HyperCycle: A Lightweight Agent-System-Based Ledgerless Blockchain Architecture, Supporting Secure, Efficient, Scalable, Cross-Chain AI Microservices,

Ben Goertzel[1], Toufi Saliba[2], Dann Toliver[2] Anton Kolonin[1], Sergei Rodionov[1]

[1] SingularityNET Foundation
[2] TODA Network
May 12, 2023

Abstract

A novel ledgerless blockchain architecture called Hypercycle is presented, oriented toward secure, inexpensive, high-speed large-scale on-chain execution of microservices, especially (but not exclusively) those carrying out AI-related functions. This fully peer-to-peer HyperCycle blockchain is founded on the TODA/IP P2P communication protocol, the TODA asset model, SingularityNET's Proof of Reputation system and the MeTTa AI programming language drawn from the OpenCog Hyperon AGI framework.

In blockchain-world lingo, one can think of HyperCycle as multilayer, layer-fluid or "Layer 0++" – meaning it starts at the level of basic secure communication protocol, then adds layers from there, and can play the roles associated with various traditionally conceived layers (core protocol, base blockchain, sidechain,etc.) depending on context..

A HyperCycle network consists of a population of autonomous agents, communicating in a secure p2p manner, each owning their own transaction history and accumulating their own reputation, grouped together in rings in which they collaborate to execute consensus mechanisms enabling execution of instantaneously deterministic nano-transactions based on cryptographic proofs of computation performed in a p2p setting with zero third party dependencies.

HyperCycle agents are constructed as specific TODA files/packets (Sato-Servers), complete with the TODA transaction and cycle trie mechanisms for secure strongly decentralized management of file and network history (the cycle trie being the inspiration for the name "HyperCycle"). Consensus mechanisms involved in HyperCycle rings may be ledgerless e.g. based on hierarchically sharded ledgers, depending on the particular requirements. However, if ledgers are used they play a supplementary rather than foundational role (e.g. supplemental usage might be for efficiently broadcasting proofs, or for ledger-based long-term backup storage of a selected fraction of on-HyperCycle transactions or aggregate of transactions).

Brief discussion is given here on the particulars of HyperCycle customization to key application areas like swarm AI, rating and reward in media networks, decentralized payments and computer processing, and public/private chain interoperability. A key strategy for enabling broad usability will be the creation of MeTTa-based DSLs focused on smart contract programming in specific vertical areas. Such DSLs may straightforwardly be presented to the user as low or no code frameworks auto-generated from the underlying MeTTa code, effectively leveraging the HyperCycle framework behind the scenes.

Contents

1 Introduction

Over the years since the Bitcoin whitepaper was released in 2008, a variety of different blockchain technologies and networks have been created, leveraging roughly Bitcoin-like underlying architecture, some cases progressive evolutions while others are regressive. However, New concepts such as smart contracts, DAOs and multi-layer networks have been added to the picture, increasing functionality but in some cases moving further rather than closer to the original vision of fully peer-to-peer (p2p) strongly secure networks.

The complexity of the interdependencies between various desirable properties such as speed, security, scalability, usability, and decentralization have become much clearer in the interval since Bitcoin emerged.

It has also become clear, given the diversity of tools and chains that have emerged with different strengths and weaknesses, that there may not end up being One Blockchain Network To Rule Them All. Rather, we may be facing an ecosystem comprising a decentralized network of decentralized networks, each architected with underlying mechanisms that combine roughly the same underlying math and computer science in ways specialized to suit particular classes of use-cases.

HyperCycle comes into this scene from a novel direction, and it can be viewed from multiple perspectives associated with the multiple "layers" commonly identified in blockchain systems:

- **Layer 0**: HyperCycle can be viewed and used as a "Layer 0" technology, in the sense that
  - It has aspects that sit inside the network packets and operates on that level below any and all existing siloed AIs, blockchains and ledgers.
  - The reputation system underlying its Proof of Reputation consensus system is most logically construed as operating across blockchains and incorporating off-chain information as well.
  - Its tools for accelerating smart contract execution work across multiple VMs, via intervening on a lower layer than typical consensus-based mechanisms for validating smart contract execution results
- **Layer 1**: HyperCycle can be viewed and used as a "Layer 1", in that it can be used as an independent blockchain network for mediating p2p interactions between participants to manage and exchange values to acquire AI computation but also to transact the value created via AI computations.
- **Layer 2**: HyperCycle can be viewed and used as having "Layer 2" aspects. It can interoperate very closely with other blockchains, so when desired it can interact with another chain much in the manner that a sidechain does. Further, in some cases it can provide proofs of the correctness of its activity which can be verified by other chains,

rendering its interaction "Layer 2" like in a comparable sense to various sorts of rollups.

- **Layer 3**: HyperCycle can interoperate with existing Layer 2 sidechains in a similar manner to Layer 1 chains, if these Layer 2 chains have unique characteristics to be leveraged

This seamless "layer-fluidity" is achieved using core algorithms and structures drawn from the TODA asset model [GGT19] and the TODA/IP ledgerless blockchain [ST19] , along with the Proof of Reputation concept [AK21] and MeTTa AI programing language developed for use within the SingularityNET AI/blockchain network.

Along with Proof of Reputation, MeTTa and other tools used within HyperCycle, SingularityNET [Goe19] [GGH+17] [MG19] also provides some of the motivation underlying HyperCycle's creation. SingularityNET is a decentralized protocol and platform for communication and coordination between AI agents and their users, originally implemented on Ethereum and now in the midst of a port to Cardano.

A core issue experienced with the Ethereum based version of SingularityNET has been that, due to the temporal and financial cost of using the blockchain, AI developers are driven toward design patterns in which individual agents carry out massive and complex AI processes internally at a modular level, using the blockchain only for relatively infrequent communications or for setting up appropriate secure persistent channels. The Cardano version is expected to palliate this problem somewhat, but not sufficiently to allow the highly granular use of blockchain within AI-system operations that would be preferred. HyperCycle will solve this problem to a significantly greater degree. Via radically decreasing the cost and increasing the speed of blockchain transactions, HyperCycle allows design patterns involving lighter-weight AI agents interoperating more frequently in more diverse ways, and therefore opening new doors for intelligence to emerge from cooperative dynamics Of course these improvements will come with some tradeoffs which are acceptable in the SingularityNET application context but would be unappealing in some other applications.

The deployment of SingularityNET on HyperCycle will require HyperCycle to demonstrate rich and fine-grained interoperability with the Cardano and Ethereum blockchains, because many useful SingularityNET agents depend for their operation on interactions with other third-party processes

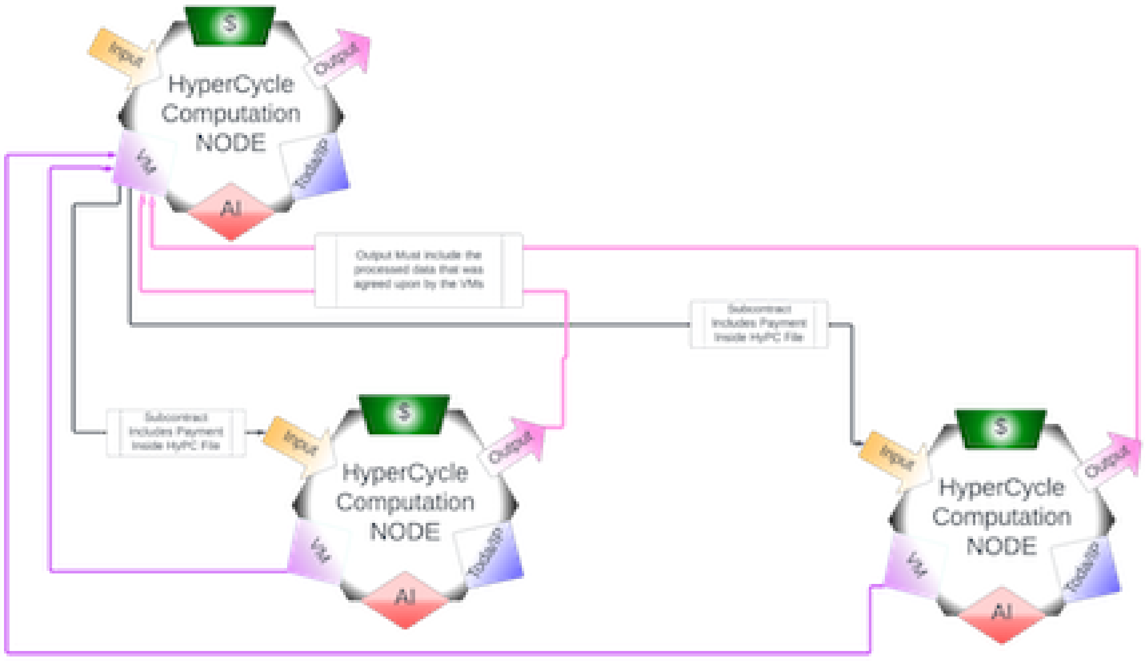
$
Input
Output
HyperCycle Computation NODE
VM
Toda/IP
AI
Output Must include the processed data that was agreed upon by the VMs
Subcontract Includes Payment Inside HyPC File
Subcontract Includes Payment Inside HyPC File
$
Input
Output
HyperCycle Computation NODE
VM
Toda/IP
AI
$
Input
Output
HyperCycle Computation NODE
VM
Toda/IP
AI

Figure 1: HyperCycle Computation Nodes Interacting

running on these other chains. The HyperCycle design in fact supports interoperability with these and other chains to an unprecedented degree. We have taken serious looks at Bitcoin, Cardano, Ethereum, Cosmos, Polkadot, Algorand, Avalanche and NEM, and believe it should be tractable for HyperCycle smart contracts to operate in close communication and coordination with smart contracts running on all these chains and others. That said, the overheads and limitations and strengths of cross-chain operation will be different in each case. In the Appendix to this document we discuss some of the particular aspects of interoperation with Cardano blockchain, which is one of the cases we've thought through in the most detail so far.

Figure 1 showcases a single HyperCycle Computation Node subcontracting 2 other nodes simultaneously and zero dependencies. Yet it is crucial to be reminded that no finality is ever reached without having something global being impacted. We cover that in details in Toda/IP.

1.1 Enabling Fast, Inexpensive, Large-Scale On-Chain Micro-transactions

One important class of use-cases for which HyperCycle is intended is where:

- You have a large number of transactions that need to happen in a short period of time such as HPC centers (a single HPC center can demand over 26 million micro transactions per second to have finalities in milliseconds, highest security and near zero cost)
- You don't need to see everyone else's transactions but only those related to you
- While prompts must happen in real time, they must be preceded by committed transaction in real-time [1]

While this is not the only class of situations HyperCycle can be applied to, it is the particular sort of use-case that has been most firmly in mind while developing the design, and it is envisioned as playing a large role in shaping the initial implementation.

The HyperCycle design fulfills these requirements (as well as those of other use-cases) via leveraging

- TODA/IP's ability to coordinate a large number of secure blockchain transactions without the necessity of maintaining any replicated ledgers. Effectively zero-dependencies
- The SingularityNET weighted liquid rank reputation system's capability to assess the contribution of an agent to a network and thus the reliability of that agent to assist with various network functions.

[1] In one of the TODA asset model, which is heavily leveraged within HyperCycle, you can "perform" any transaction you like, any time you want, even ridiculous completely made up ones, but anyone looking at that transaction can completely verify its integrity, based only on the information in the transaction's proof of provenance. So no bad transaction can provide evidence that it is a good transaction, and every good transaction has standalone evidence proving its veracity to anyone who takes time to look. However, the transaction may still take time to reach finality globally, but in the meantime and within the cycle and its reach to other cycle, the finality is immediate, which is unprecedented at this time this document was last updated. The question then becomes, in a given application, whether to accept a partially complete transaction or insist on a fully complete one before proceeding.

- The MeTTa AI language developed in the OpenCog Hyperon AGI project, made blockchain-savvy via integration with the rholang/RSpace tools developed in the RChain project [2]
- The ability to interoperate closely with smart contracts defined for other chains, including Plutus, Solidity and others

1.2 HyperCycle Multi-Tokenomics

First let's start by answering "why" does it need a token? Effectively what is the token's absolute utility? The native HyPC native token is absolutely needed for each machine to initiate any work, not by consensus but by literally "absolute need" for the packet to make it across, which is where the token resides in what's called Sato-Server, which was one of the successful evolutions that came out of Toda/IP towards a global data structure called Earth64. While this may sound unnecessary, further below folks would get the aha moment. . Given than the cryptographic proofs are embedded inside the 64KB Toda file aka the Sato-Server that is actually the native HyPC token, it means the proof of provenance of that token must also be included. It is crucial for the AI computation node to know with certainty that it will get paid if does the work. Therefore the payment is actually embedded in the set of instruction with the inquiry to do the work, if the payment is validated, then the VM sends the instructions with the data that needs to be processed to the AI machine. This means the AI machine is actually managed by the VM. In-order for the provenance to be of significance, the AI machine needs to ensure that he payment comes from a source first that has been identified by homo-sapiens as money. (why homo-sapiens? Well they are the ones that run this planet and must identify this as money, otherwise AI won't be able to buy electricity or hardware to sustain itself and evolve) Hence the token listings is the very first step towards the provenance of that proof. The fact that someone paid money for the token and everyone pays money for the token is sufficient for the machine to utilize that proof of provenance as valid. Mainly because the machine needs to know if it's done the work requested (modular AI), can it actually use this token to pay for electricity or whatnot. This gets even more interesting when the HyPC is not only used as the method of payment but to also carry payments of other currency inside it along with the actual identity of the machine.. Hence HyPC value is due to

[2]https://rholang.io/

its utility and the more demand of that utility the more it is likely to increase in value as its supply is finite but the demand is not likely to ever slow down.

There are 3 main utilities of the token, 2 of them are technical:

1- Each node must have 1024 HyPC to initially surface in the network and have an identity that can carry proofs of computations, reputations and Uptime which can enable the node to multiply without any additional licenses or tokens. Up to 1024 node.

2- Each node must be able to communicate with other nodes values and therefore when a computer has finished processing, the cryptographic proof of its work is sufficient to re-consume the token received.

3- Can embed other tokens or currency within it and therefore gives flexibility for AI compute to have any currency for it to effectively operate.

The tokenomics of HyperCycle also have unique aspects. There is no transaction fee per se for the AI compute transactions running on the network, but a similar role is played by *royalties*, the payment obligations associated with which are split up automatically and deterministically among participants in the network to ensure all parties are properly incentivized. Royalties and exchange of value between nodes involves the native HyPC token, but in particular applications can include a variety of other value instruments, say USD, EURO, WON, AGIX, ADA, ETH, USDC, Ava, Algo, etc. For instance,

- HyPC native tokens, paying for the overall AI computation including tools and mechanisms enabling various HyperCycle rings to provide consensus, information propagation and other useful functions
- AGIX tokens to pay for the AI functionality required to maintain an uncorrupted reputation system.
- ETH, BTC, ADA or others can be used to pay for the long-term storage of partial snapshots of the HyperCycle network state.

1.3 HyperCycle Multi-Ring Structure

A HyperCycle network (as loosely depicted in Figure 2) comprises a collection of "rings", each containing a set of HyperCycle agents, each of which contains and controls a record of its own transaction history, and each of which is able to enter into various sorts of relationships with other agents (e.g. sending or receiving transactions, or participating in validating transactions). Unlike in standard ledger based blockchains, and drawing key structures and methods from TODA and TODA/IP (such as the TODA transaction trie and the cycle trie that gave HyperCycle its name), in HyperCycle individual agents and their purposeful interactions are the core of the system – no replicated ledgers are needed, and where they are used they are best considered efficiency-oriented augmentations to the metadata that agents retain regarding their own histories and properties.

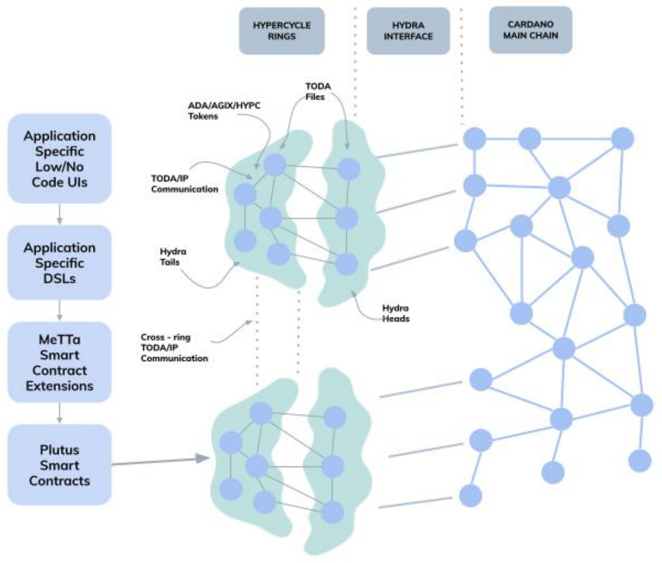

Figure 2: Conceptual illustration of two HyperCycle rings and their connection to the Any Ledger Blockchain mainchain.

Each ring in the network performs its own consensus judgments, and is connecting to pseudo-randomly selected set of nodes in the network that is triggered by time and a deterministic function on how to meet and what about. Effectively every-time a an HyPC Packet is passed from one device to the next, and while it is obvious that the requester must submit the cryptographic signature of the HyPC to the receiver node, at the time the requester will be ignored if didn't launch the token through a deterministic map from the data structure Earth64 (no single machine needs to save it or host it anywhere but everyone can compute it fairly quickly) This also means when a transaction is initiated launching the tokens through a trie makes it impossible for double spent. It is simply not possible. The imagine below shows how 2 separate temporal events happen at slight different times and size but they both must deterministically merge as they must upward navigate the BST Earth64 data structure tree forming a trie every single time. Effectively the packet is sent and as it propagates through the network with the hash of the dealing all together (irreversible)

The SingularityNET reputation system, deployed among the nodes involved in HyperCycle rings, provides a dynamically updated reputation measure to each node. Proof of Reputation means that the choice of validators within HyperCycle rings will be biased toward higher-reputation network nodes (who then get fees in HyperCycle tokens for helping with consensus decisions)... Reliance on reputation in this way helps avoid various sorts of attacks in a relatively straightforward and elegant way. To seed the reputation system we could e.g. give initial reputation boosts to AGIX or ADA stakers, and also to early purchasers of HyperCycle tokens.

Section 4 briefly reviews a few example applications ideally suited for HyperCycle – swarm AI, rating and reward in media networks, decentralized payments and computer processing, and public/private chain interoperability – and roughly indicates the sort of consensus mechanism and HyperCycle/any-mainchain interaction likely to be needed in each case.

The HyperCycle design also supports flexible public/private deployment. Given a fully implemented HyperCycle, SingularityNET applications (along with many other sorts of applications) will be deployable with subnetworks on both public and private HyperCycle rings.

1.4 Broad Usability via Smart Contract DSLs

Finally, all this underlying complexity can nevertheless be presented to application developers in a simple and usable manner, appropriate to the domain areas in which their applications are operating. Toward this end, integration of the OpenCog Hyperon [BGT21] framework's MeTTa programming language [Pot21] into Plutus is also envisioned, as a strategy for enabling broad usability via the creation of MeTTa-based DSLs focused on smart contract programming in specific vertical areas, which will often correspond to particular specialized rings. Such DSLs may straightforwardly be presented to the user as low or no code frameworks auto-generated from the underlying MeTTa/Plutus code, effectively leveraging the HyperCycle framework behind the scenes.

2 Background: TODA/IP, Proof of Reputation and the MeTTa Language

In this section we review a few key aspects of the existing blockchain networks and designs that are critical to the HyperCycle design. Then Section 3 will pull all the pieces together.

2.1 The TODA/IP ledgerless blockchain

TODA/IP [ST19] is a protocol for secure communication and coordination in decentralized networks, which relies on the same underlying data structures and encryption mechanisms as standard functioning cryptography that is running this planet including the innovations by blockchain platforms, but combines these in a radically different way to enable tremendously greater scalability.

TODA/IP significantly bypasses well known problems w/ replicated ledgers, such as

- Coordination overhead associated with distributing information from a large network to a number of replicated ledgers
- Communications overhead required for sending, receiving, and processing messages (e.g. according to gossip protocols which, while resilient, are known for their considerable communications overhead)
- Decreased performance with scale. Replicated ledgers require that every fully participating node must process every single transaction which occurs. This means that as more transactions occur, and more nodes participate, overall transaction times suffer severely. This type of protocol offers security, neutrality and censorship resistance at the cost of scalability.

Basically, in typical replicated ledger blockchain systems, every peer receives all transaction data, all hash reference values, and all block headers. Every node needs to be supplied with enough information to be able to re-create the entire chain. This is how blockchain explorers are able to provide browsing through the history of major blockchains.

Sharding palliates all these problematic factors, but only partially. To make sharding work really well requires hierarchical sharding architectures which give algorithmically more satisfactory performance but at cost of adding significant complexity and overhead. TODA/IP is in some ways similar in spirit to sharding, but keeps things simpler and lower-overhead by adopting a more fully decentralized approach, in which each local data-chunk is responsible for its own historical information (or in some cases that of its close neighbors). This changes the nature of the communication protocols a fair bit, but removes the main source of complexity and cost in ledger-based blockchains, which is the maintenance of a population of ledgers each storing broad-based information about the network as opposed to simply a population of inter-transacting agents each with their own local data and interests.

A network that is ledgerless in its foundational operation can still make use of ledgers when appropriate. For instance if one wants a rapidly searchable backup of a certain subset of network transactions, storing all these in a replicated ledger makes a lot of sense. HyperCycle is envisioned to make use of the Cardano mainchain for this purpose, along with other purposes like help maintaining certain sorts of security guarantees. In this approach, a judiciously chosen subset of HyperCycle transactions (at least within selected rings) are periodically snapshotted and pushed to Cardano mainchain where they will be stored in Cardano's replicated ledger for backup, and rapid search and access.

It is also possible to use TODA/IP together with hierarchical sharding, when one wants to achieve particular sorts of balance between security guarantees and performance. In this case the overhead of sharding is still there, but is significantly less than in cases where a sharded ledger is required to do all the work.

2.1.1 TODA/IP Transaction Processing: Decentralized and Localized

The core structural element of TODA/IP is its association of individual records with their own localized ledgers. This makes these records semi-autonomous agents in a sense that is not remotely the case for entries stored in the ledger of a traditional distributed/replicated ledger-based blockchain.

The core dynamical aspect of TODA/IP, correspondingly, is the way secure transaction processing occurs in a manner that is wholly decentralized and is also "localized" in the sense that it doesn't require access to any overall ledger of all transactions that have ever occurred in the blockchain, but requires only interaction of a small set of parties who are closer to the transaction at issue. In this dynamics, records are acting as "nodes" in a blockchain network – specifically, each record may correspond to a number of different nodes, each acting in a different portion of the tree.

Let us now briefly unpack how this works, in a simple case of a ring with a "plain vanilla" fully-replicated-ledger-less TODA/IP consensus system.

Given a record R owned by wallet A, that record can be sent to wallet B through the generation of a transaction request, which is signed by wallet A, then signed by wallet B, then distributed and signed by a set of validators. A cycle in TODA/IP consists of a round of transactions requests and ensuing validations (for those requested transactions that are valid).

But how are the validators chosen? In the simplest TODA/IP implementations, basically, any active device so long it is geographically dispersed it will get a fair probabilistic chance in the pseudo-random selection be chosen as a validator. This does not mean there is some magical proof of dispersement but instead reliance on the laws of physics and how network speed can never exceed the speed of light, yet while the selection itself gives fair chance to any node that matches the criteria of the pseudorandom function, the role they need to play requires a race amongst selected nodes and if they happen to be concentrated they null each other. More on that in the Toda/IP paper. A pseudorandom function is used to select devices in the network for participation in a computation validating a given transaction, based on their geographic dynamic disbursement. Devices cannot apply more computation than the modest amount required for the pseudorandom selection of work that is assigned to them in each block of time while the work is to run the system itself it is in parallel that the TM machines run these network of hashes to ensure the payment systems work while the AI to AI treat the entire TM as a black-box.

Because there is no way to achieve extra privileges in the network via extra work, there is no intrinsic need for "mining farms" or similar mechanisms as the work that is done is always work needed to run the entirety of the system from the TM perspective and on the other side and in parallel the work needed to run the AI computation is on a case by case. There's always a requester and a responder. Please note, those are machines requesting work and machines responding, when we say machine we mean a Hypercycle computation node that is comprised by all its elements including but not limited to, VM, AI Machine and TM. Also, while this approach enables economical incentives for devices to be on the network, it requires no disincentive for not participating in the computation. The amount of work involved here is independent of the size of the network; and the work is spread out through the system, rather than concentrated as it would be in typical replicated ledger-based blockchains.

Each validator provides four important functions:

- Confirming the validity of the transaction (structural soundness and proof correctness)
- Prevent sending a packet twice in this cycle
- Help build the consensus proofs for the transaction
- Provide matching proofs for A and B

One of the validators will also be chosen to create a new record containing the signed transaction validation message the validation process returns to the recipient. This new file will then be sent from the chosen validator to itself, resulting in the appending of this file to the validator's record.

After the transaction is done, the Merkle root of the data structure constructed as the transaction proceeds is a cryptographically secure representation of the entire transaction.

It makes logical sense for a TODA/IP-based network to charge modest transaction fees to cover the network, electricity and depreciation cost of carrying out the work involved in transaction validation. These fees would be naturally split among sender, recipient and validators, in a manner that may be different for different TODA/IP implementations. There are obvious variations such as preferentially assigning the validator role (and the modest revenue obtainable therefrom) to nodes that have provided rapid validation in the past. This leads on to the integration of TODA/IP with reputation systems that we will discuss in Section 3 below and that forms a key aspect of the HyperCycle design.

The nodes in a TODA/IP network may be divided into "rings", each of which groups a certain set of records, and which are hierarchically connected. A single record can then live in multiple concentric rings. Different rings can have different consensus mechanisms for approving transactions. Transactions made to the record in different rings can then be merged consistently within the record.

A TODA/IP system may hold a collection of rings, what we might call a "ring-set", can be configured to all share a common "TODATree" (fully saturated and balanced BST) structure, which provides a universal file address system that may roughly be thought of as a decentralized analogue to IPV6. In the context of a TODATree, each record has a unique address number that it retains over its lifetime, defined via the branch of the TODATree it occupies. A highest level TODATree of height 256 has been posited to supervene over all TODA/IP implementations, with each of the 2^{96} branches at level 160 associated with a particular TODA/IP implementation. HyperCycle will then involve a TODA/IP ring-set associated with a particular TODATree level 160 subtree.

2.1.2 TODA: Interoperable Proof Structures

While TODA/IP is a full ledgerless blockchain system, TODA simply describes a data structure.[3] The core of the TODA design is the "TODA File", which essentially is a digital data file that comes along with a per-file ledger attached as metadata. The binding between a file's internal data and its ledger allows a file to behave like a "unique digital object" (a sort of NFT) rather than a more abstract, interchangeable symbolic token.

Every transaction that a file is involved with causes a corresponding record to be attached to the file's associated ledger. These transaction records contain among other information the addresses of the other parties involved in the various transactions. The functioning of TODA is founded on the way these transaction records enable each file to have a single canonical well formed proof of provenance.

The transaction detail describing a particular transaction for a particular file specifies several aspects such as:

- the destination address: who will become the new owner
- a metadata hash, which allows users to attach arbitrary metadata to the transaction
- an encumbrance, which is a special restriction on how the transaction will occur

[3]This section is based on [GGT19]; later designs supersede this but are generally compatible with it.

- a signature on this data, to certify that the current owner really wishes the transaction to occur

The transaction detail becomes an intrinsic part of the proof of a file. By only requiring the hash of the metadata to be included in a transaction, the size of proofs is kept under control, and users are given the power to choose not to share the metadata of a transaction with someone else.

These qualities provide TODA files with the ability to move transparently between rings, regardless of the consensus mechanisms or networks involved, provided only that the capacity to create corresponding structures exists. This allows records that make use of these data structures to interoperate natively across unrelated systems.

2.1.3 Key TODA Data Structures and the TODA Cycle

Operation on a TODA file depend centrally on two Merkle trie data structures: The File Trie and the Cycle Trie. Explicating how these tries work is the best way to go to the next level of detail in the description of TODA operations.

The file trie is a Merkle trie which associates file IDs with transaction detail hashes for transactions involving that file. A transaction proof is the Merkle proof associating a single file detail with a file ID. The transaction details associated with a file's ID by its owner's file trie are the only valid representation of operations on that file.

The file trie for a given address in a given cycle contains all of the operations performed over files owned by that address in that cycle. Files owned by that address that are not present in that file trie for that cycle are said to have null proofs. Because the transaction details referenced in the transaction proofs associated with a file are the only valid representation of operations on a file, a file with a null proof is demonstrably not operated on within the context of the file trie.

Each file trie is contained within a cycle trie. The cycle trie can be built in a highly distributed way using e.g. TODA/IP, as we describe below, but it can also be built in a fully centralized setting, or something in between. The values contained in this cycle trie are nothing other than the file trie Merkle roots corresponding to the transactions occurring in the cycle. Each file only requires the small slice of the overall cycle trie that contains it.

As the cycle proceeds, the Merkle trie is built containing proofs for each transaction. For a given transaction its two proofs (one from the sender, one

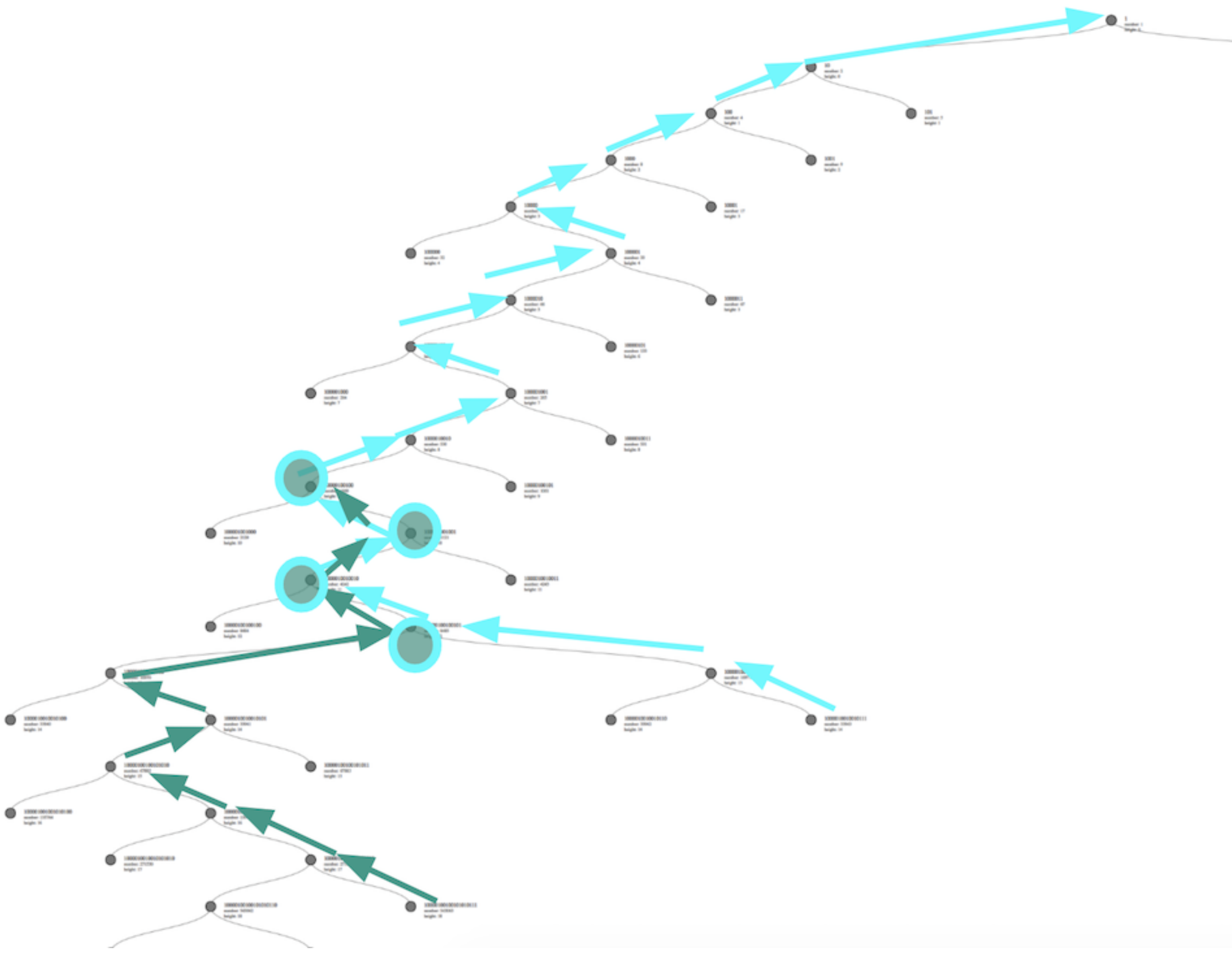

Figure 3: Tries Merging

from the receiver) will be known only to the sender and the receiver, but all are part of the cycle trie.

Not all nodes in a ring, in general, will be available to participate in a given cycle. However, cycles in any ring that a file belongs to are relevant to that file whether the node corresponding to that file is transacting in that cycle or not. In order to be able to continually prove that your file's ledgers are complete and not missing any information, you need to collect a bit of data every cycle that contains proof that your address did not contribute to the cycle. Luckily, these trimmed proofs tend to be especially short and don't have to wait for each other, nor wait for block, they can all run in parallel and those that arrive at a certain time to join and concatenates with others keep going in a one way computation to reduce every pair to one following their deterministic Earth64 static map (Figure 3)

A protocol-level way to deal with these zero-transaction proofs is to say that the nodes assembling the cycle trie during a given cycle are responsible for maintaining proofs of their non-participating neighboring nodes (where a "neighbor" is a node that a given node has transacted with before). As many neighboring nodes who didn't contribute to a cycle will generally share the same proof, this is a relatively trivial set of data to maintain. There are also other routes to managing these proofs which are more at the service level, i.e. specific rings may maintain specialized ledgers or other mechanisms for managing these proofs that in some contexts may provide greater efficiency than relying on neighboring nodes. For instance, with sufficient trickery it may be possible to store "proofs of null proofs" together with cycle roots without actually storing whole cycle tries; there is some research yet to be done in this direction.

2.2 Proof of Reputation

4

The final key ingredient woven into the HyperCycle design is the "Proof of Reputation" consensus mechanism that was incubated within SingularityNET by a team led by Anton Kolonin and fleshed out fully in collaboration with Oladotun Aluko [AK21].

SingularityNET's "weighted liquid rank" reputation system

[KGDI18] is a general mechanism for assigning reputation to members of

[4] This section is based on [AK21]

a decentralized network, integrating implicit and explicit ratings, and incorporating various subtleties such as adjusting the effect of member A's impact on member B's reputation based on member A's reputation. The length of a network member's history, the amount of currency they have spent, the predictiveness of their own ratings, and many other factors can play into reputation calculations. The framework also includes a role for machine learning driven "reputation integrity analysis" agents to identify potential reputation fraud. Several simulations of the use of weighted liquid rank in various situations have been implemented and evaluated, e.g. reputation in ecommerce marketplaces [KGP+19].

The core idea of Proof of Reputation (PoR) is to use weighted liquid rank based reputations as the basis of a consensus mechanism for blockchain networks. The PoR framework uses the interaction of nodes in a network over time to determine the amount of reputation associated with each node in the network. A node's reputation is calculated by blending together a normalized set of ratings and the corresponding reputation values of the node providing the rating at a given point in time, rather than simply the value of the direct rating given by other nodes. A node's behavior also directly influences its overall reputation value by appropriate formulae. The PoR mechanism then uses the reputation to determine a set of consensus nodes responsible for maintaining the network's shared state. Reputation values are ongoingly updated as interactions between nodes in the network progress over time.

A few of the core principles underlying this scheme are:

- . The reputation value computed for a node is based on the reputation value of the node providing the rating – this is the "liquid" aspect
- The temporal scoping of reputation, according to which reputation values accumulated by members in the longer past contribute less to the current reputation value (while at the same time allowing agent history to contribute appropriately to reputation)
- The openness of all reputation values to all members of the community so that audits can be performed.

In accordance with principles of openness and decentralization, PoR stores node reputation values on a sidechain rather than in a centralized database. In a HyperCycle context, this ends up meaning that each HyperCycle network spawns a side HyperCycle network managing the reputation values for the parent HyperCycle network. This doesn't spawn an endless recursion because the nodes in the child network (containing reputation values) have the same owners as nodes in the parent network, so the consensus mechanism in the child mirrors that in the parent.

At the start of every PoR consensus round, consensus group members need to be selected and added into a consensus group. Members of the consensus group are chosen from the nodes with the highest reputation values, e.g. with collective reputation scores that exceed 50% of the total reputation values of the network. A leader from the group is then selected, who serves the functions of:

- Packaging all valid transactions from the list of pending transactions to a block
- Calculating the new reputation values for all network nodes using data from transactions in the transaction list
- Broadcasting the commit message to the consensus group

To verify a transaction, the consensus group waits until a certain minimum number of of consensus members has sent a confirmatory message. This process constitutes a consensus group vote, in which each node has a weighted vote proportional to its reputation value from the previous round.

This general mechanism is consistent with a variety of different underlying blockchain architectures, e.g. it could be used with account-based, UTxO or EUTxO frameworks. HyperCycle layers PoR based consensus into a system based on EUTxO and TODA, resulting in a non-replicated-ledger-dependent blockchain framework supporting powerful functional-programming smart contracts but incorporating the particular security that PoR brings against various sorts of attacks.

2.3 OpenCog Hyperon's MeTTa as a Smart Contract Language

The MeTTa (Meta Type Talk) language [Pot21] developed in the context of the OpenCog Hyperon AGI project [BGT21] has a number of properties that make it favorable for use as the core smart contract language within HyperCycle. The very flexible nature of MeTTa's semantics (e.g. gradual dependent typing, and a homotopy type theory oriented treatment of equality) means that it does not impose unnecessary or arbitrary commitments on handling of TODA/IP messaging or TODA data structures; MeTTa programs can be as open-ended as they need to be for dealing with a variety of HyperCycle ring structures.

Further, MeTTa is ideally suited for the development of application-specific smart-contract DSLs, after the rough pattern of the Idris language's dependent-type-based DSL creation facility. Toward this end we envision making MeTTa available as the default, so that e.g. finance-focused, medicine-focused or evolutionary-learning-focused smart contract languages for on-chain execution could be scripted in MeTTa and then run via Plutus. It will then be possible to create generic tools for mapping MeTTa DSLs into appropriate low or no code development UIs, resulting in a HyperCycle framework with a high degree of usability for developers working in niches for which DSLs have been created.

A compiler is being created that translates MeTTa source into source-code in the rholang language originally developed for use in the Rchain blockchain [**?**]. This allows MeTTa to leverage rholang's powerful concurrency properties, such as linear or near-linear scaling in use of multiprocessors (including on GPUs) for many algorithms. It also allows MeTTa smart contracts to leverage tokenomics internally to manage computational resource allocation. This fits naturally with the AI-DSL developed for use between SingularityNET agents, which allows AI processes to specify their functional, compute-resource and other properties using dependent type expressions. It provides a framework in which smart contracts running on HyperCycle nodes can automatically manage their resources and automatically outsource work to each other and collaborate to solve problems, in a manner that integrates tokenomics richly into the functional-programming and process-calculus infrastructure making the multi-agent system work.

HyperCycle is designed to allow smart contracts in a wide variety of languages. Composition of contracts, infusion of tokenomics into the flow of contract execution and various other features will not work as efficiently or as sophisticatedly in other languages as in MeTTa. However, these features are not always critical, and there is a large advantage to being able to leverage smart contracts already written for use on other existing blockchains, and interoperate these with each other as well as with MeTTa scripts. The tightness of the interoperation with smart contracts in other languages will have some case-to-case aspects; it is clear for instance that the functional nature of Haskell will enable a quite tight integration between MeTTa contracts and Cardano's Plutus contracts.

3 HyperCycle

HyperCycle is a novel blockchain architecture which is formed by assembling key pieces from the existing blockchain networks and designs reviewed above: TODA/IP and other aspects of the TODA framework, reputation for consensus and other aspects of system regulation, and MeTTa for smart contracts. The "assembly" process involved is not entirely straightforward, however, and requires some modifications and extensions to all of the ingredients. The result of this process is a blockchain with unprecedented capability for handling high-speed, large-scale on-chain agent interactions such as is required for on-chain deployment of population-based AI algorithms, tokenomics-powered interactive media, and numerous other applications.

3.1 Core HyperCycle Architecture Concepts

The HyperCycle network is a population of "agents" which correspond to HyperCycle computation nodes exchanging TODA files through the network packet with zero third party dependency and therefore able to execute transactions in a p2p setting in a sub-second finality. Each agent has one single owner in every block of time, and one owner may control multiple agents. The agents are organized into TODA-style rings, and a given TODA file may correspond to multiple agents living within different rings. As usual with TODA, each ring may have its own particular consensus mechanism.

There are no fees per se for HyperCycle transactions, however the network dynamics can incorporate *royalties* for ever successful AI computation. Having said that the HyPC token has value and can be the mean of exchange it also has the capability to carry values inside it such as ETH, USD, ADA, AGIX and others.

Each HyperCycle ring has its own independent operation, although some rings may rely on external blockchain for some functions, e.g. long-term "external memory" style backup. In this case, a ring would have its own policy determining which of its internal transactions gets pushed back in snapshots to for example BTC, ETH, ADA or others mainchain, and how often these snapshots are pushed back.

Agent reputations and agent-owner reputations are calculated dynamically via a weighted liquid rank reputation system that operates at the overall HyperCycle network level. Three dimensions of reputation are maintained for each agent: Reliability, cooperativeness and honesty. An agent with poor uptime or slow processing may be rated unreliable even if it has never done anything fraudulent or problematic. An agent with strong technical properties may still refuse to participate in constructing proofs, which will merit it a low cooperativeness rating, but this is still important to distinguish from a fraudster agent which will get a low honesty rating.

Use of reputation and social connectivity dynamics should in many cases enable dynamic evolution of HyperCycle rings, and in rings that use hierarchical sharding (to be discussed below), potentially of shards as well.

Each ring, in each cycle of its operation, has certain cycle head parties and certain cycle tail parties, in the rough sense of the Hydra protocol. Agents in the ring with high reliability, cooperativeness and honesty may be thought of as essentially the same as conventional Hydra head parties. Any member of the ring who wants to participate actively in consensus during the next cycle can broadcast a message saying so, and will then be considered as a cycle head for the next cycle.

In HyperCycle, things will proceed more like in TODA/IP, but with added element to expedite transactions and that's the ratio a node has which incorporates proof of computation, uptime and reputation-based weighting. While each TM (Traansaction Machine) node namely the Toda/IP component is identical to other nodes Validators for each transaction proposed within a cycle will be chosen via a deterministic/pseudorandom function from the network with sifficient probable redundancy while the work required is sufficient to be provided by any single node. In fact it's a race of who gets there firs with the right proof and that's within each cycle and many cycles are chained within each block of time. While a block of time is being concluded, many cycles might be half way through their chained deterministic pathway in the computation necessary to achieve the proof required. s for the selection of validators, it has nothing to do with the reputation, however the selection of many AI providers,, the preference is naturally geared to the agents with stronger reputations.

3.2 HyperCycle Consensus Mechanisms

3.2.1 Lightweight Rings

The simplest sort of HyperCycle ring is what we call a "lightweight ring," which is wholly ledgerless in the manner of the "simple TODA/IP ring" roughly described in Section 2.1 above.

The "lightweight ring" consensus mechanism can be adequate for rings involving individual transactions of huge financial value, but is also well suited for rings oriented toward micro-transactions. The key is to have a tiny transaction in the least amount of time so we can have many transactions as each transaction can lead to emergence of intelligence, the more the system is efficient the more emergence is gained. Examples would be:

- AI agents that are participating in population-based AI algorithms like genetic algorithms, genetic programming, ensemble machine learning, swarm AI
- Small-scale tokenomic transactions such as rating articles or comments on a tokenomics-enabled media site
- Small-scale financial micropayments for any purpose

3.2.2 Hierarchical Sharding Enhanced Consensus Mechanisms

For cases where a greater level of protocol-level security and reliability is required, rings with alternative consensus mechanisms are needed. For instance one may create a "hierarchical sharding enhanced ring" (HS enhanced ring), in which the basic localized transactions between agents are supplemented by a hierarchically sharded ledger which contains proofs corresponding to the ring.

The HS ledger may be used in various different ways, yielding multiple variations of the HS consensus mechanism. One option is to use the HS ledger purely as a backup, in case the more fully decentralized peer-to-peer approach used in the lightweight consensus mechanism fails. This allows guarantees beyond what is possible with the lightweight consensus, but in the case where the agents in the ring are mostly effective and cooperative, it only modestly increases operational cost and speed.

Another option is to use the HS ledger more aggressively. E.g. if zero-transaction proofs are stored in the HS ledger then there's an option to omit them from file metadata, thus keeping proofs smaller. However this has risk to significantly increase the time cost of doing proofs because of the need for agents to download relevant portions of the HS ledger every time they do proofs. The severity of this issue will depends on the habitual distributions of the inter-agent transactions in the network – in particular, on the degree to which the transactions among the parties in the ring fall into a hierarchical statistical pattern that can be reflected in the hierarchical sharding structure.

Broadly speaking, the use of the HS ledger provides greater security against various sorts of attacks, and thus an HS ring is going to be appropriate for agents carrying out larger, less micro-level/disposable transactions than those appropriate for a lightweight ring. However this added security will come at an added performance cost – which can be reduced via clever design and efficient implementation, but will often remain a significant factor.

One general principle we see in these aspects of the HS consensus mechanism is that the TODA design providing core elements to HyperCycle does not obviate the key tradeoffs underlying all blockchain design, such as the performance versus security tradeoff – but does provide a framework in which various different tradeoff choices appropriate for different applications can be elegantly inserted, via use of ring-specific consensus mechanisms associated with their own auxiliary data structures and dynamics building on the core HyperCycle mechanisms.

3.2.3 Further Notes on Security, Reliability and Performance Tradeoffs

In this section we dig a little more deeply into the technical tradeoffs involved in the choice between the lightweight and HS ring consensus mechanisms described above. The discussion may also shed a bit of a broader light on some of the tradeoffs involved in choosing a lightweight ring versus a heavier ring for a particular application.

Incentivizing Agent Participation in Consensus Firstly, one key question underlying the operation of a ring under the lightweight consensus mechanism is: What incentivizes agents to contribute the knowledge they possess, which is necessary for other agents' in the proof construction process they need to undertake to construct the Cycle Trie? In some cases this is not a relevant question, e.g. if all the agents in the ring are strongly externally incentivized to contribute to the overall outcome of the process the ring is carrying out (e.g. AI agents collaborating to solve a problem of mutual interest to all the agent owners). But in other cases the rapid cooperation of agents in providing distributed data to feed proofs to form the cycle trie could be a potential critical bottleneck for lightweight consensus operation.

The basic answers to this question are: payment and reputation. The network design involves modest transaction fees and some of these can be directed to agents providing necessary proof data for cycle trie formation. Also, maintaining acceptable ongoing reputation will be necessary for agent-owners to continue participating in HyperCycle networks. While low-reputation agent-owners may always try to re-enter the network using a different identity, there will be machine learning driven "reputation police" mechanisms designed to spot this sort of behavior. Further, agents with higher reputation are more likely to be chosen to participate in consensus and thus are more likely to profit from their participation in the network ongoingly. One can also juice up the benefits of having high reputation even further by making transaction fees lower for high reputation agents, so that agents have a direct financial incentive for playing nice.

Another possible mechanism, to be introduced with great care and only in particularly appropriate sorts of applications, is to allow sufficiently high reputation agents to add transactions to the cycle trie even if they don't have all the necessary proof data at hand. This is not a route one would want to take in a network hosting large financial transactions, but in a ring comprising AI agents acting cooperatively to solve problems together on-chain, it may be perfectly acceptable.

Payment and reputation are "soft incentives" rather than hard protocol guarantees, so in the lightweight consensus scenario there is still a possibility that some agents will simply refuse to provide data that they hold locally, and eat the reputation loss. In this case, an external conflict resolution mechanism will need be invoked, e.g. if agent-owners involved have made deposits (either within HyperCycle or in some other associated chain) they will lose all or part of the deposits they have made. But even so, there's a possibility that some transactions will not be able to happen.

Costs and Benefits of HS Enhanced Consensus The HS consensus mechanism sketched above is, in a sense, a hybrid between the fully ledgerless TODA/IP system and a more traditional ledger-based blockchain with hierarchical sharding (see e.g. [CDD+19] for a fleshed-out description of hierarchical sharding in the Thinkey blockchain, but many other examples exist). In this case we have information which, ideally, everybody in a ring should have (with consensus): the cycle roots of the TODA/IP system. In the HS approach we increase the odds of achieving this ideal by increasing this common information a bit beyond what's there in the lightweight consensus mechanism. All head parties in a ring still construct the data structures for the record. But along with building the cycle tree, the non-zero transaction packets are added to the hierarchically sharded ledger. This system has almost the same level of privacy as in the lightweight consensus protocol (from common information it will be impossible to infer who owns items). But it automatically solves the two requirements mentioned above, and dramatically reduces the proof sizes (since proofs for zero transactions will not be required in proofs of provenance).

The big downside of the HS approach is that, in order to guarantee its ability the needed proof for any file received, an agent needs to be able to download the relevant portion of the HS ledger. In the worst case it needs to download the whole ledger, which could become huge as the system grows. On the other hand, if the transactions in the network have a strong tendency to be localized, so that most transactions occur between agents in the same small shard, then this may be only a moderate-sized problem. Still, at least in unfavorable scenarios, the HS consensus mechanism loses the lightweight consensus mechanism's favorable property of accelerating rather than decelerating as the number of network participants increases.

Two Strong (But Optional) Requirements One way to think about the value added by the HS approach (or other similar mechanisms) is to look at the following two requirements, both of which are clearly desirable for maximal network security:

1. If a user adds a non zero transaction then this transaction can make its way to the final ?cycle root? (cycle root for which we have consensus) *if and only if* user has merkle proof for this transaction. It means that it is not possible by design to fail (because of network error or intentionally) to send the element of merkle proof to the user.

2. Users should always be able to reliably recover all cycle roots and all merkel proof for all zero transactions.

In a "low security margin" ring, such as one using the lightweight consensus mechanism outlined above, we can have nonzero probability of failure for these two properties. However, when failed it can be completed in subsequent block. Failure here does not mean loss. On the other hand, it is difficult to see how one would satisfy the second requirement in particular without doing something roughly "decentralized ledger like" – i.e. without having decentralized consensus about proofs of zero transactions, and making these available for everybody. The first requirement is in some way trickier but is also less complex to address in ledger-like scenarios.

However, it is not a given that these requirements must necessarily be met in all rings via guarantees at the protocol level. Maximizing security guarantees generally also maximizes cost (computational and otherwise), which limits the scope of potential use cases. Bitcoin is currently primarily concerned with the use-case of transacting large quantities of speculative assets, and for this purpose a decentralized ledger based approach is relatively apropos. But other sorts of use-case, e.g. in agent-based AI or social media or micropayments, have dramatically different sets of requirements which point toward different trade-off choices, and some of these choices may sensibly involve giving up some protocol-level security guarantees in pursuit of performance or other beneficial aspects.

Defusing the Power of Forking It's also worth noting that essentially every TODA/IP-based system will possess, by design, certain sorts of security that elude all modern decentralized ledger-based systems – for instance, security against the chaos and economic perversity habitually incurred by blockchain forks. All too often the fallback solution to a major security issue in a ledger-based blockchain is "we can always fork the chain." This has worked disturbingly well for various cryptocurrencies on a financial basis. For instance, the results of the Bitcoin forks leading to Bitcoin Cash and Bitcoin SV, and the Ethereum fork leading to Ethereum Classic, were broadly profitable: Both pre and post fork tokens retained significant financial value, so that the forks increased the market caps of the underlying blockchains. However, in other cases the ability to fork in this way is highly undesirable; for instance real-world assets like loyalty points, fiat, video game assets, etc. The fundamental uniqueness of TODA/IP records obviates the power of this sort of forking. In a HyperCycle network with HS consensus, forking the HS ledger still doesn't fork the records underlying the HyperCycle agents. There can of course be software upgrades but there is no structural bias for these to lead to perverse forking situations as seem to routinely occur with today's standard distributed ledger-based blockchains.

File Uniqueness as a Key to Understanding Tradeoffs HyperCycle inherits from TODA the centrally important invariant that *each file must have a single canonical well formed proof of provenance*. The key questions, which we have been discussing in this section as regards lightweight versus HS consensus mechanisms are ones like: How quickly can that proof be obtained? How complex is obtaining it? How much of a security margin is there for my files? What recourse is there if something blows up (either through lack of availability or through widespread equivocation)?

The answer to each of those questions is ultimately "it depends", because, as with TODA, there is a goal of having HyperCycle workable in a variety of different use-cases with different practical requirements, while maintaining canonical uniqueness for the underlying files. This requires flexibility because the security margin you want for high value assets always costs a fair bit in terms of finance and/or performance, and continues to cost over time (nothing is free), whereas for efficient micro-transactions you have to accept significantly lower security margins to get the efficiency required. What we can ask for is not a single detailed blockchain design that addresses all possible requirements with the exact same set of algorithms, but rather a flexibly modularized blockchain design that can be elegantly customized to meet a broad set of requirements – for instance via diverse pluggable consensus mechanisms, as in the case of HyperCycle.

File Uniqueness Across Multiple Rings These tradeoff-related issues arise in particularly subtle ways when considering the dynamics of TODA/IP-based systems across multiple rings. The uniqueness of a file in a multi-ring network is guaranteed only if we have consensus about cycle roots, in an appropriately defined sense. In the simplest case, if there is agreement about all cycle roots across all rings in the network, we can pass files between rings unproblematically. However, this strong situation is not the only one of interest. For instance, if I have a file which was in another ring A for some period of time, then to appropriately carry out proofs regarding A I require cycle roots from ring A for this period of time, but not necessarily beyond this period of time. The proof data from A over the relevant period of time is included in the file's proof of provenance, and serves to link the cycle roots of A into the greater network of rings.

3.3 Smarter Smart Contracts

HyperCycle's approach to smart contracts is both subtle and inclusive, involving a unique native smart contract framework including a python VM, MeTTa (and its compilation to rholang), along with an efficient smart contract execution framework that works for many different smart contract languages, and a framework for allowing interoperation of smart contracts written in different languages and potentially interacting with different third-party blockchains including EVM but it will not be only EVM.

3.3.1 Accelerated Smart Contract Execution

Handling of smart contract execution in HyperCycle is a deep topic that we will touch only lightly here. Rather than relying on generic consensus mechanisms, greater efficiency can be achieved by use of specialized mechanisms for validating smart contract output in a secure and decentralized way. In this manner we can achieve multiple orders of magnitude speedup over existing blockchains.

In the standard Ethereum-blockchain approach, each smart contract is run on every full node in the network, thus providing consensus on what the output of the contract should be according to the network's consensus algorithm. This is secure (or at least as secure as the network's consensus algorithm) but obvious extremely inefficient. If 2000 nodes need to approve to give consensus on a smart contract's output, then the smart contract is basically running at least 2000x slower than if it were just a program running on a single machine.

A few workarounds for this smart contract execution bottleneck have been proposed, most famously zero-knowledge rollups, which are part of the Ethereum roadmap, and are being pursued for Cardano by a number of parties. However the simplest (not that simple) form of zk-rollup based system relies on a small number of centralized provers, and decentralized alternatives become increasingly complex relying on special marketplaces of nodes providing zk proofs.

The HyperCycle framework supports considerably simpler and lower-overhead methods of radically accelerating smart contract execution. Basically, a deterministic yet pseudorandom choice and a few other clever methods are used to enable each smart contract to be run only on a small set of nodes rather than the whole array. This kind of flexible mechanism becomes especially convenient in a system that has no globally shared or replicated ledger and correspondingly no universal list of nodes who would be called on to validate all smart contracts.

In the HyperCycle approach, when one node has a smart contract it wants to run, it uses a specially designed pseudorandom function to select a small number (say N=3) of other nodes to run the smart contract as well. In the simplest case of a smart contract that just gives a single output and doesn't report interim results: These other N nodes each run the contract and report an encrypted version of the result of the execution ... if the results from the N nodes are all proved equivalent then the result is decrypted and delivered to the original requestor node, and payment is delivered to the N nodes.

In the case of a smart contract that produces interim results, and does potentially expensive processing in-between the results, then the process of comparison among the N nodes doing execution may be done at appropriate check-points during smart contract execution, as regards interim results. If one is working with a pre-existing smart contract interpreter (such as the Plutus interpreter) then enabling this variation will generally require some minor tweaks to interpreter.

This approach can work across various different VMs, and can handle smart contracts running on VM A which then spawn smart contracts running on VM B, and more complex scenarios.

For most smart contracts, whose output is drawn from a reasonably large space of possibilities, a choice like N =3 will reduce the odds that all N executor nodes fallaciously return the same result very very close to zero.

Roughly speaking the result of this approach is that to run a smart contract in a provably almost-surely-correct way only takes say $3x$ (plus some not-that-huge overhead) as much compute effort as running comparable code on just one machine. Which is a very modest slowdown compared to running smart contracts on existing ledger based blockchain networks (orders of magnitude less). While maintinin the same security guarantees and arguably more.

In cases where aren't too many operating devices able to run the smart contract, or where the smart contract is being validated only within a relatively small ring of participants, and therefore guessing who's being selected becomes easier by attackers, additional obfuscation mechanisms may be valuable. To provide extra defense against collusion by the pseudorandomly chosen nodes, one can transform the functions (contracts) being sent to each node so that each node is executing a slightly different computation, and the nodes then have no tractable way to connect their assignments with those of other nodes. This additional complexity, however, is not necessary if one is dealing with small, quickly-executed smart contracts that can easily be run on any pseudorandomly selected machine in a large ring of HyperCycle nodes.

3.3.2 Smart Contract DSLs for HyperCycle: A Path to Broad Usability

While all efforts will still be to incorporate EVM for compatibility reasons, python and the MeTTa language's semantics are highly advantageous for smart contracts involving AI operations, as many of these can be concisely expressed using MeTTa's abstract type constructs. MeTTa's compilation to rholang provides a highly concurrency-friendly and tokenomics-infused implementation for MeTTa contracts. MeTTa's interoperablity with other smart contract languages from other chains will allow practical applications to be glued together fairly quickly in many cases. However, equally important for the future of HyperCycle are MeTTa's facilities for creating DSLs.

Smart contract coding can be a subtle and sensitive matter, and it's preferable to have a framework in which most application developers don't need to think much about the nitty-gritty of smart contracts and can focus on their application. In the Ethereum world, the closest thing that has emerged to usability so far is a "cut and paste culture" in which most developers work by copying others' Solidity code and tweaking it to suit their needs. However, this is not how things work in a software ecosystem powered by tools with reasonable composability. MeTTa provides powerful composability but has the disadvantage of being relatively opaque to most developers without background in functional programming, process calculus or related mathematics.

An alternate approach to achieving usable smart contract development is the creation of domain specific languages presenting developers with only those blockchain-related functions they really need in their domain of application. Cardano's Marlowe DSL-for-decentralized-finance pioneered this approach [SNST20], but also highlights some relevant caveats. While Marlowe is extremely elegant it seems not to provide out-of-the-box the most convenient approach for achieving scalability in modern DeFi applications.

So, for example, a Cardano DeFi application like SundaeSwap ended up directly leveraging Plutus (which is generic) rather than Marlowe (which is finance-specific but not tailored for the particular sort of financial operations at the heart of SundaeSwap) [Sun]. One thing this illustrates is the need for a framework enabling rapid updating and modification of DSLs within an extremely flexible base language. This course was not open for Marlowe when it was initially created as its launch preceded that of Plutus, but it is important to consider in a HyperCycle context.

MeTTa provides an elegant approach for the creation of application and algorithm specific DSLs, similar to the use of Idris for DSL creation [Bra13] [Bra20] but with greater flexibility. The simple and generic nature of the process of DSL creation in MeTTa opens the door for tools that auto-generate low-code or no-code UI frameworks for MeTTa-specified DSLs. This arguably is the most (and perhaps only) viable approach for making smart contract development adequately simple and usable for the average product developer without sacrificing security, robustness or capability.

Depending on the DSL, in many cases it may be possible to create low-code or no-code frameworks that implicitly conduct authoring in the DSL. The abstract nature of MeTTa lends itself well to the templatization of this sort of process, e.g. to the creation of general-purpose tools for mapping sufficiently simple application-specific DSLs into corresponding low/no-code frameworks. In this approach the only people who need to deal with the technicalities of MeTTa are those who are writing DSLs for new domains, or creating custom applications sufficiently recondite that they don't correspond to any of the DSLs in the library.

4 A Few Promising Early Applications

We now give a few brief notes about a few key HyperCycle application areas that we are eager to explore once a sufficiently mature version of HyperCycle is available. We note this is somewhat of a haphazard sampling of the possibility space, reflecting our current landscape of pursuits which may omit projects and verticals that will be at the top of our minds by the time HyperCycle is ready for scalable usage. Also, of course, much of the beauty of launching an extremely flexible and general purpose platform is that others can then utilize it in ways and domains one never conceived.

4.1 Swarm AI: Evolutionary Learning, Algorithmic Chemistry, Ensemble ML

One major use-case driving the design of HyperCycle is the desire to run population-based AI methods on-chain – for instance, run a genetic algorithm where each population member is an on-chain agent and crossover and mutation are on-chain transactions. ”Algorithmic chemistry” applications as explored in e.g. [Goe16] [Bul20]present similar issues and opportunities.

There is also great potential in the area of decentralized ensemble machine learning – building model ensembles confronting a common data-analysis or reinforcement learning problem, in a setting where each model in the ensemble may be owned and managed by a different party. To carry this out in a transparent and fully decentralized way, one would like as much as possible of the process of passing around datasets, assessing and merging results, running tests and so forth to be done on-chain.

These are applications where for almost all problems a lightweight consensus approach will be preferable – maximal real-time security is not key. If an attack or failure causes some interim AI results to be lost, this is unfortunate but rarely tragic. DSLs here will serve as AI algorithm configuration languages enabling developers to quickly experiment with different approaches on different datasets in different applications.

4.2 Ratings and Rewards in Online Networks

Managing ratings and rewards in online networks is another case where lightweight consensus is desirable. In this case it is important that attacks or system failures not cause participants’ ratings or rewards to get permanently lost, but it’s OK if making participants whole after infrequent unfortunate incidents takes some time and depends on a slow-paced conflict resolution mechanism. Having maximally strong real-time security guarantees is less important than having a system with very low cost of operation, and which is quick and effective under all normal circumstances. It is also easy to envision a low or no code DSL enabling developers to quickly specify the customization of the weighted liquid rank reputation framework and various related reward tokens in their applications.

4.3 Decentralizing Payments and Processing Power

Two somewhat subtler use-cases are decentralized payments processing, and the tokenized decentralized management of processing power done in the NuNet platform [5].

In each of these cases, one has a spectrum of transactions, some of which are small in magnitude (micropayments on an online media platform, utilization of small amounts of background processing on an individual's smartphone) and naturally match with a lightweight consensus protocol, others of which are more serious (larger payments, utilization of large chunks of processing such as supercomputer time or significant portions of a corporate computing network) and would merit the added protocol-level security guarantees of something like HS enhanced consensus.

These applications would appear to benefit from multiple rings with different consensus mechanisms, and the ability for some agents to participate in transactions in both rings. So for instance if I pay a few US cents equivalent to a blogger based on the time I spent reading their article, this goes through a lightweight consensus based ring; but if I pay a larger chunk of funds to an AI server farm to train a big ML model, this should go through an HS enhanced ring.

DSL-wise, the Marlowe framework would seem to have many positive lessons to teach here, and one envisions a Marlowe-like language embedded in Plutus via MeTTa leveraging unique HyperCycle transactional features based on TODA and reputation.

4.4 Adaptively Blending Public and Private Chains

TODAQ, one of the companies in the TODA ecosystem, has pioneered the use of centralized TODA rings for enterprise applications. A private TODA ring of this nature can interoperate naturally with public TODA rings running on decentralized consensus mechanisms like TODA/IP, due to the elastic nature of the TODA design in which the sovereignty of the individual data file is key.

A similar approach can be taken for enterprise deployments of HyperCycle. In fact there are multiple approaches to be pursued for different enterprise contexts.

[5] `http://nunet.io`

One strategy is to develop a HyperCycle ring with outright centralized control and data management: Basically going beyond the HS enhanced consensus and just using a centralized database as a HyperCycle ledger, with only as much replication as is needed for practical efficiency (no need to use replication to assure decentralized control in this case).

Another strategy, opposite in some ways, is to recognize that within the bounds of a particular enterprise, it may be that lightweight consensus mechanisms can safely be used, because it's safe to assume that all the agents operating in the HyperCycle ring are going to be honest and cooperative with respect to each other. This approach can enable greater efficiency within the walled garden of an enterprise than can ever be possible in the "wild west" of a public blockchain.

The choice between these two strategies depends on the degree of centralization and internal trust in the particular enterprise in question. The HyperCycle infrastructure supports a variety of custom possibilities beyond the two specific extreme strategies indicated here. Broadly, the point to be made in this regard is that the flexibility to define rings with custom consensus mechanisms, and then transact agents and files across these rings, allows hybrid public/private blockchain deployments to be created to meet the needs of essentially any enterprise. After an initial batch of enterprise deployments has been successfully executed, as a side-effect set of template "enterprise consensus mechanisms" will have been created, which will handle the vast majority of enterprise situations.

5 Conclusion: Catalyzing the Future Of AI and Blockchain Ecosystem

Blockchain technologies are still in their early days, so it would be a mistake to assume that the technologies that have become popular today represent mature solutions that compellingly solve the problems they address. Some leading blockchain frameworks do embody algorithms, structures and processes that appear solid enough to persist into the decentralized networks of the future. However, there are also major aspects of current blockchains, such as the default use of distributed replicated ledgers and the reliance on crude consensus mechanisms like Proof of Stake, that appear to us to owe their current dominance largely to historical accident rather than fundamental superiority or appropriateness.

We have articulated here a novel blockchain design, HyperCycle, which we believe has the power and the flexible customizability to serve as the basis for the decentralized networks of the future, at least in the particular arena of large-scale systems of software agents providing microservices to themselves and external consumers.

We refer to HyperCycle as "multilayer", "layer-fluid" or "Layer 0++" because it has the flexibility to service the entire ecosystem of microservices (focusing on though not restricted to AI) via adopting whatever role is appropriate in a given context: low-level secure communication protocol, base blockchain, sidechain, etc.. The goal here is to present the highest security at the lowest possible network level, thus minimizing frictional cost and enabling unprecedented capabilities. Via leveraging cutting-edge cryptography together with essential ideas, structures and processes from the TODA/IP blockchain and the Proof of Reputation and OpenCog Hyperon frameworks, we believe a radically different future for blockchain technology can be created with relatively modest (though still far from trivial) engineering effort.

Our most immediate use-cases for HyperCycle are the SingularityNET decentralized AI network, and the various spinoff projects emergent from the SingularityNET ecosystem (e.g. NuNet for decentralized processing power, Mindplex for decentralized media, Rejuve for decentralized medical data and research, SophiaVerse for decentralized metaverses, SingularityDAO for DeFi, etc.). However, HyperCycle's scope extends far beyond these particular application and we foresee a very broad range of utilization once HyperCycle's engineering and deployment are complete. Indeed we believe HyperCycle technology has the potential to play a key role in projecting blockchain beyond the niches it currently serves and into the dominant role in the global tech and financial ecosystem that various pundits have long foreseen.

Appendix: Notes on Deep Cardano Integration [6]

HyperCycle involves a unique combination of mechanisms enabling efficient, lightweight decentralized coordination of software processes; at its foundation it is not dependent nor piggybacking on other blockchains that it interacts

[6] This section largely contains information drawn from [Fou21] and [CCF+20]

with, as it already has an extremely solid foundation for all aspects of secure, decentralized software coordination. However, integrations can provide certain functionality that may very well be the most effective way to make various applications work at a particular point in time, giving available tooling and user bases.

HyperCycle is capable of robust cross-chain interaction, and in particular it's possible to run a variety of virtual machines within the HyperCycle context, enabling accelerated execution of smart contract languages associated with a variety of different blockchains. The tightness of the coupling possible between HyperCycle and another chain, however, depends on the particular nature of the other chain. One chain with which especially tight coupling appears possible is Cardano.

Cardano's smart contract language, Plutus, is a pure functional language, which makes it relatively straightforward to tightly connect Plutus contracts and HyperCycle MeTTa contracts, having them share state and exchange information while in the course of executing. At a technical level this is achieved via connecting the Rust MeTTa runtime with the Haskell PLutus runtime using foreign function interfaces. The lack of reliance on imperative programming in both MeTTa and Plutus makes it generally straightforward for contracts in either language to flexibly and rapidly subcontract work to contracts in the other language.

The potential for close linkage between HyperCycle and Cardano arises at a nore fundamental level than smart contracts, though, and is rooted in Cardano's Extended UTXO transaction model, which enables shared state between Cardano and other chains using the (still in development at time of writing) Hydra protocol. ' HyperCycle can also leverage the UTxO model such as Cardano blockchain and also the Plutus smart contract language and numerous other related features, however substituting a different consensus mechanism and a different system for data management.

The Extended UTXO Transaction Model

Cardano's Extended UTxO (EUTxO) model [Fou21] extends the standard UTxO model underlying Bitcoin in a manner explicitly designed to support smart contracts operating according to functional programming principles.

Transactions in a UTxO ledger contain a set of inputs and outputs, where outputs lock an amount of cryptocurrency, such that only authorized inputs of subsequent transactions can connect and consume those funds. The set of outputs that are dangling (unconnected) pending validation, at any given point in time, are the unspent transaction outputs (UTxOs) . In addition to the locked currency, each output also comes with a validator predicate; and each input comes with a redeemer value. To determine whether a given input of the currently validated transaction is permitted to connect to a currently dangling and unspent output, the software determines whether the validator predicate the output applies to the redeemer.

The EUTxO model extends the UTxO model in two ways:

- Instead of restricting the validator/redeemer mechanism to dealing with signatures, addresses in the EUTxO model can contain arbitrary logic in the form of scripts. When a node validates a transaction, the transaction will look up the script provided by the output's address and will execute the script if the transaction is allowed to use the output as an input.

- Outputs in EUTxO can carry essentially arbitrary data in addition to an address and value. This allows scripts to carry, for instance, information regarding the state of long-running smart contracts.

The script associated with a transaction can access the data being carried by the output, the transaction being validated, and some additional pieces of data called redeemers, which the transaction provides for every input; based on this it can carry out complex logic to determine whether the given input is permissible or not. The fact that the validator can inspect the entire validated transaction enables validators to enforce that contract invariants are maintained across entire chains of transactions.

Key to the EUTxO model is that the success or failure of transaction validation depends only on the transaction itself and its inputs, and not on anything else on the blockchain. This works naturally with the "pure functional programming" nature of the Haskell language which underlies the Plutus smart contract system; in a pure functional language like Haskell, computation is broken down into functions whose outputs depend only on their inputs, without side-effect as are rampant in imperative language. The functional nature of Plutus is the central aspect making Cardano smart contracts tractably formally verifiable.

The purely functional nature of EUTxO validation also means the validity of a transaction can be checked off-chain, before the transaction is sent to the blockchain. A transaction can still fail if some other transaction concurrently consumes an input that the transaction is expecting; however, if all inputs are still present throughout the period of execution, the transaction is mathematically guaranteed to succeed.

The differences between this UTxO-based model and the account-based models used in Ethereum and other similarly structured blockchains necessitate significantly different design patterns on the DApp level. EUTxO makes it much more tractable and straightforward to create DApps with provable guarantees regarding their behavior; and EUTxO also provides a superior foundation for powerful extensions to the core Cardano model like HyperCycle. However, it does require a way of thinking about transactions and data representation that feels initially unfamiliar to developers who got their start with account-based blockchains. EUTxO smart contracts can be formally modeled using a type of state machine called a *constraint emitting machine* (CEM). Based on Mealy machines, each one consists of a set of states S_c, a set of inputs $i = I_c$, a predicate $final_c : S_c \to Bool$ identifying final states, and a step relation $s\_ \to (s, \text{tx})$, which takes a state s on an input i to a successor state s? under the requirements that the constraints tx are satisfied. Cardano implements CEMs on a EUTxO ledger (the mainchain) by representing a sequence of CEM states as a sequence of transactions, each of which has a state-machine input i_c and a state-machine output o_c, where the latter is locked by a validator $?_c$, implementing the step relation (exceptions being the initial and final state, which have no state-machine input or output respectively).

Hydra: Cardano's Interoperability Solution

[7]

Hydra [CCF$^+$20] is the algorithmic and software framework enabling the Cardano mainchain to communicate efficiently and elegantly with other blockchains. It supports implementation of what would informally be called sidechains, and also of proxies to other fully autonomous and separate blockchains.

When Hydra is used to connect to another blockchain network, the "head parties" are a distinguished set of high-performance and high-availability participants in this other network. "Tail parties" are other participants in this other network, who may be unreliable in terms of being online at any given

[7]This section draws mainly on [CCF$^+$20] and [Kia20]

time or being computationally capable of carrying out operations critical for decentralized network functioning. The Hydra architecture can be divided accordingly into the head protocol, the tail protocol, and the cross-head-and-tail communication protocol. These components are supported by additional underlying protocols for routing, reconfiguration and virtualization.

The head protocol is the only portion of Hydra currently developed to a state of reasonable maturity. It allows the heads parties in a blockchain network connected to the Cardano mainchain to rapidly process large numbers of transactions with minimal storage requirements by way of a multiparty state channel. The tail protocol, which has not yet been publicly described in a detailed way, enables the network heads to provide scalability for large numbers of additional network participants who may use the system from low-power devices, such as mobile phones, and who may be offline for extended periods of time. The cross-head-and-tail communication protocol leverages virtualization to allow heads and tails to communicate without going through the medium of the Cardano mainchain.

Hydra's state channels are isomorphic to Cardano in the sense that they make use of the same transaction format and contract code as the underlying Cardano blockchain. This means smart contracts can be directly moved back and forth between Hydra state channels and the Cardano blockchain – so that Hydra state channels effectively yield parallel, off-chain siblings of transactions on the Cardano blockchain.

Head Protocol Transactions

In terms of transaction management, what is happening behind the scenes in using the Hydra head protocol to connect a network to Cardano mainchain is: The head parties involved cooperate to commit a set of UTxOs comprising the initial head state, which these head parties then evolve by handling smart contracts and transactions among themselves without mainchain interaction. Transaction validation, including script execution, proceeds according to the exact same rules as onchain, leveraging the exact same validation code. In case of disputes or in case some party wishes to terminate the offchain protocol, the head parties decommit the current state of the head back to the blockchain - which then necessarily results in an updated blockchain state that is consistent with the offchain protocol evolution on the initially committed UTxO set. The mainchain doesn't need to know the detailed transaction history that led to the committed state, it only. needs to know the committed state. Further, the decommit process is designed such that, when the latest state in the head is very large, the head state can be decommitted via parallel decommitment of small chunks. UTxOs can also be added to and removed from a running head without closing it.

To get a Hydra head started, any party may take the role of an initiator and ask a set of parties to participate. Each party then establishes pairwise authenticated channels to all other parties in the head. This public-key material is used both for the authentication of head-related onchain transactions that are restricted to head members and for multisignature-based event confirmation in the head. The initiator then submits an initial transaction to the mainchain that contains the head parameters. This automatically initializes a state machine for the head instance that manages the transfer of UTxOs between mainchain and head.. Each head member then attaches a commit transaction, which locks on the mainchain the UTxOs that the party wants to commit to the head.

From this point on, the head enters into an open state The head members continue running the offchain head protocol, which evolves the initial UTxO set independently of the mainchain. In the case that some head members fail to post a commit transaction, the head can be aborted by going directly from initial to final.

Once a head is in the closed state, the underlying state machine grants parties a contestation period, during which each head party has one chance to contest the closure by providing the certificate for a newer head UTxO set. Contesting leads back to the state closed. After the contestation period has elapsed, the state machine may proceed to the final state.

What's happening on the mainchain while a Hydra head is evolving offchain is that the mainchain Hydra protocol:

1. Upon head initiation, locks the mainchain UTxOs committed to the head
2. Keeps these locks and waits while the head is active
3. Facilitates the settlement of the final head state back to the mainchain after the head is closed.

The end result is a dynamic that replaces the initial head UTxO set by the final head UTxO set on the mainchain in a manner that respects but does not persist the complete set of head transactions.

Some careful record-keeping is required here behind the scenes, in order to ensure that each head member posts exactly one commit transaction and that the open transaction faithfully collects all commit transactions. To enable this, a single non-fungible "participation token" is issued to each head member, connoting a capability and obligation to participate in the head protocol.. This token must flow through the commit transaction of the respective head member and to be valid the open transaction must collect the full set of participation tokens.

Hydra Tail Protocol

The Hydra head protocol has many powerful use-cases on its own. For instance, a natural strategy for porting the SingularityNET marketplace to Cardano is to map the multiparty escrow contract in the Ethereum-based SingularityNET implementation into a Hydra head. The SingularityNET agents participating in the multiparty token and API access transaction are then the head parties and the head instance remains active until the escrow is cleared and the multiparty transaction is done.

However, if one wants to associate a sizeable external network with Cardano using Hydra as the interface in a "sidechain" like design pattern, it's necessary to also handle the situation where there are many participants in the HyperCycle "sidechain" that are intermittently online or have unreliable or minimal computational power. This situation leads to a variety of security challenges, as there are various scams a network participant can carry out more effectively when not required to be consistently online to participate in transactions. The Hydra Tail protocol [Kia20] has two key mechanisms for addressing this:

- requiring participants to put collateral on the mainchain, which will be lost if fraud is attempted
- instantiating a Challenge-Response-Protocol on the mainchain, with which clients can dispute claims by any participant

This is one aspect of the default Hydra tail protocol that will likely be heavily customized for HyperCycle, via leveraging Proof of Reputation and TODA/IP mechanisms. Given HyperCycle's particular requirements and the underspecification of the Hydra tail protocol, the approach we are planning to take in the HyperCycle design is to create a custom version of the Hydra tail protocol that suits HyperCycle's needs, incorporating mechanisms and ideas beyond those that have previously occurred in the Cardano sphere.

References

[AK21] Oladotun Aluko and Anton Kolonin. Proof-of-reputation: An alternative consensus mechanism for blockchain systems. *International Journal of Network Security & Its Applications (IJNSA) Vol*, 13, 2021.

[BGT21] Alexey Potapov Ben Goertzel and Hyperon Team. Opencog hyperon. 2021. `https://wiki.opencog.org/w/Hyperon`.

[Bra13] Edwin Brady. The idris programming language. In *Central European Functional Programming School*, pages 115–186. Springer, 2013.

[Bra20] Edwin Brady. Idris 2: Quantitative type theory in action. Technical report, Technical Report. University of St Andrews, Scotland, UK. https://www. type ?, 2020.

[Bul20] Marius Buliga. Artificial chemistry experiments with chemlambda, lambda calculus, interaction combinators. *arXiv preprint arXiv:2003.14332*, 2020.

[CCF$^+$20] Manuel MT Chakravarty, Sandro Coretti, Matthias Fitzi, Peter Gazi, Philipp Kant, Aggelos Kiayias, and Alexander Russell. Hydra: Fast isomorphic state channels. *IACR Cryptol. ePrint Arch.*, 2020:299, 2020.

[CDD$^+$19] Shan Chen, Weiguo Dai, Yuanxi Dai, Hao Fu, Yang Gao, Jianqi Guo, Haoqing He, and Yuhong Liu. Thinkey: A scalable blockchain architecture. *arXiv preprint arXiv:1904.04560*, 2019.

[Fou21] Cardano Foundation. Understanding the extended utxo model. 2021. `https://docs.cardano.org/plutus/eutxo-explainer`.

[GGH$^+$17] Ben Goertzel, Simone Giacomelli, David Hanson, Cassio Pennachin, and Marco Argentieri. Singularitynet: A decentralized, open market and inter-network for ais. *Thoughts, Theories Stud. Artif. Intell. Res.*, 2017.

[GGT19] Adam Gravitis, Norman Goh, and Dann Toliver. Toda primer. 2019. `https://engineering.todaq.net/TODA_Tech_Primer_v1.0.pdf`.

[Goe16] Ben Goertzel. Cogistry: Accelerating algorithmic chemistry via cognitive synergy. 2016. `https://blog.opencog.org/2016/10/25/cogistry-accelerating-algorithmic-chemistry-via-cognitive-synergy/`.

[Goe19] Ben Goertzel. Singularitynet whitepaper 2.0. 2019. `https://public.singularitynet.io/whitepaper.pdf`.

[KGDI18] Anton Kolonin, Ben Goertzel, Deborah Duong, and Matt Ikle. A reputation system for artificial societies. *arXiv preprint arXiv:1806.07342*, 2018.

[KGP+19] Anton Kolonin, Ben Goertzel, Cassio Pennachin, Deborah Duong, Matt Iklé, Nejc Znidar, and Marco Argentieri. A reputation system for multi-agent marketplaces. *arXiv preprint arXiv:1905.08036*, 2019.

[Kia20] Aggelos Kiayias. Enter the hydra: scaling distributed ledgers, the evidence-based way. 2020. `https://iohk.io/en/blog/posts/2020/03/26/enter-the-hydra-scaling-distributed-ledgers-the-evidence-based-way/`.

[MG19] Gabriel Axel Montes and Ben Goertzel. Distributed, decentralized, and democratized artificial intelligence. *Technological Forecasting and Social Change*, 141:354–358, 2019.

[Pot21] Alexey Potapov. Metta language specification. 2021. `https://wiki.opencog.org/w/Hyperon`.

[SNST20] Pablo Lamela Seijas, Alexander Nemish, David Smith, and Simon Thompson. Marlowe: implementing and analysing financial contracts on blockchain. In *International Conference on Financial Cryptography and Data Security*, pages 496–511. Springer, 2020.

[ST19] Toufi Saliba and Dann Toliver. Toda/ip protocol 2.3: An internet communication protocol. 2019. https://github.com/Toufi/Whitepaper/blob/master/TodaIP_ProtocolV2.3.pdf.

[Sun] Sundaeswap fundamentals, author=Pi Lanningham and SundaeSwap Team, note = https://sundaeswap.finance/papers/SundaeSwap-2021-06-01-Fundamentals.pdf, year=2022.