Received: 23 December 2021 | Revised: 9 July 2022 | Accepted: 4 August 2022

DOI: 10.1002/eng2.12567

RESEARCH ARTICLE

Engineering Reports Open Access WILEY

Machine-as-a-Service: Blockchain-based management and maintenance of industrial appliances

Viet Hoang Tran[1] | **Bernard Lenssens**[2] | **Ayham Kassab**[1] | **Alexis Laks**[2] | **Etienne Rivière**[1] | **Guillaume Rosinosky**[1] | **Ramin Sadre**[1]

[1]ICTEAM, UCLouvain, Ottignies-Louvain-la-Neuve, Belgium

[2]Proximus AG, Codit, Ghent, Belgium

Correspondence
Bernard Lenssens, Proximus AG, Codit, Ghent, Belgium.
Email: Bernard.Lenssens@codit.eu
Etienne Rivière, ICTEAM, UCLouvain, Ottignies-Louvain-la-Neuve, Belgium.
Email: etienne.riviere@uclouvain.be

Funding information
Fonds De La Recherche Scientifique - FNRS, Grant/Award Number: F452819F; Innoviris, Grant/Award Number: FairBCaaS

Abstract

Machine-as-a-Service (MaaS) is an emerging service model for industrial appliances. With MaaS, machines are rented instead of being acquired, and their lifecycle is handled by an ecosystem of specialized actors, such as different independent maintenance companies certified for interventions on specific hardware. As the number of actors, clients, and providers involved in a MaaS ecosystem grows, maintaining mutual trust relationships between all involved parties and orchestrating MaaS operations in centralized fashion quickly becomes intractable. We present a blockchain-based approach to providing MaaS in industrial settings where rented machines are equipped with IoT sensors, and where MaaS operations are orchestrated in a transparent, decentralized, and scalable way using a collection of smart contracts deployed over an infrastructure combining the Ethereum and InterPlanetary File System decentralized services. We detail the operations of MaaS, such as the lifecycle of management operations, and report on the performance of a prototype implementation deployed in the cloud.

KEYWORDS

blockchain, industry, IoT, Machine-as-a-Service, telemetry

JEL CLASSIFICATION

Computer and software engineeringε cd_value_code=εENG06

1 | INTRODUCTION

A typical factory contains many specialized machines and devices produced by original equipment manufacturers (OEMs). The maintenance and management of these machines by factory owners themselves is cumbersome as these owners typically prefer to focus on their primary business. Traditionally, OEMs have followed a capital expenditures (CapEx) business model, meaning that they would sell their product for an upfront price and offer a support policy for a fixed number of years. Although CapEx is a widely practiced model, the high upfront cost of equipment purchase can deter potential customers[1] who only want to use a machine for a short or unknown duration. The CapEx model might, furthermore, slow down the expansion of OEMs to new markets as an increase in market share is immediately associated with the need to build and operate a large, and more geographically spread, service department.

An ongoing trend in industrial businesses is to switch from the CapEx model to an OpEx, that is, *operating expenses* model,[2] where machines are rented and paid for on the basis of their usage, for example based on the number of units produced by the machine during the billing period. In that way, the product is accessible to more customers since it requires no or little upfront investment. By allowing third-party companies to service or repair machines, OEMs can offload the maintenance and repair service work. Service cost can, as a consequence, be reduced through competition and specialization. Customers can, in the end, be guaranteed a higher level of service (i.e., as expressed by service level agreement, or SLA, signed with the provider) while OEMs are in a better position to further and faster expand their market shares.

The main obstacle to the deployment of the OpEx model is to scale it up to large installations and ecosystems, and establishing trust between all involved actors.[3] The usage of rented machines has to be logged in such a way that both OEMs and customers be able to verify the correctness of the accounting. In addition, if a machine failure occurs it is important to determine the root cause of that failure in order to identify which actor will have to cover the repair costs. Finally, customers should be able to involve third-party maintenance companies of their choice. When the numbers of parties and machines are small, the different partners know each other well and trust is relatively easy to establish. This becomes a challenge, however, when the system and the number of involved parties grow. There is a risk in that case that the necessary management overhead for establishing pair-wise trust between parties offsets the advantages of the OpEx model.

In this article, we address the question of what a system should look like that allows deploying industrial appliances under the above OpEx model. In particular, we are interested in the system's ability to integrate third-party maintenance companies in an automated process where maintainers can bid for repair operations on failed machines and where customers can select the best maintainer among them according to their own criteria. We believe that the combination of Internet of Things (IoT), blockchain technologies, and decentralized storage are a sound basis to solve the challenges of trust and traceability in such a model. However, when blockchain technologies are introduced into a system design, the latency and costs of on-chain operations have to be considered,[4,5] as well as constraints related to the programming model of on-chain smart contracts.[6]

We propose a Machine-as-a-Service (MaaS) system that leverages IoT, blockchains, and decentralized storage to continuously monitor the rented equipment and store monitoring data in a trusted way, and that allows to manage the operational lifecycle of the appliance, in particular maintenance (by the maintainers) and accounting (by the OEMs). In short, we use these technologies as follows:

- IoT-based telemetry allows customers and vendors to precisely measure *Overall Equipment Effectiveness (OEE)*[7] in production, namely the availability, the performance, and the quality produced by the machines. Typical telemetry data includes the operating condition information of the machine (e.g., temperature), the operational status of the machine, the quantity of items produced, as well as information on normal and abnormal events (e.g., failures). This data further provides important insights for maximizing the effectiveness of machine usage.
- A blockchain is a distributed ledger allowing actors who do not trust each other to share common data and agree on an ordered set of transactions, without relying on a central trusted party.[8,9] We use a blockchain to build a platform where, among others, telemetry and billing data can be stored and queried, and where among other OpEx workflows customers can interact with maintenance companies through a transparent ticket and bidding mechanism implemented with smart contracts. The blockchain we use, Ethereum,[9] is permissionless and while our current implementation considers a private deployment of this system, our design is directly usable over public ones as well.
- The telemetry data is not stored on the blockchain itself for scalability and performance reasons, since collecting and storing the data at typical monitoring intervals, for example, every 15 min, would saturate the transaction processing capability of current blockchain systems. Instead, data is stored off-chain on InterPlanetary File System (IPFS) storage nodes while leveraging the blockchain for integrity and validity checking. IPFS[10,11] is a decentralized storage system that allows to store immutable data objects and operates in a peer-to-peer fashion.

The idea of using ICT technology to computerize and automate not only the monitoring and control of machines, but also the management of their life cycle (including maintenance), attracted the interest of researchers and industry early on. For example, Gilart-Iglesias et al.[12] proposed to augment the physical machines by software interfaces such that they can be integrated into existing ICT systems and be configured, controlled, and diagnosed like other ICT services. However, unlike our work, such earlier approaches did not provide an actual implementation and ignored the challenge

of establishing the level of trust necessary in multi-party scenarios. Similar to us, newer attempts to tackle this problem have made use of blockchains.[13] However, as we will see in more detail when discussing related work later in this article, they mostly focus on scenarios different from ours, in particular the scenario of *Manufacturing-as-a-Service* where product developers buy production time in factories owned and operated by others. In that scenario the machine is still property of the factory owner and, therefore, several of the aspects of the OpEx model that we address in our solution for MaaS, such as the transparent handling of failures and the selection of maintenance companies through bidding, do not have a high priority or do not appear at all. Nevertheless, the idea to use blockchain-technology for MaaS platforms has been advertised before,[14] but without providing technical details or a concrete system design.

1.1 | Contributions

Our contributions in this article can be summarized as follows:

- We analyze the system and operational model of a large-scale set of actors implementing the OpEX model, and define a model for MaaS;
- We detail the trust and security requirements associated with this model, and their implications on the requirements and constraints for an implementation of the MaaS model;
- We provide the first reference architecture for MaaS building upon decentralized, zero-trust technologies that are distributed ledgers (blockchains), smart contracts, and decentralized storage services;

In the following sections, we will focus on the description of our MaaS architecture and its prototype implementation. To address question about latency and costs of on-chain operations, we will show evaluation results from a deployment of the prototype in a representative geo-distributed scenario achieved by network emulation. Note that the design and implementation of IoT devices for appliance monitoring is not subject of this article and will be therefore not further discussed.

1.2 | Outline

We first provide in Section 2 some background information on the Ethereum blockchain and other technologies used for implementing MaaS. We then proceed to describe our contributions as follows. We outline the general principles of the MaaS service model, and detail the corresponding system and trust models in Section 3. We present the detailed design of a MaaS solution combining IoT telemetry and distributed applications deployed over Ethereum in Section 4. We provide additional elements about the implementation of MaaS in Section 5. In Section 6, we finally present the evaluation of the performance and effectiveness of our MaaS prototype. We discuss related work in Section 7, and conclude in Section 8.

2 | BACKGROUND

We start by providing background information about blockchain technologies in general, and Ethereum in particular.

A blockchain is a distributed ledger allowing different parties to share common data and agree on an ordered set of transactions, without having to trust each other or rely on a central trusted party. Transactions are grouped together into blocks for better performance before being appended into the permanent shared record. These blocks are "chained" together: each block holds a cryptographically-secure reference to the previous block in the chain. A blockchain provides, by design, data traceability, transaction verifiability, and immutability. These guarantees are achieved by consensus-based validation by a collection of distributed peers. In open blockchains, such as bitcoin[8] or Ethereum,[9] these distributed peers are termed *miners*, and are incentivized to follow the protocol through rewards associated with the execution of transactions and the generation of new blocks. This generation is the result of an open consensus protocol resisting Sybil attacks (i.e., use of multiple identities by the same miner). Such a consensus protocol can be based on the a proof of computational work (proof-of-work), proof of financial interest in the cryptocurrency implemented by the chain (proof-of-stake), or proof of their identity and reputation (proof-of-authority).

Blockchain technologies have applications that go far beyond their initial use for cryptocurrencies. Through improved transparency and the lack of need to establish pair-wise trust relationship with all parties in complex or large organizations, distributed ledgers have found use in a variety of settings. Examples include (1) *supply chain management*[15] in food production, container shipping, pharmaceuticals, minings and so forth, (2) *the enforcement of business contracts*, typically by monitoring and enforcing SLA of industrial equipment,[16,17] or (3) *electronic-voting*.[18] A foundational concept for these applications to operate is that of a *smart contract*, which is essentially a computer program whose execution results from an (implicit) agreement between all participants. The term *smart contract* was invented by Nick Szabo,[19] even before the era of blockchains. In the context of blockchains, a smart contract consists of code that is stored on the chain at a specific address, and whose execution can be verified by all parties. Smart contracts operations (or methods) may be called in a decentralized fashion as part of transactions' execution and operate on the current state of the chain resulting from the execution of all previous transactions. As this state and the smart contract code are public (on-chain) the execution can be made part of the consensus and verified by any party in the system.

The following sections give a brief overview of Ethereum and the IPFS, the blockchain system and the accompanying off-chain storage that we use in this article.

2.1 | Overview of Ethereum

Ethereum is a blockchain platform which was first proposed by Vitalik Buterin in 2013.[9] The platform is developed and maintained by an active open-source community. In contrast to earlier blockchains that focuses solely in cryptocurrency operations, such as bitcoin,[8] Ethereum has been designed to support a wide range of distributed and decentralized applications, such as supply chain management,[20] or decentralized autonomous organizations (DAOs).[21]

Ethereum enables to register code to the ledger in the form of *smart contracts* as previously discussed. A key feature of Ethereum is that these smart contracts code can implement arbitrary functions (in other words, transaction execution in Ethereum is Turing complete). These transactions can chain-call other smart contract functions, and the execution of all functions depend on the state of the ledger at the time of their execution. Ethereum's built-in cryptocurrency is the *ether*; every participant in an Ethereum network possesses an account associated with an amount of ethers. This cryptocurrency is used as the base currency for applications such as decentralized finance (DeFi)[22] or non-fungible tokens (NFTs),[23] and for Ethereum's internal operations.

An Ethereum blockchain network is composed of multiple computers or Ethereum *nodes*. These nodes run the Ethereum software (called *client*) and connect user applications to the Ethereum blockchain via a well-defined API.[24] Ethereum clients play the role of miners. They receive and propagate transactions to other clients using a decentralized gossip-based protocol. Each client executes transactions from the pool of pending transactions (*mempool*) in order to fill a new block, which they subsequently attempt to append to the existing chain through consensus. The incentive for clients to form new blocks and append them to the blockchain is monetary: a miner chosen by the consensus to append a (valid) block receives rewards associated with the execution of all transactions in that block.

The consensus protocol used in the current *public* version of Ethereum (v1) is proof-of-work, as for bitcoin. The probability for a miner to "win" the computationally-intensive challenge necessary to form a valid block depends directly on its computing power. Running multiple nodes instead of one over a given hardware capacity does not, therefore, bring benefits to a miner, making the protocol resistant to Sybil attacks. While providing strong decentralization, simple operation, and robustness, proof-of-work is energy-inefficient, offers low throughput, and requires significant resources.[25,26] Alternative protocols attempt to mitigate these issues. With proof-of-stake,[27] the selection of the miner allowed to append a new block directly depends (i.e., is proportional) to the amounts of cryptocurrency *staked* (owned and locked) by each miner. Proof-of-stake is founded on the belief that miners with investments have interest in the good functioning of the blockchain, but is sometimes criticized for the risk of centralization in the hands of a few "rich" miners that it entails. Proof-of-authority is, finally, a model where a set of *authorities* miners are registered onto the blockchain and associated with a reputation (in other words, they *stake* their identity). It is in the interest of such miners to maintain a good reputation by following the protocol, as their future selection is based on their reputation value. Proof-of-authority is well-adapted to smaller-scale and/or private Ethereum installations where the risk of Sybil miners is present but where the anonymity of the miners is not a primary concern.

Smart contracts in Ethereum are written in a high-level language, such as the object-oriented Solidity.[28] The code compiles to lower-level bytecode for execution onto the Ethereum virtual machine (EVM). Contracts are registered to specific addresses, much like user-owned accounts, and their methods can be invoked in the context of transactions.

As the EVM is a Turing-complete execution environment, the execution of such methods may take arbitrary time. Ethereum limits the impact of long-running function by associating each instruction in the EVM with an ether cost, and requiring transactions to set an upper limit on the cumulative costs their execution can require. The cumulative cost of executing a transaction in Ethereum is termed the *gas*: more complex transactions (i.e., with more loops or accessing more state) typically result in higher gas costs than simpler transactions. The gas price is paid to the miner who successfully appends a valid block containing that transaction to the ledger, even if the execution had to stop for reaching the set limit.

2.2 | InterPlanetary File System (IPFS)

Storing data on-chain in Ethereum has a cost, associated with both the additional on-chain state and the execution of the smart contract method registering that data. While acceptable for small (or highly-valuable) data, on-chain storage is ill-suited for storing large amounts of records, such as IoT telemetry data. It is preferable to use for such data an *off-chain* storage solution, while leveraging the blockchain support for integrity and validity checking (e.g., by storing aggregates such as compound hashes onto the chain).

IPFS[10,11] is a decentralized off-chain storage system initially designed for the FileCoin Ethereum-based application. IPFS is an object store: It allows storing *immutable* data objects (i.e., an update requires storing a new version of the object). It operates in a peer-to-peer fashion, disseminating and replicating objects across participating storage nodes to ensure availability. IPFS storage nodes locally cache objects and connect to other nodes using direct interactions. Objects in IPFS are identified by a content identifier (CID), which is derived from the content of data itself by a cryptographic hashing function. A distributed hash table combined with a local gossip protocol allows locating peers and objects using their CID. Each object is, furthermore, cryptographically linked to existing other objects in the store, forming a generalized Merkle Directed Acyclic Graph (MDAG).[29] This data structure allows checking efficiently the integrity and validity of retrieved objects, making the replicated storage tamper-proof. The root of the MDAG can further be published onto the blockchain, providing time-stamping capabilities to the store (i.e., the ability to establish the publication date or period of an object in the store). As a result, IPFS is positioned as a natural choice for combining off-chain, relatively low-cost storage with on-chain guarantees.

3 | OVERVIEW, SYSTEM, AND TRUST MODELS

In this section, we describe the target organizational system model from a high abstraction level, and discuss trust relationships (or lack thereof) between involved actors. On the basis of this model and these assumptions, we will present the design of our solution in Sections 4 and its implementation in Section 5.

3.1 | System model and actors

We start by delineating the actors and interactions in our MaaS service model, where factory owners rent machines from their manufacturers in order to avoid the upfront cost of equipment purchase and maintenance.

We considered three types of actors in our MaaS ecosystem (Figure 1): *OEMs*, *customers*, and *maintainers*. OEMs are the providers of industrial machines to customers. Their goal is to gain revenue from renting operational machines. *Customers* use these machines in their business, typically to produce goods or assemble parts. They want to maximize the working time and efficiency of the rented equipment in production. They typically have, for this purpose, service-level agreements (SLAs) in place with the OEMs. If a machine has to stop its operation due to an unplanned technical problem, and for a longer duration than a maximum specified in the SLA, the corresponding OEM may have to compensate production loss to the customer. Conversely, if the machines outperforms the expected working time or productivity, OEM could be awarded a bonus by the customer. OEMs may not have the capacity to operate preventive inspections and maintenance at the geographical scale of their customer base, and/or may not wish to engage in direct business-to-business interactions with customers for various reasons. A third role is, therefore, that of a *maintainer*. It represents a maintenance company validated by the OEM for providing maintenance and repair services for their customers in different regions or countries. In each region, there may be multiple maintenance companies competing to serve the same set of customers.

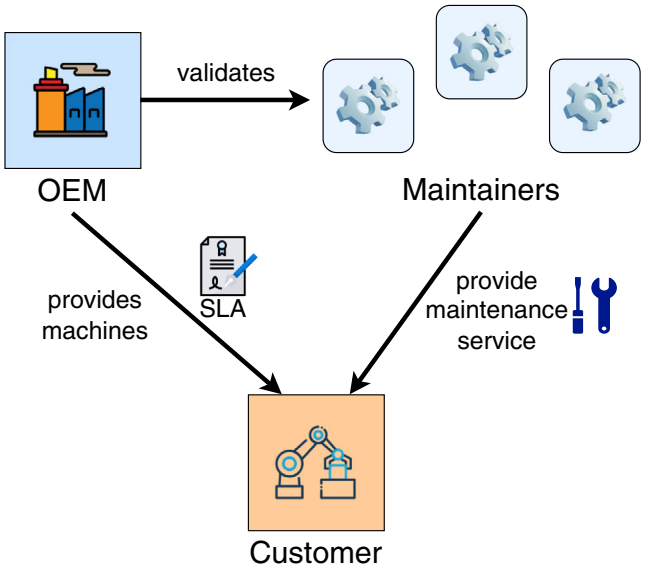


FIGURE 1 Actors in the MaaS service model and their interactions.

3.2 | Trust model and adversarial behaviors

Actors in our MaaS ecosystem have possibly conflicting economical interests: customers may wish to avoid paying premiums to an OEM whose machine has over-performed, maintenance companies may wish to block access to service requests to competitors in the same territory, and OEMs may attempt to avoid paying compensations planned in SLAs for machines' downtimes. As a result, the different actors have to trust each other for not behaving selfishly. In a small system, this may require contractual obligations between every possible pair of interacting parties in the system. As the system grows, for example, by the addition of additional maintainers in a given geographical region, setting up such peer-to-peer trust agreements quickly becomes intractable. In addition, contractual peer-to-peer trust is difficult to enforce and is generally contested *a posteriori*, possibly escalating up to the arbitration of a court—a solution that, obviously, does not scale well.

Our assumption of individual actors selfishly willing to maximize their profits is similar to the ones made in other works with similar settings.[30,31] For example, a key challenge of SLA enforcement between a customer and an OEM is to be able to correctly determine a downtime duration, its impact on production loss, and its root cause. An OEM and a customer may have a dispute on whether a machine shutdown was due to a technical failure or a user error. The customers may further question the cost and outcome of failure diagnosis by maintainers. A maintainer and a customer may also disagree on the root cause of a machine failure when a machine goes down again shortly after it has been repaired.

In our MaaS model, it is necessary to build trust in reference data about the machines and their operation. OEMs equip all shipped machines with a variety of attached IoT sensors generating *telemetry data*. This data reflects production metrics, uptime, downtime, and other indicators useful for predictive maintenance operations (e.g., detections of signs or wear or consumable levels monitoring).

Telemetry data can allow arbitrating between the conflicting positions of mutually distrustful MaaS actors and enables accurate and dynamic monitoring of all machines' states. Building trust in this data requires, however, to collect and store it transparently and avoid risks of voluntary deletion or tampering (e.g., a customer modifying stored data about a particular machine performance in order to avoid paying premiums on its operational performance). Transparency, immutability, and consensus-based validation of telemetry data can be a basis for building a common layer of trust for MaaS actors in their B2B (business-to-business) interactions.

We consider the clients are trusted for not tampering with the IoT boxes, since the cost of being caught doing so outweighs potential benefits (i.e., cancelling contractual obligations). In order to partially lift this assumption, one could leverage trusted execution environments such as ARM TrustZone,[32] an option we currently leave for future work. Furthermore, as the MaaS model depends on trusted deployment of IoT devices for certified monitoring, there is a possibility for a fourth type of actor, third-party compliance agencies, to interact in the model by attesting to other parties that physical sealing of the installations is effective, thereby convincing all parties that IoT monitoring data is trustworthy. We also leave the integration of these two measures for future work.

4 | MAAS: DESIGN AND ARCHITECTURE

We now detail the technical architecture of our blockchain-based solution to operating a MaaS ecosystem under the system and trust models defined in the previous section.

Our solution revolves around interactions using a blockchain. By default, the blockchain infrastructure we use is *private*, meaning that only MaaS participants are involved in the blockchain operation such as disseminating transactions and mining new blocks. We consider it important, however, to leave open a possible shift to a *public* blockchain for some MaaS deployments, that is, a blockchain where MaaS participants would rather be clients of a global set of dedicated mining nodes rather than act as infrastructure nodes themselves. *Consortium* blockchains such as Hyperledger Fabric[33] tightly integrate the validation of the membership of participating members in the consortium to the operation and correctness of the consensus layer, allowing to use efficient byzantine-fault-tolerant (BFT) protocols.[34] We choose, instead, to implement consortium management at the level of the *application layer*. All parties (OEMs, customers, and maintainers) must register to the consortium in order to engage in B2B interactions with other parties. This approach bears two advantages over the use of a consortium blockchain such as Fabric. First, it enables more flexible consortium management, as rules for joining or leaving the consortium can be decided by the consortium itself as part of its governance operations, or even change while the system is running, unlike rules for adding or removing organizations from a consortium blockchain that must abide by constraints of the BFT protocol. Second, it allows MaaS to run unmodified over either public or private permissionless systems. In the following, however, we focus on the case where MaaS actors run over a private, permissionless system.

All high-level business operations and events, such as maintenance requests and their assignment to a specific maintainer, are recorded on-chain. The immutability and persistency of data on-chain prevents later discussion and disputes about business operations, such as fairness in the access to maintenance markets by different companies operating over the same geographical area. Similarly, IoT telemetry data allows certifying the use, reliability, or maintenance effectiveness of the rented machines. However, since this data is voluminous we do not store it on-chain but rather use off-chain storage with anchorage on chain, allowing time-stamping of the data and integrity assessment.

4.1 | The MaaS private blockchain and consortium

Actors of a MaaS ecosystem collectively run a private blockchain network. This network supports a consortium formed of several types of application-level nodes, as illustrated by Figure 2.

The first three types of nodes directly map to the different actors present in the system, that is, *OEM nodes*, *Maintenance nodes*, and *Customer nodes*, and are operated by the corresponding parties. In addition, a *Boot node* is deployed prior to the deployment of any other node. This node is solely involved in bootstrapping the initial network, that is, by allowing the peer discovery process of the private blockchain to establish links between these nodes. In addition, and optionally, a *Consortium node* can be run by the consortium board*. This node enables dynamic membership to the MaaS consortium. It can generate invitations to the network and handles the registration of corresponding accounts with a specific role ("OEM," "Customer," or "Maintainer"). We note that, in practice, the Boot node and Consortium node have a different logic function but could be merged on a single server. While these nodes may appear as centralization points, they are never involved in business operations but only in the setup and evolution of the consortium. This corresponds to the need, in practice, for OEMs to certify the identity of other actors in the system, which they do know after delivering rental machines (customers) or through their accreditation program (maintainers). We emphasize that decentralized management would be possible, for example, by implementing a set of smart contracts for consortium governance, but would also greatly complicate the MaaS operation and in particular bootstrap. We choose, therefore, to trade off decentralization for simplicity in this case, highlighting that it only concerns the initial invitation and validation of partners and is not involved in the daily MaaS operations. We finally note that, as shown in Figure 2, we typically expect blockchain nodes operated by the different actors to be deployed over a wide area network using a combination of public clouds and on-premise servers.

4.2 | On-chain orchestration of MaaS business operations

We now proceed how MaaS business operations are orchestrated using on-chain transactions and smart contracts. As machines get deployed by OEMs, new contracts are deployed for (1) handling the lifecycle of machine setup and

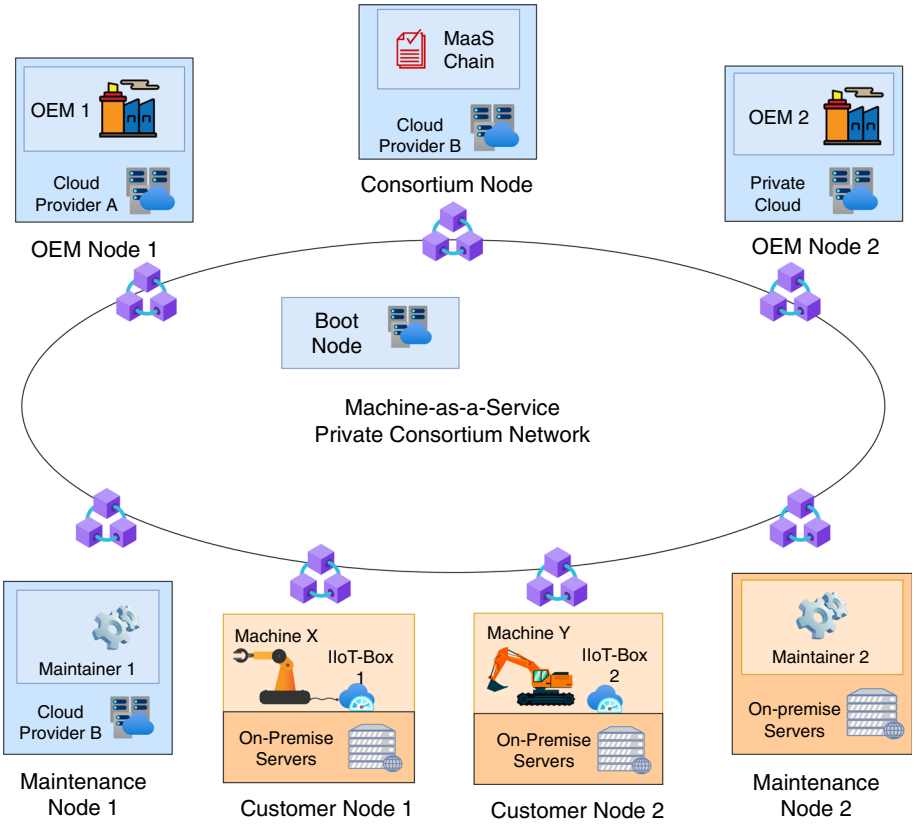


FIGURE 2 A MaaS consortium combines nodes deployed at different cloud providers and on the premises of the industrial operators. OEMs rent machines to customers under the OpEx model. Machines are monitored by industrial IoT boxes.

deployment (the *machine contract*), (2) managing payments and compensations as defined in OpEx contracts (the *invoice contract*), and (3) orchestrating the workflow of maintenance request and service operations (the *maintenance contract*).

We note already that in our current solution monetary exchanges, for example, the invoicing of customers by OEMs or maintainers, do not leverage any on-chain cryptocurrency specific to the consortium network. The motivation for this choice is twofold. First, linking the MaaS operation to a specific cryptocurrency would limit the range of blockchain systems that can be used as its foundation, impairing generality. While our current choice (further detailed in Section 5), Ethereum with proof-of-authority consensus, features such a built-in cryptocurrency, other blockchains usable in consortium settings such as Hyperledger Fabric[33] or Hyperledger Sawtooth[35] do not. Second, linking a volatile currency used within only a MaaS consortium to non-crypto currencies would require anchoring it to a large-scale public blockchain with public valuation such as Ethereum (e.g., through the issuance of ERC-20 tokens[36]), complexifying the design, making the MaaS economy dependent on the parent chain crypto-currency fluctuating valuation, and increasing its attack surface. The choice to not use an embedded cryptocurrency does not preclude, obviously, interacting actors from settling their peer-to-peer payments with an existing cryptocurrency, for example, bitcoin.

4.2.1 | Managing machines' life cycles

The *machine contract* is used by OEMs and customers to manage the life cycle of a machine rented under the OpEx model. The life cycle of a machine follows a simple three-phase workflow: the *registration* phase, the *execution* phase, and the *termination* phase. Proper orchestration of this workflow using the blockchain is mandatory to ensure that the setup of the machine is the result of an agreement between the provider and the customer, and that all necessary invoicing and maintenance provisions are in place.

The installation of a new machine follows the registration phase as illustrated by Figure 3. The OEM starts by registering and initializing a *machine contract* in the chain. This machine contract contains not only various information

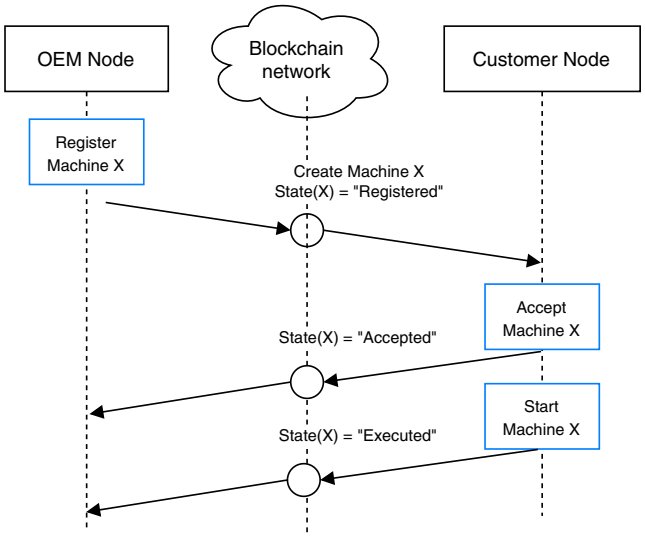


FIGURE 3 Machine life cycle, *registration* phase. An OEM node registers the machines' information on the blockchain. Once the customer accepts the machine and the accompanied invoice contract, the machine is ready and can start its operation.

about the machine, but also the identifier of the customer that it will serve. Subsequently, the OEM registers a new *invoice contract* that formalizes[†] the machine's service level agreement (SLA) such as criteria on produced items quality control, their quantity over time, payment and penalties. To proceed with the initialization, the customer needs to accept both the machine contract and the invoice contract, by calling the appropriate methods of the *machine contract*.[‡] The state of the machine then changes from *Registered* to *Accepted*.

When the customer starts to use the machine, its state transitions to *Executed* Only machines under this state can be subject to billing. The *Executed* state is associated with the mandatory collection of telemetry data for the machine using embedded IoT boxes that monitor the machines directly at the customer site. Typical telemetry data includes the operating condition information of the machine (e.g., temperature, pressure, humidity), the operational status of the machine (e.g., idle, running, malfunctioning, under maintenance) and the quantity of items produced. This telemetry data is the basis that the *invoice contract* uses to calculate payments and penalties according to the SLA. The frequency of collection of telemetry data, for example, every 15 min, is specified in the *machine contract*.

Telemetry data is not stored on chain for scalability and performance reasons. Indeed, storing this information for all machines would very rapidly saturate the transaction processing capability of current blockchain systems and lead to state explosion at all participating nodes. Instead, the storage happens off-chain with an anchorage of the data on chain allowing its later retrieval and validation. Telemetry data is propagated to the different nodes to a chosen level of replication, implementing a tradeoff between resilience to (voluntarily) erasure and state requirements for the whole network. We detail how we use the IPFS service for this purpose in Section 5.

Invoicing by the OEM happens with a frequency specified in the invoice contract, for example, every month, and generates a notification to the customer watching events on chain. The payment calculation is the result of the aggregation of telemetry data over the last invoicing period. This aggregation is specific to a machine and its associated SLA. The calculation of the invoice does not have to happen in the smart contract method itself, which would yield arbitrary long and costly transaction executions, in particular when the amount of telemetry data grows. Instead, the OEM can calculate the invoicing offline. The calculation method is published in the invoicing contract and is deterministic (i.e., it allows a single output for a given input). All input data is anchored on chain for the period since the last invoicing. The customer can thus repeat the calculation and produce, if necessary a *proof of misbehavior*[38] of the OEM and report it to the appropriate arbitration.

We do not detail the process for termination of a machine, as it follows similar principles to its setup. The customer and the OEM alike must agree on the end of the contract, and a final invoicing round is necessary to settle the termination.

4.2.2 | Managing maintenance requests' life cycle and arbitration

The operation of machines in the OpEx model entails that the maintenance of the machines is not under the responsibility of the customer itself. This applies to both preventive (e.g., planned periodically or following predictive models applied to

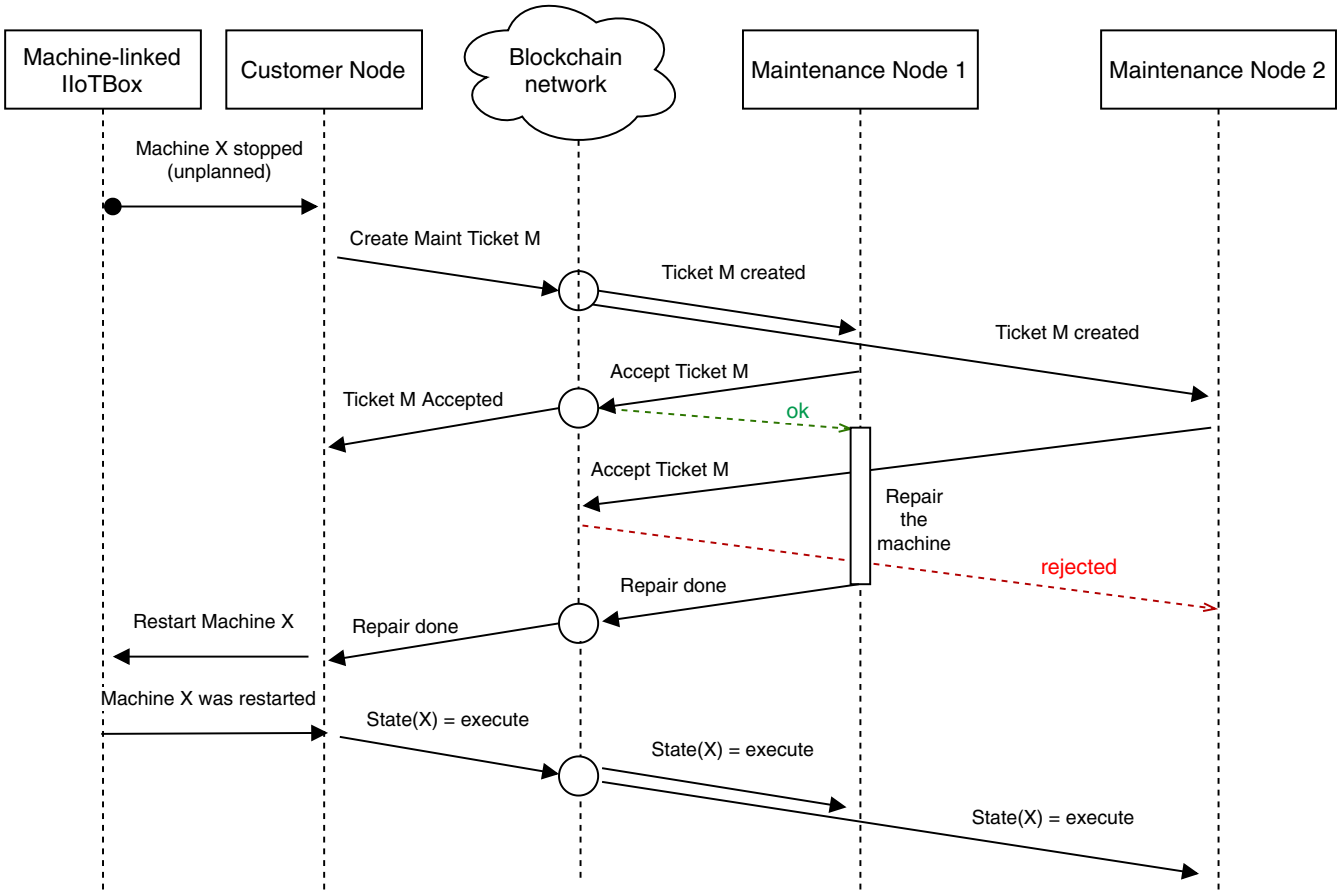


FIGURE 4 Maintenance operation life cycle. When a machine X stops operating, the customer node automatically creates a maintenance ticket on the blockchain by calling the maintenance contract. Maintainers monitor the blockchain for such tickets and bid for repair operations. In this example, the selection is based on a first-come-first-serve basis following the total order of transactions.

IoT telemetry data[39]) and reactive maintenance needs (i.e., when the machine stops operating or a quality threshold on produced items is suddenly violated). In both cases, the maintenance action is delegated to a *Maintainer* actor that has previously been approved for operating this specific machine, or class of machines, by the OEM. The reactivity (time to offer handling the maintenance operation) and effectiveness (time to resolution) are monitored on the blockchain. The maintenance operation can obey SLAs between the customer and the maintainer, between the OEM and the maintainer, or both. Similarly to the invoicing described earlier in this section, the calculation of bonuses and penalties is based on a deterministic computation over IoT telemetry data anchored on the chain. For instance, an SLA for a predictive maintenance operation could provision a bonus for an improved mean-availability-between-failures score over the next few months, or a penalty for repeated failures.

Multiple maintainers may be eligible for handling a specific maintenance request. Direct negotiation between customers and maintainers is not desirable in our context, as this would introduce unfairness between actors, prevent OEMs and other customers from monitoring the performance of maintainers, and limit the accessibility to the maintenance market for new players. We propose, instead, that the selection of the maintainer for a given event happens in a transparent and verifiable fashion using the blockchain. This process is illustrated by Figure 4 for an abrupt failure requiring an immediate repair to be able to restart the machine. The customer node receives a notification of the failure from the industrial IoT box attached to the machine. The customer node will in turn call the maintenance contract on chain to create a new maintenance ticket, registered as a transaction on the chain.

Maintenance nodes employ watchdogs on the chain content in order to be notified of a new transaction of interest, that is, for a machine for which they have the accreditation in their area of operation. If the ticket is of interest and resources are available, the maintenance node may issue an acceptance by calling the corresponding method of the maintenance smart contract. Of course, multiple maintainer nodes may positively answer to a given ticket. A selection process is necessary

to select one of them. The simplest choice is to enforce a first-come-first-serve (FCFS) strategy. The blockchain defines, indeed, a total order of operations (i.e., the order of transactions in the ledger is unique). The maintenance contract can, therefore, be assigned to the first node successful in integrating its "Accept ticket" transaction on the chain. This strategy makes sense for the case detailed in Figure 4, where the goal is to have the machine repaired as quickly as possible.

In some cases, such as preventive maintenance operations, the customer would prefer to select the best maintainer according to its own criteria (e.g., past performance of maintenance actions as recorded on chain, price, guarantees, or physical proximity). It is possible to modify the workflow to postpone the selection until either (1) a given number of "Accept ticket" transactions have been seen on chain, (2) a given number of blocks have been appended to the chain since the one containing the ticket itself, or (3) a combination thereof. While our current prototype of MaaS supports only FCFS, adding such selection would be relatively simple, provided the considerations discussed in the next paragraph are taken into account.

4.2.3 | Workflow orchestration and finality guarantees

We now make an important observation regarding the implementation of the maintenance workflow over a blockchain. This observation applies, in fact, to any workflow orchestration based on a blockchain with actors competing to advance to a new state.

Regardless of the strategy (FCFS or a more elaborate one), the implementation of the maintenance ticket workflow over the blockchain must account for *finality* guarantees offered by the support blockchain.[6,40] Finality refers to the guarantees a blockchain offers concerning the (im)possibility of revocation of transactions, even after they have been reported as successful to the client. Blockchains based on proof-of-work, proof-of-stake, or proof-of-authority typically offer probabilistic finality guarantees. A transaction included in a block and reported to the client may, indeed, be part of a block that is later revoked as part of a fork resolution. This situation happens when two independent miners propose equally-correct new blocks for the same height. To preserve the ledger chaining, only one of the two blocks remains valid and the other one is discarded. If a maintainer receives a confirmation that its "Accept ticket" transaction has been included in a new block, it is not yet able to proceed safely to the next phase immediately. It might be the case, indeed, that a fork happened and another maintainer received the same confirmation for a transaction located in the other block. As for the settlement of financial transactions in bitcoin,[41] it is necessary to wait for a number of *confirmations*, that is, blocks chained over the one where the transaction appears, in order to make a reliable decision for the advancement in the workflow. In MaaS, we implement this by simply using a timer triggered by the watchdog on the chain and a second read after this timer expires to confirm the decision.

5 | IMPLEMENTATION

We now detail our prototype implementation of MaaS. We first present the technical components of each of the nodes, and the interaction with the IoT devices.

Figure 5 presents the technical constituents of a MaaS node. Each participant runs one such node, which can be parameterized to act according to one of the roles detailed in the previous section, that is, as a customer, OEM, maintainer, boot, or consortium node. These components are packaged as Docker containers linked by Docker compose, and all are built upon open-source software.

The central point of interaction and orchestration is the "API node." It implements the high-level operations for the MaaS function the node has been parametrized for. It implements the business logic of MaaS operations as defined in the previous section, and provides a REST interface for external programmability. A Web-based user-interface further allows user-friendly operations for human operators.

5.1 | Private blockchain

We chose Ethereum[9] as the target blockchain platform for several reasons. First, it enables our objective of supporting a private blockchain solution initially with the ability to later switch to a public mode of operation. Second, it

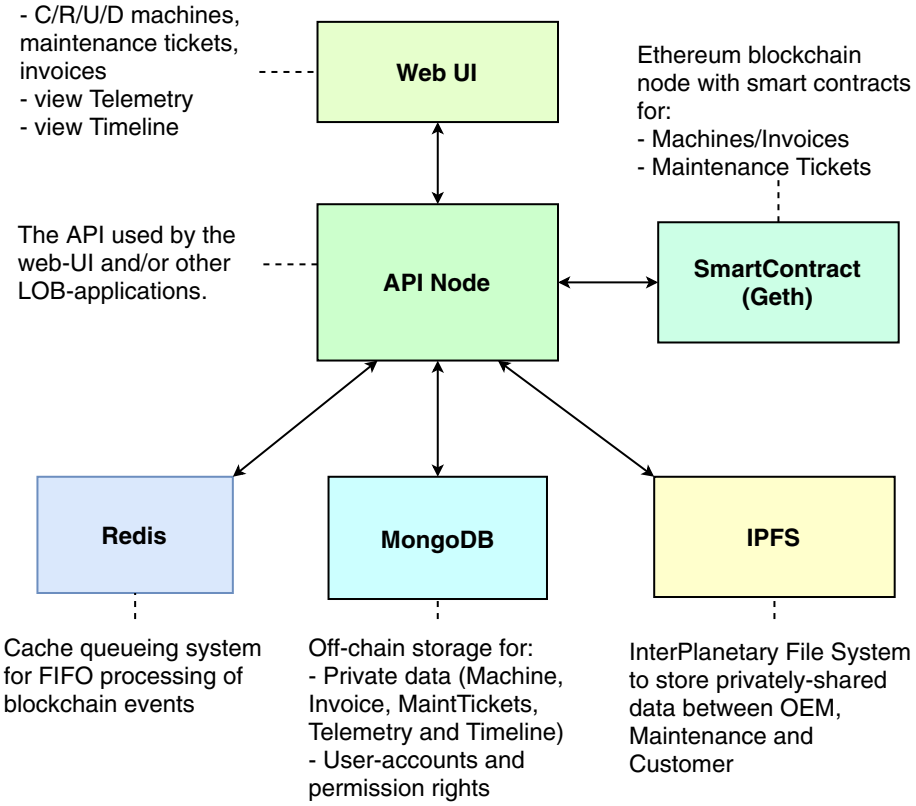


FIGURE 5 MaaS node components: Each MaaS node represents and is owned by a consortium participant. It is composed of six containerized components.

is the most widely used blockchains for generic applications, comes with a large ecosystem, and is easier to set up and operate in limited-capacity IoT boxes and servers in comparison to cloud-focused platforms such as Hyperledger Fabric.[33]

As we are running a *private* Ethereum network with a limited number of nodes, proof-of-work is not an appropriate consensus mechanism, and would impose a significant computational overhead. As per Ethereum's recommendations for private installations, we use instead the Clique[42] proof-of-authority consensus protocol. Authorities (authorized signers) can be registered in the *genesis* (i.e., first) block of the private blockchain, and through a voting mechanism amongst defined signers thereafter. In MaaS, the initial authority is the consortium node, and it is the only one allowed to register new authorities[§]. The Boot node is in charge of bootstrapping the private network and registers the consortium node as a signer in the genesis block. The consortium node can then add and remove authorized signers as it certifies the identity of new OEMs, customers, and maintainers.

The operation of Ethereum depends on its internal cryptocurrency, the *ether*. In MaaS, this cryptocurrency is not used for monetary exchanges between parties but remains necessary for allowing them to issue transactions and call smart contract operations. Controlling currency is also useful to prevent a particular malevolent party from performing a denial-of-service attack by sending an arbitrary number of computationally-intensive transactions. We control the availability of ether to the different participants through an additional service smart contract that periodically dispatches a small sum of ether to each participant. This contract can be called by the consortium node only[¶].

Each node runs the Go Ethereum (`geth`) client.[43] Interactions between the API node and the blockchain leverage the web3js Ethereum JavaScript API. The three smart contracts amount to a total of 350 lines of Solidity code. To allow all parties to verify the validity of the contracts, their source code is published by the consortium node on the chain at the time the corresponding bytecode is registered, avoiding the need for an extra authoritative repository for this information. A maintainer node may, for instance, verify that the processing of its bids to a maintenance ticket were treated fairly by the corresponding smart contract method.

5.2 | Distributed, off-chain storage

High-volume shared data required for MaaS operations, such as machine's documentation, invoicing results, and obviously IoT-generated telemetry data are stored in the off-chain IPFS database. Each node in the consortium runs one IPFS node and participates to storing redundant copies of this data. Data is shared and accessible only within certain groups, and is therefore stored encrypted in IPFS. We leverage IPFS's support for multiple isolated shared storages, accessible only to a sub-group of the consortium. For instance, an OEM may create one such group for its maintainers and customers while allowing another OEM to co-exist without access to its data. IPFS anchors the off-chain data onto the blockchain by means of its CIDs, enabling their validation and authentication.

5.3 | Local storage

In addition to the two forms of distributed storage that are the blockchain and IPFS, each nodes maintains two local database.

First, an instance of the MondoDB document-oriented NoSQL database is used as a local store. It caches data retrieved from IPFS and certified using blockchain data, or data fetched from the blockchain itself such as contractual information, in addition to local, specific data about the machine operation, pending maintenance operations and so forth that do not have to be exposed to other parties. In more details, the following business objects are stored in MongoDB: user accounts, machines, business partners, telemetry timeline data, invoices, maintenance tickets, and technical logs.

Second, a redis in-memory key-value store is used as a publish/subscribe communication layer, enabling the ordered processing of asynchronous events generated locally and from remote interactions, and simplifying the code of the API node.

Telemetry data is buffered in MongoDB until a configured-size batch is available. The processing and storage of telemetry data is illustrated by Figure 6. When a batch of data is available, it is encrypted and stored in the IPFS store. The hash of the content is collected from IPFS and anchored (written) into the chain. The validation of the write is implemented

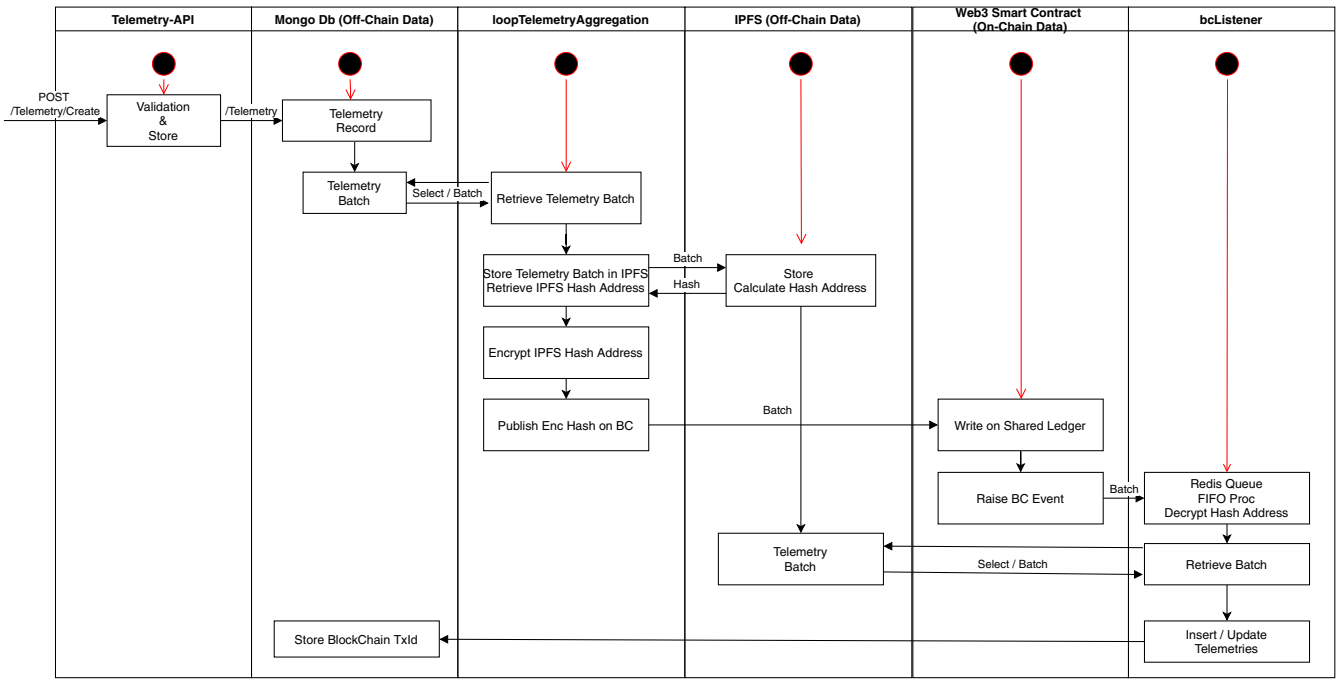


FIGURE 6 Telemetry data flow: An IoT box posts telemetry data entries via the web API to the customer node, which writes the entries into the local MongoDB instance. The node periodically aggregates these entries into a batch, stores it into IPFS, and gets and encrypts its hash address. Then, the node writes this encrypted hash address and its metadata on the blockchain. Other nodes in the consortium receiving the block containing this information can retrieve the original telemetry data on IPFS if their permission settings allow it.

through an asynchronous wait for events exchanged through redis. Once the on-chain entry is validated (and after the necessary number of confirmations have been received) then the resulting data is cached in the local MondoDB as well.

6 | EVALUATION

We now present evaluation results from a deployment of our MaaS prototype. We are interested in answering the two following questions: (1) what is the relative cost of on-chain operations, and (2) what are their relative latencies?

We run our experiments in the Azure public cloud, using virtual machines with 2 virtual-CPUs, 4 GB of RAM, a local SSD storage of 30 GB, and the Linux Ubuntu Server 18.04 operating system.

6.1 | Network emulation

We are interested in evaluating MaaS under a representative geo-distributed scenario. An option would have been to rent virtual machines in multiple Azure regions, but this option comes with significant cost and deployment complexity, and does not easily allow repeating experiments under the same conditions. We choose instead to leverage *network emulation* to reproduce, in a configurable manner, a network topology and individual link performances of our choice.

We choose to leverage Enoslib, a generic experiment and network emulation framework.[44,45] Enoslib relies on Ansible to send commands to remote nodes. Under the hood, it creates an Hierarchical Token Buckets qdisc (i.e., queuing disciplines) tree in the Linux kernel of each machine to emulate network conditions to multiple destinations. Since Enoslib only supports node-level network emulation, we extend its capabilities to also support multi-region network emulation.

Our scripts deploy the network topology presented on the left-hand side listing of Figure 7, with the mapping of the different nodes presented on the right-hand side listing. The map graphically represents the both types of information. We deploy one each of consortium, boot, OEM, and customer nodes, and two maintainer nodes.

6.2 | Cost of Ethereum operations

We set up an experiment featuring a continuous loop of transactions, and implementing two consecutive sequences of operations: (1) the provisioning and set up of a new machine, as detailed in Figure 3, and (2) the failure of the machine and the resulting maintenance operation life cycle, as detailed in Figure 4. In more details, in the first cycle, an OEM node sets up a new machine instance on the blockchain and registers invoicing and other contractual information on it. The OEM node listening to blockchain events detects the creation, checks this data, and eventually accepts the setup before effectively starting the machine. In the second cycle, the customer node emulates a *machineStop* event. This event triggers the creation of a new maintenance ticket on the chain (i.e., a call for bids). The ticket is also registered with the machine contract. Two maintenance nodes, corresponding to two authorized maintenance companies, are listening to such call for bid events on the chain. Both immediately generate a transaction featuring a call to the maintenance smart contract to accept the maintenance operation. Only the first of the two transactions, in the order of the inclusion in the

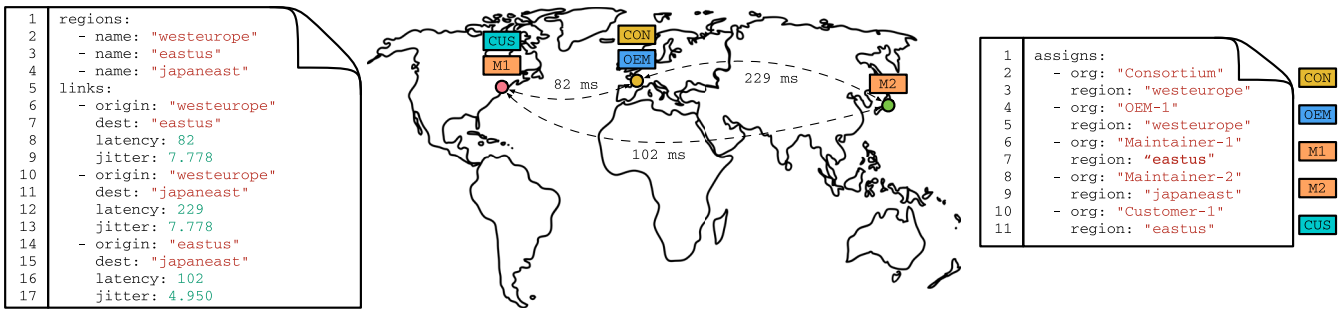


FIGURE 7 Emulated topology (left) and placement of MaaS nodes (right)

TABLE 1 Average gas cost of transactions calling smart contract methods in MaaS, sorted by decreasing costs.

Smart contract	Method	Gas cost
machine	Register a machine	1,833,417
invoice	Create invoice contract	110,287
machine	Accept a machine	72,012
maintenance	Accept a ticket	57,110
maintenance	Create a ticket contract	48,165
maintenance	Close a ticket	41,291
machine	Stop a machine	41,040
machine	Start a machine	37,157
machine	Reset a machine	37,148
maintenance	Complete a ticket	36,315

blocks' content, is successful. When it is the case the customer, which was also actively listening to ticketStart events, automatically restarts the machine and closes the ticket before restarting the whole loop.

We implemented the coordination of these operations using a workload injector implemented as a Postman script calling the HTTP/JSON API of all involved nodes.

We show in Table 1 the relative gas cost for each of the operations (i.e., calls to smart contract functions) involved in the two life cycles. The table lists the involved contract, the corresponding method, and the average amount of gas used. In Ethereum, the gas corresponds to the amount of computation required from miners to run the transaction and include it in the chain. We set the maximum gas to 10 million units, which is never reached by any of the calls. We can observe that the machine registration consumes significantly more gas and, therefore, resources than the other operations. This is, however, a *one time* operation and its cost comes primarily from the need to write various detailed information about the machine (technical specifications, related documents, e-signature for ownership …) on the chain. The invoicing is, on the other hand, a periodic operation that also has a higher cost than other operations, although less so than the machine registration (by more than one order of magnitude). This contract must, indeed, go through the history of use of the machine stored in IPFS and anchored on chain since the last invoice and apply the invoicing method agreed between the different parties upon set up of the machine. All operations that are part of the maintenance life cycle (i.e., creating, accepting, completing, and closing a ticket) only require modest amounts of gas, and only access a fixed amount of state in the blockchain.

6.3 | Latency of MaaS operations

We now discuss the latency of individual operations involved in MaaS workflows. We repeated the complete workflow described above a total of 80 times and report the distribution of execution times for individual operations in Figure 8. The latency of an operation is measured from the time the corresponding node issues the transaction with the call to the smart contract, to the time it observes the block containing the transaction. As we discussed in Section 4, observing the transaction in a block is insufficient to validate the transaction execution in the ledger, as finality requires waiting for several follow-up blocks to guarantee with a high probability that the block containing the transaction will not be discarded following a fork event. In our implementation, this wait is implemented as a fixed-duration timer, so we do not reproduce it in the figure for the sake of clarity.

We can observe that the execution time for all operations is fairly stable, and takes between two to three seconds, with outliers taking up to four seconds. We wanted to verify that the geo-distribution of nodes would not impact negatively fairness, that is, the relative performance of operations for the different nodes located in different locations. We observe that it is not the case: operations to observe a new maintenance ticket and to accept it take a medium latency that is very similar for the maintenance node co-located with the customer node in the Eastern US region and for the one located in Eastern Japan. Latencies are, in fact, dominated by the global propagation of transactions and blocks in Ethereum and their validation by all peers. It is possible, however, that this observation does not hold should MaaS be implemented

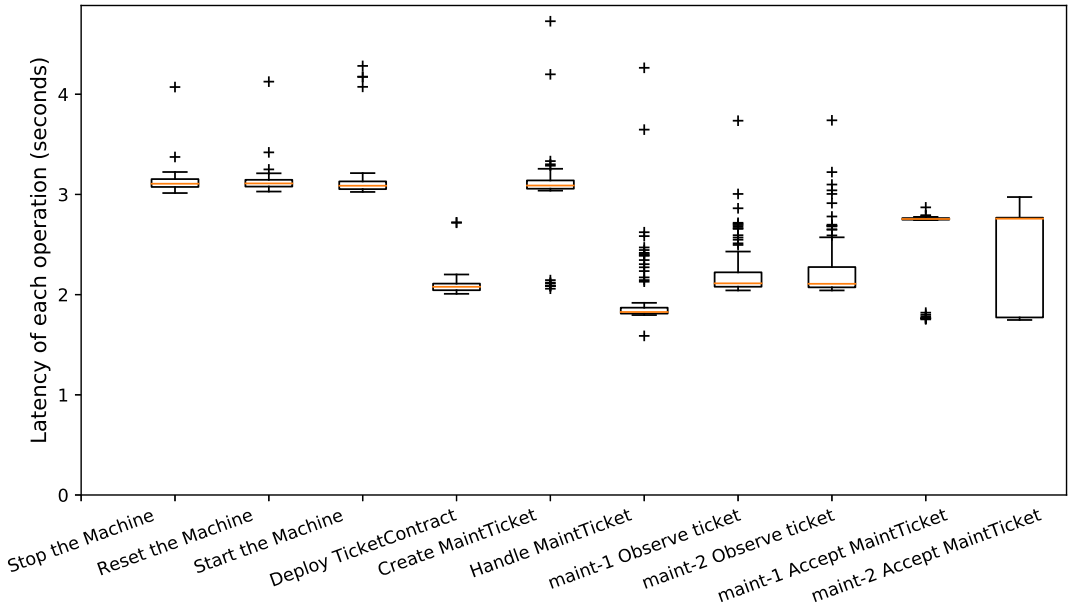


FIGURE 8 Latency of each operation (not including waiting time for finality)

with another blockchain supporting higher-throughput operations such as Hyperledger Fabric, as has been shown by previous work.[46]

7 | RELATED WORK

We review related work on the application of blockchain and/or IoT technologies in industrial contexts close to the one of MaaS considered in this article.

7.1 | Machine-as-a-Service and Industrial IoT

MaaS, as we understand it in this article, is based on a business model where a machine manufacturer sells or leases out their machines at little cost and billing their customers on machine usage, for example expressed by the number of produced items.[47] Also called *Equipment-as-a-Service*, it is regarded as "one of the most drastic examples" of usage-based (or outcome-based) business models today,[48] although its roots go back to the pre-Internet era, with for example Rolls Royce's "power by the hour" concept in the 1960s.

MaaS should not be confused with another business model that has appeared in the literature under various names, such as *Production-as-a-Service*,[49] *Manufacturing as a Service*,[50,51] or *MaaS type 2*,[2] where machine or factory owners sell production time on their (underutilized) machines to product developers who do not have their own production facilities. For that model, the research focus has been on the design and development of broker architectures that allow machine owners to advertise the available machinery and customers to book production capacities. Since the machine stays under control of the facility owner in that model, questions related to trust or tractability are not raised. An obvious drawback is that customers have to rely on the availability of production capacity owned and controlled by others.

There are various works on automating industrial machine management without blockchains. These non-blockchain solutions are usually simpler and might achieve high performance but all participants in the system have to give all trust to a centralized entity who fully controls the business logic of the system. Markeset[52] propose an approach for automating product support and the maintenance process for industrial systems. Barbosa et al.[53] collect data from a shop floor and provide them to decision-makers in real time. In the context of predictive maintenance and MaaS, IoT-based monitoring provides the necessary data to make maintenance decisions. As noted by Zhang et al.,[47] unexpected equipment failures can result in expensive downtime costs in the MaaS business model. They take advantage of failure prediction and dynamic

pricing to achieve a pricing policy that optimizes profit. Finally, before the term "IoT" became popular, Gilart-Iglesias et al.[12] suggested to "normalize" industrial machinery, that is, to equip them with embedded devices connected to a network such that they can be controlled and monitored remotely. Their goal was to integrate machinery similar to other ICT services into the business process of the company. They called this concept *Industrial Machines as a Service*, a term that is, despite its similarity, not related to the business and operational model presented in this article.

7.2 | Hardware-level trust for Industrial IoT

The usage of IoT devices to monitor machines in Industrial IoT scenarios requires that the involved parties trust the data collected and reported by the sensors. To achieve this, approaches based on trusted execution environments, in particular ARM's TrustZone,[32] have been proposed.[54] As these approaches work on device hardware level, they can be combined with our blockchain-based MaaS model.

Lesjak et al.[55] compare ARM TrustZone with a so-called security controller to implement a device snapshot authentication system for Industrial IoT. While TrustZone is a security mechanism directly implemented in the main microprocessor of an IoT device, a security controller requires a dedicated tamper-resistant secondary microprocessor that provides a set of cryptographic operations. The authors show that both technologies have their advantages and therefore propose a hybrid approach that maximizes security. Pinto et al.[56] propose a TrustZone-based architecture that implements a trusted execution environment as a low-priority thread of a real-time operating system. They envision an application scenario for Industrial IoT where end devices store sensible data in that trusted environment. Non-secure software is executed outside that environment on a classic IoT OS, such as Contiki or RIOT. The StreamBox-TZ architecture by Park et al.[57] targets edge devices, that is, devices that are positioned between the sensor devices performing the machine measurements and the cloud. The edge devices run stream analytic operations over the sensor data and only send compact results and audit records to the cloud backend. Using TrustZone, the authors demonstrate a secure stream analytic engine that delivers state-of-the-art event processing throughput on small edge devices.

7.3 | Usage of blockchains in smart manufacturing

For predictive maintenance in the Industry 4.0, Sang et al. propose, without providing technical details, to store asset manufacturer data in a blockchain to achieve data transparency and traceability.[58] This data includes measurements and control data from the manufacturer of the asset, but not production data, defect and maintenance data, nor historical operational data. No smart contracts are used. The company ZkSystems collects and sends data from a nutrunner to a blockchain in order to be able to identify and detect problems during the production.[59]

Mayer et al.[60] describe several use cases of Distributed Ledger Technologies (DLTs) in real world manufacturing scenarios. Their use case "Subscription models for machines and tools" is similar to the Production-as-a-Service business model described above. We observe that their motivation to use DLTs partially overlaps with ours, namely establishing confidence in the responsible use of rented machines.

The Blockchain-based Platform for IIoT (BPIIoT) proposed by Bai et al.[13] also targets a Production-as-a-Service model. Their goal is to build a system that supports the development of decentralized end-to-end manufacturing applications where the owners of manufacturing owners can provide on-demand manufacturing services. Smart contracts serve as an agreement between those providers and the service consumers. The equipment can also automatically send service requests to a maintenance service provider in the case that certain operational parameters (e.g., the temperature) are outside safe ranges. Like us, they have identified the computational and communication cost of the blockchain as a limiting factor and proposed to split their architecture into an on-chain network and an off-chain network. The off-chain network is responsible for encrypted data storage and calculation processing and uses a distributed hash table that can be accessed by the blockchain.

7.4 | Other usages of blockchains in industrial applications

There has been a considerable number of works on blockchain for supply chains[15,20] and other similar industries.[61] Blockchain is attractive to the supply chain industry by allowing to trace the origins of a product for safety, quality,

or ethical reasons. Applications range from food supply,[62] medicine supply,[63] to mining.[64] Other works focused on the benefit of transparency which enables anti-fraud capability. Everledger[65] helps, for instance, in tracking and preventing counterfeit of diamonds and other luxury products. In transportation and shipping industry, TradeLens,[66] a collaborative solution of IBM and Maersk, and OpenPort[67] focus on tracing products from the moment they are loaded into trucks until they reach the customers. An ambitious goal is to involve governmental custom agencies in the system, so that the export/import duties and paperwork could be simplified.

Other authors have proposed to leverage blockchain to support bidding operations for resources or services, as used for example in the FileCoin[11] system or for fog computing resource allocation,[68] and similarly to the operations associating maintenance companies to machines requiring services in MaaS. PASTRAMI[38] enables *multi-item* auctions while protecting the privacy of bidders and sellers, through a protocol requiring complex asynchronous interactions between different parties. DEAL[69] also provides privacy guarantees for auctions in the context of smart grids energy exchange, while the work of Gala and Youssef[70] discusses the verifiability of sealed-bid auctions using an implementation over Ethereum. In MaaS, our operation constraints do not require such functionalities: auctioning is implemented as a simpler, transparent smart contract, but a replacement with more advanced algorithms could be preferred (and doable) in some deployment scenarios.

8 | CONCLUSION

We presented an implementation of the MaaS ecosystem based on a blockchain as a common base of trust for all participating entities, that is, OEMs providing machines, customers using the following the operational expense (OpEx) model, and maintenance companies reacting to service requests in an open and verifiable manner.

Our work opens several interesting perspectives. First, the current model requires participating entities, such as maintenance companies and customers, to act "in the open" when bidding for service requests and requesting these services. Several emerging techniques allow performing privacy-preserving exchanges over data stored on blockchain and access by smart contracts. For instance, PASTRAMI[38] enables privacy-preserving multi-party and multi-item auctions that do not need to reveal competing bids others than the winning ones, and enable loosing bidders to submit *proofs of misbehaviors* when detecting unfair treatment by one of the other parties in the bidding workflow. Such techniques could be leveraged in MaaS to allow reputation building for participating entities on the basis of such misbehaviors, and for avoiding to reveal more than necessary for implementing business operations. Second, it would be interesting to evaluate the scalability and performance of MaaS using different blockchain technologies, keeping in mind their viability and maintainability over time, as well as the support for associated off-chain storage.

AUTHOR CONTRIBUTIONS

Viet Hoang Tran: conceptualization (lead); data curation (lead); methodology (lead); software (supporting); writing – original draft (lead); writing – review and editing (supporting). **Bernard Lenssens:** conceptualization (lead); methodology (lead); project administration (lead); software (lead); writing – original draft (supporting). **Ayham Kassab:** software (supporting); writing – original draft (supporting); writing – review and editing (supporting). **Alexis Laks:** project administration (supporting). **Etienne Rivière:** conceptualization (supporting); funding acquisition (lead); methodology (supporting); project administration (lead); software (supporting); supervision (lead); writing – original draft (lead); writing – review and editing (lead). **Guillaume Rosinosky:** conceptualization (supporting); methodology (supporting); software (supporting); validation (supporting); writing – original draft (supporting); writing – review and editing (supporting). **Ramin Sadre:** funding acquisition (lead); methodology (supporting); project administration (supporting); validation (supporting); writing – original draft (supporting); writing – review and editing (supporting).

ACKNOWLEDGMENT

This work was supported by The Brussels Institute for Research and Innovation (Innoviris) under project FairBCaaS, and by the Belgian *Fonds de la Recherche Scientifique* (FNRS) under Grant #F452819F.

CONFLICT OF INTEREST

Authors have no conflict of interest relevant to this article.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ETHICS STATEMENT

This work does not raise ethical concerns.

ENDNOTES

*This consortium board would typically be composed of representative persons from all partners, and be financed by all of them as well. The board is neutral and not involved in other business decisions or transactions verification process in the system.

†For the moment, we expect humans to check and validate contractual documents stored onto the chain. Using a machine-processable SLO representation[37] for automated validation would be a possible addition to the MaaS model that we leave for future work.

‡Note that the role of OEM and customer could be swapped, that is the customer registers a machine on the blockchain first, and then the OEM accepts the SLA for that machine.

§In the case there is no consortium node, the set of participants is a globally-known constant.

¶In a deployment of MaaS over a public blockchain, one would expect participant to contribute with their own currency to the execution of the operations.

ORCID


Etienne Rivière https://orcid.org/0000-0002-4133-394X

REFERENCES

1. Microsoft. Manufacturing trends report; 2019. https://info.microsoft.com/rs/157-GQE-382/images/EN-US-CNTNT-Report-2019-Manufacturing-Trends.pdf
2. What is the machines as a service business model? 2019. https://www.exorint.com/en/blog/2019/04/26/what-is-the-machines-as-a-service-business-model
3. Al-Jaroodi J, Mohamed N. Blockchain in industries: a survey. *IEEE Access*. 2019;7:36500-36515.
4. Evermann J, Kim H. Workflow Management on the Blockchain—Implications and Recommendations. arXiv preprint arXiv:1904.01004, 2019.
5. Li Z, Kang J, Yu R, Ye D, Deng Q, Zhang Y. Consortium blockchain for secure energy trading in industrial internet of things. *IEEE Trans Ind Inform*. 2017;14(8):3690-3700.
6. Anceaume E, Pozzo A, Rieutord T, Tucci-Piergiovanni S. On finality in blockchains. Proceedings of the Conference on Principles of Distributed Systems (OPODIS); 2021.
7. Ljungberg Õ. Measurement of overall equipment effectiveness as a basis for TPM activities. *Int J Operat Product Manag*. 1998;18(5):495-507.
8. Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. *Decentral Bus Rev*. 2008;21260. https://www.debr.io/article/21260
9. Buterin V. What is Ethereum? Ethereum official webpage, 2016. http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html
10. Benet J. IPFS - content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561, 2014.
11. Psaras Y, Dias D. The InterPlanetary file system and the filecoin network. Proceedings of the 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S); 2020:IEEE.
12. Gilart-Iglesias V, Maciá-Pérez F, Capella-D'alton A, Gil-Martínez-Abarca JA. Industrial machines as a service: a model based on embedded devices and web services; 2006:630-635; IEEE.
13. Bai L, Hu M, Liu M, Wang J. BPIIoT: a light-weighted blockchain-based platform for industrial IoT. *IEEE Access*. 2019;7:58381-58393.
14. Neil S. Blockchain and the Making of Machine-as-a-Service; 2019. https://www.automationworld.com/products/data/article/13319631/blockchain-and-the-making-of-machineasaservice
15. Blossey G, Eisenhardt J, Hahn G. Blockchain technology in supply chain management: an application perspective. Proceedings of the 52nd Hawaii International Conference on System Sciences; 2019.
16. Scheid EJ, Rodrigues BB, Granville LZ, Stiller B. Enabling dynamic SLA compensation using blockchain-based smart contracts. Proceedings of the 2019 IFIP/IEEE Symposium on Integrated Network and Service MANAGEMENT (IM); 2019:53-61.
17. Alzubaidi A, Solaiman E, Patel P, Mitra K. Blockchain-based SLA management in the context of IoT. *IT Prof*. 2019;21(4):33-40. doi:10.1109/MITP.2019.2909216
18. Noizat P. Chapter 22 - Blockchain electronic vote. In: Lee Kuo Chuen D, ed. *Handbook of Digital Currency*. Academic Press; 2015:453-461.
19. Szabo N. Formalizing and securing relationships on public networks. *First Monday*. 1997;2(9-1). https://firstmonday.org/ojs/index.php/fm/article/download/548/469
20. Gonczol P, Katsikouli P, Herskind L, Dragoni N. Blockchain implementations and use cases for supply chains-a survey. *IEEE Access*. 2020;8:11856-11871.
21. Singh M, Kim S. Chapter Four - Blockchain technology for decentralized autonomous organizations. In: Kim S, Deka GC, Zhang P, eds. *Role of Blockchain Technology in IoT Applications. 115 of Advances in Computers*. Elsevier; 2019:115-140.
22. Chen Y, Bellavitis C. Blockchain disruption and decentralized finance: the rise of decentralized business models. *J Bus Venturing Insights*. 2020;13:e00151.

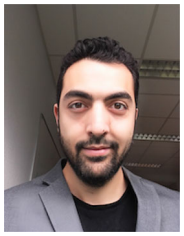
23. Dowling M. Fertile LAND: pricing non-fungible tokens. *Finance Res Lett*. 2022;44:102096. https://doi.org/10.1016/j.frl.2021.102096
24. Introduction to the Ethereum stack; July 13, 2021. https://ethereum.org/en/developers/docs/ethereum-stack/
25. Bentov I, Lee C, Mizrahi A, Rosenfeld M. Proof of activity: extending Bitcoin's proof of work via proof of stake [extended abstract] y. *ACM SIGMETRICS Perform Eval Rev*. 2014;42(3):34-37.
26. Nguyen CT, Hoang DT, Nguyen DN, Niyato D, Nguyen HT, Dutkiewicz E. Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities. *IEEE Access*. 2019;7:85727-85745. doi:10.1109/ACCESS.2019.2925010
27. King S, Nadal S. Ppcoin: peer-to-peer crypto-currency with proof-of-stake. self-published paper; Vol. 1, August 2012; 19.
28. Dannen C. *Introducing Ethereum and solidity*. Springer; 2017:318.
29. Guidi B, Michienzi A, Ricci L. Data persistence in decentralized social applications: the IPFS approach. Proceedings of the 18th Annual Consumer Communications & Networking Conference; 2021:1-4; IEEE.
30. Lev-Ari K, Spiegelman A, Keidar I, Malkhi D. Fairledger: a fair blockchain protocol for financial institutions. arXiv preprint arXiv:1906.03819, 2019.
31. Crain T, Natoli C, Gramoli V. Red Belly: a secure, fair and scalable open blockchain. Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21); 2021; IEEE.
32. Pinto S, Santos N. Demystifying arm trustzone: a comprehensive survey. *ACM Comput Surv (CSUR)*. 2019;51(6):1-36.
33. Androulaki E, Barger A, Bortnikov V, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. Proceedings of the 13th ACM EuroSys conference; 2018.
34. Sousa J, Bessani A, Vukolic M. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks; 2018.
35. Shi Z, Zhou H, Hu Y, Jayachander S, de Laat C, Zhao Z. Operating permissioned blockchain in clouds: a performance study of hyperledger sawtooth. Proceedings of the 2019 18th International Symposium on Parallel and Distributed Computing (ISPDC); 2019:50-57; IEEE.
36. Victor F, Lüders BK. *Measuring Ethereum-Based ERC20 Token Networks*. Springer; 2019:113-129.
37. Uriarte RB, Tiezzi F, De Nicola R. Slac: a formal service-level-agreement language for cloud computing. Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing; 2014:419-426; IEEE.
38. Król M, Sonnino A, Tasiopoulos A, Psaras I, Rivière E. PASTRAMI: privacy-preserving, auditable, scalable & trustworthy auctions for multiple items. Proceedings of the 21st International Middleware Conference. ACM/IFIP; 2020.
39. Carvalho TP, Soares FA, Vita R, Francisco RP, Basto JP, Alcalá SG. A systematic literature review of machine learning methods applied to predictive maintenance. *Comput Ind Eng*. 2019;137:106024.
40. Zhang S, Lee JH. Analysis of the main consensus protocols of blockchain. *ICT Exp*. 2020;6(2):93-97.
41. Gundlach R, Gijsbers M, Koops D, Resing J. Predicting confirmation times of Bitcoin transactions. *ACM SIGMETRICS Perform Eval Rev*. 2021;48(4):16-19.
42. Szilágyi P. EIP-225: clique proof of authority consensus protocol. *Ethereum Improv Propos*. 2017. https://eips.ethereum.org/EIPS/eip-225
43. Go Ethereum: official go implementation of the Ethereum protocol; 2021. https://geth.ethereum.org
44. Cherrueau RA, Simonin M. *EnOSlib*. 2021. Inria. https://discovery.gitlabpages.inria.fr/enoslib/
45. Cherrueau RA, Simonin M, Van Kempen A. EnosStack: a LAMP-like stack for the experimenter. Proceedings of the Workshops of IEEE INFOCOM 2018, IEEE Conference on Computer Communications; 2018:336-341; IEEE.
46. Berendea N, Mercier H, Onica E, Riviere E. Fair and efficient gossip in hyperledger fabric. Proceedings of the 40th IEEE International Conference on Distributed Computing Systems (ICDCS); 2020; IEEE.
47. Zhang C, Gupta C, Joichi S, Farahat A, Shao H. Risk-based dynamic pricing via failure prediction. Proceedings of the 18th IEEE International Conference on Machine Learning and Applications (ICMLA); 2019:140-147; IEEE.
48. Stojkovski I, Achleitner AK, Lange T. Equipment as a service: the transition towards usage-based business models SSRN; 2021.
49. Balta EC, Lin Y, Barton K, Tilbury DM, Mao ZM. Production as a service: a digital manufacturing framework for optimizing utilization. *IEEE Trans Automat Sci Eng*. 2018;15(4):1483-1493. doi:10.1109/TASE.2018.2842690
50. Rauschecker U, Meier M, Muckenhirn R, Yip ALK, Jagadeesan AP, Corney J. Cloud-based manufacturing-as-a-service environment for customized products; 2011.
51. Medeiros GH, Cao Q, Zanni-Merk C, Samet A. Manufacturing as a service in industry 4.0: a multi-objective optimization approach. Proceedings of the International Conference on Intelligent Decision Technologies; 2020:37-47; Springer.
52. Markeset T, Kumar U. Design and development of product support and maintenance concepts for industrial systems. *J Qual Mainten Eng*. 2003;9(4):376-392.
53. Barbosa G, Aroca R. An IoT-based solution for control and monitoring of additive manufacturing processes. *J Powder Metall Min*. 2017;6(158):2.
54. Serror M, Hack S, Henze M, Schuba M, Wehrle K. Challenges and opportunities in securing the industrial internet of things. *IEEE Trans Ind Inform*. 2020;17(5):2985-2996.
55. Lesjak C, Hein D, Winter J. Hardware-security technologies for industrial IoT: TrustZone and security controller. Proceedings of the 41st Annual Conference of the IEEE Industrial Electronics Society (IECON); 2015:002589-002595.
56. Pinto S, Gomes T, Pereira J, Cabral J, Tavares A. IIoTEED: an enhanced, trusted execution environment for industrial IoT edge devices. *IEEE Internet Comput*. 2017;21(1):40-47.
57. Park H, Zhai S, Lu L, Lin FX. StreamBox-TZ: secure stream analytics at the edge with TrustZone. USENIX Annual Technical Conference (ATC); 2019:537-554.

58. Sang GM, Xu L, Vrieze dP, Bai Y, Pan F. Predictive maintenance in industry 4.0. Proceedings of the 10th International Conference on Information Systems and Technologies; 2020:1-11.
59. Bosch nexo case study: digitizing a nutrunner; 2019. https://medium.com/zksystems/bosch-nexo-case-study-blockchain-powered-nutrunner-597851a2299
60. Mayer J, Niemietz P, Trauth D, Bergs T. How distributed ledger technologies affect business models of manufacturing companies. *Proc CIRP*. 2021;104:152-157.
61. Miller D. Blockchain and the internet of things in the industrial sector. *IT Prof*. 2018;20(3):15-18.
62. Aung MM, Chang YS. Traceability in a food supply chain: safety and quality perspectives. *Food Control*. 2014;39:172-184.
63. Sylim P, Liu F, Marcelo A, Fontelo P. Blockchain technology for detecting falsified and substandard drugs in distribution: pharmaceutical supply chain intervention. *JMIR Res Protocols*. 2018;7(9):e10163.
64. Chohan UW. Blockchain and the extractive industries: cobalt case study; 2021.
65. Gutierrez C, Khizhniak A. A close look at everledger – How blockchain secures luxury goods; 2017. https://www.altoros.com/blog/a-close-look-at-everledger-how-blockchain-secures-luxury-goods
66. Jensen T, Hedman J, Henningsson S. How TradeLens delivers business value with blockchain technology. *MIS Quart Execut*. 2019;18(4):221-243.
67. OpenPort; 2021. https://openport.com/
68. Jiao Y, Wang P, Niyato D, Suankaewmanee K. Auction mechanisms in cloud/fog computing resource allocation for public blockchain networks. *IEEE Trans Parallel Distrib Syst*. 2019;30(9):1975-1989.
69. Hassan MU, Rehmani MH, Chen J. DEAL: differentially private auction for blockchain-based microgrids energy trading. *IEEE Trans Serv Comput*. 2019;13(2):263-275.
70. Galal HS, Youssef AM. Verifiable sealed-bid auction on the Ethereum blockchain. Proceedings of the International Conference on Financial Cryptography and Data Security; 2018; Springer, New York.

AUTHOR BIOGRAPHIES

Viet Hoang Tran is currently a postdoc researcher in the Cloud and Large-Scale Computing group at UCLouvain, Belgium. He finished his doctoral study in 2019 at UCLouvain on the research topic: "Measuring and Extending Multipath TCP" under the supervision of Prof. Olivier Bonaventure. He received his engineering and master diploma in Data Communication and Computer Networks, SoICT, Hanoi University of Science and Technology, Vietnam. His research interests include computer networking, operating systems, and distributed systems.

Bernard Lenssens is founder and Chairman of Codit. In 2000 he started Codit with only 1 thing in mind: making enterprise integration better, cheaper, and faster. Bernard has over 20 years of experience in business integration. He combines his extensive "on the field" experience with an expert vision when defining how large corporations should integrate enterprise systems and B2B business processes. Bernard is currently Chief Innovation Officer, he makes sure Codit offering remains in sync with the needs of our customers today and beyond. He focused the last years on starting up an IoT expertise domain within Codit.



Ayham Kassab is currently a postdoc researcher in the Cloud and Large-Scale Computing group at UCLouvain, Belgium. He obtained his Masters and PhD in green and energy efficient High Performance Computing (HPC) from the University of Bourgogne Franche-Comte, France, in 2016 and 2019, respectively. His research interests include blockchains, performance benchmarking, distributed systems, and green computing. He is currently involved in the FairBCaaS project.

Alexis Laks joined Codit as a consultant in 2020 after graduating from Ecole Polytechnique with an AI specialization. Alexis plays the role of Machine Learning Engineer in data science projects at Codit as well as a project manager. Having a background in business he also acts as a business developer to bring companies to leverage the value in the data they hold, especially in IoT enabled environments. Lately he has focused on developing an MLOps approach to AI development within Codit, in order to deliver maximum value quickly and efficiently. Alexis plays the role of project manager for the FairBCaaS project for Codit.

Etienne Rivière is a professor of Computer Science and Systems at UCLouvain. He obtained his PhD from University of Rennes and Inria, France, in 2007. Before joining UCLouvain in 2017, Prof. Rivière has been an ERCIM post-doctoral fellow at NTNU, Norway, and a lecturer at the University of Neuchâtel, Switzerland. His general research interests revolve around distributed systems, operating systems, dependability, and security. He currently leads the cloud computing and large-scale distributed systems research group at UCLouvain. The group is active on several projects around blockchain systems, programmability of distributed cloud systems, secure smart environments, and resource management for big data infrastructures.

Guillaume Rosinosky is a postdoctoral researcher in the Cloud and Large-Scale Computing group at UCLouvain, Belgium. He obtained his PhD in 2019 from Université de Lorraine, France for his work on the elasticity of BPM engines in the Cloud. Before that, he worked for a period of 10 years as a developer and lead developer in the industry. His research interests include distributed systems, optimization, and machine learning, and he is widely interested in the performance, security, and elasticity of large scale systems.

Ramin Sadre is a professor of Security and Performance of Networked Systems at UCLouvain. Before that he was an assistant professor at Aalborg University and a postdoctoral researcher at the University of Twente. His research interests include the design and security of the Internet of Things and industrial control systems, in particular performance aspects and the detection of intrusions.

How to cite this article: Tran VH, Lenssens B, Kassab A, et al. Machine-as-a-Service: Blockchain-based management and maintenance of industrial appliances. *Engineering Reports*. 2022;e12567. doi: 10.1002/eng2.12567