III. The Witness Model using Smart Contract

In this section, the related roles in the proposed model design are introduced, especially the witness role. Then, the overall system architecture for SLA enforcement with smart contract on blockchain is illustrated, followed by a detailed description of the major responsibility of witnesses: service violation detection and reporting.

A. The Witness Role and Assumptions

There are mainly two roles in the traditional cloud SLA lifecycle. One is the cloud provider, P, which offers cloud service. The other is the cloud customer, C, which consumes the cloud service and pay the service fee. To demonstrate the key contribution of our work, we take a basic example to formulate our problem as follows.

A cloud provider, p, is an IaaS (Infrastructure-as-a-Service) provider. It provisions VMs (Virtual Machine) on demand with public addresses for its customers to use. For instance, according to the request of a customer c, provider p provisions a VM with a public IP address, IP_{pub}. During the service time, $T_{service}$, the customer, only the customer c is able to SSH and login to the VM through the corresponding address IP_{pub}. In this case, the SLA can be that the provider p claims that during the service time the provisioned VM will always be accessible. If this is true, the customer c must pay the service fee, $F_{service}$, to the provider p after the ending of the service. Otherwise, the customer c can acquire a compensation fee, $F_{compensation}$. That is the customer c only needs to pay $F_{service} - F_{compensation}$ to the provider p in the end, where we assume that $F_{service} > F_{compensation}$. For the latter case, if it happens, we define it as a SLA violation event. In addition, it is worth to mention that we should exclude the case that the inaccessibility is caused by the customer's own network problem, to be a violation event.

With only these two roles in the agreement, it is hard to ensure that the provider can get paid or the customer can get compensation paid back, if the service fee is prepaid. Hence, we leverage blockchain to play as the trusted party to afford a platform for these two roles and enforce these monetary transmissions. But it is still especially difficult to convince both roles whether the violation happens and whether it is caused by the customer's own network problem. We therefore bring in another new role in the traditional SLA lifecycle, named as witness role, W. They are also the normal participants in the blockchain and volunteers to take part in our SLA system to gain their own revenue through offering monitoring service. In order to solve the trust issue, a set of N witnesses, $\{w_1, w_2, ..., w_N\}$, is selected to form a committee in a specific SLA lifecycle. They together report the violation event and may obtain witness fee, $F_{witness}$, as rewards from both the provider and the customer. Moreover, the wallet address of a specific role on the blockchain is denoted by function, $address()$. For instance, $address(w_k)$ is the wallet address of witness w_k.

In this paper, we make the basic assumption on the witness role that it is always selfish and aims at maximizing its own revenue.

B. Overall System Architecture

Figure 1 illustrates the system architecture we design for cloud SLA enforcement. It consists of two types of smart contracts on the blockchain. One is the smart contract of witness pool, which is the fundamental smart contract of the system. The other type is the smart contract for a specific SLA enforcement. The responsibilities of

witness pool smart contract include witness management, specific SLA contracts generation and witness committee sortition. Any user of the blockchain, who has a wallet address, can register its wallet address in the witness pool to be a member of witnesses. They can keep themselves online and wait to be selected for some specific SLA contract. The incentive for the witness to participant in this system is to obtain revenue. And the more witness participants in the system, the more reliable and trustful the system would be.

The entire SLA lifecycle then becomes as follows. Certain provider p first leverages the smart contract of witness pool to generate a SLA smart contract for itself. Prior to setting up SLA, the customer c should negotiate with the provider p about the detailed SLA terms, including $T_{service}$, $F_{service}$, $F_{compensation}$, etc. Thereinto, one of the most important terms is to determine N, which is the number of witnesses would be hired for enforcing this SLA. The more witnesses involved in a SLA, the more trustworthy the violation detection results are. On the other hand, however, the more witness fee would be paid and both the customer and the provider need to afford this fee equally. According to this negotiation, the provider is able to reset these parameters of the generated SLA contract. Afterwards, a set of N witness members can be selected to form a witness committee through the sortition algorithm in Section IV-A, which is implemented by the witness pool smart contract on the blockchain. We design the algorithm to be random and being able to convince both, c and p, that most ones in the witness committee are independent and would not belong to the adverse role. After dynamically selecting the witness committee

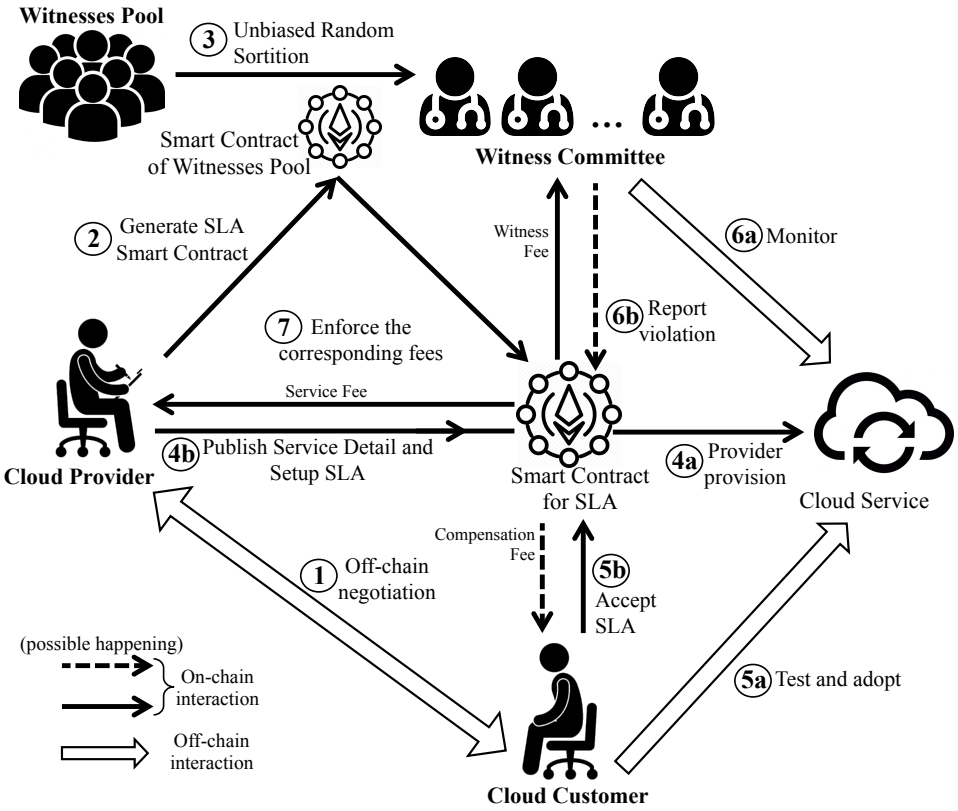

Fig. 1: System Architecture for Cloud SLA Enforcement

member, only the candidate witness can join the specific SLA contract. Meanwhile, the provider provision its cloud service for the customer to use and is able to publish its service detail in the SLA contract. The witnesses from the committee therefore start to monitor the service. In the case of our problem assumption in Section III-A, the provider p provisions a VM on demand and notify the public address IP_{pub} to all the committee members and customer c through the service detail field of the SLA contract. Therefore, the customer is able to use the VM and each witness starts to "ping" the address IP_{pub} constantly. If the violation happens during the service time, i.e. the address IP_{pub} is not accessible, the witness can report this event immediately. Section III-C describes how the violation state can be finally determined through these witnesses' reports. If the violation event is approved, then the customer can get back its compensation fee. All the dash lines in Figure 1 mean it may happen according to the actual event. Anyhow, the provider and witnesses from the committee are able to get corresponding fees.

C. Service Violation Detection and Reporting

The key role in our SLA enforcement system is the witness. It takes the responsibility of monitoring the service and determining whether there is a violation. In this section, we present our smart contract model in a specific SLA lifecycle to demonstrate key functionalities of a witness: service violation detection and reporting.

The sequential diagram in Figure 2 shows how different roles interact with the smart contract especially involving the witness in our model. After witnesses being selected, the entire lifecycle of a specific SLA begins. The provider p provisions the cloud service and deploy the smart contract on the blockchain. In order to setup a SLA, p must prepay the corresponding fee $PF_{prepaid}$ to the smart contract first. The amount of $PF_{prepaid}$ is determined by the

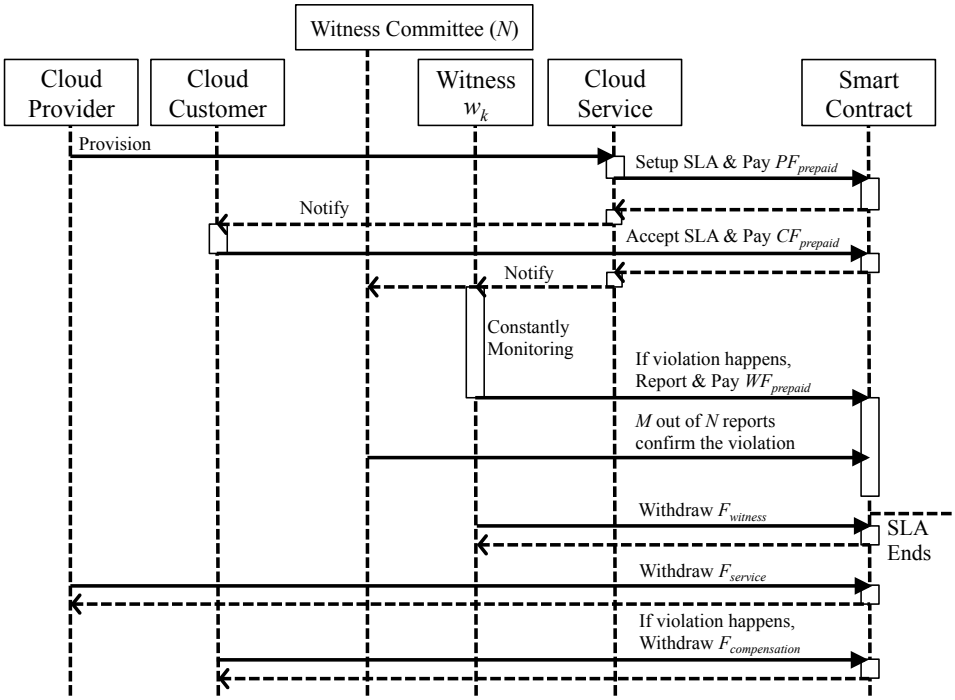

Fig. 2: Sequential Diagram of Different Roles in the Witness Model

half of the maximum witness fee. The customer c is then notified about the service and the content of the smart contract. As all the smart contract on the blockchain is public, the customer can verify the contract and the service status to decide whether to accept the SLA in a certain time window. In order to accept the SLA, the customer also needs to transfer the prepaid fee, $CF_{prepaid}$. It includes the service fee, $F_{service}$ and the other half of the maximum witness fee. As we assume $F_{service} > F_{compensation}$, the compensation fee would be directly deducted from this part of prepaid fee, if the violation happens. Afterwards, every witness in the committee is notified to start monitoring the service continuously.

During the service time, the witness can decide whether to report the event to the smart contract, if there is a service violation, for instance, the VM is not accessible. We design the rule that the witness w_k also needs to transfer a small amount of fee, $WF_{prepaid}$, to endorse its report at the same time. The incentive persuading w_k to report the event is that it would gain relative more revenue as witness fee, if the violation event is finally confirmed by the smart contract. On the contrary, if the violation is not confirmed, w_k would not get back the prepaid endorsement fee, $WF_{prepaid}$, as a penalty. This prevents w_k reporting fake violations just for maximizing its revenue. In addition, the violation is finally confirmed by the smart contract as the explanation below.

Since the first violation report, the smart contract would start counting a time window, T_{report}. Within this time window, the smart contract accept reports from other witnesses. When the time window T_{report} is over, the violation is automatically confirmed, if there are no less than M out of N reports from the witness committee received by the smart contract. M is also negotiated by p and c. It is then defined in the smart contract. Of course, M must be bigger than the half of N. Furthermore, the bigger the M is, the more trustworthy the violation is. For example, if there are $N = 3$ witnesses in the committee, the service violation can only be confirmed when at least $M = 2$ of them report the event. Here, it is worth to mention that the smart contract is designed to receive the report only from the committee member. And the second report from the same witness is refused within the same report time window. In some sense, these N independent witnesses constitute a $n-player$ game, in which each witness would like to maximize its revenue. We specially design the payoff function, shown in Section IV-B, and leverage the Nash Equilibrium of Game Theory to prove that the witness have to be a honest player in this game. That is they have to report the violation according to the real event.

Finally, the SLA ends at two cases. One case is the service time $T_{service}$ is over and there is no violation. The other case is that the SLA is violated. According to these different cases, the three roles are able to withdraw corresponding fees from the smart contract. This is explained in detail in Section IV-B.

IV. Key Techniques

In this section, we describe key techniques adopted in our witness model in detail. This model enables the automatic detection on the SLA violation, the results of which can convince both sides: the provider and customer. First, the unbiased random sortition algorithm is leveraged to guarantee that most of the witnesses selected into the committee are random and independent. It is also important to make both sides achieve consensus that most of the selected witnesses would not delegate the opponent's benefit. Based on this, we give the payoff function for the witness model in Section III-C. And also through the Nash Equilibrium theory, we prove that the "player" from

the witness committee have to behave honestly and tell the truth to maximize its revenue. Furthermore, we analyze some possible fraudulent behaviors from a malicious witness and demonstrate it can be audited through its action history.

A. Unbiased Random Sortition

It is crucial in the witness model that the witness sortition for a specific SLA contract is unbiased, i.e., neither the provider nor the customer can have advantage in the committee selection. This can be solved as a "fair item division problem" [24]. However, since Ethereum is already introduced as a trusted party in our model, we propose a simpler random sortition algorithm for committee selection, which is also implemented in the smart contract. We briefly describe it as follows.

- Firstly, there is a basic smart contract to manage the witness pool. It affords a set of interfaces for any blockchain user to register into the pool. Moreover, the registered witness is able to turn its state "Online" or "Offline", in order to indicate when it can be selected. The detailed witness state management is shown in Section V-B. a set of interfaces registration allows nodes to register as a witness and to be added in the witness pool. The addresses of the witness pool is managed as a list in the registration order.
- There are two interfaces designed in the smart contract to select N witnesses from the pool. The "request" interface is firstly called by a specific SLA contract at block B_b. It means this transaction is involved in the b^{th} index of block. The hash value of this block is B_b^{hash}. After K blocks generated by the blockchain, the other interface "sortition" can be invoked to select N online witnesses as candidates. The sortition algorithm is shown in Algorithm 1.
- It takes the hash values of the former K_s out of K blocks mentioned above as a seed. In addition, we need to wait other K_c blocks to confirm the adopted former ones, where $K_s + K_c = K$.
  - Here, K_s should be chosen such that the probability of some parties sequentially generating K_s blocks is very small.
  - K_c needs to be chosen such that the candidate blocks before are finally involved in the main chain with a dominant probability.
  - These two values depend on the blockchain's own properties. Considering the main net of Ethereum, [25] shows that the top four miners control 61% of the mining power. Thus, we recommend $K_s = 10$ so that with more than 99% probability that the seed is not manipulatable and predictable even if the top four miners collude. On the other hand, it is commonly believed that K_c should be 12 [26]. Considering the average block time of Ethereum is 15 seconds, the duration between the interfaces "request" and "sortition" is less than 6 minutes.
- Only the "Online" witness with positive reputation can be selected by the seed from the list of witness address pool. Anyhow, a new seed is generated based on the hash value of the previous seed. This process is repeated until required N witnesses are selected. In the beginning of Algorithm 1, we first check that there are at least 10 times of available witnesses than required N. It ensure the output of the algorithm as well as randomness.

Considering the difficulty itself of generating hash value for a block and combining the sequential K_s blocks as seed, we can prove that the sortition algorithm is random and unbiased, i.e., neither the provider nor the customer

Algorithm 1 Unbiased Random Sortition

Input:

Registered witness set, RW, a list of addresses;

The the size of the list, $len(RW)$;

The number of online witnesses, oc;

Required number, N, of members in witness committee;

The hash value, B_b^{hash}, of the b^{th} block B_b at request;

Following sequential, K_s, blocks;

The addresses of the provider, $address(p)$;

The addresses of the customer, $address(c)$

Output:

Selected witness set, SW, to form a committee.

assert($oc >= 10 * N$)
seed $\leftarrow 0$
for all $i = 1$; $i <= K_s$; $i++$ **do**
  seed$+= B_{b+1+i}^{hash}$
end for
$SW \leftarrow \emptyset$
$j \leftarrow 0$
while $j < N$ **do**
  $index \leftarrow$ seed$\%len(RW)$
  if $state(RW[index]) == Online$
  && $reputation(RW[index]) > 0$
  && $RW[index]! = address(c)$
  && $RW[index]! = address(p)$ **then**
    $state(RW[index]) \leftarrow Candidate$
    $oc--$
    Add $RW[index] \Rightarrow SW$
    $j++$
  end if
  seed $\leftarrow hash($seed$)$
end while
return SW

can take advantage in the committee with the assumption of trusting the security of Ethereum.

Here, we emphasis on the unbiasedness of the sortition algorithm. However, the unbiasedness before or after the sortition is not yet guaranteed. First, the witness pool is not Sybil-attack-proof. This can be solved by a reputation algorithm given in Section V-B. Second, there is no protection against the corruption after the committee candidates are determined. The provider or customer can DDOS the unwanted candidates to prevent them from joining the committee and/or bribe them when they are in. However, we argue that this is not easy as the addresses in Ethereum is not linked to their real identities or IP addresses. Another possible solution is to use verifiable random function [27] and a similar scheme used in [28] so that their identities remain hidden until they report in SLA. Due to the limitation in space, we leave this problem open for future research.

B. Payoff Function and Nash Equilibrium

Game theory targets to mathematically predict and capture behavior in strategic situation, where each player's revenue in making a decision depends on the strategies of others. There is currently a wide range of applications, including economics, evolutionary biology, computer science, political science and philosophy [29].

The strategic or matrix form, of a $n\text{-}player$ game, is the most common representation of strategic interactions in game theory [30]. The definition consists of a set of players, a set of strategy profiles and a design of payoff functions. Based on general game theory definition, we define our witness game as follow.

Definition 1. Witness Game: It is a $n\text{-}player$ game represented as a triple (SW, Σ, Π), where

- $SW = \{w_1, w_2, ..., w_n\}$ is a set of n players. Each player is a witness and they form the witness committee.
- $\Sigma = \Sigma_1 \times \Sigma_2 \times ... \times \Sigma_n$ is a set of strategy profiles, where Σ_k is a set of actions for the witness w_k. A strategy profile is therefore a vector, $\sigma = (\sigma_1, \sigma_2, ..., \sigma_n)$, where σ_k is a possible action of Σ_k, $(k = 1, 2, .., n)$.
- $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$ is a set of payoff functions, where $\pi_k : \Pi \rightarrow R$ is the payoff function determining the revenue for witness w_k, $(k = 1, 2, .., n)$. R is the corresponding revenue.

Actually, there are only two actions for each witness in this game, which is $\Sigma_k = \{\sigma_k^{(r)}, \sigma_k^{(s)}\}$. $\sigma_k^{(r)}$ means Report the service violation to the smart contract. $\sigma_k^{(s)}$ means do not report and Silence to the smart contract. In a N witnesses game, one of the strategy profile is, $\sigma = (\sigma_1^*, \sigma_2^*, ..., \sigma_N^*)$, where σ_k^* is a specific action adopted by witness w_k. We define the set of witnesses choosing the action of Report as, W_{report}, where $\forall w_k \in W_{report}$, the $\sigma_k^* = \sigma_k^{(r)}$. Respectively, $W_{silence}$ is the set of witnesses not reporting, where $\forall w_k \in W_{silence}$, the $\sigma_k^* = \sigma_k^{(s)}$. These reports determine the final state of SLA: $SLA_{status} = V$iolated, there is a service violation; $SLA_{status} = C$ompleted, service is completed without violation. We then define the violation confirmation as Definition 2.

Definition 2. Violation Confirmation: Based on the result of a strategy profile in a N witnesses game, the service violation is confirmed, $SLA_{status} = V$iolated, only when $\|W_{report}\| \geq M$, where $1 < N/2 < M < N, N, M \in \mathbb{N}$. Otherwise, it is treated as there is no violation happened, $SLA_{status} = C$ompleted.

It is worth to mention that we define $N > 2$ and $M < N$ here, in order to achieve the violation confirmation reliably and fairly. According to our witness model in Section III-C, we design that the witness needs to report the

violation along with some fee to the smart contract. This is to endorse its report. The payoff function, therefore, should give more to this witness, if the violation finally gets confirmed. As a penalty, the witness cannot retrieve back its endorsement fee, if the violation is not confirmed. The principle of the violation reporting is "no news is a good news". That is the witness does not need to report, if there is no violation happening. After the service ends, these witnesses can still be paid for service but with a relatively less amount. Since the silent witness does not report and prepay the endorsement fee, it would not get extra penalty, if the violation is confirmed. However, this behavior can be audited, which is explained in Section IV-C.

We therefore design payoff functions as Definition 3 according to above definitions and analysis. Thereinto, the value of each function is only leveraged to represent the relative quantitative relationship among the fees. Hence, 1 represents one share of profit. 10 is ten times share. -1 means losing one share of profit.

Definition 3. Payoff Functions: The values of payoff functions are designed according to the final SLA status.

- When $SLA_{status} = V$iolated:
  $\forall w_k \in W_{report}, \pi_k = 10$;
  $\forall w_k \in W_{silence}, \pi_k = 0$.
- When $SLA_{status} = C$ompleted:
  $\forall w_k \in W_{report}, \pi_k = -1$;
  $\forall w_k \in W_{silence}, \pi_k = 1$

In a $n\text{-}player$ game, if a player knows the others' actions, it would choose a strategy to maximize its payoff. This is referred as its best response. Define $\sigma_{-k} = \{\sigma_1, \sigma_2, ..., \sigma_{k-1}, \sigma_{k+1}, ..., \sigma_n\}$, as a strategy profile σ without player k's strategy. The full strategy can then be written as $\sigma = \{\sigma_k, \sigma_{-k}\}$. Therefore, the best response of witness w_k can be defined as follows.

Definition 4. Witness w_k's best response: In order to maximize its revenue, w_k's best response to a strategy profile σ_{-k} is a strategy $\sigma_k^* \in \Sigma_k$, such that $\pi_i(\sigma_k^*, \sigma_{-k}) \geq \pi_k(\sigma_k, \sigma_{-k})$ for $\forall \sigma_k \in \Sigma_k (k = 1, 2, ..., n)$.

Hence, a Nash equilibrium [31] is a stable solution, where no player has an incentive to deviate from this current situation. We formally describe it as follows.

Definition 5. Nash equilibrium: It is a specific strategy profile $\sigma^* = (\sigma_k^*, \sigma_{-k}^*)$, if for each witness w_k, σ_k^* is a best response to σ_{-k}^*, i.e., $\pi_i(\sigma_k^*, \sigma_{-k}^*) \geq \pi_k(\sigma_k, \sigma_{-k}^*)$ for $\forall \sigma_k \in \Sigma_k (k = 1, 2, ..., n)$.

Based on the Nash equilibrium definition and payoff functions, we can derive the theorem below.

Theorem 1. In a witness game, the only two Nash equilibrium points are following strategy profiles:

- $\sigma^* = (\sigma_1^*, \sigma_2^*, ..., \sigma_n^*)$, of which for $\forall w_k$, $\sigma_k^* = \sigma_k^{(r)}$;
- $\sigma^* = (\sigma_1^*, \sigma_2^*, ..., \sigma_n^*)$, of which for $\forall w_k$, $\sigma_k^* = \sigma_k^{(s)}$.

Proof. According to Definition 1 and 2, in a N witnesses game, $N \geq 3$, $N/2 < M \leq N - 1$ and $M \geq 2$, where $N, M \in \mathbb{N}$.

For the strategy, for $\forall w_k$, $\sigma_k^* = \sigma_k^{(r)}$, which means $\|W_{report}\| = N \geq M$. The SLA violation status is therefore violated, $SLA_{status} = V$. According to payoff functions design in Definition 3, for $\forall w_k$, its revenue is $\pi_k(\sigma_k^{(r)}, \sigma_{-k}^*) = 10$. If any w_k chooses the other action, *S*ilence, instead of *R*eport. The final status of SLA, however, would not be modified, due to $\|W_{report}\| = N - 1 \geq M$. Then, w_k's revenue is $\pi_k(\sigma_k^{(s)}, \sigma_{-k}^*) = 0 < 10 = \pi_k(\sigma_k^{(r)}, \sigma_{-k}^*)$. Meanwhile, $\Sigma_k = \{\sigma_k^{(r)}, \sigma_k^{(s)}\}$ in this game. According to Definition 5. This strategy profile is a Nash equilibrium point.

Analogously, for the strategy, $\forall w_k$, $\sigma_k^* = \sigma_k^{(s)}$, which means $\|W_{report}\| = 0 < 2 \leq M$. The SLA violation status is therefore not violated, $SLA_{status} = C$. According to payoff functions design in Definition 3, for $\forall w_k$, its revenue is $\pi_k(\sigma_k^{(s)}, \sigma_{-k}^*) = 1$. If any w_k chooses the other action, *R*eport, instead of *S*ilence. The final status of SLA, however, would not be modified, due to $\|W_{report}\| = 1 < 2 \leq M$. Then, w_k's revenue is $\pi_k(\sigma_k^{(r)}, \sigma_{-k}^*) = -1 < 1 = \pi_k(\sigma_k^{(s)}, \sigma_{-k}^*)$. According to Definition 5. This strategy profile is also a Nash equilibrium point.

For all the other strategy profiles, they are all a mix of actions, *R*eport and *S*ilence. It means $W_{report} \neq \emptyset$ and $W_{silence} \neq \emptyset$. When $SLA_{status} = V$, i.e., $\|W_{report}\| \geq M$, there is always $\exists w_k \in W_{silence}$, it can change the action to *R*eport. But the SLA status would not change, because of $\|W_{report}\| + 1 > M$. Hence, w_k increases its revenue, from $\pi_k(\sigma_k^{(s)}, \sigma_{-k}^*) = 0$ to $\pi_k(\sigma_k^{(r)}, \sigma_{-k}^*) = 10$. On the other hand, when $SLA_{status} = C$, i.e., $\|W_{report}\| < M$, there is always $\exists w_k \in W_{report}$, it can change the action to *S*ilence. But the SLA status would not change, because of $\|W_{report}\| - 1 < M$. Hence, w_k increases its revenue, from $\pi_k(\sigma_k^{(r)}, \sigma_{-k}^*) = -1$ to $\pi_k(\sigma_k^{(s)}, \sigma_{-k}^*) = 1$. These counterexamples demonstrate all these strategy profiles are not Nash equilibrium points.

Therefore, in a witness game, there are and only are two Nash equilibrium points. They are $\sigma^* = (\sigma_1^{(r)}, \sigma_2^{(r)}, ..., \sigma_n^{(r)})$ and $\sigma^* = (\sigma_1^{(s)}, \sigma_2^{(s)}, ..., \sigma_n^{(s)})$. □

We take the basic $three$-witness game as an example, where $N = 3$. Therefore, M can only equal to 2 based on Definition 2. Table I shows the payoff functions according to our previous definitions. The value element in Table I is the vector of corresponding payoff function values. It is represented as (π_1, π_2, π_3). According to Theorem 1, Nash equilibrium points in this game are (10, 10, 10) and (1, 1, 1) respectively. None of the witness has the incentive to leave the Nash equilibrium point, as long as it can be achieved. However, neither of these two actions are dominated based on Game theory. Taking the actions of w_1 as an example, $\pi_1(\sigma_1^{(s)}, \sigma_1^{(r)}, \sigma_1^{(r)}) = 0 < \pi_1(\sigma_1^{(r)}, \sigma_1^{(r)}, \sigma_1^{(r)}) = 10$, however, $\pi_1(\sigma_1^{(s)}, \sigma_1^{(s)}, \sigma_1^{(s)}) = 1 > \pi_1(\sigma_1^{(r)}, \sigma_1^{(s)}, \sigma_1^{(s)}) = -1$. Therefore, none of the witness can always choose one action to consistently gain the maximum revenue.

TABLE I: Payoff Functions for a $Three$-witness Game

w_1	w_3			
	$\sigma_3^{(r)}$: *R*eport		$\sigma_3^{(s)}$: *S*ilence	
	w_2		w_2	
	$\sigma_2^{(r)}$: *R*eport	$\sigma_2^{(s)}$: *S*ilence	$\sigma_2^{(r)}$: *R*eport	$\sigma_2^{(s)}$: *S*ilence
$\sigma_1^{(r)}$: *R*eport	(10, 10, 10)	(10, 0, 10)	(10, 10, 0)	(-1, 1, 1)
$\sigma_1^{(s)}$: *S*ilence	(0, 10, 10)	(1, 1, -1)	(1, -1, 1)	(1, 1, 1)

Based on above analysis, for a rational and selfish witness, who wants to maximize its revenue through offering services, would have to behavior as follows in this game. If there is a violation happening, the witness knows that most of other witnesses are more likely to report this event to gain more revenue. Hence, the higher revenue pushes the witness to report this event. On the contrary, if there is no violation, the witness knows that most of other witnesses are more likely keep silence. Although the witness wants to achieve the highest revenue, it has to take a great risk to pay a penalty for its fraudulent behavior. From the global view, when there is no violation, all the witnesses prefer to keep silence in order to stay at the Nash equilibrium point, $(\sigma_1^{(s)}, \sigma_2^{(s)}, ..., \sigma_n^{(s)})$. Then the violation acts as a signal to push them achieving another Nash equilibrium point, $(\sigma_1^{(r)}, \sigma_2^{(r)}, ..., \sigma_n^{(r)})$, with much higher revenue. At the same time, they tell the truth about the service violation.

Therefore, it is not the witness want to tell the truth. Instead, it has to be honest, in order to maximize its revenue.

C. Witness Audit

The sortition algorithm, Algorithm 1, makes sure that the selected witnesses are independent to a great extent. The payoff function design stimulates the witness to tell the truth. However, an auditing mechanism is still needed to ensure that the malicious or unrational witness can be detected and kicked out from the witness pool. As all the interactions with the smart contract, i.e., transactions, are public and permanently stored on the blockchain, it is possible to audit a witness through its behavior history. We mainly exploit following information in the history to do auditing. It is expressed as Equation 1. It represents a set of events, which involves witness w_k. The witness can adopt the action of silence $\sigma_k^{(s)}$ or the action of report $\sigma_k^{(r)}$. σ_k^* means the SLA event with any action adopted by w_k. To be specific, when the action is report, we can also further know its reporting $order$, first to report or not, and the reporting time T_r, relatively from the SLA starting time. In addition, $status$ expresses whether the SLA is violated or completed. $*$ means the event with any SLA status is counted in this set.

$$Event(w_k, \sigma_k^{(s)} | [\sigma_k^{(r)} : (order, T_r)] | \sigma_k^*, status : C|V|*) \tag{1}$$

Based on above information, we analyze that there are three types of malicious witnesses: lazy witness, speculative witness and sacrificed witness.

Lazy witness refers to the one, who prefers not to report the violation. Because there is a case that the higher incentive for reporting violation is not enough to motivate the lazy one. They can choose the strategy not to really monitor the service. Then they always keep silence and never report the violation. With this strategy, they would not pay the penalty, if the final status of SLA is violated. And considering violation is not a normal event, the lazy witness is still able to gain some revenue via multiple "games". However, this type of lazy witness can be audited through its active rate, which is defined as follows.

Definition 6. Active rate ($\eta_{active}(w_k)$): This the metric to measure the activeness of witness w_k, when there is a violation.

$\eta_{active}(w_k) = \frac{\|Event(w_k, \sigma_k^{(r)}, status{:}V)\|}{\|Event(w_k, \sigma_k^*, status{:}V)\|}$

$\eta_{active}(w_k) = 0$ means that the witness w_k never report the violation event, although there actually is one. A threshold, $\widehat{\eta}_{active}(w_k)$ therefore can be set to determine whether w_k is a lazy witness, i.e., still $\eta_{active}(w_k) < \widehat{\eta}_{active}(w_k)$, when w_k has already been involved into many SLA events.

Speculative witness refers to the one, who is more likely to report in a speculative way. Because all the actions are public and transparent on the blockchain. There is a possible speculative behavior for the witness, which is only to follow others' report. The witness does not monitor the service. Instead, it monitors transactions on the blockchain to see whether there is some other witness reporting the violation. Then, it immediately follows and reports, trying to gain the maximum revenue. Although we can set a relatively short report time window in the model design of Section III-C, its speculative reports might still be counted in the following blocks of the blockchain. Moreover, this speculative witness also needs to take the risk that the violation may not be confirmed finally. Anyhow, this type of speculative witness can be audited through its following rate.

Definition 7. Following rate ($\eta_{follow}(w_k)$): This the metric to measure the frequency that the witness w_k follows the reports of other witnesses.

$$\eta_{follow}(w_k) = \frac{\|Event(w_k, \sigma_k^{(r)}{:}(NotFirst, T_r), status{:}V)\|}{\|Event(w_k, \sigma_k^{(r)}{:}(order, T_r), status{:}V)\|}$$

Here, $NotFirst$ means that the transaction containing the report of w_k is not the first block in the reporting time window. So $\eta_{follow}(w_k) = 100\%$ means for all violated SLA events involving the witness w_k, it is never the first one to report. A threshold, $\widehat{\eta}_{follow}(w_k)$ therefore can be set to determine whether w_k is a speculative witness through following, i.e., when $\eta_{follow}(w_k) > \widehat{\eta}_{follow}(w_k)$, if w_k is involved into many SLA events.

Sacrificed witness refers to the one, who always reports at a specific time stamp. For instance, w_k always reports the violation within one minute after the SLA starts. Though the witness may pay a lot of penalty for its malicious behavior in the beginning, it can show other witnesses its behavior pattern from its history later on. In some sense, it is able to imply others that it would report at some time stamp. Then, it can most likely gain the maximum revenue, as long as others have analyzed its behavior pattern. Hence, it is important to audit this type of witness through the following fixed pattern rate.

Definition 8. Fixed pattern rate ($\eta_{fix}(w_k)$): This the metric to measure the frequency that the witness w_k report the violation at a specific time stamp, $\widehat{T}_r$.

$$\eta_{fix}(w_k) = \frac{\|Event(w_k, \sigma_k^{(r)}{:}(order, \widehat{T}_r), status{:}*)\|}{\|Event(w_k, \sigma_k^{*}, status{:}*)\|}$$

Here, $\eta_{fix}(w_k) = 100\%$ means for all of the SLA events, which involves the witness w_k, it always reports at time stamp, $\widehat{T}_r$. A threshold, $\widehat{\eta}_{fix}(w_k)$ therefore can be set to determine whether w_k is a sacrificed witness through following, i.e., when $\eta_{fix}(w_k) > \widehat{\eta}_{fix}(w_k)$, even if w_k is just involved into several SLA events.

It is worth to mention that these auditing mechanisms can also be implemented in the smart contract, in order to avoid a third party to dominate the judgement. It can be combined with the reputation value of the witness, which is further explained as the witness reputation in the implementation part of Section V-B. Therefore, when the witness' reputation decreases to zero, that witness would also be blocked automatically by the sortition algorithm.

V. Prototype Implementation and Experiments

According to the witness model and the payoff function design, we implement a prototype system based on the smart contracts of Ethereum. We leverage the programming language, Solidity [4], provided by Ethereum, to program smart contracts. In this section, we firstly would like to discuss about the overall relationship among different roles and smart contracts in our SLA system. Afterwards, we illustrate the state transition in the smart contract of witness pool and SLA, respectively. Via this, we describe the detailed functionalities of the interfaces in the smart contracts and show how they are leveraged to transit the states. Finally, we show some experimental study on the transaction cost of these interfaces on the Ethereum test net, "rinkeby".

A. Overall System Implementation

Overall, there are three roles and two types of smart contracts in our SLA enforcement system. Roles include the traditional Provider and Customer, as well as the introduced Witness. The smart contracts include the witness-pool smart contract and the SLA smart contract. Figure 3 illustrates the relationship among these entities through different interfaces. These interfaces are named as the text on the arrow. The format of the text is '$R_{role} \rightarrow [C_{type}::]N_{interface}$'. It means that only the role R_{role} can invoke the interface, $N_{interface}$, which is defined in the smart contract of C_{type}. This is achieved by the checking mechanism, which is the property of the programming language provided by Ethereum. Therefore, the smart contract restricts that only the specified role is able to interact with the smart contract in certain state. The representation of the R_{role} are P for Provider, C for Customer, SC

[4]http://solidity.readthedocs.io

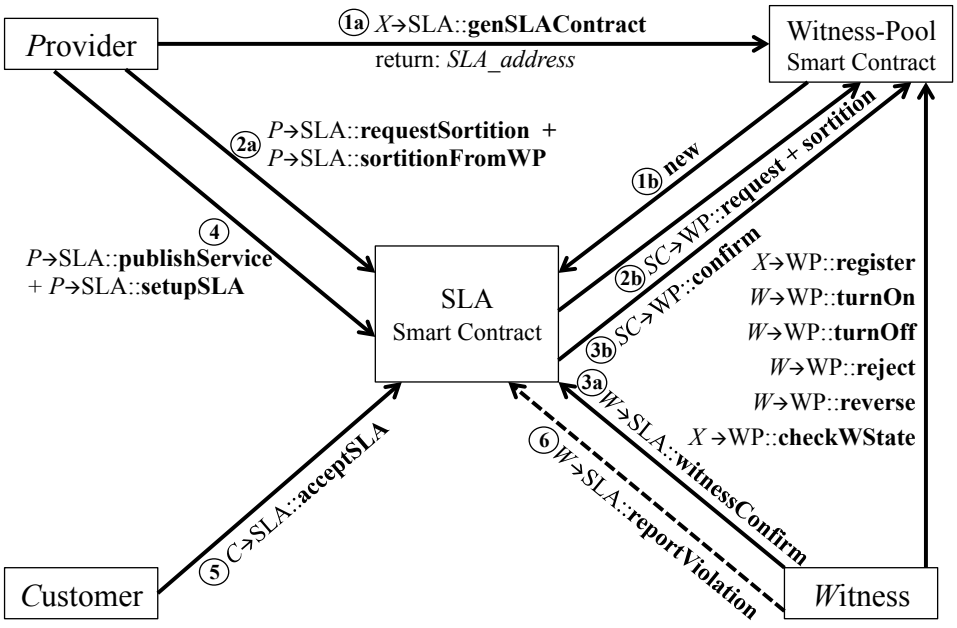

Fig. 3: Interactions among Roles and Smart Contracts

for a generated SLA Smart Contract and X for any blockchain user. And for C_{type}, WP is for the witness-pool type of smart contract and SLA is for the generated ones for SLA enforcement. In addition, this interface definition also applies to Figure 4 and 5.

The witness-pool smart contract is the basis in order to setup the system. Any blockchain user is able to participant and become the Witness role through the interface "register" provided by the witness-pool smart contract. There are also some other interfaces for the witness to invoke to change its state in the pool in order to be selected. This is discussed in Section V-B. For a specific SLA lifecycle, any user X is able to invoke the interface "genSLAContract" provided by witness-pool smart contract. Afterwards, a specific SLA smart contract is generated by witness-pool smart contract and the contract address is returned back. This address is also recorded in the witness-pool. It ensures the validity of the SLA smart contract to interact with other roles. Meanwhile, the user X becomes the Provider role of the generated SLA smart contract. It has the ability to customize the contract including setting the customer's address and other negotiated contract parameters, such as service duration, witness committee scale N, etc. It is also responsible for performing the unbiased random sortition algorithm, explained in Section IV-A. As this algorithm is leveraged by the provider through a specific SLA contract, the online witness is able to know it is selected by which SLA smart contract, basically the address, from checking its own state. Then it can make a confirmation to the SLA contract to join the witness committee. Simultaneously, the SLA smart contract further invokes interfaces of witness-pool smart contract to acknowledge the witness management. After a proper witness committee is constructed, the provider can publish its service detail on the chain to initiate the SLA lifecycle. This will be explained in Section V-C.

B. Witness-pool Smart Contract Implementation

In this part, we focus on implementation details about the witness-pool smart contract, especially the witness management. Figure 4 illustrates the states of a witness role defined in the smart contract. It includes four states: "Online", "Offline", "Candidate" and "Busy". The state transition of witness is as follows.

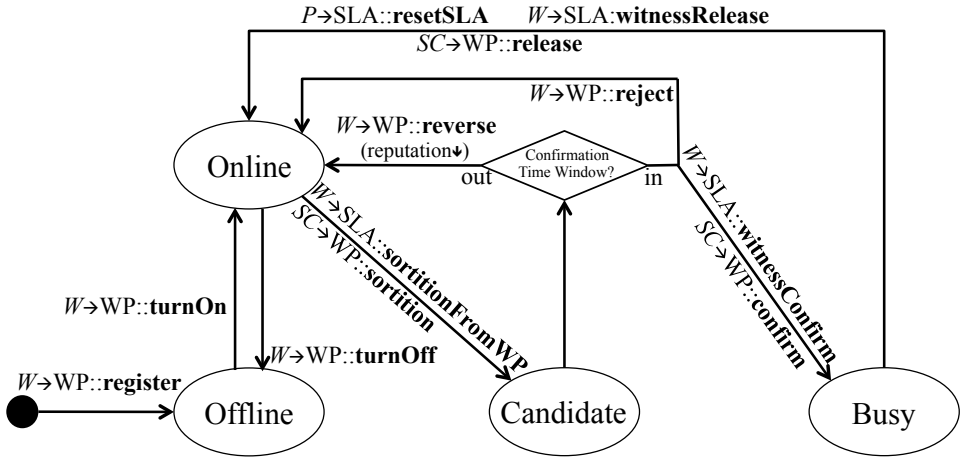

Fig. 4: Witness State Transition Diagram in the Witness-pool Smart Contract

After registration in the witness pool, only the witness itself can turn its state into “Online”. It then probably selected by certain SLA contract. Hence, it needs to monitor its own state on the blockchain continuously. This is feasible, as the read-only operation does not need any transaction fee. Once it is selected by certain smart contract through performing “sortition” algorithm, its state turns into “Candidate”. Within a confirmation time window, e.g., 2 minutes, it can look through the SLA smart contract, which makes the selection, and decide whether to confirm or reject this selection. If it rejects, the provider of the SLA smart contract has to perform another sortition. Otherwise, its state turns into “Busy”, after it invokes the interface, “witnessConfirm”, of the SLA smart contract. By the end of each SLA lifecycle iteration, the witness has the right to actively leave the SLA contract by leveraging the interface “witnessRelease”. On the other hand, it can also be passively released from the SLA contract. Because the provider invokes the interface “resetSLA” to dismiss the witness committee. Finally, the witness can “trunOff” to avoid being selected when it is not available to the Internet.

In order to prevent some malicious intentions, we bring in a reputation value for each witness to measure their behaviors. Firstly, each witness has an initial reputation value of R_{init} at registration, which could be a predetermined constant or a value depending on its stake or some deposit. Then, for instance, some witness may not turn its state into “Offline”, when it is not actually available or does not frequently check its state. Then, it would not be able to confirm the selection and join the SLA contract within the confirmation time window, if it is chosen. In this case, the witness would not be chosen again, since its state becomes “Candidate”. To reverse back to the state “Online”, in which it can be selected, the witness only have to leverage the interface, “reverse”. However, its reputation value decreases by 10. If this value becomes zero or less, it would be permanently blocked by the sortition process according to Algorithm 1. We also combine the reputation value with the auditing mechanism mentioned above in some sense. For example, the reputation of the witness, who does not report, would decrease by 1, when the violation is confirmed. It is the same with the one, who reports the violation but not finally confirmed. The lazy witness and sacrificed witness, therefore, are blocked soon. Moreover, another interface dealing with the reputation value needs to be implemented in the witness-pool smart contract to identify the speculative witness.

C. SLA Smart Contract Implementation

Figure 5 shows the SLA state transition to implement a specific SLA smart contract enforcement. This type of smart contract is generated by the witness-pool smart contract. All the interfaces annotated in this figure belongs to this type of SLA smart contract. Hence, we omit the definition scope C_{type}::. There are five states: “Fresh”, “Init”, “Active”, “Violated” and “Completed”, shown as circle in Figure 5. The dash arrows demonstrate the state transition path when violation happens. The three squares in the figure represent the corresponding roles in this smart contract. In the end of SLA, they can withdraw the revenue respectively. The dash line here also refers to the action adopted under the situation of violation.

The contract is generated in the state of “Fresh”. In this state, the provider is able to customize the SLA parameters according to the negotiated results with the customer. Furthermore, the service detail can also be published onto the contract through “publishService”. In our case, the detail is the public IP of the VM. All others are therefore being notified. However, this SLA proceeds only when the number of the members in the witness committee is satisfied.

Otherwise, the provider is unable to leverage the interface, “setupSLA”, to transit into “Init” state. According to the witness model design in Section III-C, the provider needs to prepay some fee, $PF_{prepaid}$, to the smart contract for hiring witnesses. The concrete amount of fee is calculated by the smart contract according to the scale of committee member and a basic hiring fee. In addition, this amount is one of the constraints to invoke the interface. It ensures that only that amount of prepaid fee is transferred into the smart contract. The customer then decides whether to accept. If it accepts the SLA, it also needs to prepay the fee, $CF_{prepaid}$, including the service fee and its part of hiring fee for witnesses. If not, the provider can withdraw back its money and “cancleSLA”. When the service is completed, all corresponding roles can retrieve its revenue through a set of withdraw interfaces. After all the money is withdrawn from the contract, the provider can leverage “resetSLA” or “restartSLA” to rotate back to the previous state. This is used for continuous service delivery instead of a long service duration to stuck witnesses. The difference between these two interfaces is “resetSLA” dismisses the witness committee and the provider is able to change some other terms. On the other hand, “restartSLA” would keep the committee and quickly proceed another round of SLA iteration.

It is also worth to mention that the smart contract on blockchain cannot run itself. The state transition must be triggered by some interfaces and it takes some cost to execute. Therefore, we design the interface for the role, who is the greatest beneficiary in some cases, to modify the state. Because they have the motivation to perform the state transition. For example, when the service duration ends normally, the provider is the greatest beneficiary to gain the entire service fee. It must actively leverage the interface, “providerEndNSLAandWithdraw”, to end the normal SLA and withdraw its own revenue. Meanwhile, it divides the prepaid money as the payoff function design in Section IV-B to different witnesses. Afterwards, other roles are able to withdraw their part of revenue. Analogously, when there is a violation, the customer is the most motivated one to gain the compensation fee. It can leverage, “customerEndVSLAandWithdraw”, to end the violated SLA and transit the state from “Violated” to “Completed”. It is the same for “resetWitness”, which is the customer to reset the witnesses’ state and report time window, when the violation is not confirmed. Otherwise, witnesses cannot report the violation later on, if there are real ones.

D. Experimental Study

In order to test all the functionalities of our model and system design, we deploy the implemented smart contracts on the test net of Ethereum blockchain, “Rinkeby”. It is a world-wide test blockchain for developers to debug the smart contract. The ‘Ether’, which is the cryptocurrency of Ethereum, does not worth real value on the test net and can be applied for debugging. Hence, we generate several accounts on “Rinkeby” to simulate different roles, i.e., the provider, customer and witnesses. We leverage the retrieved ‘Ether’ on each simulated account to execute the interfaces and prepay different types of fees according to the model. To conduct the experiment, we first deploy the basic witness-pool smart contract, and make all the accounts register to the witness pool. The provider then generates a SLA smart contract to start the SLA lifecycle with the customer. Afterwards, we test all possible scenarios to exploit and validate the functionality of different interfaces. The results demonstrate our system implementation satisfies our model and payoff function design.

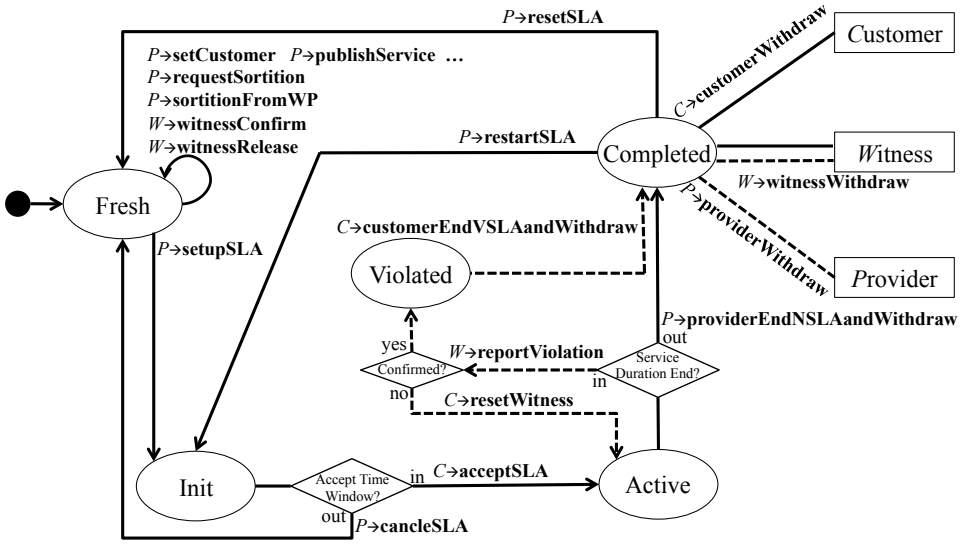

Fig. 5: State Transition Diagram of SLA Lifecycle for a Specific SLA Smart Contract

The trust part of the system is proved by Game theory and ensured by the unbiased sortition algorithm, whose credibility is endorsed by the blockchain technique. Therefore, we mainly analyze some performance information from our experimental study. Thereinto, the performance refers to the complexity of each interface in the smart contract. It determines the transaction fee needed to pay the miner in Ethereum. Because the miner needs to execute the program defined in the interface. It consumes the electricity power of the miner. The more complex of the interface is, the more transaction fee required when it is invoked. This is measured as 'Gas' defined in Ethereum, which is a unit to refer how much work taken by the miner when executing the transaction. The final fee is the product of gas amount and the gas price for each unit. Hence, the gas consumption is similar no matter on test net or main net. We therefore record all the gas consumption for each interfaces from the transaction history of the experiment.

Figure 6 illustrates the gas consumption of each interface. We show the major interfaces defined in the two types of smart contracts, which construct the system. Some simple interfaces, such as the one that sets the SLA parameters, are omitted. Their gas consumptions are similar to 'setServiceDuration' in the figure. Some other interfaces of witness-pool smart contract are also omitted. Because these are the ones can only be invoked by the SLA smart contract. Its consumption is involved in some interface in the right part of the figure. In addition, the consumption of 'genSLAContract' in witness-pool smart contract is also not shown in the figure. Because it is more than 10 times higher than others, which is around 2,200,958. However, it is acceptable for the provider to invoke this a generate a new SLA smart contract, since the SLA smart contract is reusable.

From the experimental study, it can be derived that, compared with the customer and the witness, the provider tends to require more gas in the entire SLA lifecycle. The interfaces of customer and witness consume less. This fits our model design and reality. Because in most cases, the provider earns the most revenue through offering service. It has the incentive to proceed the lifecycle. The light weight gas consumption for witness role is also able

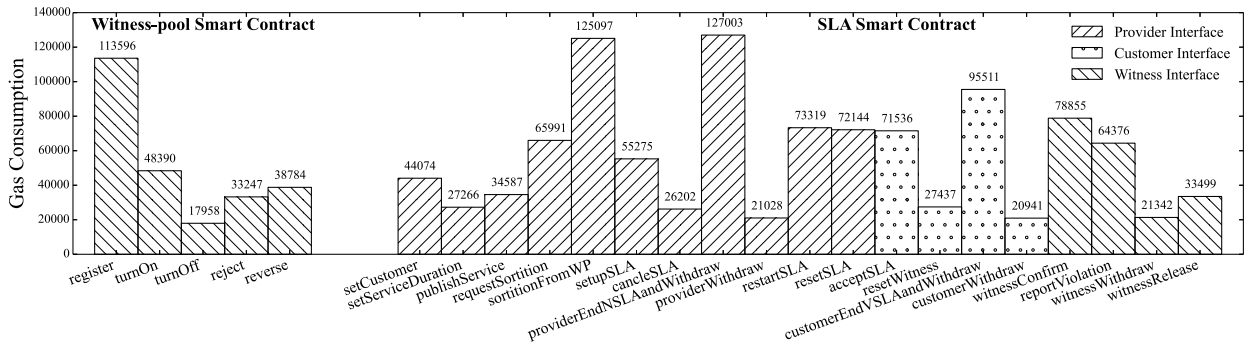

Fig. 6: The Gas Consumption for Each Interface in Smart Contracts

to convince blockchain users to take part in the system to work as a witness. Moreover, these gas consumption values are achieved through experiments based on current implementation. There is still possible space to further optimize the interface implementation to lower the gas consumption.

VI. Conclusion

In this paper, a witness model is proposed for cloud SLA enforcement and specially design the payoff function for each witness. We leverage the game theory to analysis that the witness has to offer honest monitoring service in order to maximize its own revenue. Finally, a prototype system is fully implemented using smart contracts of Ethereum to realize the witness model. Not only the SLA enforcement lifecycle but also the witness management of witness pool is implemented with the smart contract. The experimental study demonstrates our model feasibility and shows the system performance. Via this way, the trust problem is transferred into economic issues. It is not the witness itself would like to be honest, but the economic principles force them to tell the truth. Thereinto, the blockchain plays as a public and immutable monetary management platform according to some predefined rules. We also believe the witness model can be applied in other scenarios with blockchain, where originally only two roles are involved in a contract.

For the future work, there are mainly two directions: on-chain and off-chain. For the on-chain part of work, we are going to further optimize the interface implementation to reduce the gas consumption and enrich the functionalities of the smart contract. In addition, some more scenarios should be considered to apply our model. For the off-chain part of work, user-friendly tools are going to be developed for each role in the system to monitor the state on the chain and perform their corresponding interactions. On the other hand, it can also be combined with our cloud application DevOps framework, CloudsStorm [5], to construct the witness ecosystem. The vision is to insure the cloud performance for applications through automated SLA.

[5]https://cloudsstorm.github.io/