Smart Contracts for Machine-to-Machine Communication: Possibilities and Limitations

Yuichi Hanada

Stanford University
yuhanada@stanford.edu

Luke Hsiao

Stanford University
lwhsiao@stanford.edu

Philip Levis

Stanford University
pal@cs.stanford.edu

Abstract—Blockchain technologies, such as smart contracts, present a unique interface for machine-to-machine communication that provides a secure, append-only record that can be shared without trust and without a central administrator. We study the possibilities and limitations of using smart contracts for machine-to-machine communication by designing, implementing, and evaluating `AGasP`, an application for automated gasoline purchases. We find that using smart contracts allows us to directly address the challenges of transparency, longevity, and trust in IoT applications. However, real-world applications using smart contracts must address their important trade-offs, such as performance, privacy, and the challenge of ensuring they are written correctly.

Index Terms—Internet of Things, IoT, Machine-to-Machine Communication, Blockchain, Smart Contracts, Ethereum

I. INTRODUCTION

The Internet of Things (IoT) refers broadly to interconnected devices that communicate, share data, measure the physical world, and interact with people. IoT applications have been deployed in a wide variety of domains such as healthcare, manufacturing, agriculture, and transportation. They also have the potential to transform daily life through smart homes, cities, and infrastructure [1–3]. Cisco and Ericsson estimate that there will be 100 billion IoT devices by 2020 [4].

Many IoT applications rely on *machine-to-machine communication*—the communication between devices with limited or without human intervention [5]—to automate tasks, send commands, and/or distribute information. Figure 1 shows an example of this broad class of machine-to-machine applications, a smart vehicle automatically paying for its refueling. Machine-to-machine applications encounter three technical challenges that human-centric applications solve with a person in the loop.

The first challenge is transparency. IoT devices are infamously insecure [6], [7], but encryption makes it extremely difficult to verify that they are acting appropriately. In Figure 1, where actions are automatically performed by a smart vehicle on a user's behalf, it would be impossible for the user to audit the encrypted information sent from their vehicle to the cloud service to ensure that private data was not also sent—a concern that is well justified [8], [9].

The second challenge is longevity. IoT devices are often expected to function for decades. Some of these objects may still be in use long after their vendors stop maintaining them [10]. But, because these devices are often *vertical silos*—where a
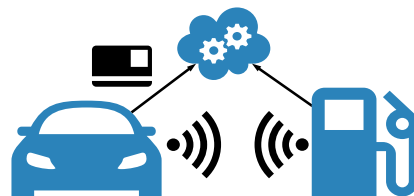

Figure 1. A traditional IoT application that stores a user's credit card information and is installed in a smart vehicle and smart gasoline pump. Before refueling, the vehicle and pump communicate directly using short-range wireless communication, such as Bluetooth, to identify the vehicle and pump involved in the transaction. Then, the credit card stored by the cloud service is charged after the user refuels. Note that each piece of the application is controlled by a single entity.

centralized entity manages application state and communication protocols—they cannot function without the cloud services of their vendors [11], [12]. In Figure 1, without the cloud service, the vehicle is unable to pay the gas station.

The third challenge is trust. In Figure 1, the user must trust the vendor with their credit card information, and the credit card company acts as a trusted third party to help manage funds and resolve disputes in exchange for non-negligible fees. IoT transactions that involve the exchange of digital or physical assets require trust, which inherently involves risks. Examples of such risks are the vendor leaking credit card information or the credit card company undermining the fairness of an exchange by colluding when resolving disputes [13]. Each of these risks can compromise an IoT application.

To summarize these challenges:

1) IoT applications acting on behalf of users benefit from *transparency*. Without it, users cannot verify the actions their applications are taking on their behalf.
2) Long-lived IoT devices may outlive the infrastructure that supports them, causing them to fail or exposing vulnerabilities. They lack *longevity*.
3) IoT applications that exchange goods or services require *trust*. This often adds financial overhead and risks that can subvert an application.

One approach to addressing these challenges is to use blockchain technologies. A blockchain provides a publicly-auditable, append-only ledger that ensures full transparency of transactions performed on the chain. It also allows state to be stored in a distributed manner among the nodes in a network

that persists as long as a network of nodes exists. Furthermore, blockchain-based ledgers that support smart contracts allow applications to embed the contractual logic of a transaction onto the blockchain. This way, the logic is executed independently and automatically by each node on the network using the data provided on the blockchain—reducing the need for trust and third party involvement in a transaction [14].

However, these beneficial properties come with impactful trade-offs: certain blockchain consensus algorithms, such as the proof-of-work used in Ethereum, significantly limit the performance of executing transactions in terms of both throughput and latency; the availability of a public log of transactions raises important privacy considerations; and smart contracts require special care in deployment. Because a smart contract acts as the absolute arbiter for transactions performed through it, bugs in a smart contract may incur costly damages that are difficult or impossible to resolve [15].

Researchers have proposed a variety of uses for blockchain and smart contracts in IoT applications (Section II-D). Our work presents an initial technical and experimental evaluation of this approach. We present AGasP, an IoT application for automated, machine-to-machine gasoline purchases that uses smart contracts to perform transactions. In contrast to the traditional centralized IoT architecture shown in Figure 1, AGasP allows users to audit all interactions and can continue to function as long as the blockchain network exists. Furthermore, by using smart contracts to cut out third parties, AGasP can save gas stations over 79% in fees for a typical transaction.

Contributions This paper makes these contributions:
1) The design, implementation, and evaluation of AGasP, an application that uses smart contracts for automated gasoline purchases between a vehicle and gas station.
2) A discussion of the practical trade-offs of using smart contracts for machine-to-machine communication. We find that transaction latencies are sensitive to transaction fees and that the 95-percentile latency can be on the order of hours (Section VI).

The remainder of the paper is structured as follows. Section II describes the building blocks for using smart contracts. Section III makes the case for smart contracts. Section IV describes AGasP's design and implementation while Section V evaluates its performance and Section VI discusses the limitations of this approach. We conclude in Section VII.

II. BACKGROUND

We describe blockchains and smart contracts abstractly, and then describe how they are implemented in Ethereum, the largest smart contract blockchain used today, to ground these abstract concepts concretely.

A. Blockchain

A *blockchain* is a distributed data structure, introduced with Bitcoin [16], that provides a verifiable, append-only ledger of transactions. As its name suggests, a blockchain is comprised of timestamped blocks, where each block is identified by a cryptographic hash and references the hash of the block preceding it—forming the links, or chain, between each block. In addition, blocks may contain *transactions*, which record a transfer of data or assets between two addresses. As a result, any node in a *blockchain network* with access to the blockchain can traverse it to construct the global state stored on the chain. Nodes in a blockchain network operate on the same blockchain and form a peer-to-peer network where each node replicates all or part of the blockchain.

In order to submit transactions to the chain, each node uses a pair of public and private keys. First, the node constructs and signs a transaction and broadcasts it to its one-hop peers. Each node validates any transactions it receives, dropping invalid transactions, before broadcasting it to its peers. These transactions form a pool of valid, pending transactions that are ready to be included in a block. *Miners* are nodes in the network that construct new blocks to be added to the blockchain and broadcasts them to its peers, who verify it before appending it. This process continuously repeats. Each blockchain employs a consensus mechanism to resolve different states, or "forks" in the network, and the choice of mechanism varies among networks. [17] details a variety of consensus mechanisms.

B. Smart Contracts

Smart contracts "combine protocols, user interfaces, and promises expressed via those interfaces, to formalize and secure relationships over public networks [14]." In other words, smart contracts allow users to execute a script on a blockchain network in a verifiable way and allows many problems to be solved in a way that minimizes the need for trust. To do so, they allow users to place trust directly in the deterministic protocols and promises specified in a smart contract, rather than in a third party. For example, in Figure 1, users must trust the vendor with their credit card number, and the gas station must trust the credit card company to pay on behalf of the user in exchange for fuel. With smart contracts, a user can pay the gas station directly by using the protocol and promises established in the contract such that neither party can manipulate the exchange.

A smart contract has its own address and account on the blockchain. Consequently, it can maintain its own state and take ownership of assets on the blockchain, which allows it to act as an escrow. Smart contracts expose an interface of functions to the network that can be triggered by sending transactions to the smart contract. Because a smart contract resides on the blockchain, each node can view and execute its instructions, as well as see the log of each interaction with each smart contract. A smart contract acts as an autonomous entity on the blockchain that can deterministically execute logic expressed as functions of the data that is provided to it on the blockchain.

C. The Ethereum Platform

Ethereum is an open-source, distributed platform based on a blockchain. In Ethereum's blockchain network, miners execute a *proof-of-work* consensus algorithm [18]. In addition, Ethereum supports smart contracts written in a Turing-complete language, like Solidity, which can be compiled to bytecode that

can be executed in the Ethereum Virtual Machine. This allows users to create arbitrary ownership rules, transaction formats, and state transition functions [19].

In Ethereum, all computation and transactions have fees, which are measured in units of gas (not to be confused with the real-world fuel discussed in our running example). Each transaction in Ethereum must specify a gasLimit, which is the maximum amount of gas that may be used while executing a transaction. Transactions also specify a gasPrice, which is the rate paid to miners in Ether (Ethereum's associated cryptocurrency) per unit of gas as a reward. If an account cannot support the maximum fee of (gasPrice∗gasLimit), the transaction is considered invalid. The amount of gas used (gasUsed) is determined by the amount of computation and storage required by a transaction. A transaction's fee is calculated as shown in Equation 1.

$$Fee = \texttt{gasPrice} * \min(\texttt{gasUsed}, \texttt{gasLimit}) \quad (1)$$

Transactors may specify any positive gasPrice and gasLimit. Likewise, miners may ignore transactions and prioritize transactions with larger fees to maximize their profits. In Ethereum, the maximum number of transactions that can be included in a block is limited by the total amount of gas used by the transactions in the block. Unlike Bitcoin's size-based limit of 1MB, this gas block limit can increase by a small scalability factor each block [18]. These block limits create an upper-bound for transaction throughput of the network if the mining rate of new blocks remains constant.

Relaxing the block size limit to improve throughput is not a simple solution. For example, increasing the block size increases the resources required to run a full mining node, which could lead to centralization of entities with high compute power. While other approaches for improving scalability such as off-chain solutions (e.g., the Raiden Network [20]), periodic merkleized commits [21], and sharding [22] have been proposed, we only consider the implementation of the current Ethereum network for this work.

D. Related Work

Many researchers have explored the application of smart contracts in the IoT domain. For example, smart contracts have been proposed as a mechanism for managing access control [23–25] and authentication [26]. Others have investigated security and privacy implications of using smart contracts in IoT applications [27], explored using smart contracts to create a shared marketplace of services between devices [28], [29], and built industrial IoT platforms using blockchains [30].

III. THE CASE FOR SMART CONTRACTS

We find that smart contracts address the technical challenges of transparency, longevity, and trust that are frequently encountered in IoT applications like Figure 1.

Transparency with Public Logs In Figure 1, the vendor may wish to gather personal information about a user (such as driving behavior, location history, or vehicle mileage) that could
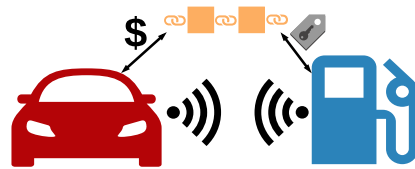

Figure 2. AGasP uses smart contracts in Ethereum to pay for fuel. There is no centralized entity that manages the state of the application. Instead, a vehicle running a decentralized application can interact directly with the public blockchain to submit funds to the smart contract. Likewise, the gas station can interact directly with the blockchain through its application to determine whether a vehicle has paid, and to record how much gas was purchased.

then be sold to advertisers for better advertisement targeting. Users would have no way to directly verify exactly what information was being sent by their vehicle if communication to the cloud service was encrypted. The use of smart contracts as the interface for machine-to-machine communication provides a public, auditable log of communication.

Longevity through Decentralization In 2016, the average age of a car in the United States was 11.6 years [31]—a time frame that may outlast the vendor of our example IoT application. By using smart contracts, the core state and logic of an application is fully distributed, allowing an application to continue to operate or be picked up by a new vendor long after the original vendor has shut down. Because smart contracts are a public interface, anyone can directly interact with the smart contract using their own applications with the assurance that the application's state will be available on the blockchain, rather than being lost with the shutdown of a vendor.

Minimized Trust using Smart Contracts The need to trust vendors or third parties with personal information like credit cards or bank accounts creates valuable targets for attack, many of which have been notoriously exploited [32]. Instead, transactors can avoid this risk by interacting directly through deterministic business logic specified in smart contracts and using smart contracts as reliable escrows for digital assets, creating a platform that supports a wide variety of applications.

IV. AGasP: AUTOMATED GASOLINE PURCHASES

To understand the trade-offs involved with using smart contracts, we design, implement, and evaluate AGasP, a decentralized application for automated gasoline purchases. In contrast to the centralized approach of Figure 1, with AGasP, the state and protocol of the application are not controlled by a single vendor, as shown in Figure 2. Instead, smart contracts on the Ethereum blockchain network are used to purchase gasoline in a verifiable and auditable way that minimizes trust.

A. System Design

In AGasP, the smart contract is the main component since it defines the protocol of a gas purchase in a general way, and stores the state and logic necessary to complete a purchase. Consequently, a single contract can be utilized by many different vehicles and different gas stations—it serves as a common interface and protocol for purchasing gasoline. We designed the

AGasP smart contract protocol to follow a sequence familiar to users from traditional gasoline purchases, while removing both the credit card company and vendor as third parties.

When exchanging digital currency, smart contracts themselves can act as an escrow to ensure that either both parties get the results of the expected exchange, or neither do (e.g., if one party were to cancel). However, in the case of exchanging a physical good, the smart contract cannot act as an escrow.

Instead, the smart contract must clearly specify the sequence of events in order to minimize the risk involved in the exchange. For example, once gasoline is dispensed, it cannot be returned. In order to eliminate the risk of a vehicle refueling and trying to avoid payment, we require that payment occur before a vehicle is allowed to refuel, following the pattern of traditional gasoline payment. Similarly, our contract protects gas stations from malicious users by placing the control of completing an exchange in the power of gas stations (i.e., a vehicle cannot refuel a large amount and then claim only a small amount was taken). Finally, the smart contract itself contains all of the information necessary to calculate the payment to send to the gas station, and the change to return to the vehicle. With this design, the vehicle cannot withdraw its deposit without the gas station's involvement in order to protect against the case where a vehicle attempts to withdraw a deposit while refueling. This places trust in the gas station to report the correct amount of fuel dispensed, as is traditionally done.

In AGasP, a smart contract for gasoline purchases is published to the blockchain. The protocol begins with a gas station publishing a minimum deposit amount, as well as prices for their various types of gasoline. A user then transfers Ether to their vehicle. When the user decides that they would like to refuel, they use the vehicle application to initiate a deposit to the gas station where they plan to refuel. The gas station is then able to view this transaction on the blockchain to identify which vehicles are authorized to fuel at the station. When the vehicle arrives at the pump, a short-range wireless protocol (e.g., Bluetooth) can be used to verify the identity of the vehicle and the gas station, and the fueling can begin. Once fueling is complete, the station sends a transaction indicating the amount and type of fuel dispensed and the smart contract calculates the payment for the gas station, returning the deposit to the vehicle after subtracting the payment (see Figure 3).

B. AGasP Implementation

To fulfill this protocol, the smart contract stores an internal list of gas stations, along with the types of fuel they sell and their respective prices. New stations are added to this list, and prices are adjusted in this list, by calling `setGasInfo`, the function to set the minimum deposit, prices, and types. A vehicle can poll the blockchain for station, type, and price information by calling `getGasInfo`. When a vehicle calls `sendDeposit` to send a deposit, it includes the address of the station so that the smart contract can keep a list of vehicles and their deposits for each station, along with the prices at the time of deposit. After verifying identities, the station can check this list to ensure that a deposit has been made before
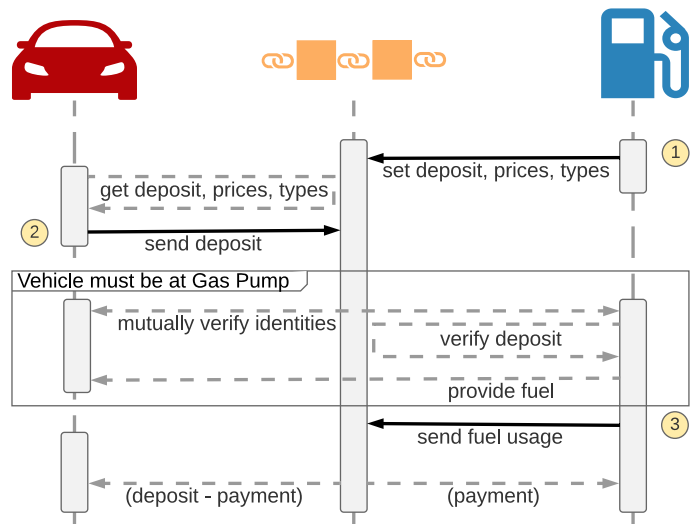

Figure 3. A typical transaction sequence between a vehicle, the AGasP smart contract, and a gas station during a gasoline purchase. Ethereum transactions that require the payment of fees are drawn with a solid black line.

dispensing fuel by calling `verifyDeposit`. Once refueling is complete and the amount of fuel dispensed is reported by calling `sendFuelUsage`, the payment and change are sent, and the vehicle is removed from the list. We implement the AGasP smart contract using Solidity, a high-level language for the Ethereum platform.

In order to interact with the AGasP smart contract, we also developed two decentralized applications—one for the vehicle and one for the gas station—using JavaScript and the web3 Ethereum JavaScript API. Each application is programmed with the address of the AGasP smart contract. The gas station application provides an interface to call `setGasInfo`, `verifyDeposit`, and `sendFuelUsage`. The vehicle application provides an interface for calling `getGasInfo` and `sendDeposit`. These applications simply help us test and develop the AGasP smart contract.

V. EVALUATION

We evaluate AGasP by seeking to answer: (1) does AGasP address the challenges resulting from traditional approaches and (2) how does AGasP compare to traditional approaches in terms of financial overhead to gas stations?

A. AGasP Compared to a Centralized Approach

Transparency As shown in Figure 3, the only operations that do not interact with the blockchain are the mutual verification of identities and the refueling. The rest of the key components of a purchase are committed as blockchain transactions. With this scheme, it is impossible for a vehicle to purchase gasoline without making an auditable deposit on the chain to a specific gas station, and it is impossible for a gas station to acquire their payment without publishing the amount of fuel that was dispensed. Similarly, it is impossible for a gas station to charge a price other than what was committed to the blockchain at
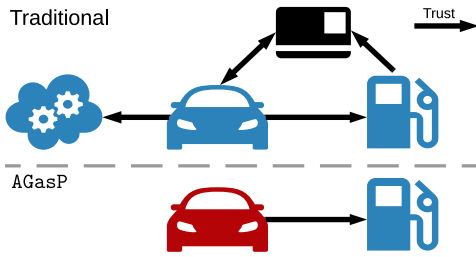
Figure 4. Smart contracts allow applications to minimize trust.

Transaction	Paying Party	Approx. Fee ($)
Credit Card	Gas Station	1.17
① setGasInfo	Gas Station	0.21
② sendDeposit	Vehicle	0.32
③ sendFuelUsage	Gas Station	**0.25**
①, ②, ③	Vehicle, Gas Station	0.78

the time of purchase, since the smart contract performs the calculation based on the published prices.

Longevity In an architecture like Figure 1, the application relies on the vendor's infrastructure and cloud services to operate. With `AGasP`, the infrastructure is completely distributed; the permanence of the infrastructure and state of the application is based on the permanence of the Ethereum network. Although Ethereum is still relatively nascent, because it is open-source, anyone can set up new Ethereum nodes to ensure that applications persist. There are threats to the longevity of using a public, proof-of-work-based blockchain like Ethereum, such as a malicious entity controlling 51% of a network's mining power, allowing them to manipulate the ledger. However, the high cost of such an attack on large public networks makes it unlikely [33], and moving away from proof-of-work is already on the Ethereum roadmap [34].

Trust In Figure 1, there are several relationships of trust leveraged to complete a transaction. The user must trust: the vendor with their credit card, the credit card to pay the station, and the station to give them fuel and charge the correct amount. Next, the credit card company trusts the user to pay back their debt. Finally, the station trusts the credit card company to pay on behalf of the user. By leveraging a smart contract, we can reduce the trust to just a single edge (see Figure 4): a user must trust the gas station to give them fuel and charge the correct amount[1]. The smart contract itself will then distribute payment without the involvement of another party.

B. Transaction Fees

In Figure 1, the gas station must pay fees to the credit card company, and many gas stations have passed those fees onto users by charging more for credit card transactions than for debit or cash [36]. Similarly, with `AGasP`, both the user and the gas station must pay transaction fees in order for their transactions to be included on the blockchain. In order to estimate these savings, we estimate the amount of fees paid for a refueling a small car. Credit card companies charge a fee comprised of a rate ($Rate$) of the transaction amount ($Amount$)

[1]While multi-signature transactions can be used to distribute the trust among several parties (i.e., requiring m of n participants to validate a transaction), ultimately, at least one point of trust is required when exchanging physical goods or services [35].

along with a flat fee ($Flat$) for each transaction, as shown in Equation 2.

$$Fee = Rate * Amount + Flat \qquad (2)$$

Assuming a typical rate of 2% (less than the 2016 average [37]), a transaction amount of filling a 12 gallon tank with gas at $3.85 per gallon, and a flat fee of $0.25, we estimate a fee of $1.17. Similarly, we can compute an estimate of the transaction fees incurred by `AGasP` using Equation 1. If we assume a `gasPrice` of 10×10^{-9} Ether per transaction, and an Ether value of $650. Then, based on our experimental measurements of `gasUsed` for transactions ①, ② and ③ (as labeled in Figure 3), the total cost in transaction fees for `AGasP` would be $0.78, a 33% reduction in transaction fees. Furthermore, note that a gas station only needs to send ① when gas prices change, and ② is paid for by the vehicle. Thus, for a typical refueling, the gas station only pays $0.25—a 79% reduction in transaction fees. These estimates are summarized in Table I.

However, unlike credit card fees controlled by a credit card company, the transaction fees of operating on Ethereum vary with the market and can be unpredictable. For example, Ether was valued at over $1300 in January 2018, and dipped back to under $380 in April 2018—a span of just three months [38].

VI. LIMITATIONS AND DISCUSSION

A. Performance

A noticeable limitation of using smart contracts on Ethereum for machine-to-machine communication is that low transaction throughput results in high latencies in the time that it takes to complete a transaction. In Figure 1, a cloud service using a distributed database such as Google Spanner [39] can perform tens of thousands of transactions per second. In stark contrast, the public Ethereum network's peak throughput was just short of 16 transactions per second [40]. Furthermore, because miners are incentivized to prioritize transactions with higher rewards, transaction time can be highly influenced by the `gasPrice` of the transaction.

In Figure 5, we vary the `gasPrice` of a transaction while holding all other variables constant and measure the time it takes from sending the transaction from a node on the public network, to seeing that transaction executed and included in a block on the chain. Each `gasPrice` was tested five times. A higher `gasPrice` helps reduce both the average transaction time and the variance of transaction times, down to an average of 14 seconds with a `gasPrice` of 64×10^{-9} Ether. However,
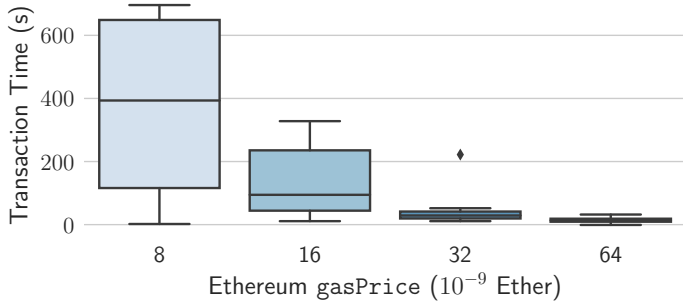
Figure 5. Transaction times for increasing `gasPrice` on the public Ethereum network with a `gasLimit` of 70,000.

Figure 6. Cumulative density function for the transaction times of `setGasInfo` on the public Ethereum network with a `gasPrice` of 10×10^{-9} Ether and a `gasLimit` of 70,000.

we see diminishing returns as eventually we are bottlenecked by the throughput of the Ethereum network. In Figure 6, we use a constant `gasPrice` and `gasLimit`, and run the same transaction 60 times over the span of 5 days. The load of the network and value of `gas` varies over time, leading to high tail latencies when using a constant `gasPrice`. In our experiments, we measure a mean latency of 14 minutes and a 95-percentile latency of 69 minutes.

The relatively low performance of smart contracts on Ethereum means that applications that utilize them must design around the possibility of transaction delays on the order of minutes to hours, or be prepared to raise `gasPrice` values to lower transaction times (down to the order of tens of seconds). In `AGasP`, we design our sequence such that transaction latency can be hidden from the user by allowing the time-consuming smart contract transactions to occur before and after the user is interacting with the gas pump (Figure 3). If a user does not pay beforehand, they must wait one transaction time at the pump. Other blockchain networks, such as Stellar, address performance by using alternative consensus algorithms [41].

B. Privacy

Recall that transactions contain an origin address, a destination address, and the data or assets to be sent. Although using smart contracts enables users to audit the transactions of their devices, the nature of a public blockchain also means that anyone else can also view these transactions. Even though addresses are not explicitly tied to a real-world identity, other nodes are still able to monitor a blockchain to learn patterns about a user's transactions.

Techniques such as *zero-knowledge Succinct Non-interactive ARguments of Knowledge* (zkSNARK) [42] have been deployed successfully in the Bitcoin network and extend the Bitcoin protocol by adding a new type of transaction that hides the origin, destination, and transferred amount from the public [43]. While a zkSNARK approach has recently been tested on the Ethereum Byzantium test network, the transaction required two orders of magnitude more `gas` to complete, making transaction fees a potential barrier [44]. Other work, such as Hawk [45], explores methods for creating privacy-preserving smart contracts that do not store financial transactions in the clear in order to retain transactional privacy from the public.
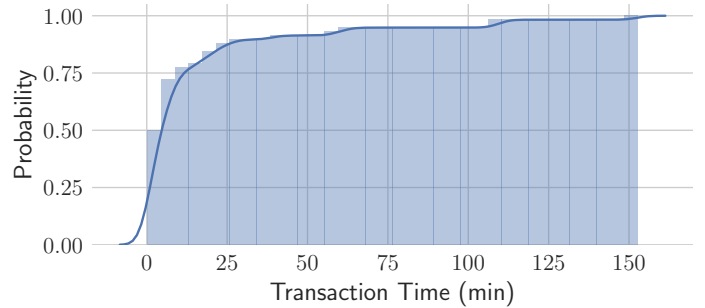
C. Impact of Bugs in Smart Contracts

In contrast to traditional software, smart contracts cannot be directly patched once deployed. This brings a unique set of challenges and considerations for designing smart contracts [46]. Two common approaches for updating smart contracts are to either (1) use a self-destruct function that releases all the internal state of the contract—typically by sending all funds to a particular address—and then publish a new one, or (2) include a version flag and a mutable pointer to the address of a new contract after a contract is deprecated. If a smart contract contains critical flaws, such as logical errors and lack a self-destruct, assets can be locked in a contract. The most noticeable example of this was the flaw in The Distributed Autonomous Organization (The DAO) smart contracts that allowed an attacker to siphon over $50M USD out of the $168M funds invested in the organization. Ultimately, this required a highly controversial "hard fork" to return the state of the blockchain to a state prior to the hack. Smart contracts may have vulnerabilities at the language, bytecode, and blockchain levels that open them to a wide variety of attacks as seen with The DAO, Rubixi, GovernMental, and others. [47] provides a taxonomy and details of these exploits.

VII. CONCLUSION

IoT applications that automatically perform tasks or exchange assets in behalf of users pose unique design challenges in terms of: a desire for transparency of the actions that a device takes on behalf of user; the longevity of these devices compared to traditional software products; and the common use case of exchanging digital or physical goods or services, which requires a trusted arbiter. We make a case for using smart contracts to address these challenges and take a first look at their trade-offs by designing, implementing, and evaluating `AGasP`, an IoT application for automated gasoline purchases using machine-to-machine communication. We find that the use of smart contracts provides transparency, longevity, and allows applications to minimize the need for trusted third parties—which we estimate can reduce fees paid by a gas station for a typical transaction by 79%. However, Ethereum smart contracts have low transaction throughput (tens of transactions per second) and 95-percentile transaction latency can be on the order of hours, limiting the types of applications that can be supported.

REFERENCES

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE IOT-J*, vol. 1, no. 1, pp. 22–32, 2014.

[2] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, and C.-H. Lung, "Smart home: Integrating internet of things with web services and cloud computing," in *2013 IEEE CloudCom*, vol. 2. IEEE, Dec 2013, pp. 317–320.

[3] Y. Zhang, R. Yu, M. Nekovee, Y. Liu, S. Xie, and S. Gjessing, "Cognitive machine-to-machine communications: visions and potentials for the smart grid," *IEEE Network*, vol. 26, no. 3, pp. 6–13, May 2012.

[4] M. Weyrich, J.-P. Schmidt, and C. Ebert, "Machine-to-machine communication," *IEEE Software*, vol. 31, no. 4, pp. 19–23, July 2014.

[5] M. Chen, J. Wan, and F. Li, "Machine-to-machine communications: Architectures, standards and applications," *KSII TIIS*, vol. 6, no. 2, Feb 2012.

[6] S. Grover and N. Feamster, "The internet of unpatched things," in *FTC PrivacyCon*, Jan 2016.

[7] Hewlett Packard Enterprise, "Internet of things research study," 2015.

[8] J. Angwin. (2015, Nov) Own a vizio smart tv? it's watching you. [Online]. Available: https://www.propublica.org/article/own-a-vizio-smart-tv-its-watching-you

[9] P. Oltermann. (2017, Feb) German parents told to destroy doll that can spy on children. [Online]. Available: https://www.theguardian.com/world/2017/feb/17/german-parents-told-to-destroy-my-friend-cayla-doll-spy-on-children

[10] W. Hartzog and E. Selinger, "The internet of heirlooms and disposable things," *North Carolina JOTL*, vol. 17, no. 4, pp. 581–598, May 2015.

[11] A. Hern. (2016, Apr) Revolv devices bricked as google's nest shuts down smart home company. [Online]. Available: https://www.theguardian.com/technology/2016/apr/05/revolv-devices-bricked-google-nest-smart-home

[12] S. Gallagher. (2016, Dec) Internet of $@!%: Google api change triggers epson printer revolt. [Online]. Available: https://arstechnica.com/information-technology/2016/12/internet-of-google-api-change-triggers-epson-printer-revolt/

[13] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan, "Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin," in *ICFCDS*. Springer, Apr 2017, pp. 321–339.

[14] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, Sep 1997.

[15] D. Siegel. (2016, Jun) Understanding the dao attack. [Online]. Available: https://www.coindesk.com/understanding-dao-hack-journalists

[16] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[17] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 3, pp. 2084–2123, Mar 2016.

[18] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project, Tech. Rep. EIP-150, 2017.

[19] V. Buterin, "A next-generation smart contract and decentralized application platform," Ethereum Project, Tech. Rep., 2014.

[20] Raiden network. [Online]. Available: https://raiden.network/

[21] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," 2017.

[22] Ethereum. On sharding blockchains. [Online]. Available: https://github.com/ethereum/wiki/wiki/Sharding-FAQs

[23] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *arXiv preprint arXiv:1802.04410*, Feb 2018.

[24] R. Xu, Y. Chen, E. Blasch, and G. Chen, "Blendcac: A blockchain-enabled decentralized capability-based access control for iots," *arXiv preprint arXiv:1804.09267*, Apr 2018.

[25] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," *IEEE IOT-J*, vol. 5, pp. 1184–1195, Mar 2018.

[26] C. H. Lee and K.-H. Kim, "Implementation of iot system using block chain with authentication and data protection," in *International Conference on Information Networking*, 2018, pp. 936–940.

[27] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *2017 IEEE PerCom*, Mar 2017, pp. 618–623.

[28] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[29] S. Huckle, R. Bhattacharya, M. White, and N. Beloff, "Internet of things, blockchain and shared economy applications," *Procedia Computer Science*, vol. 98, pp. 461–466, 2016.

[30] A. Bahga and V. Madisetti, "Blockchain platform for industrial internet of things," *JSEA*, vol. 9, no. 10, pp. 533–546, Oct 2016.

[31] IHS Markit. (2016, Nov) Vehicles getting older: Average age of light cars and trucks in u.s. rises again in 2016 to 11.6 years, ihs markit says. [Online]. Available: http://news.ihsmarkit.com/press-release/automotive/vehicles-getting-older-average-age-light-cars-and-trucks-us-rises-again-201

[32] S. O'Brien. (2017, Sep) Giant equifax data breach: 143 million people could be affected. [Online]. Available: https://money.cnn.com/2017/09/07/technology/business/equifax-data-breach/

[33] D. Cawrey. (2014, Jun) Are 51% attacks a real threat to bitcoin? [Online]. Available: https://www.coindesk.com/51-attacks-real-threat-bitcoin/

[34] A. Hertig. (2017, May) Ethereum's big switch: The new roadmap to proof-of-stake. [Online]. Available: https://www.coindesk.com/ethereums-big-switch-the-new-roadmap-to-proof-of-stake/

[35] H. Pagnia and F. C. Gärtner, "On the impossibility of fair exchange without a trusted third party," Darmstadt University of Technology, Department of Computer Science, Tech. Rep. TUD-BS-1999-02, 1999.

[36] J. Simon. (2008, Jul) Gas station owners challenge credit card fees. [Online]. Available: https://blogs.creditcards.com/2008/07/gas-station-credit-card-fees.php

[37] "Merchange processing fees in the u.s.—2016," *The Nilson Report*, May 2017.

[38] Etherscan. (2018) Ether historical prices (usd). [Online]. Available: https://etherscan.io/chart/etherprice

[39] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems*, vol. 31, no. 3, pp. 8:1–8:22, Aug 2013.

[40] Etherscan. (2018) Ethereum transaction chart. [Online]. Available: https://etherscan.io/chart/tx

[41] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," Tech. Rep., Feb 2016.

[42] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *USENIX Security Symposium*, Aug 2014, pp. 781–796.

[43] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE SP*, May 2014, pp. 459–474.

[44] R. O'Leary. (2017, Sep) Ethereum's byzantium testnet just verified a private transaction. [Online]. Available: https://www.coindesk.com/ethereums-byzantium-testnet-just-verified-private-transaction/

[45] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE SP*, May 2016, pp. 839–858.

[46] S. Porru, A. Pinna, M. Marchesi, and R. Tonelli, "Blockchain-oriented software engineering: challenges and new directions," in *ICSE*. IEEE Press, May 2017, pp. 169–171.

[47] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *POST*. Springer, 2017, pp. 164–186.