M2MHub: A Blockchain-based Approach for Tracking M2M Message Provenance

Darren Saguil, Qiao Xue, Qusay H. Mahmoud
Department of Electrical, Computer and Software Engineering
Ontario Tech University
Oshawa, ON, Canada
{darren.saguil,qiao.xue,qusay.mahmoud}@ontariotechu.net

***Abstract*—The Internet of Things (IoT) is a fast growing and popular subset of technology. One of the main features of IoT is device autonomy, the ability for the machines embedded in the devices to function without human intervention. This includes communications with other devices through Machine-to-Machine (M2M) communication. Unfortunately, M2M communications are only stored with *what* behaviours they have observed, without the causal relationship as to *how* or *why* they observed those behaviours. Since M2M messages can trigger more M2M messages, the provenance of issues inside an IoT system can be hidden behind a long chain of messages, so finding the root source of any problem, such as malicious or defective devices, is almost impossible to detect. To solve this problem, in this paper we introduce M2MHub, a centralized auditing system which collects M2M messages in an IoT system and stores them in a blockchain. Devices in the system can tell the hub if they wish to open, continue, or end transactions, allowing the hub to keep track of who is the provenance of the transaction and how their transaction affects other devices. A proof-of-concept simulation has been constructed, demonstrating how M2MHub may function in a real-world implementation. The current implementation is not scalable enough to be deployed to actual IoT networks, so several ideas for future work are offered.**

***Index Terms*—machine-to-machine communication, blockchain, ethereum, internet of things, provenance**

I. Introduction

Among the many new developments and innovations in embedded technology, the Internet of Things (IoT) is considered to be one of the most important avenues of innovation. Developing for IoT has been one of the main focuses for industries such as automotive, manufacturing, health care, and business [1]. IoT is defined by the use of sensory devices such as embedded cameras and Radio Frequency Identification (RFID) chips to connect any object to a network of numerous other devices to aid in improving the development, understanding, and quality of the space around us. These sensing devices can be embedded into everyday objects, such as kitchen appliances, cars, lamps and security cameras to enhance their functionality, making them ”smart”. It is estimated that 50 billion smart devices will be deployed in IoT by 2020 [2], where each device is capable of communication, storage, and many other functions unthought of before the 21st century.

An example usage of IoT is the development of smart cities. As the percentage of the world human population living in large cities reaches 60% [1], there needs to be a way to monitor, store, and analyze the large amounts of chaotic data which a city can produce in order to better plan out the city for the future. A large network of smart devices can be embedded into many objects around the city, such as traffic lights, bollards, and other unrelated objects with synergistic properties [3]. These smart objects can capture the physical, social, and economic ”flows” of the city, and warehouse them in a database. The collected data can then be analyzed by city developers in order to shift the direction of the city in a positive manner, such as planning to reduce poverty and pollution [1]. However, to have such a vision realized, these devices should be autonomous, meaning connection to a network is necessary.

IoT networks can be separated into three tiers: Edge, Fog, and Cloud. The ”Edge” of an IoT system contains the smart devices that have been discussed. The ”Fog” and ”Cloud” are both areas of the system which can provide extended functionality to the edge devices. The main difference between them is their proximity; clouds are huge data centers often placed a long distance away from the edge devices, while fog nodes are placed within the same proximity of edge devices as seen in Figure 1. They provide nearly the same services to the edge devices as fog nodes, but with reduced network latency [4] for better machine-to-machine (M2M) communications.

M2M communication refers to the exchange of data between machines without the need for humans to initiate or intervene with the transaction [5]. This form of communication has become one of the most integral parts of IoT [6] as there will be too many devices in an IoT system for humans to manually use together. Since the need for human intervention was a roadblock to IoT scalability, its removal allows for the rapid growth of IoT networks estimated today. However, this will introduce many issues, the paramount concern being device security [7].

With the number of devices connected to IoT is quickly outpacing the human population, it is important that these devices are not faulty or used in a malicious manner. There is a possibility that someone will use the Internet of Things with a destructive effect, such as controlling the switch of another person’s smart door lock, switching the oven on unexpectedly, or even controlling a victim’s self-driving vehicle. Even if the issue caused by the suspected device is not malicious and a result of a simple misconfiguration, the provenance of the issue should be investigated immediately to prevent further exploitation. Unfortunately, with M2M communication, this has become increasingly difficult.

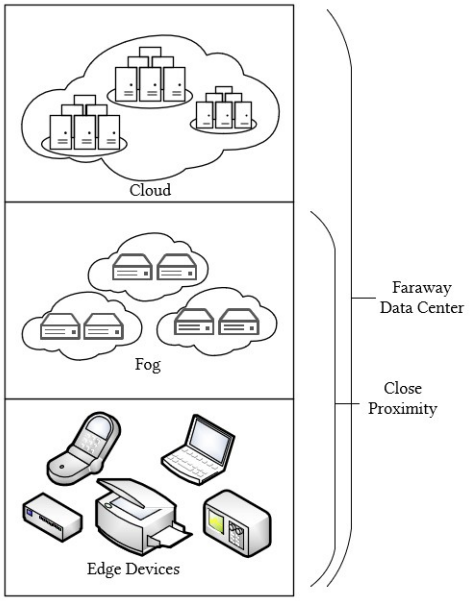

Fig. 1. The three tiers of an IoT system

Since M2M messages can trigger more M2M messages, it can be difficult to tell if an observed behaviour was caused by a single message, or a long list of chained messages [8]. Therefore, *how can we find the causal relationship between observed behaviours and the messages passed in an IoT system*?

Our proposed solution provides a centralized auditing platform which records all the messages passed through an IoT system. This system, labelled as M2MHub, was designed to track M2M transactions, their provenance, and all parties involved and store them in a blockchain. The blockchain was important as they are immutable, therefore they are resistant to tampering by potential attackers, and could potentially aid in the scalability of the system through distribution.

At the present, there is no unified definition of the blockchain, but the most widely accepted definition is a ”distributed ledger”. In general, the blockchain is a public database of hashes based on the combination of many technologies, such as cryptography and peer-to-peer networks. The blockchain is managed by a network of nodes, each node having shared responsibilities such as maintaining a copy of the blockchain or the applications associated with it. Whenever transactions are detected by any of the nodes, they broadcast them to any other holders of the blockchain to be placed in a block. This makes the blockchain a desirable way to store information; it is secure, immutable, and hashed to preserve privacy. The blockchain can be introduced in a number of ways in IoT as a solution to its issues with security, reliability, and privacy [9]. The most important function that blockchain technology provides IoT is to create trust between devices without the need for third parties [10], thus synergizing with IoT’s main tenet of device autonomy.

Although the blockchain is mainly used for cryptocurrencies such as Bitcoin and Ethereum, there are many industrial applications for it. Once such application can be tourism; tourists can carry cards holding their information, and log that information into airport or Hotel blockchains [11]. This way, the locations will have an easier time identifying and remitting tourists. Another application is a marketplace, seen in [12]. Users can upload the items they wish to sell, and buyers can place bids on that item. All of those transactions take place on the blockchain, making it a reliable place to shop. The realm of retail and manufacturing supply chains are seeing the benefits from blockchain technology as well. Nearly half of companies dealing in consumer goods are spending millions of dollars in research and development to implement the blockchain into their supply chain [13]. The energy industry sees benefits from this as well; blockchains can log power consumption in power microgrids [14]. However, the similar feature in all of these implementations is that they use an implementation of the blockchain known as Ethereum [15]. This technology was also used in the implementation of M2MHub.

This paper is divided into multiple sections, where section II details two related solutions that deal with provenance tracking. Section III details how M2MHub was developed, the tools used, and its architecture. Section IV details the results of the implementation and provides an analysis into the performance of the system. Section V summarizes the contributions and provides possible avenues for future research.

II. Related Work

A. ProvThings

ProvThings is a centralized auditing platform for IoT networks introduced by Wang et al in [8]. ProvThings provides its users with the causal relationships between behaviours observed in IoT networks. One of the most defining features of ProvThings is that it is as non-invasive as possible as it is simply overlayed on top of the network. This means that ProvThings does not need to be hard-coded into IoT devices to provide provenance tracking; it can get information on the devices used in the system through instrumentation. With this system architecture, they planned to have a system which does not only answer *what* is being sent, but also *why* and *how* it is being sent.

To provide this functionality, ProvThings deployed a number of software components around an IoT network to collect, merge, validate, and analyze records. To collect records, they implemented Provenance Collectors which monitor an IoT device’s performance and code execution. First, the source code of the device is statically analyzed for potential data sources and data sinks, places where M2M communications may exit or enter the device respectively. Since many collectors were to be deployed at once to keep track of provenance, aggregation of all the records was done a Provenance Recorder. This component merges all the records from the collectors into a single database which can be queried using SQL, and even develops provenance graphs as well.

To enforce standards set by the user in the network, the component called the Policy Monitor was used to create whitelists and blacklists of behaviours within the system. This component can be controlled by the Front-End which allows users to query, alter, and monitor the previous components

This system is capable of detecting and defending IoT networks networks from numerous IoT networks attacks, such as Drive by Downloads and Permission Abuse with a small overhead of 5%. However, their work could be extended by being able to detect possible compromised devices. As their system is not integrated with the IoT devices for the sake of back-compatibility and non-invasive monitoring, it is susceptible to device impersonation attacks. Also, as they are using a relational database to store provenance records, it is not fully tamper-proof.

B. *ProvChain*

An approach to cloud-based data provenance was introduced by Liang et al. in [16] in the form of ProvChain. ProvChain is a provenance tracking platform for cloud operations which has relatively low overhead (around 1 millisecond per recorded operation) with secure and tamper-proof records. While not directly related to IoT or M2M messages, it provided a case study in the recording of provenance and provenance validation using a blockchain.

ProvChain provides a platform for Cloud Data Provenance which includes features such as real-time data collection from the cloud, immutable data collection to ensure record integrity, user privacy through the hashing of sensitive data, and consensus-based data validation. It does this by implementing hooks and listeners to every data-altering operation to record their provenance, hashing these records and then storing them in the blockchain. To validate the information stored through consensus, a central auditor receives receipts from nodes from the blockchain network which contain information on newly mined blocks.

This approach to data provenance tracking can be communicated to IoT networks as well. Instead of implementing hooks and listeners for cloud operations, they can be implemented into device monitors, similar to what is seen in ProvThings. They could also change their storage architecture to include device names instead of cloud users, data sources instead of the names of files in the cloud servers, and data sinks instead of file actions. This could provide a tamper-proof implementation of M2M communication provenance, although it is more intrusive than ProvThings as some operations inside the devices themselves must be altered.

III. M2MHub Architecture

A. *Assumptions*

The architecture for M2MHub was designed under the assumptions that the blockchain used is immutable and encrypted, such that attempts to access or alter the blockchain is impossible or at least Pyrrhic for the attacker. Another assumption in the design is that the time it takes to mine a block in the blockchain will always take longer than the time it takes to perform a successful M2M transaction in an IoT network, and thus holding any message for validation may damage the usability of the system.

B. *Architecture*

The architecture of M2MHub follows a similar centralized auditing structure by Wang et al. in [8]. The main difference is that the IoT devices and users interact with an Ethereum- based Distributed Application (DApp) stored on the hub. By interacting with the DApp, it is possible to store and review M2M messages in the blockchain.

The hub will also function as the full node in an Ethereum-based network as it stores the blockchain and the smart contract. The full architecture for the system can be seen in Figure 2.

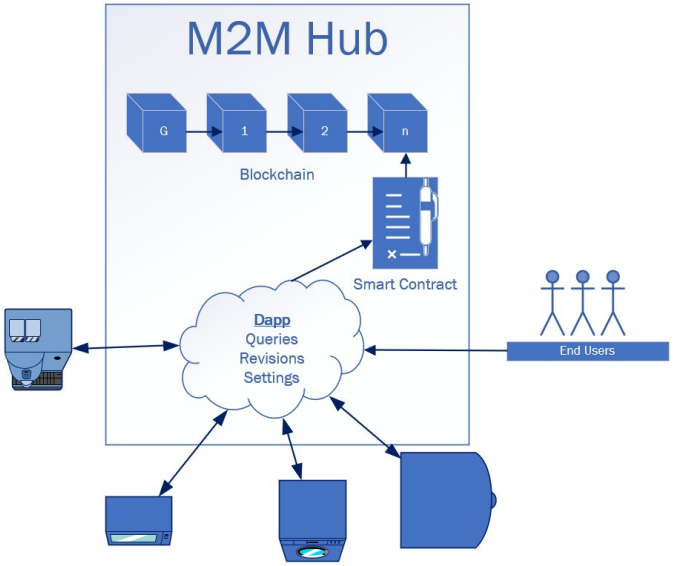

Fig. 2. Architecture of M2MHub

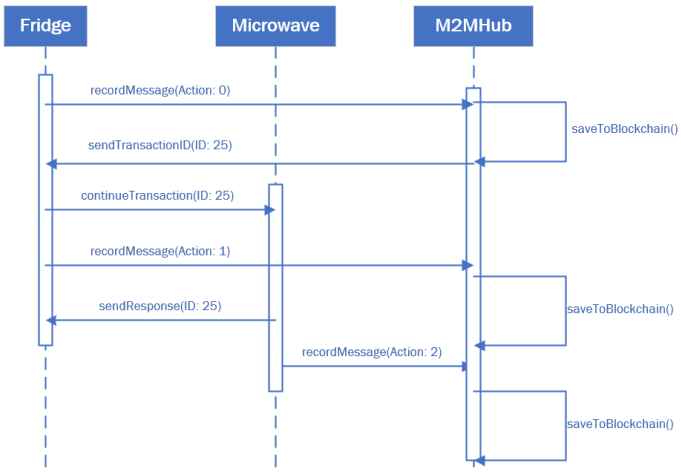

Fig. 3. Example sequence of a message passing transaction

In order for the hub to keep track of passed messages, every IoT device must first register with the system. By registering, users can give each device a human-readable name and a permission level (known in this project as clearance). Next, these devices can send each other messages as long as they provide the messaging structure provided by the M2MHub system. An example sequence as to how messages are passed in the system is presented in Figure 3.

The system follows a circuit-like structure, where the hub will assign every transaction with a specific ID, as well as the provenance of that transaction. This way, any troubleshooters will be able to keep track of the starting point of an issue by checking all the messages with that specific transaction ID, and using the provenance of the transaction.

This should reduce the search space of an M2M communication issue significantly. The full architecture for each message is as follows:

1) *TransactionID:* Keeps track of an M2M transaction in a circuit-like fashion. Every time an M2M transaction is opened, a new TransactionID will be assigned to it. All following messages which are caused as a result of this initial message will utilize the same TransactionID, thus showing the cause and effect of a single device's actions. Every device that the provenance device interacts with, as well as all every other device down the M2M message chain, must keep track of this TransactionID as in order for it to be fully tracked.
2) *ProvID:* Shows which device initiated the transaction. This data is immutable within the M2M message chain, alongside the TransactionID.
3) *SenderID:* The given name of the sender. It will mainly be used to make the M2M transactions human-readable, instead of just showing the addresses of each device as its ID.
4) *ReceiverID:* The same as the SenderID but for the recipient of the message.
5) *Action:* Indicates to the hub what the purpose of the message is. There are 3 different actions that the hub can take
    a) *Begin*: assign the message with a new TransactionID, and assign the SenderID as the ProvID
    b) *Continue*: assign the message with the same TransactionID and ProvID given
    c) *End*: push the TransactionID into a list of closed transactions to prevent further actions on this ID

When the devices interact with the DApp, the Dapp then informs the Smart Contract to update the database with the new information, thus securing registered device's information as well as any messages they send. The Smart Contract only has these two functions, thus keeping the contract feasible.

C. Implementation

To implement the proposed solution, the Ethereum blockchain was used. Ethereum is a 2nd generation blockchain which makes use of two different types of nodes: full and light nodes. Full nodes hold a copy of and performs actions on the blockchain, while light nodes run DApps to interact with it [14]. In this implementation, the M2MHub was used as both a full node and a light node in order to reduce the number of machines to add to the IoT system.

Development of the DApp was performed using the Truffle framework. Truffle is a development suite which provides developers with an easy way to create applications for the Ethereum blockchain. When Truffle is prompted, a Smart Contract coded in Solidity, the programming language used to interact with the Ethereum blockchain, will be generated. This is generated with a small template for a Node.js application that interfaces with the contract. These parts "work right out of the box", meaning that minimal configuration is needed to run the app; developers just need to provide the Solidity code and Front-End application (HTML/CSS/JavaScript).

The smart contract provides the DApp with several variables (seen in Table I) and public functions (seen in Table II, but there are only two functions which allow for the interaction between the user and the blockchain. The first one allows users to register devices to the blockchain, while the other one records messages to the blockchain. The rest of the functions were helpers to ensure validity, and help deal with using strings in validity.

In order to connect the smart contract with the web application the web application must implement the web3 JavaScript library. This can be done manually by implementing the web3 libraries into the main JavaScript file of the Node.js application. However, in this implementation, Metamask was used to inject the web3 libraries into the app. It was used as it kept a historical record of every single transaction performed in the system, as well as their transaction fees and gas usage. However, as the smart contract developed was quite large and often failing to compile during development due to its sheer size, the gas usage was extremely high.

D. Scalability

As stated in the Section III-A, since this project was using a blockchain with DApps, it can be assumed that the entire system can be distributed. However, in the current implementation, creating a local cluster was not possible as Ganache was used as the private blockchain. Ganache is great for quickly prototyping blockchain applications, but actual deployment of the blockchain to multiple machines is not supported. If we were to use another software to develop a private blockchain, such as Geth, then we could potentially share the data directory of the main full node to create copies on other machines. However, this would require a restructuring of the entire project.

Aside from the technical issues from the implementation, new devices can be easily registered using the functions provided by the smart contract. The same can be said for message passing. The biggest bottleneck to the scalability of this system is the sheer size of the contract. As the contract stores real names to keep track of the devices as strings, numerous helper methods to handle strings had to be added in order to ensure their correctness when stored. This makes deploying the contract to the blockchain and using its functions very expensive, and ultimately prevents this system from being deployed to a public network with real monetary values.

Thankfully, since this project deployed to Ganache, an application which allows developers to host their own personal blockchain. The currency used in this blockchain has no actual value, so any amount of gas used is trivial. For example, a message passed in the system usually takes up around 0.05 ether in gas, which is around 5.00$ U.S. This is considerably large if there are thousands of devices sending messages to each other rapidly. Since Ganache gives all accounts at least 10,000.00$ U.S. in ether while rapidly mining more, there are no worries about running out. Unfortunately, this means that the system is restricted to private blockchains, as deployment to a public blockchain would be extremely expensive.

E. Challenges and Solutions

Several challenges were encountered in the development of the smart contract. Solidity, a language for implement smart contracts, is not as flexible as regular programming languages; it is kept simple for the sake of low cost in the blockchain.

TABLE I
THE VARIABLES PROVIDED IN THE SMART CONTRACT

Variable	Meaning	Initial Value
Owner	The address of the node running the DApp	-
transactionCount	Keeps track of the transactions in the system	
messageCount	Keeps track of the number of messages in the system	0
deviceCount	Keeps track of the number of devices registered	0
transactions	Maps every transaction number to the last message sent in the transaction	-
lookup	Maps every device name to their address	-
numLookup	Maps every device number to their address	-
devices	Maps every device address to its information	-
endedTransactions	Stores all the inactive transactionIDs	-
messages	Stores content of every message sent to be displayed	-
deviceList	Stores every device to be displayed	-

TABLE II
THE FUNCTIONS PROVIDED BY THE SMART CONTRACT

Function	Meaning	Modifier
register()	Registers a new device to the blockchain	public
record()	Records a new message to the blockchain	public
deviceExists()	Checks if a device has been registered	public view returns(bool)
ongoing()	Checks if a transaction exists/is not closed	public view returns(bool)
canSend()	Checks if a device has the permissions to send a message	public view returns(bool)
compareStrings()	Checks if two strings are the same	public pure returns(bool)
addToLookup()	Maps a hash of a device's name to their address	public
lookupName()	Finds a device address based on its name	public view returns(address)

For example, dynamically sized variables (such as arrays and strings) cannot be used as indices for mappings, meaning that mapping a string to the device address is not possible. This is why there are many "lookup" variables and functions seen in Tables I and II. To properly store names in arrays, they had to be hashed into fixed 32-byte variables. It should also be noted that the lack of dynamically sized variables in mappings also made any array above one- dimension also impossible. To bypass this, structs containing a list of variables to be stored as an array was used instead, but these "arrays" are fixed in size.

Another challenge was the reduction of the size of the smart contract. At one point in the development, there were so many variables that the smart contract could not be compiled. To solve this, a heavy reduction in functionality was needed. For example, at one point each device could save up to 10 exceptions (i.e. other devices which could send messages to it while having a lower clearance). This was ultimately reduced to 1 exception to save space. Also, some variables were deemed unnecessary and were culled as their values could be derived from other variables (e.g. a list of active transactions was not needed as we can derive this data from the message list).

IV. EVALUATION

A. Testing

Validation testing on the backend of the application was simple; as Truffle is already bundled with the Mocha framework and Chai assertion libraries, all that is needed to perform validation testing is a JavaScript file detailing what to test. In this implementation, there were 3 main functions that needed to be tested: Device Registry, Message Recording, and Message Validation. There is no need to validate devices when they enter the registry; all the registry does is give names to every single device using the system.

A portion of the JavaScript file associated with testing can be seen in Figure 4. This is the test code for the Message Validation function. Prior to this, the Device Registry was tested by implementing 3 different devices of increasing security clearance named low level, mid level, and top level respectively. This usually executed with a time of around 250 milliseconds.

Next, the Message Recording was tested by simulating a message exchange between two devices. The top level device initiated a transaction with the low level device, and began a transaction with an ID of 0. The low level device continued the transaction, returning a message to the top level device through the same transaction. Even though the low level device does not have the security clearance to message the top level device, the message was still recorded without an error as the top level device was the provenance. Next, the top level device closes the transaction, thus pushing the transaction ID into the closed transaction stack. This usually executed with a time of around 1000 milliseconds.

The portion seen in Figure 4 shows the Message Validation portion of the testing code. It shows many of the invalid messages that can be sent through the system. The first assertion (denoted by assert.ok()) checks if a message is valid even after it sends a message to a closed transaction. The second assertion checks if the message is still valid the sender has a lower clearance than the receiver. The third assertion checks if the message is valid after if it is sent to a non-existent ID. The last assertion checks if a message is valid if the provenance device of a transaction does not have enough clearance to send to a higher clearance device. All the messages in this portion were invalid as per the requirements. This usually executed with a time of around 2500 milliseconds.

```
it("logs error messages in the blockchain", function() {
    return M2MHub.deployed().then(function(instance) {
        hubInstance = instance;

        // send a "Continue" message on a closed transaction
        hubInstance.record("top_level", 0, "low_level", "continue()", 1);
        return hubInstance.messageCount();
    }).then(function(number) {

        // use the id to fetch the message
        assert.equal(number, 4, "there are a total of 4 messages now");
        return hubInstance.messages(number-1);
    }).then(function(message) {
        assert.ok(!message[8], "invalid (continuing a closed transaction)")

        // send a "Begin" message from the mid_level to the top_level
        hubInstance.record("mid_level", 0, "top_level", "continue()", 0);
        return hubInstance.messageCount();
    }).then(function(number) {
        return hubInstance.messages(number-1);
    }).then(function(message) {
        assert.ok(!message[8], "invalid (sender has lower clearance than receiver)")

        // send a "Continue" message on a non-existent transaction
        hubInstance.record("top_level", 6000, "mid_level", "continue()", 1);
        return hubInstance.messageCount();
    }).then(function(number) {
        return hubInstance.messages(number-1);
    }).then(function(message) {
        assert.ok(!message[8], "invalid (transaction ID does not exist)")

        // Begin a valid transaction
        hubInstance.record("mid_level", NaN, "low_level", "start()", 0);
        return hubInstance.messageCount();
    }).then(function(number) {
        return hubInstance.messages(number-1);
    }).then(function(message) {

        // Send a "Continue" message from the low_level to the top_level using the ID
        hubInstance.record("low_level", message[0], "top_level", "continue()", 1);
        return hubInstance.messageCount();
    }).then(function(number) {
        return hubInstance.messages(number-1)
    }).then(function(message) {
        assert.ok(!message[8], "invalid (provenance does not have enough clearance)")
    });
```

Fig. 4. Validation testing example

B. *DApp Front-End*

The front end of the DApp was meant to be easy to read minimal, and only used Bootstrap for its CSS. It only consists of four parts: a form to register a device, a table of all connected devices (seen in Figure 5), a form to simulate sending a message between two devices, and a list of messages (seen in Figure 6).

Registered Devices

Name	Clearance
a	111
b	111
c	113

Fig. 5. List of devices as seen in the DApp

The device list simply shows the name of the devices, and their clearance. Clearance is just an integer; any device can message any other device with an equal or lesser clearance. For example, in Figure 5, device ”c” can message any device, while devices ”a” and ”b” can only message each other. The message list displays every message that has been passed in the system. It will also warn the user of possibly incorrect or malicious messages by highlighting them in red, as seen in Figure 6.

For example, the second transaction in the image was a device attempting to open up a transaction to a device that has more clearance, thus highlighting it as an error.

C. *Performance*

While this implementation is valid as a proof-of-concept, there needs to be work on the size of the smart contract. The current code contains too many operations within it, thus slowing down the performance. Although this situation can be ameliorated by running on a private network and spending an exorbitant amount of gas to normalize the speeds, it can be quite alarming when examining scalability.

As IoT networks become more densely packed with IoT devices, each one sending each other M2M messages at a constant rate, it must take a cluster of high-powered fog nodes to be able to keep the blockchain up to date with the current state of the system. This might be too expensive to implement, and therefore some of the functions of the smart contract should be transferred to the JavaScript in the DApp in order to keep the gas costs low. For example, the smart contract handles string comparisons and searching arrays for values. These non-blockchain functions, known as view functions, could possible by transferred to the DApp instead to alleviate the smart contract of its workload. However, some view functions should be kept in the smart contract for security reasons in order to prevent alterations by potential attackers. For example, clearance checks should not be performed in the front-end as it prevents unauthorized usage of potentially important devices.

These performance issues can be seen in Figure 7. Both lines inside of this figure displays the maximum gas cost of performing Register Device and Record Message operations over several repeat transactions. For the Register Device function, the initial registration had a potential maximum gas cost of 0.024121 ETH, or $2.58 USD. From this, the gas cost increased by about $0.013 USD per device registered. As for message recording, it started at a max gas cost of 0.054179 ETH, or $5.79 USD, and increased by about $0.016 USD per message sent.

Note that, for this measurement, the worst-case scenario for message passing was used in all of the messages (i.e. only ”Begin” messages were sent, thus causing extra storage operations on the blockchain). We can see that the possible cost of each operation on the blockchain is increasing by about 1 to 2 US cents per operation; if this were deployed into a large-scale IoT network with hundreds of devices sending exponentially more messages, the gas needed to power these operations may need extremely powerful hardware to compute.

The marketplace system proposed by Prasad et al. in [12] displayed similar results as well. They implemented a marketplace using the Ethereum blockchain and other similar tools. In their research, they found that their application had a gas cost of $219.24 USD to deploy, $46.72 USD for their most expensive function, and $1.90 USD for their least expensive function.

Messages

Transaction	Provenance	To	From	Op	Contents	Notices
0	a	b	a	0	open()	None
1	a	c	a	0	open()	Providence/Sender did not have required security clearance
0	a	a	b	1	respond()	None
0	a	b	a	2	ack()	None
0	a	a	b	1	hello()	Attempted message on invalid transaction

Fig. 6. List of messages as seen in the DApp

However, they were able to decrease the cost of these functions by deploying their application to the main Ethereum network. This ultimately decreased their gas costs to $17.54 USD for deployment, $1.39 USD on their most expensive function and $0.15 USD for their least expensive function. Therefore, with a steady income of money, it might be possible to deploy this type of application on the main Ethereum network instead of a private blockchain. More research must be done on the network latency, increasing costs, and security of deploying to the main network to consider it as a possible fix.

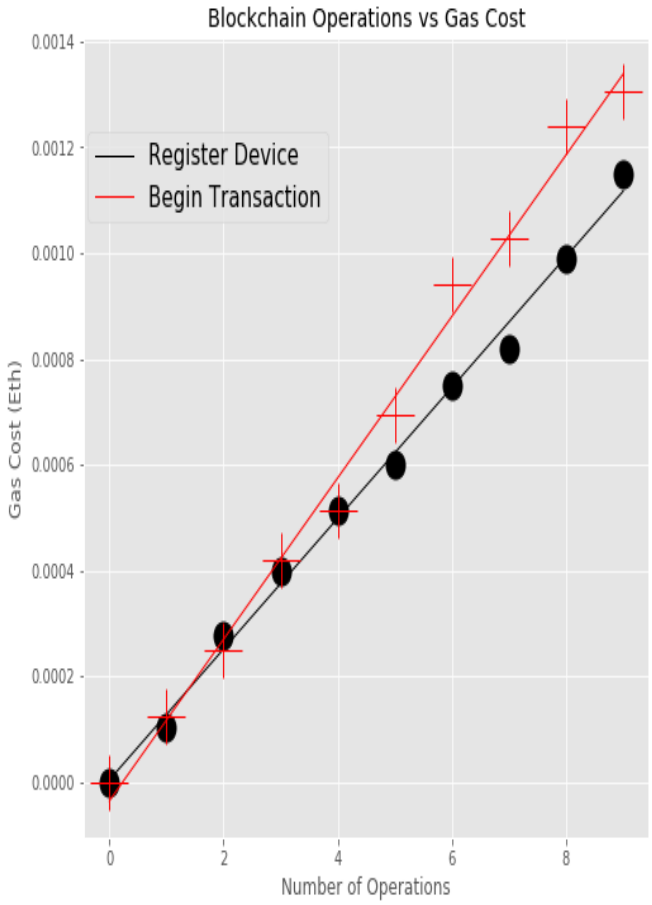

Fig. 7. Gas consumption in comparison to number of devices registered

V. Conclusion and Future Work

In this research, we have studied the usability of the blockchain in an IoT application. We studied that, as embedded systems become more prevalent in the environment around us, they need to communicate with each other autonomously in order to reduce the need for human intervention. However, these M2M communications are not stored in a way to showcase the causal relations between them. This makes it difficult to troubleshoot or find attackers in these systems, as these devices may send malicious requests whose provenance will be buried in a long list of M2M messages. To combat this, we developed M2MHub, a Blockchain-based application which can be used to track the provenance of such messages. M2MHub provides a centralized hub which tracks the provenance of every M2M transaction in the system. If a device sends a message which triggers another message, the original sender will be marked as the provenance, thus allowing users to see the origin of any issues. All messages are then stored in a blockchain.

However, as blockchain operations are resource heavy and relatively slow compared to alternatives like relational databases, the current implementation of M2MHub may not show the current state of the IoT system being monitored. To speed up the operations, two pieces of future work could be done: improving the smart contract and implementing better hardware.

The current smart contract is lengthy and resource-heavy. It can be improved by reducing the number of operations inside the smart contract and transferring those operations the DApps backend. These operations may include functions which do not necessarily operate on the blockchain. For example, hashing and comparing strings could be done outside of the smart contract.

Implementing better hardware is an obvious avenue for expansion, however it can be implemented in a multitude of ways. We could add more computers to the local cluster, or we could deploy the system to a public blockchain with better hardware. Both of these routes could be explored in future work.

REFERENCES

[1] T. hoon Kim, C. Ramos, and S. Mohammed, "Smart city and iot," *Future Generation Computer Systems*, vol. 76, pp. 159 – 162, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17305253.

[2] Y. Qian, Y. Jiang, J. Chen, Y. Zhang, J. Song, M. Zhou, and M. Pustiek, "Towards decentralized iot security enhancement: A blockchain approach," *Computers Electrical Engineering*, vol. 72, pp. 266 – 273, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045790618300508.

[3] M. Batty, K. W. Axhausen, F. Giannotti, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis, and Y. Portugali, "Smart cities of the future," *The European Physical Journal Special Topics*, vol. 214, no. 1, pp. 481–518, Nov 2012. [Online]. Available: https://doi.org/10.1140/epjst/e2012-01703-3.

[4] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[5] A. Barki, A. Bouabdallah, S. Gharout, and J. Traor, "M2m security: Challenges and solutions," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1241–1254, Secondquarter 2016.

[6] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, vol. 82, pp. 395 – 411, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17315765.

[7] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson, "M2m: From mobile to embedded internet," *IEEE Communications Magazine*, vol. 49, no. 4, pp. 36–43, April 2011.

[8] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," *Network and Distributed Systems Symposium*, Feb 2018. [Online]. Available: http://par.nsf.gov/biblio/10047686.

[9] A. Reyna, C. Martn, J. Chen, E. Soler, and M. Daz, "On blockchain and its integration with iot. challenges and opportunities," *Future Generation Computer Systems*, vol. 88, pp. 173 – 190, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17329205.

[10] A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and iot integration: A systematic survey," *Sensors*, vol. 18, no. 8, p. 2575, Aug 2018. [Online]. Available: http://dx.doi.org/10.3390/s18082575.

[11] A. O. J. Kwok and S. G. M. Koh, "Is blockchain technology a watershed for tourism development?" *Current Issues in Tourism*, vol. 0, no. 0, pp. 1–6, 2018. [Online]. Available: https://doi.org/10.1080/13683500.2018.1513460.

[12] V. P. Ranganthan, R. Dantu, A. Paul, P. Mears, and K. Morozov, "A decentralized marketplace application on the ethereum blockchain," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, Oct 2018, pp. 90–97.

[13] X. Xu, Q. Lu, Y. Liu, L. Zhu, H. Yao, and A.V. Vasilakos, "Designing blockchain-based applications a case study for imported product traceability," *Future Generation Computer Systems*, vol. 92, pp. 399 – 406, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18314298.

[14] S. Myung and J.-H. Lee, "Ethereum smart contract-based automated power trading algorithm in a microgrid environment," *The Journal of Supercomputing*, Nov 2018. [Online]. Available: https://doi.org/10.1007/s11227-018-2697-7.

[15] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," 2013. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper.

[16] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Com- puting (CCGRID)*, May 2017, pp. 468–477.