Cyber-Physical Contracts in Offline Regions

Lars Creutz
Institute for Software Systems
Trier University of Applied Sciences
Birkenfeld, Germany
Email: l.creutz@umwelt-campus.de

Kevin Wagner
Trier University of Applied Sciences
Birkenfeld, Germany
Email: s14ac0@umwelt-campus.de

Guido Dartmann
Institute for Software Systems
Trier University of Applied Sciences
Birkenfeld, Germany
Email: g.dartmann@umwelt-campus.de

***Abstract*—LoRaWAN is primarily used in the field of Internet of Things (IoT) applications to transmit sensor values at low cost and build applications on that data. However, LoRaWAN is not commonly used to actively participate in applications. In this paper, we present a universal approach to integrate LoRaWAN into larger cyber-physical systems. For this purpose, we developed an abstraction for Fides, a framework for decentralized cyber-physical contracts, and show how to participate in those contracts with LoRaWAN, even in offline regions without Internet connectivity. Therefore, we present both the necessary extensions to Fides itself, as well as the LoRaWAN protocol used for communication. These components are complemented by a secure middleware, which not only translates the LoRaWAN transactions and acts on behalf of the users but also provides information about the contracts back to the devices.**

1. Introduction

In this paper, we present a system that is particularly suitable for the digitization of structurally weak regions. On the application layer, the system is based on the software Fides, which we published in [1]. Fides is based on decentralized networks and allows the exchange of digital contracts on inexpensive (low performance) hardware with low energy consumption. In this work, we extend the Fides framework with a LoRaWAN[1] abstraction that allows to participate in contracts without an Internet connection, using a Raspberry Pi[2] as hardware platform.[3] We describe a LoRaWAN protocol, which is the basis for the abstraction, and show our approach for a middleware that translates the transmitted data over LoRaWAN on behalf of the users. Since LoRaWAN uses license-free radio resources, the combination of Fides, LoRaWAN and a secure middleware to translate and exchange the messages is a solution for many regions in the world to digitize processes and to develop new business models, even without a permanent internet connection. The approach can also be applied to other applications and may be used as a blueprint to utilize LoRaWAN in an equivalent way in other domains.

A basic principle of Fides is that it is based on natural language, so any user, regardless of their technical expertise in programming and smart contracts, can use it to create digital agreements. Fides implements the *Cypher Social Contracts* [2] concept and focuses on self-organization, decentralization and social participation. The contracts in Fides are derived from templates and then, one by one, each responsible party confirms the tasks defined in the template to change the state of a contract inside the network. Any task can be human readable, which also allows non-technical experts to use the framework.

We begin in Section 2, where we explain our concept in more detail and how it differs from related works. In Section 3 we explain some core principles of Fides, describe our abstraction to use the system with LoRaWAN and discuss our experimental setup including a performance evaluation. Section 4 goes into detail about the protocol, its design principles and introduces a scenario that illustrates the communication and the protocol. In the following Section 5, we describe our approach to the middleware in order to translate the messages for the respective users in a secure manner. In the last section, we summarize our work and the components presented, and describe future efforts to apply the results to other device types and further evaluate the system in additional experiments in offline regions.

2. Contribution and concept

LoRaWAN is used for applications in the IoT sector, where it is frequently used to transmit sensor values. The authors in [3] provide a systematic review of 71 LoRaWAN use cases. The primary areas of application of the works studied were in the fields of smart cities, farms and grids, as well as healthcare. In general, it is primarily about monitoring sensor values in different contexts, with applications built upon the data. Some works such as [4] and [5] implement message-based services, however they are not part of The Things Network [6] (TTN) we use. To the best of our knowledge, no work addresses the usage of LoRaWAN as an abstraction to use an existing application.

1. LoRaWAN is a protected trademark of Semtech. https://lora-alliance.org/ - Accessed April 2022
2. https://www.raspberrypi.com/ - Accessed April 2022
3. Our changes are included in Fides and available open source under the MIT license: https://gitlab.rlp.net/l.creutz/fides

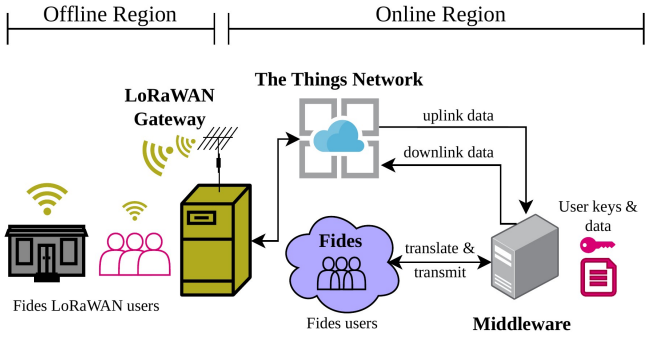

Figure 1. Overall structure of the system.

We understand an abstraction as the active participation in a system through LoRaWAN. To illustrate this, we look at [7], where a smart parking system was implemented that also uses LoRaWAN. As in other works, LoRaWAN is used to transmit sensor data describing the respective state of the parking space. However, LoRaWAN cannot be used to reserve parking spaces, which would be possible with our approach. Furthermore, only a few papers exist that bring LoRaWAN and (smart) contracts together. One paper [8] tries to aggregate pollution measurements from LoRaWAN sensors in combination with the Ethereum Blockchain. The work uses LoRaWAN devices and gateways with Ethereum light clients for this purpose. The work can be seen as a first demonstration, however, there are still open research questions to deploy this kind of system in the real world.[4]

Often the low data rate of LoRaWAN is problematic to be able to offer complex functions within an application, which would require significantly more data to be transmitted than the fair access policy [9] of The Things Network [6], which limits the uplink airtime to 30 seconds per day/device and the number of downlink messages to ten, and LoRaWAN in general [10] allows. Our contribution is novel in terms of user interaction by using LoRaWAN to participate in cyber-physical contracts. We have extended the implementation of Fides with an abstraction for LoRaWAN, which makes it possible to establish digital contracts in offline regions and thus enable digital participation and self-organization. Our approach, illustrated in Fig. 1, consists of translating LoRaWAN data into Fides transactions by our middleware and sending those transactions to a decentralized Fides network in the name of the user. We consider offline regions where users with Raspberry Pis and the abstraction for Fides communicate over the air with LoRaWAN gateways. Here, the gateways themselves, but not the users, have an active Internet connection. The LoRaWAN gateways forward the received messages to The Things Network, from where they are transmitted to our middleware. The middleware translates the messages and performs the respective action on behalf of the users within the Fides network. Other institutions/users are directly connected to the Fides network and receive the messages as if they originated from the offline devices themselves. In order to obtain information from the Fides network, the middleware can also transmit messages to the gateways via The Things Network when requested by the users, which are then sent back to the devices in the offline region using LoRaWAN downlink messages. Individual data is managed via per-user placeholders in the middleware, which replaces the placeholders with the user-specified values within the Fides transactions. In order to transmit information back to the LoRaWAN device, we also send targeted downlink information to the respective device via the middleware, for example to inform about the updated state of a contract. The downlink information contains only general information about the contracts and templates, not the entire Fides transactions because of the limited bandwidth [9], [10]. The extent to which the system is nevertheless useful despite this limitation will be addressed in Section 4.4. Now that we have briefly sketched our approach, the following sections describe the different components and explain both the necessary steps to translate from Fides to LoRaWAN messages and how the middleware operates.

4. The smart contract code provided in [8] for example does not check if the caller is the owner of the contract which allows anyone to manipulate the pollution limits to be monitored.

3. Fides with LoRaWAN

In this Section we present our abstraction to be able to use Fides with LoRaWAN. For this purpose, we first discuss core concepts of Fides so that the general procedure of the abstraction is more comprehensible. We then describe the abstraction and the hardware for which it was implemented and based on which a performance measurement was performed.

3.1. Fides

Fides [1] is the reference implementation and extension of the stated *Cypher Social Contracts* protocol [2] that allows to create digital contracts using natural language and focuses on security and privacy of the users. The contracts are stored inside open, distributed peer-to-peer networks. In order to describe how we can create digital contracts in offline regions using only LoRaWAN, we need to explain some previously mentioned core elements of Fides in more detail:

Templates. Inside Fides, a template is the base for any contract and defines the contents of the agreement. It contains the task definitions, validators assigned to a specific task, who receives the contract (the user that created the template) and more information that is used by the software. Because of the limited bandwidth [9], our protocol cannot support the creation of templates, only the creation of contracts. We also assume that the creators of templates have a regular Internet connection, as it would not be functional to obtain contract offers via LoRaWAN due to the same limits [9]. Any task to a template is a regular string object that can

TABLE 1. ELEMENTS OF A TEMPLATE INSIDE FIDES

Element	Description
$\mathcal{H}_{\mathcal{O}_j}$	Hash of the template (SHA256)
$\mathcal{K}_{j,P}$	Public key of user j
B	Description of the template
T	List of tasks
R	Mapping of who is responsible for which task
r_M	Merkle root hash of the tasks
V	Validators assigned to the tasks

TABLE 2. ADDITIONAL ELEMENTS OF A CONTRACT INSIDE FIDES

Element	Description
$\mathcal{H}_{\mathcal{C}_{i,j}}$	Hash of the contract (SHA256)
$\mathcal{K}_{i,P}$	Public key of user i
N	Number of tasks

be human readable. A template $\mathcal{O}_j$, created by user j, is defined as:

$$\mathcal{O}_j = (\mathcal{H}_{\mathcal{O}_j}, \mathcal{K}_{j,P}, B, T, R, r_M, V) \tag{1}$$

Table 1 explains the elements of a template in more detail. The owner of the template has the option to either accept or reject the contracts that refer to that template. A template is either in an *ACTIVE* or *INACTIVE* state, thus the owner can prevent a template from existing indefinitely, which allows the implementation of time-limited offers and services.

Contracts. A contract is always derived from a template, therefore contains several references to it, and is defined as:

$$\mathcal{C}_{i,j} = (\mathcal{H}_{\mathcal{C}_{i,j}}, \mathcal{K}_{i,P}, \mathcal{K}_{j,P}, N, R, r_M, \mathcal{H}_{\mathcal{O}j}) \tag{2}$$

The contract is an agreement between two parties i and j, where i creates the contract and j manages the template. Every contract is in a state which can be implicitly changed by transactions. Within Fides, so-called *full nodes* take care of the documentation of this state in the network. States for a contract, which are self-explanatory, are *OFFER*, *LIVE*, *REJECTED* and *FINISHED*. In addition to the definitions introduced, both template and contract also contain a nonce to ensure the uniqueness of the object. Table 2 explains the elements of a contract that were not introduced before. In order to create a contract, the user must have access to the template that should be used. The contract creation process can be boiled down to the following steps:

1) Load the template.
2) Set the elements that are reused from the template inside the contract object.
3) Create a random nonce to ensure the hash of the contract is unique.
4) Add the contract creators public key to the contract.
5) Calculate the hash of the contract.
6) Create a set of ephemeral public/private key for that contract.
7) Perform a check of the contract in combination with the template.

Inside Fides the hash of a contract between two parties i and j is calculated as follows:

$$\mathcal{H}_{\mathcal{C}_{i,j}} = \textbf{SHA256}(\mathcal{K}_{i,P} \oplus \mathcal{K}_{j,P} \oplus N \oplus r_M \oplus \mathcal{H}_{\mathcal{O}_j} \oplus R \oplus nonce) \tag{3}$$

$\oplus$ denotes the concatenation. The important part to create a unique contract hash is the value of the cryptographic nonce, which is a 4-byte unsigned integer value. We use this approach because it allows us to only transmit a subset of information and re-create that same contract when translating between LoRaWAN messages and Fides given that the public key $\mathcal{K}_{i,P}$ is the same for that contract and the template is present at the middleware.

3.2. Fides LoRaWAN abstraction

We have extended the implementation of Fides with a *lora* command to create a clear separation between the core components and our LoRaWAN abstraction for Fides. In the following, we will discuss the important components and take simple functionalities like saving the contracts as given.

Hardware and connection. First, it must be ensured that the hardware used can connect to the LoRaWAN network. For this purpose, a component manages the configuration of the user, such as *application keys*, *application/device EUIs* or *data rate* to establish a connection. When Fides is started, the abstraction securely connects to the LoRaWAN network once by Over-the-Air Activation (OTAA) [11]. If a connection cannot be established, or it is not possible to transmit a LoRaWAN message while using the system, the user is notified immediately so that another attempt can be made.

Contracts. Another main component is a class for managing contracts that are processed via LoRaWAN. This is based on the core contract component of Fides and contains adaptations that implement the protocol described in Section 4. An important extension are so called blind operations, which we introduced to save bandwidth. Blind operations can be used to change the state of a contract locally and to pretend the new state without obtaining it via LoRaWAN downlink. This approach is particularly helpful when users can observe tasks of the contract in real life. For example, if the contract describes a delivery that has been received by the user, the user does not have to obtain this information over the network and use the limited bandwidth for it.

Accounts. When we create a contract, we assume that the user has access to the private key, whereas this is not necessary within the abstraction, since we only need to know the public key $\mathcal{K}_{i,P}$ that belongs to the private one maintained at the middleware in order to create the same contract hashes, which increases the security of IoT devices. While contracts could be created by accessing the hardware, there is no direct access to the private key and the API key used within The Things Network [6] to authenticate at the middleware. Furthermore, the current placeholders can only be retrieved via the middleware. If an attacker has

access to the LoRaWAN device, the context of use cannot be classified. The used templates can also be obfuscated, so that the use case is only known to the parties involved and any attackers cannot draw conclusions via the metadata. Thus, the overall security of Fides and the integrity of the data, given that the devices are appropriately monitored and unauthorized access is detected, is not compromised.

Templates. In order to be able to create contracts at all, the templates must be available on the respective device as well as on the middleware. The templates do not necessarily have to be obtained via the Internet and can therefore also be transferred offline to the devices. For users who cannot bring their devices to the Internet in any way, the middleware provider can act as a service provider to provide the LoRaWAN hardware, the Fides software, and the templates offered there. We address this in more detail in Section 4.4.

Additional implementations. In addition to the components already described, the abstraction also contains functions for converting from and to LoRaWAN messages. For this purpose, we defined the protocol messages using protobuf[5], which is already used inside the core components of Fides. To make the system more understandable to the users, the abstraction also displays the number and amount of data sent and received per usage. This makes it possible to understand why, after a certain number of messages, no more data can be sent or received without waiting long enough.

Integration with Fides. The new functions of the abstraction should seamlessly integrate into the core system of Fides, so that it is easily possible to switch to the regular configuration, provided that an Internet connection becomes available on the devices. For example, to import all contracts that were created through the middleware via LoRaWAN, it is enough to execute the following *fds*[6] commands:

```
fds lora contract list \
| xargs -n 1 fds contract fetch
```

In order to be able to access the encrypted data, the temporary keys, which were generated in the sixth step during contract creation by the middleware, must be imported to be able to manage the contract locally from now on.

Differences to Fides. The implementation of the abstraction differs from the core system Fides in several points, which will now be discussed. Since the transactions cannot be obtained in a whole over LoRaWAN, the implementation of the contracts only documents their general information, i.e., their state, the hash of the contract, which task was processed, and how they need to be addressed over LoRaWAN. Further, communication only takes place through a direct action of the users. In order not to exhaust the already low bandwidth too quickly, no background updates of the contracts are performed. We only check new downlink messages when users are actively sending uplink messages, which is the general approach in LoRaWAN [10]. Furthermore, the abstraction extends Fides with an alternative form of transactions that send the protocol's messages to the communication daemon *fidesd* [1], from where they are translated and transmitted to the network.

3.3. Hardware configuration

Our implementations were evaluated and tested on a Raspberry Pi 3 and 4. For our LoRaWAN abstraction of Fides we used the "IoT LoRa Raspberry Pi Node pHat" from PiSupply[7] with the corresponding RAK811 library[8] for firmware version 2. Further combinations of hardware/software/firmware could be easily integrated within the abstraction. The module uses (due to our location in Europe) the channel frequency of 868Mhz [12]. We use data rate 5 as the default within the abstraction, which corresponds to a spreading factor of 7 [12]. According to the airtime calculator [13], this corresponds to a number of 486 messages per 24 hours with an application payload of 11 bytes, which is the largest message in our protocol that is used to create a contract. Within Fides, the minimum size of a transaction to publish a contract is about 650 bytes. The largest possible payload of a LoRaWAN message at the selected data rate is, according to [13], 222 bytes. Thus we would need at least 3 uplink messages per contract and can create a maximum of 27 contracts per day. Other data rates additionally allow much smaller payloads, for example data rate 3 allows 128 bytes and data rate 0-2 only 64 bytes [13]. Furthermore, if we were to obtain whole transactions via downlink, we could not control the size of those. Fides limits the maximum transaction size to 1 MB [1], which means that a transaction from another party with this size would have to be split into almost 5000 downlink messages.

3.4. Performance evaluation

In order to show the performance of Fides in general and our abstraction, we performed a measurement in which we created and processed ten contracts. During this period, about 50 uplink messages (400 bytes of data) and about 35 downlink messages (270 bytes of data) were sent. Although the actual downlink limit of TTN [9] was exceeded by a factor of 3.5 in our experiment, the limit was not enforced. The measurement only shows a short period, but it shows the low resource consumption in general, which would not be higher in a bigger timespan, since the system only transmits data through active sending by users and does not use background updates to not reach the strict limits of TTN [9] too quickly. We measured CPU and RAM utilization of our Raspberry Pi 4 with 4GB of RAM in a time span of about half an hour. Our results are shown in Fig. 2. We performed the measurement with top[9] and recorded CPU and RAM

5. https://developers.google.com/protocol-buffers - Accessed April 2022
6. *fds* is the command line interface of Fides
7. https://github.com/PiSupply/IoTLoRaRange/ - Accessed April 2022
8. https://github.com/AmedeeBulle/pyrak811 - Accessed April 2022
9. https://man7.org/linux/man-pages/man1/top.1.html - Accessed April 2022

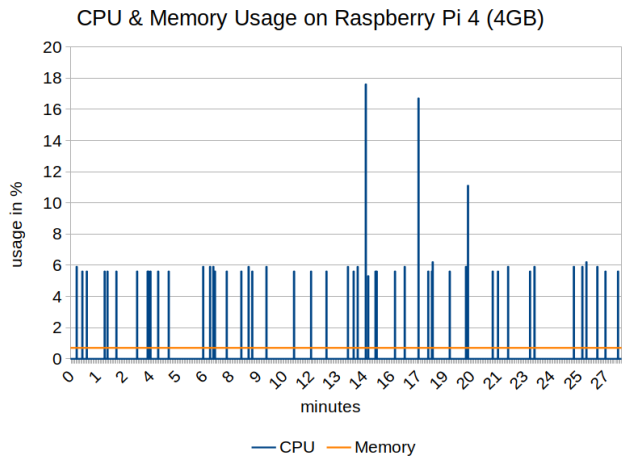

Figure 2. Overall CPU and RAM utilization of the Fides process during our experiment. The average CPU usage was 0.2% and RAM usage 0.7%.

usage in 1 second intervals. Spikes in the CPU usage were caused by downlink information that did change the state of the local contract objects. However, other downlink data of similar type did not cause the CPU to spike that high. The measurements are like those made in [1], where the performance of Fides was analyzed for regular systems. Due to the low average memory consumption (0.7%) and the low average CPU load (0.2%), Fides is a resource efficient solution for regions with limited infrastructure: it has a low energy consumption, low memory usage, low processing complexity and is efficient regarding the required data rates.

4. Fides LoRaWAN protocol

In order to allow the usage of Fides in combination with LoRaWAN, we need to implement a small protocol so that our messages can be translated to Fides transactions without creating too big uplink messages that would conflict with the fair access policy [9] of the used The Things Network [6]. The goal of the protocol was to minimize the message size while still being able to recover from errors by transmitting larger messages in rare occasions. In general, it is recommended that the payload per message does not exceed 12 bytes [14]. The largest message to be sent by our protocol is usually 11 bytes. The average size is 6 bytes for uplink messages and 7 bytes for downlink messages.

4.1. Available commands

We use the first byte of the message for the command. Here we consider a bidirectional communication, so the command is not only an instruction from the client to the middleware, but also the response of the middleware in the form of a downlink message. Table 3 describes the possible commands and their arguments of any LoRaWAN transaction inside our abstraction.

TABLE 3. PROTOCOL COMMANDS

Command	Arguments	Description
00	-	Ping message
01	Nonce (4 byte)	Create a new contract offer
02	-	Confirm a task (empty)
03	Placeholder (1 byte)	Confirm a task (placeholder)
04	-	Get the state of a contract
05	-	Get the state of a template
06	-	Contract is an offer (downlink)
07	-	Contract hash abbreviation is ambiguous (downlink)
08	-	Template is active (downlink)
09	-	Template is inactive (downlink)
0a	-	Template hash abbreviation is ambiguous (downlink)
0b	-	Delete contract at middleware
0c	-	Contract not found (downlink)
0d	-	Template not found (downlink)
0e	Current task (1 byte)	Contract is live (downlink)
0f	-	Contract was rejected (downlink)
10	-	Contract is finished (downlink)

4.2. Addressing objects

Because we are not able to continuously send 32 bytes of data (the size of a SHA256 hash) to identify an object inside Fides, we only send a subset of this hash (the first 6 bytes). This approach is similar to the behavior of git when addressing commits with an abbreviation of the hash [15]. The object identifier is always the last element of a transaction, which allows us to send bigger messages (without including the size of the message) if any error is reported from the middleware service. The protocol allows smaller messages up to the point where conflicts arise due to ambiguities that can be resolved by sending a larger portion of the identifier.

4.3. Example scenario

In order to understand the communication, we introduce the following scenario and look at the creation and processing of a contract using an example. We assume that two tasks are defined in the template. The first task must be confirmed by the party i that creates the contract $\mathcal{C}_{i,j}$ and the second task by the party j that manages the template $\mathcal{O}_j$. We further assume that party i, which creates and processes the contract through LoRaWAN, has access to the middleware. The registration with the middleware is described in Section 5. The necessary steps to be performed amount to:

Template import. The template can be imported either using the transaction data offline (e. g. on a USB drive) or by using the device with Fides in an online region.

Contract creation. The imported template is now used to derive the contract $\mathcal{C}_{i,j}$. For this purpose, the value of the automatically generated *nonce* determines the hash of the contract. Accordingly, this *nonce* must be transmitted to the middleware so that the same contract is created there that

was created locally. We assume *nonce* = 4075087775 and Template $\mathcal{O}_j$ = c40164f1c121[...]. The nonce, converted in hex format (big endian) is f2e4e79f, which results in the following uplink message:

```
01 f2e4e79f c40164f1c121
```

This results in $\mathcal{H}_{\mathcal{C}_{i,j}}$ = 1f9bdd63faed[...][10] which is also recreated at our middleware. After the middleware has created the contract and published it on behalf of the user, the contract is in state *OFFER* and must now be either accepted or rejected by the other party. For our example we use a blind operation and change the state of the contract to *LIVE* without any further up- and downlink messages.

Task confirmation. In order to be able to confirm a contract step using LoRaWAN, we distinguish between two procedures:

1) Confirmation without user input (empty).
2) Confirmation with data (placeholder).

Thus, either no input data or the value of a previously defined placeholder, which can be managed per user on the middleware, is used. There are up to 256 placeholders (1 byte) available per user, which can be arbitrarily chosen. We assume for our example that the party i passes the first placeholder as content into the confirmation of the task:

```
03 00 1f9bdd63faed
```

Contract updates. In order to update a contract, user i can either use a blind operation and assume that the action of the other party j was performed, or send a dedicated uplink message to receive the newest state of the contract via downlink through the middleware. In our example the uplink payload would be:

```
04 1f9bdd63faed
```

At this point we preempt the data transfer screenshot that follows later and assume the following as the downlink message:

```
0e 02 1f9bdd63faed
```

The middleware reported that our contract is in state *LIVE* and the second task, which party j is responsible for in our example, needs to be confirmed. This underlines that the blind operation did work and we could save up- and downlink bandwidth. Any state of a contract, which will be provided via downlink information by our middleware, needs at most two bytes, where the first byte describes the state and the second (on a live contract) the current task. Middleware feedback in the form of downlink is always considered the ultimate truth and overrides the current local state of the object, including any wrong blind operation.

10. The hash value refers to a real-world example, which will be emphasized later by Fig. 3, Fig. 4 and Fig. 5 that show the middleware and The Things Network data.

4.4. Limitations

As already explained, we can't receive the entire transaction from the other party in a reasonable way due to the limits in bandwidth [9]. This also includes text-inputs of the other party from the agreement. In our opinion, it is also more important for users in offline regions to be able to send more data than to receive it, in order to increase their own digital participation. However, since the use of Fides with LoRaWAN is a special case for a small set of users, the templates for such services can also be tailored for the usage in such a region. For this purpose, a uniform but unofficial format can be specified, which is adapted by many different templates. For example, an overall format could be defined for how addresses must be transmitted. With the possibility of validators in the templates, not only can automation take place on the part of all service providers, but all users of the LoRaWAN abstraction would only need a single placeholder to transmit their address data, regardless of which template they want to use. The biggest bottleneck is the limit of ten downlink messages due to the fair access policy [9]. We would like to reinforce that it was possible to obtain more than ten downlink messages in our experiments. However, since this contradicts the official specification and may be enforced by the network in the future, we have to assume the limit for the design of our protocol. We do not push this limit locally, so users can try to retrieve more than ten downlink messages. The possibility of blind operations additionally compensates for the strict downlink limit. Nevertheless, it should still be possible to create and process many contracts, despite the given limitations.
A much bigger problem would be users who are unable to connect to the Internet with their devices, for example in other regions, in order to be able to install the software and obtain the templates in the first place. The problem could be solved by service providers offering both hardware and software preconfigured. On the hardware side, the devices can thus be prepared for the use of LoRaWAN with TTN and the middleware. On the software side, Fides and the templates offered on the middleware can already be installed on the devices to use the system immediately in the offline region.

5. Middleware

Our middleware implements the translation of the previously introduced protocol and acts on behalf of the users. The specification is part of the open source release of the LoRaWAN abstraction for Fides and can be used within any context. The actual middleware does not necessarily have to be developed for multiple users. In our context, we run the middleware as an extension of a marketplace for local services where verified service providers create listings that reference their templates inside the Fides network. In order for the middleware to correctly translate the LoRaWAN messages, it must also be aware of those templates. We solve this by using a shared Fides instance between marketplace and middleware.

5.1. The Things Network integration

All our hardware is connected to *The Things Network* (TTN) [6], which provides an open source LoRaWAN network server with gateways all around the globe. Within TTN applications, so-called integrations[11] can be used to process the transmitted data from the devices. Within our middleware we use the webhook integration to forward the uplink data and send downlink messages to the devices. Each webhook is bound to an application in which the LoRaWAN devices of a user are registered. This allows the assignment of different devices to the respective application. To make it easier for regular users, we use TTN's API[12] directly to create our webhook automatically within the specified application. For this purpose, our users must provide the following data during registration:

- Fides private key: The corresponding private key to the public key $\mathcal{K}_{i,P}$ that the middleware should control in the name of the user i.
- LoRaWAN device: ID of the device.
- TTN access key: API key to access the TTN account of the user in order to add our webhook and send downlink data to the device.
- TTN application: The unique ID of the application where the device is registered.

After the given information is validated the webhook is created accordingly. In addition, an API key is generated, which is sent by the webhook to identify the user within our middleware.[13] This API key can also only be used for the public endpoint that is used by the webhook. It is not possible to use it to obtain information about the placeholders, past messages, or to log in to the middleware to access the temporary keys. Fig. 3 shows the information of a registered user that is logged in inside our middleware.

5.2. Translate transactions from LoRaWAN to Fides

The addressed webhook forwards all LoRaWAN messages through the integration to a public endpoint of the middleware, which is authenticated by the API key that was generated upon webhook creation. Then we verify that the message is from the device that the user specified during the registration. This ensures that several devices can exist within the same application, but that not all of them must be connected to Fides. All user private keys are stored encrypted within a database and are not part of the local Fides instance, instead they are dynamically loaded at runtime to process the respective request. In order to work on the received data in a reasonable way, we first convert the received payload into bytes, which are translated

11. https://www.thethingsnetwork.org/docs/applications-and-integrations/ - Accessed April 2022

12. https://www.thethingsindustries.com/docs/reference/api/ - Accessed April 2022

13. When creating the webhook, the HTTP header *X-API-KEY* is added with the automatically generated API key for the respective user.

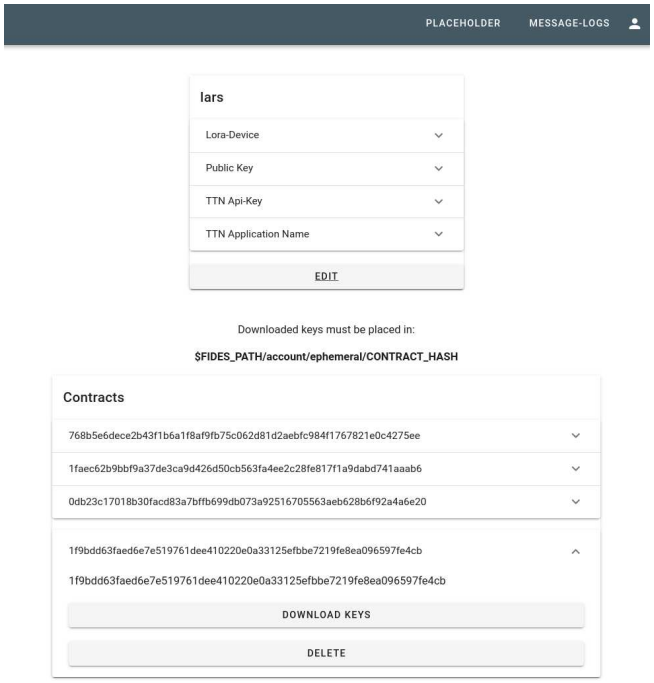

Figure 3. Main overview for any registered and logged in user. It contains the username, the added information of TTN properties, the used device and the public key of the Fides account that is managed at the middleware. Also, we display the created contracts so that the user can delete them from our middleware and download the ephemeral keypairs that were generated when creating the contracts.

by the abstraction of Fides into a protobuf message that is processed afterwards. To understand the creation of a contract, let us follow the example from before and look at the creation of the contract $\mathcal{C}_{i,j}$ from the middleware's point of view. An overview of the transmitted and received data of the LoRaWAN device can be seen in Fig. 4. Fig. 5 shows all uplink data of the device and the translation by the middleware to get a simple overview of the sent data within the application context. The first message received describes the creation of a new contract, as already illustrated before. To create the contract on behalf of the user, the middleware performs the following actions:

1) **Template check**: If the template is unknown, inactive or if the hash is ambiguous, respond with the corresponding downlink message.
2) **User identification**: Load the private key that matches the sent API key from the webhook.
3) **Contract creation**: Create the contract in the name of the user and use the same nonce.
4) **Contract publication**: Publish the contract in the name of the user. If an error occurs, the contract is immediately deleted locally and republished by the user after realizing that a request regarding the state of the contract would indicate that the contract is not found.

Now that a contract has been created and published,

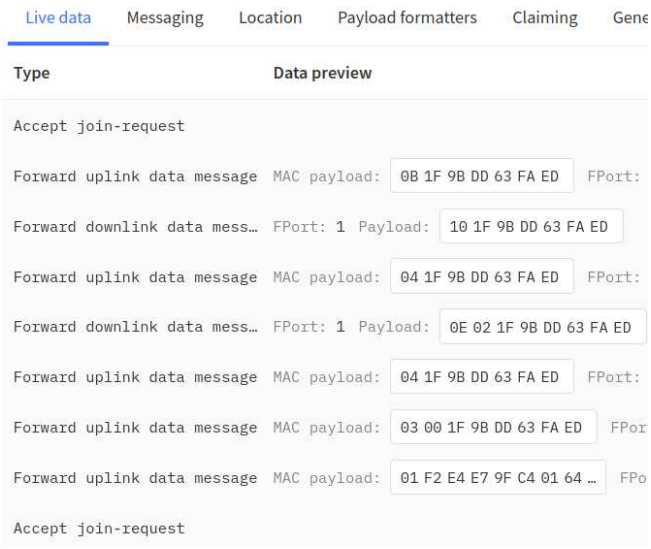

Figure 4. The Things Network live data of the LoRaWAN device. Every up- and downlink message is shown (from bottom to top).

Figure 5. Uplink messages and translations (top to bottom) for the user at the middleware.

we see in Fig. 4 that the next uplink message is already a confirmation of the contract step. This would not be possible in the regular case, since each contract offer must first be accepted by the other party. In our case we used an automated template where we know that our contract will be accepted, and thus used a blind operation to save bandwidth. In order for the middleware to confirm a contract step, it must perform the following tasks:

1) **User identification**: Load the private key that matches the sent API key from the webhook.
2) **Load contract**: Load all contracts that could be referenced by the hash abbreviation within the message. For each potential contract, check if it is from the same user identified in the first step. If no contract was found, or the given hash abbreviation is ambiguous, reply with the corresponding downlink message.
3) **Task confirmation**: Confirm the current task on behalf of the user. If a placeholder is used, replace the value within the transaction.

The next message is a request to receive the current state of the created contract. This is also reflected in Fig. 4 where we can observe that immediately after our uplink was sent, the middleware scheduled a downlink response that contains the state of the object. As described earlier we received the information that the contract is still live and the current task that needs to be confirmed is task two. In our example scenario that task was the last task of the contract and the other party was responsible to confirm it, therefore we waited and sent the exact same message to query the state of the object again and received different downlink data, indicating that the contract is now finished and the other party did confirm the last task. The final uplink message then instructs the middleware to delete the finished contract. In our experiments, the downlink was always sent immediately, which means that one can obtain the downlink message on the device immediately after the uplink message. Nevertheless, it could happen that the downlink only becomes known to the device with delays. The user can simply send an empty ping message to the middleware that allows the device to check for new downlinks locally after using only a single byte of uplink data for the ping command.

5.3. Additional functions

In addition to the actual function of translating the messages, our middleware includes other useful functions. On the one hand, registered users can specify their own placeholders directly on our platform. Here, a simple view compares the respective placeholders with the values to be replaced. On the other hand, users can view all their sent messages via the message log. Messages that are only temporarily available in the TTN live data view are stored on our server permanently. In addition, we list the created contracts and provide the possibility to download the temporary keys per contract and delete the contract in our Fides instance directly from the middleware which can be seen in Fig. 3.

6. Conclusion and future work

We have presented our approach on how to create a complete system that allows LoRaWAN to be deployed in higher application areas. We gave a brief overview of Fides and what is necessary to abstract the creation of contracts with the smallest possible amount of data. In the following, we presented our protocol with several examples

and explained design decisions of the protocol that allow us to make a complex use case like cyber-physical contracts possible in offline regions. It is worth mentioning that the protocol can exist in the real world and it is not just an edge case demonstration, as the regular message size does not exceed 11 bytes, allowing a large number of uplink messages per day per device. Any conflicts in addressing objects can be resolved by the protocol and the developed abstraction for Fides by allowing the message to grow dynamically if necessary. In addition to the abstraction and the protocol itself, we then described our approach to implement a middleware, which translates the LoRaWAN messages accordingly and acts on behalf of the users to perform the actions within the Fides network. Our implementation, as well as the specification of the middleware are open source available and can therefore be used in any project. In the future, we will evaluate our implementation in further experiments, which should reinforce the application capabilities in real-world environments. Furthermore, we aim to bring our protocol to more low-level devices like microcontrollers that do not support Fides.

Acknowledgment

This work has been funded by the Federal Ministry for Digital and Transport (BMDV) within the research initiative "mFUND" under the grant number 19F2102F.

References

[1] L. Creutz, J. Schneider, and G. Dartmann, "Fides: Distributed cyber-physical contracts," in *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, 2021, pp. 51–60.

[2] L. Creutz and G. Dartmann, "Cypher social contracts a novel protocol specification for cyber physical smart contracts," in *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2020, pp. 440–447.

[3] L. R. de Oliveira, P. de Moraes, L. P. Neto, and A. F. da Conceição, "Review of lorawan applications," *arXiv preprint arXiv:2004.05871*, 2020.

[4] A. Cardenas, K. Nakamura, E. Pietrosemoli, M. Zennaro, M. Rainone, and P. Manzoni, "A low-cost and low-power messaging system based on the lora wireless technology," *Mobile Networks and Applications*, vol. 25, 06 2020.

[5] J. Höchst, L. Baumgärtner, F. Kuntke, A. Penning, A. Sterz, and B. Freisleben, "LoRa-based Device-to-Device Smartphone Communication for Crisis Scenarios," in *17th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2020)*, Blacksburg, Virginia, USA, 2020.

[6] The Things Industries, "The Things Network," [Online]. Available: https://www.thethingsnetwork.org/ (Accessed April 2022).

[7] J. J. Barriga, J. Sulca, J. León, A. Ulloa, D. Portero, J. García, and S. G. Yoo, "A smart parking solution architecture based on lorawan and kubernetes," *Applied Sciences*, vol. 10, no. 13, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/13/4674

[8] S. R. Niya, S. S. Jha, T. Bocek, and B. Stiller, "Design and implementation of an automated and decentralized pollution monitoring system with blockchains, smart contracts, and lorawan," in *NOMS 2018-2018 IEEE/IFIP network operations and management symposium*. IEEE, 2018, pp. 1–4.

[9] The Things Network, "Duty Cycle," [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/ (Accessed April 2022).

[10] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, "Understanding the limits of lorawan," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 34–40, 2017.

[11] The Things Network, "Over-the-Air Activation (OTAA)," [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/addressing/#over-the-air-activation-otaa (Accessed April 2022).

[12] LoRa Alliance, Inc, "LoRaWAN 1.0.3 Regional Parameters," [Online]. Available: https://lora-alliance.org/wp-content/uploads/2021/05/RP002-1.0.3-FINAL-1.pdf (Accessed April 2022).

[13] avbentem, "Airtime calculator for LoRaWAN," [Online]. Available: https://avbentem.github.io/airtime-calculator/ (Accessed April 2022).

[14] arjanvanb, "Fair Use Policy explained," [Online]. Available: https://www.thethingsnetwork.org/forum/t/fair-use-policy-explained/1300 (Accessed April 2022).

[15] Scott Chacon et al., "Git - gitrevisions Documentation," [Online]. Available: https://git-scm.com/docs/gitrevisions (Accessed April 2022).