A Blockchain-based approach for Service Level Agreement Management in Cellular Network

[1, 2] Nischal Aryal, [2] Fariba Ghaffari, [1, 2] Emmanuel Bertin, [2] Noel Crespi
[1] *Orange Innovation, 14000 Caen, France*
[2] *SAMOVAR, Telecom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France*
{nischal.aryal, emmanuel.bertin}@orange.com, fariba.ghaffari@telecom-sudparis.eu, and noel.crespi@it-sudparis.eu

***Abstract*—Service Level Agreements (SLAs) serve as the cornerstone of collaboration and service delivery between providers and customers, delineating the quality of services that the provider commits to deliver through various terms and conditions. However, the complexity, manual and centralized communication processes and the potential for action denial pose challenges to the current SLA management. Moreover, while the traditional SLA process may accommodate current negotiation and contract volumes, scaling becomes problematic with increasing numbers of providers, service types, and consumers. Blockchain technology offers inherent features such as high automation and scalability, transparency, trust, immutability, and non-repudiation, which are particularly beneficial in the context of SLA management. This paper proposes a Blockchain-based solution for managing SLA Lifecycle using smart contracts and Oracles. We primarily focus on three main phases of this lifecycle: SLA Negotiation, SLA Violation monitoring, and SLA Compensation through automatic cryptocurrency-based compensation or adjustments of SLA rules based on predefined policies established during the initial phase (service credits). This approach streamlines the SLA agreement and compensation processes while offering scalability, trust, transparency, and non-repudiation. Assessments in the specific use-case of cellular networks confirm the scalability of this solution.**

***Index Terms*—SLA, Cellular Networks, Smart Contracts, Oracles, SLA Monitoring, Smart contract**

I. Introduction

Due to the increasing number of users in the telecommunication network landscape, Mobile Network Operators (MNOs) require innovative solutions to meet the diverse needs of their customers. To achieve this, MNOs plan to integrate novel technologies and services [1] into next-generation networks (beyond 5G and 6G) and collaborate with other business entities by buying or selling services. One essential requirement for collaboration is the establishment of a Service Level Agreement (SLA). SLAs are agreements between two parties that outline the type of service they anticipate from each other, the Service Level Objectives (SLOs), and procedures to handle service violations. SLOs are the target values for key performance indicators (KPIs), such as uptime, response time, and availability, and can differ based on the type of services.

Although SLAs are essential for effective collaboration, managing them is not an easy task. SLA management in the current cellular network is done manually. As the number of collaborations between MNOs and other businesses (e.g., smaller connectivity providers, RAN providers, service providers, etc.) increases, there will be a significant increase in the number of SLAs, making manual methods of management impractical [2]. Furthermore, SLAs encompass numerous metrics and conditions, making manual monitoring prone to difficulty and errors. However, a key challenge is to ensure compliance and enforce SLA terms, especially in cases of violation and disagreement between parties.

Any alternative providing trust, scalability, security, and a high level of automation would be a potential solution to address the aforementioned challenges. Blockchain technology [3] is one such approach. It is a Distributed Ledger Technology (DLT) containing a network of nodes that participate in the security of the system by providing a cryptographically secure chain of blocks where all changes in the system require the consensus of all the eligible nodes. Thanks to their features, such as immutability, transparency, automation, and consensus, there is a huge research interest in using Blockchain in different aspects of cellular network ecosystem, including collaboration management [4]–[6].

In this work, we present a distributed SLA management approach using Blockchain, smart contracts, InterPlanetary File System (IPFS), and Oracles. Smart contracts handle SLA negotiation and store the final version of encrypted SLA terms in Blockchain. Due to storage limitations in the Blockchain, the SLA metrics of each entity are stored off-chain, such as in IPFS, and a hybrid cryptosystem is applied to provide privacy for SLA terms. Oracles are the edge points allowing the smart contracts to request and receive verified off-chain data in the Blockchain and enabling the smart contracts to access the metric logs. Smart contracts utilize these SLA metrics for violation monitoring. We provide two methods to compensate for violations: service credits and cryptocurrency-based compensation.

The key contributions of proposed method are as follows:

1) Eliminating the need for **intermediaries** during SLA negotiation, deployment, and violation check.
2) Providing a **negotiation** platform on Blockchain using smart contracts to decrease the latency existing in the manual paper-based SLA negotiation process.
3) **Automating SLA Violation Check** with the help of Oracle based on the requests from each party.
4) Providing **two compensation methods** for SLA violations based on payment or enhancement of SLA terms.
5) Providing **privacy in terms of SLOs and the identity of the parties** using a hybrid cryptosystem, which provides

access only to the parties participating in the SLA.

6) Using a **secure off-chain database** (for SLOs and log files) to improve not only the privacy of the system but also the required storage.

The rest of this paper is organized as follows: Section II provides a brief background, followed by a summary of the state of the art in Section III. Section IV outlines the system design and construction of our proposed method, followed by the evaluation in Section V. Section VI provides our conclusions about the proposed method as well as some future research directions.

II. Background

A. Blockchain, Smart Contracts and Oracles

Blockchain is a peer-to-peer distributed ledger where updates can only be made by consensus among the majority of the nodes present on the network [3]. Blockchain is implemented in the form of a linked list, with each block linked to the previous block by its hash. As a result, any change in data, causing it to differ from the hash contained in the previous block, makes Blockchain considered immutable. When a transaction is transmitted to this distributed network, it must be validated by the network's participants using a consensus mechanism.

Smart contracts are computerized transaction protocols that execute the terms of a contract on top of Blockchain. The main objectives of smart contracts are to satisfy common contractual conditions, minimize malicious and accidental exceptions, and reduce the need for trusted intermediaries. Ethereum was the first to implement smart contracts in 2014 [7]. The most well-known smart contract language is Ethereum's Solidity [8] which has a "Turing Complete" virtual machine for the execution of smart contracts [9].

Oracles are important components in Blockchain technology that connect smart contracts to the off-chain world, enabling them to interact with external data and systems. There are different types of Oracles, such as software, hardware, inbound, and outbound. In this paper, software Oracles are used to handle the return of data from a predefined URL connected to a database storing the SLA-related KPIs.

B. SLA Lifecycle

A SLA lifecycle consists of the following phases [10], [11]:

1) *Identifying Service Provider*: In this phase, the consumer finds and selects a Service Provider (SP) that can provide resources according to their needs.
2) *Negotiation*: In this phase, all the key performance metrics that are expected from the service are added to the SLA in the form of SLOs. These SLOs can either be provided by the SPs in ready-made SLAs or the consumers can add them. This process runs in a loop where involved parties negotiate regarding changes in SLA. It continues until all parties reach an agreement. The parties also discuss the compensation scheme for handling SLA violations.
3) *SLA Deployment*: In this phase, both parties agree, and the final SLOs along with the compensation strategies are written in SLA.
4) *SLA Violation Monitoring*: In this phase, an entity constantly monitors the performance of the service verifying if all the requirements are met. If there is any anomaly in the data, the entity identifies the offender and alerts both parties of the agreement.
5) *Penalty Enforcement*: In this phase, the violator has to compensate for the SLA violation. This is done through the compensation scheme defined in the SLA. An example of the scheme could be paying a certain amount defined in the SLA.
6) *SLA termination*: In this phase, the agreement between the two parties is canceled, and SPs stop providing services to the consumer. This can happen due to the contract's expiration or through the decision of involved parties.

III. Related Works

Due to its decentralized, immutable, and transparent nature, Blockchain technology can provide a robust solution to ensure the integrity and accountability of SLA-related transactions and agreements. As a result, this technology has gained significant interest in research and businesses. Many studies have explored this area, highlighting the different solutions for using Blockchain to enhance SLA management. From our investigation in this area, we could categorize the proposed methods into three main groups, representing how the Blockchain can be used for SLA management: 1) as a distributed database to store the SLA terms; 2) as a distributed method for violation checking; 3) as a distributed method to apply the penalty for SLA violations.

Weerasinghe et al. [12] introduce an SLA management system on top of Blockchain using their formally proven novel consensus model as Proof of Monitoring (PoM) for Secure Service Level Agreement (SSLA), in which the auditors ensure that the service providers deliver the security-related KPIs based on the terms of their SLA. A Blockchain-based SLA management solution for the IoT ecosystem is introduced by Alzubaidi et al. [13] in which they focus on SLA monitoring and compliance assessment, believing that no single party should solely control the SLA monitoring. To do so, they use Blockchain to share the log files. They also compare the performance of Ethereum's Proof of Work (PoW) and Hyperledger Fabric's Practical Byazantine Fault Tolerance (PBFT), resulting in PBFT being the preferable solution. To automatically manage the SLAs in a fog environment, Battula et al. [14] proposed a solution in which the entities store the SLA terms in Blockchain, and other contracts, on an on-demand basis, retrieve the network log from Oracles to validate the SLA violation.

In the cloud computing environment, [10] proposed an open cloud market and one-to-many service deployment solution using Blockchain. In this method, the authors use their proposed SLA definition language (SLAC) to define the SLA terms in Blockchain after off-chain negotiation. Moreover, Ranchal et al. [15] propose SLAM, a Blockchain-based framework for continuous SLA monitoring in a multi-cloud environment. This system uses Blockchain to detect SLA violations and determine their root cause across the hierarchical SLAs. A real-time SLA monitoring solution is proposed by [16] in which the monitoring system stores the logs in Blockchain as a distributed and immutable database. Another method for a cloud environment is proposed by [17] to monitor the specific SLA violation regarding the user's data integrity.

In the cellular network sector, Luo et al. [18] proposed a network slice management solution on top of Blockchain to guarantee SLA compliance using three protocols: slice, audit, and dispute. These protocols are used for defining SLA and payment, monitoring SLA compliance, and handling SLA violations based on evidence after detection, respectively. Furthermore, [19] proposes a Blockchain-based solution for inter-provider agreements in 6G networks. They use Chainlink as the Oracle for monitoring and assessing SLA compliance; they also provide penalty calculations. Moreover, some solutions are providing automated compensation payments such as [20]. This method also checks the validity of the SLA based on the number of violations and the expiration time to manage the payment to the entities accordingly.

IV. System overview

In this section, we present our smart contract proposal and explain the system design process.

A. Smart Contract Proposal

We design the following smart contracts in our proposed approach.

1) **Negotiation Smart Contract** (SC_{Neg}) is a temporary contract that is created when one party proposes an SLA request to the other party. The parties read and submit the proposals to this contract. SC_{Neg} stores the two latest SLA proposals from each party. When both parties reach an agreement, the final agreement is written in a new contract (SC_{SLA}) and this contract is destroyed (i.e., contract byte codes will be removed from the Blockchain using predefined `selfdestruct()` function defined by Solidity language).
2) **SLA Smart Contract** (SC_{SLA}) is the final contract that stores the final information related to the agreement, which is accepted by both parties. Note that, the SLA terms are not directly stored in this contract. It also contains a `violation_check()` function to request an SLA violation check. For violation check, this contract requests SC_{Or} to fetch SLA-related KPIs from the off-chain database for involved parties.
3) **Oracle Smart Contract** (SC_{Or}) helps other smart contracts access SLA-related KPIs of involved parties, which are stored in an off-chain database.

B. System design

This section outlines the system design according to the primary phases of the SLA management lifecycle, including Service Provider Identification, Negotiation, SLA Deployment, Monitoring for SLA Violations, Enforcement of Penalties, and SLA Termination.

1) Identifying Service Provider: In the SLA management process, this stage relies on advertisement materials from MNOs and service providers concerning their services or infrastructure. For example, a provider who can deliver connectivity services in remote areas can be a potential partner (or customer) for MNO. It is important to clarify that this paper concentrates on the contracts and agreements between entities capable of reciprocally providing services to each other, rather than end-users. In this paper, one party in the agreement is the MNO, while the other party may be another provider utilizing services offered by the MNO (e.g., storage or internet bandwidth), providing services to MNOs (e.g., antennas or storage), or offering services while utilizing other services from the MNO (e.g., providing antennas and utilizing internet bandwidth). Thus, this stage emphasizes identifying suitable entities based on the requirements of each party.

2) Negotiation: In this step, we deploy SC_{Neg} contract in which two parties negotiate the services, assessment KPIs, and compensation rules or fees through Blockchain and smart contracts. Note that, to simplify the explanations, we entitle one of the parties as a provider and the other as a customer. But in a real-life scenario, these roles are fully interchangeable. To do this, the following steps will proceed (the enumeration is based on Fig. 1):

(N1): In the first step, the customers log into the DApp or the web3-based website to select their requirements for the services. This request is sent as a set of values for SLA and compensation terms. Some examples of SLA terms are as follows:

$$Terms_{SLA} = [Uptime, MTTR, packet\_Loss, CPU\_Capacity, Network\_Throughput, Storage\_Capacity, MTBF]$$

Moreover, the compensation terms are also included in this request. In this paper, two types of compensation are considered: 1) upgrading the SLA plan, and 2) direct payment.

$$Terms_{Compensation} = [Penalty\_type, Penalty\_fee, Min\_deposit, Penalty\_service]$$

where $Penalty\_type$ can be `Zero` for service augmentation, `One` for direct payment, and `Two` when both may apply. $Penalty\_fee$ defines the fee that the provider needs to pay the customer in case of SLA violation (some percentage of this fee can be defined in $Min\_deposit$); $Min\_deposit$ is the amount of fee that the provider (or both entities) needs to deposit inside that SLA contract, so in case of SLA violation, this fee will automatically and directly send to the customer's account (i.e., this money is similar to guarantee deposit). $Penalty\_service$ is the new SLA rule that will be automatically applied in case of SLA violation.

(N2): Using these data, one **temporary** smart contract, namely SC_{Neg} will be deployed in the Blockchain to store the proposed SLA and compensation terms.

(N3): Once the SC_{Neg} is deployed, an event will be emitted to the provider informing them of the request for a new SLA.

(N4): Provider reads the requirements and can either accept, reject, or start a negotiation process with the following steps: (note that the negotiation starts from *Step N3*, as shown in Fig 1.)

- `Accept`: If the provider accepts the proposed SLA and compensation terms, go to *Step N7*.
- `Reject`: If the provider rejects the proposed SLA or compensation terms and doesn't want to negotiate, the `selfdestruct()` function of SC_{Neg} will be called, and the customer will be notified of the rejection of the request.
- `Negotiate`: If the provider aims to negotiate on the proposal, go to *step N5*.

(N5): provider proposes new parameters and terms for the SLA and sends it as a transaction to the Blockchain. As already mentioned, these negotiation SLA and compensation terms are stored in a temporary SLA contract. Similar to *Step N1*, these data are sent as $Terms_{SLA}$ and $Terms_{Compensation}$.

(N6): The proposed changes are written to SC_{Neg} with the provider's signature on these changes. This signature shows that the provider has accepted the changes.

(N7): Once the provider's changes are applied to SC_{Neg} (or the accept/reject response is received by Blockchain), an event will be emitted to the customer informing them about the request for a new SLA.

(N8): The customer reads the revised requirements and, similar to the provider, can either accept, reject, or continue the negotiation process with the following steps:

- `Accept`: If the customer accepts the proposed SLA and compensation terms, go to *Step D1*.
- `Reject`: If the provider rejects the proposed SLA or compensation terms and doesn't want to negotiate, the `selfdestruct()` function of SC_{Neg} will be called, and the provider will be notified of the rejection of the request.
- `Negotiate`: If the provider aims to negotiate on the proposal, go to *step N9*.

(N9): The customer gives the changes in the proposal (similar to *Step N5*). After this step, the loop is continuing from *Step N3*.

3) SLA deployment: When the entities reach an agreement on the terms of SLA, this phase will manage the storage of SLA terms in off-chain database as well as deploying SLA contract in Blockchain. To do this, the following steps will proceed (the enumeration is based on Fig. 1).

(D1): Customer and provider accept the SLA change by signing the final values of the SLA. Note that, in each negotiation step, we keep two important data: 1) two latest values of the negotiation; and 2) the entity that proposed the values. Indeed, the person who proposed the values has accepted these terms, and we need the other party's agreement on them. Final validation will be done in the next step. The customer and provider get informed about the acceptance of the SLA.

(D2): To address the privacy issues related to the business of the entities, we propose to store the SLA KPIs in an off-chain distributed database (i.e., IPFS) instead of storing them directly in the Blockchain that can be visible for other players in the network. To do so, the agreed parameters need to be added in one file (SLA) by any of the entities (e.g., a `json`), and then this file has to be recorded in the database while strictly limiting access to data. To do so, we employed a hybrid cryptosystem for a multi-user environment, combining symmetric and asymmetric cryptography. So, the following steps can be executed by the service provider:

- Generates symmetric key K_s (this key needs to be sent to the customer using a secure off-chain channel)
- Encrypts K_s using Pub_C and Pub_{SP} and gets $EN^{K_s}_{Pub_C}$ and $EN^{K_s}_{Pub_{SP}}$
- Encrypts SLA with K_s to get $EN^{SLA}_{K_s}$

(D3): Service provider stores $EN^{SLA}_{K_s}$ in IFPS and gets a unique content identifier (CID) as an index to the stored data. The CID (let's call it $CID_{EN^{SLA}_{K_s}}$) can be used for further access to the data in IPFS. Moreover, the encrypted version of the K_s using the service provider's and the customer's public key (i.e., $EN^{K_s}_{Pub_C}$ and $EN^{K_s}_{Pub_{SP}}$) and the hash of SLA ($hash(SLA)$) are stored in Blockchain.

(D4): The provider calls function `acceptProposal()` from SC_{Neg} by sending $EN^{K_s}_{Pub_C}$, $EN^{K_s}_{Pub_{SP}}$ and $hash(SLA)$. The following verification is done by SC_{Neg}:

$$msg.sender \equiv Add_{provider} ||$$
$$msg.sender \equiv Add_{customer}$$

where $Add_{provider}$ and $Add_{customer}$ are assigned in the `constructor()` of the SC_{Neg} and are the provider's and customer's Blockchain addresses, respectively.

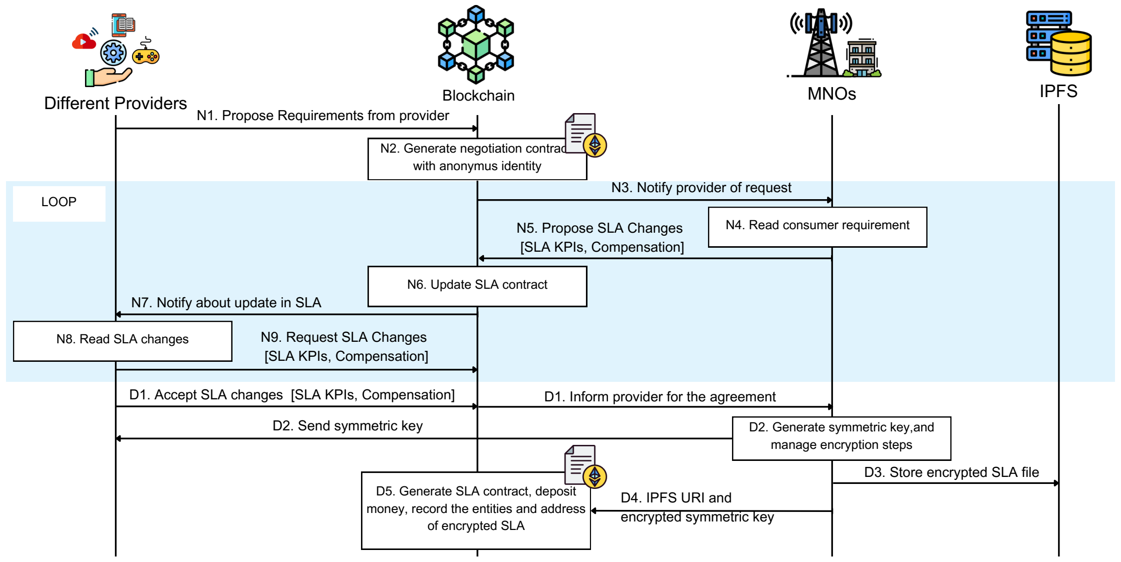
$$msg.sender \not\equiv Add_{second\_entity}$$

That means the second entity (Add_{second_entity}) that proposed the SLA terms cannot accept the proposal (i.e., the other entity, who has not read the SLA yet, needs to accept).

$$Add_{first\_entity} \not\equiv Add_{second\_entity}$$

That means the first and second entities who proposed the terms of SLA, cannot be the same. For instance, the customer cannot change the SLA terms two consecutive times and then accept them; finally, two of the following conditions must be valid:

$$hash(Terms_{SLA\_1}) \equiv hash(SLA)$$



Different Providers
Blockchain
MNOs
IPFS
N1. Propose Requirements from provider
N2. Generate negotiation contract with anonymus identity
LOOP
N3. Notify provider of request
N4. Read consumer requirement
N5. Propose SLA Changes [SLA KPIs, Compensation]
N6. Update SLA contract
N7. Notify about update in SLA
N8. Read SLA changes
N9. Request SLA Changes [SLA KPIs, Compensation]
D1. Accept SLA changes [SLA KPIs, Compensation]
D1. Inform provider for the agreement
D2. Send symmetric key
D2. Generate symmetric key,and manage encryption steps
D3. Store encrypted SLA file
D5. Generate SLA contract, deposit money, record the entities and address of encrypted SLA
D4. IPFS URI and encrypted symmetric key

Fig. 1. A flow diagram representing SLA contract deployment and negotiation procedure performed through Blockchain. As defined in Section IV-B2, three scenarios can take place during N4: acceptance, rejection, or negotiation. For acceptance, the procedure moves from N4 to D2. For rejection, the negotiation contract gets destroyed and the process stops. For, negotiation, the process moves from N4 to N5. The blue box in the figure denotes the negotiation loop.

That means the hash of already accepted KPIs has to be the same as the hash sent by the provider. Note that this scenario is verifiable by the customer.

(D5): If the conditions are addressed, SC_{Neg} deploys a new SLA smart contract, SC_{SLA}, and writes the SLA access parameters in this contract.

4) SLA Violation Monitoring and Penalty Enforcement: The main purpose of this step is to check for SLA violations upon the request of any of the parties in SLA using the agreed KPIs. To do this, the following steps will proceed (the enumeration is based on Fig. 2):

(V1): The customer and provider need to periodically send the KPI's data to a predefined database to store it there. It is important to mention that:

- This process is an automatic operation in which a predefined API, calls the existing system in the provider's and customer's sites to retrieve the data and store it in a predefined database.
- Since the procedure automatically happens with the designed API, we assume that the customer and provider are sending integrated data to Oracle (i.e., the data is not modified).

(V2): Upon receiving the violation check request, SC_{SLA} triggers a specific smart contract designed to connect to Oracle to retrieve data. To do this, the off-chain APP will automatically send a function call trigger to SC_{SLA}. Once SC_{SLA} receives this trigger, it sends a request to SC_{Oracle} by calling `fetchLogFiles()` function. It is important to mention that, each party who requests an SLA violation check, needs to retrieve the KPIs from IPFS and send the requested parameter to verify in the argument of their request. In this step, we assume that the requester party is sending the original value and is not a malicious entity. But, clearly, we can state that verifying the correctness of this value is a straightforward task from the other party's side.

(V3): Assuming that the data is stored in the Oracle (i.e., a centralized or distributed off-chain database) with the proper `timestamp`, SC_{Oracle} sends a request to the database to retrieve the data in the proper time intervals. To do so, SC_{Oracle} emits an event to Oracle's API by sending the following data:

$$< Add_{SC_{SLA}}, Add_{provider}, Add_{customer}, nonce >$$

where $nonce$ is a random id number dedicated to the current request, and the other values are the Blockchain address of the message sender (SC_{SLA}), provider, and customer, respectively.

(V4): Once the Oracle API receives the event, it fetches the customer and the provider's log data from the predefined off-chain URL or database address. When the logs are ready to send to the smart contract, the off-chain entity calls `sendLogFiles()` function of SC_{Oracle} with the following data as input:

$$< Log_{customer}, Log_{Provider}, Address_{SC_{SLA}}, nonce >$$

Once SC_{Oracle} receives the transaction on this function, it first checks the $nonce$ to verify that it is the request's destination and the ID is already on its waiting list. The received logs will be sent to SC_{SLA} for further SLA violation checks if the condition is valid. To do so, SC_{Oracle} calls function `callback()` from SC_{SLA} by sending the logs and the nonce.

(V5): By receiving the logs, SC_{SLA} verifies the ID and checks for violations as follows for both the provider and customer. To do so, different policies will be verified with the value sent by the requester as the agreed KPIs. For instance, assume that based on requester demand, in the SLA terms (i.e., the order defined in Step 1 IV-B2) the provider agreed to offer the service with the following options:

$$Terms_{SLA} = [99.9, -1, -1, 10, -1, 1000, -1]$$

where -1 means that those factors are not included in the SLA. To verify the SLA violation, SC_{SLA} checks if in the provider's and the customer's logs:

$$Log_{customer}.Uptime >= 99.9$$
$$Log_{customer}.CPU\_Capacity >= 10$$
$$Log_{customer}.Storage\_Capacity >= 1000$$

if the conditions are valid, the SLA check is successful. Note that, this example is the terms and conditions that the provider needs to pass, similar policies can be applied to the customer.

(V6): If during the SLA check, SC_{SLA} detects that the logs sent by any of the parties do not satisfy the SLA terms, it sends an event to both parties informing them about the SLA violation. The automatic compensation application is done in the next step.

(V7): After SLA violation validation, the SC_{SLA} will apply the compensation based on the $Terms_{Compensation}$ defined in Step 1 IV-B2. To do so the following changes will be applied:

- *if the $Penalty\_type$ is 1 that indicates the direct payment:* In this case, the $Min\_deposit$ that is already stored in SC_{SLA} by the entity in Step 12 IV-B2 will be automatically sent to the other entities Blockchain wallet. Moreover, the difference between $Penalty\_fee$ and $Min\_deposit$ will be sent to both parties as an event for further action and off-chain payments.
- *if the $Penalty\_type$ is 2 that indicates the change in SLA or provide other services as advantage:* In this case, the SC_{SLA} rewrites the new SLA rules in the final accepted rules of the SLA. Note that this change can be only done by the SC_{SLA} and no other parties have access to change the SLA using this function.

(V8): SC_{SLA} informs both parties about the changes, payments, or other required information.

5) SLA termination: In this step, if the cancellation request is sent by both parties or the SLA is expired, the `selfdestruct()` function of SC_{SLA} will be called. The amount of money blocked into the SLA contract will be sent to the owner's wallet and the contract code will be removed from the Blockchain to free up resources.

V. Evaluation

To evaluate the proposed SLA management approach, we deploy a private Ethereum Blockchain using the Go Ethereum (Geth) framework [21], [22]. Geth is a popular implementation of Ethereum written in Go programming language [1]. A private Blockchain is a type of network developed in one organization or test lab and only the eligible nodes, who are already authenticated, can join in the consensus process. Moreover, this network is not connected to the Ethereum `Mainnet` [2] (or other predefined `Testnets` [3]) and doesn't share the same ledger. To create the private Ethereum network, we first deployed 16 Ethereum nodes and assigned a unique `chainID` to the network. This `chainID` is the unique identifier of the network and ensures that nodes can only interact with other nodes with the same ID. Next, we select *"Clique"* [23] as our network's consensus protocol. Clique is a Proof of Authority (PoA) consensus model [24] in which the initial set of authorized validators is pre-configured and new validators can be added or removed based on a voting mechanism. The validators in PoA have formally approved accounts, and their identity is public [25]. Finally, we save all this information along with other network information, such as gas limits, the initial allocation of `ether`, and so on. in the `genesis block`. A genesis block is the very first block in the Blockchain and we configure this using the `genesis.json` file. Figure 3 represents a portion of our genesis block. We developed the smart contracts using the Solidity [26] programming language and utilized the `web3js` framework to communicate with our private Ethereum network.

The performance analysis of the suggested design is done in three parts: (1) Comparison with existing state of the art, (2) Evaluating the latency of the negotiation process, and scalability of the system in terms of increasing number of concurrent SLA violation check requests, and (3) Gas consumption of different on-chain processes.

A. Comparison with Existing work

Table I compares the proposed method with other state-of-the-art in providing Blockchain-based SLA management solutions in different sectors. To the best of our knowledge, almost no work addresses the reciprocal or mutual SLA in which both parties in the agreements are providing service for each other. In the future generation networks, it is required for MNOs to not only provide service for many businesses but also to receive different types of services from them. So, To address the existing challenge in collaboration management in current cellular networks, we need to provide the possibility of automated reciprocal SLA management.

As shown in Table I, providing the possibility of negotiation through Blockchain is rarely provided by the other methods, while due to the possibility of digital signature and non-repudiation, blockchain-based negotiation can decrease the manual negotiation time. Moreover, the privacy of the entities regarding the SLA KPIs is another subject that is hardly addressed by the other works. Indeed, providing SLA management in an open network such as Blockchain

[1]Build simple, secure, scalable systems with Go, https://go.dev/

[2]The "Mainnet" is the Blockchain's fully implemented and operational public network; e.g., Bitcoin, Ethereum

[3]The "Testnet" is deployed for development and testing uses rather than for real-value transfers and transactions; So, the users can create, design, and test their projects without unnecessary costs.

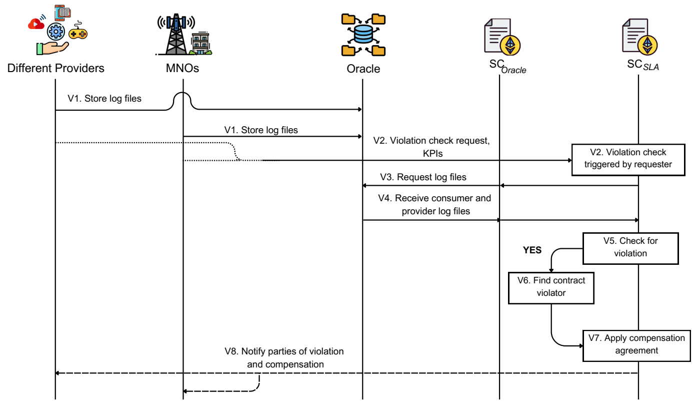
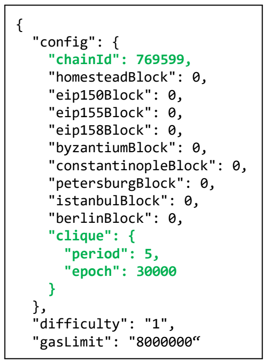

Different Providers
MNOs
Oracle
SC$_{Oracle}$
SC$_{SLA}$
V1. Store log files
V1. Store log files
V2. Violation check request, KPIs
V2. Violation check triggered by requester
V3. Request log files
V4. Receive consumer and provider log files
V5. Check for violation
YES
V6. Find contract violator
V7. Apply compensation agreement
V8. Notify parties of violation and compensation

Fig. 2. A flow diagram representing SLA violation check operated by SLA contract through Blockchain using designed Oracles.

TABLE I
COMPARISON WITH EXISTING SOLUTIONS

Ref. / **Properties**	[13]§	[12]	[14]	[10]	[15]	[16]	[18]	[20]	**This work**
Domain	IoT	CSP**	Fog computing	Cloud	Multi-Cloud	Cloud	Telecom	Telecom	Telecom
Scalability	-	✓	-	-	-	✗	✓	-	✓
SLA negotiation in Blockchain	✗	✓	✗	✗	✗	✗	✗	✗	✓
Reciprocal SLA	✗	✗	✗	✗	✗	✗	✗	✗	✓
Using Oracle for logs	✗	✗	✓	✓	✗	✗	✗	✗	✓
Privacy-preserving	-	✗+	✗	✗	-	-	✗	✗	✓
Blockchain Type++	-	P	P	-	-	P	P	P	P
Blockchain role*	4	1,2,3	1,4	1,2,4	1,2,4	4	1,2,3,4	1,3	**1,2,3**
Consensus model	-	PoM	PoW	-	-	-	-	-	**PoA**

* (1) distributed database to store the SLA terms; (2) distributed method for violation checking; (3) distributed method to apply the penalty for SLA violations, (4) Storing and sharing the log files.
** Communication Service Providers (CSP)
+ In this method some nodes monitor the traffic between the clients and providers to access the SLA's violation.
++ P: Permissioned (private or Consortium), PL: Permission-less (Public).
§ This method does not have implementation.



```
{
  "config": {
    "chainId": 769599,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "berlinBlock": 0,
    "clique": {
      "period": 5,
      "epoch": 30000
    }
  },
  "difficulty": "1",
  "gasLimit": "8000000“
```

Fig. 3. Configuration of genesis block in the system setup

brings the concern of revealing the KPIs of SLA between competitors. To address this issue, instead of storing the KPIs directly on the smart contract, we refer to the secure link to an off-chain database where the encrypted data is stored, and they are only available for the two entities of the agreement. In Table I, compared to the works that use PoW, PoS, or PBFT consensus models, our proposed model can provide higher scalability, is more efficient, and provides higher performance regarding latency. Furthermore, several methods are using Blockchain for storing the SLA monitoring logs; Since Blockchain is an append-only environment, the storage limitation is an ever-existing challenge. So, due to this challenge, our method—using Oracles to connect to the off-chain storage for the log files—can bring higher efficiency regarding storage and log management. Moreover, providing the possibility of automated violation monitoring as well as handling the penalties can not only remove any single point of failure but also decrease the latency of doing the same process in a manual manner.

B. Performance

We evaluate the performance of our proposed system based on two key phases: *SLA creation* and *SLA violation check*.

1) SLA Creation phase: In this phase, we calculate the average time taken to establish an SLA contract. It has 3 main steps: (1) *SLA proposal*, (2) *Negotiation*, and (3) *SLA acceptance*. In an SLA proposal, the consumer generates an

TABLE II
TIME TAKEN BY PROCESSES IN PROPOSED METHOD

Process	Description	Abbr	Time Taken
On-chain	Create Oracle contract	T_{SC_O}	5.53 sec
	Create SLA contract	$T_{SC_{SLA}}$	4.49 sec
	Create Negotiation contract	$T_{SC_{Neg}}$	4.49 sec
	Read SLA request/changes made	T_{Read}	13.26 ms
	Propose changes to SLA	T_{Change}	5.03 sec
	SLA Accepted by both parties	T_{Accept}	5.02 sec
	Violation check request	T_{Check}	1.12 sec
	Oracle requests for data	$T_{Request}$	9 ms
	Sending data to Oracle	$T_{Response}$	4.99 sec
	Read violation check result	T_{Result}	3 ms
Off-chain	Fetch requested data	T_{Fetch}	x*

Note: The time taken for fetching requested data (T_{Fetch}) depends on the type of data, where the data is stored, how it will be retrieved, and the Internet connection. So, we denote it by x.

SLA contract and proposes it to the provider. Then both parties enter the negotiation phase, where each proposes some changes to the contract. Once both parties are satisfied with the changes, they accept and finalize the SLA. Table II contains the time taken by each different procedure in the SLA creation phase. We can see that the time required for this phase is dependent on the number of negotiation steps. So, following formula is used to get the average time for the SLA creation.

$$\begin{aligned} T_{Create} &= T_{SC_{Neg}} + T_{Read} + n * T_{Change} + T_{Accept} \\ &\quad + T_{SC_{SLA}} + y \\ &\approx 14.013 \text{ sec} + n * 5.03 \text{ sec} + y \end{aligned}$$

where n is the number of negotiations, and y is the time when parties do not respond to SLA proposal request.

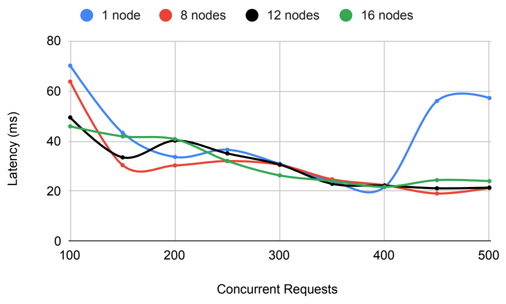


Fig. 4. Scalability of SLA Violation check procedure.

2) SLA violation Check phase: In this phase, we analyze the scalability of our method to handle concurrent requests. For this, we consider the scenario where an automatic request is triggered to check if SLA violation has occurred. Once the SLA contract receives this request, it requests that the Oracle contract fetch the consumer and provider log files. Upon receiving the log files, the SLA contract decides the violation results and notifies the entities. For analysis, we send a different number of concurrent requests (each request represents the scenario mentioned above) to the Blockchain with different number of nodes (1, 8, 12, 16). Figure 4 shows the result of our analysis. The result shows that starting from 400 concurrent requests, Blockchain with 8, 12, and 16 nodes gives stable latency while Blockchain with 1 node starts to act as a centralized system. *Note:* The higher latency during small number of concurrent requests can be credited to the time required to setup the system, which becomes less significant as the number of requests increases. Thus, our method showed scalability by increasing the number of nodes in the Blockchain.

TABLE III
GAS CONSUMPTION OF PROCESSES IN PROPOSED METHOD

Process	Transactions	Gas Used
Contract deployment	SC_O	525083
	SC_{SLA}	1840037
Read SLA	Read SLA request/changes made by each entity	0
Propose SLA changes	Write changes to SLA	99342
Finalize SLA	SLA is approved by both entities	31686
Check for Violation	Violation check request	91663
	Oracle requests for data	0
	Sending data to Oracle	25519
	Read violation check result	0

C. Gas Consumption

This section evaluates the GAS consumption of different function calls and contract deployments. The GAS is the fee that the sender must pay to submit transactions to the Ethereum network. The cost that is mentioned in this part is the cost of sending a transaction of a contract to the Ethereum blockchain (i.e., transaction cost) [27]. The GAS cost is defined in $Gwei$ (i.e., as $10^{(-9)} ETH$). Table III shows the GAS cost in different processes. It is important to mention that in private or consortium Blockchains, the price of sending or processing the transaction is based on the agreement among all participating entities, and no currency is mandatory [25].

VI. DISCUSSION AND FUTURE DIRECTIONS

In this paper, we propose a system using Blockchain technology for SLA management in cellular networks. We present a time-saving solution for conducting SLA-based negotiations using smart contracts. To address the storage challenges in Blockchain, the proposed method stores the SLA-related performance metrics data in an off-chain database and accesses it using Oracles. We also provide two compensation mechanisms, i.e., service credit and financial compensation, to address any SLA violation. The proposed method aims to provide a flexible, automated, and scalable solution for SLA management in the cellular network in which the MNOs have reciprocal agreements with other providers.

We simulated the proposed method by setting up a private Ethereum Blockchain containing a maximum of 16 full nodes (which could represent either service provider or consumer nodes within the Blockchain). Evaluating the scalability of the system involved varying the number of full nodes up to 16 and simulating up to 500 users with concurrent requests. Our findings show that the latency of the system remains largely consistent (i.e., around $20ms$ as shown if Figure 4) even as the number of nodes and/or requests increase, proving that our system is highly scalable.

However, to put our proposed system into practice, several questions need to be addressed; two of the most important ones are to define the role of different actors in the system and to establish who owns the Blockchain in real-world scenarios. The key players in this system are the mobile network operators (MNOs) and service providers. The real-world Blockchain in this scenario, can be configured as a consortium involving MNOs and service providers, determining transaction costs (if applicable), consensus models, storage, and related parameters.

Our work also provides some promising research directions. From a *business standpoint*, we require comprehensive research to analyze the compatibility of Blockchain technology with the market's diverse SLA requirements. This analysis will help to improve our understanding of industry requirements and assist in the gradual integration of Blockchain into the existing economy, avoiding abrupt shifts and disruptive changes.

From a *technical standpoint*, there are several fascinating areas to explore, such as integration with AI/ML solutions, storage/latency optimization, and privacy. Our Blockchain-based system could integrate with AI/ML solutions capable of predicting SLA violations. This integration could shift the violation check from being requested by an SLA party to checks that occur on a situational basis. Given the significance of storage complexity and system latency in various use cases, some research could focus on efficient storage/latency optimization methods, such as Layer2 solutions [28] and chain sharding [29]. Furthermore, there could be some research related to developing lightweight and secure encryption solutions for communicating with off-chain distributed databases, and trustworthy methods to access SLA-based information.

Finally, a future study could focus on developing a consensus model specific to SLA management, as done by the authors in [12]. This achievement could assist in meeting the objectives of various system requirements, including low energy consumption, higher scalability, strong privacy, and efficient resource utilization.

References

[1] Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6G wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, 2019.

[2] F. Ghaffari, "A novel blockchain-based architecture for mobile network operators: Beyond 5G," Ph.D. dissertation, Institut polytechnique de Paris, 2023.

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[4] I. Bashir, *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained*. Packt Publishing Ltd, 2018.

[5] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Blockchain for 5G and beyond networks: A state of the art survey," *Journal of Network and Computer Applications*, vol. 166, p. 102693, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804520301673

[6] K. Yue, Y. Zhang, Y. Chen, Y. Li, L. Zhao, C. Rong, and L. Chen, "A survey of decentralizing applications via blockchain: The 5G and beyond perspective," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2191–2217, 2021.

[7] "Ethereum Whitepaper." [Online]. Available: https://ethereum.org

[8] "Solidity." [Online]. Available: https://docs.soliditylang.org/en/v0.8.17/

[9] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[10] R. B. Uriarte, R. De Nicola, and K. Kritikos, "Towards distributed sla management with smart contracts and blockchain," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2018, pp. 266–271.

[11] B. Koller, "Enhanced sla management in the high performance computing domain," 2011.

[12] N. Weerasinghe, R. Mishra, P. Porambage, M. Liyanage, and M. Ylianttila, "Proof-of-monitoring (PoM): A novel consensus mechanism for blockchain-based secure service level agreement management," *IEEE Transactions on Network and Service Management*, 2023.

[13] A. Alzubaidi, E. Solaiman, P. Patel, and K. Mitra, "Blockchain-based SLA management in the context of iot," *IT Professional*, vol. 21, no. 4, pp. 33–40, 2019.

[14] S. K. Battula, S. Garg, R. Naha, M. B. Amin, B. Kang, and E. Aghasian, "A blockchain-based framework for automatic SLA management in fog computing environments," *The Journal of Supercomputing*, vol. 78, no. 15, pp. 16 647–16 677, 2022.

[15] R. Ranchal and O. Choudhury, "SLAM: A framework for SLA management in multicloud ecosystem using blockchain," in *2020 IEEE Cloud Summit*. IEEE, 2020, pp. 33–38.

[16] K. M. Khan, J. Arshad, W. Iqbal, S. Abdullah, and H. Zaib, "Blockchain-enabled real-time SLA monitoring for cloud-hosted services," *Cluster Computing*, pp. 1–23, 2022.

[17] A. Alzubaidi, K. Mitra, and E. Solaiman, "A blockchain-based SLA monitoring and compliance assessment for IoT ecosystems," *Journal of Cloud Computing*, vol. 12, no. 1, p. 50, 2023.

[18] X. Luo, K. Xue, J. Li, R. Li, and D. S. Wei, "Make rental reliable: Blockchain-based network slice management framework with SLA guarantee," *IEEE Communications Magazine*, vol. 61, no. 7, pp. 142–148, 2023.

[19] F. Javed and J. Mangues-Bafalluy, "Blockchain-based SLA management for 6G networks," *Internet Technology Letters*, p. e472.

[20] E. J. Scheid, B. B. Rodrigues, L. Z. Granville, and B. Stiller, "Enabling dynamic SLA compensation using blockchain-based smart contracts," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 53–61.

[21] "Go ethereum (Geth)," https://geth.ethereum.org/, accessed: 2024-04-07.

[22] "Go ethereum - github," https://github.com/ethereum/go-ethereum, accessed: 2024-04-07.

[23] "Clique PoA protocol rinkeby PoA testnet," https://github.com/ethereum/EIPs/issues/225, accessed: 2024-04-07.

[24] S. D. Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain," p. 11.

[25] F. Ghaffari, E. Bertin, N. Crespi, and J. Hatin, "Distributed ledger technologies for authentication and access control in networking applications: A comprehensive survey," *Computer Science Review*, vol. 50, p. 100590, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013723000576

[26] C. Dannen, *Introducing Ethereum and solidity*. Springer, 2017, vol. 1.

[27] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[28] C. Sguanci, R. Spatafora, and A. M. Vergani, "Layer 2 blockchain scaling: A survey," *arXiv preprint arXiv:2107.10881*, 2021.

[29] G. Kaur and C. Gandhi, "Scalability in blockchain: Challenges and solutions," in *Handbook of Research on Blockchain Technology*. Elsevier, 2020, pp. 373–406.