Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

A blockchain-based SLA monitoring and compliance assessment for IoT ecosystems

Ali Alzubaidi[1,2*], Karan Mitra[3] and Ellis Solaiman[1]

Abstract

A Service Level Agreement (SLA) establishes the trustworthiness of service providers and consumers in several domains; including the Internet of Things (IoT). Given the proliferation of Blockchain technology, we find it compelling to reconsider the assumption of trust and centralised governance typically practised in SLA management including monitoring, compliance assessment, and penalty enforcement. Therefore, we argue that, such critical tasks should be operated by blockchain-based smart contracts in a non-repudiable manner beyond the influence of any SLA party. This paper envisions an IoT scenario wherein a firefighting station outsources end-to-end IoT operations to a specialised service provider. The contractual relationship between them is governed by an SLA which stipulates a set of quality requirements and violation consequences. The main contribution of this paper lies in designing, deploying and empirically experimenting a novel blockchain-based SLA monitoring and compliance assessment framework in the context of IoT. This is done by utilising Hyperledger Fabric (HLF), an enterprise-grade blockchain technology. Our work highlights a set of considerations and best practice at two sides, the IoT application monitoring-side and the blockchain-side. Moreover, it experimentally validates the reliability of the proposed monitoring approach, which collects relevant metrics from each IoT component and examines them against the quality requirements stated in the SLA. Finally, we propose a novel design for smart contracts at the blockchain-side, analyse and benchmark the performance, and demonstrate that the new design proves to successfully handle Multiversion Concurrency Control (MVCC) conflicts typically encountered in blockchain applications, while maintaining sound throughput and latency.

Keywords Blockchain, Trust, SLA, IoT, Monitoring, MVCC, Performance, Hyperledger Fabric

Introduction

Due to the potential complexity of IoT applications, it may be advantageous to outsource service provision to specialised service providers [1]. In turn, service providers promise acceptable levels of service delivery, and guarantee this through the Service Level Agreement (SLA) concept, which makes them responsible for meeting a set of quality standards (i.e availability, latency, throughput, etc) [2]. In particular, SLA regulates service delivery and delineates expectations, rights and obligations of each involved party [3].

According to the ISO/IEC 19086-2:2018 standard [4], the minimal form of an SLA should clearly define a set of properties as follows. First, the SLA must define SLA participants (at least service providers and consumers). Second, it includes Service Level Objectives (SLOs) that stipulate a set of obligations and responsibilities carried out by the service provider. Optimally, an SLO should represent a measurable service quality requirement such as availability, throughput, latency, jitter, packet loss rate [5]. For instance, availability must not be less than 99.9% all the time. Finally, the SLA can state a set of violation consequences enforced on the service provider when it fails to meet the agreement. The violation consequence can be a penalty imposed

*Correspondence:
Ali Alzubaidi
aakzubaidi@uqu.edu.sa
[1] School of Computing, Newcastle University, Newcastle upon Tyne, UK
[2] Umm Al-Qura University, Makkah, Saudi Arabia
[3] Luleå University of Technology, Skellefteå, Luleå, Sweden

on the service provider in the form of financial service credit [6].

Service providers customarily employ the SLA concept to establish their trustworthiness and assure their potential consumers about the quality of their offered services [7]. While SLA guarantees quality requirements (e.g. availability, latency, etc.), it is, as other contractual methods, susceptible to breaches [8]. Therefore, service providers usually guarantee their commitment and show goodwill by accepting a set of violation consequences (i.e. penalties). In current practice, several service providers promise to process violation incidents in good faith, assuring their consumers to impose SLA consequences on themselves [6].

One can question which party to trust as the authority of SLA enforcement and compliance [9]. This question becomes even more delicate when dealing with critical systems that are less tolerable to failures. The current SLA practice commonly assumes cloud providers for holding responsibility for typical SLA lifecycle management, such as SLA monitoring, compliance assessment, incident management, and penalty enforcement [10, 11]. It is also typically the consumers' responsibility to report a service level degradation, supported by evidence deemed irrefutable by the service provider or trusted third parties [6]. This is usually a tedious process, manually handled, time-consuming, error-prone, and requires consumers' good-faith [12, 13]. In some cases, service providers may not react well to poorly formed claims, regardless of their validity [14]. Both sides of a contractual relationship, service providers or consumers, may find it inviting to intentionally fabricate or manipulate evidence of violation incidents in order to maximise profit or avoid hefty penalty [7]. In some scenarios, unresolved disputes have to be escalated to jurisdiction means [10, 15].

Contributions: By considering the possibility of deliberate corruption, misconduct, opacity, conflict of interests, and single point of failure [16], this paper argues that no single party should solely control SLA life cycle management. Blockchain technology invites revisiting traditional applications wherever trust is taken for granted [17]; the SLA practice is no exception. Accordingly, this paper considers the potentiality of blockchain features (i.e. decentralisation and smart contracts) in enabling non-repudiable monitoring, SLA compliance assessment and enforcement of violation consequences in the context of IoT. Subsequently, this paper contributes the following:

1 It proposes and evaluates a novel Blockchain-based IoT monitoring and compliance assessment framework. The framework is applied to an IoT scenario of a firefighting station which hires a specialised IoT services provider with an example SLA in place.
2 It proposes a set of design considerations and best practice at the two key components of the framework; the monitoring side and the blockchain side. The design considerations are experimentally evaluated and proven to provide reasonable performance. Also they eliminate typically encountered read-write conflicts caused by the multiversion concurrency control (MVCC) protocol employed by Hyperledger Fabric; an issue that we reported in our previous study [18].

Paper Organisation: This paper is organised as follows. Section Related Works highlights both blockchain and non-blockchain work in the context of SLA compliance within IoT. Section Preliminaries describes the research context through an overview of a blockchain-based IoT monitoring architecture applied to a firefighting station scenario; wherein IoT-related tasks are outsourced to a service provider, and the contractual relationship is governed by an SLA. Section Monitoring Mechanism and Considerations analyses the journey of fire events, and designs and validates a monitoring mechanism that collects relevant metrics, examines them and reports SLA violation incidents to the blockchain-side of our framework. Section Blockchain-based Compliance Assessment Approach proposes a set of design considerations for Blockchain-based smart contracts that (a) handle received SLA violation incidents from the monitoring-side, (b) assess the compliance of service providers with quality requirements stated in the SLA, (c) and enforces relevant penalties. Finally, Section Experiment and Evaluation of the Blockchain Performance implements and deploys the smart contracts using Hyperledger Fabric, and experimentally examines the performance of the blockchain-based smart contracts against various data loads of metrics reported by the monitoring-side. Figure 1 visually summarises the sequence of the methodology conducted by this paper.

Related works

Recently, there has been a growing interest in establishing mechanisms and schemes that attempt to resolve the trust dilemma of SLAs; examples of which are explored in [1, 7, 9], such as reputation-based mechanisms, usage of auditors, feedback and review systems, trust brokers, and mediators. However, one can question the reliability of trust mechanisms that totally depend on service providers or third parties [19]. Neidhardt et al. [16] shares our view in that traditional solutions only shift trust issues from service providers to third party solutions. For that, they propose placing

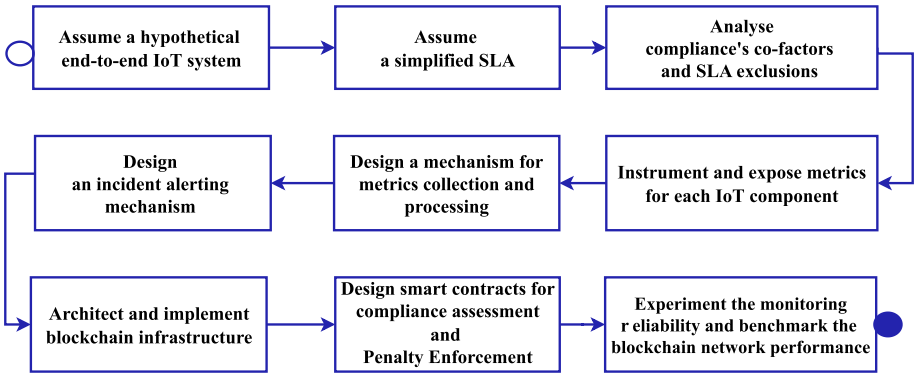


Fig. 1 Research Methodology

trust on Blockchain for SLA conformance validation. Scheid et al. [14] adopts Ethereum in their approach and provides a demonstration on the use of Blockchain for SLA monitoring and enforcement purposes. In [20], the authors note that the premissionless nature of Ethereum, means that monitoring entities can join or leave as they please, making it difficult to guarantee the stability of the monitoring service. Uriarte et al. [8], also proposes a framework that covers the potentiality of Blockchain in key phases of a typical SLA life-cycle including monitoring and penalty enforcement.

While we can find in the literature other work that aims to leverage blockchain for SLA purposes, little effort has been made to specifically address the matter in the context of IoT applications. For instance, the work by [21] proposes an SLA management architecture for IoT purposes. However, the majority of existing works, including the above, adopt Ethereum as an underlying blockchain platform, which limits the potential of their proposed solution for several reasons. For example the limited scalability and performance of public blockchain networks, unsuitability from the perspectives of the need for permissions and privacy, and uncertainty of contract execution cost.

A more recent work by [22] adopts a permissioned Blockchain platform for monitoring purposes, namely Hyperledger Sawtooth. The adopted blockchain is based on a consensus protocol called Proof of Elapsed Time (PoET) and used to establish a common truth among involved monitoring entities. However, it does not specifically focus on monitoring the requirements of IoT application SLAs. Our previous work [18] adopts Hyperledger Fabric (HLF), which is an enterprise-grade permissioned blockchain platform for monitoring QoS and SLA provision within IoT applications. To the best of our knowledge, there are no extensive studies that adopt Hyperledger Fabric for end-to-end IoT application monitoring purposes. Our work complements existing works by proposing a practical blockchain-based monitoring framework in the context of IoT, which accounts for a set of design consideration at both sides, the blockchain-side and monitoring-side. We also experimentally reveal the issue of read-write conflicts when subjecting the blockchain network to stress from the IoT monitoring-side. This is due to the Multi-Version Concurrency Control (MVCC); a protocol employed by HLF for preventing the double-spend problem. The implementation of our proposed framework experimentally proves to handle a high rate of transactions submitted by monitoring tools while maintaining a sound performance and mitigating the issue of MVCC conflicts.

Preliminaries

This section presents the research context by describing a simplified end-to-end IoT-based firefighting system, which observes fire events and reports them to a firefighting station. We presume a Service Level Agreement (SLA) between a firefighting station and an IoT Service Provider (IoTSP), which regulates their contractual relationship and governs quality requirements and violation consequences. This section also overviews the high-level architecture of the blockchain-based solution for automating distrusted processes such as monitoring, compliance assessment, billing, and imposing violation consequences.

Hypothetical IoT scenario

We assume a contractual relationship between a firefighting station and an IoT solution provider, hereafter abbreviated as *IoTSP*. The firefighting station decides to embrace an IoT based solution for quicker response to fire events and severity mitigation. In order to alleviate the burden of dealing with IoT complexity, the firefighting station outsources IoT-related tasks such

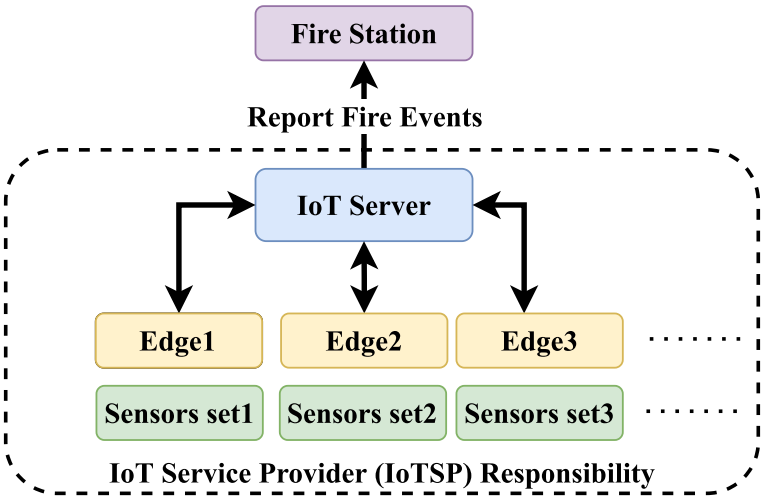


Fig. 2 Overview of an IoT-based Fire Mitigation System. Edges 1,2, and 3 represent the edge computing nodes

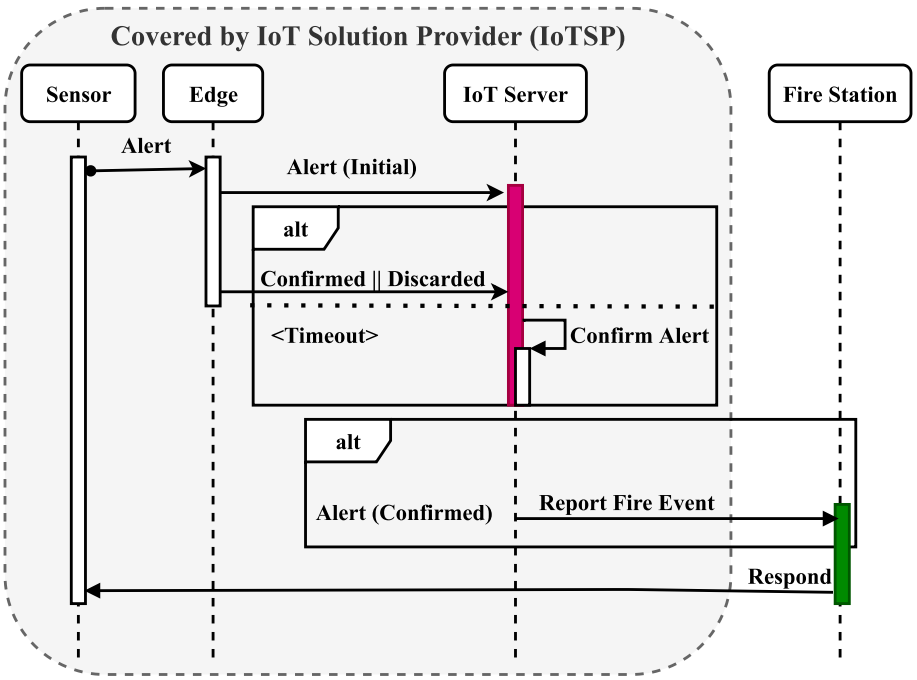


Fig. 3 Stages of a fire event from origination until being reported

as deployment, operations and management to the IoTSP. In this scenario, outsourcing such tasks leaves the firefighting station only responsible for responding to fire events emitted by the IoTSP. Figure 2 depicts the responsibility of the IoTSP, which covers geographically dispersed sensors controlled by edge computing units that locally observe their environment in a real-time manner. The IoTSP also covers a centralised cloud-based IoT server that governs these field assets.

Fire event journey

Figure 3 conceptualises a simple sequence of stages for a fire event. Simply put, there would be a set of specialised fire detection sensors deployed to observe flames within their ranges. The collected data can be roughly expressed in the form of $f \mid f \in \{0, 1\}$ where 1 indicates a detected fire event while 0 denotes the otherwise. These sensors periodically send collected data to their respective edge computing units. The latter analyses received data to

identify whether it indicates a fire event. If so, the edge computing unit must immediately notify the central IoT server of the identified fire events. When the IoT server receives an incident, it must allow a specified duration (e.g. 5 seconds) for a follow-up message from the edge unit about the same location. Meanwhile, one of the following cases may occur:

- the IoT server receives a *Discard* message from the edge computing unit, and thus no further action is taken.
- the IoT server receives a *Confirm* message, which immediately triggers a report of a confirmed fire event to the firefighting station.
- the IoT server receives neither a *Confirm* nor a *Discard* message within the specified wait time (timeout). Therefore, the IoT server must take precautionary action by self-confirming the initial fire event and reporting it to the firefighting station.

SLA between IoTSP and firefighter station

In light of the above-described IoT-based firefighting scenario, assume an SLA that governs the relationship between the IoTSP and the firefighting station, which obligates the former to comply with a set of quality requirements. For instance, the IoTSP must observe for fire events $f \in F$ (where $f = 1$) and report them to the firefighting station within a specified duration ($t_s \leq d$). The firefighting station expects quality availability all the time, especially during a confirmed fire event. Due to the scenario criticality in this study, the availability is not only limited to the IoT server but also extended to edge units. This quality requirement can be denoted as $A_{edge} \wedge A_{server} \neq down$. The SLA also specifies a set of breach categories *BC*, where it holds the IoTSP accountable for their consequences in case of violation. For example, consider three breach categories $bc_1, bc_2, bc_3 \subseteq BC$ as follows:

- bc_1: consider a situation where the IoTSP fails to report a confirmed fire event f due to a downtime of any covered component ($\neg A_{edge} \vee \neg A_{server}$), and therefore a failure to calculate the duration needed for processing and reporting the fire event, which will violates the condition of $0 < t_s \leq d$. In such a failure case, $t_s = 0$. Accordingly, a monitoring metric tuple M is classified as a breach of type bc_1 when it holds $(f,\ t_s = 0,\ (\neg A_{edge} \vee \neg A_{server}))$,
- bc_2: consider a situation where the IoTSP fails to maintain availability of the server or any edge computing unit. However, this case does not occur during a confirmed fire event f and thus it is less critical since $t_s = 0$ is perfectly normal and expected. Accordingly, M is classified as bc_2 when it holds $(\neg f,\ t_s = 0,\ (\neg A_{edge} \vee \neg A_{server}))$
- bc_3: consider a situation where the IoTSP maintains available components, and manages to process and report confirmed fire events f, but fails to do so within the specified duration where it should be $t_s \nleq d$. Accordingly, M is classified as breach of type bc_3 when it holds $(f,\ t_s \nleq d,\ (A_{edge} \wedge A_{server}))$

Other breach categories can be defined in a similar manner. Depending on the severity of each breach category $bc_j \subseteq BC \mid j \in \mathbb{N}$, the SLA defines the maximum tolerance *mt* to the violation frequency. Moreover, the SLA stipulates what penalty should be applied on the IoTSP if *mt* is reached. This done by tracking the violation rate *vr* for each bc_i, which is calculated as per Eq. 1,

$$vr = \frac{\sum_i^n b}{\sum_i^n b + \sum_i^n c} \times 100 \tag{1}$$

where b is the count of breach cases and c is the count of compliant cases. As long as the violation rate (vr) does not exceed the assigned max tolerance $vr \ngtr mt$, the SLA validity remains intact $\lambda \leftarrow true$; however, penalties are enforced whenever applicable. Otherwise, the SLA is terminated $\lambda \leftarrow false$, and a full refund is issued to the consumer. Once the SLA is established, it declares the commitment of the IoTSP towards these promised quality requirements and violation consequences.

Architecture overview

Assuming an untrusted relationship between the firefighting station and IoTSP, we consider automating and operating distrusted processes within a blockchain environment such as compliance assessment and penalty enforcement. Figure 4 envisions the overall architecture where the IoTSP's compliance level is under a continuous monitoring and examination against a set of promised Quality of Service (QoS) requirements. To materialise the blockchain-based monitoring and compliance architecture, we consider two primary components, which are the monitoring-side and blockchain-side; discussed as follows:

Monitoring-side

A monitoring mechanism is necessary for providing the awareness and visibility needed for executing SLA distrusted processes [23, 24]. As illustrated in Fig. 5, the monitoring side is responsible for metrics collection related to quality requirements stated in the SLA. For example, it ceaselessly observes fire events f and tracks their journey from the initiation stage at the edge level,

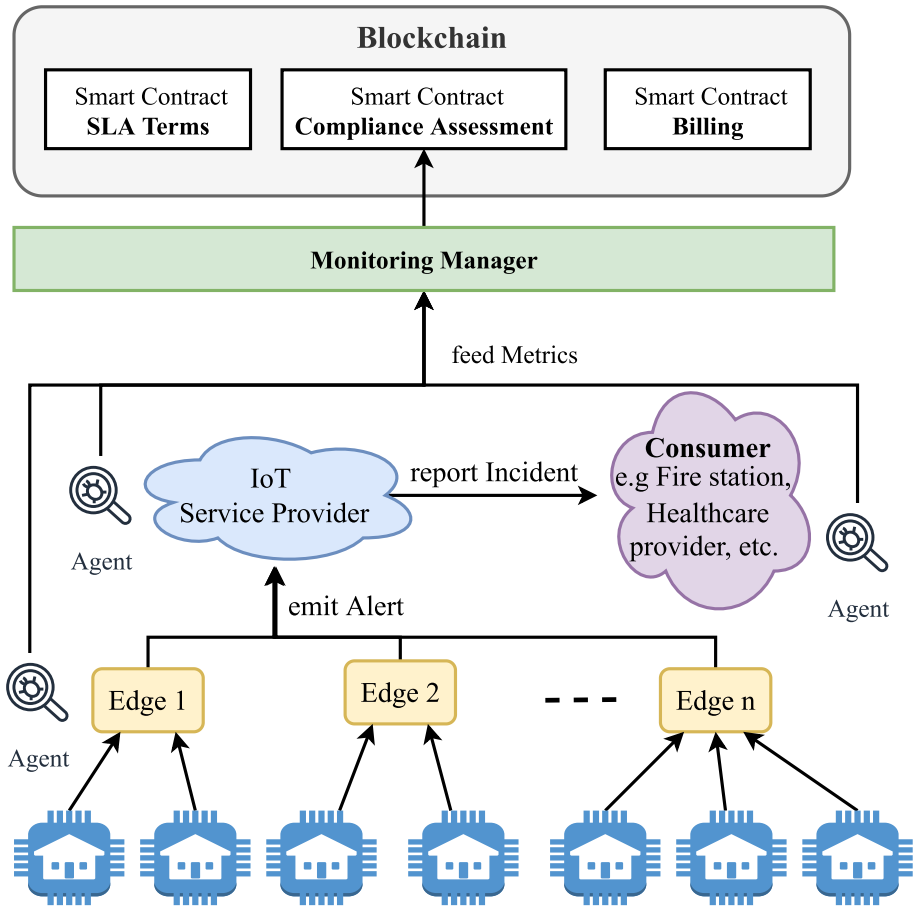


Fig. 4 Motivating IoT scenario where blockchain is employed for SLA monitoring and enforcement

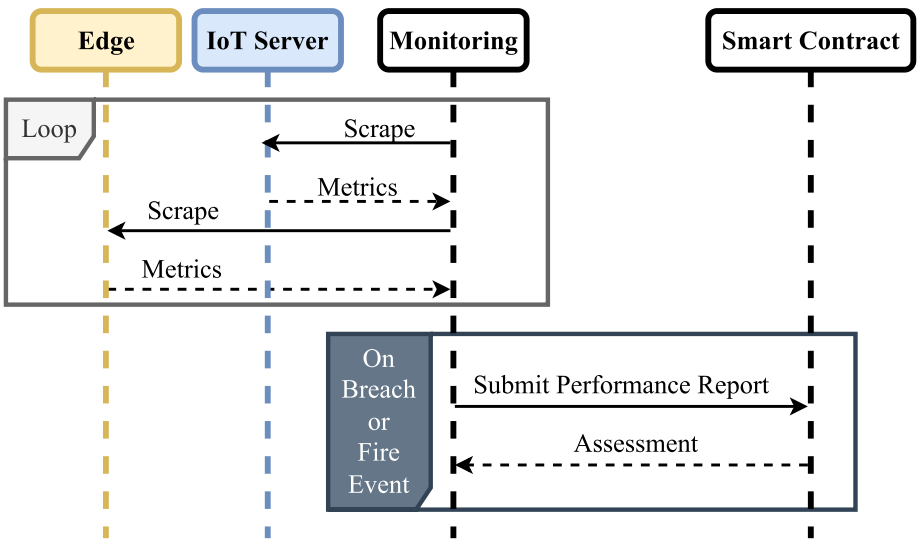


Fig. 5 Metrics collection and reporting to the blockchain-side

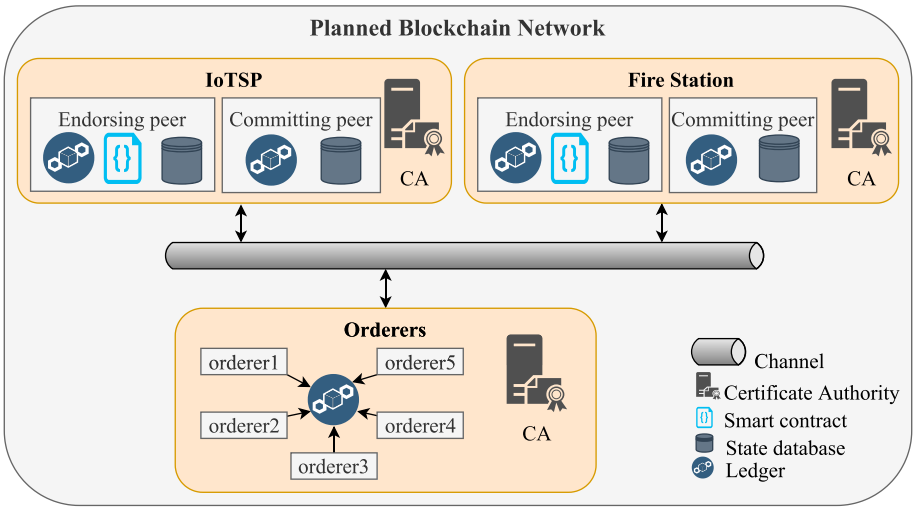


Fig. 6 Hyperledger Fabric's Blockchain network of two organisations: IoTSP and firefighting station

through the processing stage at the server level, and until the stage of reporting confirmed fire events to the firefighting station. It also continuously observes the availability of both the edge and the server computing units. Whenever the monitoring manager encounters an incident that requires attention, it alerts the smart contract by submitting a transaction consisting of a set of collected metrics $M = (f, t_s, A_{edge}, A_{server})$, such that

- f indicates whether there was a confirmed fire event.
- t_s the duration it takes the IoTSP to process and report a confirmed fire event if any.
- A_{edge} and A_{server} are availability indicators of both edge computing units and the server.

In order to avoid overwhelming the smart contract with unnecessary interactions, the monitoring manager controls the alerting mechanism such that transacting with the smart contract occurs only in the event of a confirmed fire event f or a breach b. Identifying either of them will cause the monitoring manager to submit a transaction to the blockchain.

Blockchain-side

In this study, we employ Hyperledger Fabric (HLF) platform, which enables benefiting from several blockchain principles such as decentralisation, transactions immutability, consensus mechanism, and other blockchain features. Influenced by HLF philosophy, we consider a distributed system where involved parties construct a blockchain network and contribute to the infrastructure and computing resources.

As depicted in Fig. 6, we consider at least two organisations, which are the firefighting station and the IoTSP. Every organisation hosts a set of peers for high availability. In this distrusted environment, each participating organisation holds replicas of three essential elements, that are:

- a replica of the ledger; needed for committing and appending blocks of transactions;
- a replica of the state storage: needed for reflecting the latest state of persisted records; and
- a replica of a set of smart contracts (Chaincode), which executes distrusted processes and acts as a gateway to the local state storage.

As Fig. 4 highlights, a smart contract may compose the SLA terms, the logic of compliance assessment, and the functionality of both the billing and enforcing relevant violation consequences.

Monitoring mechanism and considerations

Most distrusted SLA-related processes are of a decision-making nature, such as compliance assessment and penalty enforcement [9]. Transforming such processes into an autonomous decentralised application requires feeding them with relevant metrics from monitoring means [25]. By considering the presented SLA in section SLA between IoTSP and Firefighter Station, we examine which relevant co-factors that could impact the compliance rate of the IoTSP towards its obligations. This section describes the overall monitoring architecture, metrics collection, as well as reporting mechanism to the blockchain side. The ultimate goal of this section is to engineer a mechanism for metrics collection and reporting to the blockchain while avoiding unnecessary

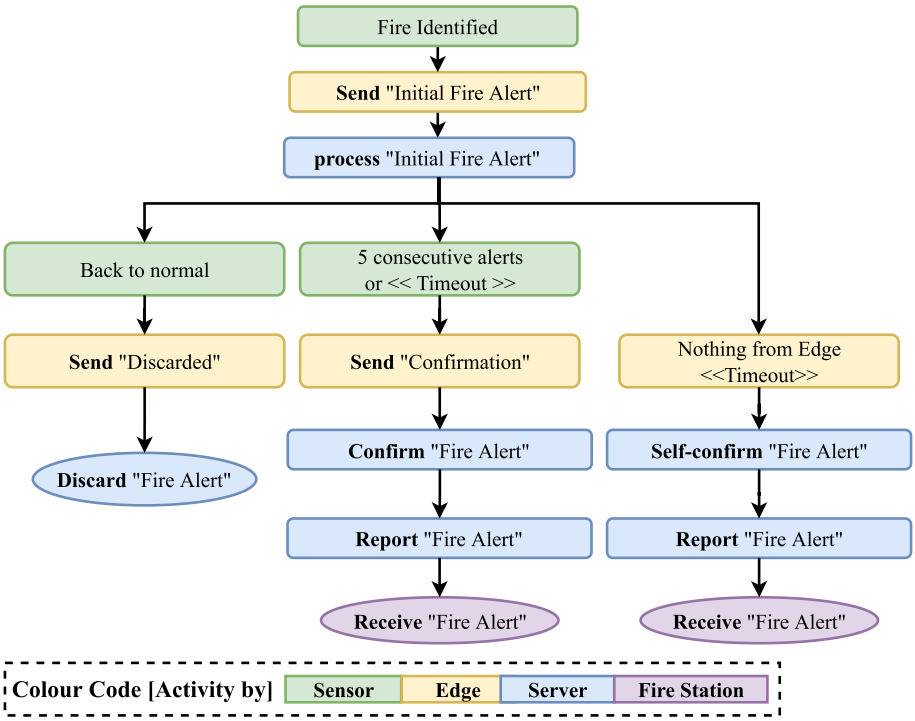


Fig. 7 key stages for a fire event across different IoT layers

transactions with the blockchain side and accounting for failed transactions.

Determining contributing factors to compliance status

In order to determine the adherence level towards a quality requirement, we need to determine co-factors that influence the compliance status. For demonstration purposes, we consider the following quality requirements:

- $QoS(Availability_{e,c})$, where $e \leftarrow edge$ and $c \leftarrow IoTserver$. We assume a centralised server, and several geographically dispersed edge computing units.
- $QoS(t_s) \leq d$, which mandates the IoTSP to process and report a fire alert to the firefighting station within a specified duration.

Determining the IoTSP compliance level with the availability requirement is a relatively straightforward process regarding; whether it is the server or an edge computing unit. That is, a binary decision tree of $\{true, false\}$ can help determine the IoT compliance level towards the availability of any covered component. However, this is not the case in terms of the second quality requirement, which relates to the transmission time of a fire alert from its origination until being reported to the firefighting station.

Consider a dispute that arises of whether the IoTSP fulfilled its duty in reporting a fire event within $t_s \leq d$. Following are some cases which can lead to a dispute regarding this quality requirement which are:

- The firefighting station's system fails to log the fire alert once received.
- The IoTSP fails to satisfy $t_s \leq d$, but it claims otherwise.
- The IoTSP satisfies $t_s \leq d$; however, the firefighting station claims otherwise.

Therefore, we analyse the journey of a fire event from its initiation until being delivered to the firefighting station. This is to unambiguously determine what co-factors precisely determines the IoTSP's compliance level towards $t_s \leq d$. Based on Fig. 3, we identify three possible scenarios where fire events that may develop from the state of being identified until being either discarded or reported to the firefighting station; as per depicted in Fig. 7. These three scenarios are as follows:

1 *False positive fire alert*: It occurs when an edge computing unit issues an initial fire alert to the server and then follows up with a discard message during the waiting period. Accordingly, the server must discard and refrain from reporting it to the firefighting station.

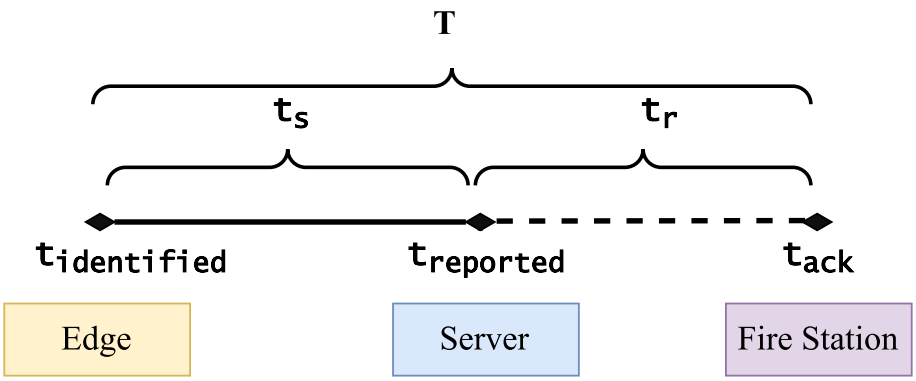


Fig. 8 Timeline for fire event development

2 *True positive fire alert*: It occurs when an edge computing unit issues an initial fire alert to the server and then follows up with a confirmation within the time limit (i.e. within five seconds). Accordingly, the server must immediately report the fire event to the firefighting station.
3 *Dangling fire alert*: It occurs when an edge computing unit sends an initial fire alert to the server; however, it fails to follow up with either confirm or discard messages within the time limit. Accordingly, the server assumes criticality at the edge side (e.g. fire damage); and thus report the fire event to the firefighting station.

Since this section focuses on co-factors contributing to the IoTSP compliance level, we can optionally omit the first scenario where fire events are classified as false positive and thus discarded. That is, the SLA obligates IoTSP to report fire events, which leaves the other two scenarios where fire events end up confirmed and reported to the firefighting station either because the edge computing unit issues a confirmation or because the IoT server self-confirms it for a precautionary reason.

For both of these scenarios, a fire alert undergoes a total transmission time as in Eq. 2, where T measures the actual transmission time of a fire alert from its origination (an edge computing unit) until being delivered to the firefighting station.

$$T \leftarrow t_s + t_r \tag{2}$$

$$t_s \leftarrow t_{reported} - t_{identified} \tag{3}$$

$$t_r \leftarrow t_{ack} - t_{reported} \tag{4}$$

Figure 8 illustrates the total transmission time T forms the total of two main elements, which are as follows:

- t_s refers to the duration that takes the fire alert from being issued at an edge computing unit $t_{identified}$ until being reported by the server $t_{reported}$ (calculated as per Eq. 3).
- t_r refers to the rest of the fire alert journey, which is the duration that takes it from being reported by the server $t_{reported}$ until being finally delivered to the firefighter station (calculated as per Eq. 4).

Figure 3 assigns the IoTSP with the responsibility of both the server and the edge computing unit. Subsequently, we can draw attention to t_s which determines the compliance level of the IoT towards the quality requirement $t_s \leq d$. On the other hand, the SLA understandably does not cover t_r because it can be subject to several factors beyond the immediate control of the IoTSP (e.g. Internet routing delay) or issues at the firefighting station system. For that, monitoring must not only aligns with quality requirements but also with SLA executions [6]. However, the blockchain-based solution can be designed to keep records of both t_r for auditing and dispute resolution purposes.

Monitoring mechanism design and implementation

Figure 9 illustrate a monitoring and alerting architecture based on a well-established open-source project, namely Prometheus[1]. Reasons for this selection are summarised in [26], which include, but are not limited to,

- it is hosted by the Cloud Native Computing Foundation (CNCF) [2] and enjoys wide adoption and community support in terms of documentation, maintenance, integration tools and libraries.

[1] https://prometheus.io/
[2] https://www.cncf.io/cncf-prometheus-project-journey

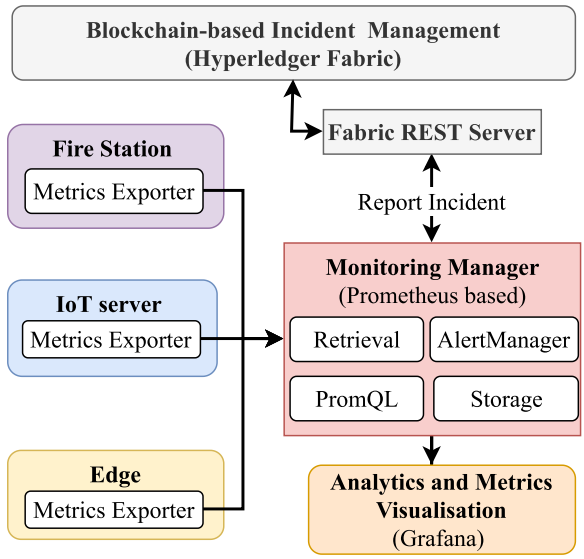


Fig. 9 Employing Prometheus monitoring tool for feeding metrics to the Blockchain-side

- it adopts a pull approach for metric collection, in which target entities (edge, IoT server, firefighting station system) can export relevant metrics via REST APIs to be scraped by the monitoring manager.
- The Prometheus's overall architecture considers high availability, replication, and fault-tolerance.
- supports flexible query language, namely PromQL, for defining rules and querying thresholds and alerts. it also provides a rich set of libraries and instrumentation tools for exporting relevant metrics from the targeted instances (application, containers, infrastructure, services, etc.)
- employs an alerting system that can be automatically triggered based on predefined conditions.

Using Prometheus, this study instruments a set of relevant metrics for each component (edge, cloud, application). It exposes these metrics via REST APIs. The monitoring manager collects and aggregates exposed metrics and stores them in a time-series database based on a set of rules. The *Alert Manager* regulates the alerting mechanism and uses a query language (PromQL) to define what thresholds to trigger associated smart contracts. There is also a component called Fabric REST Server that facilitates communication, authentication, and interaction between Prometheus (monitoring/alerting system) and smart contracts on the blockchain side. Prometheus manager also enables outsourcing metrics from different components to a visualisation tool such as Grafana for analytics and insights that we need for experimental purposes. The following sections delve further into the design and implementation of these components.

Metrics instrumentation and exporting

As shown in Fig. 9, both the monitoring manager and alerting mechanism depend on metrics exposed from each component of the IoT ecosystem. On the one hand, Prometheus's exporters enable instrumenting and exposing relevant metrics from each component via an exposed REST API. On the other hand, the monitoring manager regularly collects metrics from components covered by the IoTSP (edge and sever) and the firefighting station system.

It is noteworthy that various IoT components are deployed to different locations of distinctive timezones. For example, the firefighter station possibly deploys its system to a data centre that differs from the IoTSP server or edge computing units. Hence, there arises the possibility of different timezones. As Fig. 9 depicts, there is a metric exporter which resides at the location of each component and thus is subject to the employed timezone settings of the respective component. Consider the fact that the calculation of t_s or t_r depend on timestamps from different timezones. To prevent unintended miscalculation, we employ a Unix timestamps system, which is a standardised time representation and timezone independent. Therefore, each exporter instruments and composes metrics using this Unix timestamp system.

Edge-side exporter

As per discussed in Section Determining Contributing Factors to Compliance Status and presented in Fig. 8 edge computing units are responsible for identifying fire events. Therefore, the Prometheus exporter composes and exports the metric $t_{identified}$ at the edge side. Note that, Eq. 3 deems $t_{identified}$ as the first essential element for evaluating the IoTSP's adherence towards the quality metric t_s. Moreover, edge computing units are responsible for confirming fire alerts. Therefore, we use $t_{confirmed}$ to assert whether and when the edge computing unit was able to confirm a fire event.

Figure 10 illustrates the logic of instrumenting and exposing both $t_{identified}$, which indicates when a fire event was first identified, and $t_{confirmed}$ which indicates the time of confirming the fire event. It uses the following conventions:

- f' an initial fire alert.
- f a confirmation of a fire event

Provided that there is a capable device at the edge-side such as Raspberry PI4, a Prometheus exporter can be deployed to compose and expose relevant metrics via a REST API for collection by the monitoring manager. For example, once the edge computing unit

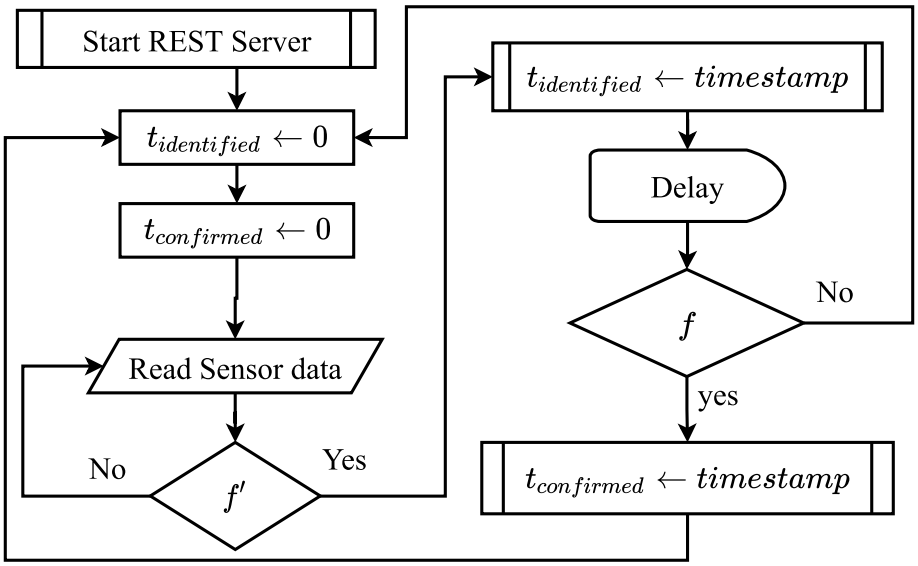


Fig. 10 Instrumenting and exposing relevant metrics at edge level

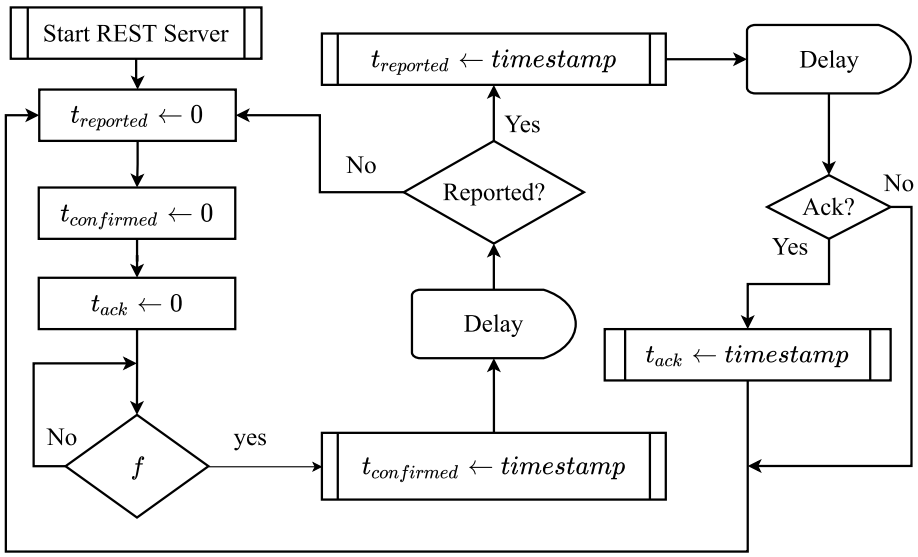


Fig. 11 Instrumenting and exposing relevant metrics at server level

identifies a fire event and sends an initial alert f', the Prometheus exporter assigns a timestamp to $t_{identified}$ and then exposes it for collection. Afterwards, the Prometheus exporter allows a delay to observe whether the edge unit confirms the fire event. When the fire event is confirmed f, it assigns $t_{confirmed}$ a timestamp to be exposed for the monitoring manager. Otherwise, it rests $t_{identified}$ to zero, which can indicate when the edge unit declares the fire event as a false positive.

Server-side exporter

Recall that the IoT Server reports fire events to the fire-fighting station only when they are confirmed either by the edge or self-confirmed by the server for precautionary reasons (refer to section Determining Contributing Factors to Compliance Status). That is why we do not only expose $t_{confirmed}$ from the edge side, but also the IoT server-side as well. Figure 11 illustrates the logic of exposing relevant metrics from the server-side, which

captures when the fire event f is confirmed $t_{confirmed}$ and reported $t_{reported}$. Note that, Eq. 3 deems the latter as second essential element for evaluating the IoTSP's adherence towards the quality metric t_s. Moreover, note that $t_{confirmed}$ metric can be assigned a timestamp by either the exporter at the edge side or the one at the IoT server. This measure is in place to account for natural disaster at the edge side which can cause a downtime to the edge computing unit, which lead to a downtime for its Prometheus exporter as well. In this case, the IoT server self-confirms the fire event. Therefore, its Prometheus exporter overrides assigns a timestamp for $t_{confirmed}$ metric to indicate when the fire event is deemed true positive.

The Prometheus exporter at the server-side rests all metrics to zero in two cases:

- *false positive*: the edge unit sends a "Discard" message.
- *true positive*: the fire alert is confirmed by either the edge unit or the IoT server itself. However, it fails to report it within the specified period.

Moreover, we track whether and when the firefighting station receives the reported fire event t_{ack}. While the latter does not contribute to evaluating IoTSP's compliance, it is exposed and collected for assertion and auditing purposes.

Metrics collection

Since Prometheus adopts a metrics-pull mechanism, the monitoring manager regularly collects, analyses exposed metrics, and decides where there is an incident to report the blockchain side (See Fig. 5). As per the SLA, Fig. 12 visualises relevant collected metrics such as availability/downtime of covered IoT components, which are Edge-side and Server-side, as per Fig. 12a. It also shows different fire states (confirmed fire f or no fire $\neg f$), as well as when a confirmed fire was identified $t_{identified}$ and reported to the firefighting station $t_{reported}$. For the sake of an example, we assume a quality requirement $QoS(t_s \not> 3)$ in order to cause deliberate breaches for experimental purposes.

The monitoring manager does not only collect metrics but also regulates when to report the IoTSP's performance to the blockchain side. As shown in Fig. 5, the IoTSP's performance is reported either on the occasion of a confirmed fire event f or a breach to a quality metric B, which can be due to unavailability of an edge computing unit $A_{edge} \leftarrow down$, unavailability of the server $A_{server} \leftarrow down$ or a breach to $t_s \not> d$. This measure is in place to avoid overwhelming the blockchain with unnecessary transactions.

Algorithm 1 illustrates the procedure of metrics analysis, providing the following:

- the unavailability of any component, edge or server, implies a breach case that triggers the compliance evaluation. The Algorithm exempts the edge unavailability as in line 8 which does consider it in a breach of the availability requirement unless there is no fire event $f = false$. In other words, it exempts edge computing units from the availability requirement in case of natural disaster caused by a confirmed fire event $f = ture$.
- $t_{confirmed} \in \mathbb{N} \mid t_{confirmed} > 0$ indicates a confirmed fire event f, which triggers the compliance evaluation. This metric is provided by both sides edge and server.
- In case of a confirmed fire event f, the monitoring manager examines whether the IoTSP reports the fire alert to the firefighting station. If so, it then examines the IoTSP's compliance towards $t_s \not> d$ in accordance with Eq. 3, which is the duration consumed by the IoTSP for processing and reporting the confirmed fire event, as shown in Fig. 8.

Require: $B \vee f$ ▷ B for Breach whereas f for fire event
Output: $Performance\ Report$ ▷ sent to Blockchain
1: **repeat**
2: **Monitoring Manager:** scrape Metrics
3: $A_{edge} \xleftarrow{R} \{up, down\}$ ▷ Edge Availability
4: $A_{server} \xleftarrow{R} \{up, down\}$ ▷ Server Availability
5: **if** $A_{server} \leftarrow down$ **then**
6: $B \leftarrow True$
7: **end if**
8: **if** $(A_{edge} \leftarrow down \wedge f = false)$ **then**
9: $B \leftarrow True$
10: **end if**
11: **if** $t_{confirmed} \neq 0$ **then** ▷ fire event? positive value other than 0 indicates a fire event
12: $f \leftarrow True$
13: **if** $t_{reported} \neq 0$ **then** ▷ reported? positive value other than 0 indicates successful
14: $t_s \leftarrow t_{reported} - t_{identified}$
15: **if** $t_s \not< d$ **then** ▷ Breach of specified duration
16: $B \leftarrow True$
17: **end if**
18: **else**
19: $B \leftarrow True$
20: **end if**
21: **end if**
22: **if** f $\leftarrow True \vee B \leftarrow True$ **then**
23: $M = (f, t_s, A_{edge}, A_{server})$
24: **Alert Manager:** report M to Blockchain-side
25: **end if**
26: **until** End of SLA

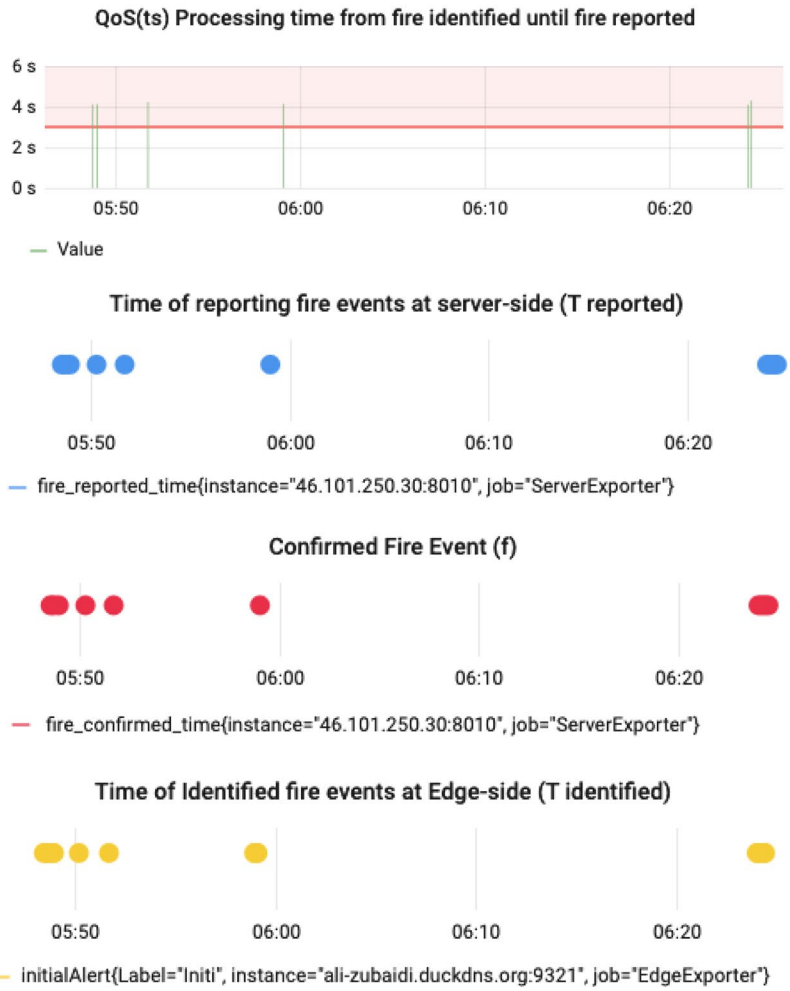
Algorithm 1 Reporting Mechanism to the Blockchain Side

Validating the monitoring approach

This section particularly focuses on validating the monitoring part of this architecture which includes the following:

- metrics instrumentation and exposure from each covered IoT component (edge and cloud).
- metrics collection from by the monitoring manager.

(a) Availability indicators of covered IoT components



(b) collecting metrics of $t_{identified}$, $t_{reported}$ and f, as well as calculation of t_s

Fig. 12 A screenshot of metric collection and Incident Identification

Table 1 Classification and summary of metrics covered by Algorithm 1

f	Transmission Time Metrics			Availability Metrics		
true	$t_{identified}$	$t_{reported}$	t_s	**Edge**	**Server**	**Incident?**
true	> 0	> 0	$\leq d$	up	up	No
true	> 0	> 0	$\leq d$	down	up	No
true	> 0	0	> 0	up	down	Yes
true	> 0	> 0	$> d$	up	up	Yes
false	0	0	0	up	up	No
false	0	0	0	down	up	Yes
false	0	0	0	up	down	Yes
false	0	0	0	down	down	Yes

- alerting system, which is used for triggering the compliance assessment smart contract.

Timestamping is the essence of each instrumented metric discussed above. Consider Fig. 4, which depicts the overall architecture of a blockchain-based IoT monitoring and compliance assessment. For simplicity, we use Digital Ocean [3] to deploy both the IoT server application and the firefighting systems to different virtual machines located at different regions of distinctive timezones. The edge computing unit is deployed using a Raspberry PI4 at a distinctive region and timezone as well. Note that, Internet is the only possible way for these IoT components to connect and communicate over HTTP protocol due to the geographically despaired deployment of each IoT component.

A Prometheus exporter (A.K.A monitoring agent) is attached to each IoT component in order to expose relevant metrics. As a result, metrics exporters are influenced by the disparity of their associated IoT components regarding the timezone difference and the need for an internet connection. The monitoring manager is also deployed to another cloud instance of different regions and timezone and thus needs to reach each Prometheus exporter in order to collect exposed metrics.

Table 1 summaries metrics used for validating the monitoring approach (see Algorithm 1). Note that each metric combination has a different degree of criticality violation severity. Therefore, each one of them should be assigned a different maximum tolerance rate *mr* as discussed in section SLA between IoTSP and Firefighter Station. Regardless, Table 1 maps each combination of these metrics to a decision of whether to consider it as an incident that leads to triggering the compliance assessment smart contract.

For experimental purposes, we deliberately cause incidents to observe whether the monitoring can correctly classify them; and thus trigger the alerting mechanism. The designed incidents are as follows:

- To cause an edge unit downtime, we intentionally disconnect from the internet to halt its operation.
- To cause downtime to an IoT server or firefighting system, we simply halt the execution of the deployed application or shutdown or suspend the cloud virtual machine.
- to cause a fire alert, we expose the flame sensing module to extreme light or fire.
- To cause a breach to t_s, we calculate the average time of processing and transmitting a confirmed fire event f from its origination until being reported to the firefighter station, which resulted in 3 seconds. Therefore, we assign the quality requirement $t_s \leq 3second$, which causes any reading below to be considered a breach.

Figure 12b depicts a visualised sample of metrics collected by the monitoring manager. To visualise the collected metrics, we design a dashboard using Grafana[4] and PromQL language. The dashboard shows the ability of the monitoring manager to constantly collect metrics from various exporters and identify incidents as well. For instance, it shows that the server maintained a constant uptime until it experienced a brief downtime, roughly between 5:40 am and 5:50 am. Regarding the edge unit, it shows a constant uptime expect three occasions as follows:

- before the server expedience a downtime ($A_{edge} \leftarrow down$ while $A_{server} \leftarrow up$).
- during the server downtime ($A_{edge} \leftarrow down$ while $A_{server} \leftarrow down$)
- after the server resumed an uptime status ($A_{edge} \leftarrow down$ while $A_{server} \leftarrow up$)

[3] https://www.digitalocean.com

[4] https://grafana.com/

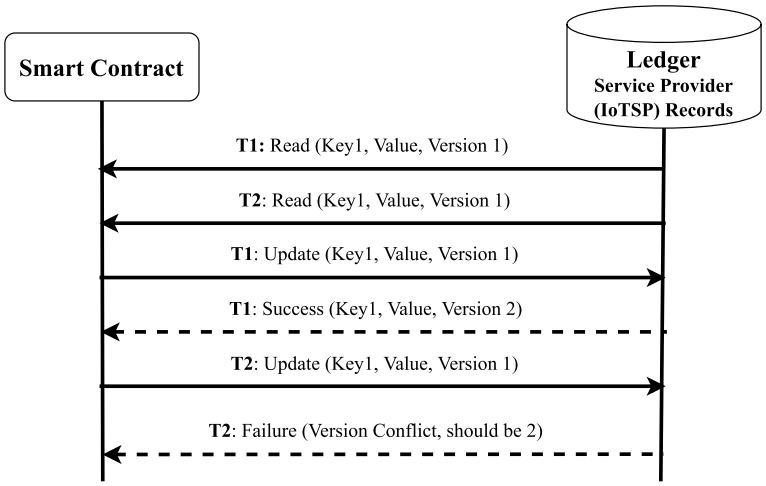


Fig. 13 Read-Write set conflicts caused by multiple transactions updating the same record

The dashboard also depicts the journey of various fire alerts, as follows:

1 *first stage*: timestamps of when fire events are being identified at the edge side (Yellow colour).
2 *second stage*: timestamps of when the fire alert is being confirmed (Red colour).
3 *second stage*: timestamps of when the fire alert is being confirmed (Blue colour).

The dashboard also maps the IoTSP's performance in terms of t_s, the duration it takes for processing and reporting each fire alert. As the dashboard shows, we calibrated t_s to 3, which the IoTSP fails to challenge at the fire events. Therefore, the red area of the *ts* in the dashboard indicates a breach by the IoTSP regarding t_s.

To sum up, the monitoring approach has proven to work properly and reliably for the purposes of this study. Furthermore, it also demonstrates the correct operation of the implemented IoT because the monitoring precisely reacted as per actions conducted on the IoT system. Accordingly, we can consider the monitoring approach reliable for triggering the smart contract compliance assessment, as discussed in the following sections.

Blockchain-based compliance assessment approach

As discussed above, we consider a monitoring manager that interacts with the blockchain-side to report SLA violation incidents by submitting a transaction that holds a set of metrics M, where $M = (f, t_s, A_{edge}, A_{server})$, as described in section Monitoring-side. Unlike conventional applications, no blockchain operation is considered valid unless undergoing through a set of validation mechanisms such as ESCC (Endorsement System Chaincode), VSCC (Validation System Chaincode) and MVCC (Multi-Version Concurrency Control [27]. We particularly draw attention to the problem of MVCC conflicts which can be resulted from high rate of interactions between the monitoring-side and the blockchain-side; a scenario that might happen when there is a rapid number of violation incidents taking place simultaneously [18].

This section proposes a smart contract design for metrics evaluation and SLA compliance assessment which does not only promise to resolve MVCC conflicts but also maintains reasonably higher throughput and less latency. First, we highlight how the MVCC protocol can impact a high rate of transactions from the monitoring side. Then, we propose an SLA data model and an improved smart contract design that encounter the challenge of MVCC conflicts. Altogether are run over blockchain to be enforced on the hypothetical IoT-based firefighting scenario approach with the aid of the above-discussed monitoring mechanism.

MVCC impact on high-throughput transactions

Different blockchain platforms apply distinctive schemes to mitigate the double-spending problem. For example, HLF employs the MVCC mechanism to control records consistency by tracking version changes of a record in the form of $(key : value, version)$. As depicted in Fig. 13, whenever there is a transaction T that causes an update operation to a record, there is a read set $(k : val, ver)$. Based on this read set, a write set $(k : val', ver')$ attempts to update the state storage. However, before applying and committing the write set, the MVCC mechanism checks whether version *ver* of the read set is applicable.

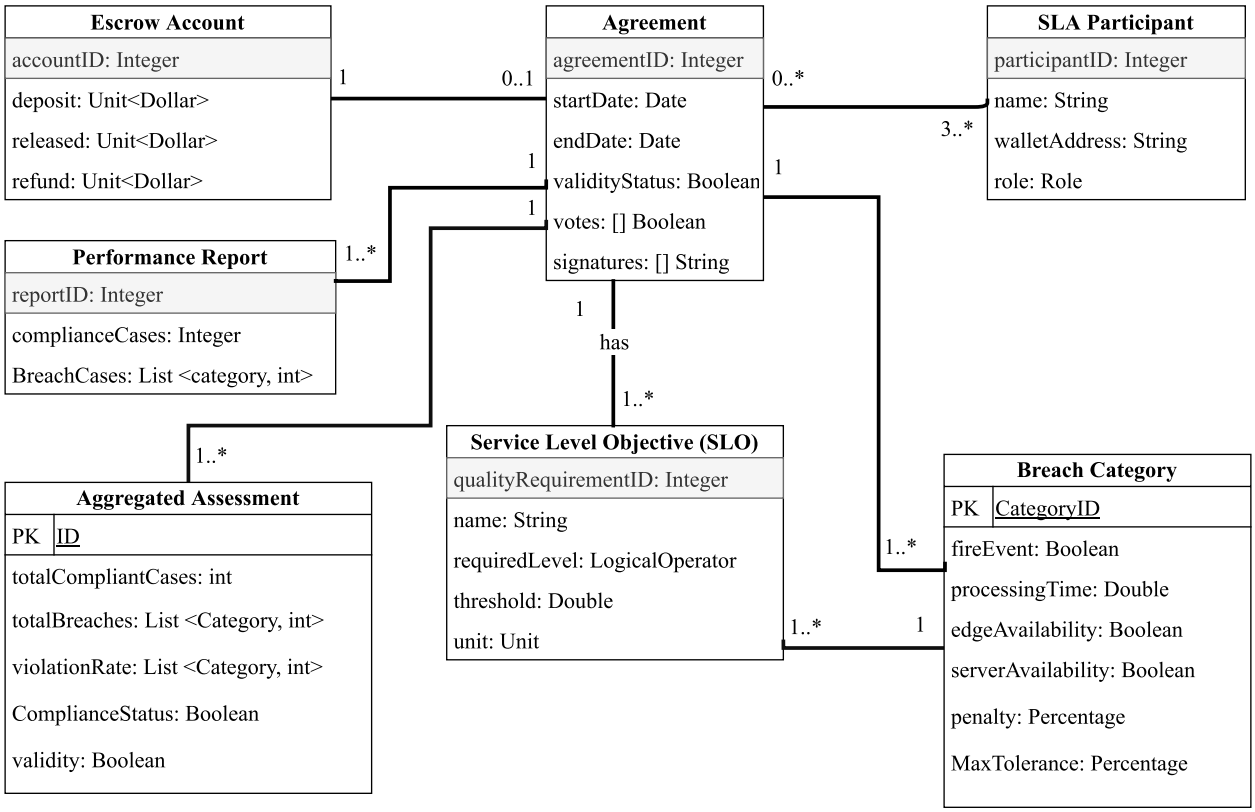


Fig. 14 Enhanced Data Model for Evaluation Compliance over Blockchain

Otherwise, the version could have been changed due to another transaction T_1 that managed to be committed. In this case, the version of the read set will be classified as obsolete. Therefore, T_2 fails when it tries to commit the write set [28]. The MVCC mechanism can pose a challenge to high-throughput applications where multiple read-write sets are the norm, and double-spending is of no issue. For example, in our case, a high rate of transactions expected from the monitoring side would typically cause multiple read-write operations on blockchain records. However, such transactions may highly likely face Read-Write sets conflicts due to the MVCC mechanism [29].

In our previous study [18], we find these conflicts are attributed to multiple update transactions that happen to update the same asset while landing on the same block. By investigating the issue of MVCC conflicts, it appears that one transaction would succeed the MVCC validation, while the rest eventually fail due to a version change caused by that successful transaction. We also find in our previous study that, adjusting HLF configurations does not completely mitigate MVCC issues. Therefore, the present work address the issue of MVCC conflicts at the smart contract level be proposing an enhanced SLA data model and improved design of the smart contract.

Enhanced compliance data model

By studying the impact of the MVCC protocol on the compliance assessment, we found that the design of both the smart contract and the data model plays a vital role in mitigating Read-Write set conflicts. Therefore, this section proposes an enhanced compliance assessment approach based on a simple but effective, which essentially prohibits update operations on performance records $pr_i \in PR$ such that $(k : val, ver) \mid \Delta ver = 0$. This is to eliminate changes on records versions, which effectively mitigates the possibility of MVCC conflicts. In practice, instead of updating an existing performance record at the occurrence of each incident, there is a new pr_i for each incident which will be aggregated with other them at the end of every billing cycle. Figure 14 presents an enhanced SLA data model, which accommodates the following:

Breach categories

The SLA data model considers a complex SLA agreement that covers an end-to-end IoT system. The complexity is drawn from the fact that the SLA, presented in this paper,

covers various breach categories $bc_i \in BC$, such that each of them is based on a combination of metrics as shown in Table 1. Therefore, the proposed SLA data model enables defining various types of breaches $bc_i \in BC$ based on multiple quality requirements. Observe Fig. 14 which associates every $bc_i \in BC$ to multiple quality requirements. Moreover, the SLA data model enables assigning distinctive max tolerance and penalty to every bc_i; considering that each of them has a different criticality level and, therefore, distinctive consequences.

Performance reports

The enhanced SLA data model accommodate various breach categories in the form of $[bc_j : b]$, where bc_j is a unique identifier of a breach category, and b is the count of its occurrence. Caped by the storage capacity, the smart contract can create as many performance reports as long as there are incidents reported by the monitoring manager. Section Processing Received Monitoring Metrics further elaborates the logic of smart contract with performance reports. Therefore, the performance report does not consist of the properties *validity* or *compliance status*.

Aggregated assessment

The enhanced SLA data model introduces a component called *aggregated assessment*. For each billing cycle (i.e. monthly), the smart contract can create a new aggregated assessment instance to aggregate all existing performance reports. As can be seen in the enhanced SLA data model, it accumulates the total count of compliant cases c and count of breaches b of each breach category bc_j. It also holds the violation rate vr for breach category against the total compliant cases c. The smart contract also uses the *aggregated assessment* to determine the overall compliance of the service provider. The validity property reflects the relevancy of this assessment to the current billing cycle.

Processing received monitoring metrics

Recall that the monitoring manager submits transactions to the blockchain-side upon the occurrence of an incident of either B or f. Refer to Algorithm 1 line 24, which reports a payload of collected metrics in the form of $M = (f, t_s, A_{edge}, A_{server})$. Based on the enhanced SLA data model, this section addresses the limitations our previous study in [18]. Algorithm 2 overviews a smart contract method for processing and evaluating received metrics M. As long as the SLA is valid $\lambda = true$, it accepts transactions from the monitoring-side and evaluates received metrics M against the respective quality requirements. As a result, the evaluation process classifies the performance of the IoTSP to be either compliant c or one of the predefined breach categories bc_i. Examples of breach categories are provided in section SLA between IoTSP and Firefighter Station and illustrated in Table 1.

For every metrics evaluation, the smart contract creates a new performance record in the form of a tuple $(k+1, pr, ver)$, where

- $k+1$ is a unique identifier of the performance report.
- pr is performance report that holds the result of a metric evaluation against breaches categories $bc_i \in BC$.
- ver is version that tracks modifications on the performance record.

We consider pr to consist of $(c, [bc_j : b])$ where c indicates the count of compliant cases, and $[bc_j : b]$ indicates a the frequency of an incident belonging to a breach category, where bc_j is a unique identifier of a breach category, and b is the count of its occurrence. Example of evaluation outcomes are as follows:

- *compliance case*: $pr \leftarrow (1, \emptyset)$.
- *breach case*: $pr \leftarrow (\emptyset, [0002 : 1])$.

As Algorithm 2 demonstrates, we opt to avoid the practice of updating an existing performance report (k, e', ver'). Instead, the proposed design dictates that there must be a newly created record $(k+1, e, ver)$ for every subsequent metric evaluation process. Once an evaluation record is created, it shall never be updated but may only be used for query purposes. In this way, we ensure there will always be one write operation, and therefore the version ver would not change at all. This perpetually mitigates the issue of conflicting read-write sets associated with the high rate of monitoring transactions per block.

Require: $M = (f, t_s, A_{edge}, A_{server})$
Output: $(k : pr)$ ▷ Evaluation Result Record
1: $bc_j \subseteq BC \mid i \in \mathbb{N}_{>0}$ ▷ Breach Category
2: $\lambda \leftarrow true$ ▷ SLA Active
3: $k \leftarrow 0$ ▷ key of Performance Record
4: **repeat**
5: **if** $M \neq \emptyset$ **then**
6: $k++$
7: **if** $M \in BC$ **then**
8: $pr \leftarrow (sp_i, \emptyset, (bc_j : b))$ ▷ breach case
9: **else**
10: $pr \leftarrow [sp_i, c, \emptyset]$ ▷ compliant case
11: **end if**
12: create $(k : pr)$ ▷ create performance record
13: **end if**
14: **until** $\lambda = false$ ▷ SLA Termination

Algorithm 2 Evaluation of Received Monitoring Metrics

Compliance assessment and enforcement

The smart contract can be instructed to periodically conduct an overall compliance assessment. We consider that *read operations* do not cause version modification, and

Table 2 Blockchain deployment and configurations

Element	Description
Hyperledger Fabric	Fabric version (2.3.2)
Blockchain Network	See Fig. 6
Blocks Frequency Configuration	- Transactions per Block: 10. - Timeout: 1s. - No size restrictions.
Smart Contract Language	Java
Chaincode Timeout	30 seconds
Benchmark Tool	- Hyperledger Caliper V0.4.2 - 5 workers
Consensus Protocol	Raft
State Storage	CouchDB
Resources Allocation	- 32 x vCPU Intel(R) Xeon(R) Gold 6140 @2.30GHz - 64GB RAM.
Operating System and Docker	- Ubuntu Linux 20.04.2 (64-bit). - Docker Version 20.10.6 (No restrictions on resources usage).

thus we do not expect MVCC conflicts. Therefore, the compliance assessment process should theoretically have the ability to aggregate all existing performance records at once $(k : pr) \mid i \leq k \leq n \mid k \in \mathbb{N}$.

Require: a set of $(k : pr) \mid i \leq k \leq n \mid k \in \mathbb{N}$ ▷ Existing Performance Reports
Output: Decision
1: $\lambda \leftarrow true$ ▷ SLA Active
2: $tc \not< 1$ ▷ Total Compliance Cases
3: $\forall bc_j \; \exists \; tb \leftarrow 0$ ▷ Total Breach Cases
4: $\forall bc_j \; \exists \; vr \leftarrow 0$ ▷ violation Rate
5: $\forall bc_j \; \exists \; mt$ ▷ Max tolerance to violation rate
6: $\forall bc_j \; \exists \; Penalty$ ▷ penalty
7: **for each** $(k : pr)$ **do**
8: **if** $true$ **then** ▷ compliant record?
9: $ts+ = c$ ▷ add count of compliant cases
10: **else**
11: **for each** bc_j **do** ▷ every identified breach type
12: $tb+ = b$ ▷ add count of violation cases
13: **end for**
14: **end if**
15: **end for**
16: **for each** $bc_j \in BC$ **do** ▷ Each Identified Breach Category
17: $vr = \frac{tb}{tb+tc} \times 100$ ▷ Calculate Breach Rate
18: **if** $vr > mt$ **then** ▷ Max Tolerance reached?
19: $\lambda \leftarrow false$ ▷ Terminate SLA
20: **else**
21: Apply $Penalty$ on the escrow account.
22: **end if**
23: **end for**
24: **if** $\lambda = false$ **then**
25: Issue full refund to consumer.
26: **else**
27: Release remaining amount to service provider, if any.
28: Refund to consumer, if any.
29: Remove all processed performance reports from the state storage.
30: **end if**

Algorithm 3 Concluding Assessment and Enforcement Logic

Algorithm 3 illustrates considering a set of performance records for the compliance assessment; for example from k_i to k_n. The overall aim is to calculate the violation rate vr of each breach category $bc_j \in BC \mid j \in \mathbb{N}$ by examining its ratio against the total compliance cases. For that, the smart contract examines every $(k_i : pr)$ and query its properties $(c, bc_j : b)$. It aggregates the total count of compliance cases, denoted as tc. Additionally, for every breach category $bc_j \in BC$, it aggregates the total count of breach cases, denoted as tb. Afterwards, the violation rate vr of every bc_j is calculated as per Eq. 1 and examined against the respective max tolerance mt. Note that the total compliance cases cannot be $tc \not< 1$ to prevent division on zero in case of no breach cases.

The smart contract can take actions based on the outcomes of the compliance assessment. For example, the smart contract can determines to terminate the SLA if the violation rate vr exceeds the max tolerance mr of any breach category bc_j (refer to Algorithm 3 Line 19). This leads the smart contract to issue a full refund and halt further metrics evaluations because Algorithm 2 does not process any incidents if the SLA is terminated. Otherwise, the smart contract can the aggregated assessment to make an informed decision on whether to enforce a penalty on the escrow account. Finally, the smart contract removes all processed performance records for the state storage to avoid reusing them for the next aggregated assessment. However, they still remain permanently stored on the blockchain for future auditing purposes.

Experiment and evaluation of the blockchain performance

The purposes of the experiment is to evaluate whether the proposed smart contract design proves to mitigate MVCC conflict issues while maintaining sound performance. Therefore, we experiment and stress the proposed approach to investigate in terms of throughput and latency. More specifically, this experiment is concerned with two tasks assigned to the smart contract, which are metrics evaluation and SLA compliance assessment as in Algorithm 2 and Algorithm 3; respectively.

Table 2 illustrates the deployment of the blockchain network as well as relevant configurations and specifications. We choose to deploy the blockchain network, as in Fig. 6, on cloud infrastructure, as specified in Table 2. We experiment on the latest HLF version, as of writing this thesis, and adopted the recommended consensus protocol; namely RAFT [30]. All default parameters of the test network provided by HLF remains intact except the block batching configurations. We employ a blockchain benchmarking tool called Hyperledger Caliper for experimenting the performance of the enhanced compliance assessment approach. The experiment considers the following:

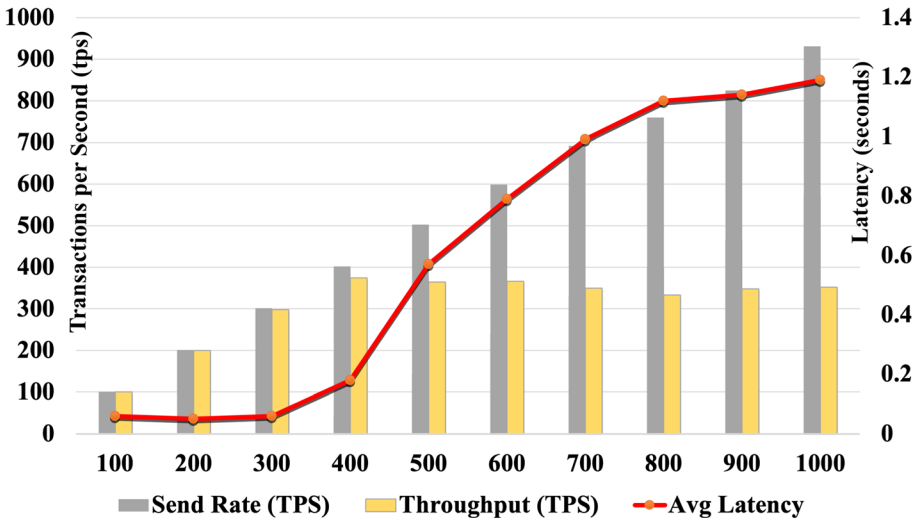


Fig. 15 Algorithm 2 performance: processing received metrics at variable rates and fixed total of 1000 transactions

- for each transaction execution, Algorithm 2 conducts a limited number of read operations (e.g. query quality requirements) and a single write operation (creating evaluation record). However, this algorithm is expected to be invoked very frequently; whenever the monitoring-side encounters an incident that requires attention. According to [31], using 5 worker for experimenting high rates of transactions seems to produce realistic results. We validated that from our side, and thus we use 5 workers for submitting transaction from the monitoring-side.
- for each transaction execution, Algorithm 3, conducts a massive number of read operations on existing evaluation records, as explained in section Compliance Assessment and Enforcement. It then results in a limited number of write operations which can include persisting the compliance assessment on the ledger, and handling the escrow account for enforcement purposes. Noteworthy mentioning that, this algorithm is only executed on limited occasions. For example, monthly billing, conducing payment, etc.

Fixed total transactions and variable rates

For Algorithm 2, we examine how it performs under various rates of transactions per second. We set 10 rounds, as in Fig. 15, where we fix total transactions to 1000. However, we increase the transaction submission rates by 100 Tps (Transactions per second) for each subsequent round. The aim is to investigate for each test round: (i) the send rate can be generated from the monitoring-side; (ii) the average throughput that can be archived at the blockchain-side; and (iii) the average round-trip transactions latency. We aim also to find out whether any of the transaction would encounter unforeseen failure such as MVCC conflicts.

As shown in Fig. 15, for all test rounds, there was no transaction failure at all. The send rate tends to be identical to the intended transactions rate until the seventh round, after which the send rate gradually exhibits a modest degradation. With regard to throughput,the benchmark shows an identical throughput to the send rate for the three first rounds. Thereafter, we observe an increasing transaction processing time and thus less throughput compared to send rate. Both studies in this [32, 33] justify this situation due the increasing queue length of transactions waiting for VSCC validation. In a study by [34], the long queue of transaction can be also attributed to a delay within the blockchain network. Nevertheless, the throughput flattened out for the rest of test round at approximately 380 transactions/second, with unremarkable changes. In a similar manner, the latency remains very low without major difference for the first 4 test round. Thereafter, it exhibits the possibility to break beyond 1 second, which is the max timeout set for block batching.

Variable total transactions and fixed average rate

To verify our outcomes, we fixed the send rate to 500 transaction/second, which is more than the best achieved throughput from the above benchmark. We relaxed the total transactions with a minimum of 1,000 and maximum of 10,000, where we increase 1000 transactions for each test round. The aim to see whether

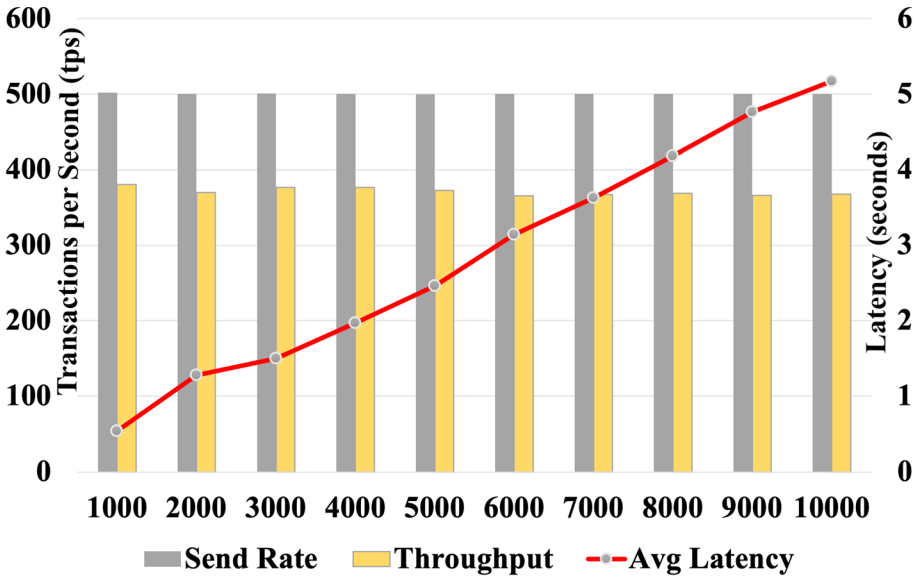


Fig. 16 Algorithm 2 performance: processing received metrics at fixed rate of 500 transactions and variable total transactions

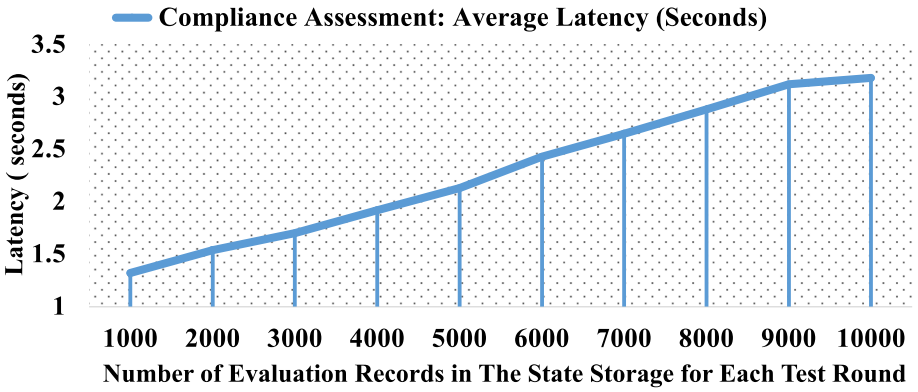


Fig. 17 Latency for executing Algorithm 3 on variable collection of evaluation records stored on HLF state storage

this relaxation would achieve better throughput or does it have a negative impact on it. We also investigate how to this relaxation can be correlated to latency. A shown in Fig. 16, there is no significant disruption in the throughput rate for all test rounds as long as the send rate is the same. Nevertheless, we observe an overall linear latency increase influenced by the linear increase workload of transactions. Consider the last round of Fig. 15 with first round of Fig. 16 , were all try to submit 1000 transaction/second but at different send rates. we observe that the latter achieve better latency than the former, which confirms a positive correlation between the send rate and expected latency [32, 33, 35].

All in all, the experiment reveals the ability of Algorithm 2 to accommodate a high rate of transactions without encountering MVCC conflicts. It also promises a sound performance, given the complexity of the smart contract logic and the blockchain configurations. We also report that, the experiments altogether did not consume more than 15% of the allocated resources, as illustrated in Table 2.

Compliance assessment execution time

Periodically, the smart contract aggregates and consumes a set of evaluation records for compliance assessment as illustrated in Algorithm 3. As shown in Fig. 17, we examine average latency of conducting the compliance on linearly variable number of stored records; increased by 1000 record for each round. For each round, we only need one transaction to trigger the compliance assessment, and thus we only focus on average latency and omit throughput. We observe that it exhibits a linear increase as a response to the amount

of stored evaluation records. Overall, the smart contract proved to execute 10,000 evaluation records within no more 3.5 seconds. We deem this to be satisfactory, given that it is a non frequent task conducted occasionally, for billing and concluding purposes, over a massive number of records. We also do not normally expect such number of records unless there is a breach B or a fire event f, as specified per section Monitoring-side.

Conclusion and future work

This paper presented a blockchain-based monitoring approach in the context of IoT application performance and SLA compliance. It focuses on shifting distrusted SLA-related processes such as metrics evaluation and compliance assessment to blockchain based smart contracts. It examines and discusses critical aspects and considerations at two key components of our framework; the monitoring-side, and the blockchain-side. While conventional software design strategies have proven to work well for centralised applications, we cannot safely assume the same in the context of distributed blockchain applications. As demonstrated in this work, it is vital to consider unique blockchain characteristics when designing a blockchain-based solution, such as transaction processing, execution behaviour, configurations and implemented protocols. This paper draws attention to the high rate of transactions emitted from IoT applications and consequently any deployed performance monitoring tools. For IoT applications that typically involve such high data rates, our work aims to resolve read-write multiversion concurrency control (MVCC) conflicts typically encountered in blockchain applications, while maintaining sound performance. From the monitoring-side of our framework, this paper demonstrates how we can determine the most critical co-factors of relation to SLA compliance assessment. It designs a monitoring architecture and reporting mechanism, which not only accounts for possible failed transactions, but also engineers a mechanism for metrics collection and reporting, with the aim to avoid overwhelming the blockchain-side with unnecessary transactions. From the blockchain-side of our framework, the approach of this paper revolves around a simple, yet effective principle that segregates between read and write operations at both levels; the smart contract design and data representation at the state storage.

Our work paves the way for future work to investigate improving the performance at the HLF infrastructure level. For example, finding optimal block configurations, which plays a vital role for throughput and latency. HLF modularity makes it also interesting to study the impact of different HLF's aspects on the overall performance, such as network size in terms of organisations, endorsing and committing peers. There is also the ordering service and employed consensus mechanism, chaincode configurations, smart contract programming languages and others.

Acknowledgements
We thank the Program Committee of IEEE SmartIoT for inviting us to extend our previous study [36] and submit to this Journal.

Authors' contributions
Ali Alzubaidi, as PhD (and post-PhD) researcher, performed detailed research including background research, literature review, methodology, implementation, and experimental evaluation, as well as writing of original drafts and later revisions. Karan Mitra: suggestions for literature review, methodology and experimental evaluation, PhD project supervision, and writing-reviewing and editing. Ellis Solaiman: project primary investigator, funding aquisition, and primary PhD project supervisor, suggestions for literature review, methodology and experimental evaluation, writing, reviewing and editing.

Funding
This work is funded in part by the EPSRC, under grant number EP/V042017/1. Scalable Circular Supply Chains for the Built Environment.

Availability of data and materials
All relevant materials are publicly available on a GitHub repository[5] since 01/December/2021.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 30 November 2021 Accepted: 14 February 2023
Published online: 31 March 2023

References

1. Huang J, Nicol DM (2013) Trust mechanisms for cloud computing. J Cloud Comput 2(1):9. http://journalofcloudcomputing.springeropen.com/articles/10.1186/2192-113X-2-9. Accessed 14 Oct 2020
2. Philipp Y, M Butler J, Theilmann W, Yahyapour R (2011) Service Level Agreements for Cloud Computing. Springer New York, New York. http://link.springer.com/10.1007/978-1-4614-1614-2. Accessed 14 Feb 2020
3. Wu L, Buyya R (2010) Service Level Agreement (SLA) in Utility Computing Systems. Grid Cloud Comput 286–310. https://arxiv.org/abs/1010.2881v1
4. ISO () ISO/IEC 19086-2:2018 - Cloud computing — Service level agreement (SLA) framework — Part 2: Metric model. https://www.iso.org/standard/67546.html. Accessed 17 Jan 2022
5. Comuzzi M, Kotsokalis C, Spanoudakis G, Yahyapour R (2009) Establishing and Monitoring SLAs in Complex Service Based Systems. In: 2009 IEEE International Conference on Web Services, IEEE, pp 783–790. http://ieeexplore.ieee.org/document/5175897/. Accessed 12 Apr 2018
6. Bakalos N, Kyriazis D, Protonotarios E, Varvarigou T, Barreto O, Juan A, Bantouna A, Demestichas P, Georgakopoulos A, Stamati T, Tsagkaris K, Vlacheas P (2016) SLA specification and reference model. https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5a07549af&appId=PPGMS. Accessed on 30 Nov 2020
7. Habib S, Hauke S, Ries S, Mühlhäuser M (2012) Trust as a facilitator in cloud computing: a survey. J Cloud Comput: Adv Syst Appl 1(1):19. http://journalofcloudcomputing.springeropen.com/articles/10.1186/2192-113X-1-19. Accessed 14 Oct 2020

[5] https://github.com/aakzubaidi/BlockchainQoT

8. Uriarte RB, de Nicola R, Kritikos K (2018) Towards Distributed SLA Management with Smart Contracts and Blockchain. In: 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, pp 266–271. https://ieeexplore.ieee.org/document/8591028/. Accessed 16 June 2019
9. Hussain W, Hussain FK, Hussain OK (2014) Maintaining Trust in Cloud Computing through SLA Monitoring. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol 8836, Springer Verlag, pp 690–697. http://link.springer.com/10.1007/978-3-319-12643-2_83. Accessed 30 Mar 2020
10. Rana OF, Warnier M, Quillinan TB, Brazier F, Cojocarasu D (2008) Managing Violations in Service Level Agreements. In: Grid Middleware and Services, Springer US, Boston, pp 349–358. http://link.springer.com/10.1007/978-0-387-78446-5-23. Accessed 21 May 2019
11. Kyriazis D (2013) Cloud Computing Service Level Agreements Exploitation of Research Results. European Commission Directorate General Communications Networks, Content and Technology Unit E2 - Software and Services, Cloud, Tech. rep. https://ec.europa.eu/digital-single-market/en/news/cloud-computing-service-level-agreements-exploitation-research-results
12. Labidi T, Mtibaa A, Gaaloul W, Tata S, Gargouri F (2017) Cloud SLA Modeling and Monitoring. In: 2017 IEEE International Conference on Services Computing (SCC), IEEE, pp 338–345. http://ieeexplore.ieee.org/document/8035003/. Accessed 12 Oct 2017
13. Alzubaidi A, Solaiman E, Patel P, Mitra K (2019) Blockchain-Based SLA Management in the Context of IoT. IT Prof 21(4):33–40. https://ieeexplore.ieee.org/document/8764077/. Accessed 13 Nov 2019
14. Scheid EJ, Rodrigues BB, Granville LZ, Stiller B (2019) Enabling dynamic SLA compensation using blockchain-based smart contracts. In: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019, pp 53–61. https://ieeexplore.ieee.org/document/8717859. Accessed 17 May 2021
15. OMG Cloud Working Group (2019) Practical Guide to Cloud Service Agreements Version 3.0. Object Management Group, Tech. rep. https://www.omg.org/cloud/deliverables/Practical-Guide-to-Cloud-Service-Agreements.pdf
16. Neidhardt N, Köhler C, Nüttgens M (2018) Cloud Service Billing and Service Level Agreement Monitoring based on Blockchain. EMISA Forum 38:46–50. http://ceur-ws.org/Vol-2097/paper11.pdf. Accessed 02 Aug 2021
17. Sunyaev A (2020) Distributed Ledger Technology. Internet Comput 265–299. https://link.springer.com/chapter/10.1007/978-3-030-34957-8-9. Accessed 06 Jan 2022
18. Alzubaidi A, Mitra K, Patel P, Solaiman E (2020) A Blockchain-based Approach for Assessing Compliance with SLA-guaranteed IoT Services. In: 2020 IEEE International Conference on Smart Internet of Things (SmartIoT), IEEE, pp 213–220. https://ieeexplore.ieee.org/document/9192398/. Accessed 02 Dec 2020
19. Chandrasekar A, Chandrasekar K, Mahadevan M, Varalakshmi P (2012) QoS Monitoring and Dynamic Trust Establishment in the Cloud. Springer, Berlin, Heidelberg, pp 289–301
20. Zhou H, de Laat C, Zhao Z (2018) Trustworthy Cloud Service Level Agreement Enforcement with Blockchain Based Smart Contract. In: 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, pp 255–260. https://ieeexplore.ieee.org/document/8591026/. Accessed 16 June 2019
21. Kochovski P, Stankovski V, Gec S, Faticanti F, Savi M, Siracusa D, Kum S (2020) Smart Contracts for Service-Level Agreements in Edge-to-Cloud Computing. J Grid Comput 18(4):673–690. https://doi.org/10.1007/s10723-020-09534-y
22. Uriarte RB, Zhou H, Kritikos K, Shi Z, Zhao Z, De Nicola R (2021) Distributed service-level agreement management with smart contracts and blockchain. Concurr Comput: Pract Experience 33(14). https://onlinelibrary.wiley.com/doi/10.1002/cpe.5800. Accessed 23 Oct 2021
23. Mubeen S, Asadollah SA, Papadopoulos AV, Ashjaei M, Pei-Breivold H, Behnam M (2018) Management of Service Level Agreements for Cloud Services in IoT: A Systematic Mapping Study. IEEE Access 6:30184–30207. https://ieeexplore.ieee.org/document/8016558/. Accessed 28 Jan 2020
24. Van der Wees Arthur, Daniele C, Jesus L, Edwards Mike, Schifano Nicholas, Maddalena SL (2014) Cloud Service Level Agreement Standardisation Guidelines. https://ec.europa.eu/digital-single-market/en/news/cloud-service-level-agreement-standardisation-guidelines. Accessed 13 May 2020
25. Pandey AK, G ND, K S (2021) SLA Violation Detection and Compensation in Cloud Environment using Blockchain. In: 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), IEEE, pp 1–6. https://ieeexplore.ieee.org/document/9580134/. Accessed 15 Jan 2022
26. Brazil B (2018) Prometheus: Up and Running: Infrastructure and Application Performance Monitoring. O'Reilly Media, Inc. https://www.oreilly.com/library/view/prometheus-up/9781492034131/
27. Androulaki E, Barger A, Bortnikov V, Cachin C, Christidis K, De Caro A, Enyeart D, Ferris C, Laventman G, Manevich Y, Muralidharan S, Murthy C, Nguyen B, Sethi M, Singh G, Smith K, Sorniotti A, Stathakopoulou C, Vukolić M, Cocco SW, Yellick J (2018) Hyperledger fabric. In: Proceedings of the Thirteenth EuroSys Conference, ACM, New York, pp 1–15. http://arxiv.org/abs/1801.10228. http://dx.doi.org/10.1145/3190508.3190538. https://dl.acm.org/doi/10.1145/3190508.3190538
28. Chacko JA, Mayer R, Jacobsen HA (2021) Why Do My Blockchain Transactions Fail? In: Proceedings of the 2021 International Conference on Management of Data, ACM, New York, pp 221–234. https://dl.acm.org/doi/10.1145/3448016.3452823. Accessed 29 Apr 2021
29. Meir H, Barger A, Manevich Y, Tock Y (2019) Lockless Transaction Isolation in Hyperledger Fabric. In: 2019 IEEE International Conference on Blockchain (Blockchain), IEEE, pp 59–66. https://ieeexplore.ieee.org/document/8946157/. Accessed 30 Jan 2020
30. Ongaro D, Ousterhout J (2014) In Search of an Understandable Consensus Algorithm. In: Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, ser. USENIX ATC'14. USENIX Association, USA, p 305–320
31. Hang L, Kim DH (2021) Optimal blockchain network construction methodology based on analysis of configurable components for enhancing Hyperledger Fabric performance. Blockchain: Res Appl 2(1):100009. https://linkinghub.elsevier.com/retrieve/pii/S209672092100004X. Accessed 15 Mar 2022
32. Kuzlu M, Pipattanasomporn M, Gurses L, Rahman S (2019) Performance Analysis of a Hyperledger Fabric Blockchain Framework: Throughput, Latency and Scalability. In: 2019 IEEE International Conference on Blockchain (Blockchain), IEEE, pp 536–540. https://ieeexplore.ieee.org/document/8946222/. Accessed 15 Feb 2020
33. Thakkar P, Nathan S, Vishwanathan B (2018) Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform. arXiv preprint arXiv:1805.11390
34. Sukhwani H, Wang N, Trivedi KS, Rindos A (2018) Performance Modeling of Hyperledger Fabric (Permissioned Blockchain Network). In: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), IEEE, pp 1–8. https://ieeexplore.ieee.org/document/8548070/. Accessed 17 Apr 2019
35. Baliga A, Solanki N, Verekar S, Pednekar A, Kamat P, Chatterjee S (2018) Performance Characterization of Hyperledger Fabric. In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), IEEE, pp 65–74. https://ieeexplore.ieee.org/document/8525394/. Accessed 17 Apr 2019
36. Alzubaidi A, Mitra K, Solaiman E (2021) Smart Contract Design Considerations for SLA Compliance Assessment in the Context of IoT. In: 2021 IEEE International Conference on Smart Internet of Things (SmartIoT), IEEE, pp 74–81. https://ieeexplore.ieee.org/document/9556177/. Accessed 30 Nov 2021

Publisher's Note