Yala: A Bitcoin-Based Asset Protocol

Yala Labs

April 10, 2024

ABSTRACT

The Yala Protocol proposes a novel approach to unlock the programmability of BTC assets, focusing on developing a DeFi layer within the Bitcoin ecosystem. The protocol aims to overcome the challenges of implementing complex transactions using Bitcoin's native assets, such as its limited scripting language and scalability issues. Yala's architecture includes an Application Layer, Consensus & Data Availability Layer, Execution Layer, and Settlement Layer, enabling native DeFi transactions for BTC assets while maintaining the security and consensus of the Bitcoin network. The protocol incorporates an overcollateralized stablecoin protocol, an insurance derivatives service, and atomic swap functionality to facilitate seamless interactions between BTC and other blockchain systems. The Yala Finance system features essential components like Vaults, a liquidation algorithm, an automatic stabilizer, and an insurance module to provide a comprehensive DeFi ecosystem for BTC assets. The Yala Protocol's innovative approach aims to unlock the full potential of Bitcoin's blockchain by extending its utility beyond its primary use as a digital currency, enabling the integration of DeFi and other complex applications within the BTC ecosystem.

Key words. BTC, DeFi, BRC20, Stablecoin, Atomicswap, Insurance, Data Availability

1. Introduction

Bitcoin has emerged as the eighth-largest asset globally, with its valuation and security mechanisms universally recognized. Despite its acclaim, Bitcoin's utilization has predominantly been as a primary digital currency. Before Ethereum's advent, numerous endeavors aimed to extend Bitcoin's utility beyond mere cryptocurrency applications, albeit these were largely experimental due to Bitcoin's intrinsic limitations.

The advent of Ethereum, with its Turing-complete smart contracts, marked a significant shift in the blockchain ecosystem. Ethereum's smart contract functionality enabled the development of decentralized applications (dApps) that go beyond simple value transfer, encompassing a wide range of use cases such as decentralized finance (DeFi), non-fungible tokens (NFTs), and decentralized autonomous organizations (DAOs). This expanded programmability facilitated the implementation of complex logic and state transitions on the blockchain, in contrast to Bitcoin's more limited scripting capabilities.

As a result, most of the subsequent complex logic implementations after Ethereum's launch took place on the Ethereum network and other newer blockchains. The approval of Bitcoin exchange-traded funds (ETFs) underscored a pivotal moment, heralding traditional finance's acknowledgment of cryptocurrency value. This necessitated Bitcoin to introduce compatible financial products to enhance its appeal and broaden its user base.

Bitcoin's Total Locked Value (TVL) from WrappedBTC is a mere $2.8 billion, in stark contrast to its market capitalization of $134.6 billion, and significantly dwarfed by the $97 billion TVL across various blockchain networks. Despite holding a 51.9% market share, Bitcoin's representation in the DeFi TVL is just 2.89%. This scenario underscores the vast potential for DeFi within the Bitcoin ecosystem, signifying a crucial trajectory for Bitcoin's evolution.

DeFi functionalities on Bitcoin primarily utilize the Lightning Network, with the RSMC (Revocable Sequence Maturity Contract) and HTLC (Hashed Timelock Contract) protocols facilitating peer-to-peer transactions and off-chain transaction executions on Bitcoin. The challenges in deploying complex logic applications on Bitcoin are primarily attributed to its scripting language, characterized by a lack of Turing completeness and scalability issues.

Ethereum's whitepaper delineated several Bitcoin scripting deficiencies, including Turing incompleteness, value-blindness, statelessness, and blockchain unawareness. These limitations are anticipated to be mitigated through various Bitcoin protocol enhancements, such as introducing the `OP_RETURN` opcode in 2014, enabling the incorporation of off-chain states. Subsequent upgrades like SegWit and Taproot have imparted a degree of Turing completeness to Bitcoin. These enhancements, coupled with an off-chain execution layer interacting with scripted contracts in the Witness component, can refine Bitcoin's computational capabilities, encompassing most of Bitcoin's Layer 2 solutions.

The Yala Protocol proposes a novel paradigm, introducing two distinct strategies to bolster the interoperability and intrinsic functionality of BTC assets while assessing their integration viability. The protocol aspires to unlock the programmability of BTC assets, with an initial focus on developing a DeFi layer within the BTC ecosystem based on this conceptual architecture. Additionally, the Yala Protocol encompasses an overcollateralized stablecoin protocol on BTC assets, an inscription stablecoin protocol, and an insurance derivatives service encapsulated within a BRC-20 framework. At the interoperability level, the protocol employs atomic swaps to facilitate seamless interactions between BTC and other blockchain systems, enabling the deployment of DeFi protocols across multiple blockchain platforms.

2. Background

2.1. Challenges for running DeFi on Bitcoin

BTC requires a scalable solution to serve as the infrastructure for DeFi and even more ecosystem applications. We first consider what issues a blockchain must solve to run DeFi applications:

Security: The blockchain must ensure robust security to prevent issues such as double-spending, unauthorized modifications to transaction records, and other malicious activities. High TVL also reinforces trust in DeFi platforms.

Programmability: Critical for DeFi, the blockchain should support complex financial transactions and protocols, ensuring reliable and accessible outcomes from contract executions [5].

High throughput and low transaction costs: DeFi applications demand high transaction throughput and low costs, necessitating a scalable and efficient infrastructure.

Governance mechanism: A robust governance framework is crucial for encouraging community involvement, consensus, and managing updates to the network.

Risk control: The financial risks in DeFi exceed those in traditional trading, requiring dependable price data sources and risk mitigation strategies.

Given these prerequisites, Bitcoin's original design seems less suitable for the complexities of DeFi. However, potential solutions include:

1. Bitcoin's capability to support advanced smart contract logic has improved with recent upgrades. Using Merkleized Abstract Syntax Trees (MAST), DeFi logic can be encapsulated into scripts within the Witness component, intended for off-chain node execution rather than on the main chain.
2. Capitalizing on Bitcoin's unmatched security, DeFi transactions could be streamlined to Unspent Transaction Output (UTXO) transactions against Pay to Taproot Hash (P2TH) addresses, ensuring secure settlements for DeFi contracts. Incorporating yield farming concepts could encourage Bitcoin users to contribute their assets, enhancing TVL.
3. Considering the high transaction fees and prolonged block times on the Bitcoin network, limiting throughput, a significant portion of DeFi transactions should be processed off-chain. Bitcoin would then finalize the settlements, ensuring consensus and security.
4. Implementing governance mechanisms for DeFi community members is crucial for protocol governance, network upgrades, and parameter settings within the Bitcoin ecosystem.
5. Bitcoin's volatility, accentuated by its 10-minute block intervals, increases risk in its DeFi ecosystem compared to other chains, necessitating reliable oracles for price feeds and effective risk management measures.
6. Introducing a Bitcoin-native stablecoin could significantly stabilize liquidity in DeFi applications, serving various roles, including lending, hedging, and settlements.

2.2. Existing solutions

There are various proposed solutions for expanding Bitcoin to support complex applications such as DeFi, primarily Bitcoin Layer 2 solutions taking the form of rollups or EVM-compatible sidechains with similar underlying approaches. Rollup solutions generate fraud or validity proofs for off-chain transaction execution, solutions that are then validated by publishing the proofs to scripts within Bitcoin's Taproot witness data. At the same time, the EVM sidechain approach involves launching a new EVM-compatible blockchain network that interacts with the Bitcoin mainchain cross-chain. However, practical rollup implementations face challenges, including constraints on proof complexity due to limited witness data size. Additionally, simply posting data on Bitcoin's blockchain does not automatically ensure the authenticity of off-chain rollup transactions. As a result, most current Bitcoin rollup projects opt for a "sovereign rollup" or "client validation model" where validators synchronize and validate the entire rollup state data off-chain, failing to leverage Bitcoin's consensus and security model, exemplified by the client-validated RGB protocol.

A practical strategy for rollups involves leveraging the BTC network to validate transaction commitments. It's crucial to also have mechanisms that ensure assets within the rollup can be safely accessed or recovered during unexpected incidents or system breakdowns, even when rollup nodes/sorters are unavailable or decline transactions. A secure emergency channel allows for the retrieval of assets. BitVM, integrating fraud proofs with Taproot's MAST scripts, is a notable solution adopted by many rollup projects. However, BitVM's major drawback is its absence of an emergency asset release feature, posing a risk of asset loss in the event of attacks or failures.

Besides the direct Bitcoin expansion BTC L2 solutions, DLC.link proposed a different solution, mapping BTC to Ethereum L2 based on the DLC protocol, to participate in the ETH DeFi ecosystem as dlcBTC. This only provides cross-chain interoperability for Bitcoin, relying on the security and consensus of other chains, not exploiting the characteristics of the BTC ecosystem. However, the use of Discreet Log Contracts (DLCs) is a very valuable BTC contract primitive, essentially a condition judgment between transaction parties, determining the final direction of funds based on an agreement by two or more parties on the outcome of a specific event chosen by one or multiple oracles.

We believe native DeFi applications and other ecosystem applications for Bitcoin should be built directly utilizing Bitcoin's native asset types, including BTC itself, BRC20, Animoca's tokens, Taproot Assets, Omni Layer tokens, and more, with data structures encoded either in `OP_RETURN` outputs or through `OP_FALSE OP_IF...OP_ENDIF` script constructs within the witness data exhibiting similar formats. These diverse native asset encodings can be unified under the "inscriptions" model enabled by the Ordinals protocol, providing a standardized format for associating arbitrary data with satoshis and storing this content off-chain in Taproot script-path spend scripts. Building upon inscriptions allows layering more complex business logic onto off-chain indexer servers, such as introducing new operational primitives like "list", "collateralize", "destroy", "authorize" alongside "mint", "deploy", and "transfer" under the inscription JSON "op" field, with combinations evolving into higher-level functionalities like swaps, lending, inscribed financial instruments ("Inscription-Fi"), and even complex decentralized applications spanning social networks and gaming ("SocialFi" and "GameFi") - thereby standardizing Bitcoin's native assets into a richly extensible framework for native DeFi and broader applications deeply integrated with Bitcoin itself.

The Yala Protocol is a proposed solution that addresses the challenges of implementing complex transactions using Bitcoin's native assets. Yala draws on the concept of decoupling to apply it to the Bitcoin ecosystem, leveraging the security of the BTC network and the Turing-completeness of other blockchains. Yala's native operability aims to realize the programmability of BTC assets, and the protocol plans to first implement a DeFi layer on Bitcoin based on this theoretical architecture.

2.3. Native operability and interoperability on Bitcoin

We evaluated the limitations and reference technologies of current solutions. Taproot's MAST enables complex transaction logic on Bitcoin, while DLCs streamline transaction processes by moving most operations off-chain, ensuring both security and privacy. Leveraging these technologies, transaction contracts are stored similarly to Ordinals, facilitating direct on-chain storage and off-chain execution by nodes. This approach allows contracts to be executed in a manner akin to Ordinals, blending efficiency with blockchain integrity.

This approach maintains the intrinsic qualities of Bitcoin assets in DeFi, addressing the need for high throughput and cost efficiency. In DeFi's multifaceted transaction processes, it's unnecessary for BTC to be involved in every step. The primary value of the BTC main chain in executing contracts lies in its security and consensus capabilities. Therefore, we should preserve these aspects while separating all other system functions from BTC. A modular approach allows us to extend Layer1 functionalities, enhancing main chain scalability. By decoupling system functions and strategically distributing tasks across modules, we significantly boost system throughput and concurrently reduce transaction costs.

Blockchain interoperability allows for the sharing of information and value across different blockchain networks, facilitating direct interaction between users and developers with multiple platforms. The aim of achieving Bitcoin interoperability is to enable those holding BTC to leverage their assets in transactions on other blockchains without needing to sell. Methods to achieve this include cross-chain bridges, atomic swaps, and relay chains. A key challenge is conducting transactions with BTC assets on other chains without significantly burdening BTC's resources. Thus, mapping BTC assets for use on trusted alternative blockchains presents a viable strategy for providing BTC interoperability, though it's not without its challenges.

Most Bitcoin Layer 2 solutions utilize cross-chain bridges, where security hinges on creating cryptographic proofs off-chain that are later verified on the Bitcoin blockchain. This approach, however, runs into considerable difficulties, especially because the Bitcoin script language, designed for simplicity and security, isn't well-suited for complex verifications. This limitation raises concerns about the practicality of efficiently executing these advanced verification processes within Bitcoin's existing framework. Furthermore, coordinating emergency risk management across different blockchains adds layers of complexity to these systems. L2 projects are exploring the reactivation of the `OP_CAT` operator to enhance data interaction capabilities, but this move could inadvertently increase the risk of Distributed Denial of Service (DDoS) attacks, adding another dimension of concern.

Another approach is Drivechain, whose idea was detailed in two proposals: BIP 300, involving a system called Hashrate Escrows, and BIP 301, related to Blind Merged Mining. The Drivechain proposal aimed to let developers customize sidechains and use Bitcoin locked on the main blockchain to manage assets on these sidechains. However, neither BIP 300 nor BIP 301 received the necessary approval to be implemented. As a result, the Drivechain approach is generally regarded as unreliable.

Atomic swaps facilitate direct cryptocurrency exchanges between two parties on the Bitcoin network, leveraging its script language without needing a central authority or intermediary. This process uses special types of scripts, such as those found in Hashed Timelock Contracts (HTLCs) within the Lightning Network. These scripts ensure that transactions can only be completed if certain conditions are met, effectively creating a secure and trustless exchange mechanism. While atomic swaps offer a decentralized way to trade cryptocurrencies, they face challenges like the "replacement cycle attack," which questions the security of the Lightning Network. Despite these concerns, atomic swaps are a significant advancement for peer-to-peer transactions.

2.4. Native SegWit, Taproot and BRC-20

The SegWit upgrade eliminated Bitcoin transaction malleability by splitting transactions into two parts. It removed the unlocking signatures ("Witness data") from the original portion and appended them as a separate structure at the end. This approach mitigated the block size limit, continuing to store sender and receiver data in the original portion with the new "Witness" structure containing scripts and signatures. The original data segment is counted normally, but the "Witness" segment is effectively counted as one-quarter of its actual size. This upgrade expanded the complexity achievable by BTC Script. Although the space provided is insufficient for complex DeFi transactions, it is enough for simple conditional statements, thus laying the foundation for the later development of the Lightning Network. Taproot includes three separate Bitcoin Improvement Proposals (BIPs): BIP 340, BIP 341, and BIP 342, corresponding to Schnorr signatures, Taproot, and Tapscript, respectively. The SegWit upgrade's Pay to-ScriptHash (P2SH) relieved senders of the costs associated with PubKey scripts occupying significant transaction volumes. As transactions become more complex, with scripts specifying numerous spending conditions, the transaction volume could become exceedingly large. This issue was effectively addressed in BIP114 and BIP117 with the MAST concept. Taproot uses the concept of Merkle branches, revealing only the part of the script executed to the blockchain rather than all possible ways the script could execute. Among the various known mechanisms for implementing this, using a Merkle tree as part of the script structure maximizes space savings. Schnorr signatures introduced key aggregation, combining multiple public keys and signatures into one, making Bitcoin transaction signatures faster, more secure, and easier to process.

Tapscript modified the `OP_CHECKSIG` and `OP_CHECKSIGV -ERIFY` signature operation codes to verify specified Schnorr signatures and simplified the `OP_CODESEPARATOR` operation code for efficiency. The signature checked by `OP_CHECKSIG` is a signature of the hash of the preimage. The preimage includes the entire transaction's outputs, inputs, and lock script. Typically, the lock script is complete. However, if `OP_CODESEPARATOR` is executed before `OP_CHECKSIG`, the preimage only includes the portion of the lock script from the position of the most recently executed `OP_CODESEPARATOR` to the end of the script, thus containing only part of the lock script. This feature can be used to reduce the size of the preimage, thereby reducing the overall size of the transaction. The `OP_CODESEPARATOR` can compress the time-lock contract script code size by 80%, greatly aiding the implementation of conditional judgment scripts.

The Taproot upgrade provides a foundation for many schemes to enhance BTC's scalability, including Ordinals, Taproot Assets and other asset interaction protocols. All of these offer a basis or reference for the underlying architecture of Yala Protocol. The BRC20 based on Ordinals provides a way to write assets into UTXOs, setting deploy, mint, and transfer functions as the basis for operating assets on BTC. This logic allows us to grant BTC assets more extensibility, initially through DeFi.

2.5. New framework from BRC-20

This compartmentalized architecture significantly eases the burden on Layer 1, making it particularly suitable for BTC, whose tasks are notably demanding and transaction computation costs on the network are quite high. With Bitcoin's consensus based on Proof of Work (PoW), its security is considered the highest among all blockchain systems. The scripting language for BTC transactions is Turing incomplete, and the cost of computing transactions on BTC is prohibitively expensive. As a result, BTC has traditionally been used solely for transfer transactions. If we could take a compartmentalized approach to extend BTC to make it a LazyLedger Tas et al. [9], By offloading tasks other than UTXO transaction state consensus and security to other components, we could obtain a low-cost BTC ecosystem capable of executing complex transactions. Then, our task would be to ensure that Bitcoin solely manages consensus and security.

The upgrades of SegWit and Taproot have laid a solid foundation for this vision. The Segregated Witness upgrade separates the Witness from transaction inputs, increasing transaction capacity. With Taproot's Schnorr signatures, transactions can be batch verified, and Tapscript, alongside MAST, enables Bitcoin to run smart contracts similar to Ethereum's, allowing for complex transactions. These upgrades have introduced the concept of BRC20, based on Ordinals, where unexecuted script codes are written into the Witness's taproot script-path spend scripts, resembling:

```
OP_FALSE
OP_IF
  OP_PUSH "ord"
  OP_PUSH 1
  OP_PUSH "text/plain;charset=utf-8"
  OP_PUSH 0
  OP_PUSH "Hello, world!"
OP_ENDIF
```

Ordinals inscription. Ordinals confer additional token value to sats, allowing them to be collected and traded as curios. Any content can be engraved on an individual sat, creating unique native Bitcoin digital artifacts that can be stored in a Bitcoin wallet and transferred using Bitcoin transactions. The deployment, minting, and transfer of Ordinals tokens became the foundation of the earliest BRC-20 standard [12].

UniSat's modular proposal presents a new method to accommodate diverse inscription-based applications within the BRC-20 framework. Modules function separately, allowing indexers to parse only relevant data, ensuring consistency across the BRC-20 system. Modules are categorized into Black Modules, supporting deposits, and White Modules, enabling withdrawals. Black Modules transition to White upon unanimous indexer approval. White Modules necessitate full indexer support and consensus algorithm implementation for operation.

The BRC-20 update introduces deployment and new withdrawal inscription formats. Deploying it requires specifying the name, source, and optional initialization data, with the source linking to the code-defining module functionality. Withdrawals requiring a module ID are permitted only for White Modules.

This design enables complex transactions that would otherwise require Bitcoin processing to be handled on another layer. Yala's approach utilizes Taproot Script to simplify complex transaction processes. It decouples transaction execution, settlement, consensus, and data availability of Bitcoin assets, entrusting all complex transactions and settlements to off-chain execution. This transforms Bitcoin into a LazyLedger, solely responsible for the system's consensus and security.

3. Yala architecture

Yala architecture includes an application layer, a consensus & data availability (DA) layer, an execution layer, and a settlement layer. The application layer is where the application runs, and the modules on it define the logic of state changes. The module can be a smart contract in the EVM or a tapscript in BRC20. The src parameter in the Vaults Module initiated by the user in the execution layer points to the contract defined by the application layer. Indexer maintains the off-chain status of the system, and transactions of BTC derivative assets are changes in off-chain status, so Indexer maintains consensus on the complex system and maintains data availability. State changes in Yala require Indexer consensus to be executed. Users initiate the Vaults Module based on BRC20 to modify the off-chain status. The Vaults Module is the execution environment for status changes in Yala. Indexer guides the status changes of the Vaults Module based on Oracle's price feed. The status change of BRC20 is eventually reflected as a UTXO transaction, and the transaction in Yala is finally settled in BTC Mainet. Yala's application layer is essentially the logic of state changes defined by smart contracts and Tapscript. The user initiates the Vaults Module and selects the application to be executed by pointing the src parameter to the corresponding contract address. Based on this design, developers can use Yala's SDK to implement customized modules to develop BTC ecological applications.

3.1. Application layer

The application layer in Yala includes loan modules, restaking modules, and stablecoin modules. The stablecoin module forms the foundation of the Yala Finance system, with Yala Finance's stablecoins also being BRC-20. Since BRC-20 cannot be minted, the system maintains a Stablecoin Reserve Pool, essentially the stablecoin module. Users initiate the transfer and withdraw inscriptions to the Stablecoin Issuer Module to simulate traditional stablecoins' mint/burn process. The owners of the stablecoin module are multiple automated running nodes governed by Yala token holders, with these automated nodes executing the withdrawal operations for the stablecoin module.

Yala's stablecoins use an over-collateralization mechanism. Borrowers can initiate a Loan Module, transferring over-collateralized BTC assets to the module through transfer inscriptions. When the Loan Module meets the conditions, automated nodes initiate a Withdraw Module to extract the corresponding $YU to the borrower's address. The Loan Module can only be initiated by over-collateralized participants, with its source pointing to the open-source lending program published by the Yala Foundation, and participants need to modify the init parameters to set their loan amount, initiating transfer inscriptions to the Loan Module based on the asset amount required for collateral as feedback from the indexer. If we expand the multi-sig channel in atomic swaps, upgrading the 2-of-3 multi-signature to an (n-1)-of-n multi-signature channel, this approach is expected to realize a restaking scheme based on BTC Team (2023). [10].

3.2. Data availability layer

DeFi transactions require high throughput and frequently generate large amounts of transaction data. Publishing DeFi-generated

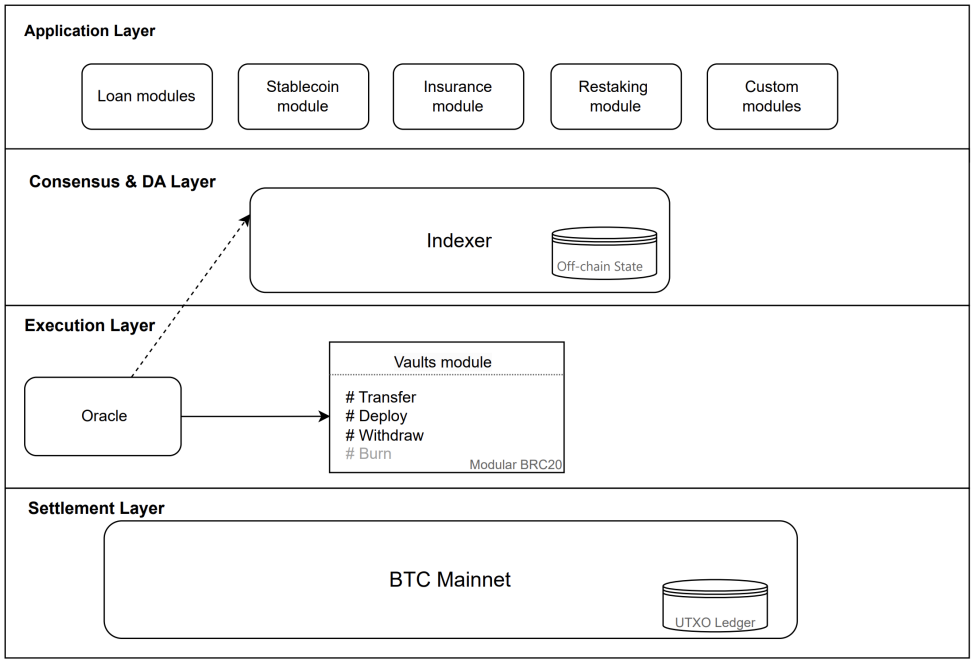

Fig. 1. Yala architecture

data to BTC would incur significant costs; hence, DeFi transaction data cannot be directly stored on BTC. In the Yala system, off-chain state updates and consensus are achieved by Indexer nodes, which also maintain the security and data availability of the off-chain state. This approach saves the cost of uploading data to BTC while providing real-time updates for DeFi states.

Data availability refers to the capability of data stored in the blockchain network to be effectively accessed and used by all participating nodes. Unlike data availability in other blockchains, BTC assets are in UTXO format, comprising two state changes, and the BTC block interval is long, about 10 minutes. When transacting BRC20 or other BTC assets, we need not deal with any data related to the said assets. Indexers capture the global state and balance in the form of witness scripts. Thus, the implementation of data availability in Yala only needs to target off-chain state changes of assets in witness script format, which is maintained by Indexers. The on-chain state change of UTXO is reflected in transfer transactions, which can provide security for off-chain state changes. The change in the off-chain state will eventually be reflected in on-chain UTXO transfer transactions, with BTC's consensus and security verifying the final state of asset transactions on Yala. Assuming that Indexers are trustworthy and that the final changes in the off-chain state synchronize with UTXO transactions, the off-chain state's data availability can be independent of the BTC main chain. Our challenge is implementing Indexers' trustworthiness and the secure real-time update of off-chain states.

Solutions for data availability include Data Availability Sampling (DAS), which has been validated in the EVM ecosystem. In Yala's DA layer, we can directly use DAS to reduce verification costs; validators only need to randomly download part of the data blocks to verify that all data is available. Erasure coding and KZG Polynomial Commitments will also be used to implement DAS in Yala's DA layer. The content to be verified by Yala's DA layer includes transaction data, the Merkle tree of transaction data, and Commitments. Transaction data will be directly verified by Indexers, who will also generate the Merkle tree. Commitments can currently be produced by Indexers as well, and we may consider adding dedicated validators in the future. The proof content will be distributedly stored and made public for a period, set to one month (referencing the storage time of blobs on ETH), during which anyone can verify the proof content. A specialized project is already in development for the BTC DA layer solution; Nubit[1] is a Bitcoin-Native data availability layer with instant finality. We are considering using Nubit's DA layer solution directly to realize Yala's decentralized Indexer network and DA layer, to ensure the trusted premise assumption of Yala's DA layer Indexers.

In addition, considering another performance requirement of DeFi based on BTC — the secure real-time update of off-chain states, we have incorporated the Mempool. Mempool is a dynamic staging area for unconfirmed transactions, storing information related to unconfirmed transactions. BTC's long block interval could limit DeFi's TPS (Transactions Per Second). By using Mempool, Yala can complete interim state changes of DeFi transactions after Indexer consensus, pack multiple DeFi transactions into a batch, compute all state changes within the batch to aggregate into the final UTXO state change transfer transaction, and submit it to the BTC main chain for settlement. This process is essentially an off-chain state change Rollup, except that its verification process is executed off the BTC chain, ultimately reflected as a UTXO transaction on the BTC main chain.

3.3. Execution layer

The execution layer includes the Oracle and BTC Vaults Module. All DeFi transaction actions in Yala occur within Vaults Modules.

Oracle module: Vaults require real-time market price acquisition of collateral assets to determine when conditional executions are triggered. The Yala Foundation is responsible for maintaining the Oracle Module and Oracle Security Module (OSM). The Oracle Module obtains price inputs from off-chain sources. Off-chain Oracle nodes retrieve necessary data through off-chain

[1] Nubit: Bitcoin-Native Data Availability Layer with Instant Finality

data APIs, format it, and return it to the Oracle Module. To protect the system from attackers attempting to control the majority of oracles, Yala Protocol uses the OSM to receive price inputs rather than directly from oracles. OSM acts as a defense layer between oracles and the protocol, publishing prices with a 30-minute delay to allow for emergency defenses in case of oracle compromise. The Yala Foundation makes decisions on emergency oracles and the duration of price delays.

BTC Vaults module: To accommodate native DeFi transactions for BTC assets, Yala has set up the BTC Vaults Module, mapping BTC to on-chain BRC-20 tokens, enabling BTC to be transferred to BRC-20 Modules and participate in DeFi transactions.

3.4. Settlement layer

The BTC main chain finalizes transactions. In the Yala system, the transfer, withdrawal, and ownership transfer of inscription assets are essentially UTXO transactions. We ultimately attribute the security of complex DeFi logic to the security of UTXO transactions, successfully relying on BTC's consensus and security and utilizing BTC's maintenance of the UTXO state ledger as the transaction state data availability for the system.

4. Native BRC20-based DeFi attempts

Yala Finance features essential components like Vaults, liquidation algorithm, automatic stabilizer, and insurance provision. The Vaults module, central to the system, manages contract-required operations, including user actions such as Collateralized Debt Positions (CDPs), insurance, stablecoin, and Yala governance token savings pools. Oracles supply real-time pricing for these components. Initially, the Yala Foundation, consisting of Yala holders, coordinates rate setting, price stabilization, lending code maintenance, and market monitoring for crises. The liquidation mechanism recovers assets from non-repaying borrowers to avoid defaults, while the automatic stabilizer, linked to the stablecoin reserve, ensures the $YU price remains anchored. Together, the stabilizer, Yala Foundation, and liquidation mechanism ensure $YU's and the lending system's stability.

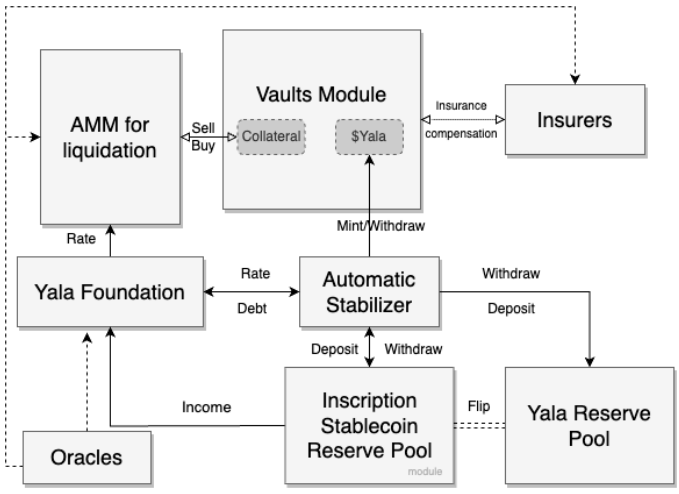

Fig. 2. Native BRC20-based DeFi architecture

4.1. Primary tokens

$YU is a USD-anchored inscription created by users over-collateralizing assets on BTC or through un-collateralized minting in a stablecoin savings pool. Once generated, $YU operates like any standard inscription on the BTC chain. $Yala, governing Yala Finance, is issued by the Yala Foundation or utilized to stabilize $YU when it deviates from its anchor. Users earn interest and rewards from successful liquidations. In case of stability threats from bad debt, the Automatic Stabilizer incentivizes keepers to manage $YU and $Yala exchanges to maintain balance, depositing acquired $Yala into the Reserve Pool. This mechanism also allows for keeper incentivization during surplus or deficit scenarios of $YU, supporting the system's economic stability. $Yala token holders can arbitrage to help stabilize $YU and participate in governance.

4.2. Vaults module

The Yala loan protocol adopts a component-based `BRC-20` framework, which encompasses functionalities such as *deploy*, *mint*, *transfer*, and *withdraw*. Contrary to the traditional *deploy* process in `BRC-10`, here users initiate a *Vaults Component* to mint $YU by transferring BTC or other assets. The borrowing process necessitates overcollateralization, with the value of pledged assets exceeding the borrowed $YU's value. Successful repayment triggers consensus conditions within the component, converting the *Vaults Component* into a *White Component* for asset withdrawal. In case of liquidation, assets are auctioned, and a keeper can unlock the component by transferring a lower-value $YU, also turning it into a *White Component*. The highest bidding keeper receives the pledged assets.

4.3. Module generation

The module is initiated by the user based on the BRC-20-module protocol's deploy script, which opens the lending process. The "source" points to the lending script code provided by Yala Foundation, and the script code is deployed in the MAST format. The "init" parameter sets numerical values for parameters such as the liquidation interval, lending interest rate, conversion rate, and so on.

```
{
  "p": "BRC-20-module",
  "op": "deploy",
  "name": "$YU_loan",
  "source": "at_code_address",
  "init": {   // set initial parameters
  }
}
```

MAST nodes contain string content for executable code and can link to multiple child nodes for different execution branches, with new branches added via the "addBr" method. Concatenating string content from a MAST path yields the code for a specific execution route.

Using BEVM's MAST as an example, the tree's root symbolizes the entire program, with subsequent nodes indicating subprograms. Each path represents a unique program execution branch, with the structure being Merkleized—leaf nodes are code hashes, and non-leaf nodes are child edge hashes. This allows a program's execution flow to be compactly represented with a hash string, making it possible to compress a program of length n to O(log n) and enabling complex DeFi programs

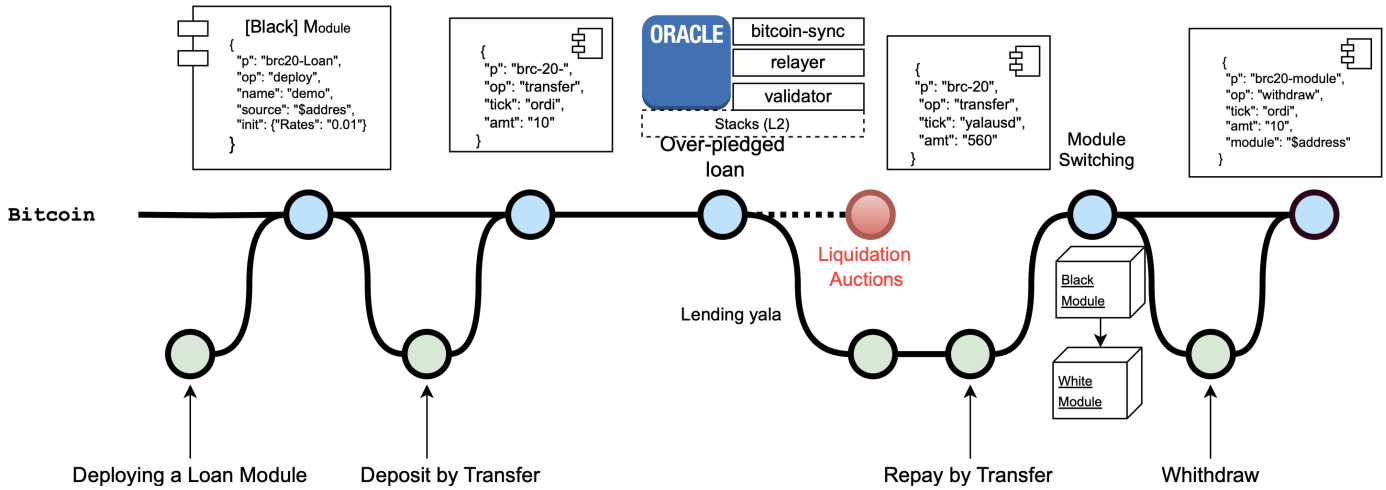

Fig. 3. Yala lending protocol flow

like Yala Protocol's to be integrated into a BRC-20 module efficiently.

For deployment, the "init" parameter must detail the user's pledged assets and their quantities. Upon initial execution, this data is sent to the $YU mint script managed by the Yala Foundation, initiating $YU minting based on asset valuations from the mint script. The script then communicates the lending outcomes and module address to all indexers for oversight and consensus.

4.3.1. Withdrawal of assets

Asset withdrawal happens under two conditions: loan repayment or asset liquidation. In the first scenario, upon timely loan and interest repayment by the user, the indexer swiftly reaches consensus. The user then uses a "withdraw" script to reclaim their pledged assets.

In liquidation, it occurs if the user defaults on the final repayment deadline or the asset value drops significantly below the liquidation threshold. Here, Keepers monitor for liquidation triggers. Once triggered, a Keeper buys the assets at the liquidation price, uses a "transfer" script for the module, unlocking it post-indexer consensus, and then employs a "withdraw" script to secure the assets. Keepers act as system arbitrageurs, facilitating asset liquidation.

5. Automatic stabilizer and the Yala Foundation

In its initial phase, the Yala Foundation acts as a centralized entity under community oversight, tasked with ensuring $YU's stability, managing the lending script, setting lending rates, and signaling emergencies. Inscriptions in the system don't allow for post-issuance adjustments, leading to the creation of the Inscription Stablecoin Reserve Pool managed by Yala holders. These holders vote on system parameters and upgrades. Initial $Yala and $YU pool inscriptions come from the Foundation's unsecured mint. However, the Foundation can't alter the Y pool inscriptions and automates the $Yala and $YU pools' operations through the Automatic Stabilizer, mimicking the minting and burning process of tokens.

5.1. Credit process

The Yala Foundation will mint all $YU to the $YU Reserve Pool until reaching its max supply, then transition to the Stabilized Lending phase. In this phase, new $YU isn't directly minted; instead, an "extraMint" method simulates minting by transferring $YU from the Reserve Pool to borrowers. Repayments are redeposited into the Reserve Pool, mimicking the original minting phase. The Automatic Stabilizer automates withdrawals and deposits related to the Reserve Pool, managing the $YU Module throughout.

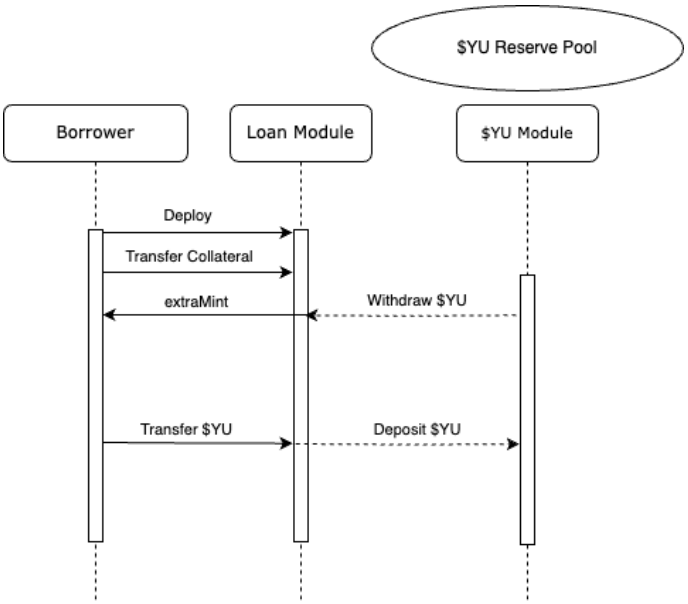

Fig. 4. Automatic stabilizer on credit process

5.2. System stabilization process

The Automatic Stabilizer not only simulates $YU's minting and burning during the lending cycle but also utilizes Yala to preserve $YU's price stability amid liquidations, splitting the process into surplus and debt auctions. $YU's surplus originates from Loan Module management fees—essentially, the interest

rate. Once a loan settles, any excess interest becomes surplus, announced alongside the module address. Keepers bid using Yala inscriptions, with the highest bid winning the surplus. The received $Yala is then placed into the $Yala Reserve Pool, mimicking the destruction of governance tokens to maintain economic stability. During rapid collateral price declines, bidders

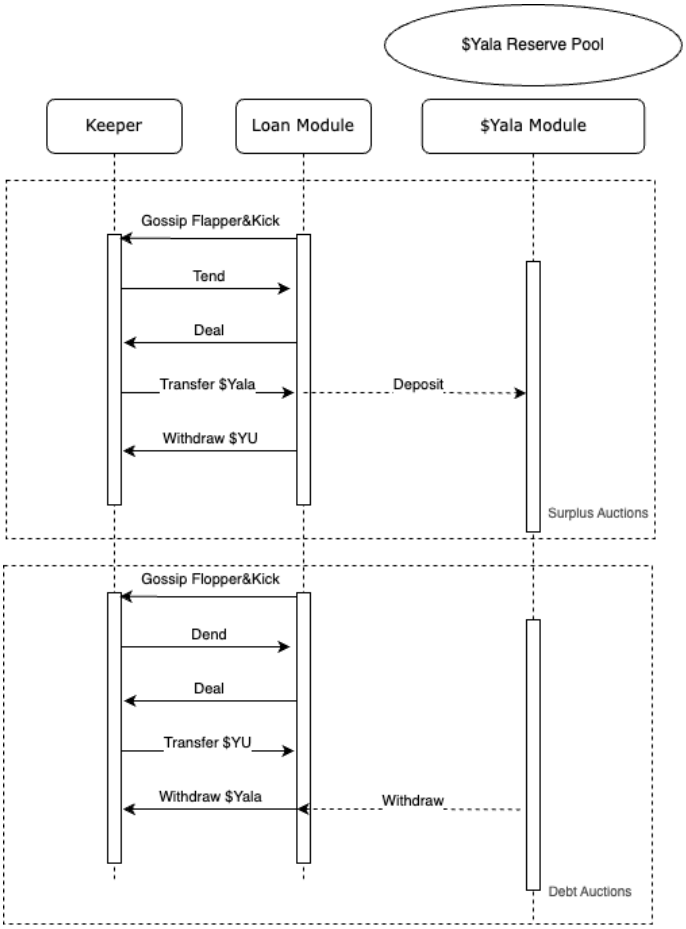

Fig. 5. Automatic stabilizer on system stabilization process

may pay significantly less than their maximum willingness, resulting in the liquidation process generating less $YU than the loan's value. This triggers the Debt Auction phase of the Loan Module, which announces the total debt and its address for bids. Bidders use $Yala for their offers, with the auction favoring the lowest bidder. The winning bidder transfers $YU to the Loan Module, secured by bidding with $Yala. The Automatic Stabilizer then compensates the winner by drawing $Yala from the Reserve Pool, effectively simulating the minting of governance tokens.

6. Liquidation mechanism

If the value of the collateral associated with a user's Loan Module experiences a significant decline and the user fails to timely settle interest payments or bolster collateral funds, triggering a drop below the stipulated minimum pledge rate, Keepers have the authority to submit a liquidation request to the Loan Module [4, 8]. Keepers simply need to adhere to the liquidation amount broadcasted by the Insurance Module, along with the corresponding Module's address. They can then execute a direct transfer to this address. Upon consensus agreement from the Indexer, the Module undergoes unlocking, transforming into a White Module. Subsequently, the Keeper initiates a withdrawal request, thereby securing the escalated collateral. The clearing fee, configured within the Insurance contract script by the Yala Foundation, presents an opportunity for Keepers to acquire collateral at a discounted rate, capitalizing on arbitrage gains.

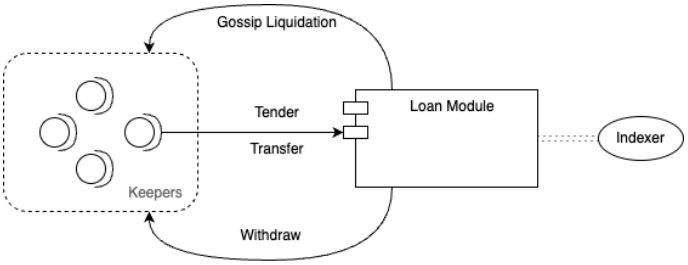

Fig. 6. Liquidation

6.1. AMM for liquidation

Facing the high volatility of the BTC market, we explored ways to mitigate borrowers' losses when asset depreciation triggers liquidation.

The Auto Market Maker[7], widely adopted in decentralized exchanges (DEX), offers a solution by automatically setting buy and sell prices through basic algorithms, functioning like an automated trader. This innovation addresses the cost and efficiency challenges of transitioning from traditional exchanges to DEXs, facilitating seamless, unauthorized, and decentralized trading of digital assets.

Curve's novel approach employs a reverse AMM within their latest protocol, Curve Stablecoin[6] , to tackle liquidation more effectively than traditional stablecoin or lending protocols like MakerDAO, Liquity, and Compound. By leveraging its AMM, Curve stablecoin enhances liquidation efficiency, significantly lowering the risk of accruing bad debts. Furthermore, the AMM aids in absorbing liquidated assets, thereby dampening market volatility induced by mass asset sales. This represents a key advancement over older stablecoin and lending models, offering a more resilient solution to liquidation challenges.

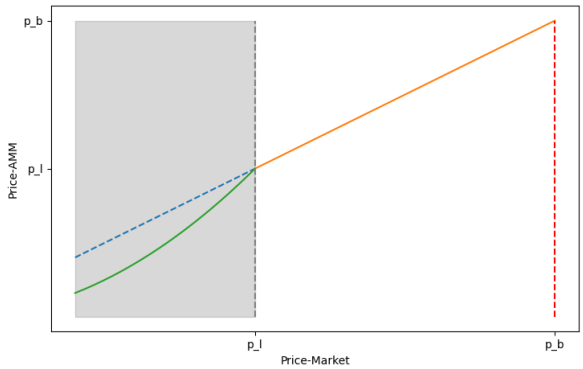

Fig. 7. AMM liquidation interval

Under the BRC-20 modular protocol, there are several issues to consider when using AMM for clearing, including the black module cannot withdraw when the conditions are not met, and the flexibility of Defi under ERC cannot be realized within the module. So we design the progressive AMM clearing process applicable to Yala Finance.

As shown in the above figure, the collateral price within the module AMM changes with the market price, p_b is the total value of the collateral when the borrower initiates the Loan Module, and the system does not do anything when the value of the collateral rises above p_b. p_l is the liquidation line set by Yala Foundation for collateral in advance according to the quotation of the prediction machine. When the market price of collateral is in the interval from p_l to p_b, the collateral price in AMM and the market price are equal, so that P_a is the collateral price in AMM, and P_m is the market price of collateral, which satisfies the function $P_a = P_m$.

The AMM liquidation mechanism is triggered when $P_m < p_l$, i.e., the market price is below the liquidation line and the borrower has not yet repaid the loan. Set

$$\frac{p_l}{p_b} = \alpha \tag{1}$$

where α is a constant set at Loan Module creation, i.e., the liquidation rate, that satisfies $0 < \alpha < 1$.AMM for Liquidation satisfies a constant equation similar to uniswap v3[11]:

$$(x + f)(y + g) = k, \tag{2}$$

where x denotes the total amount of \$YU in the Loan Module and y denotes the outstanding balance of collateral. As in Uniswap V3, the price (collateral price) in the AMM can be expressed like this:

$$P_a = \frac{x + f}{g + y}. \tag{3}$$

Unlike Uniswap V3, both f and g in the above equation are dynamic variables and are correlated with the external price P_m. Under the BRC-20 modularization standard, it is not appropriate to consider the condition of automatic buyback of collateral, which is cleared when P_m is in the interval $(0, p_l)$. The clearing process is initiated by the keeper who initiates the bidding of the collateral by the TRANSFER inscription, and only when the clearing is complete is the clearing of all TRANSFERS carried out, otherwise, once the collateral rises again to a healthy level, i.e., $P_m \geq p_l$, all transfer inscriptions are returned to revert P_a to the healthy curve after a delay of one prophet-safe time.

Next, we discuss the computation of the clearing function for the interval $(0, p_l)$, and we refer to the existing results of the LLAMMA algorithm to obtain the optimal AMM price function:

$$P_a = \frac{P_m^3}{p_b^2}. \tag{4}$$

At this point there is the function:

$$k = \frac{P_m^3 y_0^2}{(p_b - p_l)^2} \tag{5}$$

where y_0 is the total amount of collateral, the amount of collateral y = y_0, and the amount of \$YU `x = 0` when the Loan Module has just been depolymerized, at which point $P_m = p_b$, and

$$k = \frac{P_m^3 y_0^2}{(p_b - p_l)^2} = \frac{P_m y_0^2}{(1 - \alpha)^2} \tag{6}$$

, substituting into Eq. (1) and Eq. (2), we get

$$\begin{cases} f(y_0 + g) = k \\ P_m = p_b = \frac{f}{y_0 + g} \end{cases} \tag{7}$$

solves to:

$$\begin{aligned} f &= \sqrt{k p_b} \\ &= \sqrt{\frac{P_m y_0^2}{(1-\alpha)^2} \cdot \frac{P_m^3}{p_b^2}} \\ &= \sqrt{\frac{P_m^6 y_0^2}{p_b^2 (1-\alpha)^2}} \\ &= \frac{P_m^2 y_0}{p_b (1-\alpha)} \\ &= \frac{P_m^2 y_0}{p_b (1-\alpha)}, \end{aligned}$$

and:

$$g = \frac{\alpha p_b}{P_m (1-\alpha)} y_0 \tag{8}$$

The complete AMM constant product formula is then obtained:

$$(x + \frac{P_m^2 y_0}{p_b (1-\alpha)})(y + \frac{\alpha p_b}{P_m (1-\alpha)} y_0) = \frac{P_m y_0^2}{(1-\alpha)^2} \tag{9}$$

This yields y_0 that can be solved for from the current x, y values:

$$y_0 = \frac{\frac{\alpha p_b}{P_m} x + \sqrt{\frac{\alpha(1-\alpha) p_b}{P_m} + \frac{((1-\alpha) P_m^2}{p_b} y)^2 + 4(1-\alpha) P_m x y}}{2 p_0} \tag{10}$$

Estimated at the beginning of the user's deploy Loan Module until the liquidation is complete: when the collateral market price P_m decreases, the quantity obtained after all collateral in the Loan Module is traded into \$YU, which we will denote as x_{end}, is assumed:

The external price changes at an extremely slow rate, which means that the spread between the external price and the internal price of the Module at a given time is negligibly small. The internal price of the Module follows the external price, and since the spread between the internal and external prices of the AMM is negligible, the keepers do not incur losses due to the spread during the trading process.

According to Uniswap V3 (6.13), we can get the liquidation completion point estimation function:

$$x_{end} = x + \Delta x = x + \sqrt{k} \cdot \Delta \sqrt{P_a} = x + y \sqrt{p_l \cdot P_a} \tag{11}$$

The whole AMM clearing process is controlled by the clearing script pointed within the Loan Module, when clearing starts, each keeper can send transfer inscription to the Loan Module for clearing queue, the clearing script will recursively check the bids of the keeper in order, when it satisfies $\sum_i \Delta x_i = p_b \cdot y_0$ when. Unlike LLAMMA, Yala Finance's liquidation process is continuous and does not include band rounds. Because under the volatile BTC market, the plunge of inscriptions may generate bad debts extremely easily, the gradual liquidation can share the risk for the keeper, and also allow the keeper to compete for arbitrage and accelerate the liquidation.

7. BRC20-based insurance mechanism

In the insurance module design of Yala Finance, we have opted for the Takaful insurance model [3]. The Takaful insurance model is a cooperative insurance system, aiming to provide insurance protection to participants through the mutual sharing of risks and assistance.

7.1. Actor model

In this system, key roles are played by policyholders, insurers (operators), and shareholders.

Insurers, acting as operators, are tasked with:

- **Managing the Takaful insurance fund**. Their responsibilities include collecting premiums, overseeing investments, and handling claims, ensuring the fund's stability and operational integrity to meet participants' insurance demands.
- **Setting initial parameters**. Operators are responsible for establishing key parameters such as profit-sharing ratios, risk-sharing rules, and premium rates.
- **Crafting insurance contracts**. They create Takaful insurance contracts that define the rights and duties of participants, developing and implementing script contracts based on predetermined parameters and managing the insurance module.
- **Supervision and governance**. Operators implement supervision and governance mechanisms to ensure Takaful insurance activities' compliance and resilience. This includes managing fund use, protecting participant rights, and preventing fraud. For larger policies, they may need to draft additional insurance contracts, analyze policyholders' on-chain behaviors, and conduct KYC procedures as needed.

Shareholders contribute to the qard hasan (benevolent loan) fund, engaging in long-term profit-sharing and decision-making for the insurance fund. They may also serve as operators or simply invest in the fund and participate in its governance.

7.2. Takaful

Takaful operates by pooling contributions from members into a fund, with premiums financing mutual assistance for losses. This fund, along with all related activities such as investments and profit distribution, is managed by an operator. In cases where the Takaful Insurance Fund falls short of covering claims, operators use qard hasan (a benevolent loan) from company shareholders to fill the gap. Though not required, any future surpluses are first allocated to repay these loans, benefiting the shareholders.

Surpluses remaining after claims and expenses are reimbursed to participants, with the operational cycle and calculations outlined by the operator in a scripted contract. The fund employs a detailed mechanism for rebates, with thresholds designed to regulate fund withdrawals and maintain stability. Commonly, five thresholds are established: three for the Takaful fund to manage its levels and two for qard hasan to guide loans and repayments. Loans are made to elevate the fund to a safe level during deficits, and as the fund grows, reaching certain triggers allows for investments, distribution of excess to participants and the operator, and reserve setting for the future. Surplus in the qard hasan beyond a defined point triggers dividends to participants, ensuring equitable benefit distribution.

Assume that all n participants pay a premium of d to the Takaful insurance fund. The R_t denotes the reserve after all payments have been made in the period starting at time t. The net return is the total income minus the total expenses for a given time period. The total amount of benefits withdrawn from the Takaful Insurance Fund is S. The net benefit is the total revenue minus the total expenses for a given time period, i.e., $nd - S$. h is the surplus trigger point, and any amount exceeding the surplus trigger point at the point in the calculation cycle is the surplus of the fund, which should be distributed to the participants, the Takaful operator, and the shareholders. Takaful fund reserves satisfy the recurrence relation

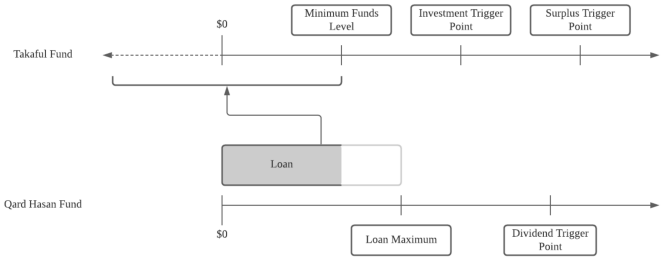

Fig. 8. KD modified risk model

$$R_t = h \wedge (R_{t-1} + nd - S)_+ \tag{12}$$

7.3. Insurance model in Yala

Yala's insurance model merges the profit-sharing and agent models, drawing inspiration from Takaful. In the profit-sharing approach, the operator allocates the Takaful fund's surplus proceeds once they surpass a predetermined surplus trigger. Conversely, in the agent model, the operator earns proceeds for fund management, creating a balanced structure that rewards both fund stewardship and communal surplus.

In the Yala insurance model, insurers oversee the Takaful Module and Qard Hasan Module. The Takaful Module, central to the system, manages the insurance pool and the lending script contract. Here, policyholders, termed Participants, choose their insurance module and initiate a script contract linked to the Takaful Module. This setup, pre-approved by Insurers, automatically entitles Participants to payouts, triggering a process where, upon permission, the indexer consensus transitions the module from Black to White, enabling Insurers to access the funds for claims.

The Insurance Pool's resources are allocated for claims and Insurers' fees, embodying shared risk among Participants. Once the pool's funds surpass the investment threshold, Insurers can invest to generate additional revenue. This excess income, beyond a set surplus trigger, is shared between Participants and Insurers, with any leftover reserves earmarked for future insurance periods.

The Qard Hasan Module is triggered when a deficiency occurs and the Insurance Pool reaches the Minimum funding level. the funds in the Qard Hasan Pool are used to cover the Insurance Pool's losses until its funding is restored to the Loan Maximum Point. That is, when $(R_{t-1} + nd)(1 - \rho^w) < S$, the Qard Hasan loan $Q = S - (R_{t-1} + nd)(1 - \rho^w)$ will be offered.where ρ^w is the agency fee rate and ρ^m is the profit sharing rate. If the body of Insurance Pool funds exceeds the surplus trigger at stage t stage, the reserve satisfies

$$R_t = h \wedge \left[((R_{t-1} + nd)(1 - \rho^w) - S)(1 - \rho^m)\right]_+ \tag{13}$$

Each participant receives benefits

$$G_t = \left[\left(\left(\frac{R_{t-1}}{n} + d\right)(1 - \rho^w) - \frac{S}{n}\right)(1 - \rho^m) - \frac{h}{n}\right]_+ \tag{14}$$

If we set the profit threshold point $h = 0$, the earnings for each participant in the model simplify to:

$$G_t = \left[\left(\frac{R_{t-1}}{n} + d\right)(1 - \rho^w) - \frac{S}{n}\right](1 - \rho^m) \tag{15}$$

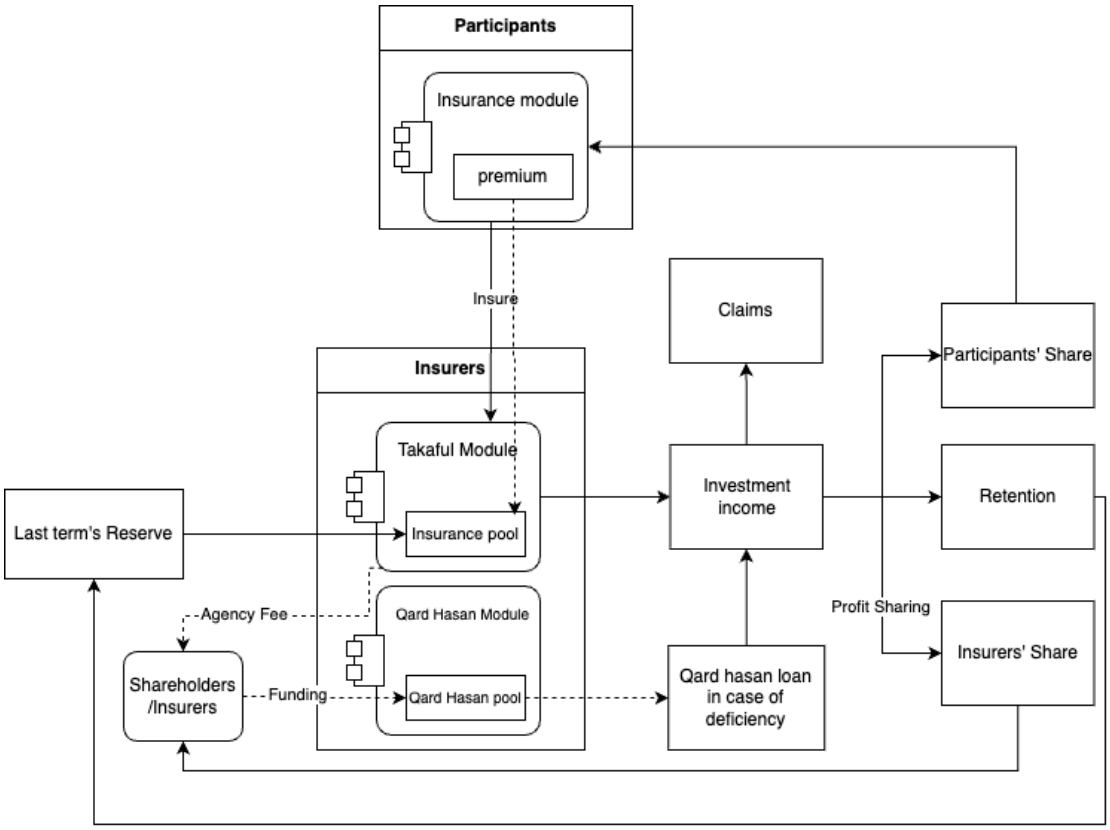

Fig. 9. Insurance model architecture

If the body of Insurance Pool funds does not reach the surplus trigger at stage t, no profit sharing will be calculated for the Insurers, at which point the reserve satisfies

$$R_t = \left[\left(\frac{R_{t-1} + nd}{1 - \rho^w} - S\right)\right]_+ \tag{16}$$

Since Yala Finance is a high-risk trading scenario, there could be a large demand for short-term insurance, Insurers could reasonably increase insurance rates for short-term traders, contract parameters would need to be changed at any time in conjunction with the market price provided by the prognosticator, and Insurers would have the right to reject high-risk trades.The Yala insurance model could also become a Defi The Yala insurance model could also be a standalone derivative in the BTC ecosystem, where policyholders and shareholders could provide capital for long-term gains.

7.4. Key external actors

7.4.1. Keepers

Keepers, often automated and independent entities, are motivated by arbitrage to enhance liquidity in decentralized systems. In Yala Finance, they help stabilize $YU's price by selling it when above the target and buying when below. During liquidations in Yala's Vault module, Keepers engage in auctions for surplus, debt, and collateral, contributing to the protocol's market efficiency.

7.4.2. Oracles

Yala Finance requires real-time information about the market prices of collateral assets in the Yala Finance Vault to determine when to trigger liquidation. Oracles provide data services for USD prices, collateral prices, and $YU prices to both the Yala Foundation, insurers, and the liquidation module [1, 2]. The Yala Foundation oversees the Oracle and OSM Modules, which source price inputs from the BTC Layer 2 Oracle node. This node fetches off-chain data, reformats it, and supplies it to the Oracle Module. To safeguard against attackers controlling prophecy machines, the OSM (Office of the Secretary) intermediates, receiving prices indirectly to enhance security. It introduces a 30-minute delay before releasing prices to the protocol, allowing time for emergency actions if needed. The Yala Foundation decides on this delay and emergency protocols.

Given the limited collateral types in early Yala Finance, the Oracle system doesn't require extensive price data. It operates on a publish-subscribe model, with the Oracle Module collecting and broadcasting price data at set intervals after OSM processing. Each node in the system runs a daemon to keep its price data current.

7.4.3. Insurers

Given the high volatility unique to Yala Finance, collateralized borrowers require insurance services. However, to preserve $YU stability, the Yala Foundation doesn't directly provide these services or assume borrower risks. Instead, when starting a Loan Module, borrowers can also activate an Insure Module tailored to that specific module.

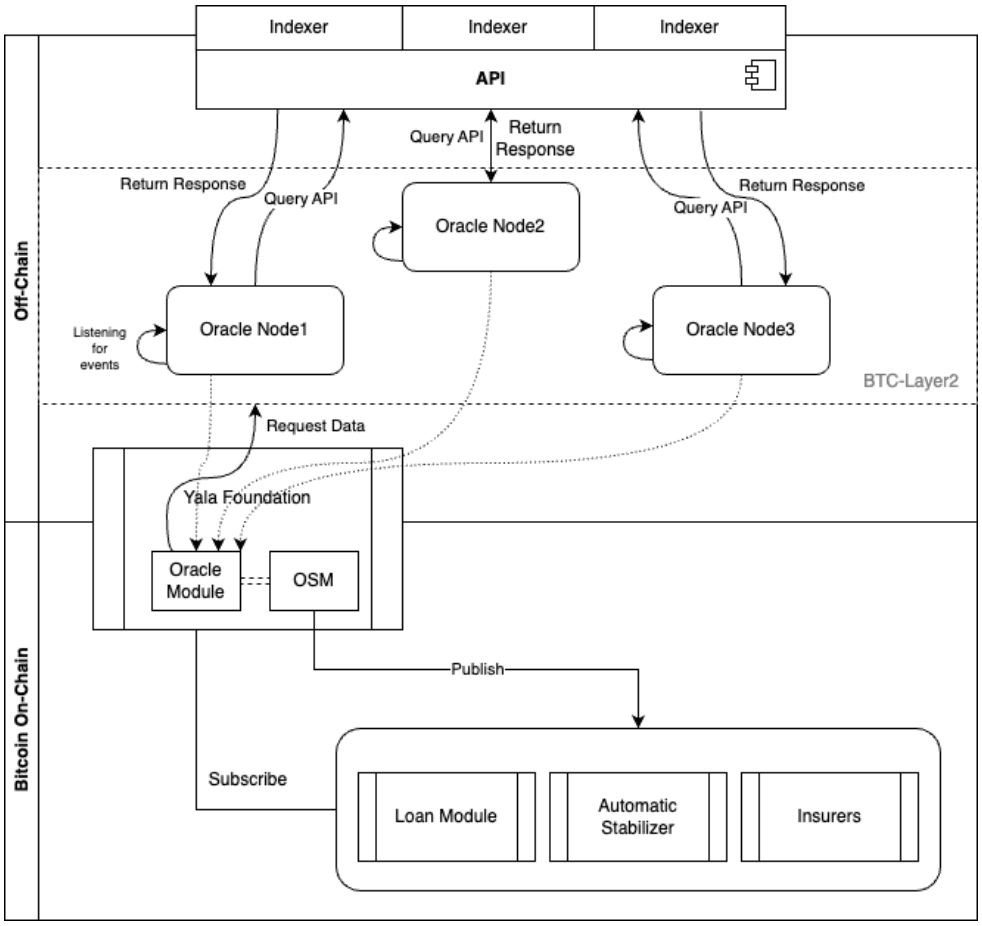

Fig. 10. Oracle service

```
{
  "p": "BRC-20-module",
  "op": "deploy",
  "name": "Yala_Insure",
  "source": "at_code_address",
  "init": {
        "module": "at_loan_module_address"
  }
}
```

Insurers are insurance operators responsible for maintaining the qard hasan fund pool, setting various parameters for insurance contracts, and publishing them on the insurance platform. Before initiating the Insure Module, borrowers need to select an insurance contract on the insurance platform and then direct the source to the script code segment to complete the publication of the Insure Module.

8. Interoperability

8.1. Secure state channels between modules

In Yala's architectural design, state changes operate in an independent environment, creating state channels running within UTXOs on BTC. These state channels provide security, with the occurring changes being off-chain, thus not burdening the BTC main chain's load. The execution function for state changes is stored in MAST conditional circuits scripts in the Witness on BTC. It simply evaluates if the conditions for state change are met without conducting substantial complex calculations, ensuring the contract's security, simplicity, and transparency.

Analyzing the size and content of BTC blocks, according to information provided by bitaps, although the block weight of BTC blocks is limited to 4MB, the average size of the block weight is only 1724885 bytes (Data on March 10, 2024) when the Base size is close to 1MB. Therefore, at least 2MB of space can be fully provided to the witness, with an average of approximately 3500 transactions per block. We can estimate that if each transaction uses taproot script-path spend scripts in the witness to place contracts, each contract averages 600 bytes.

BRC-20 Modules' conditional circuits prescribe the rules for state changes, necessitating MultiSig Vault. The interaction process broadly aligns with the MultiSig Vault setup, involving DeFi participants (excessive collateral borrowers or atomic swap participants), oracles, and Token Modules. The oracles are decentralized and considered trustworthy after processing by the Oracle Module's OSM. Token Modules are divided into Stablecoin Modules implemented by MAST and token contract modules on other chains. The Stablecoin Module determines the permission conditions allowing users to withdraw $YU, and participants also initiate transfers to it during settlement, with the transfer operation being freely executed without permission. In atomic swap scenarios, token contract modules on other chains have Turing completeness and the contract security of different chains. Token Modules are automatically executed under the consensus of Indexers and can be considered trustworthy. We believe such state channels are trustworthy under conditions where only the single transaction participant is untrustworthy.

When DeFi participants initiate the transaction module, its source will be the DeFi transaction script published by the Yala community. The transaction process forms a 2-of-3 multisig state channel between the participant, Oracle Module, and Token Module. The unlocking conditions are two: one is controlled by a time lock, and if participants do not return the borrowed assets within the specified time, the assets will be directly paid to the Token Module. MAST conditions control the other, setting the wager on whether participants can repay the loan. In the borrowing scenario, currently, only over-collateralized loans with pre-set interest values can be implemented, meaning the interest and loan duration are predefined from the start, not variable over time, due to the script size limitations currently supported by the Witness.

8.2. Atomic swap

We can readily observe the limitations of operating assets on the BTC main chain. Therefore, we can expand BTC by enhancing its interoperability to facilitate more complex DeFi transactions with BTC assets. Participants need to initiate an atomic swap module on the BTC main chain, pledge a certain amount of BTC or other assets, and then obtain an equivalent amount of wrapped tokens (yBTC) on another blockchain to engage in the DeFi ecosystem of different chains.

We refer to the atomic swap protocol proposed by TierNolan on Bitcointalk[2]:

1. Party 'A' generates some random data, x (the secret).
2. Party 'A' generates Tx1 (the payment) containing an output with the chain-trade script in it. See below for this script and a discussion of it. It allows coin release either by signing with the two keys (key 'A' and key 'B') or with (secret 'x', key 'B'). This transaction is not broadcast. The chain release script contains hashes, not the actual secrets themselves.
3. Party 'A' generates Tx2 (the contract), which spends Tx1 and has an output going back to key 'A'. It has a lock time in the future and the input has a sequence number of zero, so it can be replaced. 'A' signs Tx2 and sends it to 'B', who also signs it and sends it back.
4. 'A' broadcasts Tx1 and Tx2. Party 'B' can now see the coins but cannot spend them because it does not have an output going to him, and the transaction is not finalized anyway.
5. 'B' performs the same scheme in reverse on the alternative chain. The lock time for 'B' should be much smaller than the lock time for 'A'. Both sides of the trade are now pending but incomplete.
6. Since 'A' knows the secret, 'A' can claim his coins immediately. However, 'A', in the process of claiming his coin, reveals the secret 'x' to 'B', who then uses it to finish the other side of the trade with ('x', key 'B').

In the design of Yala's atomic swap protocol, we utilize the Modular BRC20. The protocol process begins with user A initiating an atomic swap contract white module targeted at a specific chain. The randomly generated number x and the lock time t_1 are written into the unlock script of the white module. There is a contract on the target chain that interacts with the user, defined as role B, which executes automatically. The unlock condition for the white module is [if (x for H(x) known and signed by B) or (signed by A & B),] if not successfully signed and unlocked, a refund transaction to the user will be initiated after time t_1. Then, user A initiates a transfer transaction to the white module, transferring a certain amount of BTC into it. The indexer captures the white module, triggering the execution of the atomic swap contract on the target chain, initiating an equivalent yBTC transfer transaction to the user's account on that chain with the unlock condition [if (x for H(x) known and signed by A) or (signed by A & B),] If not successfully unlocked within time t_2, the transaction will be canceled, where $t_2 < t_1$. User A signs the transaction on the target chain and uses and reveals the secret x, acquiring yBTC on the target chain. Contract B captures the secret x and simultaneously unlocks the white module. The final unlock of the white module requires consensus through the indexer; at this time, the ownership of the white module belongs to the Indexer network. The BTC assets pledged within the white module will only be used if a user from another chain exchanges yBTC back to BTC. The entire atomic swap process is monitored by the Oracle, and the contract on the other chain will decide whether to allow atomic swaps based on whether the number of pledged BTC counted by the Oracle is equal to the number of issued yBTC.

[2] Bitcointalk.org - Atomic Swap Protocol by TierNolan

9. Future

While Yala has considered implementing native DeFi solutions for BTC assets, there remain several challenges in ensuring interoperability between BTC and other blockchain networks. The initial approach of directly implementing atomic swaps and locks to facilitate two-way communication between BTC and other assets faces issues such as the lack of guaranteed counterparties, long cross-chain latency, value-blindness, and the risk of fraud.

The common approach adopted by most cross-chain bridges, involving multiple signatures from trusted organizations, could be considered as a potential solution. However, this raises significant security concerns, as it is well-known that many DeFi and cross-chain bridge projects have been subject to insider theft, where the custodians of the managed assets were responsible for the theft of funds. Decentralized asset management requires a higher level of security in cross-chain asset mapping from BTC to other blockchains.

Yala needs to address the problem of how to tolerate untrustworthy multi-signatory parties in decentralized asset management, as well as how to increase the speed of cross-chain transactions. One potential solution could be to decouple transaction confirmation from transaction execution and improve the efficiency of multisignatures, which could significantly increase the speed of cross-chain transactions. Additionally, it might be possible to prevent malicious behavior by ensuring that the benefits of an untrustworthy multi-signatory attacking the managed assets are outweighed by the costs.

Overall, Yala recognizes the need to develop robust and secure cross-chain interoperability solutions to enable the seamless integration of BTC assets with other blockchain ecosystems while ensuring a high level of decentralization and trust in the asset management process.

10. Conclusions

Yala Protocol is dedicated to building a Defi and programmability infrastructure based on BTC assets. We have designed a native asset interaction protocol based on the BTC mainchain and interoperability mechanisms from BTC to other blockchains. The native operability of BTC assets is implemented based on Modular BRC20, based on overcollateralized stabilization in-

scriptions, providing a full suite of Vaults module, liquidation algorithm, automatic stabilizer, and Yala Foundation services. Automatic stabilizer and Yala Foundation services. Due to the risk of BTC assets and the long time between BTC blocks, we added an insurance derivatives module, which not only reduces the risk of collateralized lending users but also allows users to use the insurance module to make interest-bearing profits. At the interoperability level, we use atomic exchange to complete the 2-way peg of BTC to other blockchains and utilize the Turing-complete smart contracts of other blockchains to realize the complex logic of operating BTC assets.

References

[1] Al-Breiki, H., Rehman, M. H. U., Salah, K., and Svetinovic, D. Trustworthy blockchain oracles: review, comparison, and open research challenges. *IEEE access 8* (2020), 85675–85685.

[2] Caldarelli, G. Overview of blockchain oracle research. *Future Internet 14*, 6 (2022), 175.

[3] Feng, R., Liu, M., and Zhang, N. A unified theory of decentralized insurance. Available at SSRN: `https://ssrn.com/abstract=4013729` or `http://dx.doi.org/10.2139/ssrn.4013729`.

[4] Kokorin, I., De Graaf, T., and Haentjens, M. The failed hopes of disintermediation: Crypto-custodian insolvency, legal risks and howto avoid them. *Singapore Journal of Legal Studies* (2020), 526–563.

[5] Liu, X., Ji, S., Wang, X., Liu, L., and Ren, Y. Blockchain data availability scheme with strong data privacy protection. *Information 14*, 2 (2023), 88.

[6] Michael Egorov, C. F. curve-stablecoin. `https://github.com/curvefi/curve-stablecoin/blob/master/doc/curve-stablecoin.pdf`, 2022. October 9.

[7] Mohan, V. Automated market makers and decentralized exchanges: a defi primer. *Financial Innovation 8*, 1 (2022), 20.

[8] Perez, D., Werner, S. M., Xu, J., and Livshits, B. Liquidations: Defi on a knife-edge. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25* (2021), Springer, pp. 457–476.

[9] Tas, E. N., Zindros, D., Yang, L., and Tse, D. Light clients for lazy blockchains. *arXiv preprint arXiv:2203.15968* (2022).

[10] Team, E. Eigenlayer: The restaking collective. *URL: https://docs. eigenlayer. xyz/overview/whitepaper* (2023).

[11] Uniswap. Uniswap v3 whitepaper, 2021.

[12] Wang, Q., and Yu, G. Understanding brc-20: Hope or hype. *Available at SSRN 4590451* (2023).