Spass: Spectrum Sensing as a Service via Smart Contracts

Suzan Bayhan, Anatolij Zubow, and Adam Wolisz
Technische Universität Berlin, Germany. Email: {bayhan, zubow, wolisz}@tkn.tu-berlin.de

***Abstract*—Mobile network operators can expand their capacity by aggregating their licensed spectrum with the spectrum discovered opportunistically, i.e., spatiotemporally unused spectrum by other primary users. For an accurate identification of the spectral opportunities, the mobile network has to deploy multiple sensors or it can offload this task to nearby nodes with sensing capabilities, so called *helpers*. Unfortunately, incentives are limited for helpers to perform energy-wasteful spectrum sensing. Instead, we envision *spectrum sensing as a service* (Spass)[1] in which a smart contract running on a blockchain (BC) describes the required sensing service parameters and the contracted helpers receive payments only if they perform sensing accurately as agreed in the contract. In this paper, we first introduce Spass and derive a closed formula defining the profitability of a Spass-based business as a function of the spectral efficiency, cost of helpers, and cost of the service. Moreover, we propose *two-threshold based voting* (TTBV) algorithm to ensure that the fraudulent helpers are excluded from Spass. Via numerical analysis, we show that TTBV causes almost zero false alarms and can exclude malicious users from the contract after only a few iterations. Finally, we develop a running prototype of Spass on Ethereum BC and share the related source code on a publicly-available repository.**

I. Introduction

With the increasing demand for bandwidth-hungry applications, the mobile network operators (MNOs) are under pressure to meet efficiently demands for more capacity and better quality-of-service (QoS). The continuous increase of spectral efficiency is not enough — there is a clearly articulated demand for more spectrum, e.g. in 5G [1]. Rather than using only their licensed bands, operators can utilize also unlicensed bands, e.g., as in LTE-unlicensed [2], and develop new business models such as bundling in [3]. As unlicensed spectrum has become also congested, another option known as *dynamic spectrum access* (DSA) enables a *secondary user* (SU) network[2] increase its capacity by using the spectrum holes in the licensed bands which correspond to the inactivity times of the licensed transmitters of the band, i.e., primary user (PU). However, the SU network has to ensure that it can detect a reappearing PU traffic timely and with high accuracy after which the SU network aborts its transmission in the PU channel. To satisfy this requirement, as previous research has experimentally [4] and theoretically [5] shown, the SU network has to deploy multiple spectrum sensors rather than single nodes sensing the spectrum. Instead of deploying the sensors itself, the SU network can offload the sensing task to nearby sensor nodes, called *helpers*, whose spectrum observations can reflect accurately the spectrum state at the SU's neighborhood. Helpers can be any node ranging from low-end sensors or mobile phones to more advanced spectrum scanners, owned by individuals or by companies.

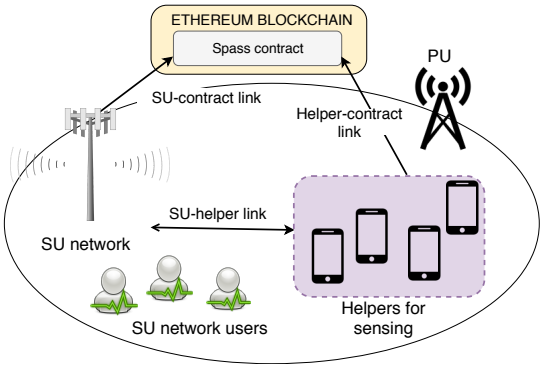


Fig. 1. Spass system model: SU network is interested in accessing the primary user's band opportunistically to serve its users. Helpers are the nodes offering sensing service. The agreement and transactions are processed through the smart contract defined in Ethereum blockchain.

However, helper nodes, especially those with strict battery-resources, are difficult to incentivize to perform spectrum sensing for others. While the previous studies [6] explored incentives as a result of social ties among the users of the wireless devices, such social-tie based models are applicable only to entities with trust relationship. Bringing cooperation to everybody is crucial to unlock the power of the crowds. To this end, blockchain (BC) paradigm is an enabler for trade without either a trusted third party or a priori thrust relation among the involved trading entities.

Particularly, *smart contracts* running on the BC [7] describe the terms of a contract, e.g., requirements on the sensing service and conditions for payment to the helpers. The idea of using smart contracts for DSA has appeared in some recent work [8]. However, none of these works has addressed the following key questions: *(i) what is the cost of running smart-contract based spectrum discovery? and (ii) given that both the helpers and the BC miners have to be paid, under which conditions an SU network operator can sustain a profitable business via smart-contract based spectrum discovery?* Our goal in this paper is to address these two questions. We can list our key contributions in this paper as follows:

- We present *spectrum sensing as a service* (Spass) as a practical way of spectrum sensing offloading via smart contracts

[1]Spass means *fun* in German.

[2]A secondary user (SU) can refer to a single node or a network. Here, we consider a network operator as an SU, hence will use the term "SU network".

which define how to execute the terms of an agreement and how to handle the related divergence from the contract terms, i.e., Service-Level-Agreements, SLAs. To the best of our knowledge, this is the first study providing a complete solution for smart-contract based spectrum discovery with a running prototype implemented in Solidity on the Ethereum BC. In contrast to the existing proposals, the task of the smart contract in Spass is not only micropayment but also acting as a trusted point where operations like deciding on the number of helpers needed to meet regulatory requirements, selection of helpers, and detection of malicious helpers can be performed.
- Considering the economic aspects, e.g., spectrum cost and cost of using Ethereum, we derive a closed formula to understand under which conditions Spass provides a profitable business to the SU network.
- We propose *two-threshold based voting* (TTBV) to identify malicious helpers in Spass. Via numerical evaluations, we show that TTBV can identify all malicious helpers after a few verification rounds and excludes them from participating in Spass contracts. Most importantly, high success in malicious helper identification does not come at the expense of frequent false alarms, i.e, honest helpers being marked as malicious and thereby denied payment, which is paramount to provide a sustainable business model where nodes are motivated to act as helpers.

II. Primer on Spectrum Sensing and Ethereum Smart Contracts

A. Spectrum Sensing in a Nutshell

In spectrum-sensing based DSA, the regulatory bodies, e.g., Ofcom in UK, assert that an SU willing to use the PU's idle spectrum opportunistically must meet certain sensing reliability criteria: PU sensing accuracy and spectrum access efficiency. The first criterion on sensing accuracy, known as *PU probability of detection* (p_d), defines the minimum probability that an active PU will be detected by the secondary network and thereby guarantees that an active PU will be identified timely so that PU communications are not drastically affected. The second criterion, known as *probability of false alarm* (p_f), is related to spectrum discovery efficiency and necessary to ensure that the spectrum resources are not wasted by a high number of false alarms. The devices due to their hardware capabilities or locations with respect to the PU transmitter differ from each other by their sensing accuracy and false alarm values.

An SU can sense spectrum locally, known as local spectrum sensing. But, prior research [4], [5] has shown that sensing with a single device may lead to incorrect conclusion of the spectrum state and collaboration in sensing with other users offers benefits to the SU, e.g., higher detection accuracy. In cooperative sensing, during spectrum sensing period, each sensing unit senses the spectrum and sends its decision, e.g., 1-bit information, to the entity which will finalize the decision on the spectrum's occupancy state. The final decision on PU activity is determined according to the *decision fusion* algorithm, e.g., Majority logic, using the local sensing outcomes delivered by the individual helpers. When the number of the cooperating users increases, the sensing accuracy is expected to improve, but at the cost of increased overhead for sensing and reporting the sensing outcomes [9]. Moreover, collaboration might harm the SU if there are malicious cooperators in the sensing group. For example, sensing result might be falsified by malicious users to block SU communications [10]. Incorrect sensing data will lead to either waste of spectrum opportunities, e.g., false alarms in case of PU inactivity during sensing, or excessive interference to and from the active PU transmissions, e.g., failure to detect an active PU.

B. Ethereum Smart Contracts in a Nutshell

Ethereum [11] is a blockchain-based decentralized computing platform which executes and validates transactions by the help of miners. As a compensation for their work, miners are paid for each transaction, e.g., writing a transaction in a block and performing other tasks such as tasks to keep transaction data safe. In Ethereum, cost of an operation is measured in *gas*, i.e., in units of "ether" (ETH) [11].

A smart contract, identified by a 160-bit unique address in Ethereum, is a computer protocol running on the blockchain to define, verify, and enforce the process of a contract [7], [12]. Each smart contract has a storage space for relevant data. Hence, smart contracts are stateful in Ethereum [7]. Ether can be send from an external account to a smart contract. To be able to make it, the contract needs to have a payable function. Moreover, a contract can send ether to an account identified by an external address.

III. System Model

We consider a system as in Fig. 1 consisting of an SU network and several spectrum sensing helpers. Below, we present our model for each entity in the considered system.

PU model: We assume a time-slotted system with timeslot duration of T. We model the traffic activity of a PU as a two-state Markov chain. Let us denote by p_0 the probability that the PU channel is idle at an observed time. In stationary state, probability of PU activity is $p_1 = (1 - p_0)$.

SU network model: The SU network is an operator interested in opportunistically accessing the PU's spectrum band to serve its customers when the PU is not transmitting. Similar to the business model in [3], the SU network might own also a licensed spectrum and aggregate its capacity with opportunistic spectrum discovered via Spass or it might use unlicensed spectrum along with its licensed spectrum as in [2]. However, we focus only on the discovered PU spectrum. To discover spectrum holes, the SU network can buy sensing service from other nodes, so called *helpers*, which are the nodes offering sensing service. Note that a channel with only very low data rate, e.g., ISM bands, is required between the helpers and the SU network. This channel is used by both the SU network to send information about its need for helpers and by the helpers to notify the SU network about PU activity. As sensing

result of the helpers need to reflect the spectrum state at the proximity of the SU network, our model considers a short-range communication technology such as WiFi between the SU and the helpers. Long-range communication technologies can also be used for informing the helpers about the Spass contract of the SU network. However, it would require a service for location estimation in the helpers, e.g., GPS. We assume that the SU network (e.g., its base station) and the helpers access the Ethereum system via light-weight remote procedure call (RPC) protocol using some other connection, e.g. wired Internet. Note that Ethereum BC is an external service to the helpers and the SU network, e.g., none of these entities run compute-intensive BC verification tasks. After receiving sensing data, SU applies a decision fusion rule, e.g., Majority.

Spass smart contract: The SU network specifies the requirements for the sensing service such as sensing rate R_s Hz or required minimum sensing accuracy for each helper.

Helper model: Helpers are nodes with sensing capability who are interested in getting monetary benefit by serving as sensing units. Such functionality can be integrated into sensing platforms like the Mobile Sensing Box [13] which are used for monitoring environmental parameters like air quality in metropolitan areas. Helper devices can be owned by a multitude of parties, including individuals or companies, e.g., micro or even nano-businesses. What matters for the SU network is each helper's sensing accuracy and associated price of sensing. Each helper who has joined the contract after being selected by the contract performs spectrum sensing with rate R_s Hz resulting in R_s bps binary sensing data.

We consider two types of helpers regarding their trustworthiness: *honest* and *malicious*. A helper is considered honest if it performs spectrum sensing and meets the accuracy requirements as agreed in the contract. But, a malicious helper, instead of performing sensing, acts as a free rider: it fabricates the sensing data and reports that the channel is busy with probability α_1. Strategy of malicious helper is to save energy by going into sleep mode and waking up from time to time to report some artificial data to the SU network and also to the smart contract. Overhearing transmissions of other helpers is possible but not promising as overhearing creates the same amount of cost for the helper as performing the actual sensing, i.e., replay attack is not promising. Another attack known as spoofing attack is also not promising as there is a significant initial cost for the helper to contact/register in the smart contract. We will discuss this case in Section VII-D. We assume that dishonest helpers act independently without cooperating with each other for further benefit. Moreover, a malicious helper does not change its strategy over time, e.g., sensing accuracy remains the same. We assume identical sensing accuracy in each helper category. For honest helpers, the probability of false alarm and detection are p_f^h and p_d^h, respectively. Malicious helpers, despite generating artificial data, might by chance correctly state that the PU is active. The corresponding detection probability for malicious helpers is $p_d^m = (1-p_0)\alpha_1$ and the false alarm probability is $p_f^m = p_0\alpha_1$ where p_0 is the probability that PU channel is idle.

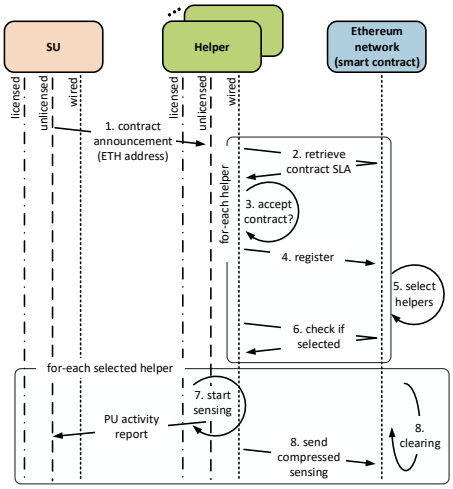


Fig. 2. Interaction between entities in Spass.

Ethereum cost model: While the exact charging model of Ethereum for running computation inside a smart contract depends on many factors, we assume a simplified model according to which we only consider write operations to the contract. The rationale behind this is that the cost of a smart contract is dominated by the amount of data written to it as only write operations need to be written and mined into the BC. Hence for each write task, the cost is calculated as the unit cost of write operation (μ_{eth} Euro/bps) multiplied by the rate of data written to the Ethereum contract (R_{eth} bps): $\mu_{eth}R_{eth}$.

IV. Spectrum Sensing as a Service (Spass) Architecture

Two design goals for Spass architecture are as follows. First, Spass must be practically implemented without the need of a trusted third party. This property is crucial to enable any party with spectrum sensing capability to join the proposed system and offer its service without going through slow or cumbersome authentication process. Second, Spass must be resilient against malicious helpers. To satisfy the first goal, we use Ethereum smart contracts whereas we design a punishment mechanism relying on malicious helper identification inside the Ethereum smart contract for the second goal. We will use Ethereum as our smart-contract environment, but our framework is a general one for any smart contract system.

While Spass is no different than cooperative spectrum sensing in cognitive radio networks, it exhibits several peculiarities when considered from a business perspective. For a profitable business, SU network has to ensure that the profit via the spectrum discovered by Spass must outweigh the cost due to Spass service which consists of smart-contract related costs and payment to sensing helpers. Upon invoking the contract, all computation in Ethereum is subject to some fee to avoid abuse of the Ethereum system [11]. For instance, if a helper is requested to report its sensing data (or some part of it)

to Ethereum for verification purposes, then the helper has to pay the Ethereum for this operation as the caller of the function. Eventually, the SU network buying sensing service has to compensate this cost (in addition to paying the helper for its work) to make helpers eager to participate in the contract. Also, the SU network must meet the sensing reliability requirements asserted by the regulators, e.g., successful detection of a reappearing PU. Hence, while Spass should employ multiple helpers for higher PU detection success [14], from both technical and economics perspective, Spass should identify *malicious helpers* whose sensing accuracy do not meet the required accuracy stated in the smart contract.

Fig. 2 shows the interaction between all entities involved in Spass after the SU network creates a contract in the Ethereum BC. In Step 1, the SU informs co-located sensing nodes about its willingness to offload spectrum sensing by broadcasting the smart contract address using the SU-helper communication link, e.g., ISM channel. A helper can access the Ethereum smart contract using the announced address (Step 2). Based on the received SLA describing the required minimum sensing accuracy for a helper in terms of p_f and p_d, a helper checks whether it is able to meet the SLA (Step 3). After acceptance, the helper registers to the contract by telling its sensing price (Step 4). Next, Spass determines the number of helpers (H) that is required for meeting the sensing accuracy asserted by the regulator.[3] Then, from the set of registered helpers, Spass selects H helpers (Step 5) for sensing, e.g., the cheapest ones. If a helper is selected (Step 6), it starts sensing with the required sensing rate. Sensing outcome is 1-bit data which is calculated based on for example energy detection. Each helper, after each sensing event, sends its sensing data to the SU network through the helper-SU link (Step 7).

Additionally, each helper communicates infrequently its sensing data, which is possibly compressed (details in Sec.V-C), to the smart contract in the Ethereum network (Step 8). Based on the collected helper reports, the clearing, i.e., verification of data and transfer of money, happens inside the smart contract (Step 8). We refer to the time between the helpers joining the contract and the time clearing takes place as a *verification round* (V). Please recall that the communication between the helpers and the SU network takes place just after each sensing period whereas the communication between the helpers and the Ethereum is in a much longer timescale determined by V. This design is motivated by two reasons. First, communication at the time granularity of spectrum sensing (in the order of msecs) task would be unrealistic in Ethereum due to the mining and block processing latency (in the order of minutes) in blockchain systems. Second, helpers need to send the Ethereum the batches of compressed sensing data (yet sufficient number of bits for successful detection of malicious helpers) to keep the smart contract and communication overhead low. As Fig. 3 depicts, after each verification round, the contract runs *malicious helper identification* algorithm (Sec.VI) which aims at discovering the helpers whose sensing accuracy do not meet the agreed accuracy. Such helpers are blacklisted as malicious helpers and will be exempted from further service in Spass. The rest will be paid for their service during the previous verification round and can be selected as helper in the next round.

[3]Either SU network specifies the regulatory requirement for p_f and p_d in the contract or Spass retrieves these values from the regulator directly.

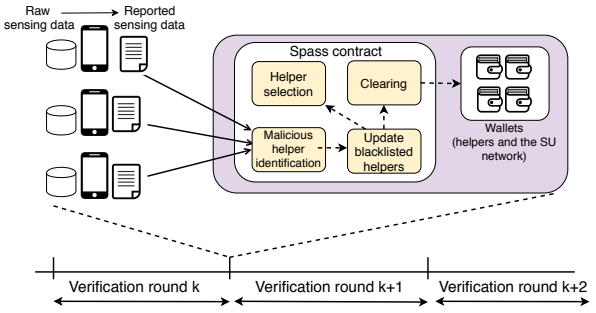


Fig. 3. Spass runs malicious helper identification algorithm at the end of each verification round.

V. Spass-based SU operation

In this section, we address the two questions we raised in Section I: what is the cost of using Spass and under which conditions Spass provides a profitable business to the SU network. First, we compute the spectrum sensing accuracy of Spass in the existence of malicious helpers in the candidate helpers set. Next, we derive the sensing accuracy under OR, AND, and Majority decision rules and find the number of helpers needed for meeting the regulatory requirements. Finally, we focus on economic aspects of Spass to address the raised questions.

A. Sensing accuracy of Spass

Let us define the utility of the SU network in getting service from Spass ($\mathcal{U}^{Spass}$) as the spectrum discovery efficiency, i.e., probability of successfully discovering the spectrum holes by the helpers involved in Spass. Obviously, to calculate $\mathcal{U}^{Spass}$, we need to know total number of helpers, number of malicious helpers, and the decision fusion rule. Let us first overview a generic decision rule K-of-N which can act as a generator for OR, Majority, and AND rules by tuning its parameter K [15]. Here, K-of-N states that at least K out of N users must agree on the sensed phenomena. Setting K to $\{1, N, \lceil \frac{N}{2} \rceil\}$ gives OR, AND, and Majority rule, respectively.

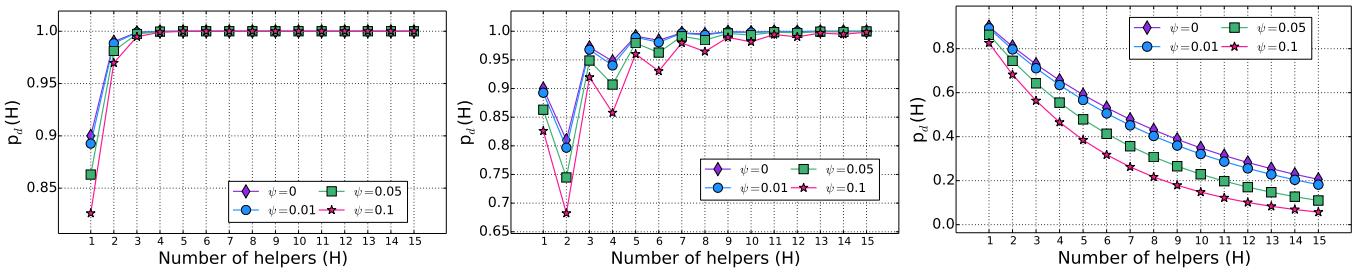
Let H be the number of sensing helpers, $p_f^m(i)$ be the probability of i out of M malicious helpers giving a false alarm, and $p_f^h(j)$ be the probability of exactly j honest helpers giving false alarm. We calculate $p_f^m(i)$ and $p_f^h(j)$ as below:

$$p_f^m(i) = \binom{M}{i} (p_f^m)^i (1-p_f^m)^{M-i} \tag{1}$$

$$p_f^h(j) = \binom{H-M}{j} (p_f^h)^j (1-p_f^h)^{H-M-j}. \tag{2}$$

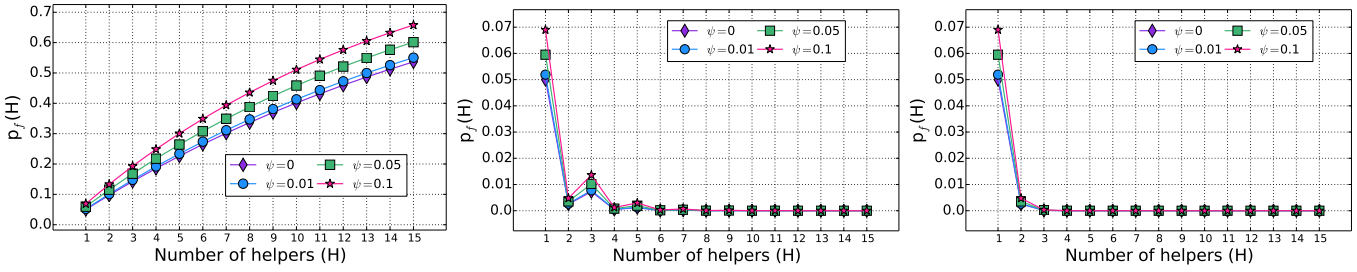
For majority logic, we can derive the false alarm probability under M malicious helpers and total H helpers as follows:

$$p_f(H, M) = \sum_{K=\lceil \frac{H}{2} \rceil}^{H} \sum_{i=0}^{\min(K,M)} p_f^m(i) p_f^h(K-i). \tag{3}$$



(a) OR logic (b) Majority logic (c) AND logic

Fig. 4. Expected PU detection probability under malicious helpers where ψ represents the probability that a node is malicious.



(a) OR logic (b) Majority logic (c) AND logic

Fig. 5. Expected false alarm probability under malicious helpers where ψ represents the probability that a node is malicious.

Similarly, we calculate the probability of detection $p_d(H, M)$:

$$p_d(H, M) = \sum_{K=\lceil \frac{H}{2} \rceil}^{H} \sum_{i=0}^{\min(K,M)} p_d^m(i) p_d^h(K-i). \quad (4)$$

Next, with selected K values, we calculate $\mathcal{U}^{\text{Spass}}$ as follows:

$$\mathcal{U}^{\text{Spass}} = p_0(1 - p_f(H, M)). \quad (5)$$

B. How many helpers are needed?

An SU network desires that Spass minimizes the number of helpers as each helper will be paid for its sensing service. However, sensing with more helpers might help decreasing false alarm probability and thereby might increase utility in (5). Moreover, Spass has to ensure that the PU detection probability is larger than the limit set by the regulator. Let us denote the regulator-asserted false alarm and detection probability by p_f^* and p_d^*, respectively. Then, Spass has to ensure $p_d(H) \geqslant p_d^*$ and $p_f(H) \leqslant p_f^*$, where $p_d(H)$ and $p_f(H)$ are expected sensing accuracies under H helpers. If a node acts maliciously with probability ψ, the expected value of p_d and p_f under H helpers are calculated as follows:

$$p_d(H) = \sum_{M=0}^{H} \binom{H}{M} \psi^M (1-\psi)^{H-M} p_d(H, M), \quad (6)$$

$$p_f(H) = \sum_{M=0}^{H} \binom{H}{M} \psi^M (1-\psi)^{H-M} p_f(H, M). \quad (7)$$

Fig. 4 and Fig. 5 show $p_d(H)$ and $p_f(H)$ for OR, Majority, and AND logic. We used the following parameters: $p_0 = 0.6$, $p_f^h = 0.05$, $p_d^h = 0.90$, $\alpha_1 = (1 - p_0) = 0.4$. Figures show that OR and Majority rules are better in identifying the PU activity compared to AND logic, which is a very conservative policy. All rules lose their accuracy with increasing number of malicious helpers. Comparing p_f of OR and Majority, we observe that Majority is more robust against malicious helpers as false alarms are unlikely in a network with more honest helpers than malicious helpers. Different than Majority and AND rules, adding new helpers does not help decreasing p_f in OR rule as shown in Fig.5a. In that case, each helper might give a false alarm and changes the final decision to a false alarm, which leads to higher p_f with increasing number of helpers. Therefore, the SU network should consider Majority logic in its fusion operation.

Based on the knowledge of ψ, the contract can decide on the number of helpers. In case the expected sensing reliabilities are not sufficient for regulations, e.g., due to insufficient number of helpers, Spass informs the SU network about the failure of the sensing service. Otherwise, the contract becomes active. In the next section, we provide a model to decide when our proposal becomes a profitable business model for an SU network.

C. When does Spass provide a profitable business model?

Let the spectral efficiency of the SU network be κ bits/(seconds$\times$Hertz). The SU network charges its customers μ €/bps for each second of service. Then, the SU network's revenue by serving its users via opportunistic spectrum access on a spectrum band with bandwidth B Hertz equals to:

$$\Upsilon^+ = \mu \times \kappa \times \mathcal{U}^{\text{Spass}} \times B \ \text{€ per second.} \quad (8)$$

However, if the SU network uses Spass for discovering idle spectrum, it has to pay for the Ethereum smart contract use and for the sensing service of helpers. Let us denote euro-equivalent cost of using Ethereum smart contract by μ_{eth} € per

bps. Here, we assume that the Ethereum cost is dominated by the cost of write operation to the Ethereum BC. In case H helpers are selected in the contract and each helper writes R_{eth} bps to Ethereum for verification of sensing data, the cost of using Spass equals to $\mu_{eth} \times H \times R_{eth}$. As a helper also spends its energy resources for sensing, it must be paid also for its sensing service. We denote by μ_s the price of sensing per bps. Note that sampling rate of sensing might be larger than the rate at which the sensing data is written to the Ethereum smart contract, i.e., $R_s \geqslant R_{eth}$. To represent this fact, we refer to the ratio as *compression factor* where:

$$\beta = \frac{R_s}{R_{eth}} \text{ and } \beta \geqslant 1.$$

Adding these cost terms that SU network has to pay for, we define the cost of Spass as follows:

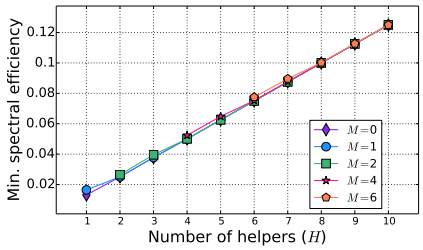
$$\Upsilon^- = (\mu_{eth}\frac{R_s}{\beta} + \mu_s R_s) \times H \quad € \text{ per second.} \tag{9}$$

For the SU network to maintain a profitable business by using Spass, its profit must be positive: $\Delta\Upsilon = \Upsilon^+ - \Upsilon^- > 0$. For this inequality to hold, we can derive the minimum spectral efficiency required for an SU network under a given cost model, i.e., μ_s and μ_{eth}, as follows:

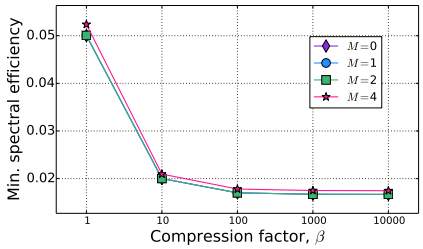
$$\kappa > \frac{R_s(\mu_{eth}/\beta + \mu_s) \times H}{\mu \times \mathcal{U}^{\text{Spass}} \times B}. \tag{10}$$

Hence, the minimum spectral efficiency $\kappa_{\min}$ corresponds to the point where the above inequality operator is set to equality operator. Similarly, using (10), we can calculate the minimum price $\mu_{\min}$ the SU network has to charge its customers. As we observe in (10), the SU network prefers keeping the amount of data written to the smart contract low while maintaining a high utility $\mathcal{U}^{\text{Spass}}$. Hence, Spass contract includes a malicious helper identification scheme to leave out malicious helpers whose inaccurate sensing data might decrease spectrum access efficiency. Also, the SU network desires high compression of the sensing data so that the rate at which data is written to the contract R_s/β becomes small.

Fig. 6 shows the impact of the number of helpers on $\kappa_{\min}$ that is required to make Spass-based operation profitable for the SU network. We plot the results under various number of malicious helpers for $\mu_{eth} = 0.1$, $\mu_s = 0.05$, and $\mu = 1$. From Fig. 6a, we observe that higher number of malicious helpers requires slightly higher spectral efficiency for the same number of helpers. Generally speaking, contracting a higher number of helpers requires a higher spectral efficiency of the SU network because all helpers have to be paid. However, recall that the SU can also increase its sensing performance with higher number of helpers (i.e., PU detection and false alarm performance). When there are more malicious helpers, the required spectral efficiency increases slightly as utility might decrease with increasing number of malicious helpers. Fig 6b draws $\kappa_{\min}$ with increasing compression factor β. An SU network with a lower spectral efficiency can still operate based on Spass by using a high compression factor or a higher price μ charged to its customers. With these insights from our model, we conclude that an SU network desires that Spass can avoid malicious helpers and can work reliably under a high compression factor value.



(a) Impact of number of helpers H under $\beta = 1$.



(b) Impact of compression β under $H = 4$.

Fig. 6. Min. spectral efficiency required for the SU to profit from Spass.

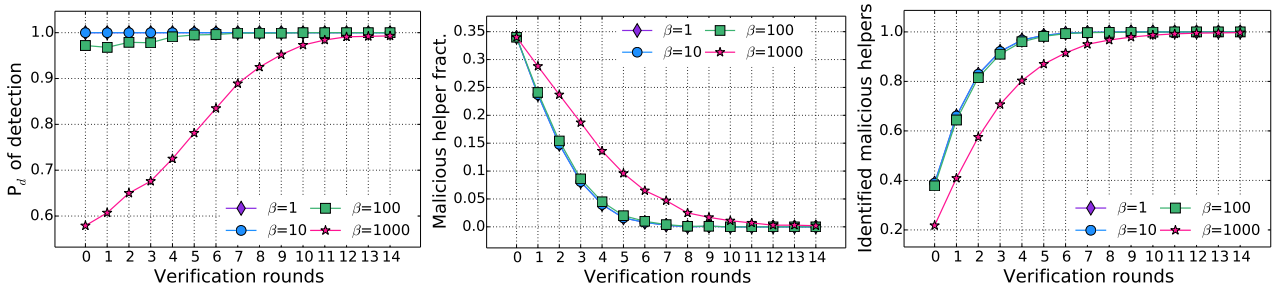
VI. Two-Threshold Based Voting (TTBV): An Approach for Identification of Malicious Helpers

Denote the sensing outcomes of helper i (h_i) by a time series $\mathcal{X} = \{X_{t,i}\}$ where $X_{t,i}$ is the PU's channel state observed by h_i at time t. We are interested in developing an algorithm for detecting malicious helpers based on the data collected at the smart contract in Ethereum (Fig. 3). Let $Y(h_i)$ denote the reported data of h_i and f be a function, e.g., average, taking the sensing outcomes as its input and returning Y_i. Then, we define the collected helper data at the smart contract as follows: $\mathcal{Y} = \{Y_i | Y_i = f(\{X_{t,i}\}), h_i \in \mathcal{H}\}$ where $\mathcal{H}$ is the set of sensing helpers. Our goal is to design f which can achieve a high malicious helper detection accuracy while maintaining a high compression value measured by $\beta = |\mathcal{Y}|/|\mathcal{X}|$. We define malicious helper detection accuracy by the ratio of number of correctly identified malicious helpers over all malicious helpers participating in the smart contract. Note that misclassifications are possible, i.e., honest helpers being identified as malicious, and should be kept to minimum.

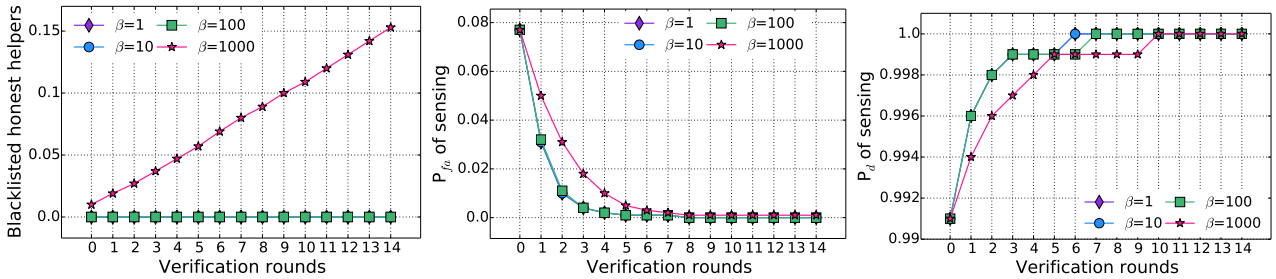
Let us define the distance between two sensing reports by:

$$d_{i,j} = \frac{\mathcal{D}(Y_i, Y_j)}{|Y_i|}, \tag{11}$$

where $\mathcal{D}$ is the Hamming distance between two vectors. We can interpret $d_{i,j}$ as the probability that h_i and h_j would differ in a randomly picked bit of their sensing report. We expect a high distance between a malicious helper report and that of the honest helpers. However, as honest helpers might experience sensing imperfections due to the noise in the sensing process, sensing report of two honest helpers might also differ from



(a) P_d in malicious helper identification. (b) Fract. of malicious helpers in selected helpers. (c) Fract.of identified malicious helpers.



(d) Fract. of blacklisted honest helpers. (e) P_{fa} in sensing. (f) P_d of spectrum sensing.

Fig. 7. Performance of two-threshold based identification with $w_{th} = 0.8$.

each other. Using the approach of [16], we calculate the expected distance between two honest helpers as follows:

$$d^{h,h} = 2p_0 p_f^h(1-p_f^h) + 2(1-p_0)p_d^h(1-p_d^h). \quad (12)$$

Next, we calculate the expected distance between a honest and a malicious helper [16]:

$$\begin{aligned} d^{h,m} =& p_0\left(p_f^h(1-p_f^m) + (1-p_f^h)p_f^m\right) \\ &+ (1-p_0)\left(p_d^h(1-p_d^m) + (1-p_d^h)p_d^m\right). \end{aligned} \quad (13)$$

Using these two distance values, the smart contract applies a voting scheme for determining the state of each helper as in Fig. 8. First, it calculates pairwise distance values. Next, for each helper h_j, the smart contract checks what the other helpers data conclude about h_j's state based on the distance metric. Vote of h_i denoted by $w_{i,j}$ can be interpreted as the probability that h_j is seen as malicious from the perspective of h_i. Then, the smart contract sets $w_{i,j}$ as follows:

$$w_{i,j} = \begin{cases} 0 & \text{if } d_{i,j} \leqslant d^{h,h} \\ 1/(H-1) & \text{if } d_{i,j} \geqslant d^{h,m} \\ \frac{d_{i,j}-d^{h,h}}{(d^{h,m}-d^{h,h})\times(H-1)} & \text{ow.} \end{cases}$$

Briefly, if $d_{i,j} \leqslant d^{h,h}$, h_i thinks that h_j is a honest helper and its decision is $w_{i,j} = 0$. When $d^{h,h} < d_{i,j} < d^{h,m}$, h_i is not sure and therefore its decision is $0 \leqslant w_{i,j} < 1/(H-1)$ and we set this weight as follows: $\frac{d_{i,j}-d^{h,h}}{(d^{h,m}-d^{h,h})\times(H-1)}$. Since there are $(H-1)$ helpers voting for helper j, we normalize each helper's vote by $(H-1)$. Summing all $w_{i,j}$, we find

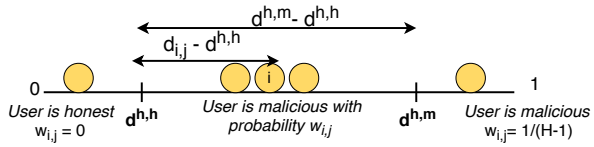


Fig. 8. Two-threshold based voting for deciding whether helper j is malicious.

the probability that h_j is expected to be malicious. Setting $w_{th} \geqslant 0.5$, the smart contract finalizes its decision as follows:

$$\text{State of } h_j = \begin{cases} \text{Malicious} & \text{if } \sum_{i,i\neq j} w_{i,j} \geqslant w_{th} \\ \text{Honest} & \text{else.} \end{cases}$$

Note that TTBV should identify malicious helpers without resulting in too high false alarms, since a blacklisted helper is excluded from Spass and is not paid for its sensing service. A solution with a high false alarm decreases incentives for honest helpers to participate in Spass. By tuning w_{th} through Monte-Carlo simulations, we can provide a trade-off between false alarms and identification success of malicious helpers. Finally, computational complexity of TTBV is $\mathcal{O}(H^2)$.

VII. Performance Evaluation

To evaluate the performance of TTBV, we simulate our system using our custom Python-based discrete-event simulator. We report the average statistics of 10^4 repetitions of each scenario for statistical significance. We set total number of nodes to 20, $M = 5$, $H = 8$, and assume identical sensing price for each helper. Spass runs the verification algorithm for 15 rounds and each verification round consists of 5000 timeslots. We set $p_f^h = 0.05$, $p_d^h = 0.90$, $p_0 = 0.6$, $\alpha_1 = 0.6$, and sensing rate R_s is 1 Hz.

We use the following performance metrics in our analysis. Let $\mathcal{M}$ denote the set of all malicious helpers, $\mathcal{M}^{Spass}$ be the set of malicious helpers selected for sensing, and $\widetilde{\mathcal{M}}$ be the set of helpers claimed to be malicious by TTBV. The malicious helper fraction in a contract is: $\frac{|\mathcal{M}^{Spass}|}{|\mathcal{H}^{Spass}|}$ where $\mathcal{H}^{Spass}$ is the set of helpers participating in the contract. Then, identification success in a single verification round with id k is: $\frac{|\mathcal{M}_k \cap \widetilde{\mathcal{M}}_k|}{|\mathcal{M}_k|}$ where $\mathcal{M}_k$ is the set of malicious helpers participating in Spass at round k. The fraction of identified malicious helpers till the current verification round K is $\frac{|\bigcup_{k \leqslant K} \mathcal{M}_k \cap \widetilde{\mathcal{M}}|}{\mathcal{M}}$. Similarly, we calculate the fraction of blacklisted honest helpers.

A. Impact of compression factor on the performance of TTBV

Fig. 7 plots the performance metrics with increasing round number under various β values. As Fig. 7a shows, TTBV can identify malicious helpers with high accuracy even under $\beta = 100$. When there is low compression, TTBV can almost perfectly detect all malicious helpers in the helper set. After each round, these malicious helpers are blacklisted and excluded from Spass. Fig. 7b shows the change in the fraction of malicious helpers in each sensing helpers set with increasing round index. After 6-7 rounds, all malicious helpers are identified for $\beta = \{1, 10, 100\}$ (Fig. 7c) resulting in no malicious helpers in the helper set as seen in Fig. 7b. For $\beta = 1000$, it takes longer to blacklist all malicious helpers, e.g., 11 rounds. Moreover, such a compression value results in higher false alarms as observed in Fig. 7d. Over time, more honest helpers are incorrectly added to the blacklisted helpers when $\beta = 1000$. This behavior is undesirable and must be avoided. On the other hand, for lower β, TTBV does not result in false alarms, e.g., no honest helpers blacklisted in Fig. 7d. Hence, under these settings, an SU network can define in its contract the compression factor as $100 \leqslant \beta < 1000$.

As for sensing accuracy, since majority logic is robust to malicious helpers, we see a less visible difference among different β values. With increasing round number, the malicious helpers are excluded from helper set and therefore we observe an improvement in sensing accuracy. Under all values, sensing accuracy is high with low false alarms in Fig. 7e and almost 100 percent detection success in Fig. 7f.

B. Practical Feasibility

Here, we discuss the practical feasibility of the proposed system taking real values. In this example, we assume that the sensing helpers are devices with a lifetime of 10 years and a total cost of operation during the lifetime of 100 €. Moreover, four helpers are selected for sensing and the SLA of the contract requires a sensing interval of 10 Hz. Regarding the costs for provisioning of the Ethereum smart contract, we use the following simplified model according to which the amount of data stored in the contract dominates the cost whereas the computational effort within the contract is negligible. According to [11], storing a kilobyte of data in a smart contract consumes 640k of gas. The lowest possible gas price at the time of writing is around 1 Gwei (10^9 Gwei = 1 ETH) whereas the market price of 1 ETH is 500 €. In contrast to the fixed payout to the sensing helpers, the cost of the smart contract depends on the used compression as with higher compression less information need to be stored in the smart contract.

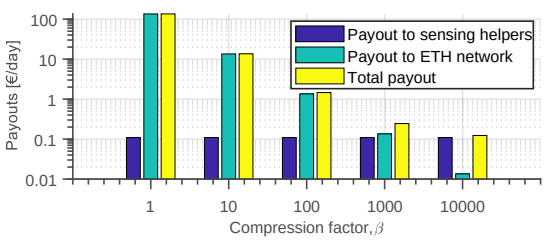


Fig. 9. SU's payouts to sensing helpers and Ethereum network.

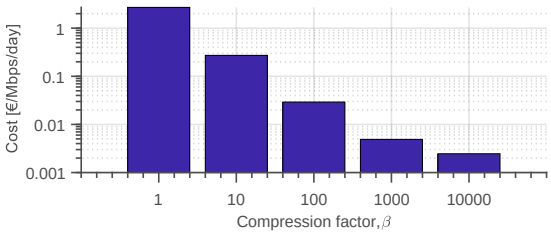


Fig. 10. Cost of Spass for delivering 1 Mbps per day.

Fig. 9 shows the daily payouts to the two parties for different compression factors β. Here, the amount of payments to the sensing helpers corresponds to the minimum amount required to cover their operation costs, i.e. 0.11 € per day and helper. We see that for low compression factors the total payment is dominated by the payouts to the Ethereum network while the cost of operation of the sensing helper is negligible. For $\beta \geqslant 1e4$, the situation is exactly the opposite where the cost of the smart contract becomes negligible.

Using the above data, we calculate the amount the SU network has to charge its customers, i.e., μ in (10), to cover the cost of Spass. In our analysis, we do not include the cost of operating the SU network itself, e.g., OPEX. We assume an average spectral efficiency of 5 bits/s/Hz for the SU and 20 MHz bandwidth for the PU channel. Under these assumptions, Fig. 10 shows the minimum price the SU has to charge from its customers for delivering 1 Mbps to be able to cover the cost of sensing helpers and Ethereum network. For a compression factor of 100, it is 0.03 € per Mbps and day.

C. Ethereum Contract

We have also implemented Spass in Solidity to understand the actual cost of each function invocation in Ethereum BC. Fig. 11 illustrates the UML class diagram representing the interactions with the contract. Moreover, we also present in the below listing our implementation to give an idea about how Spass can be represented in Solidity [4].

After running this contract in Ethereum, we measure the gas consumption of each function call (refer to Table I). Moreover, we see the price for each call at today's market price of Ethereum and SafeLow (<30 min) confirmation time. We

[4]Source code is available under: https://github.com/zubow/Spass_contract

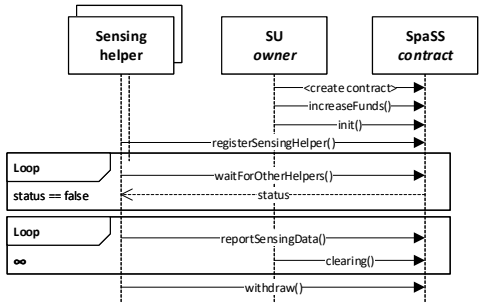


Fig. 11. UML sequence diagram showing the interaction between helpers, SU network, and the Spass Ethereum smart contract implemented in Solidity.

TABLE I
COST OF FUNCTION INVOCATION IN CONTRACT (H=HELPERS).

Function	Caller	Cyclic	TX (gas)	Ex. (gas)	€
<create contract>	SU	no	1638213	1223229	8.33
increaseFunds	SU	no	21579	307	0.06
init	SU	no	68630	46270	0.34
registerSensingHelper	H	no	178506	154994	0.98
waitForOtherHelpers	H	no	22995	1723	0.07
reportSensingData	H	yes	56814	32406	0.26
clearing	SU	yes	54372	32908	0.26
withdraw	H	no	19426	13154	0.10

use the following configuration for the contract initialization: sensing rate R_s of 10 Hz (sens_f), compression factor β of 100 (cp_f=100) and a verification round duration V of 15 min (round_s), i.e., 15 Bytes of sensing data in each round. We see that, besides the new contract generation, the registration of a new sensing helper is also costly. This is desirable to prevent spoofing attacks. In contrast, the cost for invocation of function being called periodically, i.e. *reportSensingData* and *clearing*, is sufficiently low to make the Spass a profitable business. We do not consider the cost for detection of malicious helpers in *reportSensingData* as we can assume that TTBV removes malicious helpers from the contract only after a few rounds.

```
pragma solidity ^0.4.0;
contract Spass {
    struct Helper { // relevant sensing helper parameter
        uint id; uint p_f; uint p_d; uint priceSenseBit;
        uint last_report_seq; bytes data; bool toBlock;
    }
    address public owner; // of this contract -> the SU
    mapping(address => Helper) shMap; address[] shLst;
    mapping (address => uint) public pendingWithdrawals;
    uint sens_f; uint round_s; // sens. sampling, round len
    uint data_b; // size of sensing report in bytes
    uint max_p_f; uint min_p_d; // per 1000; req. quality
    uint curr_seq; // the current expected seq
    /* Create new Spass contract */
    function Spass() { owner = msg.sender; }
    /* Called by contract owner (SU) to initialize it */
    function init(uint _sens_f, uint _round_s, uint _cp_f,
         uint _max_p_f, uint _min_p_d) public ownerOnly {
        sens_f = _sens_f; round_s = _round_s;
        max_p_f = _max_p_f; min_p_d = _min_p_d;
        data_b = _round_s * _sens_f / _cp_f / 8 + 1;
    }
    /* Add new helper to the contract; called by H */
    function registerSensingHelper(address _sHelper, uint
         _id, uint _priceSenseBit, uint _p_f, uint _p_d)
         public returns(bool) {
        if (shMap[_sHelper].priceSenseBit != 0)
            return false; // already registered
        if (rejectHelper(_priceSenseBit, _p_f, _p_d))
            return false; // reject helper, e.g. high price
        shMap[_sHelper] = Helper({id: _id, p_f: _p_f, p_d:
            _p_d, priceSenseBit: _priceSenseBit,
            last_report_seq: 0, data: new bytes(data_b),
            toBlock: false
        }); shLst.push(_sHelper); return true;
    }
    /* Check if sufficient Hs available; called by H */
    function waitForOtherHelpers() public returns(bool) {
        if (sufficientRegisteredHelpers()) return false;
        return true;
    }
    /* Periodically called by Hs to report sensing data */
    function reportSensingData(address _sHelper, uint _id,
         uint _seq, bytes _data) public returns(bool) {
        if (shMap[_sHelper].priceSenseBit == 0)
            return false; // unknown helper
        if ( (shMap[_sHelper].last_report_seq + 1) != _seq)
            return false; // outdated sensing data
        for (uint i=0; i<data_b; i++) {
            shMap[_sHelper].data[i] = _data[i];
        } shMap[_sHelper].last_report_seq = _seq;
        return true;
    }
    /* At end of each round SU makes payments to Hs */
    function clearing(uint _seq) public ownerOnly returns(
         bool) {
        if (curr_seq != _seq) return false;
        markMaliciousNodes(); // mark cheaters
        for (uint i=0; i<shLst.length; i++) {
            if (shMap[shLst[i]].toBlock == true) {
                blockSensingHelper(shLst[i]);
            } else { notifyPayment(shLst[i]); }
        } curr_seq++; /* next round */ return true;
    }
    /* Notify H of its credit for amount of sensing. */
    function notifyPayment(address _sHelper) ownerOnly
         private returns(bool) {
        uint payment = shMap[_sHelper].priceSenseBit * (
            round_s * sens_f); // Pay agreed price
        pendingWithdrawals[_sHelper] += payment;
        return true;
    }
    /* Allows H to withdraw any outstanding credit */
    function withdraw() public returns(bool) {
        uint amount = pendingWithdrawals[msg.sender];
        if (amount <= 0) return false;
        pendingWithdrawals[msg.sender] = 0;
        if (msg.sender.send(amount)) { return true;
        } else {
            pendingWithdrawals[msg.sender] = amount;
            return false; }
    }
    /* Send ETH to contract so Hs can withdraw funds. */
    function increaseFunds() payable {}
}
```

D. Discussion

Two other knobs the SU network can play with for tuning its operation are sensing rate R_s and verification round duration V. Increasing either of these parameters results in more sensing bits to be used in verification which in return increases the accuracy of malicious helper detection. However, the amount of payment to an uncaught malicious helper is also an increasing function of total sensing bits. Hence, there is an optimal setting, i.e., R_s and V, minimizing the total cost. We leave optimal setting of parameters to a further work. To avoid cases where the malicious helpers earn money with cheating, another improvement to our model could be introducing a rule in the contract that a helper will be paid only after a certain number of rounds (e.g., 6-7 considering Fig. 7c) if it still remains in the system without being blacklisted. This is to ensure that malicious behavior would become irrational as acting that way would not provide any profit to the helper given that our heuristic can identify all malicious helpers after

a certain number of rounds. Such a rule prevents also spoofing attacks as a malicious helper rejoining the system with a new id has to wait for at least a minimum number of rounds to receive a benefit.

VIII. Related Work

We can categorize the related literature into two as follows. *Abnormal user identification in spectrum sensing*: The most relevant work to ours is [16] which proposes an abnormal sensing node identification using the pairwise similarity of sensing results of helpers. While our solution is inspired by [16], we cannot directly apply the proposed approach as our malicious helper model differs from that of [16]. Moreover, our goal is also to achieve high compression value due to the cost of Ethereum while it is not a concern of [16]. Outlier detection methods in sensor networks, e.g., [17], are also relevant to our work. While we do not claim that our approach overperforms the existing approaches, we emphasize that the existing schemes should be tailored for our scenario especially considering the cost of Ethereum usage and related latency for running an operation in the contract.

Usage of blockchain in resource sharing concepts: Steering [18] proposed to use blockchain as a way to reduce transaction costs in the Citizens Broadband Radio Service through automatization of complex business-to-business workflows in contracting and brokering a data exchange. Different use cases were described including sensing-as-a-service. However, no details were provided on the design of such a service. Similarly, [19] proposes to use blockchain for dynamic spectrum sharing for vehicular CRs and introduces a virtual currency to pay for spectrum access. The work in [20] is motivated by the trend towards Small-Cell-as-a-Service, a scenario where an individual home or business user can become a service provider for MNOs. In particular, the authors proposed using Ethereum smart contracts to implement simple SLAs between small cell providers and MNOs. While our idea is in this line of works promoting the use of blockchain for practical DSA, our work differs from all its predecessors as we model the technical as well as economic aspects of such a smart-contract based spectrum access system.

IX. Conclusions

We have introduced spectrum sensing as a service (Spass) which is a smart-contract based solution facilitating a network operator (referred to as SU network) to buy sensing service from the helpers in its neighborhood. In our model, the SU network publishes its requirements for sensing in the contract and the helpers who agree on the specified terms join the smart contract. In such a system, two key questions we raised were: what is the cost of using smart contracts and under which conditions an SU network can profit from this business model? To address these questions, we first derived the expected spectrum sensing accuracy and cost of Spass under the existence of malicious helpers, whose goal is to receive payment from the contract without actually performing sensing. We proposed a malicious helper identification scheme whose accuracy remains high even when the sensing reports submitted to the contract are much sparser than the actual sensing data. As future work, we plan to work on optimizing verification rounds as well as sensing rate to minimize monetary cost of Spass while ensuring a certain spectrum sensing accuracy.

Acknowledgment

This work has been supported by the European Union's Horizon 2020 research&innovation program under grant agreement No 645274 (WiSHFUL project). We would like to thank Nadiya Romanova, Joel Stenkvist, Pratik Walvekar, and Jakob Wiren for fruitful discussions in the early phase of this paper.

References

[1] J. G. Andrews *et al.*, "What will 5G be?" *IEEE Journal on selected areas in communications*, vol. 32, no. 6, pp. 1065–1082, 2014.

[2] S. Bayhan, A. Zubow, and A. Wolisz, "Coexistence gaps in space via interference nulling for LTE-U/WiFi coexistence," in *IEEE WoWMoM*, 2018.

[3] X. Wang and R. Berry, "The impact of bundling licensed and unlicensed wireless service," *arXiv preprint arXiv:1801.01989*, 2018.

[4] P. Van Wesemael *et al.*, "Robust distributed sensing with heterogeneous devices," in *IEEE Future Network & Mobile Summit*, 2012.

[5] G. Ganesan and Y. Li, "Cooperative spectrum sensing in CRNs," in *International Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*, 2005, pp. 137–143.

[6] S. Eryigit *et al.*, "Optimal cooperator set selection in social CRNs," *IEEE Trans. on Vehic. Tech.*, vol. 65, no. 8, pp. 6432–6443, Aug 2016.

[7] L. Luu *et al.*, "Making smart contracts smarter," in *ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[8] S. Yrjola, "Analysis of blockchain use cases in the citizens broadband radio service spectrum sharing concept," pp. 128–139, 01 2018.

[9] Y.-J. Choi, Y. Xin, and S. Rangarajan, "Overhead-throughput tradeoff in cooperative CRNs," in *Wireless Communications and Networking Conference (WCNC)*, 2009, pp. 1–6.

[10] R. Chen *et al.*, "Toward secure distributed spectrum sensing in CRNs," *IEEE Communications Magazine*, vol. 46, no. 4, 2008.

[11] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[12] S. Hu *et al.*, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," *IEEE INFOCOM*, 2018.

[13] S. Devarakonda *et al.*, "Real-time air quality monitoring through mobile sensing in metropolitan areas," in *Proceedings of the 2nd ACM SIGKDD international workshop on urban computing*. ACM, 2013, p. 15.

[14] Y.-C. Liang *et al.*, "Sensing-throughput tradeoff for CRNs," *IEEE Trans. on Wireless Comm.*, vol. 7, no. 4, pp. 1326–1337, 2008.

[15] S. Althunibat, M. Di Renzo, and F. Granelli, "Optimizing the K-out-of-N rule for cooperative spectrum sensing in CRNs," in *IEEE Global Communications Conference (GLOBECOM)*, 2013, pp. 1607–11.

[16] H. Li and Z. Han, "Catch me if you can: An abnormality detection approach for collaborative spectrum sensing in CRNs," *IEEE Transactions on Wireless Communications*, vol. 9, no. 11, pp. 3554–3565, 2010.

[17] B. Sheng *et al.*, "Outlier detection in sensor networks," in *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2007, pp. 219–228.

[18] N. I. Steering, "Analysis of blockchain use cases in the citizens broadband radio service spectrum sharing concept," *Cognitive Radio Oriented Wireless Networks*, p. 128, 2018.

[19] K. Kotobi and S. G. Bilen, "Secure blockchains for DSA: A decentralized database in moving CRNs enhances security and user access," *IEEE Vehicular Technology Magazine*, vol. 13, no. 1, pp. 32–39, 2018.

[20] E. Di Pascale *et al.*, "Smart contract SLAs for dense small-cell-as-a-service," *arXiv preprint arXiv:1703.04502*, 2017.