InDaMul: Incentivized Data Mules for Opportunistic Networking Through Smart Contracts and Decentralized Systems

MIRKO ZICHICHI, Ontology Engineering Group, Universidad Politécnica de Madrid
LUCA SERENA, Department of Computer Science and Engineering, University of Bologna
STEFANO FERRETTI, Department of Pure and Applied Sciences, University of Urbino "Carlo Bo"
GABRIELE D'ANGELO, Department of Computer Science and Engineering, University of Bologna

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. However, some dislocated areas are underprivileged in providing such services due to the lack, inefficiency, or excessive cost of Internet access. Opportunistic networking techniques might aid in surmounting these problems. In this article, we propose a framework that relies on an untrusted Data Mule to carry data from an offline source to an online destination. In particular, we present a framework that enables the communication between different actors and a reward mechanism using Distributed Ledger Technologies, Smart Contracts, and Decentralized File Storage. 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 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.

CCS Concepts: • **Networks** → *Mobile networks*; *Network simulations*; • **Computer systems organization** → **Distributed architectures**; • **Security and privacy** → **Domain-specific security and privacy architectures;**

Additional Key Words and Phrases: Distributed ledger technology, mobile networking, smart contracts, state channels

ACM Reference format:
Mirko Zichichi, Luca Serena, Stefano Ferretti, and Gabriele D'Angelo. 2023. InDaMul: Incentivized Data Mules for Opportunistic Networking Through Smart Contracts and Decentralized Systems. *Distrib. Ledger Technol.* 2, 2, Article 14 (June 2023), 28 pages.
https://doi.org/10.1145/3587696

14

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 was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. This research was also funded in part by the University of Urbino through the "Bit4Food" research project.
Authors' addresses: M. Zichichi, Ontology Engineering Group, Universidad Politécnica de Madrid, ETS de Ingenieros Informaticos (ETSIINF) - Campus de Montegancedo s/n, Boadilla del Monte 28660, Comunidad de Madrid, Spain; email: mirko.zichichi@upm.es; L. Serena and G. D'Angelo, Department of Computer Science and Engineering, University of Bologna, Mura Anteo Zamboni, 7, Bologna 40126, Italy; emails: {luca.serena2, g.dangelo}@unibo.it; S. Ferretti, Department of Pure and Applied Sciences, University of Urbino "Carlo Bo", Piazza della Repubblica, 13, Urbino 61029, Italy; email: stefano.ferretti@uniurb.it.

2769-6472/2023/06-ART14 $15.00
https://doi.org/10.1145/3587696

1 INTRODUCTION

Smart cities are an ongoing breakthrough, 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 primary goal the generation of more sustainable economic development and a better quality of life [38, 46]. However, not all territories are equal, and for some underprivileged ones, it is unfeasible to implement (costly) smart city services. This is due to very different economic circumstances or unavailable, unreliable, or too expensive network infrastructures [22]. It has been recognized that what is outside smart cities is often “left behind” [27, 38]. Thus, intending to mitigate this novel form of digital divide somehow, some research efforts now focus on smart territories that include rural and dislocated locations and that distinctly emerge as opposition to fully serviced smart cities [17, 23]. What is needed is a set of novel opportunistic solutions able to dynamically exploit all the resources that the territory community can share. 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 comprehensive area network connectivity for granted, and specific networking solutions (e.g., satellite connections) might be unfit or too costly.

Clearly enough, to build such a kind of service ecosystem, adequate incentive strategies are needed, i.e., rewarding those users that offer their service to others (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** (**DFSs**), and cryptocurrencies (or tokens [20]) represents a novel and possibly beneficial solution that may foster the provision of smart services in dislocated communities [9]. Anytime a user acts as a middleman and shares/provides a resource, they earn some tokens; every time they access a resource from another peer, they consume some tokens. At the same time, some mechanisms are needed to help communities deal with possibly untrusted service providers, assuming that, in some cases, trust is brought by (entirely or partially) tracing and validating interaction processes through DLTs.

In this article, we focus on data transmission issues in territories where the broadband Internet connection is not taken for 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 with wireless communication capability and storage for data collection [6, 32, 36].

This work aims to propose and validate the framework called *InDaMul*, with the main goal of adding trust to the opportunistic data management activities provided by the Data Mule. In particular, in this article, we provide the following main contributions:

- A detailed description of the decentralized system and distributed technologies involved in the framework, i.e., DLTs, DFS, and the authorization systems;
- A detailed description of the protocol to allow 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 to reproduce actors' behavior and then evaluate the communication delay.

The remainder of this article is organized as follows. Section 2 provides the background and related works. In Section 3, the framework, together with the main protocol, is presented, while in Section 5, we discuss some

Table 1. Summary of the Features and Comparison of Related Works with Our Work

Work	Uses Data Mule	DLT-based incentives	Experimental Evaluation	Results
[14]	Yes	No incentives	None	A new Data Mule model.
[32]	Yes	No incentives	Random walks used to analyze the predicted performance of the model	The average latency of a message is inversely proportional to the number of mules and access points in the system.
[16]	Yes, vehicles in an archipelago	No incentives	Simulation of mobility model	In a network with known routes, e.g., checkpoint, spreading messages is more efficient.
[25]	Yes, vehicles in rural area scenario	No incentives	Simulation of mobility model	Guarantee a high rate of successful message delivery under different conditions.
[43]	Yes, vehicle-to-vehicle communication	No incentives	Simulation of mobility model	The model performs well for networks with frequent partitioning and rapid topology changes.
[10]	Yes, disaster management	Yes, Smart Contracts	None	A new Data Mule model based on DLTs.
Ours	Yes	Yes, Smart Contracts + State Channel Network	Simulation + Smart Contract Gas	Transmission from/to a moving mules is incentivized and can be considered viable and reliable.

security considerations. In Section 6, we present the experimental evaluation, and finally, Section 7 provides the conclusions.

2 BACKGROUND AND RELATED WORK

This section introduces the needed background and describes some related works. Our approach combines different technologies, i.e., a delay tolerant scheme typically referred to as a data mule, DLTs, and DFSs. Table 1 shows a comparison of related approaches, with a summary of which technologies are used, if a related experimental evaluation is discussed to support the proposed approach, and a brief description of the overall service provided. As shown in the table, to the best of our knowledge, this is the first proposed system that employs the benefits of different decentralized solutions to support data delivery in opportunistic networks.

2.1 Data Mules

Data Mules (an acronym for Mobile Ubiquitous LAN Extensions [32]) is a technology aimed to provide digital communication in places without direct connectivity to the Internet. It is thus a specific type of solution for offering opportunistic networking [21]. They are mobile devices that consist of a storage device and a short-range wireless communication medium (e.g., Wi-Fi or Bluetooth). They can exchange data to/from a nearby static sensor or access point that they encounter [6]. As a result of their movement between remote areas, they effectively create a data communication link [16]. Since device movement is of primary importance for message delivery, this solution is suitable for delay-tolerant services.

Data Mules allow for communication and data transfer even in the absence of the Internet, and they can be essential tools for the functioning of applications concerning the **Internet-of-Things** (**IoT**). Data Mules are generally employed for services based in smart cities or villages, with a significant dataflow coming from remote areas. Depending on the context, Mules can be either transportation vehicles like buses or cars [14] or walking persons.

In the last few years, many works have been presented on Data Mules. For instance, in [32], a study was made with several Data Mules performing independent random walks that collect data from static sensors and deliver them to base stations without a forward to other Data Mules. Other works refer to real vehicular network use cases, focusing on routing algorithms for the exchange of messages between Mules and other nodes [16, 25, 43].

We are aware of just one work on integrating Data Mules and DLTs. It is a system for exchanging the Ethereum [13] blockchain blocks in a delay tolerant network [10]. 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.

2.2 Distributed Ledger Technologies

DLTs consist of a network of nodes that maintain a distributed ledger following the same protocol. 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.

2.2.1 Smart Contracts. Smart Contracts are instructions stored in the blockchain and automatically triggered once the default condition is met. We will refer to Ethereum [13] due to its widespread public open-source blockchain use and its provision of robust Smart Contract development tools. Smart Contracts allow anyone to employ DLTs to operate well beyond just currency transactions [44]. For instance, the creation of smart services based on Smart Contracts may enable users to interact with devices/vehicles in smart transportation systems or favor the interoperability among the devices and resources of smart cities [20, 48].

In DLTs such as Ethereum [13], it is possible to build structures through Smart Contracts that act as second-layer cryptocurrencies, i.e., tokens [39]. 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 [9].

2.2.2 State Channels for Services Payments. Since transactions in Smart Contracts and DLTs can be expensive in terms of fees and latencies, 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 [18]. State channels are regulated through Smart Contracts that manage the validation of the payments in the channel. A prominent implementation in the Ethereum blockchain is μRaiden [18], an open-source framework used to implement token-based free pay-per-use payment channels. 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 (i.e., 1st transaction) by depositing an amount of the ERC-20 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. Both parties authenticate themselves using the public–private key-pair to derive their addresses on the Smart Contract. The exchanged messages are used to update a balance value between U and V, e.g., if U has to pay V, then the balance increases; otherwise, the balance decreases.
- **Closing Channel** - Both U and V can close the state channel at any time by invoking the corresponding method in the Smart Contract (i.e., 2nd transaction). To be executed, the corresponding method needs the copy of the last *balance* message exchanged and the signature of both parties. Finally, the balance value is deducted 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, U can pay W through V. These consist of establishing state channel networks, where the participants pay by using other participants as relays among many state channels, forming a connected network. It is specifically a Layer-2 network application running on

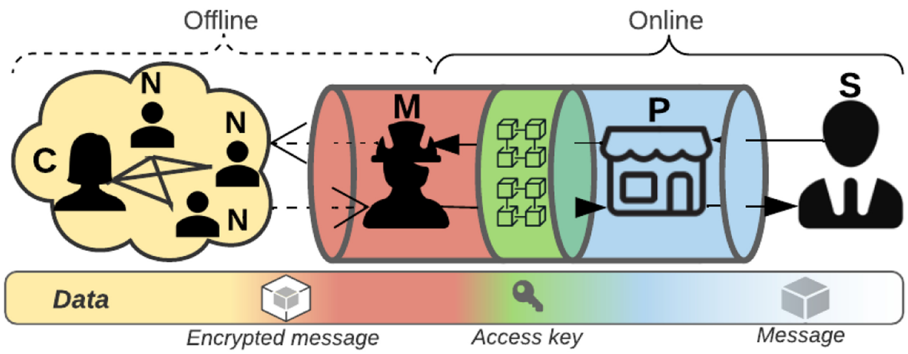


Fig. 1. Overview of the InDaMul framework. It can be compared with a tunneling protocol where the Client (C) is offline (and possibly with some Neighbors N) and manages to send data to the Server (S) via Mules (M) and Proxies (P).

top of the Layer-1 services of a cryptocurrency [18]. It is the main idea behind Bitcoin's Lightning Network [28] and Ethereum's Raiden Network [18].

2.3 Decentralized File Storages and Content Addressing

DFSs 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**) [11] is one of the most used DFS protocols. A cryptographic hash function is applied to the resources to identify the items through the **Peer-to-Peer** (**P2P**) network that runs the IPFS protocol. It means 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 [35, 40].

3 INDAMUL 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, hence having short-range wireless communication as the sole option to transmit data), 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, via short-range wireless communication technology, 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 [13];
- **State Channels** - To enable offline payments between Clients and Mules;
- **Announcement service** - To make new announcements between online nodes regarding operations to perform, e.g., a publish/subscribe system in which nodes publish requests, and subscriber nodes can decide to take charge of such requests; the service can also be divided based on 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 [47].

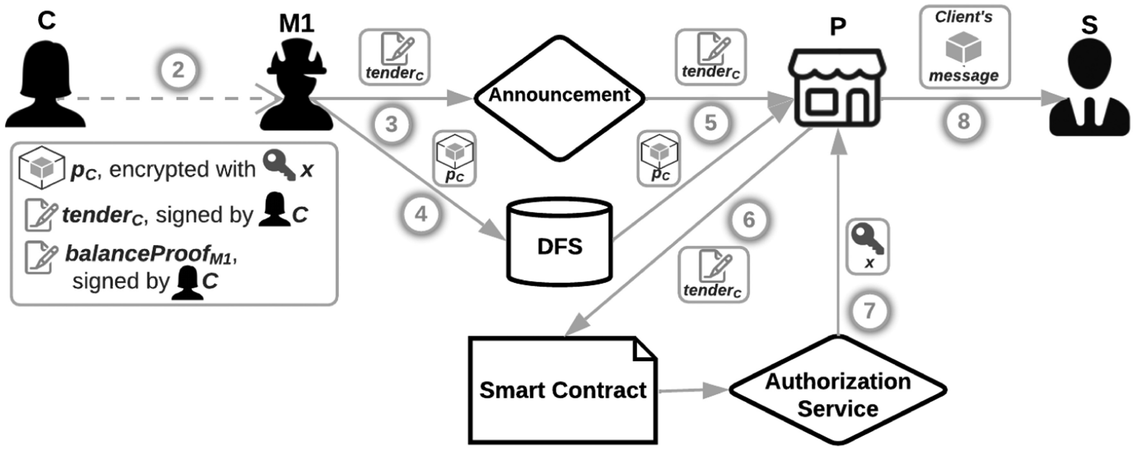


Fig. 2. Graphical representation of the Client-Server Forward Direction phase of the InDaMul protocol.

The framework components described in the previous list enable the execution of a protocol at the core of *InDaMul*. 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. Here, C and S can be distinguished because the former is always offline during the main phases of the protocol (apart from the setup phase) while the latter is assumed to be always online.

In the following subsections, we describe the protocol, aided by Figures 2 and 3. Algorithms 1–3 show the related pseudo-code for the Client-Server forward direction. For the sake of conciseness, we show only algorithms for this part. The response direction has, in essence, a mirrored behavior. We thus omit its related algorithms while describing it in detail in the text.

3.1 Data Structures

Actors in the framework use a set of data structures to create objects they exchange between them during the protocol execution. We introduce such data structures intending to make the protocol clearer.

3.1.1 Keys.

- $(Pub_{Key}, Priv_{Key})$ - Each actor maintains a pair of public and private asymmetric keys. 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 [13].
- X and Y - Two sets of symmetric keys that C generated in the setup phase. To refer to the symmetric keys within these sets, we associate an identifier, e.g., id_x identifies $x \in X$. These symmetric keys are generated randomly using C's local device as entropy source.

3.1.2 Payloads.

- $m_{C \to S}$ - The message that C wants to send to S. This message consists of a message plaintext and a nonce that have been encrypted using S's public key, i.e., $m_{C \to S} = Enc_{Asymm}(message\_plaintext||nonce, Pub_S)$, where Enc_{Asymm} is a public key encryption operation. This definition of a message (and later of a response) has been left general, but could be also referred to a message in the HTTPS protocol.

[1]Protocols such as the Dual-Key Stealth Address Protocol [15] can be implemented for higher levels of privacy. However, this work does not describe their use because they are not functional requirements and would render the framework description more complex.

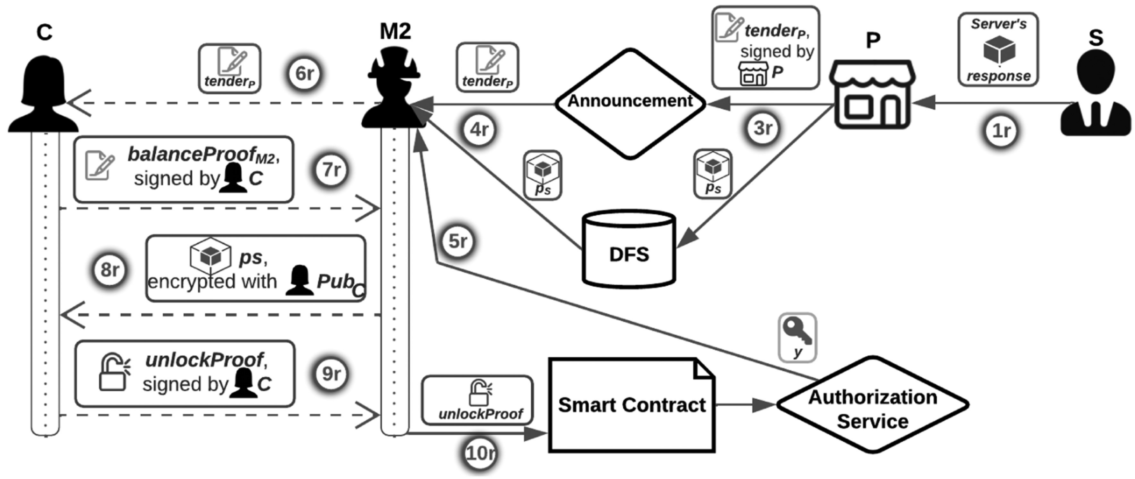


Fig. 3. Graphical representation of the Server-Client Response Direction phase of the InDaMul protocol.

- id_{geo} - The unique identifier of the geographical zone within the system.
- c_{geo} - The concatenation of a timestamp and a geodata, encrypted using a symmetric key $y \in Y$ and a symmetric encryption operation, i.e., $c_{geo} = Enc_{Symm}(timestamp||geodata, y)$.
- p_C - The payload object that is created by the symmetric encryption of the concatenation of c_{geo}, id_y, id_{geo} and $m_{C\to S}$ with a symmetric key $x \in X$, i.e., $p_C = Enc_{Symm}(c_{geo}||id_y||id_{geo}||m_{C\to S||Pub_S}, x)$.
- $m_{S\to C}$ - The response for C. It consists of a response plaintext and a nonce that have been encrypted using C's public key, i.e., $m_{S\to C} = Enc_{Asymm}(response\_plaintext||nonce, Pub_C)$.
- p_S - The payload object containing $m_{S\to C}$, encrypted by P using C's public key, i.e., $p_S = Enc_{Asymm}(m_{S\to C}, Pub_C)$.

3.1.3 Proofs.

- $tender_C$ - A data structure used by C for announcing a new tender and for operating with the Smart Contracts, containing these elements:
  - $addr_{InDaMul}$ - address of C's InDaMul contract;
  - id_{chain} - DLT identifier needed for the parties to agree on the DLT used;
  - $exID$ - an exchange alphanumeric identifier for the forward direction (also acts as a nonce);
  - $pPayID$ - an exchange alphanumeric identifier for the backward direction (also acts as a nonce);
  - URI_{p_C} - an immutable URI, e.g., an hash pointer, that identifies a payload p_C;
  - *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;
  - $addr_M$ - the DLT address of the Mule M;
  - $hash_{balanceProof}$ - the hash digest of the $balanceProof$ object exchanged between C and M;
  - sig_{tender} - signature on the above data.
- $balanceProof$ - A data structure containing a set of information used by C for updating the balance in a state channel. It contains:
  - $addr$ - address of the StateChannel contract;
  - id_{chain} - DLT identifier;
  - $id_{channel}$ - channel identifier inside the StateChannel contract;

ALGORITHM 1: Client C Sending a Message to the Server S

Global Data:
- $(Pub_C, Priv_C)$: C's key-pair
- $addr_C$: C's address
- X, Y: sets of symmetric keys
- id_{geo}: identifier of the geographical zone
- *geodata*: indicating C's position
- $addr_{InDaMul}$: address of C's InDaMul contract
- id_{chain}: the DLT identifier
- $addr$: address of the StateChannel contract

Input:
- Pub_S: S's public key
- *message_plaintext*: message to send to S
- $x \in X$, id_x: random symmetric encryption key for the payload
- $y \in Y$, id_y: random symmetric encryption key for the geodata
- *offer*: tokens' offer intended for the Proxy
- *muleOffer*: tokens' offer intended for the Data Mule

Result:
sends $balanceProof_{M1}$, $tender_C$ and payload p_C to a Mule $M1$

function:
  $nonce, exID, pPayID \leftarrow$ getRandom(3) // generate 3 random objects
  $m_{C\rightarrow S} \leftarrow Enc_{Asymm}(message\_plaintext || nonce, Pub_S)$
  $c_{geo} \leftarrow Enc_{Symm}(timestamp || geodata, y)$
  $p_C \leftarrow Enc_{Symm}(c_{geo} || id_y || id_{geo} || m_{C\rightarrow S} || Pub_S, x)$ // generate payload p_C
  $URI_{p_C} \leftarrow$ getURIThroughHash(p_C)
  $size_{p_C} \leftarrow$ len(bytes(p_C))
  $partialTender_C \leftarrow addr_{InDaMul} \parallel id_{chain} \parallel exID \parallel pPayID \parallel URI_{p_C} \parallel offer \parallel id_x$ // generate partial tender
  $notSent \leftarrow$ True
  while $notSent$ **do**
    /* broadcast request to passing mules and receive a response */
    $addr_{M1} \leftarrow$ await(broadcastToAMule($addr_C$, $size_{p_C}$, *muleOffer*))
    $id_{channel} \leftarrow$ getChannelFromAddressInLocalStorage($addr_{M1}$, $addr$, id_{chain})
    if $id_{channel}$!= *null* **then**
      $balance \leftarrow$ getBalanceFromChannelInLocalStorage($id_{channel}$, id_{chain})
      $balance \leftarrow balance + muleOffer$
      $nonceM1 \leftarrow$ getRandom()
      $balanceProof_{M1} \leftarrow addr \parallel id_{chain} \parallel id_{channel} \parallel balance \parallel nonceM1 \parallel URI_{p_C}$ // generate balance proof
      $hash_{balanceProof_{M1}} \leftarrow$ getHashDigest($balanceProof_{M1}$)
      $sig_{proof} \leftarrow$ sign($hash_{balanceProof_{M1}}$)
      $tender_C \leftarrow partialTender_C \parallel addr_M \parallel hash_{balanceProof_{M1}}$ // generate tender
      $hash_{tender_C} \leftarrow$ getHashDigest($tender_C$)
      $sig_{tender_C} \leftarrow$ sign($hash_{tender_C}$)
      sendToMule($balanceProof_{M1}$, sig_{proof}, $tender_C$, sig_{tender_C}, p_C)
      $sig2_{proof} \leftarrow$ await(listenToAMule()) // receive and store the signature of $balanceProof_{M1}$ made by $M1$
      $notSent \leftarrow$ False
    end
  end
return

ALGORITHM 2: Mule *M1* Receiving a Message Directed to the Server *S*

Global Data:
- $addr_{M1}$: $M1$'s address
- $addr$: address of the StateChannel contract
- id_{chain}: the DLT identifier

Result:
mule $M1$ publishes $tender_C$ and payload p_C

function:
    /* receive a broadcasted request and send a response */
    $addr_C$, $size_{p_C}$, $muleOffer \leftarrow$ await(listenToAClient())
    sendToClient($addr_{M1}$)
    /* receive balance proof, tender and payload */
    $balanceProof_{M1}$, sig_{proof}, $tender_C$, sig_{tender_C}, $p_C \leftarrow$ await(listenToAClient())
    $id_{channel} \leftarrow$ getChannelFromAddressInLocalStorage($addr_C$, $addr$, id_{chain})
    if $id_{channel}$!= $null$ **then**
        /* validate signature to identify C */
        $addr_1 \leftarrow$ verify($tender_C$, sig_{tender_C})
        $addr_2 \leftarrow$ verify($balanceProof_{M1}$, sig_{proof})
        if $addr_1 == addr_2 == addr_C$ **then**
            /* identity confirmed */
            $balance \leftarrow$ getBalanceFromChannelInLocalStorage($id_{channel}$, id_{chain})
            $tempBalance \leftarrow balance + muleOffer$
            if $tempBalance == balanceProof_{M1}.balance$ **then**
                $hash_{balanceProof_{M1}} \leftarrow$ getHashDigest($balanceProof_{M1}$)
                $sig2_{proof} \leftarrow$ sign($hash_{balanceProof_{M1}}$) // sign balance proof and send it to the client
                sendToClient($sig2_{proof}$)
                /* mule goes online */
                await(isOnline())
                announce($tender_C$, sig_{tender_C})
                uploadToDFS(p_C)
            end
        end
    end
return

   - $balance$ - balance amount;
   - $nonce$ - strictly monotonic value used to order transfers (starts at 1);
   - $hash$ - an additional hash digest of an application specific data payload;
   - sig_{proof} - signature on the above data.
- $tender_P$ - A data structure used by P for announcing a new tender and operating with Smart Contract containing:
   - $addr_{InDaMul}$ - address of C's InDaMul contract;
   - id_{chain} - DLT identifier;
   - $exID$ - an exchange alphanumeric identifier for the forward direction (also acts as a nonce);
   - $pPayID$ - an exchange alphanumeric identifier for the backward direction (also acts as a nonce);
   - URI_{p_S} - an immutable URI, e.g., an hash pointer, that identifies a payload p_S;
   - c_{geo} - the geolocation for C, encrypted using $y \in Y$;

ALGORITHM 3: Proxy P Takes Charge of a Payload and Sends it to the Server S

```
Global Data:
- (Pub_P, Priv_P): C's key-pair
- addr_P: P's address
- t: the threshold for the key fragments in the authorization service
Input:
- Pub_S: S's public key
- tender_C: took in charge
- sig_{tender_C}: tender signature by C
- p_C: payload to send to S
Result:
sends the message m_{C→S} to a Server S
function:
    URI_{p_C} ← getURIThroughHash(p_C)
    /* verify payload and then submit tender to InDaMul smart contract */
    if URI_{p_C} == tender_C.URI_{p_C} then
        smartContractInstance ← getSmartContract(tender_C.addr_{InDaMul}, id_{chain})
        result ← smartContractInstance.submitTender(tender_C, sig_{tender_C}, addr_P)
        if result != error then
            /* request key fragments to the authorization service */
            fragments ← []
            for i ← 0; i < t; i++ do
                /* each of the t nodes in the authorization service checks the InDaMul smart contract acl if
                   id_x is associated to addr_P */
                hash_{request} ← getHashDigest(i || id_x)
                sig_{request} ← sign(hash_{request})
                fragments[i] ← requestFragmentToNodei(i, id_x, sig_{request})
            end
            x ← reconstruct(fragments)
            // reconstruct the key used to encrypt the payload
            c_{geo}, id_y, id_{geo}, m_{C→S}, Pub_S ← Dec_{Symm}(p_C, x) // decrypt the payload and send it to S
            sendToServer(m_{C→S}, Pub_S)
        end
    end
return
```

  - id_y - the id of the key with which c_{geo} has been encrypted;
  - id_{geo} - the id of C's geographical zone;
  - sig_{tender} - signature on the above data.
- $unlockProof$ - A data structure containing a set of information used by a Mule M to unlock the payments in its favor and in favor of P. It contains:
  - $exID$ - the exchange alphanumeric identifier found in $tender_C$ for the forward direction;
  - $addr_M$ - the DLT address of the Mule M;
  - $pPayID$ - the exchange alphanumeric identifier for the backward direction, included in the $unlockProof$ object only if P behaved correctly;
  - $hash_{balanceProof}$ - the hash digest of the $balanceProof$ object exchanged between C and M;
  - $sig_{unlockProof}$ - signature on the above data.

3.2 Setup Phase

Before the protocol execution, a setup phase is needed to configure all the services actors will exploit. During the setup phase only, each actor involved in a task is required to be online, thus also C. This initialization is required once and can be executed in many different ways (e.g., off-site or by means of a trusted device). For instance, a dedicated service can be put in place just for the setup phase, in which a trusted mule is delegated by C to carry online DLT transactions already signed by C. In this article, we will not focus in detail on this part of the protocol.

3.2.1 *Keys Setup.*

- **Key-pairs** - Each actor C, M, P, S will generate its key-pair using the same algorithm and publish its public key certificate to be known by all (or part of all) participants. This certificate can be published directly in the DLT, for instance.
- **Symmetric keys** - The C actor 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 generated randomly using C's local device as an entropy source. Then, it is treated as a secret and shared among the nodes that compose the service using the Secret Sharing method [33, 47]. The symmetric key x is "fragmented" into n fragments. For the data recipient, only $t < n$ fragments are sufficient to reconstruct x and decrypt the cyphertext in question. The secret x can be represented as an element a_0 of a finite field, then $t-1$ elements are chosen randomly from this field, $a_1, \ldots, a_{t_1}$. Using these elements this polynomial curve can be constructed $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t_1} x^{t_1}$. Each authorization service node is given a point found in the curve $(x_i, f(x_i))$, with $1 \leq i \leq n$, i.e., the fragment. Therefore, an untrusted authorization node alone cannot decrypt the cyphertext because it needs other $t-1$ fragments. Indeed, in order to obtain a_0, and thus x, a subset of cardinality t of the n points $(x_i, f(x_i))$ is needed to perform the following interpolation: $a_0 = f(0) = \sum_{j=0}^{t-1} f(x_j) \prod_{m=0,\, m \neq j}^{t-1} \frac{x_m}{x_m - x_j}$.

3.2.2 *Framework Parameters Configuration.*

An initial configuration for the framework is needed and can be made by any participant.

- **Geographical zones** - The framework can be uniquely deployed to serve different smart territories and, thus, different geographical zones. For instance, InDaMul can be set up for a specific state's region and used by several users in different towns. Thus, all geographical zones interested in the framework deployment must be indexed during the setup phase. Such zones are intended to cover a wide area, including many Clients, while their precise geolocation will be later encrypted and exchanged directly with Proxies.
- **Announcement service** - An announcement channel is made up for each geographical zone, and Proxies and Mules register to the channels they are interested in serving.
- **ERC20 Token** - The framework uses a unique token deployed to the DLT during the setup phase (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. Hence in the setup phase, actors C, M, and P are required to get hold of a certain amount of tokens, which will be used in state channels or Smart Contracts to pay other actors.
- **State Channel** - The framework uses a unique Smart Contract deployed during the setup phase to manage all the state channels, as described in Section 2.

3.2.3 *Smart Contracts Configuration.*

- **State Channel** - Using part of the token amount held, the C actor opens a set of state channels with each one (or part) of the Mules that operate in C's geographical zone.
- **InDaMul** - C also deposits a number of tokens in the *InDaMul* Contract, which executes most 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.

3.3 Client-Server Forward Direction

C is willing to send a message $m_{C \to S}$ to S and waits for a Mule. The forward direction of the protocol goes through the following steps:

(0) C broadcasts, through the short-range communication medium, a request for taking charge of the payload p_C. Previously C prepared the following objects:
  - the payload is composed of $m_{C \to S}$, the identifier of C's geographical zone id_{geo}, the encrypted concatenation of the precise C's geolocation and current timestamp, i.e., $c_{geo} = Enc_{Symm}(timestamp||geodata, y)$, and the id of the key used for encrypting c_{geo}, i.e., id_y. This payload is encrypted with $x \in X$, $p_C = Enc_{Symm}(c_{geo}||id_y||id_{geo}||m_{C \to S}, x)$.
  - the partial $tender_C$ object including:
    - $addr_{InDaMul}$ - address of C's InDaMul contract;
    - id_{chain} - the DLT identifier;
    - a new exchange id *exID* for the forward direction and a *pPayID* for the backward direction;
    - the URI_{p_C} obtained using the DFS protocol for unique URIs and generated through the hash digest of the payload;
    - the tokens' *offer* intended to the Proxy that will handle message;
    - the id_x of the key used in the payload encryption;

    C will complete the creation of this object after a specific M has been identified.

(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.[2]

(2) If $M1$ accepts, then C transmits to $M1$ the following objects:
  - a $balanceProof_{M1}$ object that updates the balance between C and $M1$ in their state channel (the fields for this object are shown in Section 3.1.3). It also includes a signature on the data by C. $M1$ will create an exact copy of the object but with its own signature;
  - the complete $tender_C$ object including the data already processed and now also:
    - $addr_{M1}$ - $M1$'s DLT address;
    - the *hash* digest of the $balanceProof_{M1}$ object;
    - C's signature $sign_{tender}$ on the $tender_C$ data;
  - the payload p_C.

(3) Once $M1$ becomes online, it can directly announce the tender using the $tender_C$ object 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 $tender_C$ and be known by all actors (e.g., the IPFS CID, Section 2).

(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 *InDaMul* Smart Contract owned by C. The *submitTender* method:
  - requires as parameters:
    - the $tender_C$ object (without signature);
    - the C's signature of $tender_C$;
  - automatically checks the validity of the signature and locks the number of tokens indicated by C in $tender_C$ in favor of P.

[2]An automated negotiation thread [19] may happen here, for the price (in tokens) C is willing to pay to $M1$ for the job to be done. However, its implementation is out of the scope of this work.

- automatically and immutably binds $exID$ to $M1$ (this operation allows $M1$ to close the state channel with C in the future, using the $balanceProof_{M1}$ 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 fragment of x to this actor. Then P aggregates the fragments to obtain x using the Secret Sharing technique [47].

(8) P can finally decrypt the payload p_C (previously obtained from the DFS) through the key x and send the message $m_{C \to S}$ to S.

3.4 Server-Client Response Direction

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

(1r) P receives S's response message $m_{S \to C}$, encrypted using C's public key and then creates a new payload p_S containing $m_{S \to C}$ (or a proof that S did not reply);

(2r) P creates:
- a $tender_P$ object including:
  - $addr_{InDaMul}$ - address of C's InDaMul contract;
  - id_{chain} - DLT identifier;
  - the exchange id $exID$ and the $pPayID$ stored in the $tender_C$;
  - the URI_{p_S} obtained using the DFS protocol for unique URIs and generated through the hash digest of the payload;
  - c_{geo}, extracted from p_C;
  - id_y, extracted from p_C;
  - id_{geo}, extracted from p_C;
  - P's signature $sign_{tender}$ on the $tender_P$ data.

(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, i.e., the id_{geo} found in the $tender_P$, to allow a possible candidate Mule to know where to deliver p_S. Announcement services can be organized, thus, based on 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 5.

(4r) A Mule, $M2$, that wants to take charge of $tender_P$, downloads the payload p_S from the DFS.

(5r) $M2$ sends a signed request to the decentralized authorization service, including the $tender_P$ object, for accessing the key y identified by id_y. Each authorization node has enough information to autonomously check the state channels opened by C in the $StateChannel$ Smart Contract in order to verify that one has been opened with $M2$.[3] If so, each node releases a fragment of y to $M2$. It allows $M2$ to decrypt c_{geo} and to know where exactly 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 $tender_P$ (i.e., address, ids, and signature validation), it may accept the price request.[4] C, then, sends to $M2$ a $balanceProof_{M2}$ object for updating the balance in the state channel between C and $M2$ with the agreed sum.

[3]This means that C and $M2$ already interacted in the past and thus $M2$ can reach C's geolocation.
[4]Also here it can start an automated negotiation thread with $M2$ for reaching an agreement on the price's request.

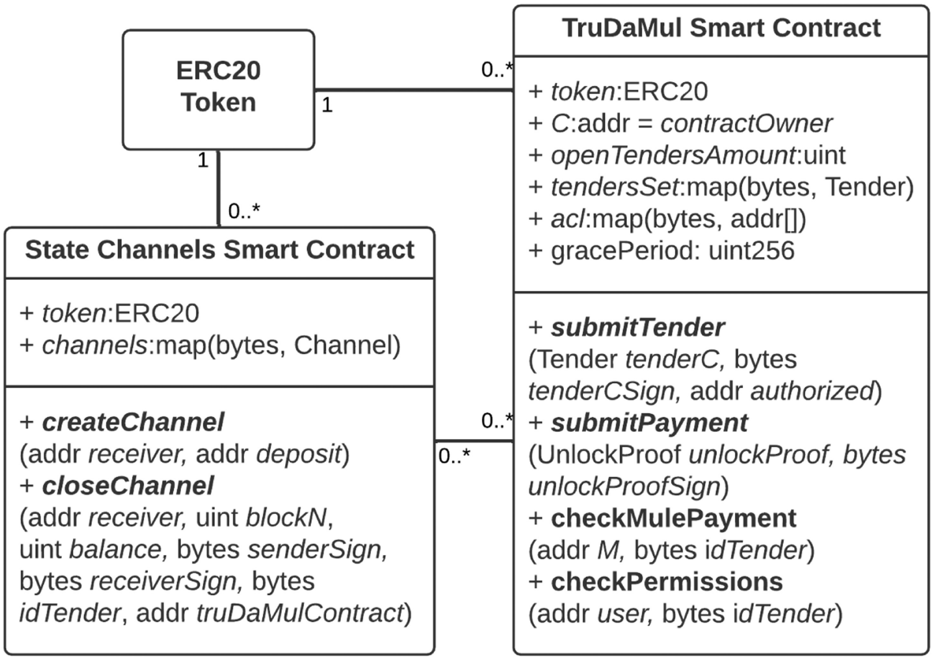


Fig. 4. UML class diagram representing the Smart Contracts used in InDaMul and their relations.

(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 an $unlockProof$ object:
- $exID$ - found in $tender_C$;
- $addr_{M2}$ - $M2$'s DLT address;
- $pPayID$ - (optional) the alphanumeric exchange identifier for the backward direction;
- $hash_{balanceProof}$ - the hash digest of the $balanceProof_{M2}$ object;
- $sig_{unlockProof}$ - signature on the above data.

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 $pPayID$ in the concatenation.

(10r) Since the presence of C signature is required for closing the state channel using the $balanceProof_{M2}$ object (and thus getting paid), then $M2$, once online, invokes the method *submitPayment* of the *InDaMul* contract. This one requires the $unlockProof$ object signed by C to unlock the payment. If $pPayID$ is found, it also unlocks P's amount.

3.5 InDaMul Smart Contract

During the execution of the protocol, mainly when Mules and Proxy are online, a set of Smart Contracts is used, i.e., the ones shown in Figure 4. We find two main methods in the *InDaMul* Smart Contract. The *submitTender* method (see Algorithm 4) automatically checks the validity of the signatures found in the data provided by $M1$ in the announcement and then binds P's address with id_x. It makes P eligible for access to the key identified by id_x. Finally, $M2$ gets paid using a $balance_{M2}$ object in the challenge-response authentication. This object, when uploaded to the *InDaMul* Smart Contract through *submitPayment* (see Algorithm 5), will unlock both $M2$'s and P's payments.

ALGORITHM 4: *submitTender* of *InDaMul* Smart Contract

Global Data:
- $addr_{InDaMul}$ address of the InDaMul Smart Contract
- $addr_C$ address of client C (the contract owner)
- *token* ERC20 Token
- *tendersSet* set of tenders used in the past
- $openTendersAmount$ amount of tokens reserved for proxies in open tenders
- *acl* access control list for X keys

Input:
- $tender_C$ object as defined in Section 3.1.3
- sig_{tender_C} the signature of $tender_C$
- $addr_P$ the address authorized to access id_x

Result:
stores $tender_C$ and P_{addr}, and unlocks payment for $M1$

function:
  `// validate signature to identify C`
  $addr_1 \leftarrow$ verify($tender_C$, sig_{tender_C})
  if $addr_1 == addr_C$ **then**
    `// identity confirmed`
    allowMulePayment($addr_{M1}$, $tender_C.exID$, $tender_C.hash_{balanceProof_{M1}}$)
    if $token.balanceOf(addr_{InDaMul}) - openTendersAmount > tender_C.offer$ **then**
      `// if enough balance then set authorized address`
      tendersSet.add($tender_C$)
      $openTendersAmount = openTendersAmount + tender_C.offer$
      acl.map($tender_C.id_x$, $addr_P$)
    end
  end
return

4 THE ISLAND: A LOCAL STATE CHANNEL NETWORK

In case a Client C does not find itself within the action range of Mules, a network can be set up between C's physical Neighbors, i.e., N. We refer to this network as an "Island", as nodes are in physical proximity, and most of them are isolated (in terms of communication) from the rest of the territory. In order to keep this Island "alive", one or more Target Neighbors, i.e., TN, must be reached by a Mule and then act as relays. Moreover, Clients that cannot interact directly with a TN have to find a path within the Island to reach it and thus have to rely on several forwarding Neighbors.

4.1 Structure

In order to incentivize Neighbors to relay messages, a State Channel Network is used. Each N starts the operations after a setup phase, where it announces through a short communication medium (e.g., Wi-Fi Direct) its presence to the other Neighbors in its vicinity. If N is not isolated, then it would receive in response (from a Neighbor) the Island configuration parameters and the current network topology. Otherwise, it will keep sending announcement messages until another reachable Neighbor (e.g., a moving device) is found. Indeed, the Island network is mainly thought to operate with nodes at fixed locations, but moving nodes might also be supported

It is worth noting that, during the setup phase only, N is required to issue at least one transaction to the DLT in order to open a channel with one of its Neighbors. This initialization is required once and it can be executed in many different ways (e.g., on-demand Data Mule or by means of a trusted device). During the operations, the

ALGORITHM 5: *submitPayment* of *InDaMul* Smart Contract

Global Data:
- $addr_C$ address of client C (the contract owner)
- *token* ERC20 Token
- *tendersSet* set of tenders
- $openTendersAmount$ amount of tokens reserved for proxies in open tenders

Input:
- $unlockProof$ object as defined in Section 3.1.3
- $sig_{unlockProof}$ the signature of $unlockProof$

Result:
unlocks payment for $M2$ and P

function:
    // validate signature to identify C
    $addr_1 \leftarrow$ verify($unlockProof$, $sig_{unlockProof}$)
    if $addr_1 == addr_C$ **then**
        // unlock mule payment
        allowMulePayment($addr_{M2}$, $unlockProof.exID$, $unlockProof.hash_{balanceProof_{M2}}$)
        $tender_C \leftarrow$ *tendersSet*.get($unlockProof.exID$)
        $openTendersAmount = openTendersAmount - tender_C.offer$
        if $unlockProof.pPayID$!= $null$ **then**
            // unlock proxy payment
            token.transfer($tender_C.P_{addr}$, $tender_C.offer$)
        end
    end
return

Neighbors within the Island will share the information regarding their "online" or "offline" status and current opened state channels capacities and fees for their relay. The messages within the Island can then be exchanged using different dissemination strategies (e.g., gossip-based dissemination protocols) [31]. These strategies' security and privacy implications depend on the threat model one can use as a reference. In a threat model where no malicious neighbors within the Island try to infer Clients' activities, the protocol we describe does not need additional mechanisms to preserve confidentiality and part of information privacy. Message content is always encrypted; the only information disclosed to neighbors would be the payment to the Mule providing the service. Nevertheless, also this information could be hidden using a different protocol for the State Channel Network [7]. On the other hand, in a threat model in which malicious neighbors monitor Clients' activities, some mechanisms such as Dandelion++ can be put in place [31]. In any case, a privacy-utility tradeoff makes it impossible to get significant gains in utility by giving up a little privacy or significant gains in privacy by sacrificing a little utility [37].

4.2 Protocol

A Neighbor that decides to send a message to a Server S becomes a Client C and seeks to reach a Data Mule M. The protocol we use in the Island is based on the Raiden protocol [18]. In the following, we model a transfer from an initiator, i.e., C, to a Target Neighbor, i.e., TN, though (zero or) some mediators, i.e., Neighbors N_i. This transfer has the aim to reach M through TN finally.

- C creates a *lockedTransfer* message and propagates it to TN through multiple Neighbors. A *lockedTransfer* message reserves the amount for the pending payment in each channel between C/N_i, N_i/N_j and N_j/TN, depending on the indicated fees.

- Once the *lockedTransfer* reaches TN, then it requests a secret from C by sending a *secretRequest* message.
- When C gets a *secretRequest* message, it checks its validity. Receiving this request means that C can safely assume the *lockedTransfer* message has arrived at TN and that the latter has all the incentives to be honest because it will be paid.
- If all checks out C sends a *revealSecret* message back to TN. The *revealSecret* message contains a secret that allows each N along the path and finally TN to claim the locked amount in the pending transfer.
- A cascade of *revealSecret* messages will begin from TN back to each N_i along the path. This message tells them that the payee (either TN or another N_j) knows the secret and wants to claim the lock off-chain. So then, they may unlock the lock and send an up-to-date balance proof to the payee. This is done by sending the secret message back to the partner who sent the *revealSecret*.
- The transfer is finished when C receives a *revealSecret* message from the first N_1 in the path.

5 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 behavior is discouraged by the protocol. In fact, if $M1$ does not announce p_C, then it cannot redeem the $balanceProof_{M1}$ in the state channel. It means that $M1$ is not paid because the balance cannot be updated. Indeed, only the $tender_C$ data (the included *exID*, in particular) and a valid signature submission to the *InDaMul* Smart Contract (through the *submitTender* method) enable $M1$ to unlock the *closeChannel* method with the latest $balanceProof_{M1}$ object. Otherwise, $M1$ can only close the channel using a previous valid balance object, i.e., a $balanceProof_{M1}$ object obtained in a previous successful interaction with C.

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: By invoking *submitTender*, $M1$ becomes the Proxy entitled to contact the Server S. A malicious $M1$, however, might never contact S while benefiting from the payment received with $balanceProof_{M1}$, which is now valid since *submitTender* has been invoked. Since this contract method makes the tender information public in the DLT, 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 not store p_C in the DFS. This misbehavior would result in a **Denial of Service** (**DoS**) attack, which can be solved by having C monitoring the Mules and keeping a "blocklist" for Mules for which a response has never come back. In most scenarios, blocklisting $M1$'s DLT 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 article.

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 *pPayID* reaches the *InDaMul* Smart Contract. It can happen only if C receives a response and this one is not corrupted. In all the other cases (e.g., even in a DoS attack by P constantly invoking *submitTender*), C can use a "welcome" list of trusted Proxies. This list is implemented directly on the *InDaMul* contract in order to allow only trusted Proxies to invoke *submitTender*.[5] Moreover, if S produces no response, P can send to C a proof of the tentative.

Finally, a malicious C might not sign $pPayID$ 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 $unlockProof$ containing $addr_{M2}$ reaches the *InDaMul* 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 $addr_{M2}$ 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., $pPayID$, might not reach the *InDaMul* 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 $pPayID$ is concatenated with $addr_{M2}$ in the $unlockProof$ object and signed by C. Thus, $M2$ needs to submit the whole $unlockProof$ 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 personal information [26]. 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 is to combine this approach with the use of Key Reuse Encryption and Single-Use Salt, minimizing the risks of de-anonymization [5, 24].

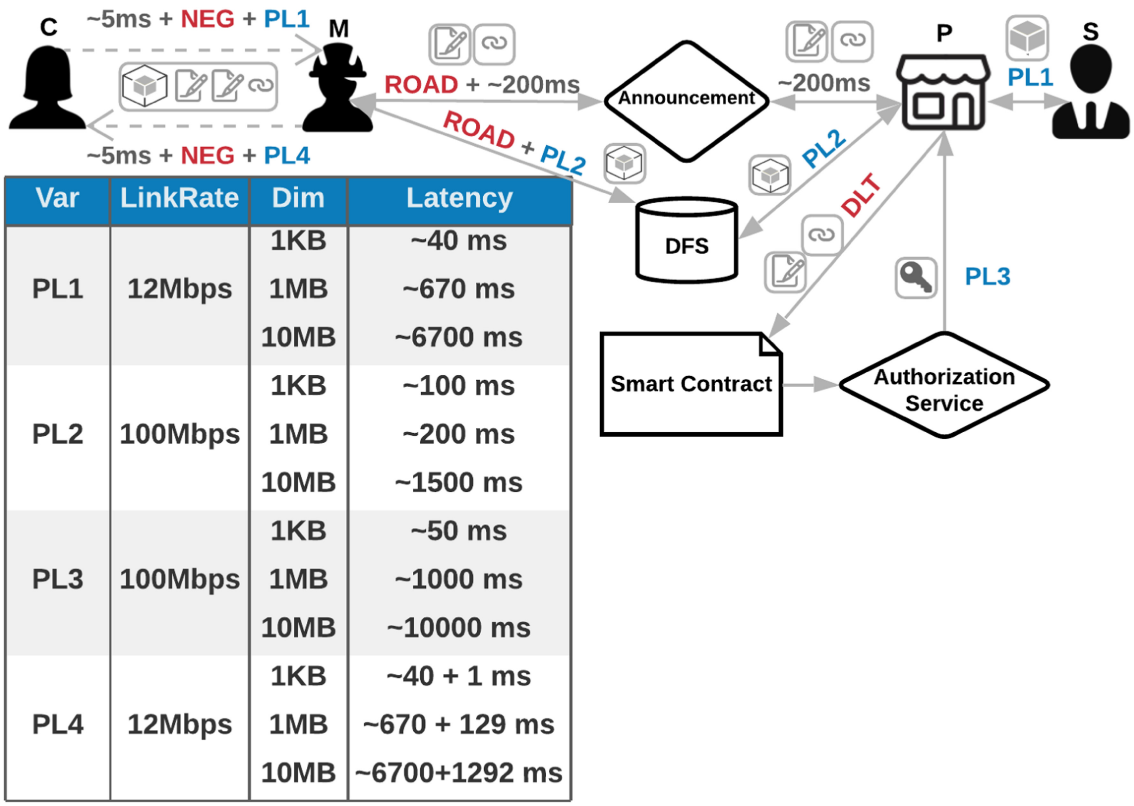
P requires in input some coarse-grained information about the location of C, i.e., in order to allow candidate Mule $M2$ to know where to deliver the payload p_S. P only can have this information since it gets id_{geo} by decrypting p_C with x. On the other hand, the candidate Mule $M2$ knows the fine-grained position by decrypting c_{geo} with y. However, this information is needed to reach C. The location information disclosed may be more coarse or fine-grained. Nonetheless, in all cases, C must be clearly informed and must consent to this use of personal information [24, 26].

6 EXPERIMENTAL EVALUATION

An implementation of the decentralized protocol was built using different technologies. In particular, smart contracts were implemented in Solidity, a popular language initially designed for the Ethereum blockchain, which is now supported by other blockchain platforms, such as Avalanche, Polygon, Wanchain, Hyperledger Besu, ConsenSys Quorum, Binance Smart Chain [45]. The software code is available on GitHub [3]. Through the software testing, it was possible to validate the implementation and assess the viability of the system deployment. We conducted a set of experiments to evaluate the framework performances in terms of latency and Smart Contracts operations cost.

However, 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 is also because, in different steps of the protocol, some different metrics and aspects need to be considered. For this reason, we separate the experimental study into different parts and analyze them in isolation. In previous works, we developed different DLT simulations that could support the evaluation of

[5]This list could be deployed to the Smart Contract considering $m_{C \to S}$ being a DLT transaction containing the welcome list and S is a DLT node, thus using the same InDaMul protocol.



Var	LinkRate	Dim	Latency
PL1	12Mbps	1KB	~40 ms
		1MB	~670 ms
		10MB	~6700 ms
PL2	100Mbps	1KB	~100 ms
		1MB	~200 ms
		10MB	~1500 ms
PL3	100Mbps	1KB	~50 ms
		1MB	~1000 ms
		10MB	~10000 ms
PL4	12Mbps	1KB	~40 + 1 ms
		1MB	~670 + 129 ms
		10MB	~6700+1292 ms

Fig. 5. Latencies (order of magnitude) for each interaction in the protocol. Each edge has a latency obtained by the sum of an average latency for the protocol execution plus some context-dependent variable latencies. Such variable latencies are shown in blue and red. Blue variables, i.e., PL_i, stand for payload latencies and indicate the average latency when varying the payload dimension; their estimations are reported in the table within the figure. Red variables represent latencies that are discussed in detail, respectively, in Section 6: "*NEG*" in 6.1, "*DLT*" in 6.2, and "*ROAD*" in 6.4.

this article [29, 30]. However, in this case, we argue that it is unnecessary to go into that level of detail. We assume that only a synthetic measure of the latency for issuing a transaction in the DLT is needed for our evaluation.

Concerning the InDaMul protocol, in Figure 5, we indicate the overall latencies related to each interaction among actors and/or systems. This figure is useful to understand our evaluation setup:

(1) Section 6.1: We firstly analyze the Client-Mule offline interaction (dashed arrows between C and M);
(2) Section 6.2: Then we discuss the interaction with Smart Contracts (arrow with the "DLT" variable);
(3) Section 6.3: We discuss the interaction with the other decentralized systems;
(4) Section 6.4: Finally, we simulate and analyze how C's data get online through the Mule (arrows with the "ROAD" variable).

6.1 Client-Mules Offline Interactions

A critical part of the protocol is about the C-M interactions. This interaction occurs offline, and its timely success depends on the availability of a communication medium. For this reason, we can only give an order of magnitude to the estimated time for completion of this interaction.

Table 2. Encryption (E) and Decryption (D) avg Latency Measured (ms)

	10KB	50KB	100KB	500KB	1MB	5MB	10MB
E	1	11	22	111	222	1108	2227
D	1	7	13	64	129	646	1292

We simulate a scenario where C is positioned in a fixed location and communicates via Wi-Fi Direct, with M traveling at a constant speed down the road. Inspired by [8, 41], we assume a maximum Wi-Fi link rate equal to 12 Mbps and that M's velocity is 36 km/h. This specific simulator was developed using the Rust language and was run on a dedicated host (i.e., Intel Core i7-6700HQ CPU, 8GB RAM). The source code can be found in [4]. Here below, we report on details about the simulation model.

6.1.1 Client to Mule. A Client C constantly broadcasts messages, including the payload dimension and the tokens offered. The time window in our evaluation starts when $M1$ receives this message (forward direction, steps 1 and 2).

(i) In the case of automated negotiation, if one of the two actors uses a time-dependent tactic [19], we can easily assume that this part of the communication lasts 1 second at most.
(ii) Reached agreement, C sends data to $M1$. The $tender_C$ and $balanceProof_{M1}$ objects and their relative signatures are more or less fixed in bytes dimension (i.e., ~500 B). In our configuration, the time required to sign and encrypt these objects is ~2 ms. Their transmission at 12 Mbps would require roughly 0.3 ms.
(iii) On the other hand, the time required for the payload p_C transmission depends on its dimension and varies in all cases, e.g., $\sim$6.7 seconds for a 10 MB payload and a 12 Mbps link rate. In this protocol step, C has already encrypted the p_C with a key x while waiting for a Mule, performed before our considered time window.
(iv) Finally, the process where $M1$ decrypts the received objects (all but p_C), plus the process of signatures verification, takes an amount of time in the order of 2 ms.

From our analysis, we can deduce that steps 1 and 2 of the protocol (forward direction) mostly depend on the p_C transmission latency. The latency of the whole process we provided as an example, i.e., ~8 sec in total would be feasible in the scenario we considered [8].

6.1.2 Mule to Client. In this subsection, we take into account steps $6r$ to $9r$ of the response direction, and the time window considered starts when C receives the first message from $M2$.

(i) $M2$'s first message includes $tender_P$ and a price request for disclosing the response payload p_S. The transmission and verification of these data require, also in this case, an amount of time in the order of a few milliseconds.
(ii) Once again, here, an automated negotiation can happen, thus requiring, as before, ~1 sec latency.
(iii) A Client C constantly broadcasts a message, including the payload dimension and the tokens offered. The time window in our evaluation starts when $M1$ receives this message (forward direction, steps 1 and 2).
(iv) Finally, $M2$ sends p_S to C. The same latency applies here for the transmission of such payload; however, here, we have to consider C's verification too. Indeed, in order to verify the correctness of p_S, C has to decrypt it. We provide latency values for some payload dimensions in Table 2. If valid, C can finally send a message that includes the signed $unlockProof$.

Also in this case, the overall latency depends mainly on the transmission time of the payload p_S, but with the addition of the decryption time. However, the total latency (i.e., ~8 seconds plus ~$1,3$ seconds) would make this protocol part feasible in our scenario. This confirms the viability of proper data transmission from the Client to the Mule and vice versa.

Table 3. Methods' Gas Usage for Each Smart Contract in the Protocol

Smart Contract	Method	Gas Usage
ERC20 Token	**approve**	44733
StateChannel	**openChannel**	92285
StateChannel	**closeChannel**	81315
InDaMul	**submitTender**	263990
InDaMul	**submitPayment**	194047

6.2 Smart Contracts Interactions and Gas Usage

The performances in terms of latency for the interactions with the Ethereum public blockchain can significantly vary depending on the transaction fees [42] and/or on the supply and demand levels in the network [34]. Generally, we can expect from 2 to 60 seconds of latency in the interaction with the Ethereum public blockchain; however, maximum and average latencies decrease with the increase in gas prices [42]. With this in view, we measure our experiments in terms of gas, the computation unit defined by the Ethereum protocol [13]. Gas is a unit that measures the amount of computational effort it takes to execute operations in Ethereum Smart Contracts. Thus, the higher the gas usage for a method, the more intense the computation of a blockchain node to execute the method is. Multiplying gas usage by a gas price indicates the actual monetary cost of operating with Ethereum. The Smart Contracts implementation can be found in [3].

Results in Table 3 show the gas usage for the methods involved in the protocol execution and their related Smart Contracts. The *approve* method in the $ERC20Token$ contract is needed to open the channel in the $StateChannel$ contract. This gas usage, together with the *openChannel* and *closeChannel* methods' gas usage, is relatively low and does not deviate much from the other similar application implementations in Ethereum. On the other hand, the $submitTender$ and $submitPayment$ methods in the *InDaMul* contract have a higher gas usage, i.e., ~$263k$ and ~$194k$, respectively. This is due to the fact that these operations involve more data and the execution of signature verification.

At the time of writing, an example of gas price for the Ethereum public blockchain for issuing a transaction within ~30 seconds is ~53 Gwei, i.e., 53 billionth of an Ether (the cryptocurrency of Ethereum). It means that the total price for $submitTender$ would be ~$263k \times 53 = 0.014$ Ether, which is currently ~345 dollars. This price does not represent a feasible option in most scenarios. However, there currently is a rise of technologies that operate using the same protocol for executing Ethereum Smart Contracts but with fewer latencies and reduced gas prices. For instance, in Polygon [2], the $submitTender$ method would cost ~ 0.005 dollars at the time of writing. Alternatively, it would be possible to set up a dedicated permissioned Ethereum blockchain to reduce costs further. Both would be viable alternatives for deploying our framework.

6.3 DFS and Authorization Service

The latencies experienced when interacting with the DFS or the decentralized authorization service are negligible compared to other operations performed in the framework (e.g., Mule's mobility, analyzed in the following subsection). Moreover, for the lack of space, we will refer to our findings in [48] (DFS) and in [47] (authorization) without reporting all the data also in this work. About DFS, for instance, we consider an implementation in IPFS that includes an announcement service (i.e., pub/sub). In this case (Figure 5, PLD2 variable), small messages take on average ~100 ms to get uploaded, while larger ones, e.g., size $1MB$, up to 1 second.

In the case of a decentralized authorization service based on Secret Sharing, the whole process of getting the fragments and decrypting (PLD3 variable) takes approximately a few milliseconds (~50) for small messages and up to ~10 seconds for larger messages ($10MB$).

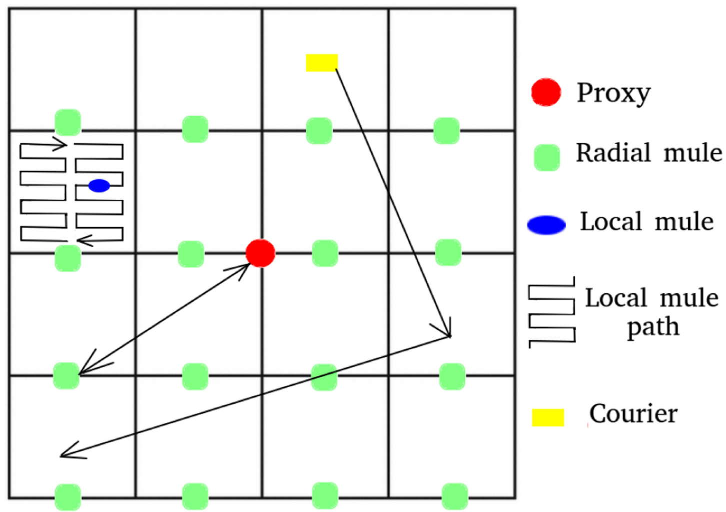


Fig. 6. Representation of the simulated area. The grid is divided into 16 regions. Thus, other than the Client nodes scattered along the area, there are 16 Local Mules, 16 Radial Mules, and the Proxy P at the center of the grid.

6.4 Simulation of Buses Acting as Mules

InDaMul is intended for applications where high latency is an acceptable drawback (delay-tolerant networks). Indeed, the most onerous operation in terms of latency is the Data Mule mobility from the Client to the Proxy, which does not depend on the framework performance but on real-world limitations and decisions the Mules takes. However, minimizing the time it takes for a Client node to contact a Proxy is still desirable. In order to have a clearer idea about the average delay needed for message delivery, we designed and implemented a simulation model for reproducing a scenario where buses and couriers act as data mules.

The idea is to have a population of Client nodes scattered in a simulated virtual environment and a certain number of buses and couriers that, when moving in the virtual space, collect messages from the Client nodes and then bring them to the destination (i.e., the Proxy). In order to model the collaboration among clients and some opportunistic networking, typical of services built-in smart territories, we also enable the possibility of Clients to relay some messages within the Islands (created using State Channels Networks, as discussed in Section 4).

We consider a scenario similar to the previous performance tests in which C, with a fixed location, communicates (using Wi-Fi Direct) with M traveling at a constant speed. Here, we are interested in only those measurements that help us to evaluate the whole communication process. For instance, we consider the maximum Wi-Fi link rate again at 12 Mbps and M's velocity at 36 km/h, as in [8].

Our experiments have been conducted using the LUNES [1] simulator, which is particularly suitable for implementing communication protocols populated by many interacting entities. Through this simulator, we analyze the possible delays (and their specific composition) ranging from the creation of a message (by a Client) up to its delivery (to a Proxy).

6.4.1 Simulation Setup. A squared discrete space composed of 1000×1000 cells was employed as a testbed for the simulations, with such an area representing a unique village populated by several Clients and equipped with couriers and a transport service, as shown in Figure 6. A cell in such a grid represents a 20m × 20m area and potentially contains one or more Client nodes. Therefore, the total surface is composed of 20km × 20km with a density of 25 nodes per km^2. We assume that some of the couriers and some of the buses in the transportation service also act as Mules, covering both the center and the peripheral regions. We thus divided the grid into N^2

squared regions. Specifically, in the proposed scenario, we envision that there are three different types of data mules: (i) Local Mule buses moving only along their region of competence, picking up the messages coming from the Clients it encounters during its path; (ii) Radial Mules buses that connect that region with the center of the village; and (iii) Couriers that are either still or in motion toward a specific destination chosen randomly.

In our model, the interactions are based on proximity; thus, the Clients can deliver their messages only to the Mules (or directly to P) within a specific communication range, which in our scenario was set to 200m (according to what reported in [8], the packet loss ratio at such a distance is only 0.08). We assume that Mules are driving on average at 36 km/h and moving to an adjacent cell at each timestep. Thus, each timestep consists of a discrete unit of time representing 2 seconds. Furthermore, it is not always necessary for the messages to be carried by a Mule. Clients (or couriers) can deliver their messages directly to P (or to the Radial Mule). This happens when they are sufficiently close to communicating with them directly.

Finally, we assume that Mules skipped the announcement service and directly transferred the data to a unique Proxy, P. From now on, we will use the generic term "message" to indicate the data that C sends to $M1$, which needs to reach P, i.e., p_C, $tender_C$, and $address_{M1}$ (shown in Section 3.3).

In our model, there are six types of simulated entities (i.e., nodes):

- **Client nodes** - They represent the generic C of the framework. At each timestep, there is a chance for them to generate a new message to be delivered to P. In our experiments, there are 10,000 Clients randomly placed in the simulated area, with either homogeneous or centralized distribution. In the former case, the nodes are put in the grid completely randomly; in the latter, the probability for a cell to host nodes is inversely proportional to the distance from the center. The centralized distribution aims to reproduce a village-like scenario, where most people live near the center, while the peripheral areas are usually less crowded.
- **Neighbors nodes** - They represent C's Neighbors in its Island. We consider the chance for the Client nodes to relay their messages within their Islands, i.e., small grid sub-regions composed of 20×20 cells. We assume that all the Neighbors can communicate and exchange information since they belong to a common communication network. When a Mule is in a Neighbor node's vicinity, the latter signals to the other Island's nodes the possibility of getting in touch with a Mule and thus delivering the messages. The Clients can deliver the messages to the Mule through one or multiple relays Neighbors.
- **Local Mules** - They move zigzagging along a particular grid region, traversing the local area and picking up the messages from sufficiently close Client nodes. Once a lap is concluded, the Local Mule delivers the messages to the Radial Mule (or directly to P if it is sufficiently close) before starting its route again.
- **Radial Mules** - One for each region of the simulated area, they collect the messages released by a specific Local Mule and then bring them to P at the center of the grid. Then, they return to their original position, waiting again for the Local Mule to complete the lap. For each Local Mule, there is a corresponding Radial Mule.
- **Couriers** - They move according to the Random Waypoint mobility model [12] (i.e., they are either still or in motion, and when they activate, they pick a random destination, moving toward such a point with the same speed of the buses). When couriers collect messages, they carry them until a mule at a reachable distance is found.
- **Proxy node** - Situated at the center of the grid, it is the final destination for all the messages. Just like the Client nodes, P is a static entity.

6.4.2 Performance Evaluation. We performed several tests to measure the delay in delivering the messages and the coverage achieved (i.e., the percentage of Clients nodes that can send messages to a Mule) by varying: (i) the number of Local Mules in the grid; (ii) the distribution of the population; (iii) the presence or the absence of Islands; and (iv) the number of couriers. In our experiments, we employed 16 Local Mules, 16 Radial Mules, and,

Table 4. Delay, Variance, and the Average Number of Hops (i.e., Message Forwarding Toward a Mule or a Proxy) Depending on the Number of Local Mules and the Population Distribution

# Mules	**Distr**	**Avg Delay** ± ***Std*** (seconds)	**Hops** (avg)
9	HOM	6 265 ± *1 984*	1.959
	CENT	5 327 ± *2 077*	1.938
16	HOM	3 345 ± *1 092*	1.959
	CENT	2 658 ± *1 078*	1.933
25	HOM	2 896 ± *968*	1.968
	CENT	2 428 ± *1 022*	1.948

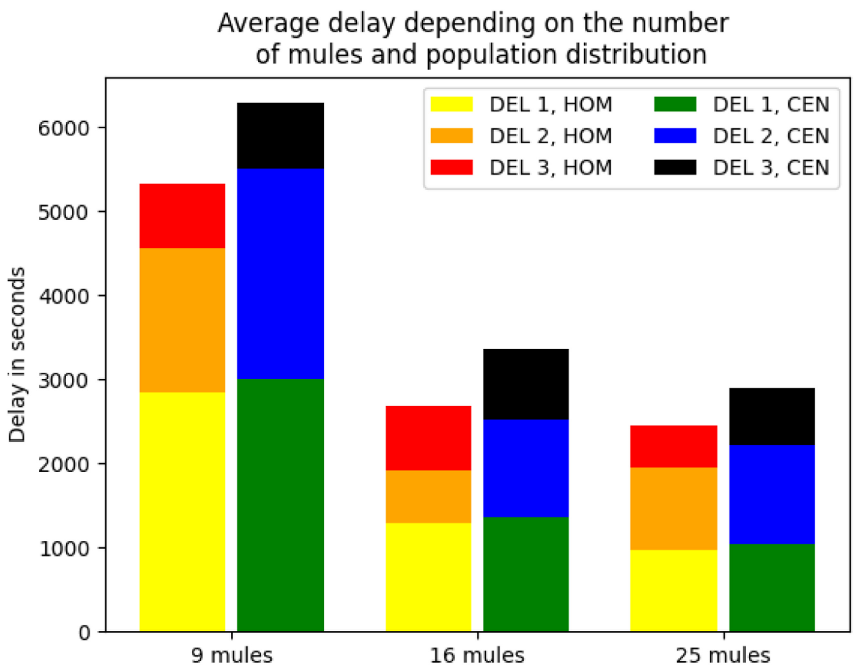


Fig. 7. DEL 1 = delay from the Client to the Local Mule; DEL 2 = delay from the Local Mule to the Radial Mule; DEL 3 = delay from the Radial Mule to the Proxy P; CEN = population with centralized distribution; HOM = population homogeneously distributed.

if any, 16 couriers. The tests lasted 30,000 timesteps to allow the Local Mules to complete their route multiple times. The first tests are performed without involving Neighbors, Islands, and Couriers, thus focusing on Local Mules and Population Distribution. The second tests include all the entities.

6.4.3 Results.

Local Mules and Population Distribution. Table 4 shows the average delay for message delivery without any Courier or Island. As expected, the delay is inversely proportional to the number of Mules. In fact, the higher the number of Mules, the smaller the region each Mule has to cover. Thus, Local Mules are faster to complete a lap and deliver the messages to the Radial Mule. Furthermore, a centralized population favors the speed of delivery since more Clients are placed at the center of the Cartesian place. Thus they can deliver their messages directly to P or a Radial Mule directed toward the center. For the same reason, in our tests, the average number of relays

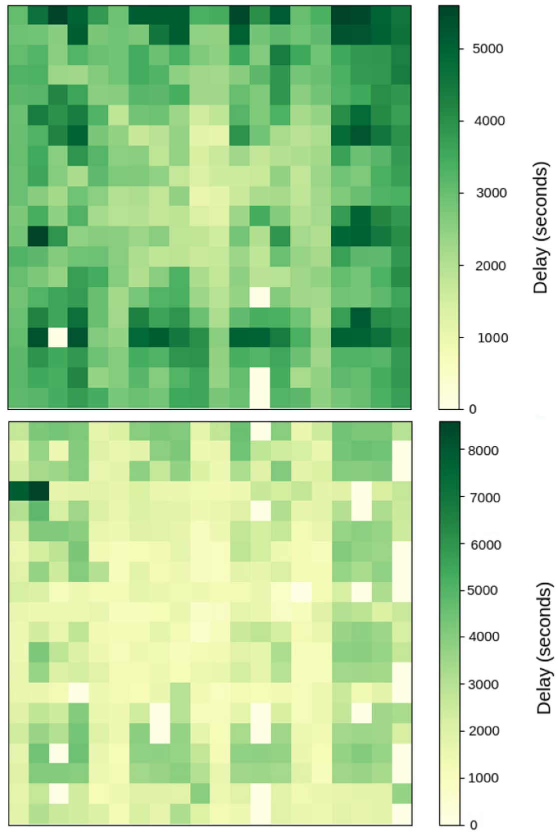


Fig. 8. Heat maps representing the average delay of small regions in a scenario with a homogeneous population. The top figure considers no presence of Islands and no couriers acting as Mules. The bottom one considers the presence of Islands and couriers.

toward mules or proxies (i.e., hops) was slightly smaller with the centralized distribution, and the delay variance was slightly greater. Let us consider, for example, the configuration with 16 Local Mules and a homogeneous population. The messages coming from Clients at the edge of the grid (i.e., distant 250 or fewer cells from the border), which are about 25% of the total messages sent, needed an average delay of 3640 seconds to be delivered to P, compared to 2540 seconds of the messages generated in other locations. Figure 7 shows the composition of the delays in the various configurations. Only messages conveyed via Data Mules have been considered, not those sent directly from Clients to P.

Islands and Couriers. We previously assessed that, as expected, the average delay is inversely proportional to the number of Mules. Thus in the following tests, we have fixed the number of Local Mules to 16 and focused on other aspects. Table 5 shows the metrics retrieved by these tests. As expected, one can notice that, also in this case, with a centralized population, the average delay is significantly reduced. It is also possible to observe how the presence of Islands and couriers positively impacts the coverage achieved, significantly boosting the percentage of nodes reachable by the Mules. The usage of couriers may also entail a higher average delay due to more nodes at the edge of the grid using the framework. Finally, it is interesting to notice that usually, with a centralized population, the achieved coverage is higher (the nodes at the center are easier to contact). It is particularly evident by comparing the two heat maps in Figure 8 where, with a centralized population (despite

Table 5. Average Delay and Standard Deviation (i.e., Std) in Seconds and Coverage Varying Distribution of the Population, Islands Presence and Couriers Presence

		Avg Delay ± ***Std*** (seconds)		**Coverage**	
		ISL	*NOI*	*ISL*	*NOI*
CUR	HOM	2 276 ± *1 202*	3 106 ± *2 324*	98.3%	39.2%
	CENT	1 500 ± *1 197*	2 340 ± *2 311*	96.7%	44.7%
NOC	HOM	2 995 ± *2 557*	3 101 ± *2 320*	61.1%	39.2%
	CENT	2 137 ± *2 415*	2 335 ± *2 298*	71.3%	44.7%

Population is HOM = homogeneous or CEN = centralized. Island is present = ISL, or not = NOI. Couriers are present = CUR, or not = NOC.

the presence of Islands), the areas at the edge of the grid cannot get connectivity. However, this behavior is overturned by employing couriers as well, bringing the coverage from 71.3% to 96.7% for centralized distribution and from 61.1% to 98.3% for homogeneous distribution.

7 CONCLUSIONS

In this article, we present a framework that is thought to ensure the delivery of messages even in areas where Internet coverage is weak or problematic. Specifically, the protocol uses technologies such as DLTs, Smart Contracts, and DFSs, allowing it to run decentralized. Furthermore, an important role is played by Data Mules, having the task of transporting the data from the source to a Proxy charged with the message delivery. After analyzing the most concerning security issues, we investigate the feasibility of the framework's usage in real-life scenarios. The time required to send data from the Client to the Mule (and vice versa) depends mainly on the payload size. Thus, the transmission from/to a moving vehicle can be considered viable and reliable for messages with a reasonable dimension. Performing such operations with Ethereum can currently be quite expensive, but this limitation can be overcome by employing emerging technologies that can help us significantly reduce gas prices. Finally, the simulation of a scenario where means of transportation act as Data Mule is performed, showing a raw estimate of the potential overall delay.

REFERENCES

[1] 2021. Dataset and scripts GitHub repository. Retrieved from https://github.com/luca-Serena/lunes-tdm-islands. Date Accessed December 2022.

[2] 2021. Polygon - Ethereum.s Internet of Blockchains. Retrieved from https://polygon.technology. Date Accessed December 2022.

[3] 2021. TruDaMul. Retrieved from https://github.com/AnaNSi-research/TruDaMul. Date Accessed December 2022.

[4] 2021. Umbral-rs and tests. Retrieved from https://github.com/miker83z/umbral-rs. Date Accessed December 2022.

[5] Agencia Espanola Proteccion Datos. 2019. *Introduction to the Hash Function as a Personal Data Pseudonymisation Technique*. Technical Report. Retrieved from https://edps.europa.eu/sites/edp/files/publication/19-10-30_aepd-edps_paper_hash_final_en.pdf.

[6] Giuseppe Anastasi, Marco Conti, and Mario Di Francesco. 2008. Data collection in sensor networks with data mules: An integrated simulation analysis. In *Proceedings of the 2008 IEEE Symposium on Computers and Communications*. IEEE.

[7] Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. 2022. Hide & seek: Privacy-preserving rebalancing on payment channel networks. In *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 358–373.

[8] Arunn Balasundram, Tharaka Samarasinghe, and Dileeka Dias. 2016. Performance analysis of Wi-Fi direct for vehicular ad-hoc networks. In *Proceedings of the 2016 IEEE International Conference on Advanced Networks and Telecommunications Systems*. IEEE, 1–6.

[9] Alexandre C. Barbosa, Thays A. Oliveira, and Vitor N. Coelho. 2018. Cryptocurrencies for smart territories: An exploratory study. In *Proceedings of the 2018 International Joint Conference on Neural Networks*. IEEE, 1–8.

[10] Souvik Basu, Soumyadip Chowdhury, and Sipra Das Bit. 2021. Using blockchain in intermittently connected network environments. In *Blockchain Technology and Innovations in Business Processes*. S. Patnaik, T. S. Wang, T. Shen, and S. K. Panigrahi (Eds.), Springer, 33–47.

[11] Juan Benet. 2014. IPFS-content addressed, versioned, P2P file system. Retrieved from https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf. Date Accessed December 2022.

[12] Christian Bettstetter, Hannes Hartenstein, and Xavier Pérez-Costa. 2004. Stochastic properties of the random waypoint mobility model. *Wireless Networks* 10, 5 (2004), 555–567.

[13] Vitalik Buterin. 2013. Ethereum white paper. Retrieved from https://github.com/ethereum/wiki/wiki/White-Paper. Date Accessed December 2022.

[14] Arnab Chakrabarti, Ashutosh Sabharwal, and Behnaam Aazhang. 2003. Using predictable observer mobility for power efficient design of sensor networks. In *Information Processing in Sensor Networks*. F. Zhao and L. Guibas (Eds.), Springer.

[15] Nicolas T. Courtois and Rebekah Mercer. 2017. Stealth address and key management techniques in blockchain systems. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*.

[16] Mauro Margalho Coutinho, Alon Efrat, Thienne Johnson, Andrea Richa, and Mengxue Liu. 2014. Healthcare supported by data mule networks in remote communities of the Amazon region. *International Scholarly Research Notices* 2014 (2014), 8 pages. https://doi.org/10.1155/2014/730760

[17] G. D'Angelo, S. Ferretti, and V. Ghini. 2017. Multi-level simulation of internet of things on smart territories. *Simulation Modelling Practice and Theory* 73 (2017), 3–21. DOI: https://doi.org/10.1016/j.simpat.2016.10.008

[18] Enes Erdin, Suat Mercan, and Kemal Akkaya. 2021. An evaluation of cryptocurrency payment channel networks and their privacy implications. arXiv:2102.02659. Retrieved from https://arxiv.org/abs/2102.02659

[19] Peyman Faratin, Carles Sierra, and Nick R. Jennings. 1998. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems* 24, 3–4 (1998), 159–182.

[20] Pietro Ferraro, Christopher King, and Robert Shorten. 2018. Distributed ledger technology for smart cities, the sharing economy, and social compliance. *IEEE Access* 6 (2018), 62728–62746.

[21] Stefano Ferretti. 2013. Shaping opportunistic networks. *Computer Communications* 36, 5 (2013), 481–503. DOI: https://doi.org/10.1016/j.comcom.2012.12.006

[22] Stefano Ferretti and Gabriele D'Angelo. 2016. Smart shires: The revenge of countrysides. In *Proceedings of the IEEE Symposium on Computers and Communications*. IEEE Computer Society, Washington, DC. DOI: https://doi.org/10.1109/ISCC.2016.7543862

[23] Stefano Ferretti, Gabriele D'Angelo, and Vittorio Ghini. 2016. Smart multihoming in smart shires: Mobility and communication management for smart services in countrysides. In *Proceedings of the IEEE Symposium on Computers and Communications*. IEEE Computer Society, Washington, DC. DOI: https://doi.org/10.1109/ISCC.2016.7543862

[24] Michèle Finck. 2019. *Blockchain and the General Data Protection Regulation: Can Distributed Ledgers be Squared with European Data Protection Law?: Study*. European Parliament.

[25] Manuel Jesús-Azabal, Juan Luis Herrera, Sergio Laso, and Jaime Galán-Jiménez. 2021. OPPNets and rural areas: An opportunistic solution for remote communications. *Wireless Communications and Mobile Computing* 2021 (2021), 11 pages. https://doi.org/10.1155/2021/8883501.

[26] Carsten Keßler and Grant McKenzie. 2018. A geoprivacy manifesto. *Transactions in GIS* 22, 1 (2018), 3–19.

[27] Julio Navío-Marco, Beatriz Rodrigo-Moya, and Paolo Gerli. 2020. The rising importance of the "Smart territory" concept: Definition and implications. *Land Use Policy* 99 (2020), 105003–105007.

[28] Joseph Poon and Thaddeus Dryja. 2015. The bitcoin lightning network: Scalable On-Chain Instant Payments. Retrieved from https://1bitcoin.ca/s/lightning-network-paper.pdf. Accessed Date December 2022.

[29] Edoardo Rosa, Gabriele D'Angelo, and Stefano Ferretti. 2019. Agent-based simulation of blockchains. In *Methods and Applications for Modeling and Simulation of Complex Systems. AsiaSim 2019*. G. Tan, A. Lehmann, Y. Teo, and W. Cai (Eds.), Springer, 115–126.

[30] Luca Serena, Gabriele D'Angelo, and Stefano Ferretti. 2022. Security analysis of distributed ledgers and blockchains through agent-based simulation. *Simulation Modelling Practice and Theory* 114 (2022), 102413–102417.

[31] Luca Serena, Mirko Zichichi, Gabriele D'Angelo, and Stefano Ferretti. 2021. Simulation of dissemination strategies on temporal networks. In *Proceedings of the 2021 Annual Modeling and Simulation Conference*. IEEE.

[32] Rahul C. Shah, Sumit Roy, Sushant Jain, and Waylon Brunette. 2003. Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks* 1, 2–3 (2003), 215–233.

[33] Adi Shamir. 1979. How to share a secret. *Communications of the ACM* 22, 11 (1979), 612–613.

[34] Michael Spain, Sean Foley, and Vincent Gramoli. 2020. The impact of Ethereum throughput and fees on transaction latency during ICOs. In *Proceedings of the International Conference on Blockchain Economics, Security and Protocols*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[35] Mathis Steichen, Beltran Fiz, Robert Norvill, Wazen Shbair, and Radu State. 2018. Blockchain-based, decentralized access control for IPFS. In *Proceedings of the 2018 IEEE International Conference on Internet of Things*. IEEE, 1499–1506.

[36] Ryo Sugihara and Rajesh K. Gupta. 2009. Optimal speed control of mobile node for data collection in sensor networks. *IEEE Transactions on Mobile Computing* 9, 1 (2009), 127–139.

[37] Weizhao Tang, Weina Wang, Giulia Fanti, and Sewoong Oh. 2020. Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–39.

[38] Steffen Hess, Katalin Kolosy, Eamon O.Hara, Veneta Paneva, and Paul Soto. 2018. Smart villages: revitalising rural services. *EU Rural Review*, Vol. 26 (2018), 1–52.

[39] Fabian Vogelsteller and Vitalik Buterin. 2015. *ERC-20 Token Standard*. Ethereum Foundation (Stiftung Ethereum), Zug.

[40] Shangping Wang, Yinglong Zhang, and Yaling Zhang. 2018. A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. *IEEE Access* 6 (2018), 38437–38450.

[41] Wenchao Xu, Weisen Shi, Feng Lyu, Haibo Zhou, Nan Cheng, and Xuemin Shen. 2019. Throughput analysis of vehicular internet access via roadside WiFi hotspot. *IEEE Transactions on Vehicular Technology* 68, 4 (2019), 3980–3991.

[42] Lin Zhang, Brian Lee, Yuhang Ye, and Yuansong Qiao. 2021. Evaluation of Ethereum end-to-end transaction latency. In *Proceedings of the 2021 11th IFIP International Conference on New Technologies, Mobility and Security*. IEEE.

[43] Mingliu Zhang and Richard S. Wolff. 2010. A border node based routing protocol for partially connected vehicular ad hoc networks. *Journal of Communications* 5, 2 (2010), 130–143.

[44] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. 2020. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems* 105, C (2020), 475–491.

[45] Mirko Zichichi, Michele Contu, Stefano Ferretti, and Gabriele D'Angelo. 2019. LikeStarter: A smart-contract based social DAO for crowdfunding. In *Proceedings of the 2nd Workshop on Cryptocurrencies and Blockchains for Distributed Systems*.

[46] Mirko Zichichi, Stefano Ferretti, and Gabriele D'Angelo. 2020. A distributed ledger based infrastructure for smart transportation system and social good. In *Proceedings of the 2020 IEEE 17th Annual Consumer Communications & Networking Conference*. IEEE, 1–6.

[47] Mirko Zichichi, Stefano Ferretti, Gabriele D'Angelo, and Víctor Rodríguez-Doncel. 2020. Personal data access control through distributed authorization. In *Proceedings of the 2020 IEEE 19th International Symposium on Network Computing and Applications*. IEEE, 1–4.

[48] Mirko Zichichi, Stefano Ferretti, and Gabriele D'Angelo. 2020. A framework based on distributed ledger technologies for data management and services in intelligent transportation systems. *IEEE Access* 8 (2020), 100384–100402.

Received 4 July 2022; revised 17 December 2022; accepted 5 March 2023