HelixMesh Protocol

Helix Cognitive Computing Team

info@hlx.ai

Version 1.0 - April 9, 2019

Abstract

We provide a technical exposition of the HelixMesh protocol and the underlying DAG-based transaction ledger. The HelixMesh is based on the double-consensus MeshCash framework and the Snowball protocol of the Avalanche family. The main advantage over existing BFT consensus protocols is a high degree of flexibility: the protocol can be deployed in both permissionless and permissioned environments. Moreover, virtually any BFT consensus is suitable for the off-chain layer of the HelixMesh. As a result, the liveness vs safety tradeoffs are not dictated by the protocol itself but may be adjusted based on the particular use-case requirements. This makes the HelixMesh a robust blueprint for IoT-sourced distributed ledgers.

1 Does IoT need a specialised DLT?

1.1 From Nakamoto to mass adoption

The Bitcoin Blockchain is considered by many to be the first successful example of a Distributed Ledger Technology (DLT) being deployed and secured by a decentralized network of peers. Efforts to increase the human usability of the Bitcoin's underlying DLT (e.g. Bitcoin Lightening Network, Ethereum, Cosmos, etc.) have focused mostly on the human demand side of things, thus increasing the scalability, extensibility and interoperability of DLTs. To the best of our knowledge, however, there has not been a similar growth in the development of DLTs for machines, specifically for internetworked devices, or simply, the Internet of Things (IoT). In the current article, we describe a new protocol for IoTs requiring DLTs, the HelixMesh protocol. We provide the rationale for our decisions in the design of the protocol, and make the trade-offs explicit.

1.2 Fat and thin protocols

Somewhat oversimplifying, there are currently two major lines of thought on how distributed ledgers should be designed [11]. On the one hand, there is the "fat protocol" attitude which assumes that all Distributed Applications (dApps) and services interact with the shared state of a network-wide distributed Virtual Machine (VM) through a general-purpose one size-fits-all protocol. On the other hand, there are "thin protocol" advocates who argue that "each dApp should have its own ledger", which suggests that network participants are expected to reach consensus about each dApp state independently of one another (e.g. Git-style agent-based approach).

We believe that for the majority of practical applications the sweet spot is somewhere between these two extremes. The design approach we have taken is a middle-ground protocol suitable for IoT clusters of moderate size (up to thousands of nodes) with specialised dApps built on top. In turn, such heterogenous networks are connected into a global infrastructure. This approach is inspired by interoperability projects (e.g. Cosmos [1], Polkadot [2], etc.), which offer blueprints to launch dedicated sub-blockchains. While we share the general vision of heterogenous interconnected DLTs, we extend our focus to IoT-networks which have more specific requirements on the protocol stack: high transaction thoughput is more important then quick transaction finality. Pursuing this idea, we set availability as a top-rate priority in our design, which partially explains our decision to adopt the MeshCash framework as the backbone for HelixMesh.

The Meshcash framework was introduced by Bentov et. al. [4], originally as a scaling solution for Bitcoin. The authors introduced a mechanism for reaching a consensus on a large number of transaction batches. We further boost their design with a powerful insight, used in the Avalanche [14] protocol family, that random peer gossiping can be turned into a robust consensus protocol stable even under network churn. We provide an in-depth exposition in Section 4.

1.3 Advantages of IoT-tailored DLTs

Specialised IoT-tailored DLTs should be able to handle the data streams of billions of devices. On the other hand, each IoT device (e.g. a sensor) typically consumes little-to-no data itself. Furthermore, all incoming and outgoing connections are typically managed by a trusted edge server. Such natural clustering introduced by the network topology allows one to soften the trust requirements necessary for general-purpose scalable blockchains (e.g. Ethereum 2.0). Additionally, since we expect that only a small fraction of IoT-generated transactions will transfer cryptocurrency value between addresses (instead, the majority are expected to carry only data payloads), our design choices depart from the usual general-purpose blockchains which have been used e.g. for decentralised and distributed financial applications (e.g. Uniswap or MakerDAO).

By design, fullnodes in the HelixMesh protocol are assumed to aggregate large volumes of transactions without congesting the network, thus providing high throughput and availability. On the other hand, the double-layered consensus of MeshCash (covered in detail in Section 4) allows one to eventually achieve the required level of safety in mission-critical applications even on rare occasions when the fast ("Hare") consensus has failed to quickly finalise the updates due to e.g. network delays.

On a longer timescale, we envision a global interconnected network of HelixMesh-based specialised DLTs, thus supporting the fat protocol narrative. That being said, a HelixMesh implementation can be adapted depending on the tradeoffs and the trust model of each particular subnetwork (which may or may not be permissioned), thanks to the pluggable architecture of the MeshCash framework.

1.4 Helix Network

We also provide some details on a practical implementation of HelixMesh, the Helix Network. The design choices are not part of the core HelixMesh protocol, so this section is defered to the Appendix.

The Helix Network is leaderless and permissionless (i.e. there are no pre-defined rules restricting writes to the ledger). The consensus about the transaction log is reached by fullnodes expected to be online 24/7 on their best effort. Light clients submit transactions to a fullnode of choice. This topology reflects a typical set-up of low-power IoT devices (light clients) connected to an edge server (fullnode).

1.5 Proof of Contribution

The weight of a node in the consensus process is proportional to the *contribution* of the node to the total *network value*. The exact meaning of contribution depends on a concrete implementation of the consensus protocol, and therefore we introduce an abstract notion of Proof-Of-Contribution. The main purpose of Proof-Of-Contribution (PoC) is to prevent Sybil attacks by attaching a weight to each participant in the global consensus process in a dynamic way. Another benefit of having a separate abstraction layer for rewards and contributions is the possibility to model and analyse game-theoretic mechanics independently from the underlying consensus protocol. This is crucial since the protocol guarantees may fail if honest behaviour (from the consensus protocol point of view) is not *rational* in the game-theoretic sense[1]. Indeed, in this case honest nodes are incentivised to become (slightly) evil and deviate from the protocol so that the honest supermajority assumption may no longer hold.

Our PoC abstraction can support both Proof-Of-Work and Proof-Of-Stake implementations and allows one to potentially take the best of both worlds. We will describe a subscription-based PoC implementation of the Helix Network in a forthcoming paper covering network monetisation, token economy and rewards distribution.

[1]MeshCash [4] is a rare example when a rigorous analysis is possible. In their setting PoC is Proof-Of-Work.

2 DLT overview and scope of the paper

Transactions are packed into *bundles*. Each bundle contains a Merkle root of the hashes of the parent bundles, so the bundles form a DAG (directed acyclic graph) with directed edges representing the child-parent relation between bundles. We ditch a more conventional *block* of transactions to emphasise that a bundle can contain heterogenous data packets not limited to the scope of value transactions[2].

Each bundle contains an ordered list of transactions[3] and has a *height* which is the length of the shortest path to the genesis bundle of the DAG. Given a DAG G, define the round $R(k)$ to be the set of bundles of height k.

We assume public key infrastructure (PKI). For simplicity, we assume that for each round k each participating fullnode is identified by a public key pk_i and an IP address suitable for duplex point-to-point communication. IP addresses may change but the time fullnodes update internal IP tables is assumed to be negligible.

Instead of broadcasting the whole bundle $B^{(k)}_{pk_i}$ we assume that fullnodes submit a binding *commitment* $R(k, pk_i)$ (e.g. a Merkle root of transactions) and the meta information in a header. Such committments significantly reduce network traffic and separate the consensus logic from validitation (not considered in this paper).

The protocol can be broken down into the following steps.

1. Consensus about the set of committments $\{R(k, pk_i)\}$ in round k
2. Ordering $R(k, pk_i)$, the set of committments
3. Obtaining the ordered sequence of transactions with committment to $R(k, pk_i)$
4. The canonical sequence of transactions is defined by successive expansion of transactions with committment to $R(k, pk_i)$ in the order defined by (2), skipping contextually invalid transactions (e.g. double-spends). A pseudocode is given by Algorithm 1.

Algorithm 1 Generating canonical transactions in round k

```
1:  function TransactionsInRound(Ctx, $k$)
2:      for R ∈ Order($\bigcup_i R(k, pk_i)$) do
3:          $\mathcal{T}$ ← GetAndVerify(R)
4:          for tx ∈ $\mathcal{T}$ do
5:              if IsContextuallyValid(Ctx, tx) then
6:                  Ctx ← Apply(Ctx, tx)
7:                  Yield(tx)
8:              end if
9:          end for
10:     end for
11: end function
```

Steps (2) and (4) are local and determinstic routines and thus always give the same output for each honest node in the network. Indeed, once an irreversible consensus on which bundles are included in the round is reached, a canonical ordering is defined by a fixed protocol rule. Such a rule provides a deterministic yet hard-to-game sortition (e.g. using XOR of bundle hashes) for an unordered set of bundles. Since all honest nodes agree on the sortition rule (it is known in advance and dictated by the protocol) as well as on the (unordered) set of bundles, the ordering is canonical[4].

Thus, the output $R(k)$ of Step (1) for round k can be assumed to be an ordered collection of bundles

$$B^{(k)}_1, B^{(k)}_2, \ldots, B^{(k)}_r.$$

[2]E.g. auditable logs of IoT-sourced data streams. We consider IoT use-cases in more detail in an economics paper.

[3]Ordering of transactions within a bundle is decided by the fullnode who signs and publishes the bundle.

[4]Hereafter *canonical* means that all nodes faithfully following the protocol have the same output.

In turn, each bundle is an ordered sequence of transactions:

$$B_i^{(k)} = T_1^i, T_2^i, \ldots .$$

Summing up, $R(k)$ uniquely defines a canonically ordered set of transactions in round k, given by Algorithm 1.

All transactions in $R(l)$ with $l < k$ precede those in $R(k)$, and all transactions in $R(m)$ with $m > k$ follow after. Thus, when consensus on $R(i)$ is irreversible for all $i \leq k$, all the transactions included in the bundles up to round k are final and canonically ordered.

The third step assumes data availablity which is beyond the scope of the present paper. For concreteness, the reader may assume that each bundle contains an IPFS address to its contents. Even such a simplistic solution will provide strong guarantees as long as there is at least one party with data satisfying the bundle commitment. In fact, in order to ensure the bundle is accepted by the network, bundle producers will likely become data seeders for own bundles.

In the long run, we believe that advances in the technology of succinct zero-knowledge proofs (ZKP) may eventually verify data availability without requesting it in full. There is already a substantial body of work on such applications of ZKP to be considered in practical implementations of the HelixMesh protocol, but we omit the details.

Further, if Line 3 of Algorithm 1 detects that a commitment does not match the transactions contained in the bundle, the fullnode who signed the commitment is economically punished. The exact mechanism depends on the PoC implementation and can either be based on stake slashing (with a Proof-Of-Stake based PoC) or simply on the fact that malformed bundles are not included in the canonical view.

We will therefore focus solely on Step (1) from now on.

3 Communication and Adversarial model

In the Helix Network light clients submit transactions to a fullnode of choice. The fullnodes are responsible for broadcasting, consensus and validation. A transaction is accepted by a fullnode after a negotiation protocol with the light client, described in Appendix. Once a transaction is accepted by a fullnode, it has a sender (the light node) and a broker (the fullnode). The fullnode commits to fair handling of the transaction. Fullnodes accepting transactions are expected to have a reliable high-bandwidth internet connection and stay online at their best effort.

For simplicity, we assume from now on that fullnodes do not censor out valid incoming transactions (for a trustless scheme see Appendix). Communication between fullnodes is done through gossiping on the overlay network. We assume that each honest node can efficiently sample random peers.

3.1 Adversarial and synchrony assumptions

We assume *strong synchrony*: a message broadcasted by an honest node is received by all honest nodes within time δ (according to the local clock of the sender)[5]. The parameter δ is fixed at the start of the protocol. We believe that HelixMesh has the same guarantees under a weaker and more realistic assumption that the network is merely *δ-intermittently synchronous*, but the local clocks are δ-synchronised for the majority of honest nodes. Roughly speaking, it means that the network latency may have spikes when no messages are delivered, but typically the messages are delivered fast. For a rigorous definition see [9], Definition 2. We defer formal proofs to a forthcoming research paper.

Further, we assume that full nodes have local clocks synchronised with the Coordinated Universal Time (UTC). The actual discrepancy is concealed in the bounded network delay δ.

The UTC time serves as a universal time beacon, common for all clients and network participants. It is used to ensure fair ordering of transactions within bundles and for a steady flow of bundle rounds: each round has a timeout after which an honest node stops accepting bundles. We will consider the possibility of implementing a decentralised and network intrinsic time in future releases[6].

Further, we assume that all local clocks are consistent and binding in the following sense. If there are two messages m_1 and m_2 with timestamps $t_1 < t_2$ signed with the same key pk_i, then m_1 was provably sent prior to m_2 (but the delivery times may of course be arbitrary).

The following lemma is well known.

Lemma 3.1. *A random gossip protocol overlayed over point-to-point links in a network with N nodes broadcasts a message to the whole network with at most $10 \log N$ hops whp.*

According to [3] it takes abount 1s to gossip a 1KB message to 90% of the nodes in the Bitcoin network, and Algorand [5] reports 10s for gossiping 1MB blocks. Thus, in a pure gossip-based network a conservative estimate for δ would be around 15 seconds (compare to 12 second block time in the Ethereum network).

However, in semi-centralised scenarios gossiping can be used only as a fall-back for a more efficient but more centralised network topology with a few high-throughput network relayers. In this case, δ can be lowered to less than a second. The exact value depends on the liveness-safety tradeoffs dictated by the use-case in mind.

We assume that an adversary can withhold messages but cannot forge signatures.

[5]The same guarantees hold under a somewhat weaker condition that *most* (e.g. 95%) honest nodes receive messages within δ. A node for which this assumption fails to hold is assumed to be offline (and not counted as honest).

[6]UTC synchronisation assumption may be dropped at the expense of having less predictable timing for protocol rounds.

4 Consensus protocol

4.1 Overview: Hare and Tortoise protocols

The backbone of the consesus protocol is the Meshcash framework introduced by Bentov et. al. [4]. The Meshcash DAG layers correspond to rounds and suit our setting particularly well. An honest node increments the round counter from i to $i+1$ when her local clock shows GENESIS_TIME + $\Delta_R i$, where Δ_R is the round duration fixed by the protocol. The first (local) time an honest node increases the layer counter is denoted by $start_i$. By our assumptions all honest nodes start their rounds by $start_i + \delta$ according to their local time. Similarly, one may introduce a fixed cutoff for round ends, but we assume for simplicity that round i ends when round $i+1$ starts.

We adopt the mechanism of two separate processeses run in parallel: an off-chain Hare consensus and a DAG-based Tortoise consensus protocol. Here off-chain means that the Hare protocol relies on messages from network peers, discarded once recieved, and not on the information available in the ledger (i.e. the bundle DAG). The Tortoise protocol, on the other hand, is on-chain in the following sense: the protocol output is fully determined by the information contained in the bundle DAG and no additional information is required in order to decide which bundle (at least old enough) is canonical. This is similar to the Nakamoto consenus of the Bitcoin blockchain.

The Hare consensus will normally terminate in time. When it does, it acts as an oracle which predicts the output of the Tortoise consensus protocol. If it does not terminate, then nodes should await the output of the Tortoise. For its part, the Tortoise depends only on the local state of the DAG and does not require any additional communication which renders it well-suited for SPV (Simple Payment Verification). Of course, the drawback to the Tortoise is that this protocol may take considerable time before it reaches a desired confirmation margin.

The output of the Tortoise is based on special bits — “votes” — which are included in each bundle in the DAG (described in detail in the subsections to follow). In order to guarantee exponentially fast convergence of the Tortoise Protocol it is sufficient that honest nodes vote for each particular bundle in a (moderately) coordinated way. Bentov et. al. introduced the following (rather mild) definition of $[s,t]$-*consistent* protocols.

Definition 4.1. *A protocol Π is $[s,t]$-consistent if for any bundle X in layer i, all honest nodes are in consensus about X whose layer counter is in the interval $[i+s, i+t]$.*

If honest nodes vote $[s,t]$-consistently (based on the output of Hare), the confidence of Tortoise on the validity of each particular bundle will grow exponentially with the number of rounds passed from the bundle publication. Such probabilistic finality is similar to the mechanism of block confirmations for Bitcoin transactions.

Theorem 4.1 (Hare/Tortoise protocol consistency)**.** *Assume honest nodes vote in a $[s,t]$-consistent way (following the Hare protocol). Then for every bundle A generated in the interval $[start_i, start_{i+1})$, the probability that two honest nodes exist who disagree on the validity of A according to Tortoise protocol at time $start_i + t$ is $\exp(-O(t))$.*

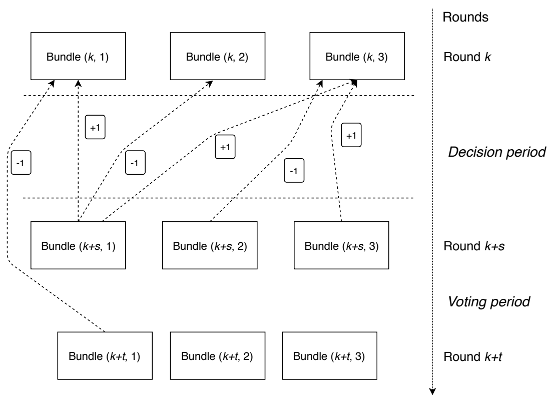
Theorem 4.1 is a rather powerful tool for quick finality: it allows one to accept transactions without waiting for the Tortoise protocol to reach enough confirmations relying on the fast Hare protocol alone.

Corollary 4.1.1. *Let X be a bundle published at round s. Assume that the Hare protocol has successfully terminated with the output “X is valid”. Then X will eventually be validated by the Tortoise protocol with an arbitrary confirmation margin.*

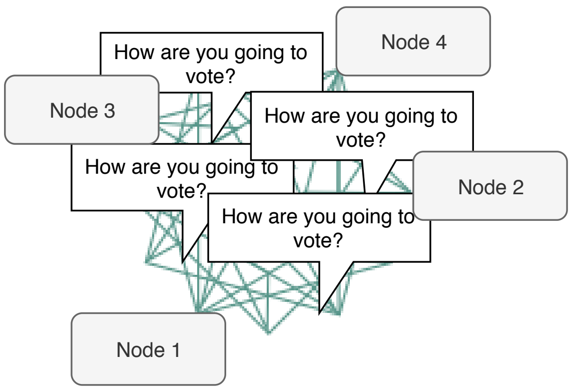
Proof. Since the Hare protocol has terminated, all honest nodes will vote “X is valid” so the votes are $[s,t]$-consistent for any t after the protocol termination time (i.e. they do not revert the decision). Thus one can apply Theorem 4.1 so the probability that the local execution of the Tortoise protocol by any honest node will produce “X is invalid” in the future is $neg(t)$. Since t can be taken arbitrarily large, the claim follows. □

In particular, as long as the fast Hare consensus successfully terminates, it's safe to assume this decision is final.

We adopt the Snowball protocol for the Hare part. The nodes ask random peers how they are going to vote. If after multiple independent queries a node observes a bias towards a particular decision, it joins the crowd. It turns out that if a node has been convinced by random peers (i.e.



(a) On-chain Tortoise voting


Node 4
How are you going to vote?
Node 3
How are you going to vote?
How are you going to vote?
Node 2
How are you going to vote?
Node 1

(b) Off-chain Hare consensus during the decision period

Figure 1: Tortioise and Hare consensus

the protocol has terminated for this node) on some decision, with overwhelming probability all other honest nodes will settle on that decision as well.

In rare occasions it may happen that the Hare protocol will not provide high enough confidence by the time the vote has to be included in the bundle, in which case the vote is neutral and the outcome is decided by the Tortoise protocol later on. Even without any assistance from Hare (i.e. even in a standalone mode) the Tortoise protocol will eventually converge (i.e. reach any given voting margin in absolute value). Let's say the consensus is ϵ-final if[7]

$$\mathbb{P}(\text{decision on } X \text{ flips}) < \epsilon.$$

Theorem 4.2. *For any given $\epsilon > 0$ the Tortoise protocol eventually reaches ϵ-final consensus about the validity of an arbitrary bundle X.*

The main drawback is that without assistance from Hare it may take some time before Tortoise accumulates enough voting margin to become virtually irreversible. The crucial property is that this voting margin is *observable*, i.e. the corresponding portion of the DAG may serve as a certificate proving to an offline merchant that a transaction is virtually irreversible (similar to SPV certificates in Bitcoin).

Theorems 4.1 and 4.2 have been proved in [4] under the following conditions:

1. Strong δ-synchronous network
2. Adversarial PoC follows a stationary Poisson process (but may have pre-generated reserver).

Our ongoing research indicates that the analysis extends to more realistic assumptions:

[7] The probability distribution is assumed to be over all possible future realisations of the local DAG of an honest node conditioned on the protocol assumptions.

1. Weakly synchronous network: we assume δ-synchronous network, but the protocol guarantess do not depend on δ (it is unknown in advance).

2. The rate of the adversarial PoC is time-dependent, but bounded by q on *average*. More precisely, for any $\epsilon > 0$ and some fixed parameter $\lambda > 0$.

$$\mathbb{P}(Q > (1+\epsilon)qW) \leq \exp(-\lambda\epsilon^2 qW), \tag{1}$$

   where Q is the amount of PoC generated by non-honest nodes and W is the total network PoC generated over the same span of consecutive rounds.

Note that (1) holds with $\lambda = \frac{1}{1+\epsilon}$ if the total network PoC follows a Poission process and the contribution from honest nodes has rate at least $1-q$. Thus, this case is already covered by [4].

4.2 Tortoise protocol bundle header structure

The Tortoise protocol relies on parent references to previous bundles embedded in a bundle. Since previous bundles are already in a local view of each party, it suffices to include only merklized roots of the parent hashes and thus bundle size overhead is $O(1)$ regardless of how many bundles are referenced.

Further, a bundle contains a milestone that is a Merkle root of all bundles (according to the current DAG view, sorted in the canonical order) in round $i-E$, where E is the epoch parameter. A sensible value for E may be set to be approximately from hours to a few months in the past (assuming rounds have approximately even time). To ensure long-range consistency, we enforce the *syntactic* bundle validity rule that immediate parents must have the same milestone root.

Each honestly generated bundle in round i has the following information:

1. *Timestamp* Fullnode's local time at which the bundle is published

2. *Round number* The round number of the bundle

3. *View edges* Tips of the local view DAG at the moment a bundle was submitted

4. *Voting edges* A $\{-1, 0, 1\}$ vote for every syntactically and contextually valid bundle in round $i-s$ through $i-t$ (where the contextual validity is derived from the total ordering of the bundles included in each round in the local view).

5. *A weak common coin bit* A 0/1 bit of the weak common coin toss.

6. *Before coin bit* This bit indicates whether the bundle was generated before the coin protocol ended (this is used to "abstain" from voting in cases where the coin bit matters).

7. *Milestone* Merkle root of all bundles in round $i-E$ accessible from the current bundle, sorted in the canonical order.

4.3 Tortoise protocol voting

Let $s < t$ be fixed parameters. During rounds i to $i+s-1$ the off-chain Hare consensus kicks in but it may or may not succeed. In any case, the node includes a vote $V_r^P(A)$ for each bundle $A \in R(i)$ in all in rounds $r \in \{i+s, i+s+1, \ldots, i+t-1\}$, which reflects the current opinion about the bundle freshness and validity. All further votes are decided solely based on the local DAG G as described by Algorithms 2 and 3. Here WEIGHTEDAVG() assigns weights to bundles using the associated PoC value.

An important feature of the Tortoise protocol is that it succeeds whenever there is a small initial bias in voting of honest nodes. For that reason, if the voting margin θ is low, the bias is enforced by the weak common coin bit – a random bit most honest nodes agree on.

4.4 Weak coin protocol

A weak coin with parameter $p_c \leq \frac{1}{2}$ is a protocol which outputs a single bit $b \in \{0, 1\}$ and has the following properties:

1. $\mathbb{P}(b = 0) \geq p_c$

Algorithm 2 Tortoise protocol voting (B votes for A)

```
function Vote($A, B$)
    if $B.round < A.round + s$ then                                  ▷ Still deciding
        return 0
    end if
    if $A.round + s \leq B.round < A.round + t$ then       ▷ Vote based on Hare consensus
        return $B.votes[A.hash]$
    end if
    if not $A.accessibleFrom(B)$ then                            ▷ This is an old branch
        return $-1$
    end if
    $vote \leftarrow$ WeightedAvg([Vote($A, B'$) for $B' \in B.parents$])
    if $|vote| \leq \theta$ then      ▷ Vote margin is too low, use common coin if not abstain
        return $B.beforeCoinBit == 0\ ?\ B.coinBit : 0$
    else
        return Sign($vote$)
    end if
end function
```

Algorithm 3 Tortoise protocol decision whether A is included in the ledger

```
function IsInLedger($A$)
    $globalVote \leftarrow$ WeightedAvg([Vote($A, B$) for $B \in G$])
    if $|globalVote| \leq \theta$ then            ▷ This will not happen for old enough bundles
        return UNDECIDED
    else
        return $globalVote > 0$
    end if
end function
```

2. $\mathbb{P}(b = 1) \geq p_c$
3. Before the beginning of the protocol, for every honest party P, the adversary cannot guess the output of P with probability more than $1 - p_c$.

A practical and efficient implementation with $p_c \geq (1 - 2q)$ was suggested by Micali [12]. For a round i, each P with public key pk_K and signing key sk_K publishes the signature of the string $(pk_K||i)$. Upon receipt, each party computes the hashes of all incoming messages and takes the least significant bit of the smallest hash in the lexical order. Such an implementation is suitable for UDP broadcasting and gossip-based aggregation.

5 Hare Protocol

Even if the Hare protocol does not reach consensus, it will be finalized by the Tortoise protocol later on. The Hare protocol for a bundle X in our design eventually terminates with the decision "VALID", "INVALID" (when the confidence reaches a pre-defined threshold). If by the time the Tortoise protocol has to include the decision on X the Hare protocol is not terminated, the output is "UNDECIDED" (in which case the vote is 0). When the Hare protocol terminates, we guarantee that with overwhelming probability all honest nodes agree on the output and the decision is irreversible.

Below we consider an implementation based on the Snowball protocol of Avalanche family [14]. The Snowball version is suited for the permissionless setting as it is resilient against fullnode churn and incomplete view of the network participants.

An alternative way to implement the Hare protocol is by selecting random committees of relatively small size for each round, and let each committee run a version of BFT consensus (e.g. HoneyBadgerBFT [9]). The way committees are chosen typically rely on Verifiable Random Functions (e.g. Algorand [15]) or threshold signatures (Dfinity [6]). The main challenge to design such a scheme is to ensure that all participants agree on the committee members and mitigate long-range and nothing-at-stake attacks. While it's a promising topic for future research, we do not consider it in the present paper to keep it succint.

5.1 Snowball-based hare protocol

The implementation is based on the Snowball protocol of the Avalanche family [14]. For a bundle X published in round i, each party P keeps a binary array holding the P's confidence on how to to vote for X. The vote included by P in the bundles during "voting window" rounds $[i+t, i+s)$ is based on P's opinion about X at the time the bundle is published.

The parameters α, β, k should-be fine-tuned based on the security assumptions, as disscussed in [14].

Algorithm 4 Snowball initialisation for a bundle X

```
procedure OnQuery($X$)
    if not IsInDAG then
        Add($X$)
        Init($X$)
    end if
    Respond($pref$)
end procedure
procedure Init($X$)
    $Conf[+1] \leftarrow 0$
    $Conf[-1] \leftarrow 0$                                   ▷ The vote towards which we currently lean
    $pref \leftarrow \{X$ has been received before $start_{i+1} + \delta\}$ ? 1 : -1
    $Conf[pref]$++
    $last \leftarrow v$
    $strike \leftarrow 0$                                     ▷ Number of consecutive consistent queries
end procedure
```

Algorithm 5 Snowball loop for a single bundle

```
procedure SnowBallLoop
    $\mathcal{P} \leftarrow$ SamplePeers($k$)
    $v' \leftarrow \sum_{p \in \mathcal{P}}$ Query($p, X$)
    if $|v'| \leq \alpha k$ then
        Continue
    end if
    $vote \leftarrow$ Sign($v'$)
    $Conf[vote]$++
    if $Conf[vote] > Conf[-vote]$ then
        $pref \leftarrow vote$
    end if
    if $last \neq vote$ then
        $strike \leftarrow 0$
        $last \leftarrow vote$
    else
        $strike$++
    end if
    if $strike > \beta$ then
        Terminate($pref > 0$ ? 'VALID' : 'INVALID')
    end if
end procedure
```

For each bundle an instance of Snowball is run, resulting in a binary consensus indicating if the bundle is included in the round.

The advantage of the Snowball protocol is that it is seamlessy integrated in the gossip protocol on the broadcasting phase. The initial value for the bit is set simply based on the condition that the first time a bundle is pulled from a peer is less (measured by the local clock) than $start_{i+1} + \delta$.

The peers are sampled with weights proportional to PoC over the last M rounds. The parameter M should be taken large enough so that whp adversarial nodes have weight at most q of the population whp, as guaranteed by 1. Note that it is not necessary for the honest nodes to explicitly agree on the value of M. At the network level sampling can be done using e.g. the FireFlies protocol [7] or using more specialised UDP-only protocols.

It follows immediately from the guarantees of the Snowball protocol that it will terminate in finite time whp. We also have safety and liveness:

Theorem 5.1. *There is a choice of parameters α, β, k such that with $q \leq \frac{1}{5}$ the following holds with overwhelming probability*[8]*:*

1. ***Liveness.*** *If X was published by an honest node, all honest nodes will terminate in the state* VALID

2. ***Safety.*** *If X has been received by a fraction of less than q of nodes by $start_{i+1} + \delta$ then all honest nodes will terminate in the state* INVALID

Proof. **Liveness.** Let N be the total number of nodes. Assume WLOG that each node has equal weight, so that peers are sampled from a uniform distribution.

It follows from the analysis in [14], Theorem 5 that if the system is in a state that at least $(1-2q)N$ nodes have $pref = x$ then confidence will grow linearly with the number of Snowball loop iterations, and thus all honest nodes will terminate with the output corresponding to $perf$ within a finite number of rounds in expectation.

But if X is published by an honest bundle, all honest nodes will receive it before $start_{i+1} + \delta$, and thus the initial state is absorbing.

Safety. The same argument as for liveness, but now at least $(1-2q)N$ nodes start with $pref = -1$.

□

[8]Say, at least $1 - 2^{-100}$

6 Related Work

DAG-based structures have gained significant attention as a possible alternative to the traditional tree-like structure of blockchains. The ability to handle concurrent updates always comes at the cost that a DAG only provides partial order, and thus the protocol should explicitly introduce a fork resolution rule.

Further, DAG can be used either to store blocks (blockDAGs) or single transactions. SPECTRE [16] is PoW-based blockDAG with block-voting. It has safety and (weak) liveness guarantees for non-conflicting transactions, and provides only partial ordering between blocks. While it suffices to build a cryptocurrency ledger on top, it has limited applications when on-chain smart contract functionality is needed. However, SPECTRE is the only PoW-based blockDAG protocol we are aware of which is provably secure merely under a *weak* synchrony assumption, which is as follows. There is a universal bound D on the message delivery time, but none of the protocol parameters depend on this bound.

Similar to that, we believe that safety and livenes of the Tortoise protocol hold merely by assuming (1). Our intuition is based on fact that the bounds in the proof of Theorem 4.2 in [4] are essentially independent of δ. We leave the task of deriving rigorous proofs for future research.

PHANTOM [13] fixes this at the expense of introducing a global network delay bound – in order to guarantee a condition similar to (1).

Both protocols favor safety over liveness (PHANTOM less so). Further, the decisive innovation of MeshCash is its cooperative nature, crucial to our IoT-inspired design with fullnodes serving as dedicated brokers for light clients. This should be compared with the typical approach when client transactions are picked up by random fullnodes.

Hashgraph [8] inspired a line of research around "virtual voting". The idea is that a DAG structure can be used in order to interpret the information available to network participants. It seems to assume perfect knowledge about network participants. Further, interpreting the way gossip messages are propagated may be computationally hard.

Avalanche [14] combines DAG references with multiple instances of Snowball run in parallel. The Tortoise consensus of HelixMesh interprets the DAG in a completely different way. Compared to Avalanche, children bundles may cast both negative and positive votes to progenitors. Further, by introducing rounds we cap the width[9] of the DAG by the number of published bundles in a round. In contrast, under high load transactions, Avalanche will likely attach to the same old parent in the DAG, thus increasing the time before children will provide a sufficient number of confirmations.

This difference aside, the Hare part of HelixMesh incorporates the main innovation of the Avalanche protocol family – namely, that one can randomly sample network peers for broadcasting and agreement *at the same time*. On the other hand, there is a clear advantage of using the Tortoise protocol on top. First, the Tortoise protocol gives an SPV-compatible certificate of acceptance. Second, since the Tortoise protocol converges as long as the majority of honest nodes vote in the same way, the acceptance criteria for the Snowball-Hare protocol can be more lax then required for a standalone Avalanche implementation. Finally, the decision of the HelixMesh is based on the time a bundle is seen by a node, and in fact the guarantees of the protocol hold even if the timeouts are different accross the network. Furthermore, it opens up a whole new avenue for *heterogenous* consensus protocols, i.e. whose execution may depend on the node. Indeed, deciding on the validity of X, the original opinion can be inferred by honest nodes in a different way. If it runs out that the opinions are correlated, the Hare protocol will converge fast, otherwise it will take longer. This can be used as a feedback loop to train individual oracles for a more and more accurate initial guess.

IOTA's [10] Tangle is an experimental PoW-based design when the DAG grows according to MCMC simulations run by network participants. It is unclear whether and under which conditions this approach can provably guarantee safety and liveness.

[9]That is, the number of nodes at a given distance from the DAG source.

Appendix: FullNode-client negotiation protocol

So far we only considered communication protocols between fullnodes. The following communication protocol ensures that fullnodes fairly handle transactions submitted by light clients, and departs from the typical assumption that client's transactions are handled by random peers.

The negotiation protocol is not a core part of HelixMesh, but rather an implementation detail of the HelixNetwork.

So off we go. We assume that each message is signed and contains a hash of the response of the previous party, so that it serves as a verifiable committment to the whole communication history. The light client is Alice, the fullnode is Bob. We assume that each fullnode holds a slashable collateral against protocol violation. The actual collateral size is decided by Bob himself, and may well be zero. In this case there are no guarantees against Bob's adversarial behaviour, which may be acceptable when both light clients and the fullnode are controlled by the same party. Otherwise, Alice likely chooses a fullnode with a collateral if censorship-resistance is critical.

1. Alice initiates a transaction
2. Bob responds with a required handling "fee". The fee will typically depend on the chosen implementation of PoC and may be
   (a) IOTA-style client-handled PoW
   (b) Transaction fee paid in native currency
   (c) Proof of deposit for subscription-type schemes
   (d) Computing task output (for fog-computing use-cases)
   (e) Pay-in-data (fullnode can sell the data in the tx payload)

   Typically the condition should be available beforehand through a node's public API or aggregated boards, so the client can choose the best conditions without additional chit-chat. In this case, Alice would proceed directly to the following steps.
3. Alice requests the task (if applicable)
4. Bob ACKs and responds
5. Alice sends a transaction with a payload satisfying the required handling contributions, with (an optional) timeout timestamp after which she'll try another node.
6. Bob ACKs with a timestamp if accepts or rejects with a code indicating the reason. Alice's tx must be included in Bob's bundle with the provided timestamp (otherwise Bob's deposit is slashed).
7. (Optional) If the timestamp is too much in the future, Alice can slash Bob by submitting Bob's message to the slashing smart-contract (provided it has accurate timestamping within a known margin and can verify that the timestamp is well in the future).
8. In case of rejection, timeout or for any other reason Alice may submit a request to any other fullnode.

If any of bundles published by Bob contain transactions *after* the timestamp, but Alice's transaction is still not published, Alice can provably slash Bob's collateral.

Attacks

Potential DoS attacks by Alice are mitigated by the onboarding contribution required for a transaction to be published. Since we assume PKI, the fullnode may sit behind proxy or load balancer.

The worst case for Alice would be that Bob "disappears" after the required payload has been provided. In case of subscription of fee-based contributions, the only inconvenience for Alice is that she has to resubmit the same transaction: even if the same transaction appears in multiple bundles only the first in the order will be spent. Thus, the fee is spent at most once.

When PoW or another external resource is required (like CPU time), the resources are essentially wasted and the only mechanism is an off-chain reputation system. By posting reviews and signed commitments, another user (e.g. lightnodes) may verify that indeed Bob went offline. In this case, the reward mechanism in the network should be on the pro-rata basis, so that the more transactions handled by a fullnode, the more coinbase rewards it gets. In this case off-chain reputation system will incentivise rational behaviour – light clients will hardly use fullnodes with zero or negative reputation.

Note that the mechanism of off-chain reputation itself can be made decentralised and censorship-resistant by running a message board on a general-purpose blockchain platform (e.g. Peepeth). Note, the *perceived* reputation (i.e. the personal opinion on wether a fullnode is reputable) can still be based solely on verifiable data (e.g. DAG itself and negotation logs) and may differ among the users.

References

[1] https://cosmos.network/.

[2] https://polkadot.network/.

[3] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. pages 1–10, 2013.

[4] I. Bentov et. al. Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies.

[5] Y. Gilad, Rotem R. Hemo, S. Micali, Georgios G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. pages 51–68, 2017.

[6] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[7] Håvard Johansen, André Allavena, and Robbert Van Renesse. Fireflies: scalable support for intrusion-tolerant network overlays. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 3–13. ACM, 2006.

[8] L. Baird. https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf. https://s3.amazonaws.com/hedera-hashgraph/hh-whitepaper-v1.1-180518.pdf.

[9] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of bft protocols. Cryptology ePrint Archive, Report 2016/199, 2016. https://eprint.iacr.org/2016/199.

[10] S. Popov. The Tangle. https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf.

[11] K. Samani. aggregation theory, thin protocols, and recentralisation: AUGUR edition. https://multicoin.capital/2018/08/08/aggregation-theory-thin-protocols-and-recentralization-augur-edition/.

[12] S.Micali. Byzantine Agreement, Made Trivial.

[13] Y. Sompolinsky and A. Zohar. PHANTOM, GHOSTDAG: Two Scalable BlockDAG protocols. https://eprint.iacr.org/2018/104.pdf.

[14] Team Rocket. Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies. https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV.

[15] Y. Gillad et. al. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. https://people.csail.mit.edu/nickolai/papers/gilad-algorand.pdf.

[16] Y. Lewenberg Y. Sompolinsky and A. Zohar. Serialization of Proof-of-work Events: Confirming Transactions via Recursive Elections. https://eprint.iacr.org/2016/1159.pdf.