DLT-based Data Mules for Smart Territories

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*—Many services that are taken for granted in smart cities are not even remotely available in dislocated areas yet, due to the lack of or too costly wide area network connectivity. With the aim to offer a practical and secure way to transport data and allow for communications in such constrained scenarios, we focus on the problem of incentivizing to data mules, i.e. devices dedicated to enable the data transfer even in the absence of the Internet. Our solution combines the use of several distributed technologies for verifying the correct behavior of all the participants and incentivize them. We focus on the use of state channels to support the flow of smart-contract-based tokens as a form of payment, in a condition where participants communicate only with others in physical proximity. Furthermore, we validate the viability of the application through the simulation of peer-to-peer interactions between the participants. In this work we achieve positive results in terms of communication latency and percentage of client nodes which are able to benefit from the system.**

***Index Terms*—Data Mules, Distributed Ledger Technologies, State Channels, Smart Territories**

I. Introduction

Nowadays, we are at a crossroad. People are to some extent being constrained to move to big, smart cities, due to the higher opportunities in terms of work and offered services. Conversely, recent events, such as the COVID-19 pandemic, have shown how this trend can be reversed, with an increasing number of people deciding to move to the countryside and rural areas. Almost, since for sure many services that are taken for granted in (smart) cities, are not even remotely available in dislocated areas yet. For some underprivileged territories 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 [1]. We argue that what is needed is a set of novel opportunistic solutions, which allows us to share and reuse data, services, computation and bandwidth. Such a solution would simplify the development of new services and the integration of legacy technologies into new ones. Well-known examples consist of technologies such as multi-homing mobile services, mobile ad-hoc networks, opportunistic networks, peer-to-peer and fog computing systems [1]. In this novel "smart territory" case, however, such applications might not be supported by a wide area network connectivity, and certain networking solutions might result as too costly (e.g. satellite connections). Data Mules (that is an acronym for Mobile Ubiquitous LAN Extensions [2]) 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 offline areas.

InDaMul [3] is a decentralized application that combines the use of Distributed Ledger Technologies (DLTs) and Decentralized File Storages (DFS), mostly for verifying the correct behavior of all the participants and to incentivize them in Data Mule-based communications. Smart contracts enable an automated validation of claims, e.g. simply by verifying a signature, but, scalability and costs issues are still a limit [4]. Generally, layer-two protocols are exploited to overcome these limits, enabling transactions between users through the exchange of authenticated messages via a medium which is outside of, but tethered to, a layer-one DLT [5]. In InDaMul, we are interested in a layer-two protocol that can also be executed without the need of constantly being connected to the Internet and where the communication conducted only among nodes in the physical vicinity suffices. For this reason, we resort to state channel protocols and to the creation of networks based on these [6]. State channels are used to support the flow of smart-contract-based tokens as a form of payment between Data Mules and offline Clients (e.g. being in a no broadband connection area), in order to send data to an online Server. Moreover, Clients and their physical Neighbors can build up a state channel network supporting the creation of "Islands" served by Data Mules. Our main contributions in this paper are about introducing the application that enables incentives in the data transport process of Data Mules using state channels and the validation of the proposed approach through the simulation of a smart territory. The remainder of this paper is organized as follows. Section II provides the background and related works. In Section III the application is presented, while in Section IV, we discuss the use of state channels networks. In Section V, we present the experimental evaluation and, finally, Section VI provides the conclusions.

II. Background and Related Work

A. Data Mules

Data Mules are some types of devices that are able to collect data from sensors and to exploit their own mobility to carry the information to destination (or to intermediaries) using a

This work has received funding from the EU H2020 research and innovation programme under the MSCA ITN European Joint Doctorate grant agreement No 814177 LAST-JD - RIoE and from the University of Urbino through the "Bit4Food" research project.

wireless short-range communication medium [7]. Depending on the context, Mules can either be transportation vehicles like buses or cars or even walking persons. As a result of their movement between remote areas, they effectively create a data communication link [8]. In the last few years, many research works have been presented on Data Mules. For instance, in [2] 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. The characteristics of the random walk mobility model are used to analyze the predicted performance of the model. 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 [8], [9].

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 [10]. Smart Contracts are instructions stored in the blockchain and automatically triggered once a required condition is met. We will refer to Ethereum [10] due to its public open source blockchain that is in widespread use and for its provision of robust Smart Contract development tools. Smart contracts make it possible to build structures that act as second layer cryptocurrencies, i.e. tokens, and in particular ERC-20 tokens [11]. Furthermore, these enable the development of rewards and incentives based crowdsourced services. In essence, participation in collaboration-based services can be traced and rewarded thanks to Smart Contracts. In particular, a Smart Contract can be devoted to the distribution of digital tokens to those parties that share its resources or cooperate in a communication protocol [12], [13].

C. 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 [6]. State channels are regulated through smart contracts that manage the validation of the channel's payments. A prominent implementation in the Ethereum blockchain is μRaiden [6], an open source framework that is 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. 1^{st} 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 by using the public-private key pair used 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. 2^{nd} 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.

The $balance$ message contains a set of information used by the two users for updating the balance in the state channel. It contains: (i) the address of the other party the state channel has been opened with; (ii) the on-chain state channel unique identifier, e.g. the block number the channel was opened; (iii) the updated balance. 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. These consist, indeed, in the establishment of state channel networks, where among many state channels the participants pay by using other participants as relays, essentially forming a connected network. It is specifically a Layer-2 network application running on top of the Layer-1 services of a cryptocurrency [6]. This is the main idea behind the Lightning Network and the Raiden Network [6].

D. LUNES

LUNES (Large Unstructured Network Simulator) is an open source discrete event simulator, specifically designed for the modelling and performance evaluation of complex networks and communication protocols. LUNES is implemented on top of ARTÌS/GAIA simulation middleware, which offers the primitives for time management and communication among simulated entities, enabling also a parallel and distributed execution [14].In previous works, some versions of this tool have been employed to model blockchain behaviour, edge computing scenarios or temporal networks and in this work a version of LUNES has been developed for InDaMul [15]. This version of the software combines a part where the mobility of the data mules is handled, a part intended to manage the exchange of messages and, finally, a part responsible for collecting and assembling data during the execution.

III. InDaMul Application

We provided a summarized description of the protocol behind the InDaMul application in a previous work [3]. In here, we provide a more detailed description of the protocol and an insight into the application through smart contracts. This will help the reader to a better understanding of the contribution of this work, which is a scalable implementation of the incentivized data mule application through the use of state channels. In the following, we use Figure 1 as reference.

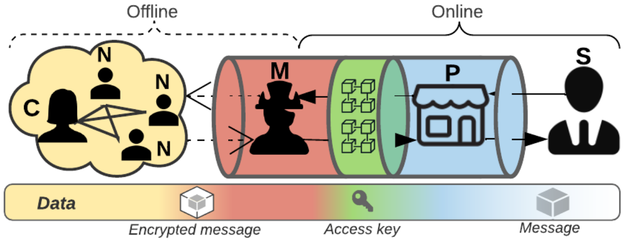

Fig. 1. Graphical representation of the InDaMul application.

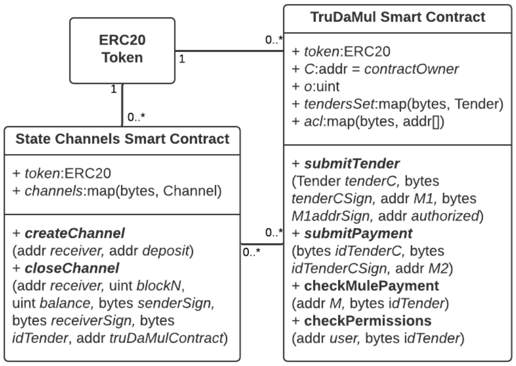

Fig. 2. UML class diagram representing the smart contracts used in InDaMul.

The InDaMul application is intended to work for any Client (C) that finds itself in an offline condition, to enable it to send a message to any Server (S) that is online. The application includes a set of technologies and a protocol where a Data Mule (M) takes care of retrieving (offline) the payload of C and to bring it to a Proxy (P), which in turn forwards (online) the message to S. A message can be returned from S to C using the same protocol but in the opposite way. The application includes: (i) a *Smart Contract enabled DLT*, that enables payments and information verification through Smart Contracts (we refer to the Ethereum specification [10]); (ii) an *announcement service*, to enable the exchange of announcements between online nodes, e.g. a publish/subscribe room; (iii) a *DFS*, for storing data using immutable identifiers and to enable asynchronous communication between Mules and Proxies; (iv) a *decentralized authorization service*, that enables access to encrypted data through a network of nodes that only operate following Smart Contract dictated policies; we refer to the implementation shown in [16] . The protocol allows C and S to communicate and it can be divided in two separate 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. During the execution of the protocol, in particular when Mules and Proxy are online, a set of Smart Contracts is used, i.e. the ones shown in Figure 2. A unique ERC20 Token [11] smart contract is used to enable any payment exchange among all the involved actors.

A. Client and Data Mule Communication

A State Channel 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 its geographical zone. Whenever C is willing to send a message to a Server S, it waits for a Mule and, meanwhile, it broadcasts a request for taking charge of the message. This message consists of a plaintext that has been encrypted using S's public key. Subsequently, a payload p_C is obtained by encrypting the message with a symmetric key x, identified through an id, i.e. id_x (this key will be used by P to obtain the message to send to S).

When a Mule, $M1$, passes nearby C, it receives a request containing the payload dimension (in byte) and the tokens offered for the job. If $M1$ accepts, then C transmits to $M1$ the payload p_C, a $balance_{M1}$ and a $tender_C$, signed by C using its private key. The $balance$ is a state channel object used for the balance update containing the identifier of the channel, the actual balance and the addresses of the smart contract and of the involved parties. The $tender$ in an object containing:

- EID - an exchange alphanumeric identifier that acts also as a nonce;
- URI_p - an immutable URI, e.g. a hash pointer, that identifies a payload;
- *offer* - a numerical value representing C's offer to P;
- id_x - the id of the key used for obtaining p_C;

Next, then, $M1$ will take care of delivering payload p_C to a Proxy P, which in turn will contact S (this process is explained in detail in the next sub section). Whenever S has to send a response message to C, P would take care of S's message by encapsulating it in a new payload p_S. Then, this payload will reach a new Mule, i.e. $M2$, that can reach the vicinity of C. The former transmits to the latter: (i) a price request (in tokens) for transmitting p_S, and (ii) a proof that it is carrying a message to deliver signed by P. This proof is contained in $tender_P$ created previously by P. Once C reaches an agreement on the price's request, it sends to $M2$ a $balance_{M2}$ object for updating the balance in their state channel. $M2$, then, transmits p_S and C can decrypt it in order to check its validity. If valid, C replies to $M2$ with a valid response that the latter can submit to the *InDaMul* smart contract signature extraction operation in order to unlock the payments (for $M2$ and for P).

B. Data Mules and Proxies Communication

A *InDaMul* Smart Contract executes the majority of the protocol tasks and thus requires that C deposits an amount of tokens to pay Mules and Proxies directly. When Mules become online they can forward the Clients' payloads to Proxies, or vice-versa, Proxies can reach Clients through available Data Mules. Whenever a Mule $M1$ is in charge of delivering C's payload p_C to a Proxy, it can directly publish an announce in an announcement service in order to reach an audience of different Proxies. While announcing the tender, $M1$ also uploads p_C to a DFS. A Proxy P, that decides to take charge of p_C, simply invokes a method in the *InDaMul* Smart Contract owned by C. The *submitTender* method (see Algorithm 1) 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. This 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, i.e. a subset of blockchain nodes maintaining shares of the key x using the Secret Sharing technique [16]. Each authorization node autonomously checks the *InDaMul* 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 and decrypts the payload p_C, previously obtained from the DFS. The obtained message is sent to S.

Whenever P has to relay a response from S to C, it can publish an announcement directly in the announcement service in order to reach an audience of several Data Mules. P creates the new payload p_S containing S's response message or a proof that S did not reply, encrypted using C's public key and uploads it 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. A Mule, $M2$, that wants to take charge of $tender_P$, downloads the payload p_S from the DFS and then sends a signed request to the decentralized authorization service for accessing the key for decrypting C's location information. This key is released only to Mules that opened a state channel with C. Then, $M2$ will be able to reach C and start communicating for delivering the payload.

C. On-chain and Off-chain Payments

During the protocol execution, several on-chain and off-chain payments are performed using a mixture of state channel payments, locks and automatic payments.

When C assigns M to carry a new message, the former also issues the latter with a $balance_{M1}$ object that updates the balance in their state channel. However, this balance would be valid for closing the state channel, i.e. getting paid, only after $M1$ announces p_C and P invokes the *InDaMule* Smart Contract's *submitTender* method. Otherwise, $M1$ can only close the channel using a previous valid balance object. For what concerns P, the *submitTender* method automatically locks an amount of tokens indicated by C in favor of P, once it has checked the validity of the data submitted, i.e. digital signatures. This amount is locked until a response reaches C through $M2$. $M2$ finally, gets paid using a $balance_{M2}$ object and response for the challenge–response authentication that, when uploaded to the *InDaMule* Smart Contract through *submitPayment* , will unlock both $M2$'s and P's payments.

IV. The Island: a Local State Channel Network

In the case in which a Client C does not find itself within the action range of Mules, a network can be set up between C's Neighbors, i.e. N. The nodes in this "Island" are in C's physical proximity and most of them are isolated (in terms of communication) from the rest of the territory. However, 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.

Algorithm 1: *submitTender* of *InDaMul*

```
Global Data:
- $this_{addr}$ address of the InDaMul smart contract
- $C_{addr}$ address of client $C$ (the contract owner)
- $token$ ERC20 Token
- tendersSet set of tenders used in the past
- $oAmount$ amount of tokens reserved for proxies in tenders
- acl access control list for keys
Input:
- $tender_C$ object containing an $id$, the sender tokens offer
for proxies and $id_x$
- $s_{tender_C}$ the signature of $tender_C$
- $M1_{addr}$ the mule address
- $s_{M1_{addr}}$ the signature of $M1_{addr}$
- $P_{addr}$ 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
    extractedAddr1 ← verify($tender_C$, $s_{tender_C}$)
    extractedAddr2 ← verify($M1_{addr}$, $s_{M1_{addr}}$)
    if extractedAddr1 == extractedAddr1 == $C_{addr}$ then
        // identity confirmed
        allowMulePayment($M1_{addr}$, $tender_C.id$)
        if $token.balanceOf(this_{addr}) - o > tender_C.offer$
         then
            // if enough balance then
            tendersSet.add($tender_C$)
            $oAmount$ = $oAmount$ + $tender_C$.offer
            acl.map($tender_C.id_x$, $P_{addr}$)
        end
    end
return
```

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 is found (e.g. a moving device).

It is worth noticing 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). For space reasons, we will not describe this procedure in detail and it is left as future work. During the operations, the Neighbors in 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) [14].

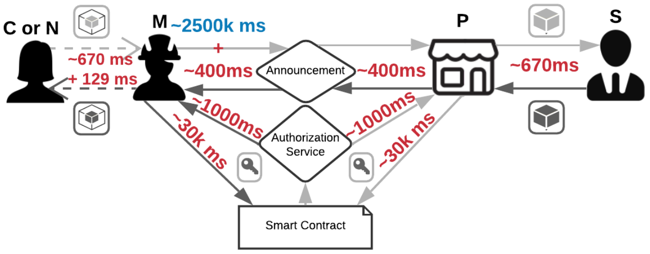

Fig. 3. Latency (order of magnitude) for each interaction in InDaMul. Each latency is obtained by the sum of an average latency for the application execution with a payload of dim. 1 MB. The latency number in blue represents the latency of the mule mobility and it is discussed in Section V.

A Neighbor that decides to send a message to a Server S becomes a Client C and seeks to reach a Data Mule M. We refer to the Raiden protocol [6]. 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 finally reach M through TN.

- 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.
- C gets a *secretRequest* message and checks its validity. Receiving this one means that C can safely assume the *lockedTransfer* message has arrived to TN, and that the latter is incentivized 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 is receives a *revealSecret* message from the first N_1 in the path.

V. Experimental Evaluation

We conducted a set of experiments to evaluate the InDaMul application performance in terms of latency and Smart Contracts operations cost. Due to the complexity of the application and the very different technologies / interactions among involved entities, it was not convenient to implement a single, unified testbed environment for evaluating the whole application. This also because in different steps of the protocol, there are different metrics and aspects that need to be considered. We can notice, indeed, that the different steps involve latencies that are of different orders of magnitude, as shown in Figure 3. InDaMul is intended for applications where high latency is an acceptable drawback. 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 application performance, but rather on real world limitations and decisions taken by the Mules. However, it is still desirable to minimize the time that it takes for a Client node to contact a Proxy. In order to have a clearer idea about the average delay needed for messages delivery, we designed and implemented a simulation.

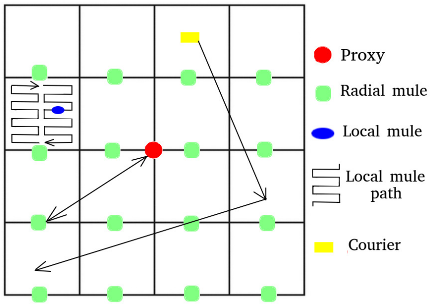

Fig. 4. Representation of the simulated area. The grid is divided into 16 regions. Thus, other than the Client nodes scattered along the area, there are 16 Local Mules, 16 Radial Mules and the Proxy P at the center of the grid.

A. Scenario

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 reproduce collaboration among clients and add some kind of opportunistic networking, typical of services to be 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 IV. We consider a scenario in which C, with a fixed location, communicates (using Wi-Fi Direct) with M that is traveling at a constant speed. The details for a more in-depth analysis on the physical and communication medium considerations of such a scenario can be found in [17], [18]. 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 at 12 Mbps and M's velocity at 36 km/h, as in [17]. Our experiments have been conducted using the LUNES [15] simulator, which is particularly suitable for implementing communication protocols populated by a large number of 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).

B. Simulation Setup

A squared discrete space composed of 1000 x 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 4. A cell in such a grid represents a 20m x 20m area and potentially contains one or more Client nodes. Therefore, the total surface is composed of 20km x 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. 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 certain communication range, which in our scenario was set to 200m (according to what reported in [17], the packet loss ratio at such a distance is only 0.08). We assume that Mules are moving on average at 36 km/h and that at each time-step they move to an adjacent cell. Each time-step, thus, consists in a discrete unit of time that represents 2 seconds. Furthermore, not always all the messages need to pass from a Mule. In fact, Clients (or couriers) can deliver their messages directly to P (or to the Radial Mule) if they are sufficiently close to directly communicate with them. Finally, we assume that Mules skip the announcement service and directly transfer 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 $address_{M1}$ In our model there are five types of simulated entities:

- **Client nodes** - They represent the generic C of the application. At each time-step, 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 aim of the centralized distribution is to reproduce a village-like scenario, where most of the 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 own Islands, i.e. small sub-regions of the grid composed of 20 x 20 cells, where we assume that all the Neighbors can communicate and exchange information since they belong to a common communication network. When a Mule is in the vicinity of a Neighbor node, the latter signals to the other nodes within the Island the possibility to relay messages.
- **Local Mules** - They move zigzagging along a certain region of the grid, 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 come back 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, i.e. they are either still or in motion, and when they activate they pick a random point, moving towards it with the same speed of buses. Couriers collect messages to carry them until a reachable mule is found.
- **Proxy node** - Situated at the center of the grid, it is the final destination for all the messages. Just as the Client nodes, P is a static entity.

TABLE I
AVERAGE DELAY & STANDARD DEVIATION (I.E. STD) IN SECONDS.
POPULATION IS HOM = HOMOGENEOUS OR CEN = CENTRALIZED.
ISLAND IS PRESENT = ISL, OR NOT = NOI.
COURIERS ARE PRESENT = CUR, OR NOT = NOC.

		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%

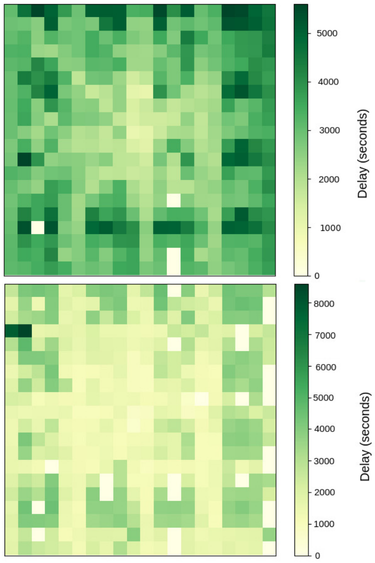

Fig. 5. Heat maps representing the average delay in a scenario with a homogeneous population. The top map considers no presence of Islands and couriers. The bottom one considers their presence.

C. Results

We performed several tests with the purpose of measuring the delay for delivering the messages and the coverage achieved (i.e. the percentage of Clients nodes which are able to send messages to a Mule), by varying: (i) the number of couriers, (ii) the presence or the absence of Islands, (iii) the distribution of the population. In our experiments, we employed 16 Local Mules, 16 Radial Mules and, if any, 16 couriers. We previously assessed that, as expected, the average delay is inversely proportional to the number of Mules, so we focus on other aspects. The tests lasted 30000 time-steps, in order to allow the Local Mules to complete their route multiple times. As mentioned earlier, the population can be either distributed in a uniform, homogeneous way (i.e. Clients are placed randomly along the grid) or following a biased distribution probability that favors the center of the area, rather than at the border. Table I shows the metrics retrieved by these tests. As expected, one can notice that with a centralized population the average delay is significantly reduced, because most of the nodes are located near to the center, where it is easier to directly get in touch with the Proxy or to meet one of the Radial Mules on the way to the center. It is also possible to observe how the presence of Islands and couriers has a positive impact on 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 application. 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). This is particularly evident by comparing the two heat maps in Figure 5 where, with centralized population (despite the presence of Islands) the areas at the edge of the grid cannot get connectivity. However, this behaviour 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.

D. Smart Contracts Performances Discussion

Generally speaking, we can expect from 2 to 60 seconds of latency in the interaction with the Ethereum public blockchain, however average latencies decrease with the increase in gas prices [4]. With this in the view, we measure our experiments in terms of gas, following the Ethereum protocol [10]. The Smart Contracts implementation can be found in [19]. The gas usage of the $approve$ (44 733), $openChannel$ (92 285) and $closeChannel$ (81 315) methods are relatively low and do 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. $\sim 246k$ and $\sim 170k$ respectively. This is due to the fact that these operations involve more data and the execution of signatures verification. At the time of writing, an example of gas price for the Ethereum public blockchain for invoking $submitTender$ in a transaction within $\sim$ 30 seconds is $\sim$ 53 Gwei, i.e. ~ 43.62 dollars. Obviously, this price does not represent a feasible option in most scenarios. However, executing it in Ethereum sidechains, such as Polygon [10], would cost, at the time of writing, around $\sim$ 0.005 dollars, making it a viable alternative for deploying the application.

is then charged with the message delivery. After describing

VI. Conclusions

We presented an application that enables the delivery of messages in locations where the Internet coverage is problematic. The main idea is to make use of technologies such as DLTs and DFS to achieve the decentralization of the service. Data Mules have a fundamental role, since they collect and transport the information from the source to a Proxy, that the application, we carried out an experimental evaluation for retrieving a raw estimate of the delay required to exploit the application. Specifically, we focused on the Mules mobility since other delays, by comparison, are negligible. We considered a village-like scenario where buses and couriers act as Data Mule. The simulations, performed with an agent-based simulator called LUNES proved that the presence of islands and couriers ensures connectivity to a significant number of nodes. The application is therefore feasible for such use case, even though gas cost in the Ethereum public blockchain might represent a concerning issue for the execution.

References

[1] S. Ferretti, G. D'Angelo, and V. Ghini, "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, 2016.

[2] R. C. Shah, S. Roy, S. Jain, and W. Brunette, "Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks," *Ad Hoc Networks*, vol. 1, no. 2-3, pp. 215–233, 2003.

[3] M. Zichichi, L. Serena, S. Ferretti, and G. D'Angelo, "Incentivized data mules based on state-channels," in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2022.

[4] L. Zhang, B. Lee, Y. Ye, and Y. Qiao, "Evaluation of ethereum end-to-end transaction latency," in *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2021.

[5] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020.

[6] E. Erdin, S. Mercan, and K. Akkaya, "An evaluation of cryptocurrency payment channel networks and their privacy implications," *arXiv preprint arXiv:2102.02659*, 2021.

[7] G. Anastasi, M. Conti, and M. Di Francesco, "Data collection in sensor networks with data mules: An integrated simulation analysis," in *2008 IEEE Symposium on Computers and Communications*. IEEE, 2008.

[8] M. M. Coutinho, A. Efrat, T. Johnson, A. Richa, and M. Liu, "Healthcare supported by data mule networks in remote communities of the amazon region," *International scholarly research notices*, vol. 2014, 2014.

[9] M. Jesús-Azabal, J. L. Herrera, S. Laso, and J. Galán-Jiménez, "Oppnets and rural areas: An opportunistic solution for remote communications," *Wireless Communications and Mobile Computing*, vol. 2021, 2021.

[10] Polygon, 2021. [Online]. Available: https://polygon.technology/papers/

[11] F. Vogelsteller and V. Buterin, "Erc-20 token standard," *Ethereum Foundation (Stiftung Ethereum), Zug, Switzerland*, 2015.

[12] P. Ferraro, C. King, and R. Shorten, "Distributed ledger technology for smart cities, the sharing economy, and social compliance," *IEEE Access*, vol. 6, pp. 62 728–62 746, 2018.

[13] M. Zichichi, L. Serena, S. Ferretti, and G. D'Angelo, "Complex queries over decentralised systems for geodata retrieval," *IET Networks*, pp. 1–16, 2022, doi: 10.1049/ntw2.12037.

[14] L. Serena, M. Zichichi, G. D'Angelo, and S. Ferretti, "Simulation of dissemination strategies on temporal networks," in *2021 Annual Modeling and Simulation Conference (ANNSIM)*. IEEE, 2021.

[15] "Dataset and scripts github repository," April 2022. [Online]. Available: https://github.com/luca-Serena/lunes-tdm-islands

[16] M. Zichichi, S. Ferretti, G. D'Angelo, and V. Rodríguez-Doncel, "Personal data access control through distributed authorization," in *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2020, pp. 1–4.

[17] A. Balasundram, T. Samarasinghe, and D. Dias, "Performance analysis of wi-fi direct for vehicular ad-hoc networks," in *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. IEEE, 2016, pp. 1–6.

[18] W. Xu, W. Shi, F. Lyu, H. Zhou, N. Cheng, and X. Shen, "Throughput analysis of vehicular internet access via roadside wifi hotspot," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, 2019.

[19] "TruDaMul," April 2022. [Online]. Available: https://github.com/AnaNSi-research/TruDaMul