HelixMesh: a consensus protocol for IoT

Dmitrii Zhelezov
dz@hlx.ai
Helix Research
Berlin, Germany

Oliver Fohrmann
of@hlx.ai
Helix Research
Berlin, Germany

ABSTRACT

We provide a technical exposition of the HelixMesh protocol and the underlying DAG-based transaction ledger. The HelixMesh is optimized for high-throughput IoT networks on the premise that most transactions carry only a data payload rather than a value transfer. The consensus algorithm is a novel implementation of the hybrid on-/off-chain MeshCash framework based on probabilistic peer sampling for the off-chain layer. We further introduce a flexible "Proof-Of-Contribution" adversarial model which supports both closed permissioned (with standard BFT assumptions) and persmissionless (e.g. based on Proof-Of-Work) networks.

CCS CONCEPTS

• **Networks** → **Network protocol design**; • **Computer systems organization** → *Peer-to-peer architectures*; Sensors and actuators.

KEYWORDS

distributed ledger; consensus protocol; byzantine fault tolerance; IoT

ACM Reference Format:
Dmitrii Zhelezov and Oliver Fohrmann. 2019. HelixMesh: a consensus protocol for IoT. In *2019 International Electronics Communication Conference (IECC) (IECC ' 19), July 7–9, 2019, Okinawa, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3343147.3343168

STRUCTURE OF THE PAPER

In Section 1 we motivate our focus on specialized DLT for IoT and argue that general-purpose blockchains are not the best fit for IoT use-cases.

Section 2 outlines how the Directed Acyclic Graph (DAG) data structure can be combined with a consensus protocol in order to reach global consensus on the transaction log. This part is largely independent of the consensus protocol *per se*, but outlines implementation options and challenges to consider.

In Section 3 we introduce Proof-Of-Contribution (PoC) adversarial model and the protocol communication assumptions.

An in-depth exposition of the on-chain and off-chain consensus layers of HelixMesh is given in Section 4 and Section 5 respectively.

A brief summary and general discussion is given in Section 6.

IECC ' 19, July 7–9, 2019, Okinawa, Japan

ACM ISBN 978-1-4503-7177-3/19/07…$15.00
https://doi.org/10.1145/3343147.3343168

1 INTRODUCTION

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, etc.) have focused mostly on value transfers represented by the underlying cryptocurrency or a token. To the best of our knowledge, however, there has not been a similar growth in the development of DLTs for machines, specifically for interconnected devices, or simply, the Internet of Things (IoT). In the current article, we describe a new protocol for IoTs requiring DLTs, the HelixMesh protocol.

Specialized 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 decentralized 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 framework introduced in [2] allows one to eventually achieve the required level of safety in mission-critical applications.

The off-chain layer is inspired by a powerful insight, used in the Avalanche [10] protocol family, that random peer gossiping can be turned into a robust consensus protocol stable even under network churn.

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 emphasize that a bundle can contain heterogeneous data packets not limited to the scope of value transactions[1].

[1] E.g. auditable logs of IoT-sourced data streams.

Each bundle contains an ordered list of transactions[2] 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 commitments significantly reduce network traffic and separate the consensus logic from validation.

The protocol is agnostic with regard to the transaction semantics. Indeed, we believe that a message-driven architecture is better suited for data-driven IoT ledgers compared to the widely accepted approach of a distributed replicated state machine, taken by Ethereum and most of the general-purpose blockchains. Since this topic is orthogonal to the consensus protocol, we leave the details for a separate publication.

The protocol can be broken down into the following steps.

(1) Consensus about the set of commitments $\{R(k, pk_i)\}$ in round k
(2) Ordering $R(k, pk_i)$, the set of commitments
(3) Obtaining the ordered sequence of transactions with commitment to $R(k, pk_i)$
(4) The canonical sequence of transactions is defined by successive expansion of transactions with commitment 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(⋃_i R(k, pk_i)) do
3:          T ← GetAndVerify(R)
4:          for tx ∈ 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 deterministic 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[3].

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

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

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

$$B^{(k)}_i = T^i_1, T^i_2, \ldots.$$

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 availability which is beyond the scope of the present paper. For concreteness, the reader may assume that each bundle contains an IPFS link to its contents. Even such a simplistic solution will provide strong guarantees as long as there is at least one party seeding the 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.

A more sophisticated approach for securing data availability may rely on erasure coding and succinct proofs (see e.g. [1]) and further developments in this active area of research. We omit further 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 protocol implementation and can either be based on stake slashing 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. 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. Communication between fullnodes is done through gossiping on the overlay network. We assume that each honest node can efficiently sample random peers.

3.1 Synchrony assumptions

For simplicity 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). The parameter δ is

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

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

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-synchronized 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 [8], Definition 2. We defer formal proofs to a forthcoming research paper.

Further, we assume that full nodes have local clocks synchronized 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. UTC synchronization assumption is not essential to the protocol and may be dropped at the expense of having less predictable timing for protocol rounds.

According to [3] it takes about 1s to gossip a 1KB message to 90% of the nodes in the Bitcoin network, and Algorand [4] 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, gossiping can be used only as a fall-back for a more efficient network topology with a few high-throughput network relayers (by using e.g. bloXroute [7]). In this case, δ can be lowered to less than a second. The exact value depends on the liveness-safety trade-offs dictated by the use-case in mind.

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

3.2 PoC adversarial model

The adversarial model we use is a novel generalization of the standard BFT assumption that at most one third of the actors is malicious. However, it is rarely the case such an assumption can be guaranteed in practical DLT implementations. First and foremost, the set of participating nodes is likely to change over time. Next, typical BFT assumptions do not distinguish between offline and adversarial node, thus silently assuming that the majority of network nodes always actively participate in the consensus protocol. In order to address this issue, we use a device we call Proof-Of-Contribution (PoC). Our adversarial model assumes that in order to participate in the network any node should submit a "contribution" to the network, and that over a long run adversaries can contribute only a fraction of the total network contribution. Formally, we assume the following. The protocol is split into consecutive independed rounds (as explained in Section 2. Let q be the parameter representing maximal Byzantine tolerance of the protocol.

Definition 3.1 (q-byzantine tolerance). *For any $\epsilon > 0$*

$$\mathbb{P}(Q > (1+\epsilon)qW) \leq \exp\left(-\frac{\epsilon^2}{10}qW\right), \tag{1}$$

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

The probability space is generated by the underlying random process of the protocol. For deterministic protocols (1) simply degenerates into $Q \leq qW$.

Definition 3.1 is natural for Proof-Of-Work based adversarial models since the block production can be modelled by a Poisson process with intensity equal to the hardware hashrate. Then the q-tolerance corresponds to the assumption that honest nodes control at least $1-q$ of the total network hashrate.

However, such an abstraction works equally well for the classical BFT assumption if one assumes that during each round network participants contribute a virtual "membership token" minted at round start. Indeed, such membership-based PoC seamlessly accommodates network churn. Similarly, Proof-Of-Stake schemes can be modelled if one assumes that a locked stake generates interest counted as PoC contribution.

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

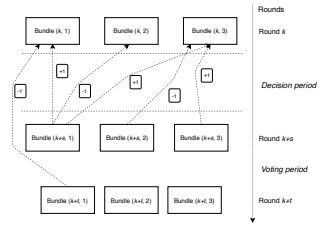
In rare occasions it may happen that the Hare protocol will not provide high enough confidence by the time the vote has to be included in the bundle, in which case the vote is neutral and the outcome is decided by the Tortoise protocol later on. Even without any assistance from Hare (i.e. even in a standalone 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[4]

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

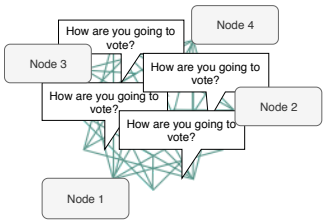
Theorem 4.2. *For any given* $\epsilon > 0$ *the Tortoise protocol eventually reaches* ϵ*-final consensus about the validity of an arbitrary bundle* X.

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).

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



(a) On-chain Tortoise voting



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

Figure 1: Tortoise and Hare consensus

Theorems 4.1 and 4.2 have been proved in [2] under the following conditions:

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

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*fk‼*s* through *i*fk‼*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.

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 ≤ 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 ← WeightedAvg([Vote(A, B′) for B′ ∈ B.parents])
    if |vote| ≤ θ 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 ← WeightedAvg([Vote(A, B) for B ∈ G])
    if |globalVote| ≤ θ then                               ▷ This will not happen for old enough bundles
        return UNDECIDED
    else
        return globalVote > 0
    end if
end function
```

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

A practical and efficient implementation with $p_c \geq (1 - 2q)$ was suggested by Micali [9]. 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 [10]. 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 [8]). The way committees are chosen typically rely on Verifiable Random Functions (e.g. Algorand [4]) or threshold signatures (Dfinity [5]). 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 succinct.

5.1 Snowball-based hare protocol

The implementation is based on the Snowball protocol of the Avalanche family [10]. 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 discussed in [10].

Algorithm 4 Snowball initialization for a bundle X

```
1: procedure OnQuery(X)
2:     if not IsInDAG then
3:         Add(X)
4:         Init(X)
5:     end if
6:     Respond(pref)
7: end procedure
8: procedure Init(X)
9:     Conf[+1] ← 0
10:    Conf[−1] ← 0      ▷ The vote towards which we currently lean
11:    pref ← {X has been received before start_{i+1} + δ} ? 1 : -1
12:    Conf[pref]++
13:    last ← v
14:    strike ← 0      ▷ Number of consecutive consistent queries
15: end procedure
```

Algorithm 5 Snowball loop for a single bundle

```
1: procedure SnowBallLoop
2:     P ← SamplePeers(k)
3:     v' ← Σ_{p∈P} Query(p, X)
4:     if |v'| ≤ αk then
5:         strike ← 0
6:         Continue
7:     end if
8:     vote ← Sign(v')
9:     Conf[vote]++
10:    if Conf[vote] > Conf[−vote] then
11:        pref ← vote
12:    end if
13:    if last ≠ vote then
14:        strike ← 0
15:        last ← vote
16:    else
17:        strike++
18:    end if
19:    if strike > β then
20:        Terminate(pref > 0 ? 'VALID' : 'INVALID')
21:    end if
22: 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 seamlessly 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 [6] or using more specialized 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*[5]*:*

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 [10], 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 $pref = -1$.

□

6 CONCLUSION

Scalability is one of the major bottlenecks of present day blockhains and significantly limits the number of use-cases for IoT.

The HelixMesh consensus protocol presented in the paper departs from the usual all-to-all transaction broadcasting model and the linear structure of blockchain in favour of the DAG structure supporting concurrent writes. The protocol is agnostic with regard to the semantics of the transaction. It scales up to thousands simultaneous bundle commitments per round, making it a robust Layer 1 protocol for an auditable log of IoT-sourced transactions. With the conservative estimate of 1000 fullnodes as edge servers each providing access for 1000 IoT devices, it is projected to accommodate up millions of connected sensors.

ACKNOWLEDGMENTS

The authors would like thank Bhaskar Krishnamachari (USC Viterbi), Gowri Sankar (USC Viterbi), Daniel Cook (Helix Research), Matt Niemerg (AlephZero) for valuable inputs and insightful discussions.

REFERENCES

[1] M. Al-Bassam, A. Sonnino, and V.k Buterin. 2018. Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities. *arXiv:1809.09044* (2018).

[2] I. Bentov, P. Hubáček, T. Moran, and A. Nadleret. [n. d.]. Tortoise and Hares Consensus: the Meshcash framework for incentive-compatible, scalable cryptocurrencies. ePrint: https://eprint.iacr.org/2017/300.pdf.

[3] C. Decker and R. Wattenhofer. 2013. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*. IEEE, 1–10.

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

[4] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 51–68.
[5] Timo Hanke, Mahnush Movahedi, and Dominic Williams. 2018. Dfinity technology overview series, consensus system. *arXiv:1805.04548* (2018).
[6] Håvard Johansen, André Allavena, and Robbert Van Renesse. 2006. Fireflies: scalable support for intrusion-tolerant network overlays. In *ACM SIGOPS Operating Systems Review*, Vol. 40. ACM, 3–13.
[7] U. Kalman and et. al. [n. d.]. bloXroute: A Scalable Trustless Blockchain Distribution Network. https://bit.ly/306YsIH.
[8] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. 2016. The HoneyBadger of BFT Protocols. ePrint: https://eprint.iacr.org/2016/199.
[9] S.Micali. [n. d.]. Byzantine Agreement, Made Trivial. preprint.
[10] Team Rocket. [n. d.]. Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies. preprint.