Permissioned Blockchain Reinforced API Platform for Data Management in IoT-based Sensor Networks

Marc Jayson Baucas and Petros Spachos
School of Engineering, University of Guelph, Guelph, ON, Canada

***Abstract*—With the rise of IoT-based sensor networks, there is an increasing need for proper security mechanisms and efficient management of crucial resources, such as energy. At the same time, as more services are integrated, more sensors are introduced into the network. As a consequence, the architecture that manages these sensors need to be improved. Blockchains can be a solution in terms of data security. To cater to the inexpensive devices used as sensors in most IoT-based sensor networks, usually, permissioned blockchains are used as its base data management structure. However, due to the diverse collection of sensors that can be incorporated into a network, a static database is not enough. A more sustainable and automative system is needed as its service administrator. Therefore, we chose to integrate smart contracts to enable adaptive and dynamic automation. Lastly, to manage the reception of all the incoming data, a proper interface is required. Therefore, we chose to encapsulate the blockchain within a REST API to enable a systematic allocation of network resources. With these technologies, we propose a low-cost platform to address the management issues of current IoT-based sensor networks. We tested our design against a commercial REST API service and standard socket communication to test its feasibility. The testing metrics were latency and throughput. Based on the results, our platform proved to be the most stable and proficient among the configurations. Therefore, our proposed design shows promise as a low-cost, secure, and systematic means of managing IoT-based sensor networks.**

I. Introduction

An Internet of Things (IoT) network is a means of interconnecting devices into a wireless mesh that transmits to a shared storage for analysis [1]. This network has a subset for sensing devices called Wireless Sensor Networks (WSN). It is a collection of small low-cost sensing devices that wirelessly interact with one another [2]. These networks are used to monitor different points of interest. However, with the recent paradigm shift in smart environments, many applications demand sensing platforms to handle large amounts of moving data in real-time.

There are two approaches to addressing this influx of data [3]. First, focus on improving the sensor design to allow better control over the data that is sent to the server. Second, improve the server by allowing to cater to the data traffic. Most sensing devices are low-cost. Therefore, sensing devices are designed to be low-end, so pre-processing data before it reaches the server is unfavourable. With limited options in optimizing these sensors, we focus on creating a platform that can manage the data as more sensing devices are introduced.

With the focus on the server-side of the architecture, we needed a database that can handle the incoming data, hence, blockchain technology was selected. It is an immutable data structure that provides a decentralized take on automated data management [4]. Once a sensing network is deployed, it also needs to be self-sustaining to maximize its data collection rates. We chose to use smart contracts to moderate the incoming data. These are condition-based functions that can be embedded within a blockchain to automate certain processes [5]. To eliminate the need for high processing power due to the protocols of the blockchain, we elected to use its permissioned variant. This type gives same amount of security while keeping its automating capabilities. Also, a decentralized approach improves the ability of our platform to balance its data traffic. To further improve this load balancing, a Representational state transfer Application Programming Interface (REST API) was implemented to manage the data within the blockchain. It is a flexible and programmable interface that can be used to define how devices can interact with a server via Hypertext Transfer Protocol (HTTP) [6]. With these presented technologies, we propose a design that a platform that can manage diverse amounts of sensor data from IoT-based sensor networks.

The rest of this paper is as follows: Section II discusses the different key technologies used in the platform. Section III features the platform and how each part was designed. Section IV presents the different preliminary experiments that were conducted to test the feasibility of the platform and its results. Finally, Section V are conclusions of this work.

II. Background

A. Permissioned Blockchain

The data structure that was selected to hold and manage the incoming data was a blockchain. A blockchain is similar to a linked list, where a collection of blocks are linked cryptographically [7]. Sensing networks rely on data integrity and security, which blockchains ensure due to their immutability [8]. There are two types of blockchains; public and permissioned [9]. These two types differ in terms of giving users access and control over the blockchain and its stored information.

A public blockchain uses a Proof of Work protocol that relies on processing power to grant user's access [4]. On the other hand, permissioned blockchains use a pre-authorized protocol where registered users are granted access upon its creation [10]. Due to the Proof of Work protocol, a public blockchain requires more processing power from its miners. On the other hand, a pre-authorized protocol lessens this processing requirement by defining its trusted devices as the blockchain is initialized [11]. This criterion points out the benefits of permissioned blockchains over its public variant in terms of resource requirements. However, it suffers from device authentication due to its pre-authorized design.

In our platform, we decided to use the permissioned approach. This alternative eliminates the need for processing power. Most sensors lack the processor requirements for complex calculations. Also, public blockchains have a high processor requirement from its miners due to its proof of work protocol [12]. A permissioned blockchain makes it more possible to incorporate low-end devices. However, this type of blockchain suffers from having its devices pre-authorized and cannot be dynamically assigned, unlike its public alternative. As a result, defining its trusted devices are more inconvenient. The blockchain would have to be reinitialized every time a new sensor is added to the network. However, before IoT Sensor Networks are deployed, the devices that are associated with the collection of data are already defined and recognized. As a result, there would be no need for a dynamic means of defining trusted devices since the sensors within the WSN is already defined at the beginning of the data collection. However, to keep a WSN continuously running and collecting data, it needs to be more self-sustaining. As a result, Smart contracts were selected to be incorporated to enable process automation within the data structure.

B. Smart Contracts

A smart contract is a term-based transaction protocol [13]. Each contract is based on a set term or agreement and a defined purpose [14]. Once this agreement is triggered, the contract is activated and the function is independently executed. Smart contracts were incorporated in blockchains to allow the automation of transaction handling [15]. Maintaining a stable uptime is important for WSNs. To be able to continuously collect data, the server and its sensors must operate almost indefinitely. Using the automative capabilities of a smart contract can make the network more self-sustaining. This technology introduces a wide range of operations that the server can do without the need for manual triggers.

Another issue is that WSN architectures need to be updated continuously due to the evolving sensors and topologies that are introduced every generation [16]. With the integration of smart contract, networks can be defined and managed to cater to a diverse collection of data types and sources. Also, their automative properties can enable a more efficient means of updating the network architecture. Thus, WSNs can cover a wider scope of sensors.

Pustisek et al proposed a mobile platform that presents the strengths of smart contracts [17]. They show how these contracts provide a modular and adaptive opportunity for any architecture that needs to manage diverse entities. In their case, they designed a platform for 5G applications. It was composed of service-specific and auxiliary smart contracts to cater to a vast array of operations. Another implementation is from Wang et al [18]. They proposed a smart contract-based architecture to improve the Quality of Service (QoS) of service-oriented computing. With the use of smart contracts, they show its capability of creating an adaptive support system for different online services.

By considering this related literature, we chose to integrate smart contracts to automate the reception of data from the different sensors connected to the blockchain. To further improve the platform, we need an effective interface to interact with the blockchain and trigger the smart contracts. Therefore, we chose REST API services to create a programmable and responsive medium.

C. REST API

A REST API is an interface that uses HTTP communication to establish a flexible and programmable medium for client-server interactions [19]. It is used to provide web-based services that allow a user to access and manage data over a wireless network. This level of access and control over a server is defined by the paths that are provided by the REST API [20]. By limiting the number of ways that a client can interact with the server, the data can be made more secure. REST APIs can be used as a monitoring infrastructure that manages the collected data from the sensors [6]. Also, it contains a dynamic means of allocating resources called 'load balancing' [21]. This protocol allows the server to cater to scaling sensor networks. As a result, our design takes this approach and incorporates Smart contracts and Blockchain technology to create a more automated structure for better sustainability and scalability.

These technologies were used in designing the platform to create a more flexible network that can sustain itself and require minimum maintenance. By using permissioned blockchains with smart contracts as the backbone of the network, we can create a smarter IoT network administrator for better automation and management of registered devices for data sensing and collection. Further reinforced with a REST API interface, the network has a more systematic means of interacting with the incoming sensor data.

III. Platform Overview and Design

A. Overview

Most issues for IoT-based data sensing networks stem from handling incoming data and database scalability. These networks need a better more secure way of managing the data that is being collected from different sensing devices. Blockchains are structures known for their immutability. This feature is beneficial for system administration to create a more secure set of constraints and processes for the network, to avoid the need for high-end processors due to the Proof of work protocol built-in most public blockchains. We elected to use permissioned blockchains instead. However, to avoid it from being a simple data-structure that holds registered devices, it must be able to sustain itself. As a result, smart contracts were included for better system automation and sustainability.

Some IoT networks cater to different sensing devices. As a result, the types of data that it receives will be diverse. With the use of the access protocol that blockchain can provide as well as the automation capabilities of smart contracts, the platform can filter through the different types of data in a more efficient and organized manner. Lastly, to allow the server to be more accessible to all sensing devices included in the network, the REST API was selected to manage the blockchain. We aim to combine these concepts to create a flexible platform for hosting data sensing applications in IoT networks. Therefore, we propose a REST API platform that uses permissioned blockchains equipped with smart contracts for system administration in an IoT-based data sensing network.

B. Components

The platform is composed of a server and multiple clients. The server made use of a REST API web service. A web API service was selected to make it easier to make the server modular and flexible. This design gives the platform the ability to accommodate any additional functions and client-server interactions in future iterations. The web interface was coded using a combination of Python 3.6 and Flask. Within the Flask script includes the initialization and management of the blockchain. This design choice was made to make sure that the blockchain is initialized whenever the web-server is deployed.

The server will be running on an Intel Xeon Processor E5-2640 computer that uses an Ubuntu 16.04 OS (operating system) for web service deployment convenience. As for the clients, we chose Raspberry Pi 3 Model B's that run on a Raspbian-Jessie OS. Using Pis made it easier for rapid prototyping as well as replicating the same scripts on multiple devices. Also, Pis were selected because they were low-end devices, which is what IoT-based data sensing networks use. To simulate an IoT data sensing network, multiple Pis were programmed to interact with the web service at the same time.

C. Design Flow

The preliminary design is of the platform is a server-client architecture. The server was built by creating a Flask Web Service in Python. Within the script is the Blockchain structure that gets initialized once the server is deployed. Each block contains the following key features; Unique ID, Smart contract, and List of associated devices. The unique ID is for clients to determine which block needs to interact with the incoming data. The smart contract is embedded into the block to contain the function that will make use of the data. The list of associated devices contains IDs of devices that are permitted to access the blockchain and input data.

The web server was initially programmed to have two main API paths. The first path is a GET request to display server and blockchain information on a simple web page. This path was added to be able to keep track of the movement of data within the blockchain. The next path is a POST request for the client to send data to the server. This POST request is identified through the ID of the sender. Also, it contains a JSON string that includes; the unique ID of the block being interacted with, the ID of the client, and the data being sent. Table I presents the API requests programmed in the server in more detail.

Resource	**Verb**	**Action**	**Response code**
/data	GET	get database	200
/send&id=?	POST	send data	200

TABLE I: HTTP Request table.

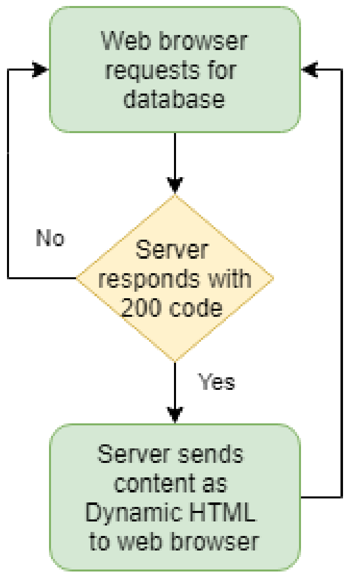
Web browser requests for database
Server responds with 200 code
No
Yes
Server sends content as Dynamic HTML to web browser

Fig. 1: GET interaction.

The general flow of the platform starts with the client sending data to the server. For simulation purposes, the client was programmed to send random data for now. This data will then be packaged into a POST request along with the required ID's. Then, the server verifies this request based on the provided information. If the block with the client IDs is validated, it will execute the smart contract and use the data as input.

Initially, the function was programmed to save the data point along with the client's ID to a dictionary file that was initialized once the Flask web server is deployed. This interaction between the client and the server is replicated to multiple Pis that have different ID's. Figure 1 shows the GET interaction between the server and a web interface while Figure 2 presents the POST interaction between the sensor client and the server in a flow diagram. The next section presents the tests that were carried out as well as their results to prove the feasibility of the preliminary design of the platform.

IV. Tests and Results

A. Testbed

The experimental testbed for the proposed platform was designed to have multiple Pis sending data to the server.

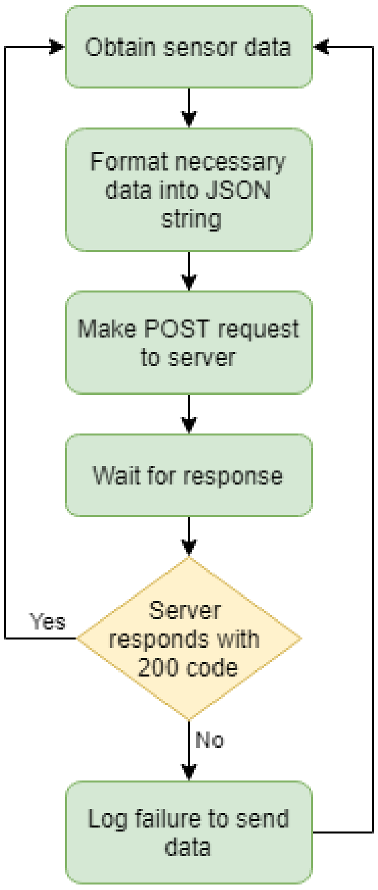
Obtain sensor data
Format necessary data into JSON string
Make POST request to server
Wait for response
Server responds with 200 code
Yes
No
Log failure to send data

Fig. 2: POST interaction.

The server then responds by either blocking or accepting the request based on the provided information. The metrics that were chosen to test the platform were average latency and maximum throughput. These metrics were tested to check the feasibility of the design in terms of network proficiency and scalability. More data means the event being measured is represented better. However, if a server is not able to accommodate the incoming data, the sensing devices will not be able to send data continuously because of the backlog.

Also, a sensor network that is not able to handle an increasing number of connected devices will not be able to serve its services adequately. Therefore, the proposed platform was set up to have multiple Pis transmit to a central server. Figure 3 shows an example of the testbed using a four-sensor arrangement. We set up this testbed to test the latency of the platform against other standard configurations of collecting data. Our test chose to compare our proposed design against a client-server socket communication and a commercial REST API.

B. Average Latency Test

The test was conducted by having the latencies of each setup was measured as the number of devices were increased from 1, 4, and 8. The latency is measured as the average elapsed time between a request from a client and a response

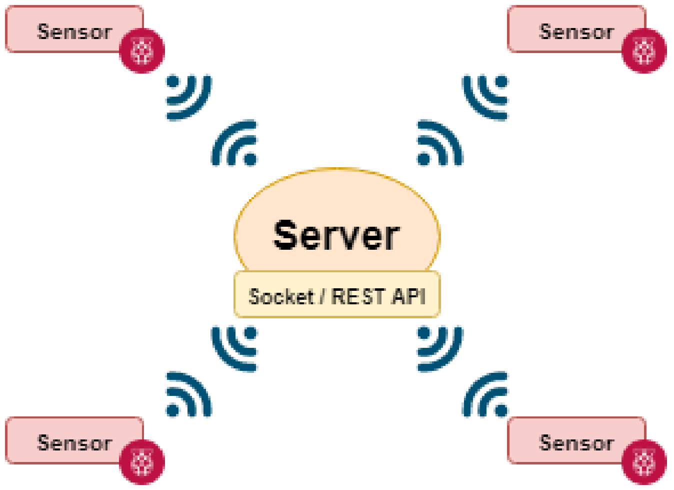

Fig. 3: Four sensor network arrangement example for testbed.

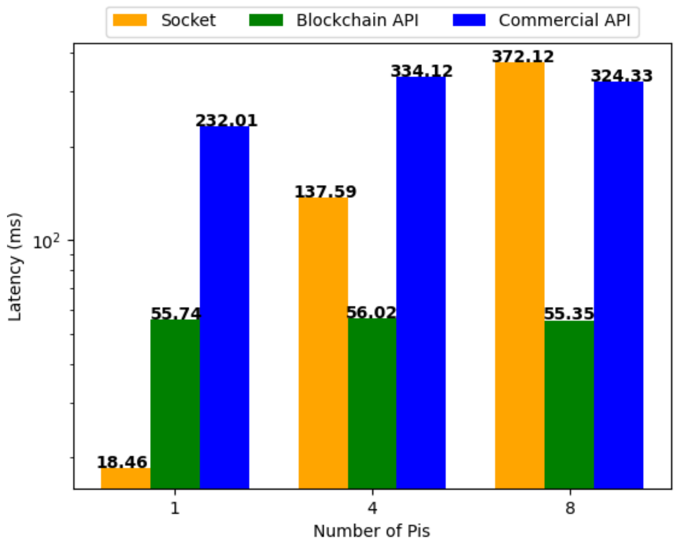

Fig. 4: Average latency of the different configurations at varying number of Pis.

from the server. The following Equation presents the logic behind calculating the average latency, including the required variables, as:

$$x = \frac{\sum_{i=1}^{n}(a-b)}{n} \tag{1}$$

where x is the average latency, a it the time the server responded to the request, b is the time the client made a request, and n is the number of samples.

We carried out this experiment by calculating the average latency of each configuration within 60 samples (n). Figure 4 shows the results of the test. Based on the results, we can observe that both the REST API setups were relatively consistent. On the other hand, the socket setup runs into issues in terms of scalability. As the number of Pis increased, its average latency increased significantly as well. Although the socket setup had the best single Pi performance, it falls off as the number of Pis sending data to the server increases. As a result, scalability becomes an issue for this issue. This observation could be attributed to the inability of the network to provide enough sockets required to support a growing network.

At the same time, the REST API setups can flexibly allocate resources to cater to the increase in connecting sensors. Between the two API setups, the performance of the commercial API was worse than the local as it yielded higher latencies in all iterations of the experiment. We can attribute this behaviour to the commercial API being hosted by another server remotely. It shows the benefits of hosting your REST API server for managing your sensor network instead of using a remote service as it could add unnecessary latency. Overall, our design yielded the most consistent and least overall latency among the tested setups.

C. Maximum Throughput Analysis

To further investigate our proposed platform, we looked to analyze the maximum throughput of each setup. This metric would allow us to understand more of the capabilities of each configuration in terms of accepted data overtime. An IoT-based sensor network needs to be running continuously to be efficient. However, if the amount of data that it can handle at a time is small, then it lacks proficiency. Therefore, we took the calculated average latency of each configuration and used it to estimate their maximum throughput. The following Equation presents the logic used to calculate the maximum throughput converted to Megabits per second, as:

$$y = \frac{c * 8}{x} * \frac{1Mbps}{1000000bps} \tag{2}$$

where y is the maximum throughput, x is the average latency, and c is the maximum TCP window size.

Since we are calculating for the highest possible outcome of the metric, we assume the largest possible Transmission Control Protocol (TCP) window size, which is 64 KB (65536 Bytes). Another assumption is that there is no packet loss during any of the transmissions to get the maximum throughput value. Using the cited equations and conditions, the desired metric was calculated and plotted. Figure 5 presents the results of each setup grouped according to the number of Pis used in each iteration. Based on the values obtained, it shows how the proposed platform yielded the best overall throughput over the tested iterations. Even though the socket setup had the highest throughput with 1 Pi connected, its performance decreased significantly as the number of Pis increased. We can attribute this behaviour to the static space allocation for socket protocols. By having the assigning of the socket variables initially, the protocol is not able to adjust as the number of sensors introduced grows. Meanwhile, REST APIs are more dynamic in terms of allotting space for devices that attempt to connect. As a result, these protocols are able to adjust more freely to the changes in the number of pis.

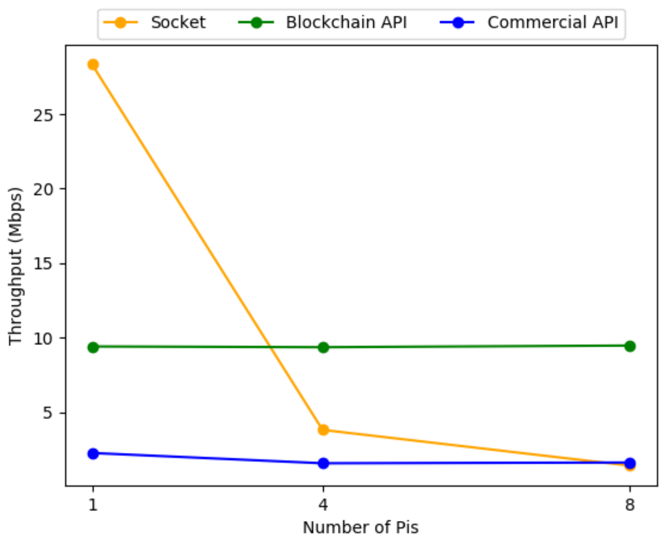

Fig. 5: Maximum throughput of the different configurations at varying number of Pis.

V. Conclusion

In conclusion, we proposed a network platform that made use of blockchains, smart contracts and REST APIs. WSNs run into issues in management and security. Our design proposed the use of permissioned blockchains to strengthen the secureness of the network. We selected blockchains due to their immutability, which creates an immutable data storage for the sensing network. Then, we added smart contracts to provide a systematic means of automating the blockchain equipped sensor network. With immutability, blockchains can be a solution to the lack of security of WSNs. We selected permissioned blockchains over its public variant due to its exclusivity in allowing authorized devices to modify the blockchain. As a result, trusted devices can be predefined, which makes the network more secure. We incorporated a REST API service to encapsulate the blockchain to create a more flexible and sustainable system due to its programmability. Also, REST APIs are an already established interface that provides convenient network administrator services.

We tested our proposed platform for its proficiency in handling requests from its sensors as the network grows. Our tests used latency and throughput as their metrics. We measured the latency as the average elapsed time between a sensor requesting to send data and the server responding to the request. Meanwhile, we calculated the throughput as the maximum possible data speed of the setup. Each experiment had our proposed platform tested against a server-client socket configuration and commercial REST API. As the number of sensors within the network increased, the results showed our proposed design as the most consistent and most proficient among the 3. It had the least overall elapsed time taken in terms of the sensor making a request to send data and the server responding accordingly. Therefore, the test shows the scalability and efficiency of our proposed design. Overall, our platform proved to be a feasible option in terms of creating a secure and effective administrator for WSNs.

References

[1] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "Iot middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, Feb 2017.

[2] H. Kim and S. Han, "An efficient sensor deployment scheme for large-scale wireless sensor networks," *IEEE Communications Letters*, vol. 19, no. 1, pp. 98–101, Jan 2015.

[3] Y. Xu and A. Helal, "Scalable cloud–sensor architecture for the internet of things," *IEEE Internet of Things Journal*, vol. 3, no. 3, pp. 285–298, June 2016.

[4] A. Nayak and K. Dutta, "Blockchain: The perfect data protection tool," in *2017 International Conference on Intelligent Computing and Control (I2C2)*, June 2017, pp. 1–3.

[5] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F. Wang, "Blockchain-enabled smart contracts: Architecture, applications, and future trends," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2266–2277, Nov 2019.

[6] A. Ciuffoletti, "Occi-iot: An api to deploy and operate an iot infrastructure," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1341–1348, Oct 2017.

[7] L. Zhou, L. Wang, Y. Sun, and P. Lv, "Beekeeper: A blockchain-based iot system with secure storage and homomorphic computation," *IEEE Access*, vol. 6, pp. 43472–43488, 2018.

[8] C. H. Liu, Q. Lin, and S. Wen, "Blockchain-enabled data collection and sharing for industrial iot with deep reinforcement learning," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3516–3526, June 2019.

[9] M. E. Peck, "Blockchain world - do you need a blockchain? this chart will tell you if the technology can solve your problem," *IEEE Spectrum*, vol. 54, no. 10, pp. 38–60, October 2017.

[10] J. Al-Jaroodi and N. Mohamed, "Blockchain in industries: A survey," *IEEE Access*, vol. 7, pp. 36500–36515, 2019.

[11] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, April 2018.

[12] M. Alaslani, F. Nawab, and B. Shihada, "Blockchain in iot systems: End-to-end delay evaluation," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8332–8344, Oct 2019.

[13] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F. Wang, "Blockchain-enabled smart contracts: Architecture, applications, and future trends," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2266–2277, Nov 2019.

[14] J. Liu and Z. Liu, "A survey on security verification of blockchain smart contracts," *IEEE Access*, vol. 7, pp. 77894–77904, 2019.

[15] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, April 2019.

[16] M. Farsi, M. Badawy, M. Moustafa, H. Arafat Ali, and Y. Abdulazeem, "A congestion-aware clustering and routing (ccr) protocol for mitigating congestion in wsn," *IEEE Access*, vol. 7, pp. 105402–105419, 2019.

[17] M. Pustišek, J. Turk, and A. Kos, "Secure modular smart contract platform for multi-tenant 5g applications," *IEEE Access*, vol. 8, pp. 150626–150646, 2020.

[18] P. Wang, J. Meng, J. Chen, T. Liu, Y. Zhan, W. Tsai, and Z. Jin, "Smart contract-based negotiation for adaptive qos-aware service composition," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1403–1420, 2019.

[19] L. Li, W. Chou, W. Zhou, and M. Luo, "Design patterns and extensibility of rest api for networking applications," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 154–167, March 2016.

[20] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés, "Metamorphic testing of restful web apis," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1083–1099, Nov 2018.

[21] Z. Li, X. Liu, T. Wang, W. He, and C. Xia, "Ghscn: A graph neural network-based api popularity prediction method in service ecosystem," *IEEE Access*, vol. 8, pp. 137032–137051, 2020.