbf{Total}(G)\} = G.V)$**.** *The total ordering log of a fully admissible DAG contains exactly all vertices of it.*

Lemma 4 $(\mathbb{I}_{\mathsf{com}_{\mathcal{T}}} \Rightarrow (\mathbb{I}_{\mathbf{epoch}}^{\mathsf{com}_{\mathcal{T}}}, \mathbb{I}_{\mathbf{switch}}^{\mathsf{com}_{\mathcal{T}}}))$**.** *The 2/3 honesty of a committee $\mathsf{com}_{\mathcal{T}}$ implies the safety of next committee election and committee switchover.*

Proof. This is a direct result from the execution model. To explain this in real practice, this part of the protocol is merely an explicit dispatch of PBFT. Recall that a 2/3 honest rate has already provided the safety and liveness of the permissioned BFT itself. Thereby, the safety of $(\Pi_{\mathsf{epoch}}^{\mathsf{com}_{\mathcal{T}}}, \Pi_{\mathsf{switch}}^{\mathsf{com}_{\mathcal{T}}})$ is directed guaranteed (see more in Appendix. A).

Lemma 5 $((\mathbb{I}_{\mathbf{epoch}}^{\mathsf{com}_{\mathcal{T}}}, \mathbb{I}_{\mathbf{switch}}^{\mathsf{com}_{\mathcal{T}}}) \Rightarrow \mathbb{I}_{\mathbf{fairness}}^{\mathcal{T}})$**.** *To each epoch $\mathcal{T}$, the safety of next committee election and committee switchover implies the fairness of PoW in $\mathcal{T}$.*

Proof. Assuming that each query to $\mathcal{H}$ returns a satisfactory nonce (smaller than the target) with probability p. Due to the safety and liveness of PBFT and the assumption on $\Pi_{\mathsf{rand}}^{\mathsf{com}_{\mathcal{T}}}$, each nonce successfully delivered to $\mathsf{com}_{\mathcal{T}}$ shares the same probability of having its corresponding pseudo-identity enter the next committee.

We denote this probability as q. Due to the linearity of expectation, a party (denoted by P) with γ fraction of total hash rate in the worst network delay can take in expectation $\gamma\Omega(T-\delta)\cdot pq$ slots in $\mathsf{com}_{\mathcal{T}_{nxt}}$ since $\mathcal{A}$ at most delays a sol for time δ before sending it to $\mathcal{F}_{\mathsf{NET}}$ and so forth $\mathsf{com}_{\mathcal{T}}$. Likewise, in the worst case to this party, all other parties suffer zero delays and thereby take $(1-\gamma)\Omega T\cdot pq$ slots in expectation. Obviously, we have $pq = \frac{k}{\gamma\Omega(T-\delta)+(1-\gamma)\Omega T}$ and we can derive

$$\begin{aligned}\gamma\Omega(T-\delta)\cdot pq &= \tfrac{\gamma(T-\delta)}{\gamma(T-\delta)+(1-\gamma)T}\cdot k = \gamma\cdot(1-\tfrac{\delta}{T}\cdot\tfrac{1-\gamma}{1-\gamma\frac{\delta}{T}})\cdot k\\ &\gtrsim (\gamma-\tfrac{\delta}{T}\gamma(1-\gamma))\cdot k = (\gamma-O(\tfrac{\delta}{T}))\cdot k,\end{aligned}$$

since $\gamma(1-\gamma)\leq\frac{1}{4}$ always holds. Therefore, the next committee $\mathsf{com}_{\mathcal{T}_{nxt}}$ includes $(\gamma-o(1))k$ members of P in expectation for the worst network delay. In the same way, we can derive that $\mathsf{com}_{\mathcal{T}_{nxt}}$ includes $(\gamma+o(1))k$ members of P in expectation for the best network delay where P suffers no delay while solutions from others are delayed for δ.

Lemma 6 ($\mathbb{I}^{\mathcal{T}}_{\mathsf{fairness}} \overset{\mathsf{P}}{\Rightarrow} \mathbb{I}_{\mathsf{com}_{\mathcal{T}_{nxt}}}$)**.** *The fairness of PoW in an epoch $\mathcal{T}$ entails the safety of the next committee $\mathsf{com}_{\mathcal{T}_{nxt}}$ with an overwhelming probability.*

Proof. Recall that we have assumed an adversary controlling $\alpha < 1/3-\epsilon$ fraction of total hash rate. Setting indicators I_i and J_i for each slot of the next committee. I_i is 1 if this slot is owned by the adversary or 0 otherwise. We denote $Y = \sum_{i\in[k]} I_i$ as the random variable of total slots controlled by the adversary for the next committee. From the fairness of PoW in $\mathcal{T}$, $E[Y] = \sum_{i\in[k]} E[I_i] = \hat{\alpha}k$ where $\hat{\alpha} = \alpha + o(1) < 1/3$. By Chernoff bound,

$$\Pr[Y \geq (1+\xi)\hat{\alpha}k] \leq exp\Big(-[(1+\xi)\ln(1+\xi)-\xi]\hat{\alpha}k\Big).$$

Letting $\xi = \frac{1}{3\hat{\alpha}} - 1 > 0$, it shows

$$\Pr[Y \geq k/3] \leq exp\Big(-(\frac{1}{3}\ln\frac{1}{3\hat{\alpha}} - \frac{1}{3} + \hat{\alpha})k\Big) = exp(-\Theta(k)).$$

Therefore, the safety of the next committee is achieved except for a probability negligible in κ since $k = \Theta(\kappa)$.

Lemma 7 ($(\mathcal{T}\neq\mathcal{T}_0 \Rightarrow \mathbb{I}^{\mathsf{com}_{\mathcal{T}_{pre}}}_{\mathsf{consensus}})$, $\mathbb{I}_{\mathsf{com}_{\mathcal{T}}} \Rightarrow \mathbb{I}^{\mathsf{com}_{\mathcal{T}}}_{\mathsf{consensus}}$)**.** *The safety of the consensus protocol of the previous epoch $\Pi^{com_{\mathcal{T}_{\mathrm{pre}}}}_{consensus}$ and the safety the current committee $\mathsf{com}_{\mathcal{T}}$ ($\mathbb{I}_{com_{\mathcal{T}}}$ alone if it is the initial epoch $\mathcal{T}_0$) imply the safety of the consensus protocol of the current epoch.*

Proof. – For simplification, we assume that $t + w < \mathsf{end}(\mathcal{T})$. In real-world, an epoch is significantly long so a transaction proposed exactly by the termination of an epoch (and hence not confirmed before $\mathsf{end}(\mathcal{T})$) can be regarded as a tolerable miss. The liveness assumption of PBFT has guaranteed that there is an output $\widetilde{G_t}$ in time t if over 2/3 committee members are honest and that they are reaching a consensus according to a shared view. Obviously,

the first condition is guaranteed by the safety of $\mathsf{com}_{\mathcal{T}}$ and the second condition is essentially $G^i_{t-\delta} \sqsubseteq G^j_t$ [7] for each $P_i, P_j \in \mathsf{com}_{\mathcal{T}}$ since the protocol asks members to issue proposals in PBFT according to the view δ ago. According to the $(\mathcal{A}, \mathcal{Z})$-compliant execution model, $G^i_{t-\delta} \sqsubseteq \sqcup^{t-\delta}_{r=0} \Delta G_r$. Also, $\Delta G_r \sqsubseteq G^j_{r+\delta}$ holds for all $0 \leq r \leq t-\delta$. Therefore, $\sqcup^{t-\delta}_{r=0} \Delta G_r \sqsubseteq \sqcup^{t}_{r'=\delta} G^j_{r'}$. From the rule $(t > 0 \Rightarrow G^j_{t-1} \sqsubseteq G^j_t)$, we can observe $\sqcup^{t}_{r'=\delta} G^j_{r'} \sqsubseteq G^j_t$ by a simple induction. By transitivity, $G^i_{t-\delta} \sqsubseteq (\sqcup^{t-\delta}_{r=0} \Delta G_r) \sqsubseteq (\sqcup^{t}_{r'=\delta} G^j_{r'}) \sqsubseteq G^j_t$ infers $G^i_{t-\delta} \sqsubseteq G^j_t$.

- In the protocol, the proposal of a newly added key node is asked to include all orphan nodes in its view $(\delta + \delta_c)$ time ago. Namely, $G^i_{t-\delta-\delta_c} \sqsubseteq \widetilde{G_t}$. From the model of $(\mathcal{A}, \mathcal{Z})$-complaint execution, $\Delta G_{t'} \sqsubseteq G^i_{t-\delta-\delta_c}$ holds for all $t' < t - 2\delta - \delta_c$. Thereby, we reach $\Delta G_{t'} \sqsubseteq \widetilde{G_t}$ for all $t' < t - 2\delta - \delta_c$ by transitivity.
- The safety of PBFT has guaranteed that the outcome satisfies our predetermined rule that at least one key node is proposed to link all orphan nodes in a shared part of the view of DAG. Without loss of generality, we assume that only one key node u is added into $\widetilde{G_t}$ in time t after $t-1$. With an admissible execution guaranteed by the safety of $\mathsf{com}_{\mathcal{T}}$, $\widetilde{G_t}$ should be in the form of

$$\widetilde{G_t} = \widetilde{G_{t-1}} \sqcup \left(\{u, v_1, v_2, \ldots, v_\ell\}, \{(u, \hat{v}_1), (u, \hat{v}_2), \ldots, (u, \hat{v}_{\hat{\ell}})\}\right) \tag{3}$$

  hat $v_i \notin \widetilde{G_{t-1}}.V$ for all $i \in [\ell]$ and $v \in \{v_1, v_2 \ldots, v_\ell\}$ for all $v \in \{\hat{v}_i\}_{i=1}^{\hat{\ell}}$. From here, it is obvious that $\widetilde{G_{t-1}} \sqsubseteq \widetilde{G_t}$.
- First, we introduce two observations.
  Observation i) $\widetilde{G_t}$ is an admissible DAG if $\widetilde{G_{t-1}}$ is admissible. For convenience, we set $\widetilde{G_{\mathsf{begin}(\mathcal{T})-1}} := G_{\mathsf{end}_{\mathcal{T}_{pre}}}$. To prove that $\widetilde{G_t}$ is an admissible DAG providing that $\widetilde{G_{t-1}}$ is admissible, we essentially need to prove three properties: a) $\widetilde{G_t}$ is a DAG; b) succinctness; c) key units are totally ordered. Each of them can be easily observed from formula (3) by leveraging a simple contradiction.
  Observation ii) $G_{\mathsf{end}(\mathcal{T}_{pre})} = \widetilde{G_{\mathsf{end}(\mathcal{T}_{pre})}}$. This is guaranteed from the complaint execution model.
  With this observation, the final proof is an obvious induction where the initial step requires that $\widetilde{G_{\mathsf{begin}(\mathcal{T})-1}} := G_{\mathsf{end}(\mathcal{T}_{pre})} = \widetilde{G_{\mathsf{end}(\mathcal{T}_{pre})}}$ is a fully admissible DAG (from the hypothesis $\mathbb{I}^{\mathsf{com}_{\mathcal{T}_{pre}}}_{\mathsf{consensus}}$). For the induction step, apart from the first observation, we remain to show that all vertices are reachable from a key node in $\widetilde{G_t} = \widetilde{G_{t-1}} \sqcup \left(\{u, v_1, v_2, \ldots, v_\ell\}, \{(u, \hat{v}_1), (u, \hat{v}_2), \ldots, (u, \hat{v}_{\hat{\ell}})\}\right)$. We prove this by contradiction. Assume that $v \in \widetilde{G_t}$ is not reachable from any key node. Then, either $v \in \widetilde{G_{t-1}}.V$ or $v \in \{u, v_1, v_2, \ldots, v_\ell\}$. For the first case, it contradicts that $\widetilde{G_{t-1}}$ is a fully admissible DAG. The second case directly contradicts the complaint execution model.

[7] For rigorousness, we regard $G_r = (\emptyset, \emptyset)$ for all $r < 0$.

By assuming that $\mathbb{I}_{\mathsf{GEN}}$ implies $\mathbb{I}_{\mathsf{com}_{\mathcal{T}_0}}$, we are allowed to link up the first two rows of the proof chain of Tab. 1.

Lemma 8 (Proof Chain Foundation). *For arbitrary large polynomial $\mathsf{poly}(\cdot)$, assuming $\mathbb{I}_{GEN}$, $(\mathbb{I}_{com_{\mathcal{T}_i}}, \mathbb{I}^{com_{\mathcal{T}_i}}_{epoch}, \mathbb{I}^{com_{\mathcal{T}_i}}_{switch}, \mathbb{I}^{\mathcal{T}_i}_{fairness})$ holds for all natural number $i \leq \mathsf{poly}(\kappa)$ except for a probability negligible in κ.*

Proof. According to previous lemmas, we are allowed to draw out the first row of proof chain in Tab. 1. Each straight arrow in the figure stands for a deterministic entailment and the curved line stands for an entailment for a probability $1 - \mathsf{negl}(\kappa)$ for some negligible function $\mathsf{negl}(\kappa)$. Thereby, it is easy to observe that we need only to show that $\mathsf{poly}(\kappa)$ many probabilistic entailments are achieved. This happens except for a negligible probability of $1 - (1 - \mathsf{negl}(\kappa))^{\mathsf{poly}(\kappa)} \approx \mathsf{negl}(\kappa)\mathsf{poly}(\kappa)$.

Lemma 9 ($\{\mathbb{I}_{\mathsf{com}_{\mathcal{T}_i}}\}_{i\in[\mathsf{poly}(\kappa)]} \Rightarrow \mathbb{I}_{\mathsf{consistency}}$). *The safety of all committees of $\mathsf{poly}(\kappa)$ epoches starting from the initial epoch guarantees $\mathbb{I}_{consistency}(view, 1^\kappa, \mathcal{A}, \mathcal{Z})$.*

Proof. Recall that the consistency property asks that for any large polynomial $\mathsf{poly}(\cdot)$, $\mathsf{Total}(\overline{G}_s) \preccurlyeq \mathsf{Total}(\overline{G}_t)$ holds for all $0 < s \leq t \leq \mathsf{end}_{\mathcal{T}_{\mathsf{poly}(\kappa)}}$.

- To begin with, we show that $\overline{G}_s \sqsubseteq \overline{G}_t$ within any epoch for all $0 < s \leq t \leq \mathsf{end}_{\mathcal{T}_{\mathsf{poly}(\kappa)}}$. We show this in two steps. Firstly, from the compliant execution model, $\mathcal{T} \in \{\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{\mathsf{poly}(\kappa)}\}$, $\overline{G}_{t-1} \sqsubseteq \overline{G}_t$ holds for all $\mathsf{begin}(\mathcal{T}) < t \leq \mathsf{end}(\mathcal{T})$ (with the hypothesis of $\mathbb{I}_{\mathsf{com}_{\mathcal{T}}}$). Secondly, for any epoch any epoch $\mathcal{T} \in \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{\mathsf{poly}(\kappa)}\}$, it actually holds that $\overline{G}_{\mathsf{end}(\mathcal{T}_{pre})} \sqsubseteq \overline{G}_{\mathsf{begin}(\mathcal{T})}$ since
$$\overline{G}_{\mathsf{end}(\mathcal{T}_{pre})} \sqsubseteq G_{\mathsf{end}(\mathcal{T}_{pre})} = G_{\mathsf{begin}(\mathcal{T})-1} = \overline{G}_{\mathsf{begin}(\mathcal{T})}.$$
- It remains for us to show that $\overline{G}_s \sqsubseteq \overline{G}_t$ implies $\mathsf{Total}(\overline{G}_s) \preccurlyeq \mathsf{Total}(\overline{G}_t)$. This is a direct application of Lemma. 2.

Lemma 10 ($\{\mathbb{I}_{\mathsf{com}_{\mathcal{T}_i}}\}_{i\in[\mathsf{poly}(\kappa)]} \Rightarrow \mathbb{I}_{\mathsf{liveness}}$). *The safety of all committees of $\mathsf{poly}(\kappa)$ epoches starting from the initial epoch guarantees $\mathbb{I}_{liveness}(view, 1^\kappa, \mathcal{A}, \mathcal{Z})$.*

Proof. Recall that the liveness property asks that for any large polynomial $\mathsf{poly}(\cdot)$, $\Delta G_t.V \subseteq \{\mathsf{Total}(\overline{G}_{t+3\delta+\delta_c})\}$ holds for all $0 < t \leq \mathsf{end}(\mathcal{T}_{\mathsf{poly}(\kappa)})$. $\{\mathbb{I}_{\mathsf{com}_{\mathcal{T}_i}}\}$ guarantees $\{\mathbb{I}^{\mathsf{com}_{\mathcal{T}_i}}_{\mathsf{consensus}}\}$ and hence for any $t' < t - 2\delta - \delta_c$, $\Delta G_{t'} \sqsubseteq G_t$ within the first $\mathsf{poly}(\kappa)$ epoches. Along with Lemma. 3, we only need to show that $G_{t-\delta} \sqsubseteq \overline{G}_t$ (note that $\overline{G}_t$ is a fully admissible DAG). The compliant execution model guarantees that $\widetilde{G_{t-\delta}} \sqsubseteq \overline{G}_t$ and that $G_{t-\delta} = \sqcup_{i=0}^{t-\delta} \widetilde{G_i}$. Also, $\widetilde{G_{i-1}} \sqsubseteq \widetilde{G_i}$ holds always. This results in $G_{t-\delta} = \sqcup_{i=0}^{t-\delta} \widetilde{G_i} = \widetilde{G_{t-\delta}} \sqsubseteq \overline{G}_t$.

Finally, we are allowed to conclude in the final proof of Thm. 1 which states that $(\mathcal{A}, \mathcal{Z})$-compliant execution guarantees both consistency and liveness with a safe bootstrapping.

Proof. With a safe bootstrapping $\mathbb{I}_{\mathsf{GEN}}$, according to Lemma. 8-10, $(\mathcal{A}, \mathcal{Z})$-compliant execution guarantees $\{\mathbb{I}_{\mathsf{com}_{\mathcal{T}_i}}\}_{i\in[\mathsf{poly}(\kappa)]}$ and hence both $\mathbb{I}_{\mathsf{consistency}}$ and $\mathbb{I}_{\mathsf{liveness}}$ with an overwhelming probability, which are essentially properties of consistency and liveness.

C Incentive Compatibility

A distributed ledger system requires to be incentive-compatible, nor participants will have the motivation to deviate from the expected behaviors for higher reward. In other words, the property of incentive compatibility is essentially a prerequisite of the stability of a distributed ledger system. Hence in Haootia, we have adopted an elaborate incentive mechanism and its incentive compatibility is shown as follows.

Incentive Mechanism. The incentive mechanism of Haootia consists of two components: transaction commission and committee reward. In terms of commission, each transaction proposer needs to pay a commission in proportion to the size of his/her transaction node. Such a transaction commission will be collected by the transaction node who is the *first* to reference this transaction as a parent in the DAG ledger. Here, the concept of *first* denotes the children with the smallest total order (see Def. 6) among all the children of the given transaction. On the other hand, the committee reward mechanism awards the committee members if they behave honestly during the execution of the Haootia protocol. This reward comes directly from block reward as compensation for their work for maintaining and securing the whole system.

Transaction Proposer. For the convergence of DAG ledger and fast confirmation of transaction nodes, we expect each transaction proposer to reference as many and as recent tip nodes in the DAG leader as possible, which is achieved through the transaction commission mechanism mentioned above. Since the commission of nodes will be collected only by the first node referencing it, transaction proposers are motivated to reference as recent tip nodes as possible. Meanwhile, to get more rewards, they are encouraged to reference as many tip nodes as possible. Thus, each transaction proposer will apparently stick to our expectation to obtain the highest commission reward.

Committee Member. During the execution of Haootia protocol, the committee as an entirety outputs key nodes continuously to linearize transactions and confirm nonce solutions from miners, and when enough solutions are received it launch a switchover to generate the next committee and hence the next block. By behaving honestly, a committee member will get its reward as compensation for its investment of computing resources in the mining. Otherwise, if it's found Byzantine faulty during the execution of Haootia protocol, it will risk losing all of the rewards above. Thereby, rational committee members will honestly follow the protocol.