Dispute Mitigation Approach using Blockchain in The Context of IoT

Ali Alzubaidi, *Member, IEEE,* Ellis Solaiman, *Fellow, OSA,* and Someone else, *Life Fellow, IEEE*

***Abstract*—The abstract goes here.**

***Index Terms*—Blockchain, Smart contract, IoT, Monitoring, Performance, SLA.**

A. Basic IoT SLA Model

For the purposes of this paper, we formally model a basic SLA structure in the context of IoT. Recall our abstract definition of a typical set of IoT layers, as per defined in Equation 1, and consider the following:

- a set of participants, denoted as $\Upsilon = \{p \mid p \subseteq \{provider, consumer, auditor, ...\}\}$
- a set of measurable quality of services (QoS), denoted as $\Psi = \{q \mid q \subseteq \{uptime, latency, ...\}\}$
- a set of Service Level Objectives (SLO), denoted as $S_{1:n} = \{s_x \mid s_x = \{\Phi, \kappa, \alpha, \Upsilon\}$. That is, each QoS metric q_x maintains a set of properties which are
  - a definition that unambiguously states the intended service denoted
- a set of exclusions

I. Overview on Generic IoT Architecture

We understand the fact that IoT complexity does not enable smooth consensus on what a typical IoT architecture should look like, and that the composition of an IoT architecture is highly governed by distinctive industry requirements and various viewpoints. However, we conventionally abstract a basic set of IoT layers, covered in this paper, denoted as

$$\lambda = \{p, e, c, a\} \tag{1}$$

where $p \leftarrow$ *Physical layer*, $e \leftarrow$ *Edge layer*, $c \leftarrow$ *Cloud layer*, and $A \leftarrow$ *Application layer*.

This abstraction follows a bottom-up approach governed by a simple rule such that each layer does not delegate a task to the upper, unless it is justifiable and reasonable to do so. Following, we delve further to present how this paper views each layer.

M. Shell was with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA e-mail: (see http://www.michaelshell.org/contact.html).

J. Doe and J. Doe are with Anonymous University.

Manuscript received April 19, 2005; revised August 26, 2015.

A. Physical Layer

at this layer, we consider a set of field deployed entities that can observe it environment (sensors) or able to carry out physical actions (actuators). an example of sensors are flame detector that can be used to observe for fire events. One the other hand, example of actuators cab be alarm devices that are triggered by fire events.

These physical entities are typically resources-constrained in terms of power/battery, memory, processing and storage [1]. It is also typical for such entities to lack essential capabilities such as:

- connectivity (e.g. WiFi, Bluetooth, Zigbee, NFC, etc.).
- communication (e.g. HTTP, MQTT, XMPP, CoAP, etc.).
- recognised data format (e.g. XML, JSON, etc.).

B. Edge Layer

at this layer, we consider a capable edge computing unit that and manages such IoT field assets deployed at the physical layer. In a basic point of view, we consider the edge computing unit to play, at least, the following roles [1]:

- empowers resources-constrained entities by providing extra capability that they lack.
- executes and monitors the business logic and relevant processes from near proximity.
- it forms an entry point that that bridges the gap between local field deployment and the external world.

While the edge paradigm brings computation and capabilities closer to the field, it limited in with respect to long terms storage, big data analysis, and handling server capabilities such as large network traffic.

C. Cloud Layer

in this paper we consider a central point where geographically dispersed IoT assets (including edge nodes and their associated IoT field assets) can be authenticated, accessed, managed and supported [2]. It also plays a key role in data governance, persistence, analysis and decision-making process. In this sense, the cloud side, c represents provided services that are either other classic cloud service such as infrastructure, platforms, software, etc.) or IoT-specific services such as remote accessibility, IoT assets management, visualisation, big data storage and analytics. It also facilitate typical IT overhead such as scalability, security, and maintenance.

Finally, it enables external entities, such as the fire-station, to remotely consume generated data via REST APIs in a loosely-coupled manner. That is, the level of abstraction and

separation of concern enable external applications and relevant stakeholders to consume data without needing for brute-force integration with sources of these data sets (IoT assets), or concerning how they are being implemented.

D. Application Layer

This paper considers this layer to represent applications that generate actionable value out of IoT assets (e.g. datasets, deployed sensors, actuators, devices, etc.). For example, we can think of an application that visualises IoT datasets, remotely monitors and actuate field assets, automates business logic, and enables informed decision-making. For instance, consider a Fire station that aims to embrace IoT for mitigating fire risks in the shortest time possible. In this case, the Fire station actuates data generated by field sensors deployed at residents premises. The exchange between fire station and remote IoT assets is enabled via REST API hosted and governed by a centralised cloud service.

II. Description of the Example IoT System

This section describes a basic IoT ecosystem (an IoT-based fire fighting system) that we designed and implemented for examination purposes, and to clearly define the scope of this study.

A. Fire detection and Alert Processing

the IoT-based fire mitigation system presented in Figure **??** is centred around fire detection and alert processing. Therefore, this system is event-driven in nature, such that once a threshold is reached, an alert is triggered. For that, sensors readings are evaluated in real-time against certain thresholds to help forming a decision on whether to trigger a set of actions. For example, if the reading of the flame sensor reached the specified threshold, An alert will be issued to notify the fire station. Following, we overview main workflow of defecting, forming a decision and reporting fire events processes at both levels, the edge and IoT server.

1) Alerting Logic at The Edge-Side: In our presented scenario, the edge layer is one of the responsibilities assigned to the IoTSP. The edge computing unit conducts the logic presented in Figure 1.

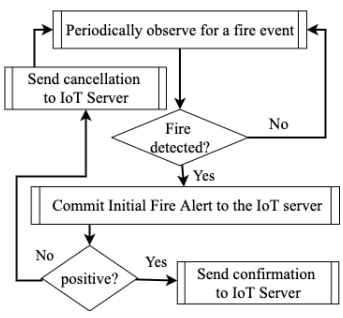


Fig. 1: Fire detection and alerting at edge level.

The main task for it is to ceaselessly observe the environment in real-time (e.g. every second) and investigate, reasonably enough for demonstration purposes, whether a fire alert should be issued. Once a fire event is detected, it instantiates a fire alert and informs the IoT server for a possible fire event. That is, the alert will not be confirmed immediately in order to prevent false positive. To form a decision on the validity of alert, the edge will subsequently undergo an evaluation processes within a specified duration. As a result, the edge will follow the initial alert with a confirmation or cancellation to the IoT server.

2) Alerting Logic at The Server-side: The responsibility for the IoT server is also assigned to the IoTSP. One of the IoT server tasks is to listen for fire events. Whenever it receives an initial alert, it expects to receive further exchange within the specified duration about the same alert; as to whether confirm or discard the initial alert. If confirmation is received, then the IoT server must report the fire event to the fire station; otherwise, it will discard it. In th event that no further exchange is sent by the edge computing unit, it will assume the worst case scenario as a safety measure. For instance, the reported fire has been extended to edge computing unit or the Internet gateway. In this case, the IoT server will take the initiative to self-confirm the alert and, which will trigger the action of sending it to the fire station.

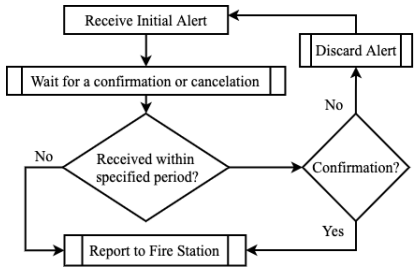


Fig. 2: Alert handling at IoT sever level

B. Fire Detection

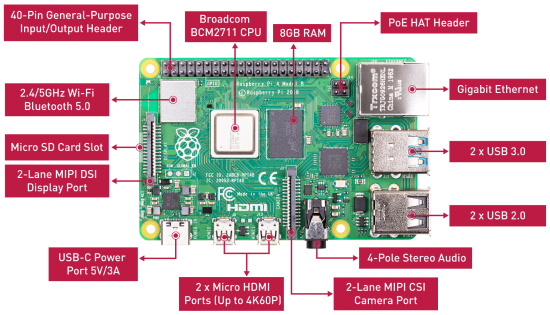
For this study, we employed a real flame sensing module, conventionally known as KY-026. Although it does not scale to industrial level in terms of specification or coverage, we deem it sufficient for the purposes of detecting infrared flames radiation. For that, it depends on optical detection method using YG1006 infrared diode, typically able to identify wavebands within the range of $\approx 0.7\,\mu\text{m}$ to $\approx 1.1\,\mu\text{m}$ [3].

Figure 3a illustrates both the pinouts and the infrared diode of the flame sensing module. In our study we do not make use of the analog output since the digital output is sufficient for our purposes, which emits signals that are either true or false as illustrated in Table I. That is, once the infrared diode detects a flame we expect the digital output to emit true; otherwise false.

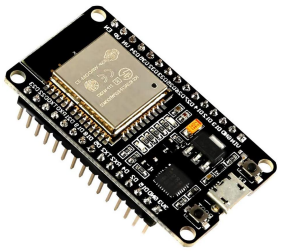
TABLE I: Mapping flame states to digital outputs

Active Flame	Infrared Radiation	Logical Output
Yes	High	True
No	Low	False

(a) Components of the flame sensing module, known as KY-026.



(b) Raspberry PI 4



(c) ESP-WROOM-32 NodeMCU module

Fig. 3: A set of hardware components considered for the edge layer

C. Edge Computing Unit

We understand that, the flame sensing modules, presented in section II-B is capable of observing its environment and emitting data about the flame activities. While essential, it lacks several capabilities such as sufficient processing, memory and storage. It is also short of connectivity and communication methods required for publishing machine-readable data over the internet. For example, but not limited to,

- such a sensing module is not IP-enabled, and thus does not support transfer protocols such as TCP or UDP and typical application protocols such as HTTP or MQTT.
- it lacks connectivity mechanism such as Ethernet, Wifi, Bluetooth, ZigBee, LoRa etc.
- it does not support machine-readable representation protocols such as XML or JSON.

Such shortcomings can be complemented by physically connecting to external capable components. To this end, a wide variety of alternatives can be employed, for bridging the gap and expanding further capabilities. Alternatives can includes, but not limited to, Microcontroller-based development boards (μCU), Single-Board Computer (SBC), extended computing units, gateways, etc. Depending on distinctive edge requirements, these can be selected individually or combined to serve different purposes.

*1) **Hardware Selection**:* in the presented scenario, an internet connectivity gateway (e.g. a typical WiFi router) and reasonable computing units such as SBC, or μCU, are sufficient for implementing the the edge layer of our scenario. Popular examples of SBC and μCU are Raspberry PI (RPI4), as shown in Figures 3b, and ESP32 Nodemcu as shown in Figure 3c; respectively. See both in and However, we selected the former over the latter because it provides adequate capabilities needed for demonstrating our monitoring approach, as will be discussed in section **??**.

Table II presents most relevant factors that we considered for running a comparative analysis between RPI 4 and ESP-WROOM-32 Nodemcu. Both can be qualified candidates for implementing the intended logic of the edge layer for many reasons, First, both of them provide General-Purpose Input/Output (GPIO) header compatible for integrating with and connecting to the flame sensing module. For instance, using their GPIOs enable data acquisition from the flame module and provide a power supply of 3.3V needed for operating it. Second, they provided connectivity to the internet via a Wifi interface.

TABLE II: Related analysis factors between Raspberry PI4 and ESP-WROON-32

Factor	Raspberry PI 4	ESP-WROOM-32
Type	SBC	μCU
Processing unit	Quad core Cortex-A72 ARM 64-bit @ 1.5GHz	Dual-core Xtensa LX6 32-bit @ 240 MHz
Memory	8GB – LPDDR4	512 KB SRAM
Persistence storage	SD CARD	N/A
Operating systems	Unix-like	FreeRTOS
GPIO support	yes	yes
Wifi Connectivity	802.11b/g/n/ac 2.4/5GHz	802.11 b/g/n only 2.4 5GHz
Output voltage	3.3V and 5V	only 3.3V

It is a fact that, both of them can provide processing sufficient for realising an HTTP client and handling our basic edge implementation. Nevertheless, since the ESP-WROOM-32 is specifically designed to be constrained and cost-effective, it lacks native support for adequate local storage, memory and processing capabilities. It also does not support general purposes operating systems required for our approach for edge monitoring; as will be explained in following sections. For that, we opted for Raspberry PI 4 to serve as follows:

- acts as a computing hub for the flame module.
- hosts and operates HTTP client implementation.
- implement our edge monitoring approach.

*2) **Implemented Edge Logic**:* Being a client to the IoT server, it seeks authentication from the IoT server for itself as well as for associated assets (the flame sensor). It facilitate communication with the IoT server in a machine readable format; namely JSON. Whenever it classifies sensor data to be alarming, it must notifies the IoT server. When a first positive fire event is identified, the edge emits an initial alert to the server. In order to reduce false positives and negatives to minimum as possible, the edge does not issue a confirmed fire alert unless whichever of the following occurs first:

- a predefined quantity of consecutive positive sensor reading occur within an imposed timeframe, which increases the probability of a fire event.
- the time-frame elapses since the first positive flame read-

ing, but no subsequent sensor readings received, possibly because of a damage occur to the sensor since the last positive reading.

Algorithm 1 Edge Layer: simple event-trigger logic

Input: s ▷ sensor reading every second
Output: $Initial \parallel Confirmation \parallel Discarded$

$Intial\ Alert \leftarrow flase$ ▷ flag indicating an an initial alert
$Counter \leftarrow 0$ ▷ number of consecutive fire alert
Register to an IoT Server
while True **do** ▷ continuos operation
  if $s \leftarrow true$ **then** ▷ Fire event
    $Intial\ Alert \leftarrow true$
    $counter++$
    if counter $\leftarrow 1$ **then**
      emit $Possible\ fire$ to IoT server
    end if
    if $clock$ is NOT started **then**
      start $clock$ ▷ seconds
    end if
    if $counter \leftarrow threshold \parallel clock \leftarrow threshold$ **then**
      emit $Confirmed\ Alert$ to IoT server
      $Intial\ Alert \leftarrow false$
    end if
  else ▷ No fire
    $counter \leftarrow 0$
    if $Intial\ Alert \leftarrow true$ **then**
      Stop $clock$
      Send $Discarded$ to the IoT server
      $Intial\ Alert \leftarrow false$
    end if
  end if
end while

Since IoT ecosystems are complex, this design cannot safely claim to achieve ultimate reliability measures. While the presented design does not cover such reliability measures, we assume that it is sufficient enough for building an edge component for the purposes of this paper. That is, mitigating false positive or negatives should also handle other sources of failures such as end-user misbehaviour, electric faults, unintended disconnection between the sensing module and the computing unit, possible Internet connection issues, etc. Other reliability measures can include redundancy of sensors and computing units.

D. Data Model at IoT Server

For the purposes of this paper, we designed a data model at the server-side level, as illustrated in Figure 4, for representing field assets (Edge computing units and sensors). We use this model for persisting associated data, and for conducting typical CRUD operations (short for: Create, Read, Update, and Delete).

For readability, we can perceive the data model as follows:

1) for each edge unit, it may have a set of sensors.
2) for each sensor, it may have a record of associated alerts.

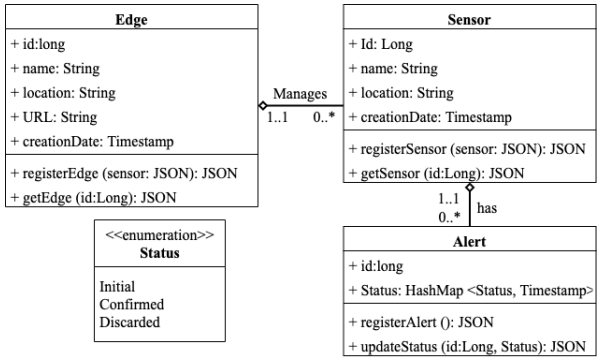


Fig. 4: Data Modelling at the IoT server side

3) for each alert, we keep track of its lifecycle (Initialised, Confirmed, or Discarded) as well as when a state change occurs.

E. REST HTTP API

We exposed a set of relevant CRUD functionalities as REST HTTP APIs to be consumed by the edge layer, as follows:

- registration of edge computing units and their associated assets (e.g. flame sensor).
- report a new alert (initial state).
- update the alert state to be either Confirmed or Cancelled.

F. Websocket Protocol

Since alerting requires immediate notification from the IoT server to Fire station, conventional HTTP request/response protocol is not perfect for that regard. That is, HTTP protocol would require the fire station to repeatedly request the IoT server querying for any fire events. A suitable alternative would be the concept of Websocket which maintains an open connection over TCP, enabling full-duplex communication established between the fire station and the IoT server [4]. Accordingly, this protocol enables the IoT server to actively send notifications about fire alert, and not merely responding to requests.

1) Limitations: The fault tree analysis presented above does not claim to be exhaustive or comprehensivly covering every possible source of a failure. However shows an example of how we systematically identify possible root-causes of a failure, which helps composing a monitoring strategy and smart contract decision making logic. While this paper uses a well-known failure analysis method, it is only for demonstrating the in-cooperation of such methods for identifying root-causes failures and their probabilities. This would help increase the confidence of the decision-making processes. For example, if a sensor has operated for a t duration, it implies that the a higher failure rate and thus the likelihood to be a root-cause of a failure. That is being said, this paper does not claim to model an ultimate failure analysis or restrict the construction of a failure model in this context. It also does not dictate what failure probabilities should be for each component or when how does it increase of time or under certain conditions. It only shows an example for demonstration purposes since the

main focus is to aid smart contracts in producing decisions with higher confidence. Therefore, it is best practice to refer to existing failure rate databases and concerned publications to observe what should be considered when constructing a failure model and conducting associated analyses. It is also an interesting problem to work on the event the dispute is caused by a claim that no alert has ever been delivered.

III. Methodology

1) Representing SLA as Blockchain Asset: Hyperledger Fabric provides a state storage that reflects latest state of persisted assets. It is important to recognise the difference between state storage and immutable ledger. State storage is amendable to reflect latest state of an asset. That is, Hyperledger Fabric enables typical CRUD operations at the state storage level (Create, Read, Update, and Delete). However, such operations do not reflect on any asset unless supported with a validated transactions that successfully meet the consensus protocol requirements. On the other hand, Hyperledger Fabric provides an immutable ledger to immutably record such transactions on the ledger. Having recognise the difference between immutable ledger of transactions and state storage of assets, we exploit the latter to design and store a data model that represents SLA and associated elements within blockchain, as shown in Figure 5.

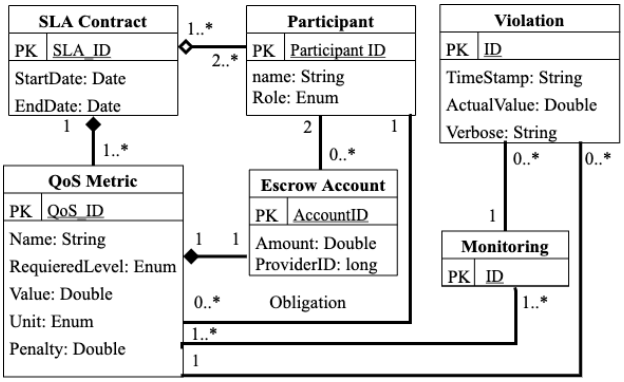


Fig. 5: Representing SLA and associated elements as blockchain assets

For the purpose of demonstrating our proposal, the presented UML diagram is simplified to support the minimum SLA structure. Essentially, we base our domain model on a standardised SLA grammar, contributed by Alqahtani et al [5], which provides a method for capturing and specifying an SLA for end-to-end IoT requirements. We adjust and extend the SLA model in that paper by considering an escrow accounts, monitoring tools, and violations. The association between different components are also adjusted.

The overall SLA model can be generally read as follows; an SLA is uniquely identifiable and is valid for a specific duration. It may comprise a set of participants and a set of QoS metrics. For every QoS metric, (e.g. QoS($Availability_{e,c}$) = 100%), there is exactly one participant obligated to satisfy the required level and responsible for violation consequences. There is an authenticated monitoring manager that is responsible for monitoring a set of metrics and reporting violations. There is also an escrow account which is valid as long as the associated QoS metric exists. An escrow account has to be associated with exactly two participants, which are the obligated participant and consumers. There consumer deposits the payment in advance to be realised when it is due.

For the purposes of this paper, the diagram only supports quantifiably measurable QoS metrics. Every QoS has to state the required service level, which could be whether GreaterThan, LessThan, NOT, or Equals against a predefined value. There is also defined a violation consequence that applies a financial penality on the escrow account. Notwithstanding, the presented SLA model is only for demonstrating the purposes of this paper, and can be extended and adjusted to serve other needs and purposes.

IV. Conclusion

The conclusion goes here.

References

[1] OpenFog Consortium Architecture Working Group, "OpenFog Reference Architecture for Fog Computing," *OpenFogConsortium*, no. February, pp. 1–162, 2017. [Online]. Available: https://www.openfogconsortium.org/wp-content/uploads/OpenFog{_}Reference{_}Architecture{_}2{_}09{_}17-FINAL.pdf

[2] X. Zeng, S. K. Garg, P. Strazdins, P. P. Jayaraman, D. Georgakopoulos, and R. Ranjan, "IOTSim: A simulator for analysing IoT applications," jan 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762116300662

[3] O. Khalaf Mohammed, O. Bayat, and H. M. Marhoon, "Design and implementation of integrated security and safety system based on internet of things," *Int. J. Eng. Technol.*, vol. 7, no. 4, pp. 5705–5711, apr 2018. [Online]. Available: www.sciencepubco.com/index.php/IJET

[4] "ISO - ISO/IEC 30141:2018 - Internet of Things (IoT) — Reference Architecture," 2018. [Online]. Available: https://www.iso.org/standard/65695.html

[5] A. Alqahtani, E. Solaiman, P. Patel, S. Dustdar, and R. Ranjan, "Service level agreement specification for end-to-end IoT application ecosystems," *Softw. Pract. Exp.*, vol. 49, no. 12, pp. 1689–1711, dec 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2747

PLACE PHOTO HERE

Michael Shell Biography text here.