TruDaMul: a Trusted Framework for Smart Territories based on Untrusted Data Mules

Mirko Zichichi*†, Luca Serena†, Stefano Ferretti‡, Gabriele D'Angelo†
*Ontology Engineering Group, Universidad Politécnica de Madrid, Spain
†Department of Computer Science and Engineering, University of Bologna, Italy
‡Department of Pure and Applied Sciences, University of Urbino "Carlo Bo", Italy
mirko.zichichi@upm.es, luca.serena2@unibo.it, stefano.ferretti@uniurb.it, g.dangelo@unibo.it

***Abstract*—The rise of Internet-of-Things enables the development of smart applications devoted to improving the quality of life in urban and rural areas, thus fostering the creation of smart territories. Some dislocated areas, however, are underprivileged in the provision of such smart services due to the lack, inefficiency, or excessive cost of Internet access. To surmount these problems, some techniques related to opportunistic networking might come to aid. In this article, we propose a framework that is centered on relying on an untrusted Data Mule to carry data from an offline source to an online destination. In particular, we present a framework that enables communication between different actors and a reward mechanism for their work, based on the use of Distributed Ledger Technologies, Smart Contracts and Decentralized File Storages. The protocol involved in bringing a Client's message online (and getting back a response) is thoroughly explained in all its steps and, then, discussed on the most important trust and security issues. Finally, we proceed to evaluate such a protocol and the whole framework through a series of communication latency tests, an analysis of the Smart Contract usage, and simulations in which buses act as Data Mules. Our results suggest the feasibility of our proposal in a Smart Territory scenario.**

I. Introduction

Smart Cities are an ongoing breakthrough that is gradually pushing a tidal wave of technological change into our daily lives. The generalization of such a term, i.e. a Smart Territory, can be defined as a geographic space that, through the use of digital technologies, pursues as its main goal the generation of more sustainable economic development and a better quality of life [1]. However, not all territories are equal and for some underprivileged ones it is not possible to implement (costly) Smart Cities services due to the very different economic circumstances or due to unavailable, unreliable or too expensive network infrastructures [2], [3]. It has been recognized that what is outside of Smart Cities is often "left behind" [1], [4]. Thus, with the aim of somehow mitigating this novel form of digital divide, some research efforts now focus on Smart Territories that include rural and dislocated locations and that distinctly emerge as an opposition to fully serviced Smart Cities [5], [6]. We argue that what is needed is a set of novel opportunistic solutions, able to dynamically exploit all the resources that can be shared by the territory community. These solutions would foster a plethora of possible services and applications, ranging from the provision of (delay-tolerant) connectivity, smart farming applications, traceability and remote monitoring in the agrifood supply chain, up to structural health monitoring of country roads, bridges and buildings. In such applications, we cannot give wide area network connectivity for granted, and certain networking solutions (e.g. satellite connections) might result as too costly.

Clearly enough, in order to build such a kind of service ecosystem, adequate incentive strategies are needed, i.e. rewarding those users that offer their service to other (being a datum, a wireless relay, etc.), as well as mechanisms that provide an adequate level of trust in data sharing processes. To this regard, the use of distributed technologies, such as Distributed Ledger Technologies (DLTs) and Decentralized File Storages (DFS), and of cryptocurrencies (or tokens [7]) represents a novel and possibly beneficial solution that may foster the provision of smart services in dislocated communities [8]. Anytime a user acts as a middleman and shares/provides a resource, he/she earns some tokens; every time he/she accesses a resource from another peer, he/she consumes some tokens. At the same time, some mechanisms are needed to help communities to deal with possibly untrusted service providers, assuming that, in some cases, trust is brought by (completely or partially) tracing and validating interaction processes through DLTs.

In this paper, we focus on the issues of data transmission in territories where the broadband Internet connection is not taken as granted as in Smart Cities. We propose a framework that enables any Client that finds itself in an offline condition (e.g. being in a no broadband connection area), to send data to any online Server. We designed a Proxy tunneling service between such a Client and a Server enabled by a set of decentralized and distributed systems. In particular, DLTs and Smart Contracts are used for rewarding service providers and validating the processes. Moreover, a Data Mule acts as the ferryman for data retrieved offline to bring it online. In literature, Data Mules are mobile devices that have wireless communication capability and are equipped with a storage for data collection [9]–[11].

This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie International Training Network European Joint Doctorate grant agreement No 814177 Law, Science and Technology Joint Doctorate - Rights of Internet of Everything.
This work has received funding from the University of Urbino Carlo Bo under the BIT4Food initiative

The aim of this work is to propose and validate the framework called *TruDaMul*, with the main goal to add trust to the opportunistic data management activities provided by the Data Mule. In particular, we provide the following contributions:

- A description of the decentralized system and distributed technologies involved in the framework, i.e. DLTs, DFS and the authorization systems;
- A description of the protocol that allows the Clients to communicate with a Server, i.e. through untrusted Data Mules and Proxies;
- An experimental evaluation of the proposed framework based on two phases, i.e., (i) evaluation of Client-Mules offline interactions and Mules-DLT-Proxy online interactions, and (ii) a simulation with the purpose of reproducing actors behavior and then evaluating the communication delay.

The remainder of this paper is organized as follows. Section II provides the background and related works. In Section III the framework together with the main protocol is presented, while in Section IV, we discuss some security considerations. In Section V, we present the experimental evaluation and, finally, Section VI provides the conclusions.

II. Background and Related Work

In this section, we introduce the needed background and describe some related works.

A. Data Mules

Data Mules (that is an acronym for Mobile Ubiquitous LAN Extensions [9]) are mobile devices that consist of a storage device and a short-range wireless communication medium (e.g. Wi-Fi or Bluetooth) and can exchange data to/from a nearby static sensor or access point that they encounter [11]. As a result of their movement between remote areas, they effectively create a data communication link [12].

Data Mules allow for communication and data transfer even in the absence of Internet, and they can be important tools for the functioning of applications concerning the Internet-of-Things (IoT) and in general for services based on the concept of Smart Cities or Villages, where there is a significant flow of data coming from remote areas. Depending on the context, Mules can either be transportation vehicles like buses or cars [13] or even walking persons.

In the last few years, a plethora of works has been presented on Data Mules. For instance, in [9] a study was made with a number of Data Mules performing independent random walks that collect data from static sensors and deliver it to base stations, without forwarding the data to other Data Mules. Other works refer to real vehicular networks use cases, with a focus on routing algorithms for the exchange of messages between Mules and other nodes [12], [14], [15].

We are aware of just one work on the integration of Data Mules and DLTs. It is a system for the exchange of the Ethereum [16] blockchain blocks in a delay tolerant network [17]. However, in that paper the authors do not provide an experimental evaluation of the proposal and do not include DLT-based incentives for the participating actors.

B. Distributed Ledger Technologies

DLTs consist of a network of nodes that maintain a distributed ledger following the same protocol and, in the case of the blockchain, the ledger is organized into chronologically ordered blocks where each block is sequentially linked to the previous one. Thus, DLTs are cryptographically guaranteed to be tamper-proof and unforgeable, enabling the creation of a "trusted" mechanism that can be exploited by multiple users in a distributed environment with no need for third-party intermediaries.

1) Smart Contracts: Smart Contracts are instructions stored in the blockchain and automatically triggered once the default condition is met. We will refer to Ethereum [16] due to its public open source blockchain widespread use and for its provision of robust Smart Contract development tools. The use of Smart Contracts allows anyone to employ DLTs to operate well beyond just currency transactions [18]. For instance, the creation of smart services, based on Smart Contracts, may enable users to interact with devices/vehicles present in smart transportation systems or to favor the interoperability among the devices and resources of Smart Cities [7], [19].

In DLTs such as Ethereum [16], it is possible to build structures through Smart Contracts that act as second layer cryptocurrencies, i.e. tokens [20]. The use of tokens as a complementary monetary system has the potential to create clusters of existing community resources that can be traded with each other, in order to promote Smart Territories [8].

2) State channels for services payments: State channels have been introduced to provide rapid DLT payments without the need to store all transactions on-chain, i.e. directly on the ledger, but mostly off-chain, i.e. outside of the ledger. The state channels protocol can be summarized in a few steps:

- **Opening Channel** - A user U opens a new state channel in a Smart Contract, by depositing an amount of the same token and indicating the other channel party, V.
- **Updating Balance** - Both U and V, now, can communicate off-chain by exchanging digitally signed $balance$ messages. These messages are used to update a balance value between U and V, i.e. if U has to pay V then the balance increases, otherwise the balance decreases.
- **Closing Channel** - Both U or V can close the state channel at any time. The corresponding Smart Contract method needs the copy of the last $balance$ message exchanged and the signature of both parties. Finally, the balance value is deduced from U's deposit in favor of V, while the remaining part is sent to U. A dispute mechanism can be implemented to freeze the transfers.

When U has a channel opened with V and V has one with W, then it is possible for U to pay W through V. This is the main idea behind the Bitcoin's Lightning Network [21] and the Ethereum's Raiden Network [22].

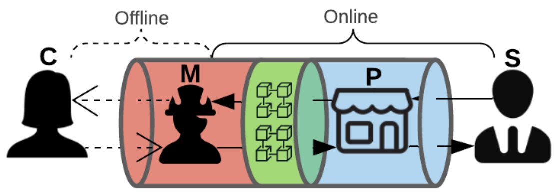

Fig. 1. Overview of the TruDaMul framework. It can be compared with a tunneling protocol where the Client (C) is offline and manages to send data to the Server (S) via Mules (M) and Proxies (P).

C. Decentralized File Storages and Content Addressing

Decentralized File Storages (DFS) enable a content-based addressing approach, where the users, rather than establishing a connection with a Server, query the network asking for specific items. InterPlanetary File System (IPFS) [23] is one of the most used DFS protocols. To identify the items through the Peer-to-Peer (P2P) network that runs the IPFS protocol, a cryptographic hash function is applied to the resources, creating a unique Content Identifier (CID) that can be used to retrieve and share files. In the literature, it is possible to find some related works that involve the use of Smart Contracts and DFS to share data between actors [24], [25].

III. TruDaMul Framework

In this work, we are interested in describing a framework that enables any Client (C) that finds itself in an offline condition (e.g. it is in a no broadband connection area), to send a message to any Server (S) that is online. This framework is supported by a Data Mule (M), which takes care of retrieving (offline) the payload of C and bringing it to a Proxy (P), which in turn forwards (online) the message to the Server (S). P, then, sends back a possible response through another (or, possibly, the same) Mule. An overview of this framework is shown in Figure 1. A tunneling protocol is put in place where C is offline; C interacts only with the Data Mules, while S may not even be aware of the protocol. All the "dirty" work is performed by M and P that, in turn, communicate and organize themselves through a set of distributed systems and technologies (i.e. the green slice):

- **Smart Contract enabled DLT** - To allow the execution of payments and information verification in a distributed way, thanks to the use of Smart Contracts; in our description we refer to the Ethereum blockchain [16];
- **Announcement service** - To make new announcements between online nodes regarding operations to perform, e.g. a publish/subscribe room such as the one in IPFS [23]; the service can be divided also on the basis of specific geographical zones;
- **DFS** - To store data using immutable identifiers and to enable asynchronous communication between Mules and Proxies; we used IPFS;
- **Decentralized authorization service** - To enable access to encrypted data through a network of nodes that only operate following Smart Contract dictated policies; we refer to the implementation shown in [26].

TABLE I
Data models used in the protocol.

Name	Description
key-pair	Each actor maintains a pair of public and private asymmetric keys $(Pub_{Actor}, Priv_{Actor})$. For simplicity[1], we make use of a unique key-pair for the encryption and signature operations, as well as for generating the actor's DLT address [16].
X and Y	Two sets of keys that C previously generated. To refer to the keys within these sets we associate an identifier, e.g. id_x identifies $x \in X$.
$m_{C\to S}$	The message that C wants to send to S. This message consists of a plaintext that has been encrypted using S's public key, i.e., $m_{C\to S} = Enc(plaintext)_{Pub_S}$.
c_{geo}	The concatenation of the timestamp and the geolocation of C, encrypted using a symmetric key $y \in Y$, i.e. $c_{geo} = Enc(timestamp\|\|geodata)_y$.
id_{geo}	The id of C's geographical zone.
p_C	The payload object that is created by C by encrypting the concatenation of c_{geo}, id_y, id_{geo} and $m_{C\to S}$ with a symmetric key $x \in X$, i.e. $p_C = Enc(c_{geo}\|\|id_y\|\|id_{geo}\|\|m_{C\to S})_x$.
$m_{S\to C}$	It is the message that S replies to C (or that proves that P has contacted S). This message consists of a plaintext that has been encrypted using C's public key, i.e., $m_{S\to C} = Enc(plaintext)_{Pub_C}$.
p_S	The payload object containing $m_{S\to C}$, encrypted by P using C's public key, i.e. $p_S = Enc(m_{S\to C})_{Pub_C}$.
tender	A model used by C and P to generate an object for announcing a new tender and for operating with the Smart Contracts, containing a subset of these elements: • EID - an exchange alphanumeric identifier that acts also as a nonce; • URI_p - an immutable URI, e.g. an hash pointer, that identifies a payload; • *offer* - a numerical value representing C's offer to P; • id_x - the id of a symmetric key with which $m_{C\to S}$ has been encrypted; • c_{geo} - the geodata for C, encrypted using $y \in Y$; • id_y- the id of $y \in Y$; • id_{geo} - the id of C's geographical zone.
balance	A model containing a set of information used by C for updating the balance in a state channel. It contains: • *addr* - the address of the other party the state channel has been opened with; • the on-chain state channel identifier; • the updated balance.

The framework components described in the previous list enable the execution of a protocol that is at the core of *TruDaMul*. The protocol allows C and S to communicate and can be divided in two directions, almost mirrored in their behavior: (i) the sending of a message from C to S, and (ii) the answer replied by S to C. In here, C and S can be distinguished because the former is always offline during the main phases of the protocol (apart of the setup phase) while the latter is assumed to be always online.

Moreover, actors in the framework make use of a set of data models to create objects that they exchange between them during the protocol execution. We introduce such data models in Table I, with the aim to make the protocol clearer.

[1]Protocols such as the Dual-Key Stealth Address Protocol [27] can be implemented for higher levels of privacy. However, their use is not described in this work because they are not functional requirements and would render the framework description more complex.

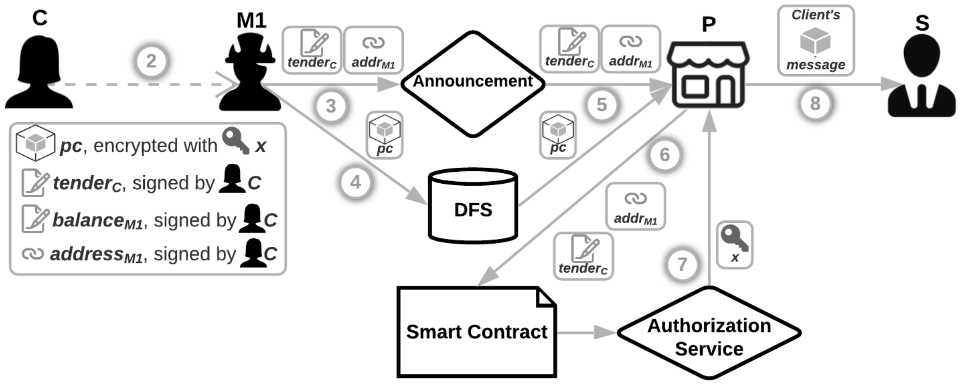

Fig. 2. Graphical representation of the Client-Server Forward Direction phase of the TruDaMul protocol.

A. Smart Contracts and setup phase

During the execution of the protocol, in particular when Mules and Proxy are online, a set of Smart Contracts is used. In the following, we provide a general specification for these:

- **ERC20 Token** - In the protocol description, we make use of a unique token (i.e. an ERC20 Token) for allowing any payment exchange, i.e. on-chain or off-chain. This token is used by all the actors involved and we assume that the ERC20 Smart Contract regulating the transfers has been previously deployed to the DLT. Hence in the setup phase, C is required to get hold of a certain amount of tokens, which will be used in state channels and other Smart Contracts in order to pay other actors.
- **State Channel** - A Smart Contract has been deployed beforehand in order to manage all the state channels, as described in Section II. Using part of the token amount held, C opens a set of state channels with each one (or part) of the Mules that operate in C geographical zone.
- **TruDaMul** - C also deposits an amount of tokens in the *TruDaMul* Contract, that executes the majority of the protocol tasks and thus requires some tokens to pay Mules and Proxies directly. This contract is deployed for each C at the setup phase and is owned by this one.

It is worth noticing that, during the setup phase only, C is required to be online. This initialization is required once and it can be executed in many different ways (e.g. off-site or by means of a trusted device). For space reasons, we will not describe this procedure in detail and it is left as future work.

Finally, C stores two sets of keys, X and Y, in the decentralized authorization service. In particular each key, $x \in X$ and $y \in Y$, is treated as a secret and shared among the nodes that compose the service using the Secret Sharing method [26], [28]. In the following subsections, we describe the protocol, aided by Figures 2 and 3.

B. Client-Server Forward Direction

C is willing to send a message $m_{C \to S}$ to S and waits for a Mule. Meanwhile, C broadcasts, through the short range communication medium, a request for taking charge of the payload p_C containing $m_{C \to S}$ and the geodata. The forward direction of the protocol goes through the following steps:

1) When a Mule, $M1$, passes nearby C, it receives a request containing the payload dimension (in byte) and the tokens offered for the job. An automated negotiation thread [29] may happen here, for the price (in tokens) C is willing to pay to $M1$ for the job to be done.
2) Then C transmits to $M1$ the following objects:
   - the payload p_C, encrypted with $x \in X$.
   - a $tender_C$ object, signed by C, including:
     - the exchange id EID;
     - URI_{p_C} (e.g. generated as an IPFS CID)
     - the tokens *offer* for a Proxy;
     - id_x;
   - a $balance_{M1}$ object, signed by C, that updates the balance between C and $M1$ in their state channel.
   - $address_{M1}$ signed by C.
3) Once $M1$ becomes online, it can directly announce the tender using the $tender_C$ object, in order to reach an audience of different Proxies. This process happens in a dedicated announcement service.
4) Before (or while) announcing the tender, $M1$ also uploads the payload p_C to the DFS. The id to get p_C from the DFS shall derive from the URI_{p_C} indicated in the $tender_C$ (e.g. the IPFS CID, Section II).
5) Any Proxy can check $tender_C$, as well as download p_C and check its integrity.
6) A Proxy P, that decides to take charge of $tender_C$, simply invokes a method in the *TruDaMul* Smart Contract owned by C. The *submitTender* method:
   - requires as parameters:
     - the $tender_C$ object;
     - the signature of $tender_C$;
     - $address_{M1}$;
     - and the signature of $address_{M1}$.
   - automatically checks the validity of the signatures and locks the amount of tokens indicated by C in $tender_C$ in favor of P.
   - automatically and immutably binds EID to $M1$ (this

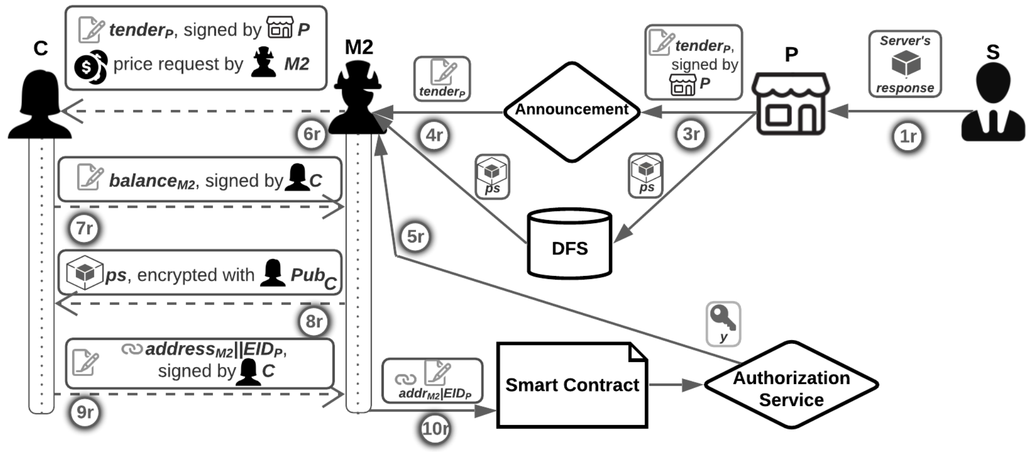

Fig. 3. Graphical representation of the Server-Client Response Direction phase of the TruDaMul protocol.

operation allows $M1$ to close the state channel with C in the future, using the current $balance_{M1}$ object).
- binds P's address with id_x.

7) The last method's task makes P eligible to get access to the key identified by id_x. Thus, P sends a signed request to the decentralized authorization service for accessing the key x. Each authorization node autonomously checks the Smart Contract to verify that P is eligible for accessing the secret x, and then releases a share of x to this actor. Then P aggregates the shares to obtain x using the Secret Sharing technique [26].
8) P can finally decrypt the payload p_C (previously obtained from the DFS) through the key x and send the message $m_{C\rightarrow S}$ to S.

C. Server-Client Response Direction

Up to this point, the payment for P is still locked. If no response is needed to return to C, P shall send a proof of the interaction with S. However, in any case protocol continues through the following steps:

1r) P creates the new payload p_S containing S's response message (or a proof), i.e. $m_{S\rightarrow C}$, encrypted using C's public key.
2r) P creates:
- a $tender_P$ object including:
  - a new exchange id EID_P;
  - URI_{p_S};
  - c_{geo}, extracted from p_C;
  - id_y, extracted from p_C;
  - id_{geo}, extracted from p_C.

3r) Then $tender_P$ is published in the announcement service and p_S is uploaded to the DFS. This announcement also requires the information about the location of C, extracted from p_C, in order to allow a possible candidate Mule to know where to deliver p_S. Announcement services can be organized, thus, on the basis of the possible ids_{geo}, in such a way that Mules get only messages for the zones in which they operate. We discuss the related privacy issues in Section IV.
4r) a Mule, $M2$, that wants to take charge of $tender_P$, downloads the payload p_S from the DFS.
5r) $M2$ then sends a signed request to the decentralized authorization service for accessing the key y identified by id_y. Each authorization node autonomously checks the state channels opened by C in the $StateChannel$ Smart Contract in order to verify that one has been opened with $M2$. If so, each node releases a share of y to $M2$. This allows $M2$ to decrypt c_{geo} and to know where to find C.
6r) Once $M2$ reaches the vicinity of C, the former transmits to the latter $tender_P$ together with a price request (in tokens) for transmitting p_S.
7r) Once C checks the validity of $tender_P$, it may start an automated negotiation thread with $M2$ for reaching an agreement on the price's request. C, then, sends to $M2$ a $balance_{M2}$ object for updating the balance in the state channel between C and $M2$ with the agreed sum.
8r) $M2$ transmits p_S.
9r) C decrypts p_S and checks the hash of p_S with the URI_{p_S} found in $tender_P$. If valid, C replies to $M2$ with a signed concatenation of $address_{M2}$ and EID_P. If p_S's hash corresponds to the information contained in URI_{p_S}, but the data seems to have been corrupted by P, then C can reply to $M2$ omitting EID_P in the concatenation.
10r) Since the presence of C signature is required for closing the channel using the $balance_{M2}$ object (and thus getting paid), then $M2$, once online, invokes the method *submitPayment* of the *TruDaMul* contract. This one requires $address_{M2}$ signed by C to unlock the payment. If EID_P is also found, it unlocks also P's amount.

IV. Security and Privacy Considerations

In this section, we discuss the most relevant issues of the framework in terms of security and privacy concerns.

Misbehavior 1 $M1$ *takes charge of C's payload p_C AND does not announce p_C.*

Discussion: This behaviour is discouraged by the protocol. In particular, by the fact that if $M1$ does not announce p_C, it cannot close the state channel with the updated balance. It means that $M1$ does not get paid, because the balance cannot be updated. Indeed, only the $tender_C$ data (the included EID, in particular) and a valid signature submission to the *TruDaMule* Smart Contract (through the *submitTender* method) enable $M1$ to unlock the *closeChannel* method with the latest $balance_{M1}$ object. Otherwise, $M1$ can only close the channel using a previous valid balance object.

Misbehavior 2 *$M1$ takes charge of C's payload p_C AND announces p_C AND (invokes the submitTender method OR does not store p_C in the DFS).*

Discussion: Invoking *submitTender* would make $M1$ the Proxy entitled to contact the Server S, which is considered a legal behavior. A malicious $M1$, however, might never contact S, while benefiting from the payment received with $balance_{M1}$, that is now valid since *submitTender* has been invoked. Due to the fact that this contract method makes the tender information public in the blockchain, a possible solution is to set up a $gracePeriod$ after which any new Proxy can invoke *submitTender* again and substitute the previous Proxy. A malicious $M1$ might continue to invoke the method several times or might not store p_C in the DFS. This misbehavior would result in a Denial of Service (DoS) attack, that can be solved by having C monitoring the Mules and keeping a "blocklist" for Mules which a response has never come back. In most scenarios, blocklisting $M1$'s blockchain address would be sufficient, since payments are bounded by an already opened state channel. Indeed, a dedicated protocol can be employed when opening new state channels, e.g. a reputation mechanism for Mules, but it is not the scope of this paper.

Misbehavior 3 *P invokes the submitTender method AND (does not contact S OR produces a corrupted response).*

Discussion: P is incentivized to correctly execute the protocol, since it will only get paid once the signed EID_P reaches the *TruDaMule* Smart Contract. This can happen only if C receives a response and it has not been corrupted. In all the other cases (e.g. even in a DoS attack by P by constantly invoking *submitTender*), C can use a "welcome" list of trusted Proxies, but this time implemented directly on the *TruDaMul* contract in order to allow only these to invoke *submitTender* (the same protocol can be used with $m_{C \rightarrow S}$ being a DLT transaction containing the welcome list and S is a DLT node). Moreover, if S produces no response, P can send to C a proof of the tentative. Finally, a malicious C might not sign EID_P for a valid response. In this case P can blocklist C.

Misbehavior 4 *$M2$ does not bring p_S to C.*

Discussion: $M2$ is incentivized to execute the protocol, since it will only get paid once the signed $address_{M2}$ reaches the *TruDaMule* Smart Contract. $M2$ can perform a DoS attack when it is the only Mule available for C. Indeed, P announces the $tender_P$ for all the Mules in the geographical zone, and other Mules can reach C before or after $M2$. A malicious C might not sign $address_{M2}$ for a valid response. In this case $M2$ can blocklist C.

Misbehavior 5 *$M2$ does not invoke submitPayment.*

Discussion: The data needed from P to get paid, i.e. EID_P, might not reach the *TruDaMule* contract due to a malicious $M2$ that does not interact with the Smart Contract after the communication with C. However, $M2$ is incentivized to avoid this misbehavior, because EID_P is concatenated with $address_{M2}$ in the signature released by C, and the latter is needed from $M2$ to unlock its payment.

Privacy Concern *C's personal data and location privacy*

Discussion: The public disclosure of C's geodata is a possible conflict point with personal data protection regulations, since the location of an individual (C) is considered a personal information [30]. The actual fine grained location information is never shown publicly nor stored on-chain. We follow the approach to reference personal data, i.e. p_C, and their content on-chain, i.e. through URI_{p_C}, and to store them off-chain in a DFS. A data protection compliant solution, indeed, is to couple this approach together with the use of Key Reuse Encryption and Single-Use Salt minimizing the risks of de-anonymization [31], [32]. P requires in input some coarse grained information about the location of C, i.e. in order to allow candidate Mule $M2$ to know where to deliver the payload p_S. P only can have this information since it gets *id*$_{geo}$ by decrypting p_C with x. On the other hand, the candidate Mule $M2$ knows the fine grained position by decrypting c_{geo} with y. However, this information is needed to actually reach C. The location information disclosed may be more coarse or fine-grained, however, in all cases C must be clearly informed and must consent to this use of personal information [30], [32].

V. Experimental Evaluation

We conducted a set of experiments to evaluate the framework performances in terms of latency and Smart Contracts operations cost. Due to the complexity of the protocol and the very different technologies / interactions among involved entities, it was not convenient to implement a single, unified testbed environment for evaluating the whole framework. This also because in different steps of the protocol, there are different metrics and aspects that need to be considered. For this reason, we separate the experimental study in different parts and analyze them in isolation.

With reference to the TruDaMul protocol (i.e. Figures 2 and 3), in Figure 4 we indicate the overall latencies related to each interaction among actors and/or systems. This figure is useful to understand our evaluation setup: (i) we analyze the Client-Mule offline interaction (dashed arrows between C and M) in subsection V-A; (ii) how C's data gets online through the Mule (arrows with the "ROAD" variable) is discussed in subsection V-D; (iii) the interaction with Smart Contracts (arrow with the "DLT" variable) is discussed in subsection V-B; (iv) the interaction with the other decentralized systems is discussed in Section V-C.