WIRELESS SERVICE ACCOUNTING BASED ON ETHEREUM STATE CHANNELS - MASTER THESIS

A. EGIO, L. GONZÁLEZ, A. PARDO, T. ROMERO

ABSTRACT. This document contains authors' "*Blockchain technologies master*"[1] final thesis. The project described in the document consists on a technological proof of concept of an internet access service consumption using a state channel smart contract as mean of payment. After a brief introduction, there's a first motivation of the use case included; in the following sections there's a detailed explanation of the architecture, implementation and security considerations; after that it is described a more general approach about using blockchain technologies and state channels in particular to provide mechanisms to reduce advanced wireless (i. e. 5G) services market friction between neutral host providers, virtual operators and operators purchasing licenses of spectrum bands to nations' governments; guarantee transparency to consumers, and provide a fully GDPR compliant schema relying on a self-sovereign identity system.

CONTENTS

DISCLAMER

All software described and linked in this document is open source under the MIT license; so anyone can freely use it and make modifications to the source code to fulfill their use cases. But if you do so, please take into account that the software at the time of this writing is merely a proof of concept and is not properly audited; although authors tried to follow best practices regarding smart contract, front-end and back-end design and implementation there are still some tradeoffs and recognizable security issues described in this document's security considerations section. Please take this into account when using and/or modifying this software and be conscious that authors do not provide any guarantee in case of security issues, economical loses or regulation infringement.

1. INTRODUCTION

As exposed in [1] although radio spectrum "*is often referred to as a 'scarce resource'*" there are spectrum management strategies being researched to improve and support a brand new model of services into the market; in particular there are multiple 5G[2] innovative projects and initiatives that try to solve this challenges using Software Defined Network[3] and Network Function Virtualization[4].

Since spectrum is a scarce resource, it can be tokenized being suited this way for blockchain/DLT technologies business models; in addition, since atmospheric spectrum can be considered a public asset, it's initially managed by nations' governments and its licensing and commercial usage should feature a high level of transparency and at the same time allow for efficiency and fair-competition making it feasible to model on public blockchain systems.

Taking this into account, lead the authors' to develop a first bottom-up proof of concept of a simple system demonstrating how can a wireless WiFi hotspot deployed on a commodity hardware asset[31] be managed to allow users to access the internet paying in exchange for Ethereum on a public network.

One of the first intiatives to create a bandwidth reselling market was La Fonera[5] back in 2006. Main goal of the initiative was to allow ADSL users to share unused bandwidth using a specific WiFi hotspot deployed on an ad-hoc built router.

Figure 1. *La Fonera's initial project router.*

A more recent project like Hotspot Me[6] and also very aligned with the technological demonstration described in this document consist on using mobile APIs to allow users to share WiFi hotspot tethering to users nearby in exchange for ETH supported by a micro-payment channel contract.

2. TECHNOLOGICAL BACKGROUND

Blockchain technologies support a ledger of records in continuous growth called blocks. They are linked and secured by cryptography. Adopt the P2P protocol (*peer-to-peer*) in order to be distributed with not a single point of failure.

The consensus mechanism ensures a common order of unambiguous transactions and blocks, and guarantees integrity and blockchain consistency a through geographically distributed nodes. By its design, the blockchain has characteristics such as: decentralization, integrity and auditability. The blockchain can serve as a new type of software connector, which should be considered as a possible decentralized alternative to storage of existing centralized shared data.

In addition, depending on the different levels of access permission, block chains can split into two types: 1) public (such as Bitcoin and Ethereum); and 2) private (such as Hyperledger). Blockchain serves as a platform for smart contracts (hereinafter, *smart contracts*). For example, programmed in the Solidity language of Ethereum, stay and run. Blockchain is a technology concept DLT distributed ledger (*distributed ledger technology*). It can be integrated into multiple business areas.

Ethereum blockchain is proposed as a first optimal candidate platform to support this project PoC attending to following reasons:

- Is the most used and mature turing-complete-programmable blockchain technology.
- Support for high quality and audited libraries in Solidity (most used smart contract language in Ethereum).

- Developments on Ethereum can be deployed over public (main or testnets) or private/permissioned networks.
- Unlike Bitcoin, Ethereum supports miner fees using gas instead of directly ETH; this can mitigate the effects of volatile crypto-currency price by adjusting gas price depending on ETH value.

3. Proof of Concept description

3.1. **Architecture.** In order to support the PoC a Raspberry Pi 3 Model B (see figure 2) has been used as a single board computer to host all the required software. Tested operative system was OSMC[10] (a Debian based GNU-Linux flavor) mainly used as media-center. A WiFi access point using hostapd[7] daemon over Raspberry's WiFi chipset. Raspberry Pi was also connected using ethernet cable to an optical fiber commercial router (*see figure 2*).

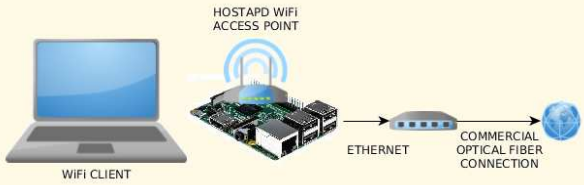

Figure 2. Physical Setup.

Specs	Raspberry Pi 3 Model B
CPU	Broadcom BCM2837, Cortex-A53 (ARMv8) 64-bit SoC @ 1,2 GHz
RAM	1 GB
Connectivity	Wi-Fi 802.11 b/g/n (2,4 GHz) Bluetooth 4.1 Ethernet card up to 100 Mbps
Ports	Full HDMI, 4 USB 2.0, MicroSD, CSI camera, DSI display
Memory	MicroSD

Table 1. Raspberry Pi 3 Model B specs Although it is not expressly indicated if it is free hardware (*open hardware*) or with trademark rights, on the official website there's an explanation regarding that Raspberry have distribution and sales contracts with two companies and guarantee that anyone can become a reseller or redistributor of Raspberry Pi cards [RaspberryPiBuy 19].

Dnsmasq[9] was used as dynamic host configuration server for the wireless network users and iptables[8] was used as a firewall running both inside the Raspberry; by default, iptables was configured to drop all packet forwarding traffic from users connected to the wireless domain and at the same time to redirect all traffic to Raspberry's HTTP port where a captive portal was listening incoming connections.

3.2. **Main connection workflow.** Main idea is that the users can join freely to the wireless hotspot and access the web captive portal (*hosted on the same Raspberry*) application that communicates with a back-end that controls iptables rules and also an Infura light proxy able to relay RPC calls arriving on 8545 port of the Raspberry.

To keep users' freedom to interrupt the service consumption whenever they want and also avoid the overhead of transacting many times with the smart contract; authors have implemented what is known as a state channel in a smart contract; this makes possible to keep control of the time fraction of the service that's already consumed and at the same time cryptographically guarantee that the smart contract is going to unlock the funds when the provider desires to do so. State channels are a standard pattern in most blockchain applications and in all cases (*independently of the complexity*) present three main steps: opening, transaction and closing; project's particular implementation described below:

3.2.1. *Channel opening.* The connection flow as shown in figure 3, can be summarized as follows:

1) The user associates the laptop with the WiFi hotspot.
2) Once connected is redirected to the captive portal front-end.
3) Front provisions MetaMask as Web3 provider.

4) Front-end uses MetaMask to allow the user to send value to the contract
5-6) Signed transaction is relayed through a light Infura proxy listening on port 8545.
7) Once transaction is confirmed, the back-end iptables controller start forwarding user's traffic.

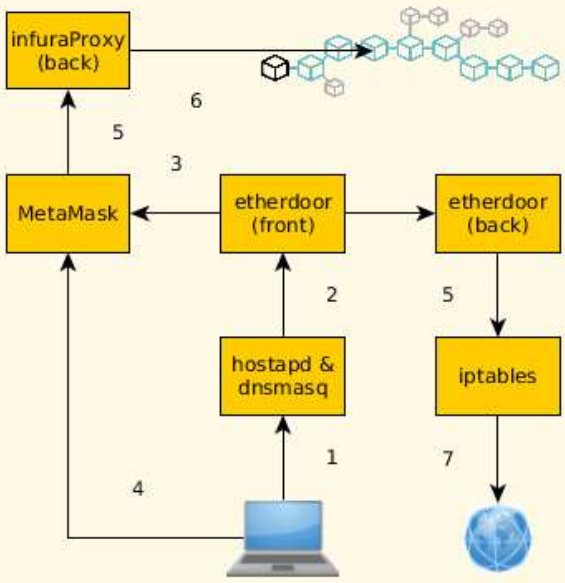

Figure 3. Connection work-flow.

Figure 4. Raspberry Pi 3 Model B.

3.2.2. *Channel transactions.* Before sending the initial value transaction (*proportional to the initially expected duration of the service*) user's front-end generates an ephemeral elliptic curve secp256k1 keypair and sends the public key to the contract to open the state channel. Corresponding ephemeral private key is the one that's going to be used to periodically (each 10 s approximately) produce increasing value signatures proportional to the service consumption ellapsed time. This signatures are being send to the backend instead of the blockchain; backend verifies and validates the signatures and stores the last one (*the most valuable*).

3.2.3. *Channel closing.* At the end, when the connection time is exhausted or the backend stops receiving channel transactions for approximately 60 seconds, the backend sends the last seen transaction to the smart contract using the method `closeChannel`; this method, when called with a valid channel signature as parameter, unlocks user's original funds and transfer the signed amount to the contract owner (*provider of the service*) and the corresponding return to the user.

3.3. **Detailed software description.** This section includes a detailed rationale software description of all the components and subcomponents.

3.3.1. *Smart contract.* The smart contract[11] used to implement the proof of concept consists on a simple uni-directional micropayment channel that is Ownable depending on OpenZeppelin contract; uses also OpenZeppelin's SafeMath library in order to perform accounting operations and finally uses also OpenZeppelin's ECDSA library to support the signature validation on the closing channel operation.

Main data stucture of the contract described below:

`ChannelData`. Includes payer ethereum address, its state channel ephemeral address, deposit amount, opening time and a boolean representing the open/close status of the channel.

Contract's storage variables:

`channelCount`. uint256 variable that increments in a unit every time a user opens a channel. This operation does not rely on SafeMath since it's virtually impossible to overflow a uint256 storage record.

`channelMapping`. Holding a key-value map between channelCount values and channelData structures.

`pricePerSecond`. Constant keeping track of service price per second.

The main methods of the smart contract:

`openChannel`. receives as parameters, the ephemeral address of the user and also the purchased service value.

`closeChannel`. accepts the amount of ETH to refund to sender, the `channelId` and the `signature` of the payer; checks the signature and after updating the state, transfers the amount to the owner of the contract and the return to the channel opener.

`claimTimeout`. method that allows the payer to close the channel in case contract owner does not close it after some `expirationTime`.

3.3.2. *Etherwall.* Consists on a NodeJS backend that fulfils three main system functions:

- Controls the state of the current connections.
- iptables interfacing.
- Micro-payment signature and amount validations.

On execution uses the module `lib/iptables.js` to setup and initialize system's iptables redirecting all 80 port traffic to the ip address to Raspberry's ip address; to do so, relyes on the node module `child_process` to spawn system commands. Also redirects all traffic trying to reach 8545 port to itself the same way in order to redirect MetaMask's RCP calls to the lightweight infuraProxy[13] developed also for this project and contained in etherwall's root directory.

After initializing, the backend using the web framework[17] etherwall exposes the following endpoints:

`GET /generate_204`. As a way to enable captive portal detection to Android OS. HTTP Status 204 means the file exists but, is empty. This lets Android know that the Internet is accessible. If the request receives HTTP Status 302 (temporary redirect) instead of HTTP 204, Android will follow the redirection URL to display the Captive Portal to the user.

`POST /mac`. Endpoint to request a new connection; mainly accepts the time left to use the service as a parameter and internally gets the MAC address of user's device making an arp table query based on client's ip address. After that invokes `lib/iptables.js` `grantAccess` function in order to insert a rule to enable packet forwarding for this particular MAC address on top of iptables. Also uses module `lib/connections.js` to keep persistence of the connection status. Once connected after posting `timeLeft` to this endpoint, in module `lib/connections.js` there's a `setInterval` triggering getting connection status and time since last micro-payment each 5 seconds. In case `timeSinceLastPayment` exceeds 60 seconds or `timeLeft` goes negative, disconnects the client deleting the forwarding rule from iptables.

`GET /mac`. Retrieves the status of client's connection from `lib/connections.js` module; this module in addition, provides some serialized persistence using a JSON file under the `data/` directory to keep state integrity between server restarts.

`POST /payment`. Endpoint that allows the front-end to send a payload containing a payload passing the corresponding contract data:

- `channelId`
- `amount` (to unlock)
- `signature`

In order to verify the correctness of the payload, uses Web3 module to compute the `solidtySha3` and compares it to the signature's `messageHash`; after that recovers the public address from the signature and compares it to the associated smart contract mapping structure based on its `channelId`; if everything matches, it additionally checks that the unlocked amount is approximately what it corresponds taking into account connection ellapsed time (*using a kind threshold of 60 seconds marginal cost*); in case everything is correct, stores the payment together with a timestamp as the last one corresponding to the connection using again `lib/connections.js` module. This is done to allow bootstrapped `setInterval` on the previously described `POST mac/` to process the timeout business logic.

3.3.3. *etherdoor.* A very simple front-end based on React using material styling; App's main `componentDidMount` initializes the state including Web3 object and smart contract wrapper, gets current connection status from backend's `GET mac` endpoint and creates the micro-payment state channel ephemeral account; at the end triggers a `setInterval` loop that when connection activates, will be responsible of sending proper signed micro-payments.

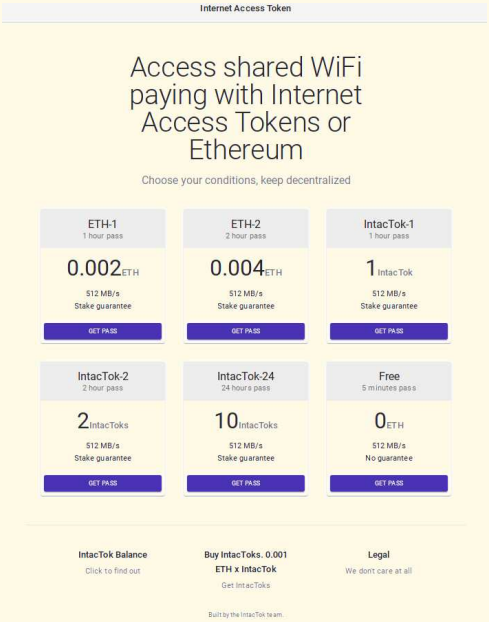

Figure . *etherdoor front-end landing page*

Usage depends on some button clicking, triggering `someMinutesPass` function invokation that launches MetaMask as Web3 provider making possible to the user to open a channel on the smart contract building a transaction and relaying it through `infuraProxy`; it also invokes `startChannel` that will update React's App state and will cause `signMicroPayment` function to periodically on a 10 seconds interval to sign and post to the backend the increasing amount micropayments.

4. Quality considerations

To validate the PoC, a battery of tests has been defined. For example, given that smart contract normally handle money, it is essential to ensure that its number of failures and vulnerabilities is low [Hegedus 18].

4.1. **State channel contract tests.** During development and in order to ease the automatic deployment of the contract over a Truffle's suite Ganache-CLI**??** instance there are two util scripts on the repository[11] `utils/compile.js` and `utils/deploy.js` that take care of merging all the Solidity source code required and deploying it to the Ganache instance.

In repository[11] module `tests/main.test.js` there are the following Mocha framework[18] based tests; tests use Chai assertion library[19] using its *should* mode whenever possible but some of the assertions still have to be done by `try-catch` pattern due to Chai not supporting at the time of writing `BigInt` operation.

To help developers and make technology more mature, we need analysis tools [ConsenSys 19].

For this project, we used Mythril from ConsenSys. Mythril is a security analysis tool for EVM (Ethereum Virtual Machine) bytecode. It detects security vulnerabilities in smart contracts built for Ethereum.

We analysed the Solidity-based smart contract called StateChannel.sol (of this project).

The installation steps are as follows in:

https://github.com/ethereum-internet-access/mythril Firstly, we analysed the local source code:

```
$ cd contracts
$ myth analyze StateChannel.sol \
  > ./security-report
$ cat ./security-report
```

Secondly, we analysed on-chain smart contract. Because the smart contract was deployed on Ropsten testnet we executed:

```
$ myth analyze --rpc \
  infura-ropsten -a \
  contract_address \
  > ./security-report
$ cat ./security-report
```

5. Deployment

Docker is a tool that allows you to deploy applications inside of software containers. This can be useful for our Raspberry Pi because it allows users to run applications with very little overhead, as long as the application is packaged inside of a Docker image.

We simply install Docker and run the container over Raspbian/ARM. It would deploy the done project. The script to execute is as follows:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install curl
$ curl -sSL https://get.docker.com | sh
$ sudo apt-get install git
$ git clone \
  https://github.com/ \
  ethereum-internet-access/docker.git
$ cd docker
$ sh ./install-project\
  -inside-docker-container.sh
```

6. Further steps

6.1. **5G Technologies and scarcity model.**

In the recent years blockchain technologies and in particular smart contracts have made possible an innovative way to rule business and processes between organizations.

One of the main application of smart contracts consist on modeling a scarce resource representing it using fungible (or not) tokens; in addition to this, smart contracts can be programmed to suit any possible business intelligence according to rules well known and accepted in advance by the parties transacting.

In the case of 5G technologies for practical reasons and the high density of antennas required, different operators shall be able to dynamically share the same resources (i.e. neutral host infrastructure). At the same time in order to manage service operation efficiently through its whole life-cycle there's the need to automate as much as possible all the operations; this requirements perfectly fit smart contract capabilities, in order to model and operate over a defined set of rules supporting payments, escrows, rewards or penalties depending on different conditions.

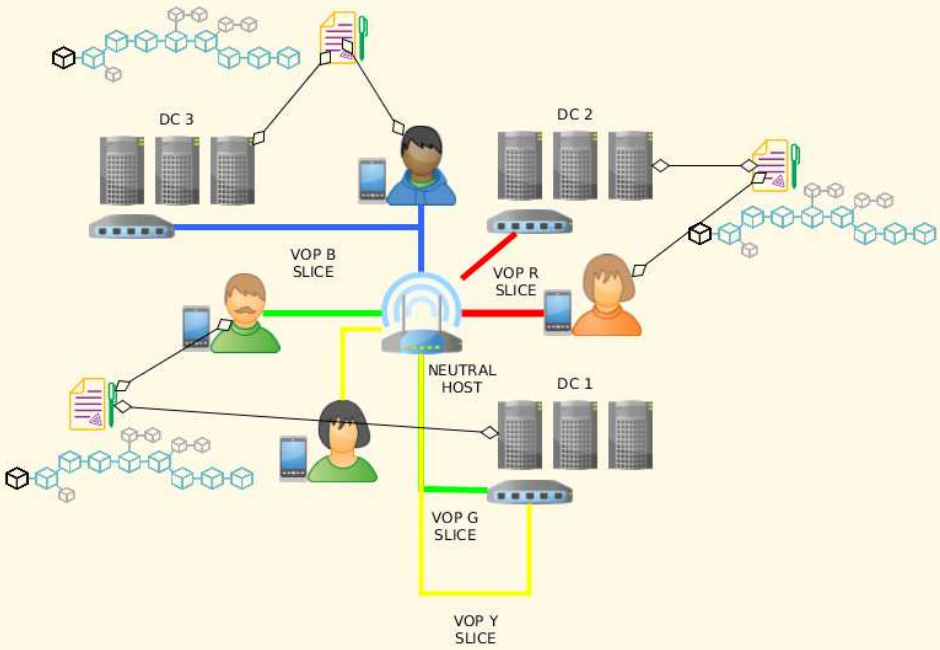

Figure 5. Slicing model.

As an example, in figure 5 it's shown how three virtual operators over different infrastructures shared or not in a slice-model of a neutral host infrastructure; a slice can be described as a dynamic virtualization allowing multitenancy of some physical resources like access points, physical switches and datacenters including mobile edge computing appliances; this is the case of currently ongoing projects like 5GCity[20] where a distributed cloud and radio platform for municipalities and infrastructure owners acting as 5G neutral hosts is being developed and tested. Platforms targeting 5G business models such as 5GCity can benefit from adding this smart contract business logic in order to bill end-users and automate all the accounting layer.

6.2. **Know your customer issue.** Although this is a promising approach; there's an issue with it regarding regulation, in particular in the European Union, there's the requirement that a telecommunication service provider must know their customers; this is required for example to allow judge requirements of users identity in case there's some evidence of illegal activities. As a single payment system, state channel smart contracts are pseudoanymous hence do not support the *Know Your Customer - KyC* required by regulation.

To take into account the problem illustrated in previous paragraph we can take as a basis a model of self-sovereign identity[21] and in particular the one recently anounced by the authorities of Catalonian government[22] that will provide credentials to citizens using DLT technologies as a Public Key Infrastructure; this model could allow in some near future to give the citizen the option of registering identity in such a way that the credential can be linked to some administration database that could be queried only by a judge requirement and guarantee the users identities are recognized

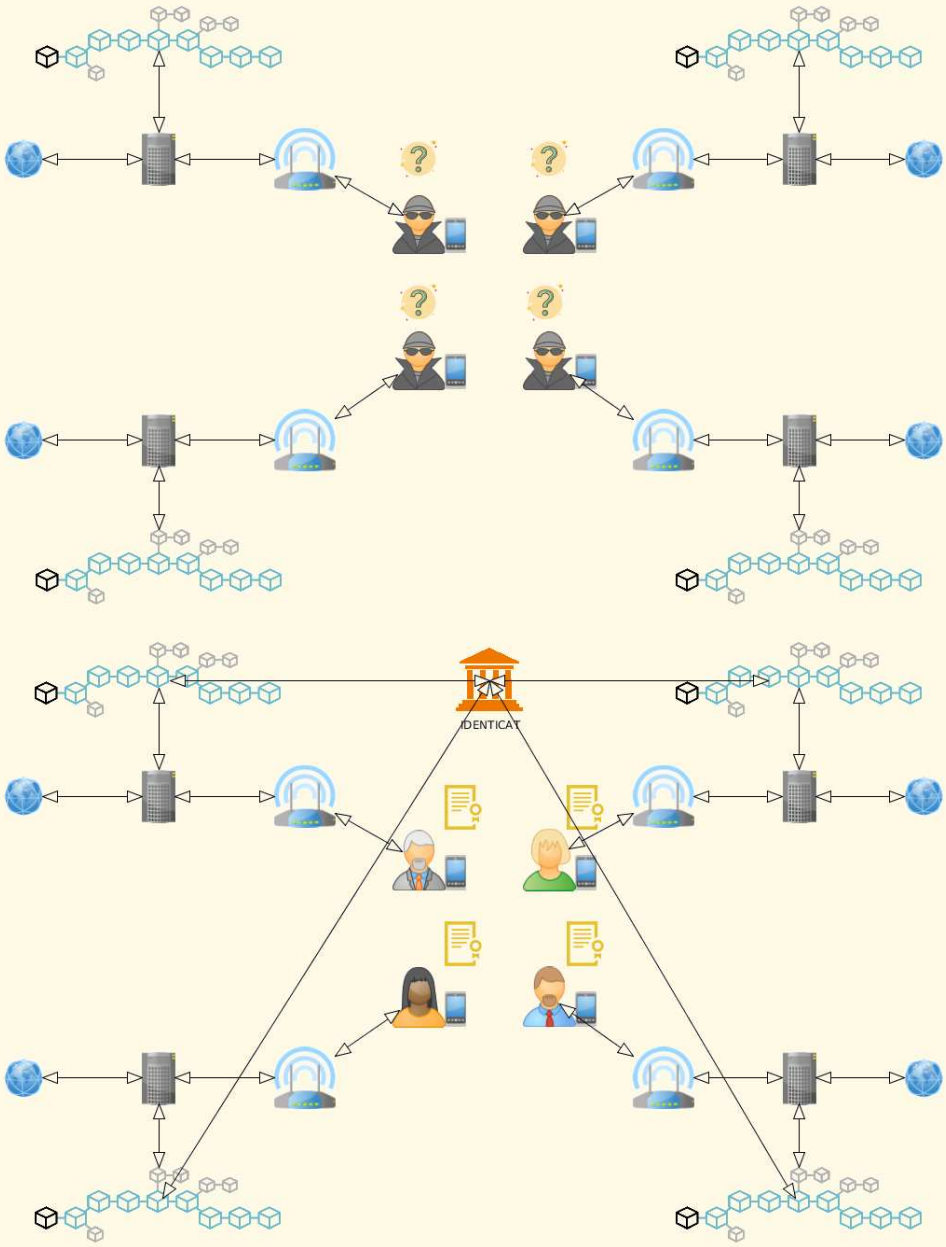

Figure 6. Upper figure shows a scenario where different users connect to antennas controlled by a state channel smart contract like the one described in this thesis; users remain pseudoanonymous and hence, the schema is not regulation compliant regarding the KyC that a telecommunications operator should guarantee in case of legal issues. On the other hand, relying on a self-sovereign decentralized identity bottom figure shows up how the citizens can authenticate themselves to the system with the guarantee there's a public administration validating their identities.

by a public administration; this way, the service provider (*in our case a telecommunications operator*) does not need to know anything about a customer appart from a public key and a valid credential signed by the administration. This schema could be a real game changer regarding operators competitivity since they get rid-off from several and costly processes that do not add any value to the service they provide:

- Fully automated accounting and billing layer.

- Regulation compliance without KyC on-boarding process.
- GDPR compliance by default. Operator does not need to know user's name, bank account, etc.

6.3. **Instant portability.** This approach to the way customers interact with telecommunication operators relying on a publicly recognized self-sovereign identity can support a new way of handling portability granting full customer empowerment when they want to switch from one operator to another; in figure 5, is represented a process on a user opens a channel in operator's A smart-contract; after that begins transacting with this first operator paying as they consume the access service sending micro-payments to the operator.

After that, user decides to switch to a second operator, opening a channel over operator B's contract; since operator B, stops receiving the signed micropayments, opts for closing the channel backdrawing the earnings in operator's account and return value for the user. Such kind of process can be accomplisehd in a matter of seconds for a few cents fee literally obliterating any operator lock-in mechanism thus supporting a really zero friction service market.

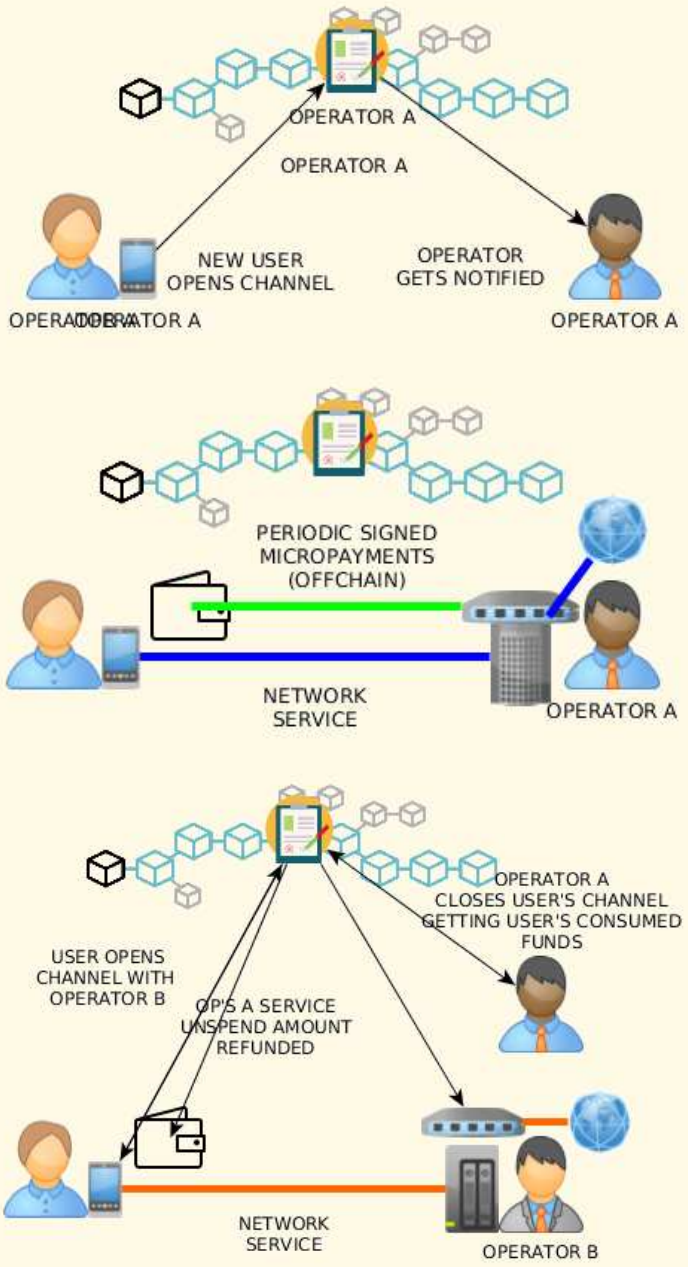

Figure 5. State channels support instant portability between different operators with minimal fee.

6.4. **Towards telco market full tokenization.** This model can be extended without limits towards a full wireless telecommunication market tokenization; as depicted in figure 6, from the spectrum auctioning, including infrastructure providers virtualizing their appliances for virtual operators, end to users service provision and accounting relying on a self-sovereign identity system.

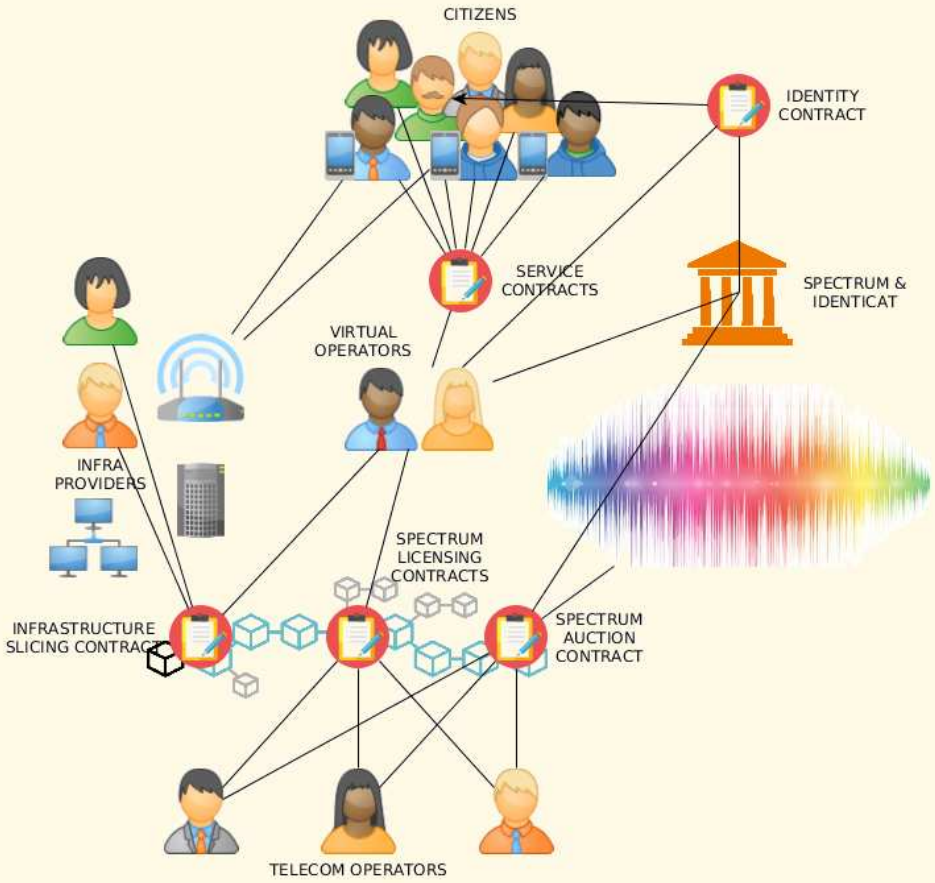

Figure 6. Full tokenization of wireless services from the spectrum licensing to the citizen.

6.4.1. *Smart contract.*

6.4.2. *Front-end.*

6.4.3. *Back-end.*

6.5. **Regulation considerations.** TODO; talk about GDPR, telco KyC, european market regulation, etc.

7. Conclusions

World we live today is interconnected and wireless internet access has a foundamental role. In this project a Proof of Concept has been build. This PoC based in commodity free hardware asset (Raspberry Pi) can serve as an example of end-user wireless provision pay-as-you-go using a state channel Solidity smart contract; KyC issues and identity consideration have been discussed also leading to the conclusion that complementing this approach with what a self-sovereign identity can provide there's no need to a telecommunication operator to perform a KyC process in order to provision a service to a client.

One of the major problems with this approach has to do with ETH-fiat exchange volatility (*see figure 7*), anyway, including a feature to regulate minute price in the state channel contract and taking into account gas prices can adjust in order to properly represent computation, bandwidth and storage costs this problem can be completely avoided.

Figure 7. ETH Price chart (11/08/2019). Source: [Coinbase 19]

7.1. **Usar Docker Swarm.** Se podría presentar una aplicación descentralizada basada en microservicios. Por ejemplo en microservicios, un contenedor Docker (como en la sección 4) podría verse como un servicio. Entre las ventajas que aporta microservicios estarían:

- Pequeños
- Independientes
- Despliegue sencillo
- Reutilizables
- Externalización
- Escalabilidad

Para ello, Docker ofrece un orquestador (de código abierto) llamado Docker Swarm (véase figura 4). Un nodo *worker* sería nuestra Raspberry Pi y un nodo *manager* sería un centro de control.

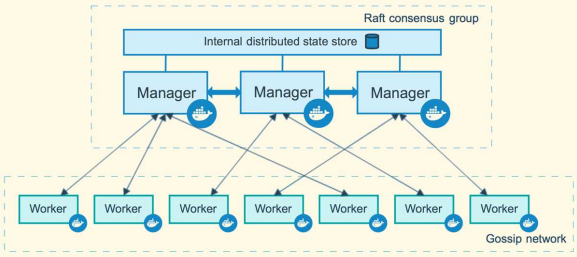

Figura 4. Diagrama Docker Swarm. Fuente: [Docker 19]

8. Repositorio

El código fuente generado en el proyecto está publicado en un repositorio público de GitHub bajo licencia MIT.

`https://github.com/ethereum-internet-access`

References

[1] 5G Spectrum and Neutral Hosting, Discussion paper, February 2019, DCMS Phase 1 5G Testbeds & Trials Programme
[2] https://en.wikipedia.org/wiki/5G
[3] Ian F. Akyildiz and Shih-Chun Lin and Pu Wang, *Wireless software-defined networks (W-SDNs) and network function virtualization (NFV) for 5G cellular systems: An overview and qualitative evaluation*, Computer Networks, 93, 2015
[4] Jose Ordonez-Lucena et al. *Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges*, IEEE Communications Magazine (Volume: 55 , Issue: 5 , May 2017)
[5] https://es.wikipedia.org/wiki/La_Fonera
[6] https://devpost.com/software/hotspot-me
[7] https://w1.fi/hostapd/
[8] https://linux.die.net/man/8/iptables
[9] https://en.wikipedia.org/wiki/Dnsmasq
[10] https://osmc.tv/
[11] state-channel-contract-a on GitHub
[12] etherwall on GitHub
[13] infuraProxy on GitHub
[14] etherdoor on GitHub
[15] https://expressjs.com/
[16] https://github.com/trufflesuite/ganache-cli
[17] https://expressjs.com/
[18] https://mochajs.org/
[19] https://chaijs.org/
[20] https://www.5gcity.eu/
[21] C. Allen, The path to self sovereign identity, 2016
[22] Catalonia to Build DLT Identity, Coindesk, 2019
[23] Coinbase. Intercambio de moneda digital https://www.coinbase.com/price/ethereum
[24] Ethereum Smart Contract Best Practices. Security tools. ConsenSys. 12 Agosto 2019.
[25] Docker Documentation. How nodes work. 6 Septiembre 2019. Docker Swarm
[26] Hackathon ETH Berlín Zwei. Twitter. 6 Septiembre 2019. https://twitter.com/ETHBerlin
[27] P. Hegedus. "Towards Analyzing the Complexity Landscape of Solidity Based Ethereum Smart Contracts". 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). 27 Mayo-3 Junio 2018. https://ieeexplore.ieee.org/document/8445056
[28] MakerDAO. Stability for the blockchain. https://makerdao.com/en/dai
[29] J. L. Muñoz. "Docker Containers". Apuntes de las clases magistrales. Máster en Tecnologías Blockchain. 1, Edición. UPC School. 29 Noviembre 2018.
[30] S. Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". 31 Octubre 2008. https://bitcoin.org/bitcoin.pdf
[31] Raspberry Pi 3 Modelo B - Placa Base (1,2 GHz Quad-Core Arm Cortex-A53, 1 GB RAM, USB 2.0). Amazon. 11 Agosto 2019.
[32] Raspberry Pi FAQs. BUYING AND SHIPPING. Where can I buy a Raspberry Pi?. Página Web oficial. 11 Agosto 2019. https://www.raspberrypi.org/help/faqs/#buyingWhere

[33] W. Stallings. “Comunicaciones y redes de computadores”. 1 Septiembre 2004. Páginas 904. Pearson Prentice Hall. ISBN-13: 978-8420541105.