HyperService: Interoperability and Programmability Across Heterogeneous Blockchains

Make Web3.0 Connected!

Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, Yih-Chun Hu
Blockchain Proliferation

Payment Network

Smart Contract Platform

Total # of Projects Listed on CoinMarketCap

2.3K
“Make Blockchains Great”

Consensus Protocols

Blockchain X

Sharding & Layer-II Channels

Blockchain Y

Privacy & Program Analysis

Blockchain Z

Privacy
Security
Scalability
Decentralization
In a world deluged with isolated blockchains, interoperability is power.
Blockchain interoperability is complete only with programmability ...

**Passive Distributed Ledgers** → **Programmable State Machine**

```
pragma solidity 0.4.22;

contract Broker {
    uint constant public MAX_OWNER_COUNT = 50;
    uint constant public MAX_VALUE_PROPOSAL_COUNT = 5;

    // The authoritative output provided by this Broker contracts.
    uint public StrikePrice; StrikePrice = $10

    @public
    @payable
    def CashSettle(uint256, uint genericPrice: wei_value):
        assert self.remainingFund > MIN_STAKE
        assert self.optionBuyers[msg.sender].valid
        assert not self.optionBuyers[msg.sender].executed

        if genuinePrice > self.strikePrice:
            // Your code here...

    public
    function CashSettle(unit shareCount, uint genuinePrice) {
        optionAvailable
        require(optionBuyers[msg.sender].valid && !optionBuyers[msg.sender].executed
        // Only execute the option if it is profitable.
        if (genuinePrice > strikePrice) {
            // Your code here...
        }
```
Challenge I: A virtualization layer to abstract away heterogeneity

Cross-chain dApps: how to uniformly define operations among heterogeneous contracts and accounts …
Challenge II: Cryptography protocols to realize cross-chain dApps

Cross-chain dApps in the era of Web3.0

- Transactions on different Blockchains;
- Transactions in specific order;
- Downstream transactions depend on state resulted from upstream transactions;

dApp Executables

Contain more complex operations than just token transfers

How to realize transactions via decentralized protocols?

T3  T1  T2  T4
Our Proposal — HyperService

• A developer-facing programming framework
  - Universal State Model: a blockchain-neutral model to describe dApps
  - HyperService Language: a high-level language to program dApps

• A blockchain-facing cryptography protocol to realize dApps on-chain
  - Network Status Blockchain: a decentralized trust anchor
  - Insurance Smart Contract: a trust-free code arbitrator

A universal platform for developing and executing dApps across heterogeneous Blockchains
Programming Framework — Universal State Model

\[ \mathcal{M} = \{ \mathcal{E}, \mathcal{P}, \mathcal{C} \} = \{ \text{Entities, Operations, Constraints} \} \]

Entities: objects extracted from underlying blockchains

<table>
<thead>
<tr>
<th>Entities</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>address, balance, unit</td>
</tr>
<tr>
<td>contract</td>
<td>state variables[], interfaces[], source</td>
</tr>
</tbody>
</table>
Programming Framework — Universal State Model

\[ M = \{ E, P, C \} = \{ \text{Entities, Operations, Constraints} \} \]

Operations: computation performed over several entities

<table>
<thead>
<tr>
<th>Operations</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>payment</td>
<td>from, to, value, exchange rate</td>
</tr>
<tr>
<td>invocation</td>
<td>interface, parameters[], invoker</td>
</tr>
</tbody>
</table>

An example invocation operation:
Y::Option.CashSettle(10, X::Broker.StrikePrice)
Programming Framework — Universal State Model

\[ M = \{ E, P, C \} = \{ \text{Entities}, \text{Operations}, \text{Constraints} \} \]

- **Entities**: objects extracted from underlying blockchains
- **Operations**: computation performed over several entities
- **Constraints**: dependencies among operations

<table>
<thead>
<tr>
<th>Entities</th>
<th>Attributes</th>
<th>Operations</th>
<th>Attributes</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>address, balance, unit</td>
<td>payment</td>
<td>from, to, value, exchange rate</td>
<td>precondition</td>
</tr>
<tr>
<td>contract</td>
<td>state variables[], interfaces[], source</td>
<td>invocation</td>
<td>interface, parameters[], invoker</td>
<td>deadline</td>
</tr>
</tbody>
</table>
HyperService Language (HSL): A high-level programming language

**import**: include the source code of all contracts defined in the HSL program

**account & contract**: defining entities extracted from underlying blockchains

**payment & invocation**: defining operations among entities

**before, after & deadline**: defining dependencies among operations

---

```plaintext
1  # Import the source code of contracts written in different languages.
2  import ("broker.sol", "option.vy", "option.go")
3  # Entity definition.
4  # Attributes of a contract entity are implicit from its source code.
5  account a1 = ChainX::Account(0x7019..., 100, xcoin)
6  account a2 = ChainY::Account(0x47a1..., 0, ycoin)
7  account a3 = ChainZ::Account(0x61a2..., 50, zcoin)
8  contract c1 = ChainX::Broker(0xbba7...)
9  contract c2 = ChainY::Option(0x917f...)
10 contract c3 = ChainZ::Option(0xefed...)
11 # Operation definition.
12 op op1 invocation c1.GetStrikePrice() using a1
13 op op2 payment 50 xcoin from a1 to a2 with 1 xcoin as 0.5 ycoin
14 op op3 invocation c2.CashSettle(10, c1.StrikePrice) using a2
15 op op4 invocation c3.CashSettle(5, c1.StrikePrice) using a3
16 # Dependency definition.
17 op1 before op2, op4; op3 after op2
18 op1 deadline 10 blocks; op2, op3 deadline default; op4 deadline 20 mins
```

Figure 2: A cross-chain Option dApp written in HSL.
Programming Framework Core -- HSL Program Compilation

Extract state variables and interfaces from imported contracts

Unify different contract languages

• Compatibility: type check
• Verifiability: state variables
• Feasibility: no dep-loop

Unified Type | Solidity | Vyper | Go
---|---|---|---
Boolean | bool | bool | bool
Numeric | int, unit | int128, decimal, ... | int, float, ...
Array | array, bytes | array, bytes | array, slice

Figure 3: Workflow of HSL Compilation.
Transaction Dependency Graph (TDG) — HSL Program Executables

- Each vertex defines:
  - Full information for computing a blockchain-executable transaction
  - Metadata to ensure correct execution
- Edges define the transaction order

- Resulting state of T1 is used subsequently
- A state proof needs to be collected after T1 is finalized.

Transaction T1 on ChainX:
- from: a1.address
- to: c1.address
- Meta:
  - data: c1.getStrikePrice
  - <amt, dst>: <0.1 ncoin, 0x1...>
  - state_proof: collect from NSB

Transaction T2 on ChainX:
- from: a1.address
- to: VES.relayX.address
- Meta:
  - value: 50 xcoin
  - <amt, dst>: <25 ncoin, 0x2...>
  - deadline: 4 NSB blocks

Transaction T3 on ChainY:
- from: VES.relayY.address
- to: a2.address
- Meta:
  - value: 25 ycoin
  - <amt, dst>: <5 ncoin, 0x3...>
  - deadline: 6 NSB blocks

Transaction T4 on ChainY:
- from: a2.address
- to: c2.address
- Meta:
  - data: c2.CashSettle(10, c1.StrikePrice)
  - <amt, dst>: <0.1 ncoin, 0x4...>
  - value_proof: T1.meta.state_proof

Transaction T5 on ChainZ
HyperService Architecture

Developer-facing Programming Framework

Universal Inter-Blockchain Protocol
Universal Inter-Blockchain Protocol (UIP) Overview

- A protocol spoken by all parties to co-execute cross-chain dApps
- Fully decentralized: no authorities and no mutual trust among parties

- Provable security properties
  - Correctness assurance, financial atomicity, and accountability

- Network Status Blockchain: a decentralized trust anchor
- Insurance Smart Contract: a trust-free code arbitrator
UIP Security Properties

Security properties of dApps executed by UIP  
(Proved in UC-Framework)

TDG is realized as desired

- dApp execution either finishes correctly or being financially reverted

Accountability

- Regardless of at which stage the execution fails, the misbehaved parties are held accountable for the failure

Correctness Guarantee

- If blockchains are modeled with bounded transaction finality latency, dApps are guaranteed to finish correctly if all parties are honest

Financial Atomicity
NSB: Provide unified and objective views on the status of dApp executions

- Consolidate transactions and state from underlying blockchains
- Provide unified representations for transaction status and state in form of verifiable Merkle proofs

Proof of Actions (PoAs): allow parties to construct proofs to certify their actions taken during executions

Figure 5: The architecture of NSB blocks.
Insurance Smart Contract (ISC)

Merkle Proofs

if CorrectExecution:
    Pay service fee
else:
    Revert effective fund
    Enforce accountability

Decision Logic
Implementation and Source Code Release

• Incorporate Ethereum and a permissioned blockchain built on Tendermint
  - Different consensus efficiency and transaction finality definition
  - Different contract languages: Solidity VS. Go

• Three categories of cross-chain dApps
  - Financial derivative, asset movement and federated computing

• Released source code: https://github.com/HyperService-Consortium

35K Lines of Code
Demo: End-to-end executions on HyperService

1. Invoke E::Broker.ComputeStrikePrice()
2. Invoke T::Option.cash_settle(E::Broker.StrikePrice)
3. Invoke E::Option.CashSettle(E::Broker.StrikePrice)
The execution of each transaction goes through multiple stages and HSL complication and TDG generation.
HyperService: A universal platform for developing and executing dApps across heterogenous Blockchains

Q & A
Thank You
hyperservice.team@gmail.com