A Blockchain based Witness Model for Trustworthy Cloud Service Level Agreement Enforcement

Huan Zhou, Xue Ouyang, Zhijie Ren, Jinshu Su, Cees de Laat, Zhiming Zhao
Outline

• Cloud SLA / Blockchain: background and challenges
• Witness Model Design and Key Techniques to Ensure Trustworthiness
  • Witness Management and Unbiased Random Selection
  • Payoff Function Design and Nash Equilibrium
  • Witness Audit Mechanism
• Experimental Study
• Conclusions
What is Cloud SLA?

Cloud SLA (Service Level Agreement) is a business concept which defines the contractual and financial agreements between the Cloud customer and provider.
What is Cloud SLA?

Cloud SLA (Service Level Agreement) is a business concept which defines the contractual and financial agreements between the Cloud customer and provider.

Example:
A Cloud customer, $C$, buys a VM (Virtual Machine), $X$, from an IaaS Cloud provider, $P$, for one hour. They make an agreement: in this one hour,
- If the VM, $X$, does not crash, $C \rightarrow P$ 1000 credits. (payment)
- If the VM, $X$, crashes, $C \rightarrow P$ 500 credits. (compensation)
Cloud SLA (Service Level Agreement) is a business concept which defines the contractual and financial agreements between the Cloud customer and provider.

Example:
A cloud customer, C, buys a VM (Virtual Machine), X, from an IaaS Cloud provider, P, for one hour. They make an agreement:
- If the VM, X, does not crash, C → P 1000 credits. (payment)
- If the VM, X, crashes, C → P 500 credits. (compensation)

Provider is in a centralized and dominating position:
- Less fair;
- Lack of violation proof;
- Manual enforcement.
What is Cloud SLA?

Cloud SLA (Service Level Agreement) is a **business** concept which defines the contractual and financial agreements between the Cloud **customer** and **provider**.

1. **How to ensure** the fairness of the provider and customer in the agreement?
2. **Who and how to detect** and **prove** the SLA violation?
3. **How to automate** the process of payment and especially the compensation?

- If the VM, $X$, crashes, $C \rightarrow P$ 500 credits. (compensation)
Blockchain: decentralized and immutable ledger

- Blockchain is a technique, which makes every participant having consensus on a decentralized ledger, e.g., through PoW (Proof of Work).
- Bitcoin is the first generation application of blockchain, from 2009.
Blockchain: smart contract

- Ethereum is the second generation blockchain, from 2015.
- It proposes EVM (Ethereum Virtual Machine), which is a set of byte values to represent a virtual machine state.
- Ethereum works as a world-wide computer. The program running on this computer is named as Smart Contract.
A Cloud customer, $C$, buys a VM (Virtual Machine), $X$, from an IaaS Cloud provider, $P$, for one hour. They make an agreement: in this one hour,

- If the VM, $X$, does not crash, $C \rightarrow P$ 1000 credits. (payment)

- If the VM, $X$, crashes, $C \rightarrow P$ 500 credits. (compensation)

Customer invokes this Payment interface

Smart Contract CODE:

```java
Payment()
{
    if( !X.violated )
        C.transfer($P$, 1000)
    else
        C.transfer($P$, 500)
}
```
Challenges: violation detection?

In the context of Cloud SLA, **who** can be the **judge** to convince both, the provider and customer, that the service violation really happens? **How**?
Current solution: oracle

**Off-chain** Events ➔ **On-chain** Transactions

**Who?**
**Oracle**: Perform as “Data Carrier”

Is it trustworthy?

Third trusted party

Software Oracle ➔ Hardware Oracle

- Must trust the third party;
- Single point of failure;

Distributed oracles

For oracles:
- Require them independent and trustworthy;
- No incentive;
- Consensus issue;

Third trusted party

Orisi
Our proposal: 

decentralized witness model
Our proposal: decentralized witness model

Cloud Service Customer

Witness

$w_1$

$w_2$

$w_N$

SLA

Smart Contract

Cloud Service Provider

$M$-out-of-$N$ reports? violated?
Our proposal: decentralized witness model

Witness

Cloud Service Customer

Smart Contract

Cloud Service Provider

\( w_1 \)

\( w_2 \)

... 

\( w_N \)

rewards come from the deposit of provider and customer
How does it work in Cloud SLA

1. Off-chain negotiation
2a. Provider provision
2b. Publish Service Detail and Setup SLA
3a. Test and adopt
3b. Accept SLA
4a. Monitor
4b. Report violation
5. Enforce the corresponding fees

Cloud Provider

Cloud Customer

Decentralized Witnesses

SLA

Smart Contract

Cloud Service

Witness Rewards

Compensation Fee

On-chain interaction

Off-chain interaction

(possible happening)
How does it work in Cloud SLA

How to **motivate** the witness to tell the **truth** about the service violation detection?

How the witnesses are **managed** and **selected** to be independent?
Decentralized Witnesses Pool

Any Blockchain User register to the Witness-Pool Smart Contract.

Some deposits for resisting Sybil attack

Witnesses Pool

- \( U_1 \):
  - ID: 0x9a6baf8cb84cc3614f544fbb8c15e89e5a9311f2
  - State: Online/Offline/…

- \( U_2 \):
  - ID: 0x2e5727a1ae83f0c885e62b62b5561a1456b4bb65
  - State: Online/Offline/…

- \( U_T \):
  - ID: 0x4cee3a18a79ee7ce25f35bb7a8606e3a2131fd82
  - State: Online/Offline/…
Unbiased Random Selection Procedure

Witness-Pool Smart Contract

Provider/Customer invokes

\[ seed = H_{i+1} + H_{i+2} + ... + H_{i+j+1} \]

FOR \( x = seed \% T + 1 \)
check \( U_x \) reputation
check \( U_x \) state (online?)

\[ seed = \text{Hash} (seed) \]

END FOR
RETURN \( N \) selected witnesses

Witness Committee \((W)\)
\( N > 2 \)

Witnesses Pool

Underlying Blockchain

Request

Wait for new \( j \) blocks generated

Selection
Witness-as-a-Game
Strategic Form Game with Complete Information

Players: \( w_1 \), \( w_2 \), \( \ldots \), \( w_N \)

Actions:
- \( \sigma_k^{(r)} \): Report the service violation to the smart contract
- \( \sigma_k^{(s)} \): Do not report and keep silence to the smart contract

Payoff function \( \pi_k (\sigma_k, \sigma_{-k}) \): Rewards of \( w_k \) in this strategy profile

Cloud Service Customer

Witness Committee \( (W) \)

Cloud Service Provider

SLA

M-out-of-N reports? \( (1 < N/2 < M < N) \)

violated?
Payoff: witness incentive model

Witness Committee ($W$)

$w_1$ $w_2$ $...$ $w_N$

$M$-out-of-$N$ reports? (1 < $N/2 < M < N$)

Cloud Service Customer

Smart Contract

Cloud Service Provider

$\forall w_k \in W_{\text{report}}, \pi_k (\sigma_k^{(r)}, \sigma_{-k}) = 10$

$\forall w_k \in W_{\text{silence}}, \pi_k (\sigma_k^{(s)}, \sigma_{-k}) = 0$

Payoff function

$W_{\text{report}} : \forall w_k \in W_{\text{report}}, \sigma_k = \sigma_k^{(r)}$

$W_{\text{silence}} : \forall w_k \in W_{\text{silence}}, \sigma_k = \sigma_k^{(s)}$

not violated

$\forall w_k \in W_{\text{report}}, \pi_k (\sigma_k^{(r)}, \sigma_{-k}) = -1$

$\forall w_k \in W_{\text{silence}}, \pi_k (\sigma_k^{(s)}, \sigma_{-k}) = 1$
Proof: Nash Equilibrium

In the witness game, there are two and only two Nash equilibrium points:

- $\forall w_k \in W, \sigma_k = \sigma_k^{(r)}$
- $\forall w_k \in W, \sigma_k = \sigma_k^{(s)}$

Take the example of three-witness game $(N = 3, M = 2)$

<table>
<thead>
<tr>
<th>$w_1$ : Alice</th>
<th>$w_3$ : Candy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_1^{(r)}$ : Report</td>
<td>(10, 10, 10)</td>
</tr>
<tr>
<td>$\sigma_1^{(s)}$ : Silence</td>
<td>(0, 10, 10)</td>
</tr>
<tr>
<td>$w_2$ : Bob</td>
<td></td>
</tr>
<tr>
<td>$\sigma_2^{(r)}$ : Report</td>
<td>(10, 0, 10)</td>
</tr>
<tr>
<td>$\sigma_2^{(s)}$ : Silence</td>
<td>(1, 1, -1)</td>
</tr>
<tr>
<td>$w_2$ : Bob</td>
<td></td>
</tr>
<tr>
<td>$\sigma_3^{(r)}$ : Report</td>
<td>(10, 10, 0)</td>
</tr>
<tr>
<td>$\sigma_3^{(s)}$ : Silence</td>
<td>(1, -1, 1)</td>
</tr>
</tbody>
</table>

Key Techniques to Ensure Trustworthiness
Reputation: Witness Audit

- All behaviors of a witness are recorded in the blockchain, which are trackable and immutable.
- The audit mechanism is leveraged to calculate the reputation value of a witness.
- If the reputation of a witness is too low, it would not be selected by the selection algorithm.
- The malicious or unrational witnesses can be: 

  *Lazy witness* → someone prefers *not to report* the violation.
Implementation: Ethereum

- The entire system is implemented based on the **two** types of smart contracts
- Leverage **Solidity** to program smart contracts
- Code: [https://github.com/zh9314/SmartContract4SLA](https://github.com/zh9314/SmartContract4SLA)
• We deploy the implemented smart contracts on the test net of Ethereum blockchain, Rinkeby.
• We test all possible scenarios to exploit and validate the functionality of different interfaces.
• The gas consumption of an interface determines the transaction fee needed to pay the miner in Ethereum, when invoking that interface.
• The more complex of the interface is, the more transaction fee required when it is invoked.
Conclusions

1. A decentralized witness model is proposed for Cloud SLA enforcement;

2. Witness-as-a-Game for incentive: in order to maximize the rewards, a witness always has to offer honest monitoring service;

3. The trustworthiness is proved through game theory;

4. A prototype system is fully implemented based on Ethereum blockchain.
Future Work

• **Break the limitation** of this work: the witness can only provide Boolean value, “TRUE” or “FALSE”;

• Further **optimize** the interface implementation to reduce the gas consumption;

• Consider some **more** application **scenarios**, not only Cloud SLA;

• Develop **user-friendly tools** to interact with the smart contract.
To the best of our knowledge, this is the first work of trustworthy decentralized oracles based on economic principles (game theory).

Thank you!

Questions?
Backup Slides

Overall Relationship

Provider

Witness-Pool

Smart Contract

Customer

Witness

SLA Smart Contract

1a) \textit{X} \rightarrow \textit{SLA}::\texttt{genSLAContract}
return: \texttt{SLA\_address}

2a) \textit{P} \rightarrow \textit{SLA}::\texttt{requestSortition} +
\textit{P} \rightarrow \textit{SLA}::\texttt{sortitionFromWP}

4) \textit{P} \rightarrow \textit{SLA}::\texttt{publishService} + \textit{P} \rightarrow \textit{SLA}::\texttt{setupSLA}

3b) \textit{SC} \rightarrow \textit{WP}::\texttt{confirm}
\textit{X} \rightarrow \textit{WP}::\texttt{register}
\textit{W} \rightarrow \textit{WP}::\texttt{turnOn}
\textit{W} \rightarrow \textit{WP}::\texttt{turnOff}
\textit{W} \rightarrow \textit{WP}::\texttt{reject}
\textit{W} \rightarrow \textit{WP}::\texttt{reverse}
\textit{X} \rightarrow \textit{WP}::\texttt{checkWState}

5) \textit{C} \rightarrow \textit{SLA}::\texttt{acceptSLA}

6) \textit{W} \rightarrow \textit{SLA}::\texttt{witnessConfirm}
Witness-pool Smart Contract Implementation

*Witness state* transition →

- **Online**
  - $P \rightarrow \text{SLA::resetSLA}$
  - $W \rightarrow \text{SLA::witnessRelease}$
  - $\text{SC} \rightarrow \text{WP::release}$

- **Offline**
  - $W \rightarrow \text{WP::register}$

- **Candidate**
  - $W \rightarrow \text{WP::turnOn}$
  - $W \rightarrow \text{WP::reverse}$ (reputation↓)

- **Busy**
  - $W \rightarrow \text{WP::reject}$
  - $W \rightarrow \text{SLA::witnessConfirm}$

Confirmation Time Window?

$W \rightarrow \text{WP::confirm}$
A Specific SLA Smart Contract Implementation

SLA state transition →

- **Init**: Accept Time Window?
  - yes: **Customer**: acceptSLA → **Active**
  - no: **Customer**: resetWitness → **Init**
- **Active**: Service Duration End?
  - yes: **Customer**: resetSLA → **Completed**
  - no: **Provider**: providerEndNSLAandWithdraw → **Completed**
- **Completed**: **Customer**: resetWitness → **Init**
- **Fresh**: **Customer**: setCustomer → **Init**
- **Violated**: **Customer**: resetWitness → **Completed**
- **Witness**: reportViolation
  - Confirmed?: yes → **Customer**: reportViolation → **Completed**
  - no → **Customer**: resetWitness → **Completed**
- **Provider**: providerWithdraw → **Completed**