IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 29, 2021, accepted March 30, 2021, date of publication April 5, 2021, date of current version April 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3070911

Design and Implementation of CryptoCargo: A Blockchain-Powered Smart Shipping Container for Vaccine Distribution

OMAR ALKHOORI[1], ABDURAOUF HASSAN[1], OMAR ALMANSOORI[1], MAZIN DEBE[1], KHALED SALAH[1], (Senior Member, IEEE), RAJA JAYARAMAN[2], JUNAID ARSHAD[3], AND MUHAMMAD HABIB UR REHMAN[1], (Senior Member, IEEE)

[1]Department of Electrical Engineering and Computer Science, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates
[2]Department of Industrial and Systems Engineering, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates
[3]School of Computing and Digital Technology, Birmingham City University, Birmingham B5 5JU, U.K.

Corresponding author: Raja Jayaraman (raja.jayaraman@ku.ac.ae)

This work was supported by the Khalifa University of Science and Technology under Award CIRA-2019-001.

ABSTRACT A large number of shipments are moved everyday domestically and internationally. A considerable number of items such as food, commodities, and pharmaceutical drugs are prone to damage in transit. This can be caused due to various reasons such as improper storage conditions and exposure to air or sunlight. The Internet of Things (IoT) has been used to enhance fundamental shipment tracking by improving transparency and visibility to such transport systems. This paper introduces a blockchain-powered smart container system (CryptoCargo) that monitors the conditions of the shipment and detects any violations that may damage its contents. These violations are recorded on the blockchain via smart contracts, which provides a secure and immutable storage thereby improving its trustworthiness in an inherently trustless environment comprising of multiple stakeholders. We present the design and implementation of CryptoCargo including architectural concerns and implementation details using a test Ethereum blockchain platform and cloud services. Moreover, we present details of thorough evaluation of the system to validate its function as well as to assess its effectiveness with respect to performance efficiency and real-time operation. We have made our smart contract code publicly available on Github.

INDEX TERMS Blockchain, IoT, cloud computing, supply chain, smart shipping, Ethereum, trust.

I. INTRODUCTION

A large number of containers are being shipped on a daily basis among numerous entities. As shbbown by [1], 753 million twenty-foot equivalent units (TEUs) of containers were handled in ports in 2017 worldwide. The absence of accurate real-time visibility to the state of freight throughout its shipment process is despised by stakeholders. Disputes and claims are likely to be raised if the shipped goods sustain any damage. In addition to the possible loss of the items, such disputes can result in additional costs and losses to all parties involved. Therefore, there is an omnipresent need for a practice that would help protect the rights of stakeholders and shippers and resolve conflicts.

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

As a shipment is being transported, it is susceptible to various types of damages. This depends on the sensitivity of the product being transported to the environment. A cold chain is typically implemented for these type of shipments. As shown in Fig. 1, a cold chain is a temperature-regulated supply chain that manages items that require storage in specific temperatures. The most common sources that could damage these items is subjecting them to extreme temperatures, humidity levels, and luminosity conditions. Special containers are used to move such sensitive items such as human organs, vaccines, chemicals, meat, dairy products, etc. Poor and incorrect packing of these transport units contributes to 65% of cargo damage claims [2]. Additionally, the integrity of resilient and reliable containers may still be compromised if it was opened or subjected to severe vibrations or shock movements. Shipments are also often vulnerable to burglary attempts; a recent study by BSI Supply Chain Services and

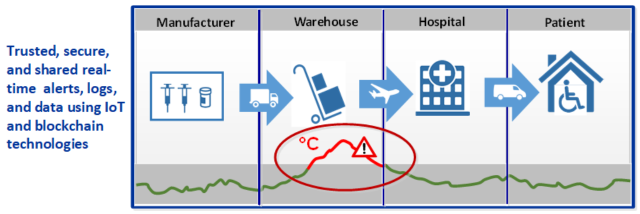

Trusted, secure, and shared real-time alerts, logs, and data using IoT and blockchain technologies
Manufacturer
Warehouse
Hospital
Patient
°C

FIGURE 1. Cold supply chain for temperature-sensitive shipments.

Solutions highlighted global cargo theft from road vehicles accounted for 84% of all modalities of theft [3]. Therefore, tracking the shipment path is as important as monitoring the vitals of shipment such as its temperature.

A report by Maersk revealed that a shipment of refrigerated goods from East Africa to Europe could move through nearly 30 different parties, with more than 200 different interactions among them [4]. At each stage, several risk factors threatens the health of the shipment and makes it susceptible to damage, which might increase the cost on the involved stakeholders. A study [5] declares that there are $140 billion tied up daily in disputes for payments in the transportation industry. This insinuates that organizations and individuals in a supply chain need real-time data about their shipments to make decisions. Therein, a dependable solution is required to ensure the safe-guarding of the cargo's integrity and the continuous tracking of the transported goods. In particular, the shipment needs to be continuously monitored for vital signs and provide immediate alerts to the stakeholders in case of detection of any abnormalities. Furthermore, all gathered data need to be securely stored and made available to stakeholders in a trustworthy manner as to protect against potential collusion or data tampering.

Our blockchain-powered smart shipping container aims to provide an enhanced supply chain management experience by offering real-time insights and increased visibility throughout the shipment process. Hence, allowing involved parties to collaborate together and work in a proactive manner. In brief, the main contributions of this paper can be summarized as follows:

- We examine the underlying causes that lead to shipments' damage and eventually disputes between the parties involved in a shipment's life-cycle.
- We propose a solution to monitor sensitive shipments by tracking it using a smart container. This container is capable of continuous measurement and recording of vital signs of the shipment's well-being and integrity, as well as constantly tracking its path.
- We design an automated monitoring system that detects real-time violations or abnormalities. This system utilizes blockchain smart contracts to record these violations by pushing immediate alerts on the immutable blockchain network where it is broadcasted to all relevant parties.
- We provide a cloud service as an off-chain storage solution for enhanced usability. The cloud server stores monitoring data that are not required to be recorded on the blockchain. We also create a user-friendly front-end decentralized application (DApp) that offers global access to the data stored in the cloud and on the blockchain.
- We implement the solution including the blockchain smart contracts,[1] cloud services, and web applications using their respective platforms. We integrate all parts of the system and perform elaborate testing using different scenarios. Finally, we provide comprehensive evaluation of the system performance.

The remainder of this paper is organized as follows: Section II presents background information about the technology used for the proposed system. Section III provides a brief review of the work done that is relevant to ours. Section IV presents the system architecture and design of the proposed system, while the implementation of this design is elaborated in section V. Section VI presents a thorough evaluation of the proposed system including testing and validation with respect to primary requirements. Section VII present the challenges faced and future improvements for this project. Section VIII concludes the paper.

II. BACKGROUND INFORMATION

This section will discuss important technologies used to develop the solution, including Microsoft Azure, blockchain, and Infura.

A. MICROSOFT AZURE

Microsoft Azure is a cloud computing service provided by Microsoft creating, developing, deploying, and managing applications, resources, and services. According to the NIST definition, Cloud computing is a model for providing ''ubiquitous, convenient, on-demand'' access to a shared pool of computing resources that can rapidly be launched with minimal service provider intervention [6]. Azure provides many services that provide different service models; infrastructure, platform, software, function, artificial intelligence, and smart contracts as services. For example, there are services for communicating with IoT devices, storage services, database servers, virtual computing machines, and web app services. Using the Azure Cloud, programs can be developed with availability, security, and scalability guarantees.

B. BLOCKCHAIN

Blockchain has been mostly known for its application in cryptocurrencies such as the Bitcoin [7]. Blockchain provides an immutable record that consists of a series of time-stamped, chained blocks of data that hold transactions [8]. Before executing and confirming transactions, miners first validate transactions and blocks of anonymous identities and then verify the availability of resources. Each block is linked to the previous block, thus ensuring that the chain is never broken and that the new block is permanently recorded. Depending upon the consensus algorithm used and other dynamics of the blockchain implementation, miners usually get rewarded for

[1] https://github.com/CryptoCargo/violations

verifying transactions, forming blocks, and appending them to the chain of blocks.

Blockchain's collective trust model is based on four main pillars: *a shared ledger, cryptography techniques, consensus protocols, and smart contracts*. It consists of a shared, immutable ledger which is not managed by a single node, but by a cluster of nodes in a decentralized peer-to-peer manner. The cryptographic technology used in the blockchain involves SHA-2 and Elliptic-curve cryptography (ECC). The blockchain can incorporate consensus protocols that guarantee consent between different entities making it more secure and fault-tolerant. Additionally, through the collective consensus, one single truth is always reached. Examples of such protocols are the Proof-of-Work (PoW) and Proof-of-Stake (PoS). For instance, PoW consensus protocol functions by having miners use the block header hash and nonce values to compete in solving a puzzle. The nature of the puzzle is asymmetric, meaning that it is hard for miners to solve it, yet it is easy for the network to verify it. The first miner to successfully solve it is considered the winner and is rewarded. This protocol is effective as attacking the network is not possible through since 51% of the peers in the network will be needed to collude together. In contrast, a different consensus protocol, Proof of Stake, 51% of the cryptocurrency available is needed to be acquired by one entity or a group of colluding members to put the network down. However, it can be assumed that the attackers would not be interested in a network where they own over half the stakes in it, and therefore, there is no need for it to exist [9]. The integrity of blocks in the blockchain is secured by including the Merkle tree in the header, which stores the hashes of state root, transaction root, and receipt root. A smart contract is a computer protocol that allows for code execution and helps transform the blockchain into a computational framework. It allows executing logic and storing state. However, smart contract functionality is not available on all blockchain distributed computing platforms.

These fundamentals concepts help make the decentralized blockchain network a permanent, append-only distributed ledger that represents a conclusive trust model. Through these features, the blockchain brings about increased transparency and auditing potential and increases the resiliency and integrity of the data [10], [11].

C. ETHEREUM

Blockchain-based distributed computing platforms are well known for their application in cryptocurrency. As opposed to Bitcoin, Ethereum allows for embedding business logic through its smart contracts besides the transfer of value (Ethers). A smart contract can be seen as a piece of software intended to enforce a set of pre-defined rules without the need for any third parties. Smart contracts allow for code execution and thus help transform the blockchain into a computational framework. Smart contracts are written in a high-level language called Solidity. Once compiled, they are transformed into bytecode, which is then read and executed on decentralized virtual machines known as Ethereum Virtual Machines (EVM). EVMs make use of their built-in instructions set and execute such scripts using international networks of public nodes. Smart contracts allow for increased autonomy, automation, and usability of the Ethereum network.

D. INFURA

A smart contract's development life cycle often iterates through a local development stage using a locally simulated blockchain network such as Ganache. Afterward, it goes through being deployed on the test Ethereum network (Testnet), before being finally deployed on the main Ethereum network (Mainnet). However, to interact with the Ethereum blockchain network, access to an Ethereum node in the network is needed. Infura is a hosted Ethereum node cluster that allows blockchain users to run their applications without the need to install full Ethereum nodes locally [12]. It provides a Web3.js API suite that enables the communication with the Ethereum network through the use of a remote node over Remote Procedure Calls (RPC) [13]. Web3.js, which is the most used JavaScript interface to the blockchain, allows for interaction with smart contracts as if they were JS objects. It converts JSON interfaces and calls into low-level Application Binary Interface (ABI) over RPC. Furthermore, Metamask is a browser extension that injects Web3 into the browser and hence delivers the same capabilities as Infura. In fact, Infura is the Ethereum provider that powers Metamask. Besides being a Web3 provider, Metamask also delivers an Ethereum wallet that acts as a browser-based wallet RPC client. Such features of Metamask has shaped it into becoming the standard for DApps. A DApp is an application that acts as the front-end for a distributed computing back-end that is typically paired with the blockchain. Integrating a DApp's front-end using Web3.js often requires additional libraries that can be found from Node.js. Node.js is an open-source JavaScript run-time environment that executes JavaScript code on the server-side. It is cross-platform and uses Chrome's JavaScript in run time for deploying fast and scalable network applications [14].

All of these technologies were utilized in order to achieve the system requirements. Explaining the foundations and concepts behind them is vital to fully comprehend the proposed approach and the technical choices. Cloud services, and specifically Microsoft Azure services were exploited along with the Ethereum blockchain network to complement the use of IoT in tracking sensitive shipments. The shipping container has a controller that connects to the cloud server to transfer data about the shipment. In addition, it contacts the Ethereum blockchain through the Infura gateway to report violations.

III. RELATED WORK

This section presents an overview of the relevant research in the field of monitoring and tracking shipments using decentralized technology. The blockchain technology has not yet been fully adopted to track shipments but several efforts have been made to achieve blockchain-based solutions for this use-case.

Some research work, such as in [15]–[17], argue that the application of the blockchain along with IoT devices can greatly complement supply chain solutions and provide extra value to the work flow by increasing its effectiveness and efficiency. Saberi *et al.*explored the role of blockchain in building a sustainable supply chain [15]. They presented some of the main obstacles that hinder the adoption of blockchain in a comprehensive manner. The authors categorizes these into four categories: inter-organisational, intraorganisational, technical, and external barriers. Francisco *et al.*provided a conceptual model based on the Unified Theory of Acceptance and Use of Technology (UTAUT) [16]. The authors utilize this model to present several research propositions that empower the endorsement of the blockchain technology in the field of supply chain. This model offers better transparency and traceability of the supply chain by endorsing six of the research variables mentioned in the UTAUT. In addition to general application of the blockchain in the realm of supply chain, Rejeb *et al.*address this implementation by combining the blockchain technology with IoT [17]. The authors in this paper discuss how this consolidation can help enhance the current supply chain applications. They specifically propose six key suggestions that summarize how leveraging the blockchain technology can benefit IoT applications.

A lot of research has been conducted on applying the blockchain technology to IoT-based solutions. However, we are specifically interested in employing the blockchain network on IoT devices used for shipping purposes. Practically implemented blockchain-IoT solutions for shipping are hardly found in the literature. Despite that, the theoretical work for potential adoption of the blockchain in this industry is sufficient enough to be used as building blocks to develop and implement our solution. Yang discusses the intent for adopting the blockchain technology to digitalize current maritime shipping [18]. The author conducts a study and presents key factors that affect this adoption and how can they increase the efficacy and effectiveness of maritime shipping. The study also examines the usability of the blockchain technology to the operators of existing supply chains. In addition, the challenges that face implementing the blockchain technology are presented in this paper, such as revising and unifying existing standards, the system scalability, interoperability, and the ambiguity regarding regulatory affairs. Di Gregorio *et al.*also describe in a study the likeliness of introducing blockchain in maritime shipping [19]. This is done through presenting research findings in addition to conducting several interviews with professionals from the shipping industry. The aggregated is analysed with scenario-based techniques and using a model known as TASC model. The authors presented their findings and showcased the opportunities of deploying the blockchain technology as well as the potential risks. This helps managers make an informed decision by taking advantage of the available opportunity of blockchain while minimizing risks and uncertainties. Jović *et al.*also review several noteworthy examples of attempts at digitizing seaports [20]. The authors also identify some crucial drawbacks that are inhibiting the introduction of blockchain, such as the lack of a common standard, scalability issues, and the learning curve for learning blockchain concepts. Li *et al.*present a comprehensive survey of seven blockchain-based applications in the shipping industry [21]. In light of these applications, the authors discuss the value added by the blockchain and improvements in terms of cost, transparency, and effectiveness. Furthermore, several critical aspects are recognized in order to utilize the blockchain efficiently. These include but are not limited to scalability concerns, security and interoperability, and legal compliance.

Tsiulin *et al.*also examined in their paper the tendency of employing the blockchain in the shipping industry [22]. The paper also discusses the feasibility of using the blockchain technology into the current existing supply chain workflow. The authors conduct a thorough review of the available literature and projects and construct a framework realizing the effect of blockchain in the realm of maritime port management. They report that current blockchain-based applications tend to prioritize the same features in contrast to traditional solutions. Bavassano *et al.*published a study to explain clearly the possible impact that the blockchain technology could have on the shipping industry and logistics [23]. By exploring the literature, data from the media, and input from, this study specifies the actors that affect the blockchain adoption the most in addition to the biggest challenges as well. The paper also demonstrates various options available on the market. The authors conclude from their study that the biggest challenge that faces full adoption of blockchain is the regulatory standard, or rather the lack thereof. In addition to the research done on the potential of blockchain in the field of shipping, a paper by Iakushkin *et al.*presented an implementation for a blockchain-supported smart container [24]. This container provides users the ability to track the location of the shipment as it is transported. In addition, the container is locked and can only be accessed by users that are pre-registered users recorded using smart contracts.

Besides the work done in the literature on the implementation of blockchain in the supply chains, several implementations of blockchain-powered shipping have been introduced by leading firms. With the recent advancements in IoT and blockchain solutions, companies in the transportation and logistics industry have adopted such technologies to improve their quality of service. Maersk, which is responsible for 18% of the global container trade, has introduced its Remote Container Management (RCM) solution to its fleet of 300,000 refrigerated containers [25]. Through its piloted RCM, Maersk enabled its containers to monitor and send data such as temperature and location to its cloud. By doing so, Maersk has managed to lower its physical inspection to only inspect 60% of its containers. In 2018, Maersk and IBM ventured to develop the TradeLens platform, which they described as the first blockchain-powered platform for supply chain applications in the industry [26]. Their use of blockchain is intended to offer better paper trail management, where the paperwork regarding shipment information and officials' approvals are all submitted through a blockchain digitized infrastructure. In doing so, they aim to reduce delays by automating the paperwork, eliminate frauds and errors, and minimize costs, which is expected to be the equivalent of only 1/5 of the actual physical paperwork costs. TradeLens has been recently put in a trial, as Maersk has been working with ecosystem partners to identify opportunities to prevent delays and have managed to record more than 154 million shipping events. Similarly, in an ICO, the Swiss company Smart Containers has revealed its aims to combine IoT sensors and blockchain to rent out airfreight containers used for food and medicine transportation [27]. Smart Containers claim that their products, Food Guardians and Sky Cell, allow them to lower the temperature deviations of sensitive pharmaceutical shipments to 0.1%, compared to the average rate of 8.5%, which rises to 15-20% in the Middle East [5]. Likewise, the purpose behind their adoption to the blockchain is to better manage paper documentations.

IV. BLOCKCHAIN-BASED SOLUTION

In this paper, we propose a cost-effective blockchain-powered shipping container that aims to address the need to have a traceable supply chain, where each stakeholder can view the status and progress of its shipment. Accurate vital data points about the container are continuously monitored with built-in sensors and pushed through a real-time exchange mechanism to the cloud. Users will be equipped with a mobile application that pulls the data from the cloud. Similarly, in case of detection of any abnormalities, users will be immediately notified with data that is retrieved from a security-rich, transparent blockchain network.

We propose a blockchain-based container that is designed to achieve the integrity and safety of the items being delivered by continuously tracking its location and vital signs through its journey. The smart container is designed to encompass sensors that will provide precise data readings pertaining to temperature conditions, container integrity, and position tracking. Our choices of the actual physical components used are impacted by our intentions of making the solution highly efficient, as well as maintaining cost-effectiveness and size flexibility. They are also affected by the characteristics of the environment in which our product operates. The container accurately processes and manages the data received from the sensors as well as communicating them in real-time to the different stakeholders. After refining the data, it is pushed to an always-available cloud server. Users can then retrieve backups of the data by pulling them via the mobile application. Most importantly, any violations recorded need to be registered on the blockchain, and subsequently, all relevant participants will be notified via the connected application. By utilizing the blockchain, we inherit its peer-to-peer trust model, where no single entity can modify any record without the consensus of others on the network.

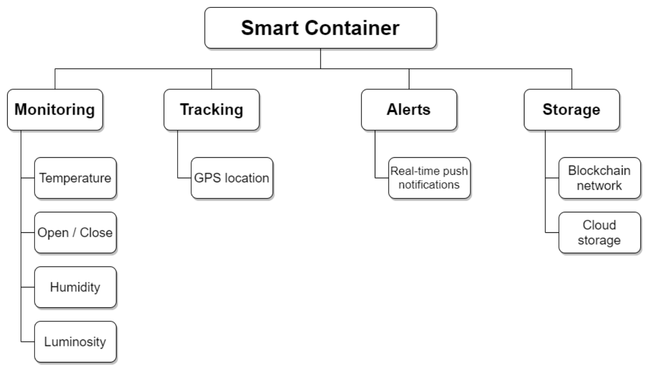


FIGURE 2. Main capabilities of the blockchain-powered smart container.

As highlighted in Fig. 2, the key features of our proposed solution are: **monitoring item status**, **tracking shipment**, **alerting relevant entities of any breach**, and **secure and reliable record of shipment data and events for customer consumption**.

In order to achieve its key features, sensors are placed to monitor and measure the temperature, humidity levels, and light exposure, as well as detecting when the container is opened. The container is being tracked at all times using a GPS logger. This can be used to alert if a shipment is going out, of course, in critical scenarios. Whenever one of the monitoring metrics is violated, such as opening the container without permission, the user gets a real-time update and is notified instantly on their mobile device. Finally, all records and events need to be stored in a way that makes them accessible from anywhere. The storage feature will involve both cloud services and the blockchain. The cloud services are used for providing live monitoring metrics of the container as well as other non-critical data. The blockchain is used to securely record any detected violations, which means they are tamper-proof. The cloud is used to store some data as the blockchain is expensive to conduct transactions and is not suitable for transactions of larger sizes. Furthermore, blockchains by design are not computationally efficient for query processing and therefore storing all data on blockchain risks performance efficiency.

A. SYSTEM ARCHITECTURE

The proposed system consists of four major subsystems: the container, the cloud, the blockchain, and the web application. The overall system follows a client-server model and uses different communication mediums and protocols to handle the interoperability of these diverse subsystems. A high-level overview of the proposed architecture is presented in Fig. 3. The IoT container communicates with both Ethereum blockchain (to record violations) and the cloud (to store non-critical data). To connect the container with Ethereum and the cloud, a 4G connection is used. The Infura Ethereum client is utilized to perform API calls between both the container and mobile client with Infura and thus, with Ethereum. The mobile client adopts a simpler way of

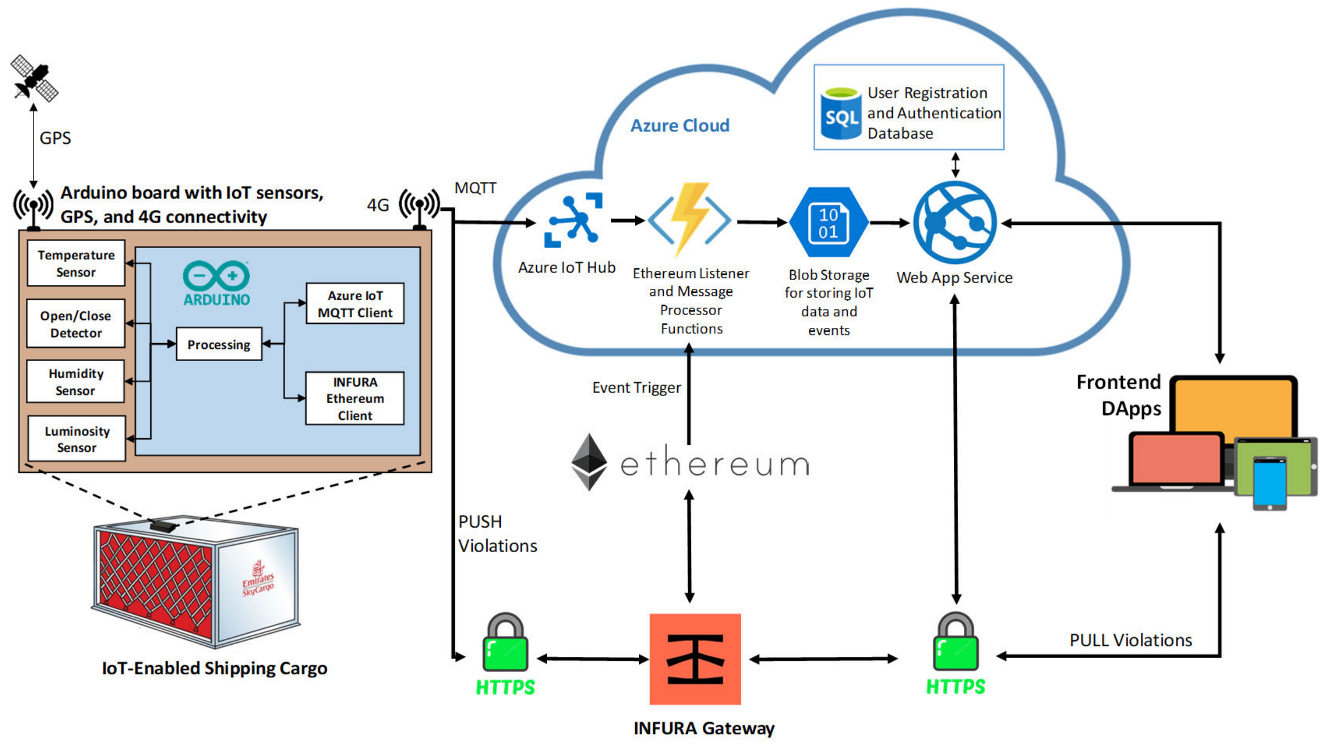


FIGURE 3. Proposed system architecture for the blockchain-based smart shipping container.

performing transactions between the clients and the Ethereum smart contract. It sends messages to Azure's IoT Hub service, which acts as a gateway to communicate with the cloud over MQTT.

The cloud subsystem of the smart shipping container handles the monitoring and tracking of the container, as well as the user to device messaging during the setup stage. A web application is hosted on the cloud to provide the monitoring means for users. Sensor readings are routed from the IoT Hub and stored in the blob storage. The stored data is then picked up by the web application and is presented visually to the users through the application front-end. In addition, the web application utilizes a database to store different users, along with their corresponding registered shipping containers. Moreover, the application listens for any violation on the Ethereum smart contract and sends a notification email to the user. Users can access monitoring and tracking features via a web browser through the web application.

B. COMMUNICATION MODEL

In view of the complex interactions between different subsystems, it is paramount to generate a communication model to understand how these subsystems exchange information. The smart shipping containers are equipped with SDKs and libraries to communicate with different APIs of various servers. The on-board Infura client uses APIs that communicate over HTTPS, while the Azure IoT Hub will use its own protocols to communicate with the shipping container via MQTT. The user can access the data stored in the cloud's database via an HTTPS request. In addition, it is important to note how the container will be able to establish this communication. For the use-case of the shipment container, it is most logical to use a 4G connection rather than Wi-Fi as the package would constantly be on the move. This setup required a separate slave device connected serially to with our master device, which utilizes the Web3 APIs separately. With all these points in perspective, the model presented in Fig. 4 was devised.

C. SYSTEM DESIGN

This section discusses the technology choices taken to achieve the objectives of the system. The decisions were made regarding cloud services and the blockchain infrastructure.

1) CLOUD AND SERVER TECHNOLOGY

Choosing to work with a cloud solution for this IoT system is a lot more favorable as compared to establishing an MQTT or HTTP server from scratch. Cloud platforms provide high availability, secure and safe data transfer, and storage and processing capabilities. In addition, it significantly minimizes the developmental time and expense. The cloud platform is used to handle messages from the devices, manage data storage, and host the web-based application of the system, as well as programs that need to be always running. There are many available IoT Cloud solution providers in the market. Some examples include Azure IoT Hub, AWS IoT Core, and Google Cloud IoT Core. All of the mentioned services

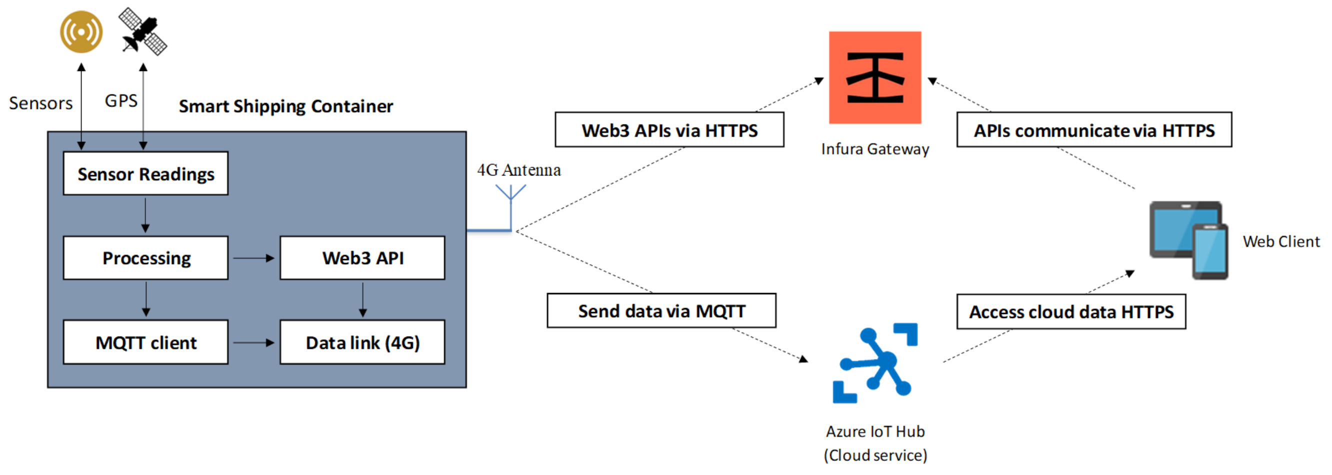


FIGURE 4. **Communication model demonstrating how the system components exchange information.**

provide a reliable and secure IoT solution platform that could scale to millions of devices. Azure's IoT Hub was selected because it can be integrated with other Azure services seamlessly. Azure IoT Hub is also versatile as it supports MQTT, HTTPS, AMQP, and web sockets [28]. Moreover, using Azure IoT Hub for the development of this purpose was suitable as a free pricing tier that allowed for up to 8,000 messages per day [29], which is sufficient for the development and testing purposes. The technology selection choices and decision making does not stop with selecting the cloud service provider. It was also critical to design the architecture of the smart shipping container system on the cloud, which includes selecting services to use and interactions among them. Fig. 5 showcases the design of the architecture of the Azure-hosted subsystem of the system.

The IoT Hub acts as the gateway between the cloud and the device. All device messages will be received on the hub. This ensures the separation of concerns between the IoT device development and cloud development. The services on the cloud are not aware of how the messages are sent to the hub and are not concerned with the used protocol and communication security. This is all handled by the IoT Hub service. Messages are not automatically stored by Azure's IoT Hub. They are rather events that can be listened to. Azure IoT Hub does provide a built-in feature to route messages to a blob storage account. However, the data is organized according to the order of the message's arrival as presumed by their time-stamp, rather than being classified by other dynamic parameters, such as the source device ID and the shipment ID the device is linked to. Thus, a program is needed to process the messages as they arrive at the hub. A microservice could be created and launched on a virtual machine on the cloud. However, it is more cost-effective to use *Azure Functions*, which follow the Functions-as-a-Service (FaaS) model, in which users are charged for the actual compute performed. This is desirable as the program is only needed when a message arrival trigger happens and does not need to be on all the time in addition to the functionality being relatively lightweight. Thus, dedicating a microservice for it would be inefficient. In addition, using FaaS speeds up the development process, as only the source code needs to be published without having concerns over hosting. Additionally, another function was deployed to send email notifications when a transaction is recorded on the smart contract on Ethereum.

The data in the messages received from devices is in JSON format. To visualize the data on the web application using JavaScript, JSON format is used. Thus, it is more efficient to store the data as it is in JSON rather than converting to SQL tables, for example, and then converting back when retrieving. In addition, SQL databases are more useful for referential data that could be used for indexing and searching. Since sensor readings are more of raw bulk block data that are more likely to be read sequentially, it is better to use Azure's Blob Storage for this scenario. Azure blobs are easier to scale for object storage of text and binary data. In addition, a blob storage account is less costly for storing the large sizes of data expected from containers than an SQL server [30], [31]. A Web App Service was selected to be the means of hosting the web application in this solution. A web app service provides a Platform-as-a-Service (PaaS) model for developing, launching, and hosting web applications. This approach was chosen rather than running a server on a computing resource to save development time and avoid having concerns on the running stack. The web application will be developed in PHP with MySQL, with a front-end website running HTML, CSS, and JavaScript, along with the Bootstrap framework. The flow of data receiving the IoT client message in the IoT hub till the storage in Azure blob storage is presented in Fig.6 for better readability.

2) BLOCKCHAIN INFRASTRUCTURE & SMART CONTRACT

Ready end-user DApps have their smart contracts deployed on the main Ethereum network (Mainnet). Mainnet is the open-sourced functional blockchain where actual

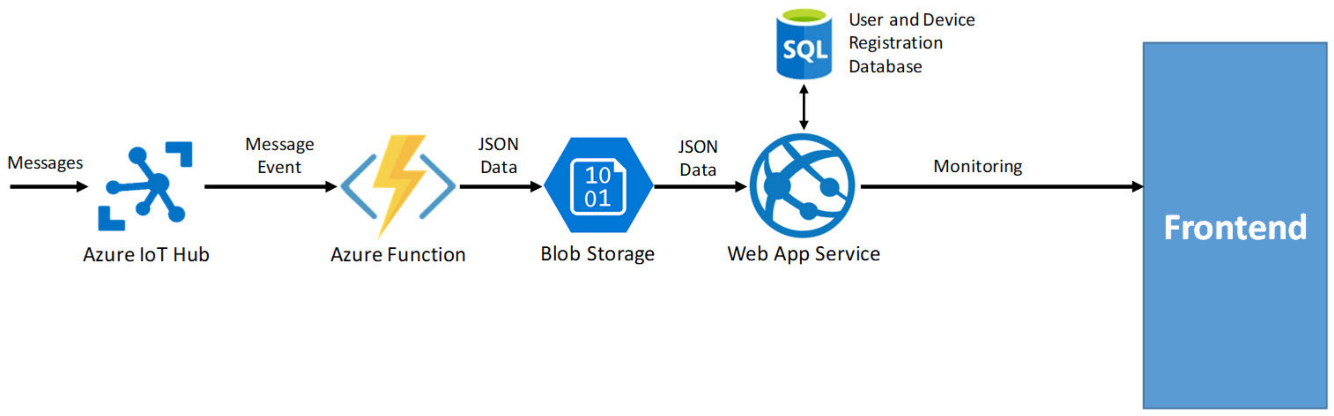


FIGURE 5. Cloud services subsystem architecture hosted on Azure.

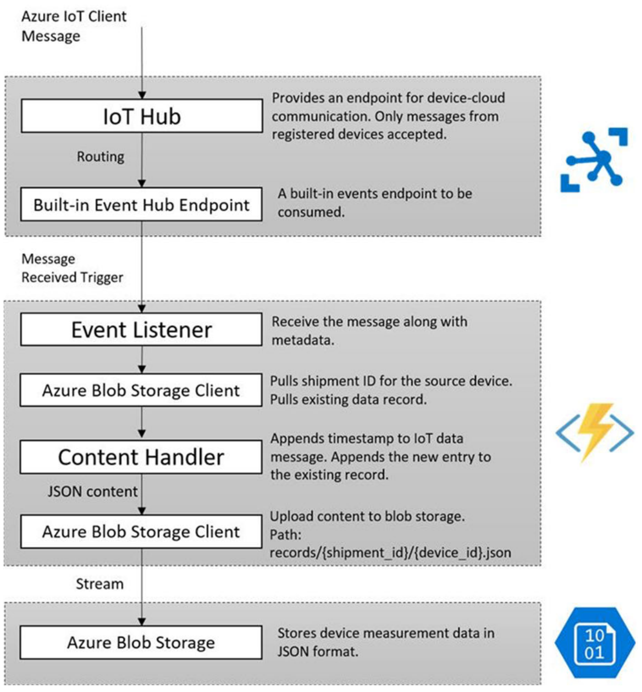

Azure IoT Client Message
IoT Hub
Provides an endpoint for device-cloud communication. Only messages from registered devices accepted.
Routing
Built-in Event Hub Endpoint
A built-in events endpoint to be consumed.
Message Received Trigger
Event Listener
Receive the message along with metadata.
Azure Blob Storage Client
Pulls shipment ID for the source device. Pulls existing data record.
Content Handler
Appends timestamp to IoT data message. Appends the new entry to the existing record.
JSON content
Azure Blob Storage Client
Upload content to blob storage. Path: records/{shipment_id}/{device_id}.json
Stream
Azure Blob Storage
Stores device measurement data in JSON format.
10 01

FIGURE 6. Data flow in the Azure cloud server.

transactions take place in the publicly verifiable distributed ledger. The cryptocurrencies found in the Mainnet hold real monetary value. Miners are incentivized to validate transactions as they are rewarded with native coins, as well as the transaction fees paid by the participants [32].

Test Networks (Testnets) are blockchain environments that exist to offer more convenient development and testing opportunities for smart contracts and DApps while avoiding the disruption of the main Ethereum Network. The cryptocurrencies in Testnets do not possess any real economic value. Testnets use the same technology and software as those found in the Mainnet Ethereum blockchain; hence, simulating a real-world Ethereum network and EVMs environment that would exist on the Mainnet, but without real tokens and Ether. Testnet Ethereum wallets can be filled for free from the available pool of Ethereum faucets. Besides the free transaction costs and lower transaction frequency rates, miners on the Testnets do not receive any economic benefits [32], [33]. From the available Testnets, Ropsten most closely resembles Ethereum's Mainnet as it runs similar protocols. For instance, it uses the same consensus algorithm as the Mainnet, which is Proof of Work. It allows developers to upload and interact with smart contracts without paying the cost of gas, as well as simulate our protocol just as it would work on the Mainnet itself. Once the development is complete, we can then easily deploy and run our smart contract and DApps on Ethereum's mainnet.

An alternative to locally deploying a full Ethereum node is to remotely connect to a hosted Ethereum node. Infura, which is operated by the Ethereum Development Studio, Consensys, hosts full Ethereum nodes and opens up interfaces to allow access to their full nodes. Its microservice-driven architecture offers a world-class decentralized architecture that promotes JSON-RPC over HTTPS and WebSocket [4]. It also supports Web3.0 apps, which can be used as a JS library to integrate with the DApp's front-end. In 2018, Infura's Ethereum full nodes accounted for around 10% of the total 11,803 Ethereum full nodes available worldwide. Infura relies on cloud servers hosted by Amazon, which can indirectly explain how its high reliability is achieved. It also powers several other tools used in DApps such as Metamask and Truffle and offers access to Ethereum Mainnet, as well as Testnets [34].

The use of Infura can shift the focus of developers towards solely focusing on the development of DApps, while Infura can simultaneously ensure high availability, 99.9% uptime guarantee of its nodes, as well as respond to maintenance concerns. Consequently, consumers achieve better energy and power consolidation and hence endure fewer costs. Besides, Infura makes a wide variety of development tools, APIs, and documentation available to the end-user. It also provides users with a dashboard that may help them gain better insights regarding statistics of their DApps and network performance metrics. Choosing to use Infura rather than deploying a full Ethereum Node helps us accelerate the development of our DApps while ensuring higher availability and scalability

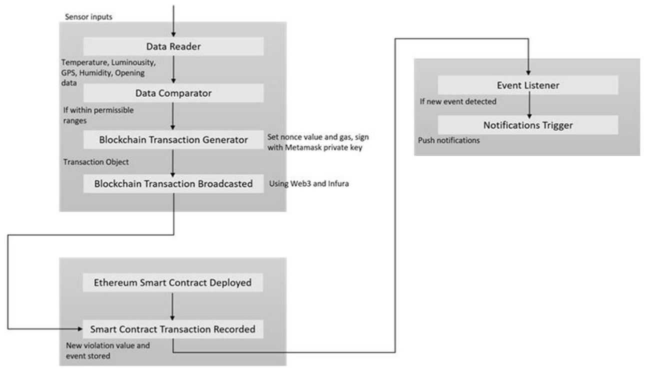

Sensor inputs
Data Reader
Temperature, Luminousity, GPS, Humidity, Opening data
Data Comparator
If within permissible ranges
Blockchain Transaction Generator
Set nonce value and gas, sign with Metamask private key
Transaction Object
Blockchain Transaction Broadcasted
Using Web3 and Infura
Ethereum Smart Contract Deployed
Smart Contract Transaction Recorded
New violation value and event stored
Event Listener
If new event detected
Notifications Trigger
Push notifications

FIGURE 7. **Blockchain logical flow of data from function inputs to triggering events.**

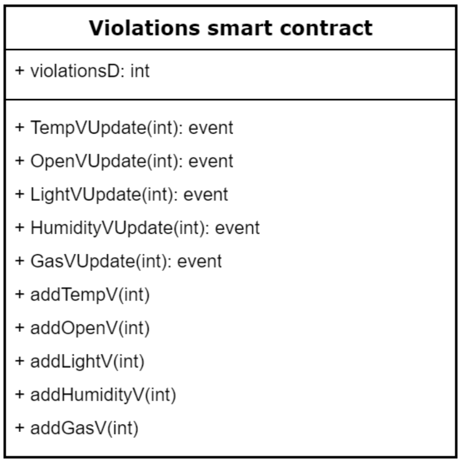

Violations smart contract
+ violationsD: int
+ TempVUpdate(int): event
+ OpenVUpdate(int): event
+ LightVUpdate(int): event
+ HumidityVUpdate(int): event
+ GasVUpdate(int): event
+ addTempV(int)
+ addOpenV(int)
+ addLightV(int)
+ addHumidityV(int)
+ addGasV(int)

FIGURE 8. **UML diagram of the Ethereum smart contract.**

prospects. The privacy challenges that accompany it remain under control, as Infura does not deal with the private keys of the Ethereum wallet, Metamask in our case. Privacy leakage concerns are limited to the wallet address and the IP address.

The recordings of violations in our IoT container are crucial for our system, and they need to be essentially accessed by both DApps and smart contracts. Therefore, we designed our smart contract to store such violations as part of contract storage. Additionally, they are recorded inside event logs. However, the purpose of using events is to provide a means of continuously watching violations and notify users during their occurrence, as discussed in the following sections. Events, a cheaper form of storage, allow the convenient usage of the EVM logging facilities. The passed arguments in an event are treated as logs instead of data and are stored in the transaction's logs in addition to the blockchain. Transaction logs, a special data structure in the blockchain, are associated with the contract's address and are incorporated within the blockchain. Since they are not part of the blockchain itself, they do not directly undergo the applied consensus protocols. However, they are verified by the blockchain since the transaction receipt hashes are stored inside the blocks. Events can be useful in DApps since they allow the DApps to subscribe to them. However, they are not readable by smart contracts [35].

The violations are split into five categories; *temperature, opening/closing of the container, luminosity levels, humidity levels, and the location breach*. Each of the five violations has a separate function to record its occurrence. The function takes as a parameter an integer that would be sent from the microcontroller in an encoded form. Within the function, an event associated with each violation type will be emitted. The design of our smart contract is shown using the UML diagram in Fig. 8.

Using the Infura endpoints, connections to the deployed smart contract can be established given the API and the contract address. Web3, an Ethereum JavaScript API, provides us with a collection of libraries that allow the interaction with the remote node using an HTTP or IPC connection. Web3 also provides means to have DApps subscribe to events since they provide an event filtering functionality that can keep watching for the occurrence of an event. Accordingly, designing the smart contract to emit certain events pertaining to their respective violations allow the development of an event listener app using Web3. In case the event listener detects an event, notifications are triggered to inform the respective stakeholders of the occurrence of a violation. The event listener and trigger can then be converted to Azure functions.

After live measurements about the well-being of the container and its integrity are collected on the microcontroller (Arduino), they are processed to check if they are within the accepted values. If violations are detected, a transaction should immediately be broadcasted from the Arduino to the blockchain, in order to document the occurrence of the relevant violation. Pre-defined raw transactions pertaining to the different types of violations can be prepared and signed. Once the violation occurs, the associated raw transaction can then be sent to the blockchain network to be mined. Similarly, collected data is sent and stored in the cloud. This guarantees that the occurrence of a violation has been immediately sent to the blockchain, without the possibility of having any tampering made to it when the data resides in the edge node. In addition, the nature of our system, which functions during the different stages of cargo transportation, makes it inapplicable to rely on hosting edge nodes. They may be transported over different regions, including remote areas, by land, sea, as well as air, which all together present obstacles that hinder our ability to have enough edge nodes throughout. The flow of data, shown in Fig.7, in blockchain smart contracts starts by calling the Ethereum functions and processing the request and could end up with triggering relevant events. On the DApp level, the violations are pulled from smart contract events, as seen in the flow shown in Fig.9

V. IMPLEMENTATION

This section shows the implementation of the aforementioned smart shipping container. The design choices are presented regarding the master device, slave device, cloud services, and the blockchain platform. The laser-cut shipment container

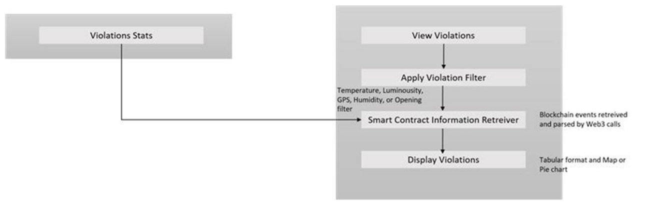

Violations Stats
View Violations
Apply Violation Filter
Temperature, Luminousity, GPS, Humidity, or Opening filter
Smart Contract Information Retreiver
Blockchain events retreived and parsed by Web3 calls
Display Violations
Tabular format and Map or Pie chart

FIGURE 9. Pulling violations from Ethereum events.

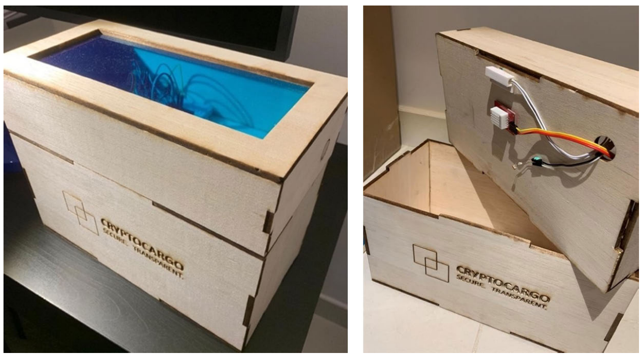


FIGURE 10. CryptoCargo shipment container prototype.

is shown in Fig. 10, which is the transportation box that encompasses all the physical components that are going to be discussed in greater detail. All microcontrollers, sensors, and evidently, the shipment itself are available in the shipment box, that is connected to the cloud servers, as well as the blockchain network. As a matter of fact, these sensors and microcontrollers can be embedded into any shipping container in order to be used for this purpose. The rest of the system components, including the blockchain smart contracts and cloud services can be slightly adjusted for the entity that is implementing this approach.

A. MASTER DEVICE

The Arduino MKR GSM 1400 with an Arduino MKR GPS Shield was utilized to be the master device. In order to connect to the cloud, the device first connects to the MQTT broker using the username and password defined on the cloud. The device is verified by using a self-signed certificate. The connection is made through the Arduino MQTT Library. Fig. 11 shows a part of the code used to set up the connection in order to connect and send messages to the cloud.

To send a transaction to the blockchain, the master device formats and then sends a string to the slave device. The string contains the function name, which determines the violation type, as well as all the sensor readings at the time of the violation. A violation is only sent once every minute to avoid overwhelming the slave device. The same violation type will only be sent once every ten minutes to avoid repeating transactions. A snippet of the code used to send sensor readings to the cloud is shown Fig. 12.

The sensor readings are read frequently to detect any violations instantly, and all the values are stored in global variables so that they can be easily read by the different functions. The stored sensor readings are then compared to the normal values

```
void mqttSetup(){
 if (!ECCX08.begin()) {
    Serial.println("No ECCX08 present!");
    while (1);
  }
  // reconstruct the self signed cert
  ECCX08SelfSignedCert.beginReconstruction(0, 8);
  ECCX08SelfSignedCert.setCommonName(ECCX08.serialNumber());
  ECCX08SelfSignedCert.endReconstruction();
  // Set a callback to get the current time
  // used to validate the servers certificate
  ArduinoBearSSL.onGetTime(getTime);
  // Set the ECCX08 slot to use for the private key
  // and the accompanying public certificate for it
  sslClient.setEccSlot(0, ECCX08SelfSignedCert.bytes(),
                          ECCX08SelfSignedCert.length());
  // Set the client id used for MQTT as the device id
  mqttClient.setId(deviceId);
  /// Set the username to "<broker>/<device id>/
  //  api-version=2018-06-30" and empty password
  String username;
  username += broker;
  username += "/";
  username += deviceId;
  username += "/?api-version=2018-06-30";
  mqttClient.setUsernamePassword(username, "");
  // Set the message callback, this function is
  // called when the MQTTClient receives a message
  mqttClient.onMessage(onMessageReceived);
}
```

FIGURE 11. MQTT setup by connecting to the MQTT broker.

```
void loop() {
    if (gsmAccess.status() != GSM_READY || gprs.status()
            != GPRS_READY) {
      connectGSM();
      gsmLocation.begin();
    }
    if (!mqttClient.connected()) {
      // MQTT client is disconnected, connect
      connectMQTT();
    }
    // poll for new MQTT messages and send keep alives
    mqttClient.poll();
    // publish a message roughly every 25 seconds.
    if (millis() - lastMillis > 25000) {
      updateValues();
      lastMillis = millis();
      publishReadingsMessage();
      detectViolations();
    } else {
      updateValues();
      if(doorState == 1){
        postTransaction(Open);
      }
    }
```

FIGURE 12. Publishing messages comprising sensor readings to the cloud.

to determine if a violation is present. In case a violation is found, the transaction is sent to the blockchain. The sensors are connected as shown in Fig. 13. Additionally, the GPS Shield is connected through the I2C bus and does not have any pin connections.

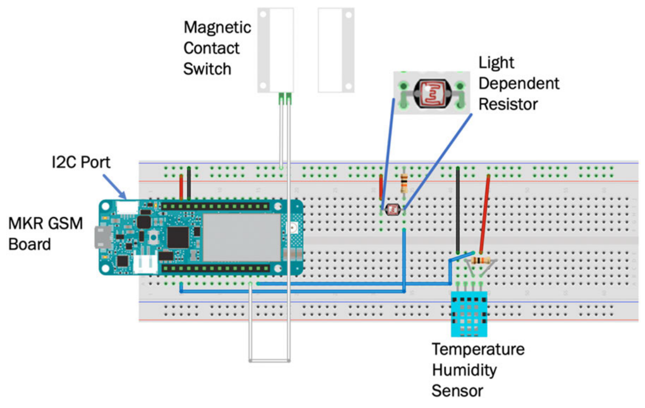


FIGURE 13. Connection scheme for the sensors with the master device.

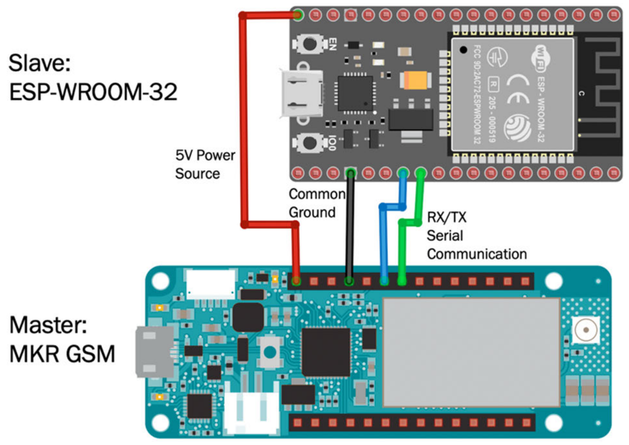


FIGURE 14. Physical connection between master and slave devices.

B. SLAVE DEVICE

Our choice of slave device was based on library compatibility, in addition to its size and capabilities. The master and slave devices are connected together as shown in Fig. 14. The devices share a common ground, as well as the 5V power source. Additionally, in order to have a serial connection we have two wires, RX/TX to transfer the required information. On the slave device, the device continuously listens for messages. It checks the string to make sure it is suitable and passes it to the function. The function prepares all the transaction parameters, signs the transaction, and sends it to the smart contract. All this is done using the modified Web3 of the Arduino library, as shown in Fig. 15.

In addition to improving the Arduino library to fix major compilation and run-time errors, we changed the way the function works because the data parameters were previously formatted by functions in the library. This presented numerous difficulties and logical errors. Finally, the gas price for each transaction was stored and passed from an integer variable. This limited the gas price to the maximum value that can be stored in an integer variable. To fix this, we changed the library to read the value from a string. To summarize, the flow of data in both master and slave IoT devices is presented in Fig. 16. This flow encompasses the flow of data as from

```
void eth_send_example(string *p) {
     //Initialize contract
    Contract contract(&Web3, (string*)CONTRACT_ADDRESS);
    //Initialize private key
    contract.SetPrivateKey((uint8_t*)PRIVATE_KEY);
    //Get nonce value for transaction
    uint32_t nonceVal = (uint32_t)Web3.EthGetTransactionCount
                            ((string *)MY_ADDRESS);
    USE_SERIAL.println(nonceVal);
    //Setting gas price for transaction
    string gasPriceVal = "0x1ffffffff";
    uint32_t  gasLimitVal = 8000000;
    string toStr = CONTRACT_ADDRESS;
    string valueStr = "0x00";
    uint8_t dataStr[100];
    memset(dataStr, 0, 100);
     //Send transaction with parameters
    string result = contract.SendTransaction
                (nonceVal, &gasPriceVal, gasLimitVal,
                        &toStr, &valueStr, p);
    string result = contract.SendTransaction
                (nonceVal, &gasPriceVal, gasLimitVal,
                        &toStr, &valueStr, p);
}
```

FIGURE 15. Formulating the transaction parameters and sending it to the smart contract.

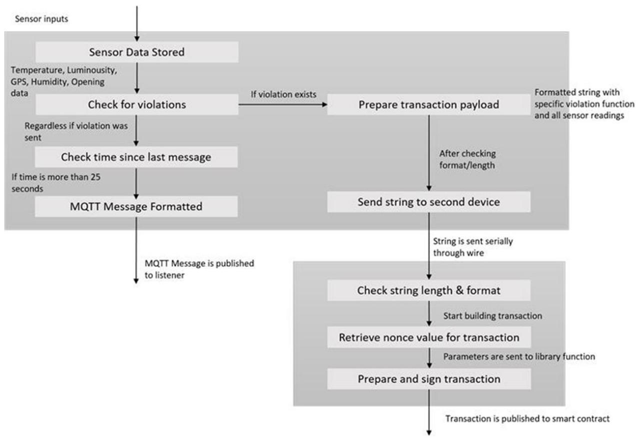

Sensor inputs
Sensor Data Stored
Temperature, Luminousity, GPS, Humidity, Opening data
Check for violations
If violation exists
Prepare transaction payload
Formatted string with specific violation function and all sensor readings
Regardless if violation was sent
Check time since last message
After checking format/length
If time is more than 25 seconds
MQTT Message Formatted
Send string to second device
String is sent serially through wire
MQTT Message is published to listener
Check string length & format
Start building transaction
Retrieve nonce value for transaction
Parameters are sent to library function
Prepare and sign transaction
Transaction is published to smart contract

FIGURE 16. Flow of data in master and slave IoT devices.

sensor readings to preparation of transactions to be published to the Ethereum smart contract.

C. CLOUD SERVICE

A resource group was created on Azure to hold all the Cloud services and resources for this project. The web application, IoT Hub, Azure Storage Account, and server-less functions were deployed and configured successfully. The IoT Hub was given the namespace *SmartShippingContainerHub*, and the storage to hold the device data was called *sscontainerrecords*. The *CryptoCargo* app service is the platform to host, configure, and manage the web application. When creating this service, PHP was chosen as the code stack to be used. The app service comes with an ''App Service Plan,'' which basically acts as an abstraction of the infrastructure the web app would run on. The app service plan can be used to scale the web app resources and monitor their health. An ''Application Insights'' service is also provided with app service.

This can be used to monitor the application through analytics dashboards. The *sscmessageroute* Azure function similarly has an App Service Plan and Application Insights. The MySQL database cannot be seen as a service as a MySQL server was not created. However, MySQL was deployed within the web app service with Azure's new feature that allows this for PHP applications. When deployed, the MySQL database would share the same hardware resources as the web application, and the required credentials would be found in an environment variable that can be accessed with web app code. This configuration model is great for developmental and testing purposes; however, an independent MySQL server should be deployed for a high-scale production environment.

The Azure IoT Hub was created through the Azure Market Place on the Azure Portal. A free pricing plan was selected for the purpose of this implementation. The hub was given the name *SmartShippingContainer Hub* and was hosted on the closest region (EU-West) for the best performance. After the service was successfully created and deployed, it was managed through a dashboard or by using Azure's CLI. For testing purposes, a simulated device is run to send telemetry to the hub. The device simulation program is written in the C language and uses the Azure IoT Hub C SDK libraries, which would be used for Arduino devices. However, the simulated device is authenticated with the symmetric key method, where the device and hub would share the same key generated by Azure. The simulated device was registered with *MyCDevice* as its ID. An Azure Function was developed and deployed to process incoming messages to the Azure IoT Hub.

The messages arrive from a source with a device ID and have a time-stamp showing the arrival time. The Azure function is triggered by the arrival of messages. The function simply takes the device ID and finds the associated shipment the device is currently on. The message data is then stored on an Azure storage account. Thus, that file will contain all sensor measurement data sent to the cloud by that particular device when it was on that particular shipment. This way, when retrieving the data by the web app using only the shipment ID and device ID, all the data for that shipment and device will be retrieved. In addition, the function appends the time-stamp of the message arrival to the message content itself. This is to keep a record of the time the measurements were taken. This is because the originally recorded message time-stamp refers to the arrival time to the cloud, which is slightly different (in milliseconds) compared to the time it is recorded on the device. To be able to develop this function, the Azure Storage Client SDK package was needed to be added to the source code of the function [36].

The shipment ID can be found in a blob that will be stored on the storage account beforehand called the *IDreference*. The file, or blob, will basically hold all the device IDs along with the corresponding shipment ID that is linked to it as a device can only be on a single shipment at a time. This file is constantly updated when a device is launched on a shipment and when a shipment is completed. After finding the shipment ID from the *IDreference* file and appending the time-stamp to the message data, a BlobClient is created with the path format specified previously. If no blob exists with that path and name, then one is created with just the message data that was received. The data is stored as a JSON array of data objects. If a blob does exist, then the content is downloaded, the new data is appended to the existing array, and the new content is uploaded to overwrite the existing file. For testing purposes, a simulated device program is run to send random temperature and humidity data to the hub. The simulated device is given the ID *MyCDevice* and is linked to the shipment ID *sim1*. As messages arrive at the IoT Hub, the function gets triggered and executes. Using the Azure portal's storage explorer for Storage Accounts, the saved messages can be found in their appropriate blob. The Azure Function was published to Azure using the publishing profile available in the Azure Functions Service. The function is triggered whenever a message is received by the IoT Hub.

D. BLOCKCHAIN & SMART CONTRACTS

To gain access to a full Ethereum node, an account was set up with Infura. Infura presents an interface to access such Ethereum nodes on both Ethereum mainnet and testnets. Based on the API keys, endpoints to the different Ethereum networks are provided through HTTP. Infura does not hold any private keys and hence does not generate Ethereum wallets. Metamask was used to generate our Ethereum wallet. Once a Metamask account was set up, we set it to use the Ropsten Testnet. The wallet was filled with funds (Ethers) using free online (Ether Faucets), such as from https://faucet.metamask.io/. As with every Ethereum wallet, Metamask provides our account with a unique account address and a private key. Details about our account and its transactions can be found on Etherscan, the leading BlockExplorer for the Ethereum blockchain. A BlockExplorer is a search engine that allows users to look up, confirm, and validate transactions being carried out on the Ethereum blockchain. The smart contract is written in Solidity language. Solidity is an object-oriented, high-level language, designed to target the EVM and for implementing smart contracts. The smart contract was then deployed on a test blockchain network using Remix. Remix is a web browser-based IDE that allows writing to Solidity smart contracts, deploying, and running them. Having Metamask installed and logged in on our browser allows us to make use of its injected Web3 environment to deploy the smart contract. Another possible alternative is to use an endpoint to a Web3 provider, through HTTP or IPC. For testing and development purposes, Remix also promotes a sandbox blockchain in the browser to execute transactions on, called JavaScript VM environment.

Once the smart contract is deployed, it can be found on Etherscan using its uniquely assigned contract address. Since it was deployed using our Metamask Ethereum wallet account, our wallet account address can be seen in the

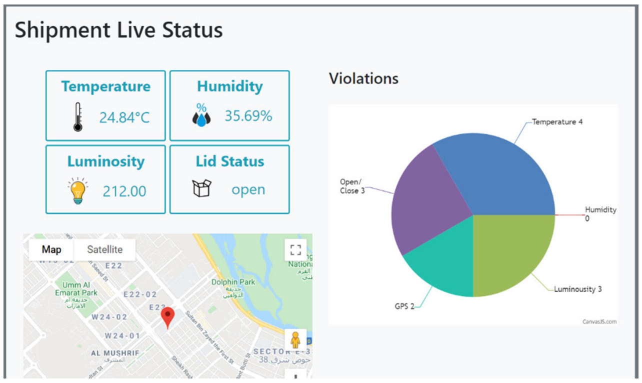


FIGURE 17. Live status page on the web app displaying sensor readings in real-time.

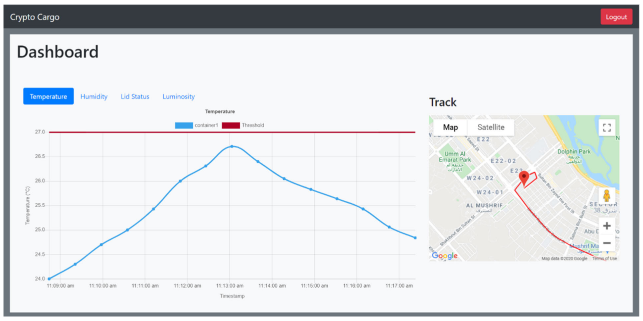


FIGURE 18. tracking the shipment journey while monitoring all vitals and location.

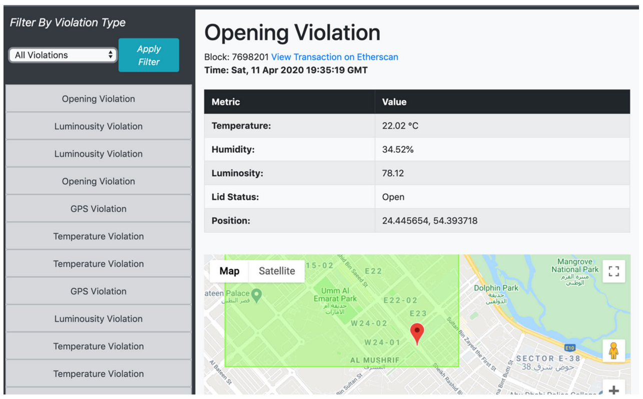
contract creator field. To call functions and retrieve data from the smart contract in our DApps, Web3 is needed. A Web3 library named 'Web3.min.js' is downloaded and is imported into our code. In addition, inside our DApps code, the Web3 provider is defined through an HTTP link to our Infura endpoint. Using Metamask's injected Web3 is also possible. To create a Web3 contract instance, two arguments need to be passed to the Web3.eth.Contract: the contract's unique address and the contract's ABI. The smart contract is then converted from the high-level Solidity language to bytecode to run on the Ethereum Virtual Machine (EVM) when compiled. ABI comprises a list of the contract's functions, variables, constants, and arguments in JSON format. It is intended for encoding and decoding data into and out of transactions. ABI also indicates the functions' signatures and variables' declarations that are needed to achieve understandable calls to the contract's bytecode in the EVM.

E. DECENTRALIZED APPLICATION

A PHP web application was fully implemented according to our design to provide users with monitoring and tracking capabilities. The user experience can be seen in the figure below, showcasing the user perspective flow of the application. The application has a home page, live status page, monitoring, and tracking page, and a violations view page. The bootstrap framework was used to help building the GUI of the application, which was built from scratch. The application starts with the home page after the user has been authenticated. This page simply queries the user shipments and displays them along with their details. This information is pulled from the MySQL database. The user can select a shipment from here to be redirected to its live status page.

The live status page, shown Fig. 17, displays to the user the latest readings received on the cloud from the IoT device. The temperature, humidity, luminosity, and the lid status of the container are displayed. A Google map *iframe* is also presented with the latest location of the container pinned. The map is generated with Google maps API for JavaScript. This page also has a pie chart that breaks down the recorded violations on the blockchain by type. The Web3 JavaScript library is used to pull the number of transactions for each type of violation. When loading the page, the latest data entry is pulled from the Azure Blob Storage using the Azure Blob Storage SDK for PHP, which helps us to manipulate a storage account service along with its containers and blobs. The connection string to the storage account is used to create the storage client on PHP. To retrieve the data, a blob client is created with the connection string, and then the client is used to retrieve a blob by passing its container and path. The content stream of the blob is then retrieved. These values are passed to the front-end to display. To have the page dynamically update the data, a recursive function was written in the JavaScript code to call a built Rest API for the latest readings. The function then updates the displayed data and the Google Map location. Rather than having the JavaScript connect directly to the storage, the PHP API was built to keep the connection to the storage secure by obscuring it from the user by applying it on the server side. The API pulls the latest data, similarly to what was explained before, and sends it as a reply. The shipment and device IDs are expected as input. From the live status page, the user can navigate to the monitoring and tracking, or the violations pages using buttons at the bottom of the page.

To monitor and track the shipment journey, the entire shipment data is pulled from the Azure Blob Storage, using a similar approach to the one explained earlier. On this page, however, the entire record is taken rather than just the latest readings, as shown in Fig. 18. The data is split into arrays that hold each metric independently (temperature array, humidity array, etc.). These arrays are then passed to the front-end for visualization. An open-source JavaScript library called chart.js was used to graph this data. The data arrays are passed to the Chart.js function to create charts. The monitoring page provides the generated charts by having them in Bootstrap tabs that can be navigated through. The page also displays a map with the route taken by the shipment plotted. The route was plotted using polylines defined in the Google Maps API. In the front-end DApp, users are given the feature of viewing violations' information directly retrieved from the blockchain. As shown in Fig. 19, on the left side of the page, a scrollable pane exists that displays headings of all violations currently recorded. Filters can also be applied to restrict the search to a certain type of violation type only. Once a heading is clicked on, all violation information is retrieved from the

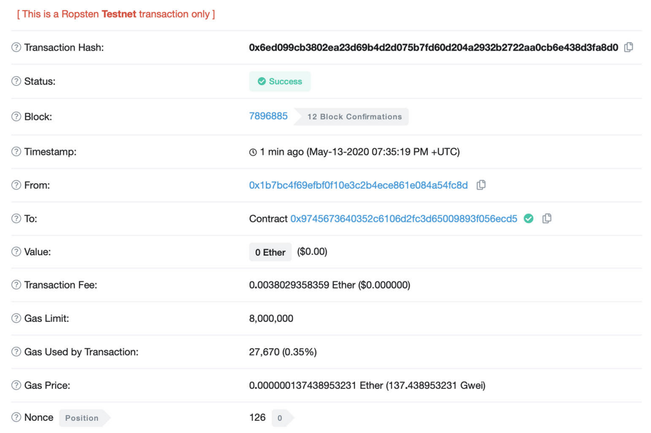


Opening Violation

Block: 7698201 View Transaction on Etherscan
Time: Sat, 11 Apr 2020 19:35:19 GMT

Metric	Value
Temperature:	22.02 °C
Humidity:	34.52%
Luminosity:	78.12
Lid Status:	Open
Position:	24.445654, 54.393718

FIGURE 19. Frontend page retrieving blockchain-stored data.


[This is a Ropsten **Testnet** transaction only]

Transaction Hash:	0x6ed099cb3802ea23d69b4d2d075b7fd60d204a2932b2722aa0cb6e438d3fa8d0
Status:	Success
Block:	7896885 12 Block Confirmations
Timestamp:	1 min ago (May-13-2020 07:35:19 PM +UTC)
From:	0x1b7bc4f69efbf0f10e3c2b4ece861e084a54fc8d
To:	Contract 0x9745673640352c6106d2fc3d65009893f056ecd5
Value:	0 Ether ($0.00)
Transaction Fee:	0.0038029358359 Ether ($0.000000)
Gas Limit:	8,000,000
Gas Used by Transaction:	27,670 (0.35%)
Gas Price:	0.000000137438953231 Ether (137.438953231 Gwei)
Nonce Position	126 0

FIGURE 20. Transaction details displayed on the testnet Ropsten blockchain explorer.

smart contract and is displayed to the user. Again, methods of Web3.js are used to interact with the smart contract, and Google Maps APIs from a Google Cloud Platform account are utilized to employ the maps. Bootstrap is used to improve the CSS and JavaScript-based user interface design of the page.

Infura and Web3 provide two types of Ethereum APIs: "Call," which does not cost any gas value, and "Transaction," which requires gas to be sent. "Call" can be used to access the functions defined in the smart contract and hence ABI. Since reading data from the blockchain requires no transactions to be carried out, there is no need for mining nor for paying gas. Thus, the Ethereum account address and its private key are not included in the front-end code, as opposed to including it in the Arduino where the transactions are sent and generated. Consequently, such function calls to read state can be made directly from the Ethereum node we are connected to, which is the Infura node in our case. When the page loads, the *"allEvents"* filter is applied by default to display all the available recorded violations. A different filter pertaining to a different violation type can be applied. Based on the chosen filter, the event name of the violation type is passed as an input to the displayViolations function.

Inside the *displayViolations* function, the Web3 method *getPastEvents* followed by the event name retrieves an array of event objects, with all their details. To search for all events, the *"allEvents"* keyword can be used. Variables and methods such as block number, event name, and the return value of the event object can then also be individually called. Row handlers are then initiated from within the *displayViolations* function and the associated event elements like the block number and transaction hash are called and displayed when the violation is clicked on.

Based on the way our violation details were generated and sent from the Arduino, the violation details are stored on the blockchain in an encoded decimal form stored in a pre-defined format. As a result, the *returnValues* returned by the event object are passed as inputs to our *getBlockchainInfo* function to be decoded. The *getBlockchainInfo* function decodes the blockchain-stored violation by first converting the long decimal to hexadecimal. The message is long enough that the built-in JavaScript decimal to hex converters fail to convert them correctly. As a result, an open-source function decToHex is used instead. Once the data is converted to a 33 digits Hex output, they are parsed and split into separate variables representing the GPS longitude and latitude positions, temperature, humidity, luminosity, and container open/close. All results are then displayed on the output, whereas the GPS longitude and latitude coordinates are passed to Google Map's *initMap* function. In the *initMap* function, the GPS longitude, and latitude coordinates are used to display a marker at the position of the shipment, with other possible options like a marker title. The coordinates of the allowable geographic boundaries that do not cause a GPS violation are also set and are highlighted on the map with a green color. The map is then used with all features provided by Google's APIs. The pie chart shown in Fig. 17 visualizing the violations transactions recorded on the blockchain, uses Canvas.js library for the chart along with Web3.js to interact with the smart contract. The *getPastEvents* function of Web3.js is used to retrieve an array of all the events stored on the blockchain for each of the five different types of violations. Because of its asynchronous callback nature, all five functions are called successively, with every call being encapsulated in the previous one. The chart is then populated with the respective violation data by accessing the length of violations stored in each array of particular events, and their respective labels are given following which chart is rendered and displayed.

The entire process described above is presented visually for a better grasp of the process flow. Figs. 21 and 22 present the entire flow of data between the different DApp files, including the usage of different APIs throughout. These compromise all the features offered by the DApp including monitoring, shipment tracking, and violation reporting.

VI. TESTING AND VALIDATION

To test the master and slave devices, we tested the Arduino in a local environment where the output could be seen on a computer. Additionally, to verify the functionality, we tested

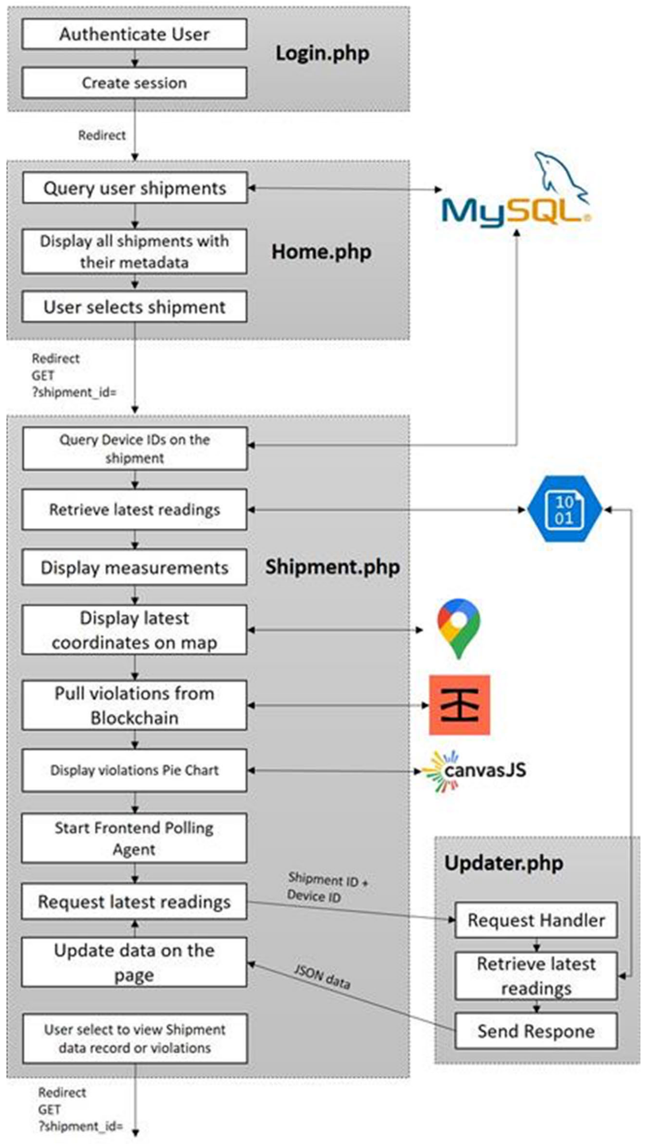
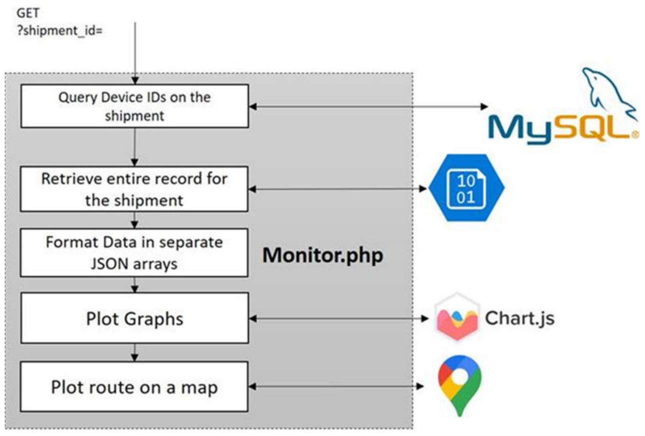


FIGURE 21. Transaction details displayed on the testnet Ropsten blockchain explorer.


GET ?shipment_id=
Query Device IDs on the shipment
Retrieve entire record for the shipment
Format Data in separate JSON arrays
Plot Graphs
Plot route on a map
Monitor.php
MySQL
Chart.js

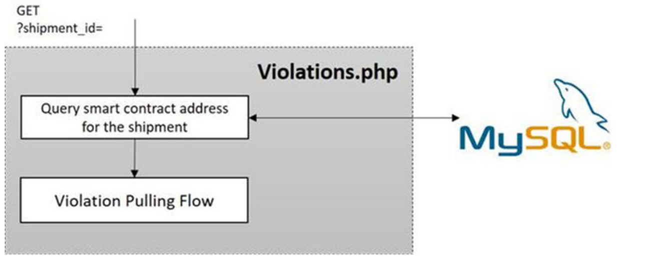


FIGURE 22. Transaction details displayed on the testnet Ropsten blockchain explorer.

to see if a transaction was successfully posted, as shown in Fig. 20. In this figure, we can see that the board publishes a message roughly every 25 seconds, meaning that it is meeting the time requirements. Also, we notice that when a violation occurred, it sent out the formatted string correctly. To verify the transaction, we looked at the smart contract on Etherscan, as shown in Fig. 20. By looking at this page, we notice that the transaction was posted a few seconds after the device sent it. Furthermore, we also noted that the input data field was identical to the string sent from the device. This tells us that the system is functioning as expected.

Various tests were conducted to verify and evaluate the functionality of the system. The solution objectives were validated as messages successfully reached the user application from the IoT device, by going through the IoT Hub, the Azure Function, and the Azure Storage. The performance of the Cloud back-end can be evaluated using Azure's built-in analytics for the services deployed for this project as they follow the PaaS and FaaS models. The Azure function analytics are shown in Fig. 23 for the final testing period of the system. It can be seen that 359 IoT messages were processed with 0 failures. It can also be seen that the average server response time is 1.37 seconds. This takes into account the firing delay of the function (as it is server-less), the processing time, and the delay of uploading the data to the storage. Thus, an average delay of 1.37 seconds is due to storing the IoT Hub messages on the Blob Storage.

The performance of the Azure Storage Account can be seen in Fig. 24 for the duration of the testing period. It can be seen that the storage had an availability score of 100%, so there was no downtime faced. The average latency can be seen to be 43.36ms. Over the 4,206 recorded transactions, only 16 authorization errors were encountered. These could be due to accessing attempts from outside the project scope, as the functionality of the project was fully validated. Further investigation is needed to specifically identify the causes of these errors. Regardless, the success rate of transactions was over 99.6%.

The server response time for the web app can be seen in Fig. 25. No failures were encountered for the testing duration, and the server averaged a response time of 13.65ms. The tier used does not provide the ‘‘on all the time'' feature. Instead, the server is started up when URL access is attempted. This causes a small delay when first loading

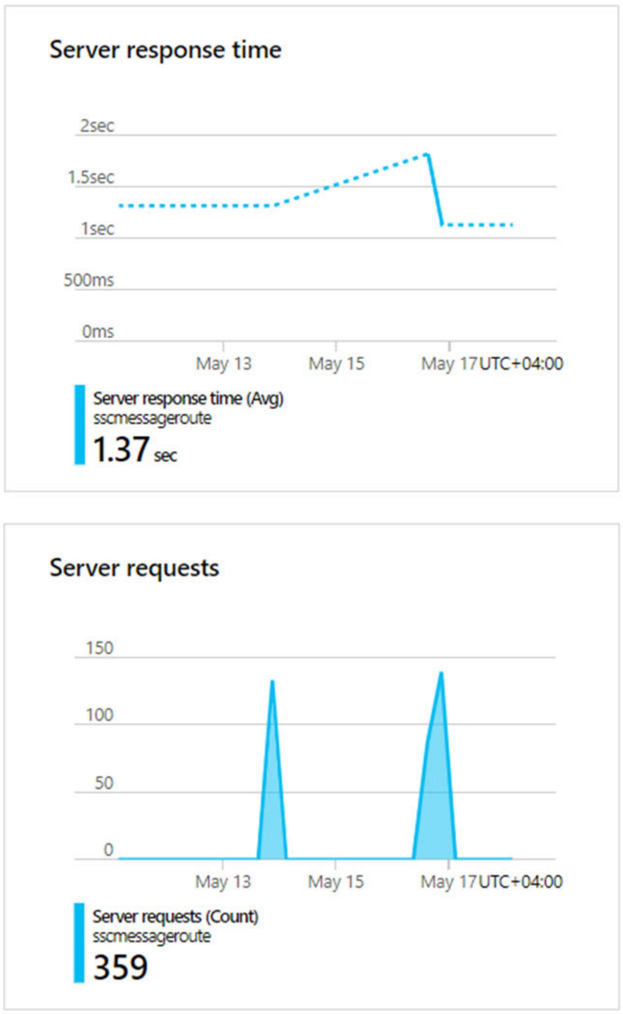


FIGURE 23. Azure Function Analytics for the testing period.

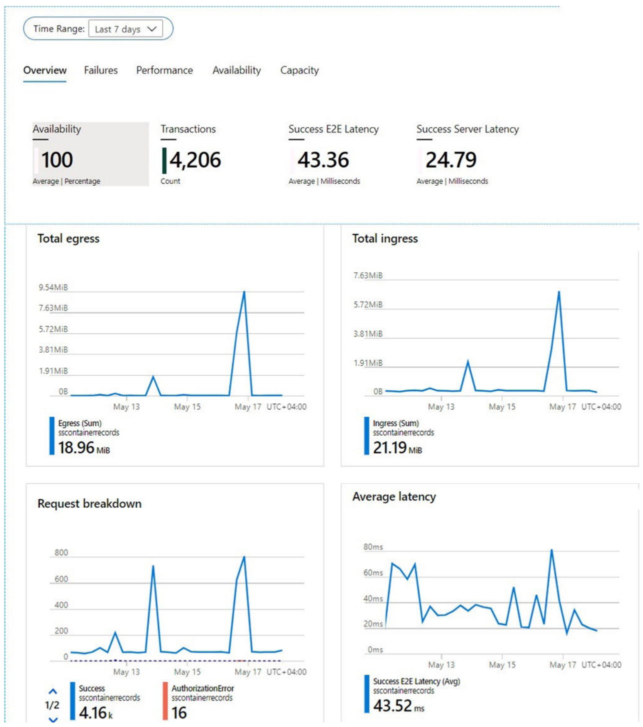


FIGURE 24. Azure storage analytics and insights.

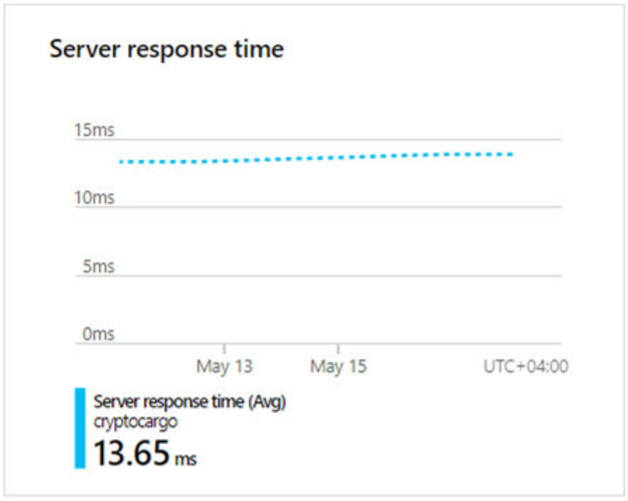


FIGURE 25. Server response time for the duration of testing derived from the web app analytics.

up the website. However, browsing is seamless afterward. By combining all the service delays, we expect an average response time for the entire cloud subsystem to be around 1.43 seconds. Such delays are considered as the IoT device sends messages every 25 seconds rather than 30 seconds. Thus, with these delays, the user should still receive the live updates within 30 seconds, which was the soft real-time requirement of the project.

VII. CHALLENGES AND FUTURE WORK

The blockchain and its underlying infrastructure are still considered emerging technologies. In addition, many of the available tools, such as the Web3 Arduino libraries, are not yet mature, or do not receive consistent patches or improvements. These issues hinder the adoption of such applications in the industry at the current time as most companies are not ready yet to transfer their work flows and substitute them with other technologies, such as the blockchain, regardless of the added benefit gained. As future improvements to our product, our solution can have its services adapt more effectively to a larger scale of shipments by having the smart contract and their linked shipment pages generate automatically, rather than manually. Improvements can also be done on the frontend DApps to enhance the user experience of the website. For instance, additional features can be added to allow users to zoom in into the available charts and receive more detailed visualizations of the time period they are interested in. The responsiveness of the DApp can be further enhanced to allow better handling of different types of devices, including phones and tablets. Native apps can also be configured as an alternative to the browser-based apps. On the hardware level, a customized single chip can be used instead of the 2 boards we are currently using as a proof of work. Additional local storage capabilities to support cases of overseas shipments that pass through phases of no cellular coverage can also be utilized.

VIII. CONCLUSION

In this paper, we introduced a fully functional, blockchain-powered solution (CryptoCargo) that comprises a smart container system to track shipments and identifies any threats to the health of the package. CryptoCargo utilises blockchain's cryptographic foundations and smart contracts to achieve a trustworthy log of violations reported by the container. Additionally, CryptoCargo uses cloud-based services such as Azure IoT Hub to achieve real-time communication with the smart container facilitating accurate reporting and analytics. In order to achieve a usable solution, CrypoCargo implements a secure web-based dashboard application for end-users to query shipment information, visualise shipment location, and perform analytics. CryptoCargo has been evaluated to validate its function as well as performance efficiency which demonstrated its effectiveness in achieving a secure end-to-end shipment tracking in a trustworthy manner.

REFERENCES

[1] J. Hoffmann, R. Asariotis, M. Assaf, and H. Benamara. (2018). *Unctad Review of Maritime Transport 2018*. Accessed: Sep. 13, 2020. [Online]. Available: https://unctad.org/en/PublicationsLibrary/rmt2018_en.pdf

[2] L. Doe. (2017). *Top Four Dangerous Shipping Statistics*. Accessed: Sep. 13, 2020. [Online]. Available: https://www.porttechnology.org/news/top_four_dangerous_shipping_statistics/

[3] H. Manaadiar. (2019). *Cargo Theft Statistics and Trends*. Accessed: Oct. 6, 2019. [Online]. Available: https://shippingandfreightresource.com/cargo-theft-statistics-and-trends/

[4] T. Groenfeldt. (2017). *IBM and MAERSK Apply Blockchain to Container Shipping*. Accessed: Oct. 6, 2019. [Online]. Available: https://www.forbes.com/sites/tomgroenfeldt/2017/03/05/ibm-and-maersk-apply-Blockchain-to-container-shipping

[5] (2019). *8 Ways Blockchain is Revolutionizing Transportation and Logistics*. Accessed: Oct. 6, 2019. [Online]. Available: https://www.winnesota.com/blockchain

[6] P. Mell and T. Grance. (2011). *The NIST Definition of Cloud Computing*. Accessed: Jan. 20, 2020. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-145/final

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019. [Online]. Available: https://git.dhimmel.com/bitcoin-whitepaper/

[8] S. Underwood, "Blockchain beyond bitcoin," *Commun. ACM*, vol. 59, no. 11, p. 15–17, Oct. 2016, doi: 10.1145/2994581.

[9] I. Bentov, A. Gabizon, and A. Mizrahi, "Cryptocurrencies without proof of work," in *Proc. Int. Conf. Financial Cryptography Data Secur. (ICCSA)*, in Lecture Notes in Computer Science. St. Petersburg, Russia: Springer, 2019.

[10] C. Maldonado. (2018). *Introduction to Blockchain and Ethereum*. Accessed: Jan. 22, 2020. [Online]. Available: https://www.packtpub.com/product/introduction-to-blockchain-and-ethereum/9781788835251

[11] R. Tas and O. O. Tanriover, "Building a decentralized application on the Ethereum blockchain," in *Proc. 3rd Int. Symp. Multidisciplinary Stud. Innov. Technol. (ISMSIT)*, Oct. 2019, pp. 1–4.

[12] *Ethereum Api | Ipfs Api Gateway | Eth Nodes As A Service | Infura*. Accessed: Jan. 20, 2020. [Online]. Available: https://Infura.io

[13] G. McCubbin. (2020). *Intro to web3.js Ethereum Blockchain Developer Crash Course*. Accessed: Jan. 21, 2020. [Online]. Available: https://www.dappuniversity.com/articles/web3-js-intro

[14] (2016). *Web3.eth.subscribe*. Accessed: Jan. 21, 2020. [Online]. Available: https://Web3js.readthedocs.io/en/v1.2.0/Web3-eth-subscribe.html

[15] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, "Blockchain technology and its relationships to sustainable supply chain management," *Int. J. Prod. Res.*, vol. 57, no. 7, pp. 2117–2135, Apr. 2019.

[16] K. Francisco and D. Swanson, "The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency," *Logistics*, vol. 2, no. 1, p. 2, Jan. 2018.

[17] A. Rejeb, J. G. Keogh, and H. Treiblmaier, "Leveraging the Internet of Things and blockchain technology in supply chain management," *Future Internet*, vol. 11, no. 7, p. 161, Jul. 2019.

[18] C.-S. Yang, "Maritime shipping digitalization: Blockchain-based technology applications, future improvements, and intention to use," *Transp. Res. E, Logistics Transp. Rev.*, vol. 131, pp. 108–117, Nov. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1366554519307045

[19] R. D. Gregorio, S. S. Nustad, and I. Constantiou, "Blockchain adoption in the shipping industry," in *A Study of Adoption Likelihood and Scenario-Based Opportunities and Risks for IT Service Providers*, vol. 272. Frederiksberg, Denmark: Copenhagen Business School, 2017.

[20] M. Jović, M. Filipović, E. Tijan, and M. Jardas, "A review of blockchain technology implementation in shipping industry," *Pomorstvo*, vol. 33, no. 2, pp. 140–148, Dec. 2019.

[21] L. Li and H. Zhou, "A survey of blockchain with applications in maritime and shipping industry," *Inf. Syst. E-Bus. Manage.*, pp. 1–19, Sep. 2020, doi: 10.1007/s10257-020-00480-6.

[22] S. Tsiulin, K. H. Reinau, O.-P. Hilmola, N. Goryaev, and A. Karam, "Blockchain-based applications in shipping and port management: A literature review towards defining key conceptual frameworks," *Rev. Int. Bus. Strategy*, vol. 30, no. 2, pp. 201–224, Jun. 2020.

[23] G. Bavassano, C. Ferrari, and A. Tei, "Blockchain: How shipping industry is dealing with the ultimate technological leap," *Res. Transp. Bus. Manage.*, vol. 34, Mar. 2020, Art. no. 100428. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210539519301646

[24] O. Iakushkin, D. Selivanov, E. Pavlova, and V. Korkhov, "Architecture of a smart container using blockchain technology," in *Proc. Int. Conf. Comput. Sci. Appl. (ICCSA)*, in Lecture Notes in Computer Science. St. Petersburg, Russia: Springer, 2019.

[25] C. Li. (2019). *Maersk–Reinventing the Shipping Industry Using IoT and Blockchain*. Harvard Business School Digital Initiative. Accessed: Oct. 6, 2019. [Online]. Available: https://digital.hbs.edu/industry-4-0/maersk-reinventing-shipping-industry-using-iot-Blockchain/

[26] (2018). *Maersk and IBM Introduce Tradelens Blockchain Shipping Solution*. Accessed: Oct. 6, 2019. [Online]. Available: https://newsroom.ibm.com/2018-08-09-Maersk-and-IBM-Introduce-TradeLens-Blockchain-Shipping-Solution

[27] (2018). *This Swiss Based IoT Company Did a Double ICO and Raised More Than 15 CHF MIO*. [Online]. Available: https://fintechnews.ch/blockchain_bitcoin/smart-containers-ico-15-mio/21779/

[28] R. Shahan. (2018). *Azure IoT Hub Communication Protocols and Ports | Microsoft Docs*. Accessed: Jan. 21, 2020. [Online]. Available: https://docs.microsoft.com/en-us/Azure/iot-hub/iot-hub-devguide-protocols

[29] (2020). *Azure IoT Hub Pricing*. Accessed: Jan. 22, 2020. [Online]. Available: https://Azure.microsoft.com/en-us/pricing/details/iot-hub

[30] (2020). *Azure Storage Blobs Pricing | Microsoft Azure*. Accessed: Jan. 22, 2020. [Online]. Available: https://Azure.microsoft.com/en-us/pricing/details/storage/blobs

[31] (2020). *Azure SQL Database Pricing*. Accessed: Jan. 22, 2020. [Online]. Available: https://Azure.microsoft.com/en-us/pricing/details/sql-database/managed

[32] Aziz. (2020). *Crypto Mainnet vs Testnet: What is the Difference?* Accessed: Jan. 22, 2020. [Online]. Available: https://masterthecrypto.com/mainnet-vs-testnet-whats-the-difference

[33] G. Hayes. (2018). *The Beginners Guide to Using an Ethereum Test Network*. Accessed: Jan. 22, 2020. [Online]. Available: https://medium.com/compound-finance/the-beginners-guide-to-using-an-Ethereum-test-network-95bbbc85fc1d

[34] R. O'Leary. (2018). *The Race is on to Replace Ethereum's Most Centralized Layer*. Accessed: Jan. 22, 2020. [Online]. Available: https://www.coindesk.com/the-race-is-on-to-replace-Ethereums-most-centralized-layer

[35] (2018). *Solidity 0.4.24 Documentation*. Accessed: Jan. 22, 2020. [Online]. Available: https://solidity.readthedocs.io/en/v0.4.24/index.html

[36] M. Hopkins, T. Myers, G. Wallace, C. Casey, B. Harvey, D. Patil, R. Shahan, and N. Estabrook. (2019). *Quickstart: Azure Blob Storage Client Library v12 for.Net*. Accessed: Jan. 23, 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-dotnet

OMAR ALKHOORI received the bachelor's degree (Hons.) in computer engineering from the Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates, in 2020. He is currently working with Digital14, United Arab Emirates, as an Associate Professor. His research interests include blockchain technology, the Internet of Things (IoT), and cloud computing.

ABDURAOUF HASSAN received the bachelor's degree (Hons.) in computer engineering from the Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates, in 2020. He is currently working with GamaLearn, United Arab Emirates, as a Web Applications Developer. His research interests include blockchain technology, the Internet of Things (IoT), and cloud computing.

OMAR ALMANSOORI received the bachelor's degree (Hons.) in computer engineering from the Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates, in 2020. He is currently working with Beacon Red, United Arab Emirates, as a Cyber Security Consultant. His research interests include blockchain technology, the Internet of Things (IoT), and reverse engineering.

MAZIN DEBE received the B.Sc. degree in computer engineering and the M.Sc. degree in electrical and computer engineering from the Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates. He is currently working with the Center for Cyber-Physical Systems, Khalifa University of Science and Technology, as a Research Associate. He has published four research articles in highly ranked IEEE conferences and journals. His research interests include blockchain technology, the Internet of Things (IoT), fog computing, and supply chain applications.

KHALED SALAH (Senior Member, IEEE) received the B.S. degree in computer engineering with a minor in computer science from Iowa State University, Ames, IA, USA, in 1990, and the M.S. degree in computer systems engineering and the Ph.D. degree in computer science from the Illinois Institute of Technology, Chicago, IL, USA, in 1994 and 2000, respectively. In August 2010, he joined Khalifa University, where he is teaching graduate and undergraduate courses in the areas of cloud computing, computer and network security, computer networks, operating systems, and performance modeling and analysis. Prior to joining Khalifa University, he worked for ten years with the Department of Information and Computer Science, King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia. He is currently a Full Professor with the Department of Electrical and Computer Engineering, Khalifa University, United Arab Emirates. He has over 190 publications and three patents. He is a member of IEEE Blockchain Education Committee. He was a recipient of the Khalifa University Outstanding Research Award 2014/2015, the KFUPM University Excellence in Research Award of 2008/2009, the KFUPM Best Research Project Award of 2009/2010, and the Departmental Awards for Distinguished Research and Teaching in prior years. He has been giving a number of international keynote speeches, invited talks, tutorials, and research seminars on the subjects of Blockchain, the IoT, Fog and Cloud Computing, and Cybersecurity. He serves on the Editorial Boards for many WOS-listed journals, including *IET Communications*, *IET Networks*, JNCA (Elsevier), SCN (Wiley), IJNM (Wiley), J.UCS, and AJSE. He is the Track Chair of IEEE Globecom 2018 on Cloud Computing. He is an Associate Editor of IEEE Blockchain Newsletter.

RAJA JAYARAMAN received the bachelor's and master's degrees in mathematics from India, the M.Sc. degree in industrial engineering from New Mexico State University, and the Ph.D. degree in industrial engineering from Texas Tech University. His postdoctoral research focused on technology adoption and implementation of innovative practices in the healthcare supply chain logistics and service delivery. He is currently an Associate Professor with the Department of Industrial and Systems Engineering, Khalifa University, Abu Dhabi, United Arab Emirates. He has led several successful research projects and pilot implementations in the area of supply chain data standards adoption in the US healthcare system. His primary research interests include blockchain technology, the Internet of Things (IoT), systems engineering, and process optimization.

JUNAID ARSHAD received the Ph.D. degree in computer security from the University of Leeds, U.K., in 2011. He is currently an Associate Professor with the School of Computing and Digital Technology, Birmingham City University, U.K. He has been actively involved in publishing high quality research within cybersecurity. He has successfully published at high quality venues, including journals, book chapters, conferences, and workshops. His research interests include investigating security challenges for diverse computing paradigms, such as distributed computing, cloud computing, the IoT, and distributed ledger technologies. He is an Associate Editor of the *Cluster Computing* and IEEE ACCESS journals. He regularly serves on program and review committees for several journals and conferences.

MUHAMMAD HABIB UR REHMAN (Senior Member, IEEE) received the bachelor's and master's degrees from COMSATS University Islamabad, Pakistan, and the Ph.D. degree from the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. He is currently working with the Center for Cyber-Physical Systems, Khalifa University, United Arab Emirates, as a Postdoctoral Research Fellow. He is also working on trustworthy blockchain technologies for intelligent cyber-physical systems. He has authored or coauthored 40 international publications, including journal articles, conference proceedings, book chapters, and magazine articles, whereby his four articles are categorized as highly cited publications by Web-of-Science. His main research activities include research and development of trust models for decentralized and trustworthy artificial intelligence applications for cyber-physical systems. His research interests include blockchain technologies, cyber-physical systems, secure key management, big data, edge computing, and the industrial IoT. He is a Bright Spark Fellow. He has been an alumnae of DAAD's Postdoctoral Network, since September 2019. He received gold medals and 100% fee-waiver scholarships from COMSATS University Islamabad.