Incentivized Data Mules Based on State-Channels

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, i.e. "smart territories". With the aim to offer a practical and secure way to transport data in such constrained scenarios, we focus on the problem of incentivizing to Data Mules, i.e. devices dedicated to enable communication even in the absence of the Internet. We combine decentralized technologies and State-Channels to verify the correct behavior of participants in an offline scenario.**

***Index Terms*—Data Mules, State Channels, Smart Territories**

I. Introduction and Background

Events linked to the COVID-19 pandemic have shown that more and more people decides to move towards countrysides and rural areas, i.e. to the smart territories [1]. For some underprivileged smart territories cases it is not possible to implement (costly) smart city services due to not being supported by a wide area network connectivity, and certain networking solutions might result to be too costly (e.g. satellite connections). 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 [1]. Data Mules (acronym for Mobile Ubiquitous LAN Extensions), for instance, are some types of devices that, even in the absence of Internet, are able to collect data from sensors and to exploit their own mobility to carry the information to destination by using a wireless short-range communication medium, and effectively create a data communication link [2]. In this paper we present InDaMul, 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. DLTs are cryptographically guaranteed to be tamper-proof and unforgeable, enabling the creation of a "trusted" mechanisms. On top of that, Smart Contracts are instructions stored in the ledger and automatically triggered once a required condition is met. In this work, we will refer to the Ethereum Smart Contracts implementation and to second layer cryptocurrencies, i.e. ERC-20 tokens [3].

We make use of 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, i.e. State Channels [3]. A state channel is opened between a sender and a receiver, through some tokens deposit in a Smart Contract, and then both actors can interact to adjust the channel token balance through off-chain, i.e. outside of the ledger, messages from time to time. When more channels are opened involving the same actors, then this consists of an establishment of state channel networks, where the participants pay by using other participants as relays [3]. Different sets of Clients and their physical Neighbors can build up a state channel network supporting the creation of "Islands" served by Data Mules.

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

II. InDaMul Application

We provide a summarized description of the protocol with Figure 1 describing an implementation of the InDaMul dApp.

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.

a) Client to Mule: Whenever C is willing to send a message to a Server S, it waits for a Mule that picks up its 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. When a Mule $M1$ accepts to carry the message, then C transmits to $M1$ the payload p_C, a $balance_{M1}$ and a $tender_C$, signed by C. The $balance$ is a state channel object used for the balance update between C and $M1$. The $tender$ in an object containing: (i) URI_p - an immutable URI, e.g. a hash pointer, that identifies a payload in a DFS; (ii) *offer* - a numerical value representing C's offer to P; (iii) id_x - the id of the key used for obtaining p_C; Next, $M1$ will take care of delivering payload p_C to a Proxy P.

b) Mule to Proxies: When $M1$ becomes online it can directly publish the $tender$ 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 Smart Contract named *InDaMul* and owned by C executes the majority of the protocol tasks. A Proxy P, which decides to take charge of p_C, simply invokes a method in *InDaMul* using the $tender$ object as input. This *submitTender* method automatically checks the validity of the signatures found in the data provided by $M1$, i.e. the $tender$, and then binds P's address with id_x. This makes P eligible to get access to the key identified by id_x. Then, 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

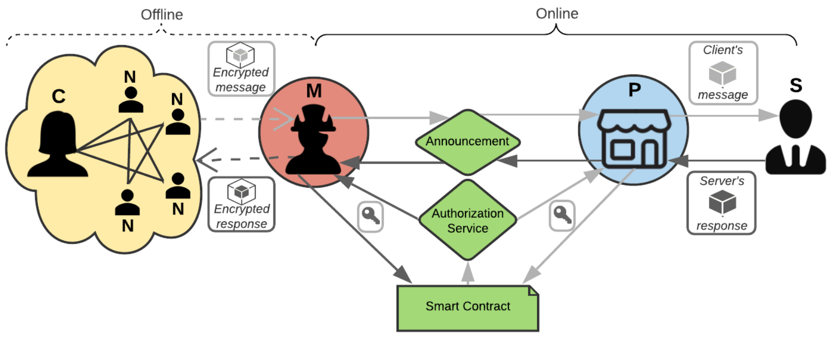

Fig. 1. Graphical representation of the InDaMul application.

x using the Secret Sharing technique [4]. Each authorization node autonomously releases a share of x to P and this one aggregates such shares to decrypt the payload p_C, previously obtained from the DFS. The obtained message is sent to S.

c) On-chain and Off-chain Payments: When C assigns the carriage of a new message to $M1$, it also issues a $balance_{M1}$ object to update 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*'s *submitTender* method. 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 This amount is locked until a response reaches C through another Mule, e.g. $M2$. In fact, whenever P has to relay a response from S to C, it publishes directly in the announcement service through a $tender_P$, in a process similar to the one described above but inverse. A Mule, $M2$, that wants to take charge of $tender_P$ will reach C in order to deliver the response payload. Finally, $M2$ gets paid using a $balance_{M2}$ object and response for the challenge–response authentication that, when uploaded to the *InDaMule* through the *submitPayment* method, will unlock both $M2$'s and P's payments.

d) 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 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. multiple Neighbors, i.e. a pending payment in each channel

In order to incentivize Neighbors to relay messages, a State Channel Network is used (we refer to the Raiden protocol [3]). The Neighbors in the Island will share the information regarding their availability status and current opened state channels capacities and fees for their relay. The messages can then be exchanged using different dissemination strategies [5]. A Neighbor that decides to send a message to a Server S becomes a Client C and seeks to reach a Data Mule M through a TN and through (zero or) some mediators, i.e. Neighbors N_i. C propagates a *lockedTransfer* message to TN through between C/N_i, N_i/N_j and N_j/TN, where all N_i are nodes in the (already known) channels path between C and TN. Then, TN requests a secret from C by sending a *secretRequest*. C checks its validity and replies with a *revealSecret*. This message contains a secret that allows each N along the path, and finally TN, to claim the locked amount in the pending transfer, after that a cascade of *revealSecret* is exchanged between each N_i along the path. The transfer is finished when C receives a *revealSecret* from the first N_i in the path.

e) Smart Contracts Performances: 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 transaction cost for invoking $submitTender$ in the Ethereum public blockchain is ~ 43.62 dollars, however, in a sidechain such as Polygon it would cost around ~ 0.005 dollars, making it viable for deploying the application.

III. Future Works

In future works we will focus on the Mules mobility since other delays, by comparison, are negligible. We consider a smart-village-like scenario where buses and couriers act as Data Mule. The simulations, performed with the LUNES agent-based simulator [5], will be aimed to show the potential delays depending on the presence of islands and mule couriers.

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*, ser. ISCC '16. Washington, DC, USA: IEEE, 2016.

[2] 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.

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

[4] M. Zichichi, S. Ferretti, and G. D'Angelo, "A framework based on distributed ledger technologies for data management and services in intelligent transportation systems," *IEEE Access*, 2020.

[5] 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.