Hylochain and Blockchain Cohomology

Wyatt Meldman-Floch

February 17 2018

Abstract

We formally define the finite automata of the constellation protocol as well as prove the security, consistency and liveness of reputation based delegate selection.

Introduction

As it is not possible to globally parameterize blocks by time, we follow the body of research defining blockchain state as a topological fiber bundle[1]. Specifically, we formulate the constellation protocol as a probabilistic state machine carried by finite automata in a non-cooperative repeated interaction game. We go further proving security, consistency and liveness in a hierarchy of these state machines.

The Constellation Protocol

We formally define Constellation as the chain complex C [2] which in our context is just an ordered set of unordered sets of disjoint Radials

$$C : \{O_0 \dots O_n\} \forall n \in \mathbb{N}$$

where there is naturally an ordering given by n. This is isomorphic to the topological sorting of the set of all radials, where direction is given by "flow".

[1]

[2] chain complex link

Flow

There is a notion of "flow" between radials. There are two directions to our flow. The rightward direction gossips blocks from a Radials in a lower Order "upward" into the highest order, which maintains the *globalState*. The leftward direction gossips a blocks from Radials in higher orders "downward". These blocks contain the result of a higher order consensus, which "validates the validators". The contents of this block is used by the generator function for delegate selection, and partitioning.

Protocol Topology

We define the primitives of our DAG chain. We come to the conclusion that our DAG chain is a directed acyclic graph who's direction is given by simplex graph[3] κ and topological ordering follows the tiered ordering of R_K. The simplex graph $\kappa(G)$ of an undirected graph G is itself a graph, with one node for each clique (a set of mutually adjacent vertices) in G. As all of our simplexes are independent, $\kappa_j \mid j \subseteq t-1$ for tier t is formed as a disjoint subset of simplexes in $t-1$. We define κ as a mapping from an undirected graph G to a new graph $\kappa(G)$ who's vertices are cliques and compositions of cliques of G

$$\kappa : \sigma_j \to \sigma_k \mid j \subset t-1, \sigma_k \in R_{t-1}.simplices \tag{1}$$

Chain Fibering

Chain fibering is the formation of consensus blocks out of consensus blocks instead of transactions. This is varifiably sound up to isomorphism. We formulate notion of chain fibers by defining consensus in terms of a simplex graph κ. A chain fiber is given by

$$f \circ \kappa_j^k : \sigma_k \to Block_k \tag{2}$$

where κ_j^k is a simplex graph made up of chain fibers $Block_j$ in a preceding tier. The chain fibers from the preceding tier, $Block_j$, become the domain of the consensus function for tier k. We can say equivalently

$$c_k : \{Block \dots Block\}_j \to Block_k \tag{3}$$

We can formally define the mempool mem of simplex σ_k in tier t as

$$mem_k \equiv \{Block \dots Block\}_j \mid j \subset R_{t-1}.simplices \tag{4}$$

[3] wikipedia/Simplexgraph

Finite Automata and State Transitions

Node Abstract Data Type

There are four states a node can occupy, each node state corresponds to an interval of Orders in our chain complex.

$$\begin{aligned} S &= \{Gossiper, Delegate, Offline\} \\ \Sigma &: FSM[S] \\ \tau &: Receive[S] \\ s_0 &= Gossiper \\ F &= \{Offline\} \end{aligned}$$

Radial Distributed Data Type

A Radial is an instance of distributed data structure. A Radial is hosted by a collection of what in a traditional blockchain would be called a "node". As a Radial is a distributed data structure, it is formed by a collection of distributed machines sharing a state. A Radial shares an execution context across machines which seeks to increase the amount of valid data in a local cache, called a mempool, by providing an economic incentive to the nodes who propose the most transactions in the shortest amount of time. Formally we define a Radial as

$$\begin{aligned} R_k : & \\ & nodes : \{n_0 \dots n_k\}_k, \\ & mempoolCache : \{transactions/blocks\}, \\ & shard : \{(account, merkleTree)\}, \end{aligned}$$

where $mempoolCache$ is the contents of the mempool. $mempoolCache$ contains what will become a block through consensus. $shard$ is the shard of the global chain state which is used for validation.

A hypergraph represents an arbitrary set of subsets of it a graph's vertex set, where each subset is called a edge[4]. We define the the Radial abstract data type in terms of a hypergraph and mappings between vertex sets within that hypergraph. Specifically for given Tier $t \in \mathbb{N}$ (see above), we define the Radial monoid[5] R_t as a functor[6] over the category of simplexes such that

[4] Pal S. et. al. http://www.facweb.iitkgp.ernet.in/ spp/geomgraph.pdf
[5] https://ncatlab.org/nlab/show/monoid
[6] https://typelevel.org/cats/typeclasses/functor.html

```
case class Sigma(nodes: Node*)

trait Radial[T <: Tier] {
/*
simplex and starCluster are hyperedges
*/
   val starCluster: Sigma[T-1]
   val simplices: Sigma[T]* \\ note this is variadic
   def hyperPlane[T1, T2]: Sigma[T1] => Sigma[T2]
}
```

The hypergraph for all simplexes σ_k within Tier t is given by the two element set

$$H_{t,k} = \{I_{g_{t-1}} \mid g_{t-1} \subset G_{t-1}, \sigma_k\} \tag{5}$$

where $I_{g_{t-1}}$ is a subset of all simplexes in tier $t-1$ (as we know all simplexes are independent) and $\sigma_k \in R_t.simplices$. $hyperPlane$ is a function that maps between two hyperedges, potentially across tiers. Note that $Sigma[T-1]$ and $I_{g_{t-1}}$ are equivalent via homotopy as shown in Defnition 2.1 (footnote 5).

In the degenerate case of our definition of R_T, namely when $R_t = \{R_0\}$, $starCluster$ is a mempool as defined below, but of transactions which are isomorphic to all subtypes of BlockData via covariance.

Simplexes

The clique complex of a graph G is a simplicial complex whose simplices are the cliques of G[7]. We define a simplex σ as a completely connected graph (clique) of nodes, with which we can perform cryptographic consensus. Given $\sigma = \{n_0 \dots n_i\}$, σ is a simplex iff:

1: σ is a complete graph such that for a set of edges e, corresponding to nodes n,

$$\forall n \in \sigma, \exists e \in E \mid e \equiv \sigma \setminus \{n\} \tag{6}$$

Corollary: for each n in a a simplex σ_K, the simplicial star[8] of n, $st_K(n)$, for simplex K is equivalent to the σ_K that is:

$$\forall n \in \sigma_k, st_K(n) \equiv \sigma_K \tag{7}$$

[7] https://arxiv.org/pdf/1007.0418.pdf
[8] Closure, star, and link: wikipedia/Simplicialcomplex

2: There exists a deterministic mapping given by an immutable generating function g

$$\exists g : \sigma \to d, \mid d \subseteq \sigma \tag{8}$$

3: There exists the notion of a star cluster[9] Such that star cluster σ forms an independence complex of K.

$$SC(\sigma_k) = \bigcup_k st(n) \in I_G \ \forall k \in K \tag{9}$$

Where I_G is the set of all independence graphs of K[10]. It follows from corollary (4) that

$$\begin{aligned} SC(\sigma_k) &= \bigcup_k st(n) \\ &\equiv st(n) \ \forall n \in k \end{aligned} \tag{10}$$

Order ADT

Consider the disjoint set[11], Order: $O = \{R_0 \dots R_k\} | k \in \mathbb{N}$. An Order, like a Radial is a distributed data structure. It's purpose is to provide a distributed execution context for maintaining state across network partitions. A network partition is a subset of resources within a distributed system. A similar process to *consensus*, called *partition*, occurring within a Radial in ordering O, broadcasts instructions to Radials in the Order $O - 1$. These instructions tell nodes which Radial they need to join, and Radials what shard they need to host.

The shared execution context an Order uses is a distributed hash table. We follow the UrDHT[12] protocol to implement a state manager for our DHT. We dynamically rebalance caches of global chain state (shards) across the Radials within an ordering through our *repartition* subroutine. The primary purpose of the Ordering abstraction is to encapsulate this rebalancing within the

[9] https://arxiv.org/pdf/1007.0418.pdf

[10] Definition 2.1 https://arxiv.org/pdf/1007.0418.pdf

[11] disjoint over nodes that constitute Radials (a node belongs only to one Radial)

[12] `https://scholarworks.gsu.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1108&context=cs_diss`

repartition subroutine. Below we define the Order ADT

$$
\begin{aligned}
&O_n : \\
&\quad partition : \{(account, k)\}, \\
&\quad DHT : \{shard \dots shard_k\}_k,
\end{aligned}
$$

where the DHT is a service that maintains the sharding scheme (redundancy) routing between radials and partition maintains the valid partition of nodes across radials in the Order below.

Block Anatomy

We define a block as a sheaf on our protocol topology. The topological ordering between Orders follows the topological ordering of our blockchain. Our main block is a merkle tree that can be decomposed into a recursive hierarchy of merkle tree blocks. This hierarchy is a directed acyclic graph, just like our network topology which forms a dendrite like trie. The mirror symmetry of a dag within a dag is indicative of our blockchain cohomology[13] where each block is a sheaf[14].

This symmetry arises out of our use of chain fibers[15], which feed blocks from lower orders to upper orders. At the lowest order (sleeping state), blocks are transactions sent from individual nodes into the mempool of a Radial. At the first order, blocks are formed as the result the *consensus*, compressing transactions into a block of pointers then sending them to right, towards a higher order. At the highest order, blocks define the *state* of the entire chain, which we will refer to as *globalState*.

The cohesion of our blockchain is maintained by defining a node's economic incentive to other node's economic incentive through interweaving signatures. Blocks sent downward are referenced in blocks sent upward and every node signs every message passed during consensus. Trust, in this sense is formed by a merkle tree of signatures, allowing us to explicitly trace every action back to the node who performed it. Nodes that are highly trusted are allowed a spot in the higher orders.

Transactions are, type equivalently, blocks. In the same way we can prevent malicious forking through redundant sharding, we can use redundancy to prevent double spends. Double signing transactions allows a counterparty to enforce transaction ordering.

[13] Hylochain and blockchain cohomology
[14] links to sheafes and how they represent local data about a large dimensional space
[15] chain fibers

In order to prevent censorship via sybil attack, we engineer an economic incentive to perform consensus faster than it would take to forge the signatures that stitch together the contents of a block. This is almost like a lazily evaluated proof of work. We don't actually do work, we just use make it computationally infeasible to forge the validity of our data before the data is notarized by a radial in a higher order.

State Transitions

The following functions are used to form an edge between Radials. The most important is the *generator* method which is used in both *consensus* and *partition*. The symmetry in the use of *generator* to select delegates for *consensus* and for balancing shard/nodes in *partition* is what implements our economic incentive. *generator* is a deterministic function and at any point if a proposition is made by a node that does not adhere to the output of this function (is invalid), it will be notarized by all other nodes. The "right handedness" vs "left handedness" of flow correlates to the methods *consensus* and *partition*.

generator

The generator function implements delegate selection and is used in both the consensus and partition methods. The generator function essentially "reads" data that has been stored in a block and tells the node what to do during this round of consensus. *generator* is a function that takes two blocks and returns a subset of nodes within a Radial. *generator* is given by the following typedef

$$g : (block_{i-1}^{O}, block_{j}^{O+1}) \to \{n\} \subset R_k$$

where $i, j \in \mathbb{N}$ are block numbers and $i \geq j$. The result gives selected delegates when used for *consensus*. When used for *partition* it returns the set of nodes allowed to constitute a radial. *generator* internally uses MEME[16] to select nodes when implementing *consensus* and *partition*.

consensus

A Radial forms a block when a deterministic set of delegates agrees upon the contents of the radial's mempool. This block is broadcasted "upward" $O_n \to O_{n+1}$ towards Radials of higher Order. For R_k generating block number i within

[16] see section on proof of MEME

given Order O, the corresponding $block_i^O$ is given by

$$delegates_i : g(block_{i-1}^O, block_{i-1}^{O+1})$$
$$c : (delegates, mempool) \rightarrow block_i^O$$

Where c is our consensus algorithm which is currently the HufflepuffBFT[17]. We now refer to consensus blocks as cBlocks.

partition

The *partition* function is homotopically dual to consensus[18], operating in the reverse direction. Metadata in the form of a block (which we will refer to as a pBlock) is broadcasted "downward" $O_n \leftarrow O_{n+1}$. The pBlock is composed of instructions for a lower order radial to repartition itself. When a $pBlock_i$ is received from a Radial in O_{n+1} a Radial in O_n must repartition itself before creating $cBlock_i$. It is then used as a deterministic seed, along with a the previous $cBlock_{i-1}$, to determine delegates for a lower order radial. Using our new notation, we have

$$p : (cBlock_{i-1}^O, pBlock_i^{O+1}) \rightarrow R_{k_i}$$

Rebalancing shards across partitions is handled as a subprocess and detailed as a state transition.

Proof of Meme

MEME is a probability distribution of reputation scores that is used for delegate selection. It was designed to increase fault tolerance to over 50% in some cases[19] by preventing the possibility of a sybil attack. MEME is a variation on GURU[20] which is an equivalent of GHOST for Proof of Stake blockchains. MEME is our version of GURU like CASPER is Ethereum's GHOST implementation.

Security consistency and liveness guarantees

[17] ACS with interwoven signatures
[18] Hylochain, section on metamorphism
[19] guru abstract
[20]