Hitchhiking in Cognitive Radio Networks : Spectrum Sensing Assisted By Cores and Clusters

Ying Dai and Jie Wu
Department of Computer and Information Sciences
Temple University, Philadelphia, PA 19122
Email: {ying.dai, jiewu}@temple.edu

***Abstract*—Spectrum sensing is a key part of cognitive radio networks (CRNs). There have been many works done on improving the accuracy of spectrum sensing results. In this paper, we focus on the phase before spectrum sensing is performed, that is, how to select a channel for sensing. We make use of the fact that nodes located in close geographical locations share similar channel information or sensing results at a given time. Our model enables each node to hitchhike in its spectrum sensing phase, by using other nearby nodes' recent sensing results. Then the total number of channels that a node needs to sense before finding an available one can be reduced. We propose two frameworks for assisting nodes in selecting channels for spectrum sensing. One is the core-only structure, and another is the mixed cluster-core structure. We present the construction of two structures and the distributed spectrum sensing schemes, respectively. Considering the unpredictable primary user (PU) activities and channel dynamics, we also propose an evolution scheme for each node to adjust their selections of cores, in order to gain the most help. We conduct extensive simulations to study the performance of our frameworks.**

Keywords*—*Spectrum sensing, cognitive radio networks, cores, clusters, assistance-based.

I. Introduction

Cognitive radio networks (CRNs) [1] are a promising solution to the channel (spectrum) congestion problem, which creates more spectrum opportunities for nodes in CRNs. The nodes, also called secondary users (SUs), are able to make use of channels when primary users (PUs) on those channels are inactive. The PUs are privileged users, whose data transmissions cannot be interfered by SUs. To ensure that the PUs' transmission is not interfered with, one important phase is the spectrum sensing. Before a node transmits data, it needs to perform spectrum sensing and make sure the channel (spectrum) it selects is not occupied by PUs. In addition, when previously inactive PUs become active during nodes' data transmissions, the affected nodes need to quit that channel and perform spectrum sensing again.

Due to the key importance of spectrum sensing in implementing CRNs, there has been much work done in this field [2]. Many of the work focuses on how to increase the accuracy of spectrum sensing results, how to reduce the spectrum sensing time cost to save more time for data transmission, and so on. Most of the work considers the spectrum sensing phase as well as the data transmission phase after it. However, in CRNs with multiple channels, one aspect that may be neglected is the phase before spectrum sensing happens. That is, choosing which channel to sense first can reduce the total number of channels to sense before finding a channel that is not occupied by PUs. Therefore, it is potentially very beneficial if the node can know which channels have higher probabilities available before sensing them.

One intuitive idea is that each node remembers its own spectrum sensing results from the last time. Next time, when the node has data to transmit, it selects the channels that it will sense first, shown as free from PUs in its previous sensing results. This can be true and useful if the time gap between the two sensing phases is not too large. Otherwise, the channel dynamics in CRNs could cause the previous sensing results to be inaccurate, and could be less helpful for the next spectrum sensing, because of the PUs' unpredictable activities.

Although a node's own previous sensing results may not be helpful for its current spectrum sensing, it can act as a hitchhiker and make use of other nodes' most recent results to benefit its current sensing. Now the problem is how to choose nodes for hitchhiking. Consider the fact that when PUs become active, nodes in their transmission area are all affected. This means that, nodes in the close geographical locations are very likely to face the similar channel availabilities. Therefore, combining both the time and location dimensions, a node can hitchhike in its spectrum sensing phase by using other nearby nodes' most recent sensing results. However, another problem exists here: if every node needs to update the channel sensing order based on every other neighbor node, the overhead during the information exchange could be harmful to the network performance, e.g., an increment in the delay. An efficient information collection and distribution scheme regarding the spectrum sensing results is necessary.

In our paper, we focus on the phase before spectrum sensing happens, which is how to select channels for sensing in CRNs. We aim at reducing the total number of channels that a node needs to sense before finding an available one. We propose two frameworks to assist each node in hitchhiking in its spectrum sensing. The first one is the core-only structure assisted, in which each node designates a neighbor or itself as its core, and can gain help from its core during the spectrum sensing phase. Given the dynamic channel situations, we also propose an evolution model for each node to adjust its core selections, and seek the core that can provide the most help. In addition, we consider that, in a sparse network, the number of nodes designating a same core is very small. This results in a core-only structure unable to provide enough assistance for spectrum sensing. Therefore, we provide a 2-layer structure of both clusters and cores, and a 2-layer spectrum sensing scheme to fit the mixed cluster-core structure.

The main contributions of our paper are:

- We improve the spectrum sensing performance by considering the phase before sensing, and propose two frameworks: core-only structure based, and mixed cluster-core structure based. To our best knowledge, this is the first work to apply the core structures in CRNs for assisting distributed spectrum sensing.
- We propose an evolution scheme for each node to find the core that can provide the most help during the spectrum sensing phase. With the evolution scheme, our model can adapt to the dynamic channel availabilities and unpredictable PU activities in CRNs.
- We present the corresponding spectrum sensing scheme with the assistance of the two structures. Each node can gain help from others, and can also provide help to others through the cores and cluster heads. Our schemes only require limited information exchanges among nodes, cores and cluster heads.

The organization of our paper is as follows. In Section II, we discuss the related works; the system model is introduced in Section III; we introduce the framework of core-only structure in Section IV; the core evolution is presented in Section V; we discuss the mixed cluster-core structure in Section VI; we present the performance evaluation in Section VII; finally, we conclude our paper in Section VIII.

II. Related Works

Our related works can be organized into two categories: current cluster applications in CRNs and the cooperative spectrum sensing.

1) Cluster Applications in CRNs: Many recent works have been done on the cluster structures in CRNs [3]–[5]. In [3], the cluster structure is used on the allocation of control channels in CRNs. Different channels for control are allocated at various clusters in the network, and the problem is solved by the bipartite graph. Authors in [4] construct clusters according to the event that happened in cognitive radio sensor networks. The detection of an event forces the eligible nodes in the network to form into clusters, and better deliver the message. Our work in [5] proposes a virtual backbone construction, based on the cluster head selections, in CRNs without a common control channel. It relies on the geographical location information, and does not need other global information. The above models apply cluster structures in CRNs, and solves the control channel or data transmission related problems. Our work here applies both core and cluster structures to assist the spectrum sensing in CRNs.

2) Cooperative Spectrum Sensing: Authors in [6] consider, in mobile CRNs, how to solve the problem of uncorrelated users to improve the cooperative spectrum sensing performance. [7] considers a soft combination of the observed energies based on the Neyman-Pearson criterion. An optimization scheme is proposed in [8], which uses a coordinator to make decisions. The above works focus on the spectrum sensing phase, which is based on the collaboration among different secondary users. Their key point lies in the fusion of different sensing results and the throughput optimization. However, our model is about how to select channels for sensing before the spectrum sensing phase, in order to reduce the costs of energy and delay. Our work in [9] considers the pre-phase of spectrum sensing. However, it requires the communication among all users in the same area, which sometimes can cause a large overhead. The two frameworks of core-only and mixed cluster-core structures can efficiently reduce the overhead.

III. System Model

A. Network Environment

We consider a CRN with multiple PUs, which are randomly distributed. The total channel set in the network is M. There are V nodes (SUs), which opportunistically make use of the channels without interfering with PUs. The network model for PUs and SUs are as follows:

1) PU network: The PUs are randomly distributed in the network. Their locations and active patterns are assumed to be unpredictable. Each PU has its own privileged channel in M. When a PU becomes active, it would occupy its channel.

2) SU network: First, we assume that there is a common control channel (CCC) for nodes in CRNs to exchange information. This assumption is for simplicity; our model can also apply to CRNs without a CCC, using the approach in [5], [10].

Every time slot is divided into two parts, the spectrum sensing and data transmission. For a node v ($v \in V$), before transmitting data on a channel m_v ($m_v \in M$), it needs to perform spectrum sensing on m_v, to make sure PUs of m_v are not interfered with. We say that m_v is available on node v, if v's transmission on m_v does not interfere with any PU on m_v. For channels that are free of PUs but occupied by other nodes, v can compete for access equally with other nodes. A channel that is free of both PUs and other nodes is the most ideal, since no or less competition is needed. The goal of the spectrum sensing by v is to find an available channel. It is better if the channel is also free of other nodes, but this is not required. If all the channels are occupied by PUs, v cannot transmit data until one channel becomes available.

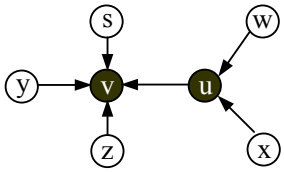
Unlike some spectrum sensing models, our model does not have a centralized decision fusion center. This is because, the PUs are randomly distributed in our model with different transmission ranges. It means that the channel availabilities are not the same in the whole network. Therefore, a centralized decision fusion center is unnecessary.

B. Problem Formulation

We assume that the average cost for sensing one single channel is a constant γ. Then, the spectrum sensing performance is measured by the number of channels that a node v needs to sense, denoted as X_v, until one available channel is found. The objective of our model is to minimize the spectrum sensing cost, which can be formulated as:

$$\text{Min}(X_v\gamma), \qquad \forall v \in V. \tag{1}$$

For a channel to be successfully used for data transmission, it should be free of PUs. Moreover, the node does not want to use a channel that is also chosen by many other nodes, since there would be too much cost spent in the competition for channel access. Therefore, the goal of each node sensing the



u	v	w	x	y	z	s
{1,2}	{1,2,3}	{2,3}	{1}	{1,2}	{3}	{1,2}
4	7	1	1	2	1	2

d_u : {w,x}

d_v : {u,y,s,z,v}

Fig. 1. An Example of Constructing Core Structure.

channels is to find one that must be free from PUs, and better to be free of other nodes may cause interferences to it.

Our focus here is to reduce the number of channels that need to sense for each node to find a channel for data transmission, instead of the sensing technique itself. Therefore, we assume that the sensing is precise enough for each node to detect PUs, and to decide if one channel is occupied by PUs or other nodes. In the following sections, we will discuss our model to reduce X_v in Eq. 1, since γ is a constant here. There is no optimal solution because of the fact that the PUs' activities are dynamic and cannot be pre-known. We give our heuristic solutions, while bringing limited extra cost.

IV. Core-Only Structure

In this section, we will first describe the formation of the core structure. Then, how the cores assist the spectrum sensing scheme is proposed.

A. Core Construction

Our distributed core construction approach is designation-based and only requires one-hop information. The process overview is described below:

1) Each node v exchanges its current available channel set M_v with nodes in its neighbor set N_v;
2) Based on the gathered information in Step 1), node v calculates its weight, which is related to the size of both its available channel set M_v and neighbor set N_v;
3) Node v exchange its weight information with nodes in its neighbor set N_v, and designate the one neighbor with the highest weight (can be v itself), to be its core.

To present the above process in detail, first, we give the definition of weight for each node, which acts as a basis for later core selection.

***Definition** 1:* For a node v, the weight of it is defined as $W_v = \sum_u |M_u \cap M_v|$, $\forall u \in N_v$.

$M_u \cap M_v$ is the intersection of nodes u and v's available channel sets. Based on the above definition, the nodes with more neighbors, and more common available channels with their neighbors, have a larger weight.

Having the weight defined, the core definition is:

***Definition** 2:* c_v is the core of node v, if and only if $c_v \in N_v$ and $W_{c_v} \geq W_u$, $\forall u \in N_v \cup \{v\}$.

After each node exchanges its weight with all its neighbors, then every node can designate the core node from its own perspective. Of course, for a node v, it can have its own core, and can also be selected by some of its neighbors as their core. The set of those nodes that select v as their core are denoted as D_v. In other words, if $u \in D_v$, then $c_u = v$. We say that D_v contains members of core v.

Algorithm 1 Find c_v and D_v for node v.

Input: N_v, $\{W_u, \forall u \in N_v \cup \{v\}\}$;
Output: c_v, core of v; D_v, set of nodes selecting v as core;
1. $Marked_v = false$, $D_v = null$, $c_v = null$;
2. Find $max(W_u)$, $\forall u \in N_v \cup \{v\}$;
3. c_v = the node has $max(W_u)$;
4. v sends its selecting decision to c_v;
5. $Marked_v = true$;
6. **for** every $u \in N_v \cup \{v\}$ **do**
7. **if** $Marked_u = false$ **then**
8. Wait until $Marked_u = true$;
9. **if** $c_u = v$ **then**
10. $D_v = D_v \cup \{u\}$;
11. **return** c_v, D_v;

The algorithm for a node v to get c_v and D_v, is given in Algorithm 1. The exchanged information, including the core selecting decision, is sent through the CCC. There is no order constraint, which means nodes can run Algorithm 1 in parallel. For node u, c_u is the core of u, which has the maximum weight among all nodes in N_v. It also scans its neighbors, and put the nodes that select v as their core in D_v. Node v ends the algorithm until all the nodes in N_v have selected their cores. If node v does not have any neighbor, it would designate itself, which means $D_v = \{v\}$ and $c_v = v$. Every node is covered by a core structure after running Algorithm 1.

An example of the above core construction is shown in Fig. 1. There are 7 nodes and neighbor nodes are connected. The total channel set M is $\{1, 2, 3\}$. The table in Fig. 1 lists the available channel set of each node on the second row and its corresponding weight on the third row, calculated according to Definition 1. There are two cores here, which are v and u, marked in black. The node set that designates v and u as the cores are shown as d_u and d_v in Fig. 1.

B. Core Size Constraint

From the above core construction, it may result in too many cores in the network, and each core only has a very small number of members. This could reduce the efficiency of our core structure when assisting the spectrum sensing, as introduced in the next part. Moreover, the number of members for each core cannot be too large. Otherwise, the overhead of the communication among each core and its members would be too much. Therefore, for a core v, we add the size constraints S_{min} and S_{max}, where S_{min} and S_{max} are minimum and maximum size of D_v. For node v, if the core c_v chosen by Algorithm 1 does not fit the size constraint, the selected c_v would be removed from the original neighbor set $N_v \cup \{v\}$. A new core would be selected from the rest of the nodes in N_u,

which satisfies the size constraints. If no nodes in N_v are left, or v itself is chosen as its own core.

C. Spectrum Sensing With Cores

After the core structure of the network is constructed, every node will gain help from its core for spectrum sensing. This basic idea is that, when a node has data to transmit, it will ask its core about which channel to sense so that the total spectrum sensing cost is reduced. The core node needs to maintain such information to give assistance when help requests are received. We will discuss our scheme from the node side and the core side.

On the node side: When a node has data to transmit, the works that it needs to do are divided into four categories, which contain the four actions:

1) *Pull*: pull the latest channel list from the core, which sorts channels according to their recent updated information;
2) *Sense*: sense the channels using the given order in the list, until one available channel is found;
3) *Transmit*: perform data transmission on the selected channel, and repeat *sense* again if the current channel becomes unavailable because of PUs;
4) *Push*: After data transmission is completed, push the updated channel situations to the core, along with the updated time. The information contains all the channels the node sensed during the above phases.

If there are multiple nodes sending requests to their common core at the same time, the interfering nodes would back off, and send again until receiving the list from the core. When a node has a certain amount of data to transmit, the *pull* and *push* only happens at the beginning and the end of the data transmission. The frequency of exchanging information between a node and its core is changeable. The influences of different settings are studied by the simulations in Section VII.

It is possible that a node can also be a core. For a node v, if $c_v = v$, which means v is its own core, then v can simply sense based on the channel list maintained by itself. If v is a core to other nodes, but $c_v \neq v$, then v provides the channel list for nodes that pick v as the core, but v itself still needs to interact with its own core. For example, in Fig. 1, $c_w = u$ and $c_u = v$. The node w interacts with u to get and update the channel list, while u interacts with v. The two channel lists are independent.

On the core side: When the core receives a help request from its members, it needs to retrieve the current channel list and return it to the request sender. Also, when the core receives channel updates from its members, it needs to update the channel list for use next time.

Suppose $c_u = v$, the interaction between them is shown in Fig. 2. Having clarified the process between a node and its core, the remaining problem is this: How is the channel list maintained and updated by a core? To begin, we need to find a way to calculate the priority of each channel for the core to decide the channel order in the list. First, we associate each channel m, where $m \in M$, with 3-tuple attributes, $< TA(m), TN(m), TP(m) >$:

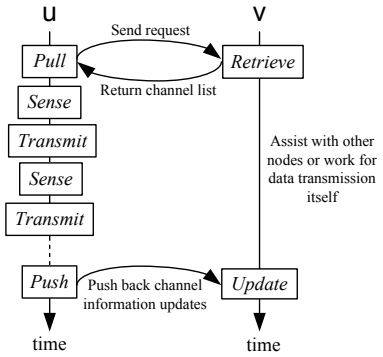


Fig. 2. The interaction example between u and v ($c_u = v$).

- $TA(m)$: The last time when the core receives an update which reports m as free from PUs and other nodes;
- $TN(m)$: The last time when the core receives an update which reports m as occupied by other nodes;
- $TP(m)$: The last time when the core receives an update which reports m as occupied by a PU.

Based on the 3-tuple attribute of each channel, for a core v, it can sort the channels in M, which is shown in Algorithm 2. Every time v receives a help request from any node in D_v, it will run Algorithm 2 and return the sorted channel list, denoted as $L_v(M)$. The sorted results are three categories in $L_v(M)$. The first category contains the channels that are most recently updated as free from both PUs and other nodes. The second category contains the channels that are most recently updated as occupied by other nodes. Although these channels can also be used, the competition cost for channel access with other nodes could potentially be very large. The third category contains the channels that are most recently updated as occupied by PUs. Among channels in the same category, they are sorted according to their recent update times. The one that is most recently updated takes the more prior position in $L_v(M)$, since it is more reliable.

For example, in Fig. 1, if u sends a help request to v, v will run Algorithm 2 to sort the channels. Suppose the 3-tuples associated with the three channels are: channel 1, $< 13, 8, 9 >$; channel 2, $< 11, 10, 7 >$; channel 3, $< 5, 6, 12 >$. Channels 1 and 2 are in the first category, and channel 3 is in the third category. Therefore, the sorted results of the three channels will be $L_v(M) = \{1, 2, 3\}$. Node v will return the $L_v(M)$ to u. After u gets the $L_v(M)$, it first senses channel 1 at time 16, and finds that it is occupied by a PU. Then u records this information and senses channel 2. If channel 2 is available, u will use channel 2 for data transmission. Suppose u finishes its data transmission at time 20, and the channel 2 is free of PUs during the the data transmission, which means u only needs to keep sensing channel 2 in each *Sense* phase. Then u quits channel 2, and channel 2 is available at time 20. During the push phase, u will push the newly updated channel 1 and 2's information to v, along with the update times, 16 and 20. Node v then updates channel 1's 3-tuple as $< 13, 8, 16 >$, and channel 2's 3-tuple as $< 20, 10, 7 >$. Next time, when v receives another help request, the returned $L_v(M)$ is $\{2, 3, 1\}$.

Algorithm 2 Calculate $L_v(M)$ by core v.

Input: $< TA(m), TN(m), TP(m) >, \forall m \in M$;
Output: $L_v(M)$, the sorted channel list by their 3-tuple attributes.

1. $p_c = 0$; // Point to the current index.
2. $p_t = 0$; // Point the tail indexes of previous part.
3. **for** every $m \in M$ **do**
4. Δ_m = Max $\{TA(m), TN(m), TP(m)\}$; // Get the last updated time.
5. **for** i = 1 : 3 **do**
6. **for** every $m \in M$ **do**
7. **if** $(i = 1$ & $\Delta_m = TA(m))$ $\|$ $(i = 2$ & $\Delta_m = TN(m))$ $\|$ $(i = 3$ & $\Delta_m = TP(m))$ **then**
8. Insert m to the position p_c of $L_v(M)$;
9. $p_c = p_c + 1$;
10. Sort $L_v(M)$ between positions [p_t, p_c] based on Δ_m descendingly; // The more recently updated channel is in a more prior position.
11. $p_t = p_c$;
12. **return** $L_v(M)$;

V. Core Evolution

It is not always the case that one node shares similarities with other nodes which initially select the same core. For example, in Fig. 1, nodes $\{u, y, s, z, v\}$ designate v as their core. When u sends a help request to v, v needs to return the $L_v(M)$ to u. Assume that nodes $\{y, s, z, v\}$ have sensed channels before u. Then, based on Algorithm 2, the $L_v(M)$ is updated according to the sensing results provided by $\{y, s, z, v\}$. However, it is possible that the channel available situation around u, and nodes in $\{y, s, z, v\}$, are not very similar, due to the factors on interference range boundaries of PUs and the different geographical terrains. Therefore, it is inappropriate and inefficient for u to designate v as its core, which means it is necessary for u to find a new core that can help more on its spectrum sensing.

A. Basis for Core Update

How does a node know if it designates a wrong core, itself? The answer is to evaluate the help that the core can provide. If the channel list provided by the core does not increase the spectrum sensing performance, the node should designate a new core. We use the estimated number of channels that a node needs to sense until finding a channel free of PUs and other nodes, to represent the spectrum sensing performance. Therefore, we define the basis for core update as:

***Definition** 3:* For a node u and its core v ($v = c_u$), u needs to update c_u if and only if $A_{uv} > A_u$, where A_{uv} is the estimated average number of channels to sense if u receives assistance from v; A_u is the estimated average number of channels to sense if u senses itself u and gains no assistance from others.

However, for a node, it does not know its sensing performance without its core, or A_u, since it always senses based on the channel list provided by the core. Another question arises here: how can a node know if its sensing performance is increased or decreased by the core? Obviously, it is difficult for a node to evaluate the assistance by its core. But, from the opposite direction, a core can evaluate its assistance to its member. Our basic idea is: when a node pushes back the updated channel information, the core evaluates the assistance it could give to the node, under the virtual situation that if the node sends a request now, rather than pushing back its current channel information. The detailed process is:

1) When a node u pushes its newly updated channel information to its core, v ($v = c_u$), v calculates the $L_v^0(M)$ without using the new information from u;
2) Based on the new channel information from u, v calculates the new $L_v^1(M)$, finds the channel in $L_v^1(M)$ that is recently updated as available, and also has the minimum index in $L_v^0(M)$, which equals to the estimation of A_{uv};
3) Then, v calculates A_u, that takes to u to sense randomly until finding a channel free from PUs and other nodes. The current channel availabilities are assumed to be consistent with the channel information in $L_v^1(M)$;
4) Core v compares A_{uv} and A_u. If $A_{uv} > A_u$, then we claim that the assistance from v does not increase the sensing performance of u.

In Step (1), $L_v^0(M)$ is actually the channel list that v would send to u without using any u's new information, if u sends a help request to v now. In Step (2), the new $L_v^1(M)$ denotes the real channel availability currently on node u. In Step (3), based on the information in $L_v^1(M)$, v can know the channel situation on u. Then, A_u is the expected number of sensing if u randomly picks a channel to sense, which can be simply calculated using basic probability theory. In Step (4), v compares A_{uv} and A_u to see if u's sensing performance can be improved by v's assistance, compared to its random-sensing performance.

For example, in Fig. 1, suppose the current 3-tuples on node v associated with the three channels are: channel 1, $< 13, 8, 9 >$; channel 2, $< 11, 10, 7 >$; channel 3, $< 5, 6, 12 >$. Now u sends its updated channel information to v, and the new channel 3-tuples are : channel 1, $< 13, 8, 16 >$; channel 2, $< 20, 10, 7 >$; channel 3, $< 5, 6, 12 >$. Then, v calculates $L_v^0(M) = \{1, 2, 3\}$, and $L_v^1(M) = \{2, 3, 1\}$. The channel that is recently updated as available in $L_v^1(M)$ is channel 2, since 20 = Max $\{20, 10, 7\}$. The index of channel 2 in $L_v^0(M)$ is 2. Therefore, $A_{uv} = 2$.

Since in $L_v^1(M)$, only channel 2 is available, the expected number to find an available channel if u randomly picks a channel to sense is: $A_u = 1 \times \frac{1}{3} + 2 \times \frac{2}{3} \times \frac{1}{2} + 3 \times \frac{2}{3} \times \frac{1}{2} \times 1 = \frac{7}{6}$. In this example, $A_{uv} > A_u$, which means u can have a better performance if it randomly picks a channel for sensing without v's channel list. Therefore, based on Definition 3, u has the need to designate a new core.

B. Core Reselection with Limited Propagation Cost

When a core node finds that one of its members needs to designate a new core, it would send the update notification to that node after the phase *Push*. Then, the notified node needs to find a new core.

One important benefit with the core structure is that it is easier to propagate, compared to traditional cluster structures. Suppose node u has the necessity to find a new c_u; the process is very efficient, as described here:

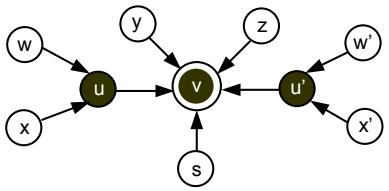


Fig. 3. Cluster head selection from cores.

1) u replaces the input N_u of Algorithm 1 with $N_u - \{v\}$;
2) u reruns Algorithm 1, and the output is the new c_u;
3) u sends a notification to inform the new c_u.

The above process shows that only the node u and the new c_u are affected, and no other nodes are involved. Therefore, the core evolution here takes a very limited cost.

VI. Mixed Cluster-core Structure

Although the core structure is very convenient to construct and update, it is possible that in a sparse network, the core structure cannot assist too much during the spectrum sensing. For example, many nodes are their own cores, whose members are also themselves only. Under these circumstances, we consider having nodes within more than one-hop distances help each other for spectrum sensing. In this section, we propose a mixed structure of clusters and cores, to involve nodes within multiple-hop distances.

A. Cluster-core Construction

To increase the assistance that a core can provide, we select cluster heads from the cores, and build a cluster structure on top of the core structure.

First, for a core u, we use NC_u to denote the set of its neighbor cores. If $v \in NC_u$, then $v \in N_u$ and v is a core with $D_v > 0$. Then, we can define the weight WC_u of a core u, which is later used for cluster head selection:

***Definition** 4:* For a core v, the weight of it is defined as $WC_u = \sum_v |M_u \cap M_v|$, $\forall v \in NC_u$.

The definition for core weight is similar to Definition 1. The difference is that the weight is calculated based on the core neighbor set, NC_u, instead of the original neighbor set, N_u.

The cluster head selection is based on the weight of each core, and applies the classical cluster construction scheme [11]:

1) All cores are initially uncovered;
2) An uncovered core u becomes a cluster head, denoted as h_u, if it has the highest weight (based on Definition 4) among NC_u;
3) The selected cluster heads and their connected 1-hop neighbor cores are marked as covered;
4) Repeat Steps 2 and 3 on all uncovered cores (if any).

The coverage of the cluster algorithm has been proved in [11]. One example is in Fig. 3: we added nodes u', w', and x' compared to the example in Fig. 1. The newly added three nodes have the same channel information with nodes u, w, and x. Therefore, $c_{u'} = v$, $c_{w'} = u'$, and $c_{x'} = u'$. Now there are three cores, u, v, and u'. Based on the cluster selection scheme, core v is the cluster head for u, and u', which means $h_u = v$, $h_{u'} = v$ and h_v is v itself.

TABLE I. Simulation Settings

Number of PUs	10
Number of nodes	$[50, 600]$
Number of channels	$[5, 20]$
PU's transmission range	60
Node's transmission range	$[40, 50]$
Single data task duration	3
Size constraints for cores	$[1, 14]$
Minimum PU active duration	$[5, 15]$
Information exchange frequency for cores	$[1, 3]$
Information exchange frequency for clusters	$[3, 9]$

B. Spectrum Sensing With Cluster-core

Since the cluster-core structure is 2-layered, the spectrum sensing scheme based on it should contain not only the interaction between cores and their members, but also the interaction between cores and their clusters. Here, under the cluster-core structure, the work on the node side remains unchanged as in Section IV-C. The work on the core side needs to change to enable the communication with cluster heads.

One option is similar to the scheme in Section IV-C, which is that every time the core receives a request, it pulls the latest channel list for the cluster head, and pushes the updates of channel information back to the cluster head. However, since a cluster head usually has more members than a core, using the same scheme as in Section IV-C could potentially cause very large overhead when the request to the cluster head becomes more frequent.

Here, we choose another option, which is to have the cluster head periodically push the updated channel information to the cores. This is a tradeoff between reducing the communication overhead and getting the most recently updated channel list, on cluster heads. For a core v and its cluster head h_v, the process works as follows:

1) The cluster head h_v periodically collects information from the cores in the same cluster, labels each channel m, $m \in M$, with its associated 3-tuple attributes, $< TA(m), TN(m), TP(m) >$, and then broadcasts the channel set M with corresponding 3-tuple attributes to all the cores in its cluster;
2) After v gets the information from h_v, v updates its channel information. Next time, when v receives a help request from nodes in D_v, it calculates the channel list $L_v(M)$ based on the newest channel information.

For example, in Fig. 3, node v periodically collects information from u and u', and shares the channel information with them. When node w sends a help request to u, u will return the channel list to w. When node w completes data transmission and pushes the updated channel information to u, this update from w would be picked up by v, during the next time v collects information from u. Also, this update will be shared to u', and assist nodes in $D_{u'}$, which contains v' and w', for their spectrum sensing.

It is not true to claim that one of the core-only and cluster-core structures is always better than the other. In a dense network, each core in the core-only structure may contain enough members, and is able to provide effective assistance to its members. A cluster head on the cores in this situation is unnecessary, and may cause additional communication costs. In a sparse network, it is possible that cores in the core-only

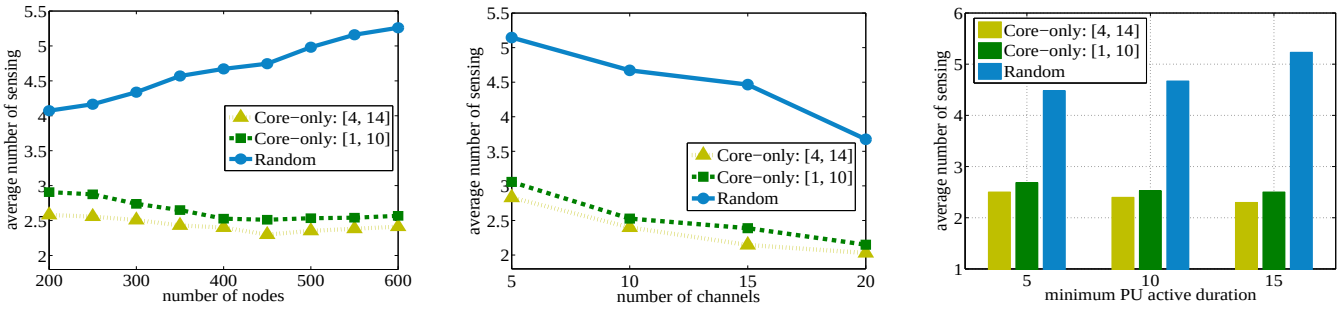


Fig. 4. Comparison of core-only scheme with different size constraints and random sensing scheme.

structure do not have enough channel information to provide assistance. Then a cluster-core structure is a better fit. We will show the performance differences in the simulation.

VII. Performance Evaluation

A. Simulation Settings

We randomly distribute nodes in a 200×200 unit square. The overall simulation settings are listed in Table I. The number of nodes varies for network density control. The network is time-slot based, and each time slot is 1s. We generate data traffic every 10 time slots on a randomly picked set of nodes in the network for broadcasting. For later comparisons about different information exchange frequencies with cores, every single generated data requires 3 time slots to finish transmission by a single node. Each PU occupies one certain channel when it is active. Since each PU's transmission is usually longer than that of other nodes, we set the minimum number of time slots for an active PU longer than the node's average data transmission time. At the beginning of each time slot, the PUs are randomly selected to be active for a generated time duration. If a selected PU is already active, then its status and remaining active time slots would not be changed.

We study the influence of different algorithm parameters, based on our discussions in previous sessions, which are: size constraints of the core-only structures, information exchange frequencies between a node and its core, and comparison between cluster-core structures and core-only structures. Based on the objective function, Eq. 1, the performance we compare is the average number of channels that a node needs to sense in each sensing phase. We also implement a random sensing scheme for better comparison, in which each node randomly picks a channel for sensing every time.

B. Simulation Results

1) Size constraints of core-only structures: The comparison results of core-only structures with different size constraints are presented in Fig. 4. We compare the two size constraints, $[1, 10]$ and $[4, 14]$ with the random sensing scheme. In Fig. 4(a), the average number of channels to sense in the random scheme increases when there are more nodes. In the core-only structures, the number of sensing first decreases and then increases. This is because, at first, when more nodes need to use channels, the channel lists on the cores get updated more frequently and are more accurate. However, when the number of nodes reaches a certain amount, the channel availabilities become much less, and each node senses more channels to find an available one. In Fig. 4(b), the number of sensing in all three lines decreases when there become more total channels. In Fig. 4(c), when a PU's minimum active duration increases, the number of sensing increases for the random scheme. However, both core-only structures decreases. This is because, when a PU occupies its channel for a longer time, the channel availabilities are more static.

In addition, compared to Fig. 4(a) and Fig. 4(b), the number of sensing in Fig. 4(c) changes less obviously, which means the minimum PU active durations have less influences compared to the previous two parameters. Overall, in Fig. 4, the core-only structure with size constraint $[4, 14]$ achieves the best performance, and both core-only structures have devastatingly less number of sensing compared to the random scheme.

2) Information exchange frequencies: We vary the information exchange frequencies between cores and their corresponding members from every 1 time slot to every 3 time slots. The settings of the three network parameters are similar as in the above part. The core size constraints here are $[1, 10]$. The results are shown in Fig 5. The trends of the three frequency settings are similar. And the number of sensing for frequency 3 is the largest among all three, while that for frequency 1 is the least. However, setting the frequency as 1 requires the information exchange with the core every time slot, which is very expensive in a real-life scenario. In addition, the results are more stable when the information exchange frequency is 1. This is because each node can know the instant channel list on its core, which is very accurate, and less influenced by different network parameter settings.

3) Applying cluster-core structures: As proposed in Section VI, the mixed cluster-core structure is applied when facing sparse networks. Therefore, in this part, we set the number of nodes in the network from 50 to 200. The size constraints for selecting cores are $[1, 10]$, and the information exchange frequency between cores and their members is 3. We first show the results of selecting cluster heads from cores in Fig. 6(a). The number of nodes in Fig. 6(a) is 100. We mark partial cores, whose sizes are less than or equal to 2, which means the core structures for those nodes are not very helpful. Then the cluster heads are marked out using green diamonds. The number of cluster heads is 8 in this case.

In Fig 6(b), the performances of the core-only structure and the cluster-core structure are compared. We keep the information exchange frequencies between cluster heads and cores as 6. Since the comparisons of the two schemes by varying other network parameters are similar, we only show

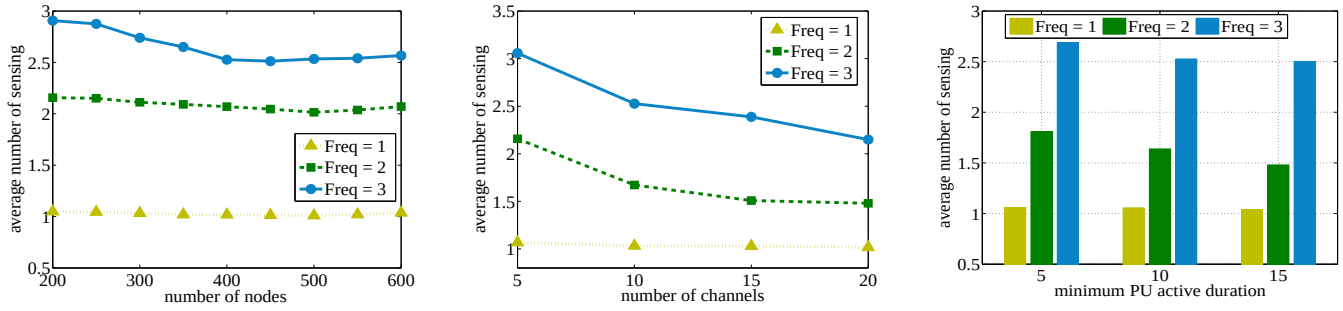


Fig. 5. Comparison of core-only scheme under different information exchange frequencies.

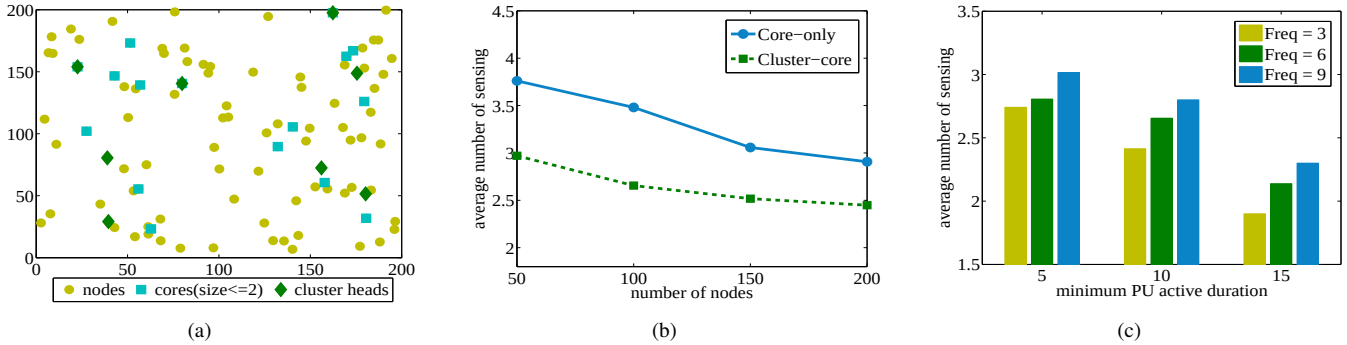


Fig. 6. Construction and performance comparison of cluster-core scheme.

the results of changing network densities here. We vary the number of nodes from 50 to 200. The cluster-core structure requires less number of sensing than the core-only structures. In addition, as the number of nodes increases, the gap between the two schemes reduces.

In Fig 6(c), we vary the frequency of information exchange between cluster heads and cores from every 3 time slots to every 9 time slots. The number of nodes is set to 100, and the minimum PU active duration changes from 5 to 15. From the results, we can see that, the more frequently that the cluster head collects information and sends to the cores, the less sensing there is. Moreover, for all three bars, when the average time that a PU takes a channel is longer, the less number of sensing both schemes need. This is caused for a similar reason as in core-only structures, which is when a PU occupies a channel for a longer time, the channel availabilities are less dynamic, and the cluster-core structure can provide more accurate information.

VIII. Conclusion

In this paper, we propose two frameworks to assist nodes in CRNs during spectrum sensing. One is the core-only structure based, and the other is the cluster-core structure based. We describe the construction of the core-only structure, and the spectrum sensing scheme assisted by it. Also, we consider the evolutions of cores by taking the channel dynamics in CRNs into account. Our core structures take very limited propagation costs. In the cluster-core structure, we describe the application scenarios and how to select cluster heads from current cores. The sensing scheme assisted by the core-cluster structure is discussed. We perform numerous simulations to testify to the performance of our frameworks, and study the influences of different parameter settings.

References

[1] I. Akyildiz, W. Lee, M. Vuran, and S. Mohanty, "Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey," *Computer Networks*, 2006.

[2] T. Yucek and H. Arslan, "A survey of spectrum sensing algorithms for cognitive radio applications," *IEEE Communications Surveys Tutorials*, 2009.

[3] S. Liu, L. Lazos, and M. Krunz, "Cluster-based control channel allocation in opportunistic cognitive radio networks," *IEEE Transactions on Mobile Computing*, 2012.

[4] M. Ozger and O. Akan, "Event-driven spectrum-aware clustering in cognitive radio sensor networks," in *Proceedings of IEEE Infocom*, 2013.

[5] Y. Dai, J. Wu, and C. Xin, "Virtual backbone construction for cognitive radio networks without common control channel," in *Proceedings of IEEE INFOCOM*, 2013.

[6] A. Cacciapuoti, I. Akyildiz, and L. Paura, "Correlation-aware user selection for cooperative spectrum sensing in cognitive radio ad hoc networks," *IEEE Journal on Selected Areas in Communications*, 2012.

[7] J. Ma, G. Zhao, and Y. Li, "Soft combination and detection for cooperative spectrum sensing in cognitive radio networks," *IEEE Transactions on Wireless Communications*, 2008.

[8] R. Fan and H. Jiang, "Optimal multi-channel cooperative sensing in cognitive radio networks," *IEEE Transactions on Wireless Communications*, 2010.

[9] Y. Dai and J. Wu, "Sense in order: Channel selection for sensing in cognitive radio networks," in *Porceedings of CrownCom*, 2013.

[10] M. Abdel-Rahman, H. Rahbari, M. Krunz, and P. Nain, "Fast and secure rendezvous protocols for mitigating control channel dos attacks," in *Proceedings of IEEE INFOCOM*, 2013.

[11] J. Wu and F. Dai, "Virtual backbone construction in manets using adjustable transmission ranges," *IEEE Transactions on Mobile Ccomputing*, 2006.