Cuckoo Filter-Based Location-Privacy Preservation in Database-Driven Cognitive Radio Networks

Mohamed Grissa, Attila A. Yavuz, and Bechir Hamdaoui
Oregon State University, grissam,yavuza,hamdaoub@onid.oregonstate.edu

***Abstract*—Cognitive Radio Networks (CRNs) enable opportunistic access to the licensed channels by allowing secondary users (SUs) to exploit vacant channel opportunities. One effective technique through which SUs acquire whether a channel is vacant is using geo-location databases. Despite their usefulness, geo-location database-driven CRNs suffer from location privacy threats, merely because SUs have to query the database with their exact locations in order to learn about spectrum availability.**

In this paper, we propose an efficient scheme for database-driven CRNs that preserves the location privacy of SUs while allowing them to learn about available channels in their vicinity. We present a tradeoff between offering an ideal location privacy while having a high communication overhead and compromising some of the users' coordinates at the benefit of incurring much lower overhead. We also study the effectiveness of the proposed scheme under various system parameters.

***Keywords*—Database-driven spectrum availability, location privacy preservation, cognitive radio networks, Cuckoo Filter.**

I. Introduction

Cognitive radio networks (CRNs) have emerged as a key technology for addressing the problem of spectrum utilization inefficiency [1]. CRNs allow unlicensed users, also referred to as Secondary Users (SUs), to access licensed frequency bands opportunistically, so long as doing so does not harm licensed users, also referred to as Primary Users (PU)s. In order to enable SUs to identify vacant frequency bands, also called white spaces, the Federal Communications Commission (FCC) has adopted two main approaches: *spectrum sensing-based approach* and *geo-location database-driven approach*.

In the sensing-based approach [2], SUs themselves sense the licensed channels to decide whether a channel is available prior to using it so as to avoid harming PUs. In the database-driven approach, SUs rely on a geo-location database (DB) to obtain channel availability information. For this, SUs are required to be equipped with GPS devices so as to be able to query the DB on a regular basis using their exact locations. Upon receipt of a query, the DB returns to the SU the list of available channels in its vicinity, as well as the transmission parameters that are to be used by the SU. This DB-driven approach has advantages over the sensing-based approach. First, it pushes the responsibility and complexity of complying with spectrum policies to the DB. Second, it eases the adoption of policy changes by limiting updates to just a handful number of databases, as opposed to updating large numbers of devices [3]. Companies like Google and Microsoft, among others, were selected by FCC to administrate these geo-location databases, following the guidelines provided by PAWS.[1]

Despite their effectiveness in improving spectrum utilization efficiency, DB-driven CRNs suffer from serious security and privacy threats. The disclosure of location privacy of SUs has been one of such threats to SUs when it comes to obtaining spectrum availability from DBs. This is simply because the users have to share their locations with the DB to learn about spectrum availability. The fine-grained location, when combined with publicly available information, could lead to even greater private information leakage; it could, for example, be used to infer private information like shopping patterns, preferences, behavior and beliefs, etc. [4]. Being aware of such potential privacy threats, SUs may refuse to use the DB for spectrum availability information, thus making the need for location-privacy preserving schemes for DB-driven spectrum access of high importance.

In this paper, we propose a new scheme that preserves the location privacy of SUs in database-driven CRNs. We show that our proposed scheme preserves the location privacy of SUs while outperforming existing approaches in terms of the amount of overhead to be incurred in the process of protecting users' location privacy, thereby making it more scalable and practical. In addition, we show that a significant reduction in the scheme's overhead can be further achieved by allowing the leakage of some information that makes little to no compromise of the users' location, yet reduces the overhead substantially. We also study the impact of system parameters on the performances of our proposed scheme, and compare them against those obtained via existing approaches.

The rest of this paper is organized as follows: we present the related work in Section II. In Section III, we provide our system model and a brief overview of the Cuckoo filter. In Section IV, we present our proposed scheme. We evaluate and analyze the performance of the proposed scheme in Section V, and conclude in Section VI.

II. Related Work

Despite the importance of protecting the location privacy of users, little attention was drawn to cope with it in the literature. While some works focused on addressing this issue in the context of collaborative spectrum sensing [5]–[7], others

[1]PAWS (Protocol to Access White-Space) is a protocol introduced to enable interoperability between devices and databases [3].

DOI: 10.1109/WSCNIS.2015.7368280

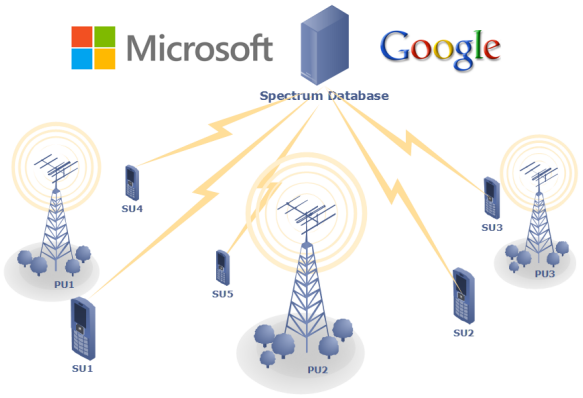


Fig. 1: Database-driven CRN

addressed it in the context of dynamic spectrum auction [8]. However, these works are skipped here since they are not within the scope of the paper.

In the context of DB-driven CRN s, Gao et al. [9] identified a new attack that can compromise the location privacy of SUs that have to communicate with DB through a base station. The area controlled by the base station is viewed as a grid containing multiple cells, where the location of each SU is determined by the cell in which the SU is located in. In this attack, DB, which is assumed to know the content of the communication between users and the base station, can infer the location of SUs based on their channel utilization pattern. Basically, since a SU cannot use a channel unless it is outside the coverage area of any PU currently using the channel and given that a SU has to keep switching from one channel to another to avoid interfering with PUs, this allows the DB to narrow down the location of users by finding the intersection area of the complements of PUs' coverage areas over time. To thwart this attack, they propose a Private Information Retrieval (PIR)-based scheme, termed $PriSpectrum$, that, despite its merits, has several limitations: (i) It assumes that the users are static over time; (ii) It does not offer ideal location privacy; and (iii) PIR-based approaches are known to be expensive in terms of communication and computation overhead. Troja et al. [10] proposed another approach that allows users to communicate in a peer-to-peer manner to share their spectrum availability information that they obtained from previous queries. This reduces the number of executions of the *PIR* protocol used by the users to privately retrieve spectrum information from DB. This scheme, just like the previous one, is designed for limited areas and has also the drawbacks of PIR-based approaches.

III. System Model and Cuckoo Filter

In this section, we first begin by stating our system model. Then, for completeness, we overview the Cuckoo filter approach, which is used in our privacy-preserving scheme that we present in the next section.

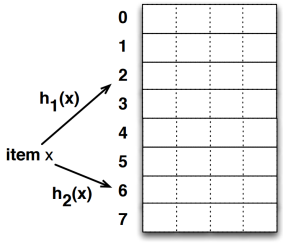


Fig. 2: Cuckoo Filter: 2 hashes per item, 8 buckets each containing 4 entries

A. Database-driven CRN Model

We consider a CRN that consists of a set of SUs and a geo-location database (DB). SUs are assumed to be enabled with GPS capability, and to have access to the DB for obtaining spectrum availability information, as shown in Figure 1. To obtain spectrum availability information, a SU queries the database by including its location and its device characteristics. DB responds with a list of available channels at the specified location and a set of parameters for transmission over those channels. The user then selects and uses one of the returned channels, and while using the channel, needs to recheck the channel's availability on a daily basis or whenever it changes its location by 100 meters as mandated by the *PAWS* protocol [3]. For more information about the requirements and policies of database-driven CRNs, readers are referred to [3].

B. Cuckoo Filter

The novelty of our proposed privacy-preserving scheme, to be presented in the next section, lies in the use of the *Cuckoo Filter* technique. Therefore, for completeness, we provide in this section a quick overview of this filter.

Essentially, Cuckoo Filter is a new data structure proposed in [11] to replace Bloom Filter as a method for testing set membership; i.e., for testing whether an element is a member of a set. It uses *Cuckoo Hashing* [12] and was designed to serve applications that need to store a large number of items while targeting low false positive rates and requiring storage space smaller than that required by Bloom Filters. A false positive occurs when the membership test returns that an item exists in the Cuckoo Filter (i.e., belongs to the set) while it actually does not. A false negative, on the other hand, occurs when the membership test returns that an item does not exist while it actually exists. In Cuckoo filters, false positives are possible, but false negatives are not, and the target false positive rate, denoted throughout this paper by ϵ, can be controlled and has a direct impact on the size of the filter.

Figure 2 shows an example of a Cuckoo filter that uses two hashes per item and contains 8 buckets each with 4 entries. A Cuckoo Filter has mainly two functions: An $Insert$ function that stores items in the filter, and a $Lookup$ function that checks whether an item exists in the filter. We describe the operations of these two functions in Algorithms 1 & 2 [11].

For the $Insert$ operation, explained in Algorithm 1, Cuckoo Filters store a fingerprint f of each item x, as opposed to

storing the item itself. For this, each item is first hashed into a constant-sized fingerprint (step 1). This fingerprint is then stored in the filter as follows. The algorithm checks if there is an empty entry in one of the two buckets indexed by i_1 and i_2 (steps 2 to 5). If an empty entry is found, then f is added to the bucket. Otherwise, one of the two buckets is picked randomly (step 6), and f is swapped with one of the items in the bucket while the victim item (being swapped with f) is relocated to its alternate location, as shown in Algorithm 1. The space cost, in bits, of storing one item in the Cuckoo Filter using the $Insert$ function depends on the target false positive rate ϵ and is given by $(log_2(1/\epsilon) + 2)/\alpha$ where α is the load factor of the filter which defines the maximum filter capacity. Once the maximum feasible, α, is reached, insertions are (non-trivially and increasingly) likely to fail, and hence, the filter must expand in order to store more items [11].

Algorithm 1 $Insert(x)$

1: $f = fingerprint(x)$;
2: $i_1 = hash(x)$;
3: $i_2 = i_1 \oplus hash(f)$;
4: **if** $bucket[i_1]$ or $bucket[i_2]$ has an empty entry **then**
5: add f to that bucket;
return $Done$
// must relocate existing items if no empty entries;
6: $i =$ randomly pick i_1 or i_2;
7: **for** $n = 0; n < MaxNumKicks; n++$ **do**
8: randomly select an entry e from $bucket[i]$;
9: swap f and the fingerprint stored in entry e;
10: $i = i \oplus hash(f)$;
11: **if** $bucket[i]$ has an empty entry **then**
12: add f to $bucket[i]$;
return $Done$
// Hashtable is considered full;
return Failure;

The $Lookup$ operations are highlighted in Algorithm 2. In order to check whether an item x belong to the filter, we only need to compute its fingerprint and its potential locations i_1 and i_2 and then check whether $bucket[i_1]$ or $bucket[i_2]$ contains the fingerprint of x.

Algorithm 2 $Lookup(x)$

1: $f = fingerprint(x)$;
2: $i_1 = hash(x)$;
3: $i_2 = i_1 \oplus hash(f)$;
4: **if** $bucket[i_1]$ or $bucket[i_2]$ has f **then**
return $True$
return False;

In this paper, Cuckoo Filter is used to construct a representation of the spectrum geo-location database as explained in Section IV. What motivated the use of the Cuckoo Filter is that it offers the highest space efficiency among all existing approaches, and is much more efficient than Bloom Filters, especially for very large sets, which is the case of geo-location databases that contain entries corresponding to spectrum availability with a location resolution that can go up to $50\ meters$. Cuckoo Filter enjoys extremely fast $Lookup$ and $Insert$ operations, thus reducing the computation overhead of our proposed scheme substantially as will be seen later. Cuckoo Filter is the building block of our scheme that we present next in Section IV.

IV. Location-Privacy Preservation: The Proposed Cuckoo Filter-Based Scheme

Protecting the location privacy of SUs in database-driven CRNs is a very challenging task, since the users need to provide their locations to the database to be able to learn about spectrum opportunities in their vicinities. There have been some proposed techniques that do protect such privacy in these database-driven CRNs, but not without incurring substantial overhead in terms of communication and/or computation (e.g., [9]). One straightforward and trivial approach, which provides *ideal* location privacy preservation of the users, is to simply send the whole database to the user, and let the user search the database itself to figure out whether spectrum is available in its vicinity. This is of course very costly and unpractical and just mentioned here to show the tradeoffs between having ideal privacy and incurring lots of overhead. Other more efficient approaches, such as the one proposed in [9], do reduce the amount of overhead while still providing a high level of location privacy.

In this paper, we propose an approach that strikes a good balance between achieving high location privacy level and incurring little overhead. The novelty of our proposed scheme, referred to as *Location Privacy in DataBase-driven CRNs* ($LPDB$), lies in the use of the Cuckoo Filter technique, explained in the previous section, to construct a compact (space efficient) representation of the database that can be sent to the SU to figure out about spectrum availability. Our reliance on the Cuckoo filter to represent the spectrum availability reduces the amount of communication overhead substantially without needing to compromise the location privacy of users. In our proposed $LPDB$, instead of sending its location, a user sends its characteristics (e.g., its device type, its antenna type, etc.), as specified by PAWS [3], to the DB which then uses them to retrieve the corresponding entries in all possible locations. The DB then puts these entries in a Cuckoo Filter and sends it back to the user. Upon receiving the new representation of the database (i.e., the Cuckoo Filter representation), the user constructs a query that includes its characteristic information, its location, and one of the possible channels with its associated parameters, and then looks up the received Cuckoo Filter using this constructed query to see whether that channel is available in its current location. As will be shown later, this method does incur substantially small amounts of overhead, thanks to the Cuckoo Filter technique.

An example of parameters that could be included in the response of DB in addition to the time stamp, the location, and the available channels is the transmission power to be

considered when using those channels. User characteristics and DB parameters could be agreed upon beforehand between DB and SUs to make sure that the user queries the Cuckoo Filter with the right parameters.

We provide a simple structure of the geo-location database that we follow in the description of our scheme as shown in Table I. Each row corresponds to a different combination of location pairs ($locX$,$locY$) and channel chn. One location may contain several available channels at the same time. The

TABLE I: Simplistic structure of DB

	$locX$	$locY$	ts	chn	avl	par^1	$\cdots$	par^n
row_1	$locX_1$	$locY_1$	t	chn_1	0	par_1^1	$\cdots$	par_1^n
row_2	$locX_1$	$locY_1$	t	chn_2	1	par_2^1	$\cdots$	par_2^n
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
row_i	$locX_2$	$locY_2$	t	chn_1	1	par_i^1	$\cdots$	par_i^n
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
row_r	$locX_r$	$locY_r$	t	chn_1	0	par_r^1	$\cdots$	par_r^n

$avl = 1$ means channel is available and $avl = 0$ means a channel is being used by the primary user

different steps of the proposed approach are illustrated in Algorithm 3 and are briefly explained as follows: Each user SU_i starts by constructing the query $query_i$ that it will send to DB by including a set of characteristics that are specific to the device querying DB and a time stamp ts. DB then retrieves the entries that correspond to $query_i$ and constructs Cuckoo Filter (which could be done offline). Since DB contains availability status for each channel in each location, the number of entries satisfying $query_i$ will still be huge and one way to further reduce it is to retrieve only the information about available channels and ignore the other ones. DB then concatenates the different data in each row to construct x_j as illustrated in Step 7 and inserts it to Cuckoo Filter. DB then sends Cuckoo Filter to SU which constructs a string y by concatenating its location coordinates with a combination of one channel and its possible transmission parameters and tries to find if y exists in the filter by using the $Lookup$ operation of the *Cuckoo Filter*. The user keeps changing the channel and the associated parameters until it finds the string y in the filter or until the user tries all the channels. If the user finds y in the filter, it can conclude that the channel used to construct y is free and thus can use it. Note that, depending on the false positive rate ϵ of Cuckoo Filter, even if the $Lookup$ operation returns $True$ it doesn't necessarily mean that the specified channel is available. Setting ϵ to be very small makes the probability of having such a scenario very small, as well and limiting the risk of using a busy channel, but this cannot be done without increasing the size of Cuckoo Filter. If after trying out all possible combinations, SU does not find y in Cuckoo Filter, this certainly means that no channel is available in the specified location as the Cuckoo Filter does not incur any false negatives.

When the size of the database is not too large (e.g., when the location resolution is not too small and the area covered by the database is not too large), then this proposed scheme works well (as will be shown later in the evaluation section) by providing ideal privacy with reasonably small amounts of overhead. However, a serious scalability issue may arise when the location resolution is very small (resolution used in the database could be as small as 50 meters) and/or the area covered by the DB is large. In this case, the number of locations, and thus the number of entries in the database, can be very large, and then even after relying on the Cuckoo Filter, the size of the data to be transmitted may still be unpractical/huge. This depends on the desired resolution as well as on the area the DB covers.

Algorithm 3 $LPDB$ Algorithm

1: SU constructs query $query \leftarrow f(char, ts)$;
2: SU queries DB with $query$;
3: DB retrieves $resp$ containing all possible r entries satisfying $query$ each having c columns;
4: DB constructs the Cuckoo Filter $CuckooFilter$;
5: **for** $j = 1, \ldots, r$ **do**
6: **if** $avl_j = 1$ **then**
7: $x_j \leftarrow (locX_j \| locY_j \| ts \| \ldots \| row_j(c))$;
8: DB inserts x_j into $CuckooFilter$: $CuckooFilter.Insert(x_j)$;
9: DB sends $CuckooFilter$ to SU;
10: SU initializes $decision \leftarrow$ Channel is busy
11: **for** `all possible combinations of` ***par*** **do**
12: SU constructs $y \leftarrow (locX \| locY \| ts \| \ldots \| par^n)$;
13: **if** $CuckooFilter.Lookup(y)$ **then**
14: $decision \leftarrow$ Channel is free; **break**;
return $decision$

In this work, we address this scalability issue through the following observation. When the covered area is very large and/or the location resolution is very small, allowing the DB to learn one of the coordinates of the user can drastically reduce the number of entries that DB retrieves and thus considerably reduce the size of the Cuckoo Filter to be transmitted, thus making the approach scalable. Interestingly, in the case of large areas, revealing one coordinate of the user does not make it any easier for the DB to infer the user's location. To illustrate, let's for example assume that the DB covers the entire United States, as shown in Figure 3. Allowing the DB to learn about one coordinate (say the latitude only) means that all what the DB learns is that SU is located somewhere on the blue line that spans the latitude of the whole country. But since the DB does not know the longitude of the SU, then knowing the latitude only is as if nothing is known about the SU's location. Throughout, we refer to this proposed scheme as $LPDB$ with leakage.

It is worth reiterating that when the covered area is not very large, then the size of the Cuckoo Filter is practical and there is no need to reveal one coordinate of the user. In this case, our scheme, $LPDB$, provides ideal privacy without incurring much overhead.

The system regulator can decide about which approach to follow depending on the system constraints; that is, $LPDB$ (for small areas) or $LPDB$ with leakage (for large areas).

V. Evaluation and Analysis

In this section, we evaluate the performance of our proposed scheme and compare it to that of $PriSpectrum$ [9] in terms of: (i) location privacy, (ii) computation overhead, and (iii) communication overhead. But before starting our evaluation analysis, we begin by briefly describing $PriSpectrum$.

$PriSpectrum$ was proposed by Gao et al. [9] to thwart a newly identified attack. In this scheme, the area controlled by the database is modeled as a grid containing multiple cells, and the user's location is determined by the cell in which the user is located. In the identified attack, DB, which is assumed to know the content of the communication between users and the database, can infer the location of SUs from their channel utilization patterns. The observation the authors made is that a SU cannot use a channel unless it is situated outside the coverage area of PU that transmits over that channel, since otherwise SU will interfere with the primary transmission. Now by looking at the different PU channels used by a user over some time period, the DB can narrow down a user's location by intersecting the complements of PUs coverage areas. $PriSpectrum$ was designed to prevent this attack and preserve the privacy of the location information contained in the query of SUs. It uses a blinding factor to hide the indices of the cell that contains the user within the location grid. Now instead of sending indices i and j of the cell in their queries, SU sends two vectors containing i and j with blinding factors that only the user can remove.

A. Location Privacy

We start by evaluating $LPDB$ in terms of location privacy. Our goal in this work is to preserve the location privacy of SUs as stated previously. This means that this information has to be hidden from DB, which usually gets the location from the query sent by the user, or any other entity that can intercept the query and retrieve the location from it. $LPDB$ can achieve an optimal and ideal location privacy since SUs, through this scheme, do not have to include their location in their queries in order to learn about spectrum availability. Furthermore, DB sends to the user all available channels in different locations that comply with the query sent by the user. This prevents it from learning which entry SU picks up and thus its location is unconditionally unknown to DB. This allows our scheme to have better location privacy than $PriSpectrum$ which cannot reach an ideal privacy for SUs since a small number of users may have their location exposed under $PriSpectrum$ [9].

As discussed previously, one way to further reduce the size of the *Cuckoo Filter* is to allow SUs to reveal one of their coordinates. This, as shown in Section V-B, will drastically reduce the size of the filter transmitted by DB at the cost of loosing the ideal location privacy of the users. However, when the coverage area of DB is very large, even revealing one of the coordinates still achieves high location privacy of SUs. Indeed, since our scheme is designed for locating spectrum availability in database-driven CRNs and databases (like those managed by Microsoft and Google) cover an entire nation of the size of the United States, the leaked information is not sufficient to localize the user, yet reduces the lookup complexity substantially. As discussed in the previous section, the example of the United States in Figure 3 shows that our scheme can offer high privacy even when one of the coordinates is revealed. We can see, through this Figure, that all what DB can learn is that SU is located somewhere on the blue line that spans the latitude of the whole country when the latitude coordinate is leaked to the DB.

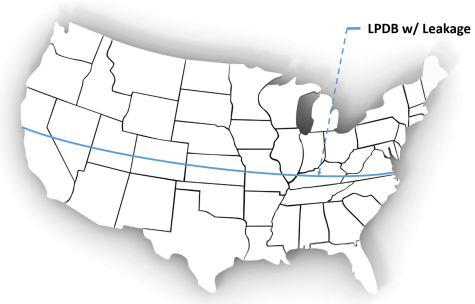


Fig. 3: Location Leakage

B. Communication and Computation Overhead

Now we evaluate the overhead incurred by our scheme. We provide the notations that we use in the rest of this section in the table below:

Here we consider the same setup used in [9] where DB's covered area is modeled as a $\sqrt{m} \times \sqrt{m}$ grid that contains m cells each represented by one location pair ($locX$,$locY$) in the database. We use a large prime p of size 2048 bits for $PriSpectrum$ as in [9]. We use the efficient *Cuckoo Filter* implementation provided in [13] for our performance analysis with a very small false positive rate $\epsilon = 10^{-8}$ and a load factor $\alpha = 0.95$. In addition, since personal/portable $TVBD$ devices of SUs can only transmit on available channels in the frequency bands 512-608 MHz (TV channels 21-36) and 614-698 MHz (TV channels 38-51), this means that users can only access 31 white-space TV band channels in a dynamic spectrum access manner [14]. Therefore, in our evaluation we set s to be equal to 31.

We also ran an experiment to learn what a realistic value of ϱ might be, where again ϱ represents the percentage (averaged over time and space) of channels that are available. We used the Microsoft online white spaces database application [15] to identify and measure the percentage of available channels by monitoring 8 different US locations (Portland, San Faransico, Houston, Miami, Seattle, Boston, New York and Salt Lake City) for two days with an interval between successive measurements of 3 hours. Our measurements show that ϱ is about 6.8%.

TABLE II: Communication and computation overhead of proposed and existent schemes

Scheme	Communication	Computation	
		DB	***SU***
LPDB w/ leakage	$query + \varrho \cdot s \cdot \sqrt{m} \cdot (log_2(1/\epsilon) + 2)/\alpha$	$\varrho \cdot s \cdot \sqrt{m} \cdot insert$	$s \cdot lookup$
LPDB w/o leakage	$query + \varrho \cdot s \cdot m \cdot (log_2(1/\epsilon) + 2)/\alpha$	$\varrho \cdot s \cdot m \cdot insert$	$s \cdot lookup$
PriSpectrum	$(2\sqrt{m} + 3)\lceil \log p \rceil$	$\mathcal{O}(m)$	$4\sqrt{m} \cdot Mulp$

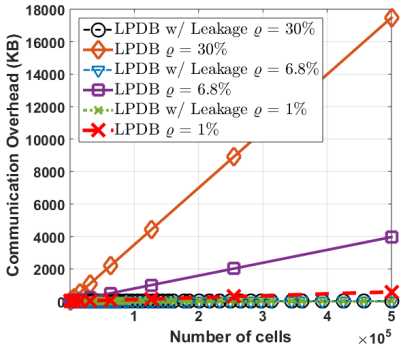
Variables: $insert$ and $lookup$ denote the cost of one $Insert$ and $lookup$ operations in the Cuckoo Filter. $Mulp$ is a modular multiplication operation over modulus p.

m	number of cells of DB coverage area
ϱ	percentage of DB entries with available channels
ϵ	target false positive rate in $Cuckoo\ Filter$
α	load factor ($0 \leq \alpha \leq 1$) in $Cuckoo\ Filter$
s	number of tv channels
p	large prime used in the blinding factor of $PriSpectrum$

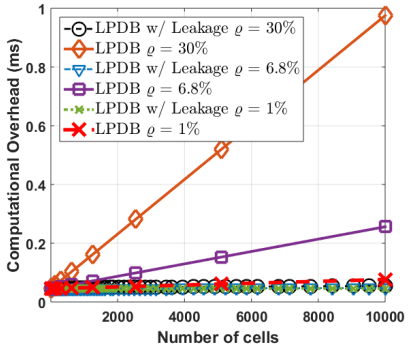
Communication Overhead: We first study the communication overhead of our scheme and we compare it again against $PriSpectrum$. We provide the overhead of both schemes in Table II. For $LPDB$ we provide two expressions of the overhead with respect to two scenarios: first, when one of the coordinates is leaked by the user and second, when there is no leakage. In both scenarios the data transmitted consists basically of the query sent by an SU, $query$, and the response of DB to that query. The size of the response generated by DB depends on the number of entries in the database that satisfy $query$ and on the space needed to store each of these entities in the *Cuckoo Filter*. The number of entries for $LPDB$ is given by $\varrho \cdot s \cdot m$ and reduces to $\varrho \cdot s \cdot \sqrt{m}$ when one of the coordinates is revealed by the user. $s \cdot m$ and $s \cdot \sqrt{m}$ provide the number of entries in DB that satisfy the query of SU for both scenarios. ϱ gives the percentage of those entries with available channels.

Computational Overhead: We also investigate the efficiency of our proposed scheme in terms of its computational overhead. We evaluate the computation required at DB and SU sides separately, as shown in Table II. Again we provide two estimated costs for both scenarios of $LPDB$. The computation of DB is given in terms of the number of insertions it has to perform into Cuckoo Filter. This depends on the number of DB entries that comply with $query$ considering only the available channels. When there is no leakage, this number is equal to $\varrho \cdot s \cdot m$ and when there is leakage of one of the coordinates, the number becomes $\varrho \cdot s \cdot \sqrt{m}$.

For the computation cost in the SU side, $LPDB$'s overhead depends solely on the number of possible channels, s, and the cost of one $Lookup$ operation, $lookup$, for both scenarios, as shown in Table II. One of the reasons that motivated our use of the Cuckoo filter, as we mentioned earlier, is that it is characterized by an extremely fast $Lookup$ operation. This allows the users to check whether a specific combination, y, exists in the filter, i.e. whether channel is available, very efficiently. $LPDB$'s overhead does not depend on the size of the database since any lookup query to Cuckoo Filter always reads a fixed number of buckets (at most two) [11], which makes our scheme more scalable than $PriSpectrum$ in terms of computation when the size of DB increases.



(a) Communication overhead



(b) Computational Overhead

Fig. 4: Performance Comparison

1) Impact of varying ϱ: We also study the impact of ϱ on the overhead incurred by our scheme for both scenarios: with and without leakage. For this, we plot in Figure 4 the communication and the end-to-end (from SU to DB) computation overheads, using the expressions established in Table II, as a function of the number of cells m.

As shown in the Figure, both overheads behave similarly in the way that decreasing ϱ when one of the coordinates is revealed doesn't impact much our scheme. $LPDBs\ w/\ Leakage$ have the smallest overhead compared to the case where no leakage is allowed. In the other hand, decreasing this parameter drastically reduces the overhead of $LPDB$ and even makes it comparable to $LPDBs\ w/\ Leakage$ in terms of communication and computation. This means that in the case where only 1% or less of DB entries have available channels, there is no need to reveal one of the coordinates to reduce the overhead.

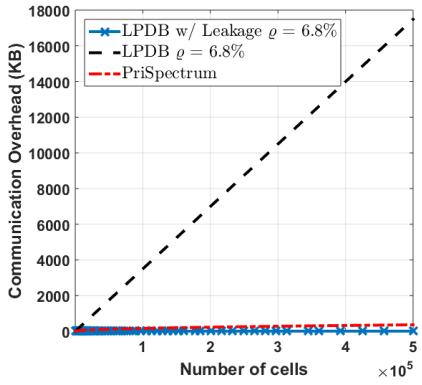
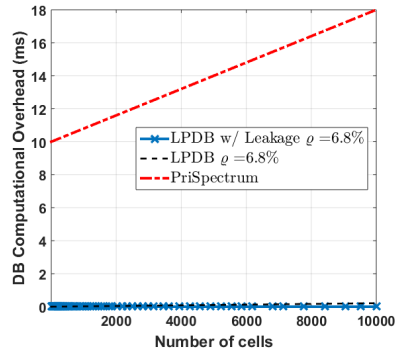
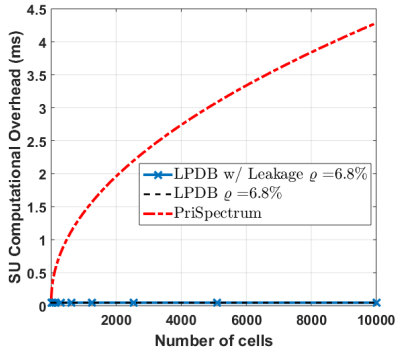


Fig. 5: DB Computational Overhead



(a) DB Computational Overhead



(b) SU Computational Overhead

Fig. 6: Computation Comparison

2) Comparison with PriSpectrum: We now compare the performance of our scheme to that of $PriSpectrum$. We first compare the communication overhead incurred by the different schemes. For this, we plot in Figure 5 the expressions in Table II as a function of the number of cells m.

As shown in the Figure and as expected, $LPDB$ is clearly more expensive than $PriSpectrum$ in terms of communication even when ϱ, determined experimentally, is equal to 6.8%. However, revealing one of the coordinates brings a huge gain and makes our scheme even better than $PriSpectrum$, yet without compromising the location privacy.

We also compare the computation overhead incurred at SU and DB sides for the different schemes as shown in Figure 6. Our scheme is much more efficient than $PriSpectrum$ in both scenarios even for $\varrho = 6.8\%$ and at both DB and SU sides as in Figure 6(a) & 6(b). The gap keeps increasing considerably as the number of cells (i.e., the size of DB) increases. This is due to the fact that $PriSpectrum$'s cost is dominated by an increasing number of modular multiplications which are very expensive compared to the $Insert$ and $Lookup$ operations of the Cuckoo filter.

VI. Conclusion

In this paper, we proposed an efficient scheme, called $LPDB$, that aims to preserve the location privacy of SUs in database-driven CRNs. It uses the concept of Cuckoo Filter to transmit the content of the geo-location database to the user that can query the filter to check whether a specific channel is available in its vicinity. This technique offers an ideal or very high location privacy to SUs and is very efficient especially in terms of computational overhead.

Acknowledgment

This work was supported in part by the US National Science Foundation under NSF award CNS-1162296.

References

[1] "Spectrum policy task force report," Federal Communications Commission, Tech. Rep. ET Docket No.02-135, 2002.
[2] W. Wang and Q. Zhang, *Location Privacy Preservation in Cognitive Radio Networks*. Springer, 2014.
[3] L. Zhu, V. Chen, J. Malyar, S. Das, and P. McCann, "Protocol to access white-space (paws) databases," 2015.
[4] S. B. Wicker, "The loss of location privacy in the cellular age," *Communications of the ACM*, vol. 55, no. 8, pp. 60–68, 2012.
[5] S. Li, H. Zhu, Z. Gao, X. Guan, K. Xing, and X. Shen, "Location privacy preservation in collaborative spectrum sensing," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 729–737.
[6] M. Grissa, A. A. Yavuz, and B. Hamdaoui, "Lpos: Location privacy for optimal sensing in cognitive radio networks," in *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE, 2015.
[7] W. Wang and Q. Zhang, "Privacy-preserving collaborative spectrum sensing with multipleservice providers," *Wireless Communications, IEEE Transactions on*, 2015.
[8] S. Liu, H. Zhu, R. Du, C. Chen, and X. Guan, "Location privacy preserving dynamic spectrum auction in cognitive radio network," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 2013, pp. 256–265.
[9] Z. Gao, H. Zhu, Y. Liu, M. Li, and Z. Cao, "Location privacy in database-driven cognitive radio networks: Attacks and countermeasures," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 2751–2759.
[10] E. Troja and S. Bakiras, "Leveraging p2p interactions for efficient location privacy in database-driven dynamic spectrum access," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2014.
[11] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 75–88.
[12] R. Pagh and F. F. Rodler, "Cuckoo hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, 2004.
[13] "Cuckoo filter implementation," https://github.com/efficient/cuckoofilter.
[14] F. C. Commission, "Electronic code of federal regulations title 47, chapter 1, subchapter a: Part 15-television band devices," 2015.
[15] "Microsoft white spaces database," http://whitespaces-demo.cloudapp.net.