# A formal definition and statistical model checking of the constellation blockchain

February 27, 2018

**Abstract**

## Introduction

## Fundamental Data Structures, Types and Functions

### T <: Numeric

Numeric type delineating tier within multi-tier hierarchy

### BlockData[T] <: U

Consider a type *BlockData* which serves as the universal parent type of all data going on chain. All transactions, validator requests, meme data etc. All BlockData is isomorphic and thus all BlockData can be compressed into a Block.

### Block[U]

Contains compressed form of Block Data. This is the result of consensus. Each unit of *BlockData* must reference the previous round of consensus, this is satisfied by containing the *Block* hash. it is worth noting that $Block[U] \equiv Block[BlockData[T]]$ through covariance.

### Transaction <: BlockData[0]

Transaction is a subtype of BlockData that can only exist on the bottom tier. It is the building block of our currency.

## LeftHand <: RightHand <: Transaction

Note: LeftHand and RightHand are symmetric subtypes of Transaction. For a Transaction to be valid it must be 'signed' by the counterparty. This is satisfied by sending a RightHand transaction that references a LeftHand transaction.

#### Definition: Consensus Cluster $:= collection[node]$

A set of validators, undergoing cryptographic consensus to produce a *Block*.

#### Definition: Validator

A node that has 'woken up' by starting a *consensus* process. It now gossips transactions to it's neighbors and is waiting to be selected as a Delegate.

#### Definition: Delegate

A node who has been selected to perform consensus. Delegates are implicitly chosen on each node locally. Once a new *Block* is received, it is passed to the Generator Function, which tells this node if it is a delegate.

### Generator Function $g : Block \rightarrow$collection[node, reputation score]

This function is used to determine the next set of delegates from a given set of validators (consensus cluster). This function is also used to update reputation scores. See GURU for examples.

### Consensus (function) $f : collection[node] \rightarrow Block$

A function that maps a consensus cluster to a Block.

## General Architecture

### Node Architecture

A node consists of two processes, and a remote procedure call (rpc) interface.

```
/**
  * This needs to be refactored to spin up one or two processes, one for
      protocol and one (or zero) for consensus
  * the main method to this singleton needs to execute a hylomorphic
      method that takes types representing these actors
  * ex:
  * def hylo[F[_] : Functor, A, B](f: F[B] => B)(g: A => F[A]): A => B =
     a => f(g(a) map hylo(f)(g))
  */
object BlockChainApp extends App with RPCInterface {
  implicit val system: ActorSystem = ActorSystem("BlockChain")
  implicit val materializer: ActorMaterializer = ActorMaterializer()
  implicit val executionContext: ExecutionContextExecutor =
      system.dispatcher

  val config = ConfigFactory.load()

  override implicit val timeout: Timeout = Timeout(
    config.getInt("blockchain.defaultRPCTimeoutSeconds"),
        TimeUnit.SECONDS)

  val logger = Logger("BlockChainApp")

  val id = args.headOption.getOrElse("ID")
  val chain = new Chain()
  val protocol = system.actorOf(new Protocol (chain), "constellation")
  val consensus = system.actorOf(new Consensus, "constellation")
  Http().bindAndHandle(routes, config.getString("http.interface"),
      config.getInt("http.port"))
}
```

The *protocol* process handles transaction signing and essentially all functionality to send and receive payments. The *consensus* process implements the responsibilities of a validator node. Nodes can be 'sleepy', turning on the *consensus* process at will.

The *chain* class is a local blockchain made up of all transactions/interactions with the chain that this node has made. Each link in the chain is a sub type of BlockData.

The rpc interface $RPCInterface$ connects the node to the outside world through a set of endpoints for sending/signing transactions and 'waking up' the node by starting a *consensus* process.

## Underlying protocol

### Consensus

Consensus is the process of forming a $Block$, it can be thought of as a function $f : collection[node] \to Block$. Consensus clusters are formed in a tiered hierarchy. The top most tier forms the global state of the chain, which is made up of $Block$s from preceding tiers. Each tier, from top down until the second to last, has responsibility for routing transactions and validating the blocks from consensus clusters.

## What is the lifecycle of a transaction

When a transaction is sent, it is sent from a node to a higher tier which 'routes' the transaction to the consensus cluster(s) that host its 'shard', or the shard of the blockchain that host's it's public key's history. Each transaction has a left and right half. The initiator of the transaction sends the $LeftHalf$ to the network. Its $LeftHalf$ is then referenced by the $RightHalf$ which is sent by the counterparty.

## How is consensus performed

Consensus $f : collection[node] \rightarrow Block$ is the act of creating notarized data in the form of $Blocks$. In our case, we are using the HoneyBadgerBFT, which prevents against sybil using encryption [1]. Nodes are rewarded based upon successful completion of consensus, the number of transactions they provide, and metadata about their performance. This is all calculated post facto by proof of meme and rewards are given within a set interval of blocks.

## How is this secure/fit in with our incentive model

Double spends across asynchronous consensus is prevented by double signing transactions. Consensus clusters are rewarded for processing transactions and sybil attacks are mitigated via encryption in honeybadgerBFT. We also are able theoretically improve typical byzantine fault tolerance over 40% thanks to GURU and our reputation model. Ddos attacks can be mitigated via throttling of accounts with low reputation scores. I propose an incentive for routing where each node that routes a transaction signs the tx, and when a tx is notarized each account that routed the tx is given a reputation increase.

[1] ch. 4.3 https://eprint.iacr.org/2016/199.pdf

## Scaling Tiers

### Hylochain: Recursive Parachains

### Routing and Distributed Data Storage

## DAG Architecture

### Definition using chain complexes

### Scaling formula

### Dynamic Partitioning for Recursive Parachains

## Proof of meme

### Proof of meme and consensus

### Proof of meme and routing