IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 May 2024, accepted 27 May 2024, date of publication 31 May 2024, date of current version 7 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3408029

RESEARCH ARTICLE

Provisioning Deterministic Finite Automata for QoS Monitoring in Blockchain Decentralized Applications

TAWFIQ ALRAWASHDEH[1], KHALED ALMI'ANI[2], (Member, IEEE), YOUNG CHOON LEE[3], TAHA H. RASHIDI[4], AND ZEESHAN HAMEED MIR[2], (Senior Member, IEEE)

[1]Computer Science Department, Faculty of Information Technology, Al-Hussein Bin Talal University, Ma'an 71111, Jordan
[2]Higher Colleges of Technology, Fujairah, United Arab Emirates
[3]School of Computing, Macquarie University, Sydney, NSW 2109, Australia
[4]School of Civil and Environmental Engineering, University of New South Wales, Sydney, NSW 2052, Australia

Corresponding author: Taha H. Rashidi (rashidi@unsw.edu.au)

ABSTRACT To address the critical need for enhancing Quality of Service (QoS) monitoring in the logistics service delivery domain, this paper introduces a blockchain-based QoS monitoring framework that aims to automate service delivery and dispute resolution processes. Traditional QoS monitoring solutions rely heavily on human judgment and intervention, which may increase operational costs and reduce service reliability. Such behaviour highlights the demand for more efficient and transparent logistics services, which underlines the importance of utilizing blockchain technology's immutable and decentralized nature. The proposed framework in this paper employs a graph-based approach to transform QoS requirements into Deterministic Finite Automata (DFA) format. This strategy simplifies delivery monitoring and the identification of service violations through efficient DFA traversal. By using the Ethereum network as the deployment environment, we demonstrate that traversing a DFA is computationally efficient and reduces operational costs. Extensive experiments were conducted to evaluate the cost-effectiveness of the framework, showing that monitoring a delivery using this framework costs approximately $2.59. This finding underscores the framework's advantages in operational cost optimization compared to traditional human-based methods. Moreover, the decentralized nature of our proposed framework allows customers and businesses to define and monitor QoS parameters jointly. Therefore, the business partners can establish a transparent and trust-based relationship.

INDEX TERMS Blockchain oracle problem, smart contracts, distributed ledger technology, Ethereum, decentralized applications.

I. INTRODUCTION

Due to its transparency and temper-proof characteristics, blockchain technology has the potential to revolutionize the decentralized applications (DApps) domain. In a blockchain, participating entities have an identical copy of the business transactions (ledger) arranged as blocks. Each block's hash value is inserted into the following block to link the blocks together. In blockchain-based applications, business logic is enforced using smart contracts, which are written programs (code) designed to be executed in a decentralized manner. The execution of such contracts starts based on predetermined application-related events, such as registering a new user and the arrival of a new order. The ability of the smart contract to automate business processes has emerged as a promising paradigm to streamline the flow of business transactions.

In several application domains, services (or goods) are expected to be exchanged between participating entities. For instance, in the transportation domain, such services could include the delivery of goods or people to a specific

The associate editor coordinating the review of this manuscript and approving it for publication was P. K. Gupta.

location. The delivery performance metrics of these services are typically monitored to ensure business growth. In this context, the monitoring process aims to gather the required information to obtain Proof-of-Delivery (PoD) [1], [2], [3], [4], [5], [6] and to ensure the satisfaction of the performance metrics. On-time and on-full delivery can be considered the most critical metrics [7]. On-time refers to the requirement of delivering the goods within the agreed-upon time window. Whereby in-full denotes the requirements of delivering the agreed-upon quantity and quality. Additionally, in the event of a dispute, the collected information to monitor the delivery performance metrics is used to settle the dispute.

The complexity of a dispute is mainly influenced by the nature of the provided services (or goods). For instance, in e-commerce applications, services typically involve physical goods. For such goods, breaking the service agreement can be easily detected since the physical condition of the goods can be used to indicate any breach in the agreement. However, when the provided services are not physical or digital, detecting any violation of the service agreement requires further effort. For example, in the transportation domain, a service such as guaranteeing the delivery of goods or people while satisfying conditions such as the used path and storage temperature are not of physical or digital nature. In such a scenario, ensuring the satisfaction of end-user conditions in case of a dispute may require human judgment and the execution of several auditing functionalities.

Utilizing the decentralized nature of blockchain technology to automate Quality-of-Service (QoS) monitoring is expected to enhance services for customers in the transportation sector and improve business flow. Blockchain technology offers flexibility in defining participant roles and rewards. For example, consider trucks equipped with GPS responsible for delivering goods, where commercial logistics management software is not necessarily attached to the trucks. In this scenario, not only the truck owners but also the owners of the logistics software used to validate delivery requirements will receive rewards. Additionally, blockchain's decentralized nature can differentiate between Business-to-Business (B2B) and Business-to-Customer (B2C) cases, acknowledging the distinct differences between these two models. B2B cases often prioritize operating costs, whereas B2C customers focus on timely and high-quality delivery, not necessarily the shortest path. It is important to note that early arrivals might be problematic for customers if they are not home, although this may not be an issue for businesses equipped with digital delivery monitoring mechanisms.

Accordingly, this paper introduces a transformative blockchain-based framework tailored to redefine Quality of Service (QoS) monitoring within the logistics services delivery domain. By utilizing blockchain technology, the proposed framework automates the monitoring of service quality and the resolution of dispute processes. A key factor of the proposed framework lies in its flexibility. It enables participants to define performance metrics that match their needs, aiming to expand the traditional ''on-time, on-full'' delivery concepts [1], [8]. The proposed framework distinguishes between customers and business requirements. Customers and businesses monitor different types of requirements. Furthermore, we introduce a ranking mechanism that evaluates service providers and logistics personnel based on their performance, encouraging a competitive environment that incentivizes high service quality. Toward this goal, the proposed framework employs a Deterministic Finite Automata (DFA) based strategy to monitor and detect any service violation.

In logistics, QoS monitoring involves tracking various performance metrics such as delivery times, transit times, and paths. QoS agreements must be satisfied in a pre-determined sequence (pattern). Such a pattern could simply restrict the pickup and delivery time. In this domain, the conditions of the QoS agreements are expected to monitor a limited, well-defined number of service aspects, such as pickup and delivery time. The limited number of service aspects highlights the potential of representing the QoS agreement as a regular expression, where service aspects and their associated requirements can be represented as strings. For example, the requirement of picking up goods and delivering them before 11 can be represented as PDB11. Using this representation, P refers to the pickup requirement, and D refers to the delivery requirement. Additionally, B denotes a delivery time restriction (11). Accordingly, a DFA can be used to recognize this regular expression and to enforce the QoS conditions. A DFA will comprise several states representing the delivery status (e.g., in transit or picked up). The transition function would map the input (delivery transaction) to the following status of the DFA. The DFA will also include accepting states that represent the fulfillment of the QoS agreement. The traversing process of any DFA is computationally inexpensive, and such behavior underlines the benefits of proposing DFA-based strategies to be executed on a blockchain network. Accordingly, this paper addresses the following research questions (RQs):

- **RQ1**: Is it economically feasible to implement the proposed QoS monitoring framework?
- **RQ2**: What are the computational and operational challenges related to the implementation of the proposed DFA-based monitoring strategy on the blockchain?

In response to these questions, the contributions of this paper can be summarized as follows:

- We develop a blockchain-based QoS monitoring framework for logistics-oriented services.
- We designed a ranking mechanism that evaluates the participants based on their contribution to the delivery.
- We implement the proposed framework in solidity (0.8.17), where Hardhat is used as the development environment. Whereby the Ethereum blockchain is identified as the targeted network.

We have performed several experiments to analyze the proposed framework's cost feasibility. Accordingly, the results have shown that the cost of monitoring a service delivery consisting of five transition states (pickup, delivery,$\cdots$)

is around 2.59$. Considering the proposed framework's flexibility in designing the QoS requirements and the business process automation, such cost can be considered justifiable. Additionally, as we will discuss in the results section, the functionality of the proposed framework is implemented using several smart contracts to ensure code reusability and reduce the cost of re-deployment. Compared to prior studies where human intervention is required, the presented framework offers a more dynamic and cost-effective solution to QoS monitoring, highlighting its potential to revolutionize how logistics services are managed for businesses (service providers) and customers.

The rest of this paper is organized as follows. Section II discusses the most related proposals from the literature, where Section III presents a detailed discussion of the proposed framework. Results and discussion are presented in Section IV, where we conclude the paper in Section V.

II. RELATED WORK

The use of blockchain technology to monitor and facilitate the delivery of goods has been investigated by several proposals [9], [10], [11], [12], [13], [14], [15], [16]. Nandi et al. [11] have shown that adopting blockchain technology in supply chain management is expected to improve compliance and operation-level functionalities. Rohan et al. [2] have also discussed the importance of incorporating an IoT system along with blockchain technology to improve the reliability of the monitoring process. In this line, Demir et al. [17] have highlighted the benefits of using blockchain technology to improve the flow of information between the supply chain participants. Harish et al. [12] proposed a logistic financing blockchain-based framework. As part of the proposed framework, the authors have discussed the benefits of employing blockchain technology to facilitate the monitoring of goods delivery. Moreover, in [13], [14], and [15], the authors have discussed the impact of adopting the blockchain in the international trade supply chain and its benefits toward simplifying the authentication and monitoring components in such a supply chain.

Madhwal et al. [1] proposed a blockchain-based system to monitor delivery performance in the supply chain domain. In the proposed system, a smart contract is deployed by the buyer's address for each order delivery. Accordingly, the seller has to confirm the implementation of the agreement represented in the deployed contract. When public blockchain is used, deploying a smart contract for each order will increase the operational cost. Additionally, in the proposed system, resolving any dispute is not presented as an automated process. In [1], the authors have also highlighted the importance of constructing a negotiation process that helps to establish agreements between the entire supply chain participants. The importance of such a process has also been emphasized in [18] and [19]. Concerning delivery performance metrics, Gunasekaran et al. [7] have discussed in detail each performance metric, where the authors have argued that on-time and in full can be considered the most critical metrics. In [8], the authors addressed the on-time, in-full delivery requirements by proposing the DelivChain framework. In this framework, the smart contract layer is expected to have the needed functionalities to calculate the on-time, in-full matrices.

In [16], the authors proposed a blockchain-based framework architecture for sustainable urban logistics, where the presented framework focuses on delivering food material. The authors adopted four performance metrics that contribute to customer satisfaction in the proposed architecture. These metrics are on-time, in-full, cost and Information transparency. Accordingly, the authors employ a fuzzy analytic hierarchy process to determine each performance metric's weight and calculate the overall score. Additionally, an LSTM model is used to predict customer satisfaction, where the used model is trained on historical available information. Moreover, the authors proposed a compensation mechanism to refund the customer in case of unsatisfactory service. The presented mechanism works on the assumption that trading parties act honestly. In this line, a refund is issued based on manual verification for the four performance metrics.

To address the requirements for providing proof of delivery for software updates, Zhao et al. [3] proposed a blockchain-based protocol to perform such delivery securely and reliably. The proposed protocol addresses privacy issues raised due to IoT device use. To handle the on-time and proof of delivery requirements in the restaurant food delivery domain, Talukder et al. [20] proposed a blockchain-based food delivery system. Using the proposed system, the involved restaurant will be penalized automatically in case of late delivery. The benefits of adopting a blockchain-based system in food delivery have also been discussed by Tokkozhina et al. [21]. Several other proposals have security and performance issues related to the proof of delivery requirements [22], [23]

Unlike the proposals discussed in this section, the proposed framework aims to proactively track the status of delivered services and/or goods. The delivery status is processed and reflected in the blockchain at each step of the service delivery. Accordingly, any QoS violation is detected on time, which helps automate the violation dispute component. Such mechanisms can also help by allowing businesses to act responsibly about any service violation to ensure customers' satisfaction. Furthermore, inspired by the predictable steps of the service delivery, the proposed framework aims to establish a unified lightweight delivery tracking component that customers and businesses can use.

III. THE PROPOSED FRAMEWORK

Figure 1 shows the conceptual workflow of the proposed framework functionalities. The user (customer/service provider) must register with the framework through the registration component to use the provided functionalities. Registered users (customers and service providers) are expected to submit the logistics service details to the

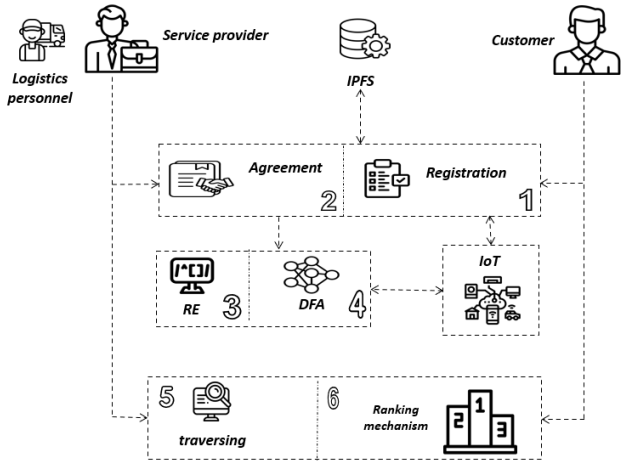

FIGURE 1. The proposed framework conceptual workflow.

framework for monitoring. Accordingly, using the service component, the QoS requirements for any active logistic service can be submitted by the service providers. The provided QoS requirements are presented as a regular expression, which is used to build the DFA that recognizes the utilized regular expression. To track the fulfillment of the QoS requirements, an IoT component is employed to gather any requested external data in an event-driven fashion. Additionally, the monitoring component is used to rank the logistic entities and penalize QoS violators.

A. REGISTRATION

To use the proposed framework, users must register as customers or service providers. User can link their account to an existing Ethereum address during registration. If a new Ethereum address must be created, the end-user must submit identification documents, which will be linked to the newly generated Ethereum address. All identification documents will be stored off-chain on the InterPlanetary File System network (IPFS[1]). A registered service provider can (de-)register participants as data sources in the proposed framework. These actors represent the logistic personnel (e.g., drivers) and any registered devices from the IoT component.

B. SERVICE COMPONENT

Figure 2 illustrates the steps performed by the service component. Initially, to activate QoS monitoring requests, the service agreements must be signed by the service provider or both the service provider and the customer. The identity of the required signing parties depends on the nature of the QoS constraints, internal or external. Internal requirements are mainly imposed by the service provider, aiming to monitor the trading aspects that impact the operational cost. For instance, a transportation company may introduce QoS constraints to monitor the drivers' efficiency, which could cover the drivers' used path. The service provider must sign internal constraints. External constraints represent the QoS requirement that the service provider has guaranteed to the customer, and therefore, the service provider and the customer must sign the external constraints. Additionally, concerning external constraints, the service provider is expected to deposit an insurance fee to the collection account. This is an Ethereum account the proposed framework uses to automate reimbursement in case of service violation. The external constraints are expected to cover the pickup and delivery time. Other service-related constraints, such as the used path and the temperature of goods vehicles (trucks), are also part of the external constraints. Once the QoS requirements are approved (signed), an agreement number (AN) for the submitted QoS requirements is issued. For each active QoS agreement, the submitted QoS requirements are represented as regular expressions that are then transformed into DFA.

To further clarify the processes employed by the proposed framework, we will use the example shown in Figure 3 throughout the rest of this section. In this example, the service provider will arrange for goods to be picked up from customer A and delivered to customer B before $20:00$ on a pre-determined date (d). Once the service provider and/or the customer sign the agreement, an agreement number will be issued. Then, the service provider is expected to determine the logistics personnel involved in the service delivery and the steps they must perform to fulfill the service agreement. The service provider will also register the logistics and the tracking devices that will be used as part of this delivery. In this example, we assume that driver D will handle this delivery, whereas his logistic device will also handle the tracking responsibility (time and date).

1) QOS CONSTRAINTS AND REGULAR EXPRESSION

Using the proposed framework, each QoS requirement is represented as a set of characters and symbols that define a pattern. Such patterns capture all logistics transactions that result in satisfying the QoS constraints. For instance, let us assume that the QoS constraint for a logistic service is picking up goods before 8:00 am and delivering them before 12:00 pm. In such a situation, if we refer to the picking-up constraint as A and the delivering constraint as B, the expression that captures the QoS constraints can be represented as follows: AB. This paper represents the QoS constraints as a set of regular expressions. The notations of logistics transactions must be pre-determined to facilitate such a process. Table 1 summarizes all the notation and symbols used to construct the regular expression. However, the service provider can add a new notation to the defined list in Table 1, where the newly added notation can be used for future service agreements. The presented table consists of two parts: transactions and constraints. The transactions capture the logistic step that the service delivery is expected to encounter, where the constraints represent the QoS requirement associated with each transaction. The

[1]https://ipfs.tech/

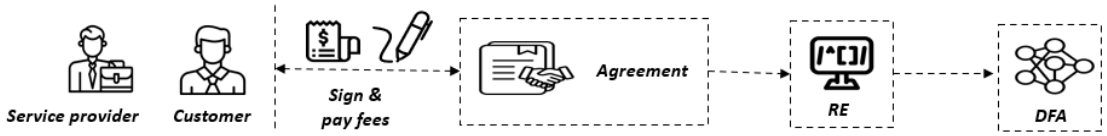

FIGURE 2. The service component processes flow.

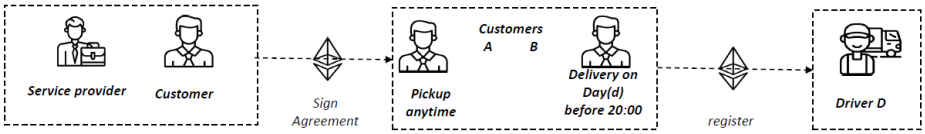

FIGURE 3. Example: agreement signing and logistic personnel registering.

table shows that the in-transit transaction has an optional field $[*, +]$. If the star symbol is selected, zero or more in-transit transactions are expected to occur consecutively. In comparison, the plus symbol denotes the possibility of one more in-transit transaction. The in-transit transaction must occur once if neither of these symbols is used. This table works as a lookup table, and it is small in size. Thus, it can be implemented off-chain or on-chain.

To simplify the process of repressing the QoS constraints as a regular expression, we divide the roles associated with each regular expression into two types: (1) sequence and (2) constraint roles. Each regular expression has a single sequence role and one or more constraint roles. A sequence role represents the order of the events (transitions), and it is represented as follows:

$$r \rightarrow [t_1][t_2][t_3]\cdots \tag{1}$$

r is a root notation that refers to starting the regular expression. The presence of each transaction in the sequence role depends on the QoS constraint. Additionally, we assume that order dependence exists between the transactions. For instance, goods cannot be delivered before performing a pickup transaction. When QoS requirements require a set of transition transactions to be performed several times, such requirements can be divided into regular expressions. For instance, if the imposed requirement is t_1, t_2, t_1, t_2. Such requirements can be represented as two regular expressions, each denoted as t_1, t_2. The employed division strategy does not impact the overall performance since regular expressions are closed under addition. Constraint roles represent the requirements associated with each transaction, and it is represented as follows:

$$t_i \rightarrow [c_1][c_2]\cdots. \tag{2}$$

Each transaction (t) can be associated with one or more constraint roles. At the same time, each constraint role is represented as a function that takes the requested QoS requirement as input and returns a true/false value. Once all internal and external requirements are specified as regular expressions, they are sent as input to the DFAs construction process.

To clarify this process further, let us revisit the example shown in Figure 3. Now, the requirements will be processed to build the regular expression representation. We only assume that external constraints are used to simplify the presented example. The sequence role for the presented example is as follows:

$$r \rightarrow PID$$

In this example, the service provider adds the in-transit state (I) since the delivery process is expected to be in one or more transit states. The constraints roles represent the location of the customers and the requirements of performing the delivery before $20:00$ and these constraints roles are represented as follows:

$$P \rightarrow l(A)$$
$$D \rightarrow l(B)T(20:00, B, d, on)$$

2) DFA CONSTRUCTION

A DFA is a 5-tuple $(Q, \sum, \delta, q_0, F)$, where Q represents a finite set of states and $\sum$ represents the input alphabet. The input to the transition function δ represents a state and an input symbol, whereas the output is a state. q_0 is the initial state, and F represents the accepted states. In situations where the DFA is not in an accepted state after processing the entire input, the processed input is rejected by the DFA. Otherwise, the processed input is considered accepted.

In this work, the states (Q) represent the logistics transition transactions (e.g., pickup, delivery, etc.). Whereas the transition function δ captures the QoS requirements. The proposed framework employs a graph-based strategy to transform the specified regular expression to the equivalent DFA (Algorithm 1). The graph used to represent a DFA consists of $|Q| + 1$ vertices representing the states and a virtual vertex representing the root of the regular expression (initial state q_0). Starting with the sequence role, a direct edge between the service transactions is added based on

TABLE 1. Transactions and constraints notation summary.

	Description	Notation
Transition transactions (t)	Pickup	P
	Delivery	D
	In-transit	$I^{[+,*]}$
Constraint (c)	Time and date constraint	$t(time[B,A], date, [B,A,on])B : before, A : after$
	Path associate constraint	P(list of locations)
	temperature constraint	$D(degree, [above, below])$
	Location constraint	$l(location)$

Algorithm 1 transformREtoDFA

Data: *RE*: regular expression, *LT*: notations list, *AN* :agreement number
Result: *DFA*

1 $s \leftarrow getSequanceRole(RE)$
2 $C \leftarrow getConstraintRoles(RE)$
3 $DFA \leftarrow < V, E >$
4 $V \leftarrow createVertics(LT)$
5 $T' \leftarrow extractVistingSequance(s)$
6 $E \leftarrow createDirectEdges(DFA, T')$
7 $lableInitialstate(DFA, T')$
8 $lableFinalstate(DFA, T')$
9 **for** $c_i \in C$ **do**
10 $\quad sn \leftarrow registerIoTevent(c_i, AN, S)$
11 $\quad attachSNtoVertix(sn, DFA, c_i)$
12 **return** *DFA*

the appearance order in the regular expression. For instance, if the regulation expression for the root notation is described as follows: $r \rightarrow PD$, a direct edge is established from vertices r to vertices P, followed by a direct edge between vertices P and D (lines 5-6). If any of the transactions in the regular expression has a repeat symbol $(*, +)$, a self-loop direct edge will be added on the corresponding vertex. This work assumes that such a symbol can only be associated with the in-transit transaction. However, such an assumption can be eliminated if a new transaction with a repetition nature is added to the lookup table (Table 1). The constraint roles will be processed once the sequence role is mapped (line 9). constraint roles require the involvement of the IoT component. Each constraint role is submitted to the IoT component, the agreement number (AN), and the script to be executed. The IoT component returns a list of events distinguished by their serial numbers. Each constraint's (event) serial number is added to the vertices associated with it. For instance, if the delivery must be before 8:00, the IoT component will have a unique event number that captures this requirement. If such an event is executed, the IoT will return true if the time is before 8:00. Otherwise, it will return false.

The computational complexity of the DFA construction process can grow significantly while increasing the number of sequence and constraint roles. Such behavior may increase the operational cost since computations are not free of charge in the Ethereum network. However, if a high number of roles are expected, the DFA construction process can be implemented off-chain. In this paper, the DFA is maintained and stored on-chain. However, the process of constructing the DFA is performed off-chain.

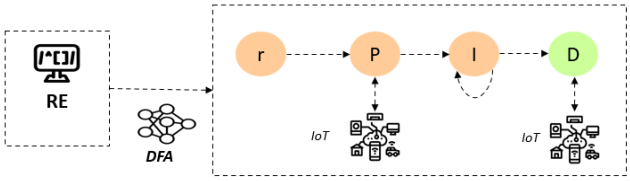

FIGURE 4. Example: DFA construction.

Regarding the example shown in Figure 3, the DFA construction process uses the obtained sequence and constraint roles to obtain the equivalent DFA. During the construction of the DFA, the registerIotEvent() will be called three times to register two location events and one delivery event. Accordingly, the serial numbers of these events will be attached to the corresponding vertices in the DFA graph, as shown in figure 4.

C. IOT COMPONENT

The proposed system supports various business-related processes involving several IoT devices, such as sensors, PDAs, and mobile phones. These devices serve as transaction feeders and initiators. Based on their roles, IoT devices can be categorized into trading, logistics, and tracking devices. Customers and service providers use trading devices to finalize the QoS agreement and manage other administrative tasks. Any device supporting phone and/or web applications can perform such tasks where logistics personnel use PDAs or/and any other type of logistics devices. Tracking devices such as sensors and GPS are utilized to monitor the fulfillment of QoS requirements.

Trading devices host applications for customers and service providers. Thus, they are expected to interact with the proposed framework without significant restrictions, as a registration process must be completed before using these applications. Logistics devices are part of the service provider's infrastructure and can be registered or deregistered

Algorithm 2 registerIoTevent

Data: C: list of conditions); AN: aggrement number; S: list of conditions scripit

1 **for** $c_i \in C$ **do**
2 $\quad T'_c \leftarrow identifyTaskCode(c_i)$
3 $\quad d' \leftarrow identifyDevice(T'_c, AN)$
4 $\quad e' \leftarrow createEvent(d', AN, s_i)$
5 $\quad sn \leftarrow genarateSN(e')$
6 $\quad$ add $< sn, e' >$ to L_{event}

by the service provider to interact with the system. Tracking devices can be owned by the service provider or any other external participant. For example, a service provider might lease tracking devices from other entities. Thus, the responsibility of registering these devices is assigned to the service provider. These tracking devices are registered to interact with the framework as needed.

Registered tracking devices interact with the framework using a publish-subscribe event approach, as described in Algorithm 2. Each tracking device is identified by the combined AN and task code (t_c). Each task code corresponds to a single constraint role (Table 1). During the DFA construction process, a request is submitted to the IoT component for each constraint role to create and register corresponding events. A request comprises the AN, a list of conditions expressed in the constraint role, and a script to be executed for each condition. The provided constraints list is then used to map and identify the requested task code for each condition. Using the AN and the obtained task codes, the IoT component identifies the tracking devices needed to address the constraint role under consideration. For each identified tracking device, an event outlines the requested task (e.g., location and temperature). The IoT component returns a list of events for each processed constraint role, with the serial numbers of these events attached to the request-initiating vertex in the constructed DFA.

Typically, in an IoT-based application, performing threat modeling is a crucial step in identifying all vulnerabilities and determining countermeasures to overcome the risks imposed by such vulnerabilities. However, performing threat modeling is considered outside the scope of this paper, and we assume that the IoT devices are trustworthy. Nevertheless, the integrity of the provided data by the IoT devices can be ensured using oracle components [24], [25].

D. MONITORING COMPONENT

Figure 5 illustrates the flow of processes employed by the monitoring component. From a business perspective, logistics devices work as event initiators that capture the current status of active service (e.g., transit, delivering). Accordingly, once a new transaction is received from the logistics devices, the monitoring component traverses and updates the status of the constructed DFA. Besides DFA traversing, the monitoring component hosts the ranking mechanism that updates the rank of the participated entities based on the provided QoS. This mechanism also works by penalizing any service violator entity.

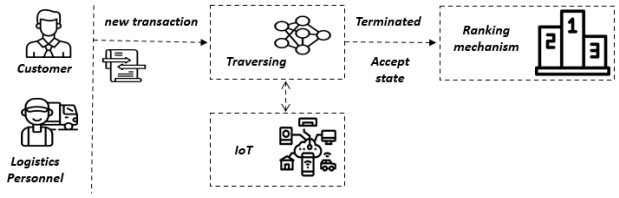

FIGURE 5. The monitoring component processes flow.

1) DFA TRAVERSING

Algorithm 3 illustrates the DFA traversing steps. The traversing of a DFA starts with its initial state (r). At any stage, this component expects a new transition transaction to be received, where each transaction is associated with two values, AN and the transaction type (e.g., pickup or delivery). If there is a direct edge between the current state and the new transaction state, the new state will be considered as the current state. If such an edge does not exist, a QoS violation will be reported (lines 2-5), and the process of traversing the DFA will be terminated since having an unexpected transaction can be considered a major issue.

When the new state has role constraints (line 12), the registered event in the IoT component will be triggered to evaluate the constraint. If the IoT component returns false, a QoS violation will also be reported (lines 13-14). In such a situation, the DFA traversing will not be terminated since an overall evaluation for the violated requirement is expected to help the service provider and the customer.

Once the last state (transaction) in the sequence role is reached (accept state), the traversing of the DFA under consideration will be considered complete. At that stage, the service provider and the customer will have full access to the reported violations (line 16). Meanwhile, the ranking mechanism uses the reported violations to evaluate the satisfaction of the QoS constraints. Moreover, each DFA is associated with a pre-determined time deadline. Suppose such a deadline is reached, and the accept state for the DFA under consideration has not been reached. In that case, the users (customers and/or service providers) can terminate the DFA traversing. In such a situation, a major QoS violation will be reported.

With respect to the example shown in Figure 3, during the fulfillment of this delivery, the first expected transaction is the pickup transaction (P). Once the pickup state is reached in the DFA, the IoT component will be triggered to confirm the pickup location ($l(A)$). After picking the goods, the in-transit transaction is expected to be received at least once. When the DFA's current state becomes the in-transit state, receiving any new in-transit state will not change the DFA's current state. The accept state in the presented DFA is the delivery state

Algorithm 3 DFATraversing

```
Data: AN: aggrement number; TT: transaction type
Result: R_v: violation report
1  DFA ← retrieveDFA(AN)
2  if checkTransaction(DFA,TT) not valid then
3      e' ← createTermationLog(DFA, TT)
4      add e' to R_v
5      rank(AN, R_v)
6  else
7      updatedDFA(DFA, TT)
8      s' ← getCurrentState(DFA)
9      if hasConstraint(s') then
10         event ← callIoT(s')
11         if event is false then
12             e' ← createMVLog(DFA, TT, s')
13             add e' to R_v
14     if s' is accept state then
15         rank(AN, R_v)
16 return R_v
```

(*D*), and once it is reached, the IoT component will be called twice to confirm the delivery location and time. The last step in this process is called the ranking mechanism. Using this mechanism, since no violation is reported, the ranking of the service provider will be updated, and the service delivery will be considered fulfilled.

During the service delivery, several unplanned events could occur. For instance, upon reaching the customer's *B* location, the customer was not present to receive the goods. In this situation, upon contacting the customer *B*, the delivered goods could be left on the premises. If this is not an option, a new delivery request could be arranged where the customer *B* has an additional charge. In this situation, the location of the driver and any established communication will be performed and authenticated using the driver's logistic device. Moreover, if the delivery deadline is reached and the delivery status is still open, the customer can issue a service termination request that the ranking mechanism will handle.

2) RANKING MECHANISM

This mechanism penalizes any service violator based on reported QoS violations. Such mechanisms help the service provider track the logistic personnel's performance. The customer benefits from this mechanism by automatically receiving reimbursement in case of confirmed service violations.

Once a DFA has reached an accept state or been terminated due to a major violation, the associated QoS report is submitted to the ranking mechanism (Algorithm 4). The actions performed by this mechanism depend on the type of violated requirements (internal or external). As mentioned, the service provider submits the internal requirements to capture the operational cost and performance. The external requirements capture the guaranteed QoS requirements of the service provider. Accordingly, violating the internal requirements impacts service providers, whereas violating the external requirements impacts service providers and customers.

If the QoS shows confirmed violation incidences, the rank of the involved logistics personnel or service provider is updated to reflect the reported violation. Violating internal requirements results in updating the logistic personnel rank (lines 5-9), whereas violating the external requirements impacts the service provider rank (lines 12-14). We employ a weighted strategy to determine the rank of each participant (logistics personnel or service providers). Accordingly, each participant is associated with a set of counter variables $V = \{v_p, v_d, v_l, v_t\}$ representing the constraints shown in Table 1. v_p denotes the path counter, where v_d captures the time and date counter. v_l representes the location counter and v_t denotes the time counter. The value stored in a counter variable represents the number of deliveries the involved participant has performed without violating the targeted constraints. For instance, if the delivery of a service that involves a path constraint has been performed without any violation, the v_p value for the participant is incremented by one. Regarding the external requirements, the same process will be applied; however, the service provider's constraint variables will be updated in this case. Accordingly, the rank of a participant a (logistics personnel and/or service provider) is calculated as follows:

$$Rank(a) = \sum_{i \in \{p,d,l,t\}} \frac{v_i}{d(i)} \times w_i \tag{3}$$

where $w_i = [0, 1]$ represents the weights for each performance metric and $d(i)$ represents the total number of deliveries participant (a) has performed with a constraint of type i. The highest value for a participant rank is $|C|$, where C is a set of defined constraints in the lookup table (Table 1). This value is achieved when each constraint weight is assigned to one.

Using on-time and in-full metrics to evaluate the quality of delivery has been extensively studied in the literatures [1], [8], [26], and [27]. In this work, we expand the evaluation metrics by allowing the user (customer or service provider) to introduce new metrics, such as path and temperature. Furthermore, this work adopts a weighted mechanism to calculate the users' ranks, enabling the framework to re-rank based on users' interests (customers or service providers). Other mechanisms could be used to calculate the rank [1], [8], [26], [27]. However, this paper adopts a weighted version to accommodate more general application scenarios.

Regarding the reimbursement process, the customer can be entitled to full or partial reimbursement if external requirements are violated. The customer is entitled to full reimbursement if the violation terminates the DFA traversing process (major violation) (line 16). Moreover, the customer will receive the security fee deposited by the service

Algorithm 4 rank

Data: AN: aggrement number; R_v: violation report;W: weights

1 $s' \leftarrow getServiceProvider(AN)$
2 $c' \leftarrow getCustomer(AN)$
3 **if** $|R_v| > 0$ **then**
4 **if** *ANN is internal* **then**
5 $P \leftarrow determineLogisticP(AN, R_v)$
6 **for** $p_i \in P$ **do**
7 $\{v_p, v_d, v_l, v_t\} \leftarrow getCounters(p_i)$
8 $updateCounters(v_p, v_d, v_l, v_t, R_v, p_i)$
9 $updateRank(p_i, W)$
10 **else**
11 $\{v_p, v_d, v_l, v_t\} \leftarrow getCounters(s')$
12 $updateCounters(v_p, v_d, v_l, v_t, R_v, s')$
13 $updateRank(s', W)$
14 **if** R_v*has termination* **then**
15 $reimburse(c', AN, F)$
16 **else**
17 $reimburse(c', AN, P)$
18 $close(AN)$

provider. When a minor violation occurs, the customer receives a partial reimbursement representing the security fees deposited by the services provider (line 18).

IV. RESULTS

This section describes the experiments' settings, including the implementation details and the evaluated scenarios. Then, we discuss in detail the results of the performed experiments.

A. EXPERIMENTAL SETTINGS

The proposed framework's design and implementation followed the experimental methodology (Figure 6). This methodology begins with the design phase, during which the framework's initial design is created. Then, in the implementation phase, the framework is implemented in Solidity[2] (0.8.17), where Hardhat[3] is used as the development environment and Ethereum as the targeted network. The interaction with the Ethereum network is chargeable, where each participant pays for the computation and storage used by his/her interaction. Accordingly, each participant address must have enough Ether balance to execute the requested functionality. The performance of each proposed algorithm is evaluated in the experimentation phase through an extensive set of experiments. Based on the outcomes of the experimentation phase, the structure of the proposed framework is re-optimized. The final phase of the adopted methodology is the validation phase, where the performance

[2]The source code can be obtained by contacting the first author.
[3]https://hardhat.org/

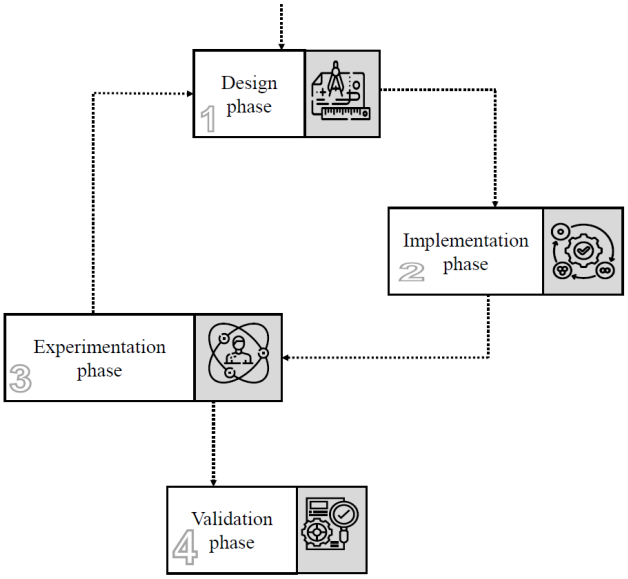

FIGURE 6. The steps of the adopted experimental methodology.

of the proposed framework is evaluated to investigate its efficiency.

The proposed framework is implemented as four smart contracts: the Trade, DFA, DIDRecord, and IoT. Splitting the framework functionality into four contracts aims to ensure the upgradability of the proposed framework. At any time, the code for one of the used contracts can be updated without impacting the others. In this case, the updated contract must be redeployed, whereas the other contract should only update the stored address for the redeployed contract. Table 2 summarizes the main functionalities of the four contracts. The implementation of the IoT contract depends on the application scenario. Accordingly, examining the contract's performance is outside this paper's scope. However, for completeness, the main functionalities of this contract are implemented.

Figure 7 illustrates the expected interaction between the participants (seller and buyer) and the implemented contracts to activate a new service request in the proposed framework. Once the buyer approves the service details submitted by the seller, the Trade contract is called to construct and store the corresponding DFA. To simplify the implementation of the "createDFA" function, the actual DFA is constructed off-chain and passed to the createDFA function as a 2-dimensional array. The IoT contract is called to create the requested events during this construction. Using the DIDRecord contract, the seller registers the logistic personnel, who are expected to provide status information Regarding the newly activated service. Figure 8 shows the triggered processes once a new transaction is received from logistic personnel. The presented diagram represents the main scenario where no QoS violation has occurred. A new transaction's arrival triggers the DFAGraph contract's traverse function. When the new state is associated with an

TABLE 2. Smart contracts main functionalities.

Contract	Function	description
Trade	processInput	process a new transaction recived from logistic personnel or any participants
	createService	create service and perform all of the registeration activities
	processViolation	is used by the processInput function to handle any QoS violation
DFA	createDFA	construct the DFA graph
	traverse	process the newly received transaction
	isAccepted	check if the constructed DFA has reached an accept state
	payDataSources	process the payments for the trustworthy group of data sources
DIDRecord	activateService	activate a new service
	closeService	close a service
	registerLPersonnel	give logistic pernonel access to a given service
	deregisterLPersonnel	remove logistic personnel access to a give service
	rank	update a given logistic personnelrank
IoT	registerEvent	register and store new event
	runEvent	execute a pre-registered event

event, the IoT contract is called to execute the event. Once the traverse function is executed, a confirmation is propagated back to the logistic personnel device.

To reduce the number of stochastic variables, the experiments presented in this section assume that no service violation occurs. Additionally, the upper bound for the number of transactions is three, the transactions presented in table 1.

B. COST ANALYSIS AND DISCUSSION

The experiments presented in this section aim to investigate the cost of using the proposed framework. Accordingly, in addition to the cost incurred by using each smart contract functionality, we aim to investigate the impact of the number of QoS requirements and the number of involved logistic personnel on the operational cost.

Next, we ran the experiments to explore the cost of using the framework functionalities. At the same time, we assumed that the seller and the buyer had agreed and submitted the QoS requirements for a single service. Accordingly, the seller registered single logistic personnel to update the service status during delivery. For the sake of this experiment, we assume that the registered logistics personnel is expected to update the status once (one constraint) during service delivery. Furthermore, we ran the experiments ten times and populated the framework with n of services in each experiment. n is a random value selected to be in $[1 : 100]$. Populating the framework helps by capturing the impact of an operational lifetime on cost.

Participants are charged to interact with the framework functionalities based on the computational and/or storage requirements of the called functions. Such cost is typically represented as a ''gas'' fee, and for each function call, the consumed gas fee is eventually converted to Ether. Table 3 shows the cost associated with each performed action (transaction), where the used price for gas is 14.3 Gwei, and one Gwei is equal to Ether^{-9}. To obtain the cost in USD, we used the exchange rate provided by Coinbase.[4]

From the table, we can see that the cost of deploying the contracts is noticeable. However, this is an initialization cost since the contracts are not expected to be redeployed often. In the case of a contract redeployment, breaking the framework functionalities into four contracts will reduce the redeployment cost. Regarding the cost of execution of the createService function, this cost also includes the cost of the following functions (from the DFA, IoT, and Trade): activateService, createDFA, and the registerEvent. Combined with the processInput cost, we can see that the total charge of creating and traversing the service during the service delivery is around 3$. The processInput cost includes the cost of the traverse, isAccepted and payDataSources functions. The number of involved constraints and logistic personnel typically impacts such costs. However, the trade supply chain for the addressed service in this work is not expected to involve many logistics personnel and constraints. Therefore, such a cost is doable.

To investigate the impact of the number of transactions (QoS requirements) on the overall monitoring cost, we ran experiments while varying the number of transactions. Figure 9 shows the results of this experiment, where we assume that single logistics personnel handles the delivery. As we can see from the results, increasing the number of transactions has a semi-linear impact on the cost. To clarify this behavior, let us reconsider the mechanisms employed by the proposed framework. Increasing the number of QoS requirements (transactions) mainly impacts the performance by increasing the number of calls to the traverse function. This behavior is established because the DFA graph size (number of nodes) is fixed and bounded by the number of transactions in the lookup table (Table 1).

[4]https://www.coinbase.com/

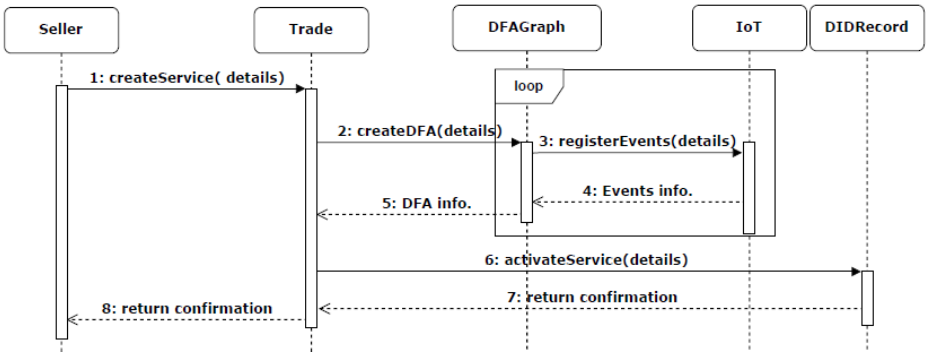

FIGURE 7. Sequence diagram showing the steps of activating new service.

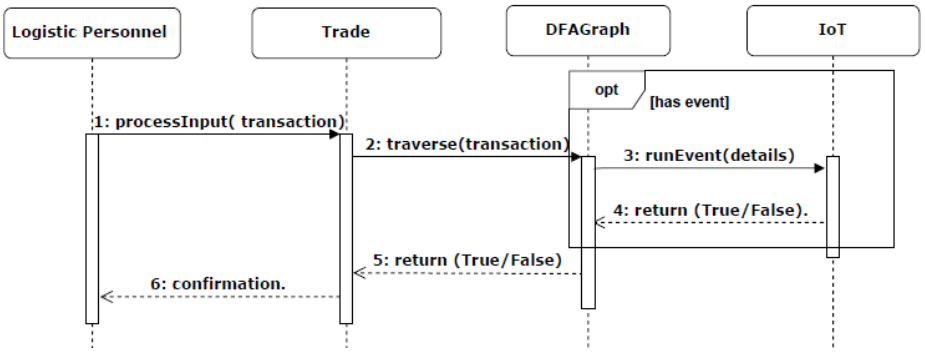

FIGURE 8. Sequence diagram showing the steps of processing the new transaction.

TABLE 3. Cost of transactions.

Transaction Type	Gas Used	Cost (ETH)	Cost (USD)
processInput	11972.5	0.000171	0.28
createService	72365	0.001034	1.73
contracts deployement	2,708,299	0.038728	64.8
registerLPersonnel	33331.5	0.000476	0.79
deregisterLPersonnel	13480	0.000192	0.32

Next, we evaluated the impact of the number of involved logistic personnel on the overall cost. Accordingly, we conducted experiments while varying the number of logistic personnel. Figure 10 presents the results of this evaluation, where the number of used transactions (meeting the Quality of Service requirements) is five. The figure indicates that an increase in the number of involved logistic personnel leads to a slight increase in the operation cost. This behavior is expected, as increasing the number of logistic personnel primarily augments the costs associated with their registration.

During the operational lifetime of the framework, logistic personnel are expected to serve multiple services before being deregistered. Accordingly, in the next experiment, the cost incurred by registering the personnel is counted once, and 100 services are created and used for each report result point. Figure 5 shows the average cost of executing a monitoring request on the proposed framework, where we have varied the number of QoS requirements. The figure shows that the average cost has dropped significantly to become $2.59 compared to Figures 9 and 11, where the average cost is shown to be around $3.8 and $6; respectively. The average cost can be even reduced further if we allow the initialized service-related variables for a closed service to be reinitialized and used by other services. Considering the proposed framework's advantages, the average reported cost of $2.59 can be considered doable. Moreover, the cost of executing the framework can be reduced significantly

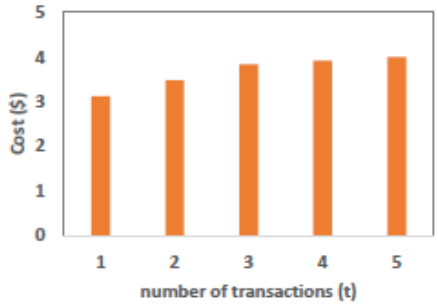

FIGURE 9. Impact of the number of transactions on the overall cost.

FIGURE 10. Impact of the number of logistic personnel on the overall cost.

FIGURE 11. Impact of the number of transactions on the average cost.

by employing other deployment environments with cheaper cryptocurrency exchange rates, such as the Avalanche blockchain,[5] which is known for its support for the designing and the implementation of DApp. In such a scenario, the proposed framework's cost can be reduced to significantly less than $1, ensuring the proposed framework's economic feasibility (**RQ1**).

Next, we discuss the cost of using the employed framework, where we compare the cost of using the framework against a human-based mechanism. This mechanism assumes an employee will oversee the delivery process and resolve any service dispute that has been raised. In such a configuration, we assume that the employee will require s minutes to monitor the delivery of a service that consists of 5 transactions. In addition, we assume the probability of a service violation for such a service is $e\%$, where the time required to resolve the service violation is d minute. Accordingly, the average time ($t_{average}$) an employee requires to handle a single delivery is $s + e\% \times d$. In this line, if the cost of using the proposed framework to handle such a service is $\leq t_{average}$, the framework can be considered efficient in terms of cost. Besides the salary of the employee and the expected gas cost, the evaluation depends on the values of s, $e\%$, and d. For instance, let us assume that the employee is paid $100 for eight hours and values for the three variables are 5, 25%, and 30, respectively. In this example, the cost of using an employee to handle a single service delivery is around $2.6 compared to $2.59 using the proposed framework. This further highlights the efficiency of the proposed framework in terms of cost.

V. DISCUSSION

Employing decentralized QoS monitoring empowers the framework to support transparency and allow users (businesses and customers) to determine their monitoring requirements. The cost of using and operating the proposed framework is mainly influenced by the number of transactions, logistics personnel involved, and the nature of the IoT events. From a service delivery perspective, increasing the number of transactions and logistics personnel is expected to increase the cost linearly. This behavior is accepted since the number of transactions and logistics personnel are indirectly bounded by the number of transactions in the lookup table (Table 1). Furthermore, the proposed framework's cost can be reduced significantly by moving the DFA traversing functionality off-chain. Services' DFAs have the same number of nodes (transactions) and this simplifies the process of moving this functionality off-chain since DFA's edges can be stored on-chain using smaller space compared to the entire DFAs. The ability to shift part of the functionalities off-chain helps in addressing any raised computational and operational challenges (**RQ2**).

Regarding IoT events, the nature of these events is application-dependent. From a blockchain perspective, event registration and results are stored on-chain, while the actual event evaluation is performed using an external call. This behavior allows businesses (service providers) to register and de-register IoT devices as required

VI. CONCLUSION

The presented discussion has highlighted the benefits of the proposed framework tailored for the logistics service delivery domain, where precision, efficiency, and reliability are crucial requirements. The proposed framework distinguishes itself by providing customers and businesses with the ability to define their own QoS monitoring requirements, employing a lightweight DFA-based mechanism to monitor QoS constraint satisfaction.

The demonstrated empirical evidence illustrates the benefits of adopting the proposed framework, with the cost of monitoring a delivery comprising five transactions at

[5]https://www.avax.network/

an affordable $4. The potential penalties and reputational damage arising from non-compliance with QoS standards underscore the financial viability of the proposed framework.

DFA is memoryless and, therefore, cannot capture constraints that require memory capabilities. For instance, DFA cannot be used to maintain an average temperature below a predetermined threshold, as computing the average temperature requires memory capabilities not supported by DFA. Accordingly, as part of our future work, we plan to study the applicability of employing more powerful automata, such as pushdown automata. Additionally, we aim to explore the adoption of application scenarios to investigate the application-dependent component (IoT component) performance. Such adoption is also important to quantify the proposed framework's benefits. Additionally, we plan to investigate the use of profiling methods to predict the performance of logistics personnel involved in the delivery.

REFERENCES

[1] Y. Madhwal, Y. Borbon-Galvez, N. Etemadi, Y. Yanovich, and A. Creazza, "Proof of delivery smart contract for performance measurements," *IEEE Access*, vol. 10, pp. 69147–69159, 2022.

[2] R. Rohan, S. Varma, and M. Sivaramakrishna, "Blockchain-based solution for proof of pick-up of a physical asset," in *Proc. Int. Conf. Mainstreaming Block Chain Implement. (ICOMBI)*, Feb. 2020, pp. 1–7.

[3] Y. Zhao, Y. Liu, A. Tian, Y. Yu, and X. Du, "Blockchain based privacy-preserving software updates with proof-of-delivery for Internet of Things," *J. Parallel Distrib. Comput.*, vol. 132, pp. 141–149, Oct. 2019.

[4] K. Park, K. Cho, D. Han, T. Kwon, and S. Pack, "Proof of delivery in a trustless network," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2019, pp. 196–200.

[5] L. M. Corbett, "Delivery windows—A new view on improving manufacturing flexibility and on-time delivery performance," *Prod. Inventory Manage. J.*, vol. 33, no. 3, p. 74, 1992.

[6] F. Li, J. Wang, and L. Zhou, "Two-way delay compensation mechanism based on full-load delivery," in *Proc. 16th Int. Conf. Intell. Syst. Knowl. Eng. (ISKE)*, Nov. 2021, pp. 117–123.

[7] A. Gunasekaran, C. Patel, and R. E. McGaughey, "A framework for supply chain performance measurement," *Int. J. Prod. Econ.*, vol. 87, no. 3, pp. 333–347, 2004.

[8] M. H. Meng and Y. Qian, "A blockchain aided metric for predictive delivery performance in supply chain management," in *Proc. IEEE Int. Conf. Service Oper. Logistics, Informat. (SOLI)*, Jul. 2018, pp. 285–290.

[9] K. Almiani, M. A. Alrub, Y. C. Lee, T. H. Rashidi, and A. Pasdar, "A blockchain-based auction framework for location-aware services," *Algorithms*, vol. 16, no. 7, p. 340, Jul. 2023.

[10] P. Gupta and K. N. Jha, "A decentralized contracting system in digital construction," *J. Legal Affairs Dispute Resolution Eng. Construct.*, vol. 15, no. 1, Feb. 2023, Art. no. 02522002.

[11] M. L. Nandi, S. Nandi, H. Moya, and H. Kaynak, "Blockchain technology-enabled supply chain systems and supply chain performance: A resource-based view," *Supply Chain Manag., Int. J.*, vol. 25, no. 6, pp. 841–862, Jul. 2020.

[12] A. R. Harish, X. L. Liu, R. Y. Zhong, and G. Q. Huang, "Log-flock: A blockchain-enabled platform for digital asset valuation and risk assessment in e-commerce logistics financing," *Comput. Ind. Eng.*, vol. 151, Jan. 2021, Art. no. 107001.

[13] Y. Cao and B. Shen, "Adopting blockchain technology to block less sustainable products' entry in global trade," *Transp. Res. Part E, Logistics Transp. Rev.*, vol. 161, May 2022, Art. no. 102695.

[14] S. Cao, H. Johnson, and A. Tulloch, "Exploring blockchain-based traceability for food supply chain sustainability: Towards a better way of sustainability communication with consumers," *Proc. Comput. Sci.*, vol. 217, pp. 1437–1445, 2023.

[15] Y. Khaoua, Y. Mouzouna, J. Arif, F. Jawab, and M. Azari, "The contribution of blockchain technology in the supply chain management: The shipping industry as an example," in *Proc. 14th Int. Colloq. Logistics Supply Chain Manage.*, May 2022, pp. 1–6.

[16] Z. Tian, R. Y. Zhong, A. Vatankhah Barenji, Y. T. Wang, Z. Li, and Y. Rong, "A blockchain-based evaluation approach for customer delivery satisfaction in sustainable urban logistics," *Int. J. Prod. Res.*, vol. 59, no. 7, pp. 2229–2249, Aug. 2020.

[17] M. Demir, O. Turetken, and A. Ferwom, "Blockchain and IoT for delivery assurance on supply chain (BIDAS)," in *Proc. IEEE Int. Conf. Big Data*, Dec. 2019, pp. 5213–5222.

[18] N. Six, C. Negri Ribalta, N. Herbaut, and C. Salinesi, "A blockchain-based pattern for confidential and pseudo-anonymous contract enforcement," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Dec. 2020, pp. 1965–1970.

[19] S. Azzopardi, G. J. Pace, and F. Schapachnik, "On observing contracts: Deontic contracts meet smart contracts," in *Proc. Int. Conf. Legal Knowl. Inf. Syst.*, 2018, pp. 1–12.

[20] A. A. Talha Talukder, M. A. I. Mahmud, A. Sultana, T. H. Pranto, A. B. Haque, and R. M. Rahman, "A customer satisfaction centric food delivery system based on blockchain and smart contract," *J. Inf. Telecommun.*, vol. 6, no. 4, pp. 501–524, Oct. 2022, doi: 10.1080/24751839.2022.2117121.

[21] U. Tokkozhina, B. M. Mataloto, A. L. Martins, and J. C. Ferreira, "Decentralizing online food delivery services: A blockchain and IoT model for smart cities," *Mobile Netw. Appl.*, pp. 1–11, Feb. 2023.

[22] H. R. Hasan and K. Salah, "Blockchain-based proof of delivery of physical assets with single and multiple transporters," *IEEE Access*, vol. 6, pp. 46781–46793, 2018.

[23] H. R. Hasan and K. Salah, "Proof of delivery of digital assets using blockchain and smart contracts," *IEEE Access*, vol. 6, pp. 65439–65448, 2018.

[24] K. Almi'Ani, Y. C. Lee, T. Alrawashdeh, and A. Pasdar, "Graph-based profiling of blockchain oracles," *IEEE Access*, vol. 11, pp. 24995–25007, 2023.

[25] N. Truong, G. M. Lee, K. Sun, F. Guitton, and Y. Guo, "A blockchain-based trust system for decentralised applications: When trustless needs trust," *Future Gener. Comput. Syst.*, vol. 124, pp. 68–79, Nov. 2021.

[26] H. Forslund and P. Jonsson, "Integrating the performance management process of on-time delivery with suppliers," *Int. J. Logistics Res. Appl.*, vol. 13, no. 3, pp. 225–241, Jun. 2010.

[27] J. Zhang, W. H. K. Lam, and B. Y. Chen, "On-time delivery probabilistic models for the vehicle routing problem with stochastic demands and time windows," *Eur. J. Oper. Res.*, vol. 249, no. 1, pp. 144–154, Feb. 2016.

TAWFIQ ALRAWASHDEH received the Ph.D. degree from Universiti Sultan Zainal Abidin, Malaysia, in 2020. He is currently a Lecturer with the Computer Science Department, Al-Hussein Bin Talal University, Jordan, where he also acts as the Computer and Information Technology Center Director. His research interests include cloud computing and task scheduling.

KHALED ALMI'ANI (Member, IEEE) received the Ph.D. degree in information technology from The University of Sydney, in 2010. He is currently a member of the Faculty of Computer Information Science, Higher Colleges of Technology, United Arab Emiratis. His research interests include algorithms for distributed systems, network optimization, and transportation network modeling.

YOUNG CHOON LEE received the Bachelor of Science (Hons.) and Ph.D. degrees from The University of Sydney, Australia, in 2004 and 2008, respectively. He is currently a Senior Lecturer with the School of Computing, Macquarie University, Sydney, Australia. His research interests include distributed systems and high-performance computing.

TAHA H. RASHIDI is currently a Professor of transport engineering with the School of Civil and Environmental Engineering, University of New South Wales (UNSW), and the Director of the Research Centre for Integrated Transport Innovation (rCITI). He is leading research into the interconnectivity between travel behavior and time use and the potential of new mobility technologies to influence this paradigm. He is a Board Member of the International Association for Travel Behaviour Research (IATBR) and an Editor of IATBR NEWS. He serves as the Managing Editor of *Transportation Letters* journal. He sits on the editorial board of several journals, including *Transportation Research—A: Policy and Practice*, *Transportation Research—C: Emerging Technologies*, *Transportation*, and *Travel Behaviour and Society*.

ZEESHAN HAMEED MIR (Senior Member, IEEE) received the B.S. degree from SSUET, Pakistan, in 1999, the M.S. degree from NUST, Pakistan, in 2004, and the Ph.D. degree from Ajou University, South Korea, in 2009. From 2013 to 2016, he was with QMIC, Qatar University, Qatar, as a Research Scientist. From 2009 to 2012, he was a Member of the Technical Staff (MTS) with the Electronics and Telecommunication Research Institute (ETRI), South Korea. From 2001 to 2005, he was a Faculty Member of the CS Department, Institute of Business Administration (IBA), Pakistan. He is currently an Assistant Professor and the Division Chair (DC) with the CIS Division, Higher Colleges of Technology (HCT), United Arab Emirates. He has published his research work in major research publications worldwide and also served on the program/reviewer committees of several reputed conferences and journals. His research interests include mobile/ubiquitous computing, wireless networking/communications, and smart mobility and urban analytics. He is a fellow of HEA.