On Detecting Termination in Cognitive Radio Networks[*,†]

Shantanu Sharma[1] and Awadhesh Kumar Singh[2]

[1]Department of Computer Science, Ben-Gurion University of the Negev, Israel {to_shantanusharma@ieee.org} [‡]

[2]Department of Computer Engineering, National Institute of Technology, Kurukshetra, Haryana, India
{aksinreck@ieee.org}

Abstract

The cognitive radio networks are an emerging wireless communication and computing paradigm. The cognitive radio nodes execute computations on multiple heterogeneous channels in the absence of licensed users (a.k.a. primary users) of those bands. Termination detection is a fundamental and non-trivial problem in distributed systems. In this paper, we propose a termination detection protocol for multi-hop cognitive radio networks where the cognitive radio nodes are allowed to tune to channels that are not currently occupied by primary users and to move to different locations during the protocol execution. The proposed protocol applies credit distribution and aggregation approach and maintains a new kind of logical structure, called the *virtual tree-like structure*. The *virtual tree-like structure* helps in decreasing the latency involved in announcing termination. Unlike conventional tree structures, the *virtual tree-like structure* does not require a specific node to act as the root node that has to stay involved in the computation until termination announcement; hence, the root node may become idle soon after finishing its computation. Also, the protocol is able to detect the presence of licensed users and announce strong or weak termination, whichever is possible.

Keywords: Cognitive radio network, credit distribution and aggregation, heterogeneous channels, termination detection, virtual partitioning and merging, virtual tree-like structure.

[*] **Accepted in International Journal of Network Management (Wiley-IJNM).**

[†] A preliminary version of this paper has appeared in proceeding of the 17^{th} Pacific Rim International Symposium on Dependable Computing 2011 (PRDC) [35].

[‡] This work was carried out when Shantanu Sharma was a student in the Department of Computer Engineering, National Institute of Technology, Kurukshetra, Haryana, India.

1 Introduction

A vast growth of small and portable devices has culminated into the problem of bandwidth scarcity. Hence, it is becoming difficult to provide seamless connectivity while executing various applications, *e.g.*, email, web surfing, gaming, and video conferencing (see Exhibit 10 in [14]). It is also noted that currently allocated spectrums have their significant portions underutilized [1]. The Cognitive Radio Networks (*CRN*s) [16] are a smart solution with a complex network structure to enhance the spectrum utilization.

The termination detection [11, 17] is a fundamental and non-trivial problem in distributed systems because the processors do not have the complete knowledge of all the other processors in the network, and there is no global clock in the distributed computing environment. A solution to the termination detection problem informs termination of the task being executed in the network.

1.1 Cognitive radio networks

A cognitive radio network (see [4, 20, 8, 42, 5, 3, 41]) is a collection of heterogeneous cognitive radio nodes (or processors), called secondary users. The cognitive radio nodes (CRs) have sufficient computing power and power backup to operate on multiple heterogeneous channels (or frequency bands) in the absence of the licensed user(s), termed as primary user(s), of the respective bands. The cognitive radio nodes have the *LEIRA* (learning, efficiency, intelligence, reliability, and adaptively) capability to scan and operate on different channels.

A channel that is not currently occupied by a primary user is called an *available channel*. Any two nodes that are in the transmission range of each other and tuned to a common available channel during an identical time interval, are called *neighboring nodes*. The appearance of a primary user (hereafter, the primary users will be known as PUs) on an available channel is a reason for CRs to switch the channel and to tune to another available channel, because the CRs are not allowed to interrupt the primary users in any case.

The modern networking benchmark, *CRN*, presents many unique challenges in the field of communication as well as computing, such as cognitive capability, reliability, and efficiency. Several challenges of *CRN*s are presented in [8]. Interested readers may refer to [44, 29, 21, 28] for more details on cognitive radio networks.

In this paper, unless otherwise indicated, the words "cognitive radio node," "cognitive radio," "node," and "processor" have the same meaning, and similarly, the words "cognitive radio network," "network," and "system" have been treated as synonyms.

1.2 Termination detection (in cognitive radio networks)

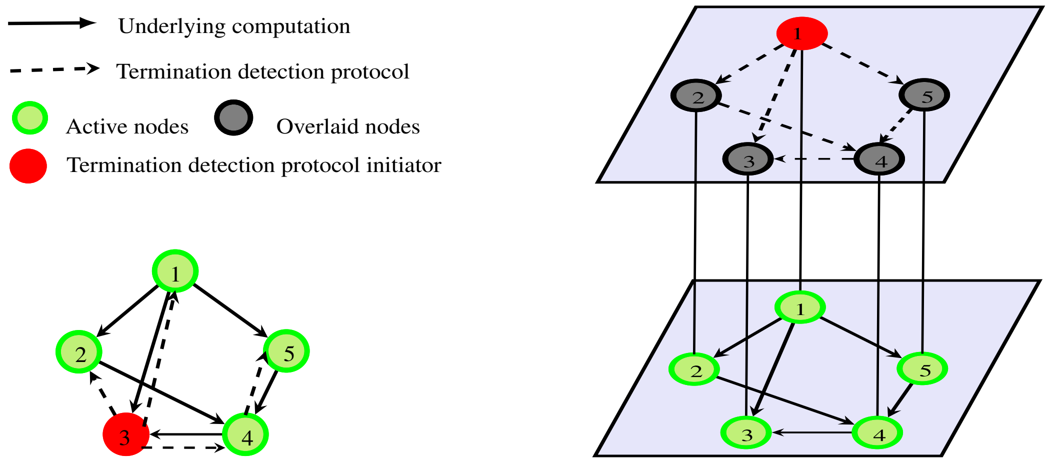
Nowadays, a large number of distributed applications — *e.g.*, mutual exclusion, leader election, checkpointing, global state detection [26] — are executed on portable devices. In general, an application that executes on processors is known as a *normal computation* or an *underlying computation*. A termination detection (TD) protocol [11, 17] is used to announce termination of the normal computation. The termination declaration of a normal computation, when it has indeed terminated — in a group of mobile devices that are geographically distributed and tuned on different channels — is an interesting challenge in cognitive radio networks. Hereafter, we use the word "computation" that refers to the "normal computation."

A node may be in *active* or *passive* state during a computation. The nodes in the active state are called *active nodes*, and the nodes in the passive state are called *passive nodes*. The active nodes execute an assigned computation, and usually, after completion of the computation, they become passive. A passive node can become active on reception of a message from an active node. Hence, it is clear that only active nodes can send messages; however, both the active and passive nodes can receive messages at any time.

Initially, all the nodes are passive in the network. Since only active nodes can send messages, we assume that there exist a passive node that becomes active on reception of a message from outside world, and subsequently it initiates the computation. A computation is said to be terminated if and only if all the nodes are passive and there is no message in-transit. A brief summary about TD protocols can be found in Chapter 7 of [26] and Chapter 9 of [18].

Any termination detection protocol can be initiated in two ways, as follows:

- *Delayed initiation.* The TD protocol is triggered by any node, i, that has been assigned a computation, and the same node i is responsible for the announcement of termination; such an initiation is known as delayed initiation [33]. Here, it is not mandatory that the node i was also the initiator of the computation; refer to Figure 1a, where node 1 initiates the computation and node 3 initiates the termination detection protocol.
- *Concurrent initiation.* In the concurrent initiation, the TD protocol is overlaid on a computation and executes concurrently. Here, the initiator of the computation is also responsible for the announcement of termination; refer to Figure 1b, where the lower part represents the execution of the computation and the upper part represents the


Underlying computation
Termination detection protocol
Active nodes
Overlaid nodes
Termination detection protocol initiator
1
2
3
4
5

(a) Delayed initiation.

(b) Concurrent initiation.

Figure 1: Two ways of termination detection protocol's initiations.

execution of termination detection protocol that is being executed concurrently with the computation; and node 1 is the initiator of both, the computation and the termination detection protocol.

Any termination detection protocol should satisfy the following properties:

- *No false termination detection* (*safety*). The termination of a computation is declared only when the computation has indeed terminated (and only a single designated node can announce termination).
- *Eventual termination detection* (*liveness*). A single (designated) node announces termination within a finite amount of time.

The termination detection in *CRN* is more challenging as compared to the conventional wireless networks because of the following reasons:

- *Network structure and communication links.* The *CRN* is a network of time and space varying channels. Any two neighboring nodes, which must be tuned to an identical available channel, can communicate directly (using a communication link). The appearance of a PU on a channel forces the neighboring CRs to vacate that channel and to tune to another identical available channel. However, finding another identical alternative available channel, for neighboring CRs, is not an easy task due to reasons like topological dynamics and varying capabilities of the nodes [6]. Hence, the communication link endurance during execution of any protocol is hard to guarantee.
- *Reaction to a communication link break.* In classical wireless networks, the nodes operate on a pre-decided channel that provides them communication links. The communication links may break due to node mobility or node failure; hence, a new communication link detection takes place in a highly reactive manner without considering parameters like endurance of the link. On the other hand, in *CRN*, the communication links may also break due to the appearance of a PU leading to *spectrum mobility* that emphasizes on several factors before creating the new communication links [5].
- *Sufficient resources.* The computing nodes in wireless domain suffer from limited resources like bandwidth, memory, and battery power. Thus, several protocols focus on the reduction of the number of messages exchanged to minimize the need of bandwidth, memory, and battery power. However, *CRN*s have sufficient resources, especially temporarily unused spectrums (known as *spectrum holes* [5]) and computing power. Consequently, the focus of research has been shifted to other challenges related to the execution of various applications.
- *No definitive logical structure.* Most of the computing protocol use quasi-stable logical structures, *e.g.*, tree, ring, to leverage the design difficulties. The *CRN*s restrict a direct engagement of such logical structures due to time and space varying channels.

In addition, unlike other ad hoc networks, the CRs show very loose synchronization, poor tolerance to the heterogeneity of mobile devices as well as channels, and an extra cost for searching a new channel (on the appearance of primary users). The presence of these challenges in cognitive radio networks make the design of computing and communication protocols harder.

1.3 Our contribution and outline of the paper

The paper presents a concurrent initiation (see Figure 1b) based Termination detection protocol for Cognitive RAdio Networks, called *T-CRAN*, henceforth. Moreover, our protocol can also be implemented in other dynamic networks, *e.g.*, cellular networks, mobile ad hoc networks (MANETs), vehicular ad hoc networks (VANETs). In this paper, we provide:

1. A credit distribution and aggregation based termination detection protocol for *CRN*, in Section 4, that declares termination of computations despite the presence of PUs. Our protocol recognizes the cognitive radio nodes that lose their single available channel due to the appearance of PUs and are unable to find other available channel.
2. A new logical structure, called *virtual tree-like structure* (Figure 2), where the root node can be passive when it completes its computation, unlike conventional (logical) tree structures, where it is mandatory for the root node to stay in active state till the end of computation; in Section 4.
3. The *T-CRAN* protocol as guarded-actions, in Section 5. Section 6 explains the complete working of the proposed protocol.
4. The analysis of message and time complexities of the proposed protocol, in Appendix A. The correctness proofs of the proposed protocol are given in Appendix B.

1.4 Related work

The termination detection (TD) protocol has been studied extensively in static distributed systems [11, 17, 9, 23, 31, 32, 37, 19]. A detailed classification of TD protocols is given in [26, 30]. However, none of the existing TD protocols for static networks can be implemented straight forwardly in dynamic networks due to frequent topology changes in dynamic networks. Although, some TD protocols [39, 10, 27, 24, 25] exist for sensor networks and mobile ad hoc networks, they can also not be implemented in *CRN*s due to unique challenges of cognitive radio networks, as mentioned in Section 1.2.

A novel algorithm for TD using credit distribution and aggregation was proposed by Mattern [31] and Huang [22, 23]. A similar TD protocol for faulty distributed systems was proposed by Tseng [38]. However, these protocols failed to work in dynamic networks. A TD protocol for mobile cellular networks [39] based on credit distribution and aggregation is proposed that assumes the existence of the mobile switching center (MSS), which provides a centralize support to the mobile nodes.

Johnson and Mittal [24] have tried to reduce the waiting time for termination declaration in dynamic networks. However, they consider the existence of an initiator node until termination declaration. The protocols proposed for dynamic networks [39, 27, 24, 25] have three major limitations: (*i*) they assume the existence of an initiator node until termination declaration; however, the mandatory existence of the initiator node increases the waiting time for the node that has completed its computation earlier than other nodes in the network. Also, the existence of an initiator node until termination declaration is not easy to guarantee in *CRN*s, (*ii*) they work on a single pre-decided channel, whereas, in *CRN*, computations and nodes work on multi-channels, and (*iii*) they consider only node mobility; they do not consider the presence of some special users (like primary users) that also prevent the nodes to work.

In *CRN*, Mittal et al. [46] presents a neighbor discovery protocol with TD; however, they consider only termination of the particular neighbor discovery scheme. The lightweight termination detection of Mittal et al. [46] is not related to our termination detection scheme. Note that in [46], the term “lightweight” has been used to highlight the fact that the number of control messages used in their protocol is minimal.

2 The System Settings

This section outlines the preliminary assumptions about the environment, various types of messages (Table 2), and data structures (Table 3). All the notations used in our protocol are given in Table 1.

Cognitive radio nodes. We consider a cognitive radio network of N cognitive radio nodes ($CR_1, CR_2, \ldots, CR_N$), where each node has a unique identity. However, a group of n CRs executes a single computation, where $n \leq N$, in finite time. The nodes are heterogeneous in terms of their computing capabilities, and they are allowed to move during protocol execution.

Each CR is aware of *global channel set*, *local channel set*, to be defined soon, and also the total number of nodes, N, in the network. Each node has a *scan transceiver* (a transceiver is a transmitter-receiver pair) that is responsible for scanning multiple heterogeneous channels. Such a scanning is beneficial for fast channel switching. However, a transceiver cannot transmit and receive simultaneously.

Communication channels. We divide communication channels into two sets: (*i*) *global channel set* (GCS): a set of all the, g, channels in the network, where $g = |GCS|$; (*ii*) *local channel set* (LCS): a set of, l, available channels[1] at a node, CR_i, where $l_i = |LCS_i|$ and $l_i \leq g$. However, the appearance of PU(s) on all g channels results in the value of the local channel set to be zero, at each node. On the appearance of a PU, a CR is assumed to tune to another available channel, from its LCS, without interrupting the ongoing computation [5], similar to the handoff in mobile cellular networks.

[1] Recall that a channel that is not currently occupied by a primary user is known as an available channel.

CRN	Cognitive radio network		N	The total number of cognitive radio nodes
$\mathcal{CG}$	Communication graph		$\mathcal{IG}$	Interaction graph
$\mathcal{V}$	A set of cognitive radio nodes		$\mathcal{E}$	A set of edges between neighboring nodes
v	A set of cognitive radio nodes in an interaction graph		e	A set of edges in an interaction graph
C_E	Chief executive node (or initiator of the protocol)		n	Number of nodes involved in an identical computation
LCS	Local channel set		GCS	Global channel set
l	Number of available channels at a node		g	Number of the channels in GCS

Table 1: Notations.

A node, CR_i, that does not possess any available channel in its LCS_i (*i.e.*, $l_i = 0$) due to the appearance of PU(s), is called an *affected node*. An affected node is unable to send and receive messages. On the other hand, a node, CR_j, that has at least one available channel in its LCS_j (*i.e.*, $l_j \geq 1$) is called a *non-affected node*. The communication channels are non-FIFO (first-in-first-out) and unreliable. However, the sent messages must be received at the receiver nodes without omissions, duplications, and in the same order as they were sent [12], if the receiver is not an affected or a failed node (see failure model for details).

Network structure. We consider an asynchronous multi-hop cognitive radio network of N independent nodes. We represent the network by a *communication graph*, $\mathcal{CG} = [\mathcal{V}, \mathcal{E}, LCS]$. In the communication graph, $\mathcal{CG}$, $\mathcal{V}$ represents a set of vertices (or processors in the network), $\mathcal{E}$ represents a set of edges where an edge between a pair of neighboring nodes shows a bidirectional, direct, and non-FIFO wireless communication link, and LCS represents the local channel set of each CR.

Further, we define an *interaction graph* of size $n \leq N$ as: $\mathcal{IG} = [v, e]$. In the interaction graph, $\mathcal{IG}$, $v \subseteq \mathcal{V}$ represents a set of CRs that are currently executing an identical computation and $e \subseteq \mathcal{E}$ represents a set of edges where each edge connects any two neighboring nodes, $CR_i, CR_j \in v$, if they are executing an identical computation. Note that we assume different interaction graphs for different computations.

Failure model. We assume that a cognitive radio node may fail in three different ways, as follows:

1. Due to the appearance of a PU and the node has only a single channel in its LCS, then the node is unable to send and receive messages, and such a node is called an affected node.
2. Due to the swift movement of the node that may result in frequent topology change and transient non-interaction of the highly mobile node with other nodes in the network. We call such nodes the *failed nodes*.
3. Crash, *i.e.*, when a node does not possess enough resources, like battery and computing power, it results in permanent failure of the node, and such a node is called a *crashed node*. When a crashed node recovers by users' intervention, it does not possess the knowledge of updated data structures.

In this protocol, we focus on the impact of PUs on the nodes, and after that the recovery of such nodes when PUs disappear. We do not consider any specific approach for recovery of failed nodes. The approach that works in MANET to handle failed nodes is also applicable in *CRN*. In other words, we consider the *failure-recovery model* [2]. Whenever a node recovers, its state may be active or passive. It is possible that the failures occur frequently and, thereafter, the nodes recover soon. Such frequent failures and recoveries are not useful for any practical application; hence, we do not focus on these issues in our protocol. In addition, we assume that the affected and crashed nodes are detected by at least one of the nodes, whose state is active. We also assume that the nodes do not exhibit Byzantine behavior.

Storage media. The termination cannot be detected as the decision variable itself can be corrupted by transient failures leading to a false detection; hence, we store all the data structures in the non-volatile storage (*i.e.*, stable storage). However, a consistent copy of the data is always available in the volatile memory. In the beginning, all the data structures are initialized.

Furthermore, we assume that the cognitive radio nodes have sufficient battery and computing power, and an appropriate routing protocol is in place for message delivery. For ease of presentation and understanding, we consider a single instance of a single computation (*i.e.*, a single interaction graph, $\mathcal{IG}$) in the network; however, the proposed protocol is able to handle multiple instances of multiple computations. We do not specify any neighbor discovery protocol; however, we assume that each CR knows its neighboring nodes using some existing neighbor discovery protocols, *e.g.*, [34].

Types of messages. In our protocol, we use various messages (message details are given in Table 2, and a simplified illustration of messages transmission is shown in Figures 3 and 4) that are classified into *control messages* and *non-control messages*. The control messages have the highest transmission priority, and they are forwarded by (intermediate) passive

Control messages

COMputation message	$COM(C)$	send by CR_i to its active/passive neighboring nodes to distribute the computation.
I am Passive with Credit message	$ImPC(C, b)$	send by CR_i to all the active nodes that had sent credits to CR_i previously including the parent node of CR_i. An $ImPC(C, b)$ message contains credit information, C, and the total number of active child nodes, *b*, of the sender CR_i. The value of *b* is set to 0 if the $ImPC$ is sent to nodes other than the parent nodes.
I am Passive message	$ImP(p)$	send by CR_i to all its child nodes piggybacked with a new parent's, CR_p, information.
AcKnowledgement message	AcK	send by CR_j to CR_i in order to acknowledge credit receipt, if CR_i has surrendered its credit to CR_j.
Acknowledgement of AcK message	$AAcK$	send by CR_i to CR_j if CR_i has received an AcK message from CR_j. The highest priority messages, *i.e.*, AcK and $AAcK$, provide a three way handshake when CR_i surrenders its credit to CR_j. The reception of AcK and $AAcK$ messages are assumed to be atomic (and the delivery time of AcK and $AAcK$ messages is very small, unlike other control messages).
Termination Message	TM	send by the chief executive node, C_E, to all the nodes of the interaction graph to declare termination of the computation.

Non-control messages

Primary user affected Nodes message	PaN	send by CR_i that is neighboring node of CR_j to C_E. This message holds the identity of the affected node, CR_j, and the credit that had sent to CR_j by CR_i or from CR_i to CR_j.
Nodes released by Primary user message	NaP	send by CR_i to C_E and all its neighbors whose states are active. A NaP message holds the identity of CR_i, that was an affected node earlier; however, now CR_i is a non-affected node.

Remarks: (*i*) We use a notation $SEND_i(m, j)$ to show the message transmission of m from CR_i to CR_j.
(*ii*) The message transmission is shown in Figures 3 and 4.

Table 2: Message types used in the *T-CRAN* protocol.

nodes too. It is worth noting that only control messages require communication cost, and non-control messages can be piggybacked on the control or heartbeat messages.

All the control and non-control messages include a tuple $\langle session, initiator\_id \rangle$, that (*i*) avoids the need of a logical clock, which is hard to implement in *CRN*, (*ii*) distinguishes any two messages, (*iii*) distinguishes a message from *stale messages* (a message that is received after the termination declaration, and so belongs to the terminated computation, is known as a stale message, throughout the paper).

Types of data structures. In our protocol, the data structures are divided into two categories: (*i*) at all the cognitive radio nodes, and (*ii*) at the chief executive node, C_E, (a node that is responsible for the announcement of termination). Details of these data structures are given in Table 3.

3 Background

A large number of termination detection (TD) protocols have been introduced for fault-free and faulty distributed systems. They are based on different scheme, *e.g.*, snapshot, credit distribution and aggregation, logical tree, and ring structures [30]. The snapshot based TD protocols require complex data structures to be maintained at each participating node because the amount of information exchanged is usually very high. Consequently, they have a large waiting time for the announcement of termination that is unsuitable for ad hoc networks. On the other hand, the maintenance of logical structures (rings and trees) is a computation intensive task, and it requires frequent exchange of coordination messages to handle dynamic topology in ad hoc environment. Such a high overhead is deterrent in the use of logical structures. Therefore, we consider the credit distribution and aggregation approach to design a TD protocol for *CRN*.

For the sake of completeness and understanding of credit distribution and aggregation based TD protocols, we present the first credit distribution and aggregation based TD protocol, given by Mattern [31]. This protocol assumes the existence of an *oracle* that is responsible for initiation of the computation and termination detection.

Initially, the network has all the nodes in passive state, and the credit at each node is zero. In the beginning of a computation, the oracle, which is supposed to have credit value 1, distributes the credit value among the nodes using *activation messages*. Thus, each activation message holds a credit value, $0 < C < 1$. Now, the oracle waits to receive credits back. Once, the cumulated credit has value one, the oracle announces termination. This protocol uses four rules, as follows:

R1 When a passive node receives an activation message with credit $0 < C < 1$, the node becomes active, holds the credit C, and executes the assigned computation.
R2 When an active node receives an activation message with credit $0 < C < 1$, the credit value, C, is transferred to the oracle.
R3 When an active node, having credit C, sends an activation message, the node sends only $\frac{C}{2}$ credit with the message.
R4 When a node becomes passive, it surrenders its credit to the oracle.

Data structure	Description	Initial value
At all the cognitive radio nodes		
$parent_i$	The parent node of CR_i. It is the first node that sent credit to CR_i since CR_i became active.	0
$hold_i$	The credit received from the parent node of CR_i.	0
$in_i[]$	$in_i[j]$ represents the received credit at CR_i from CR_j such that $j \neq parent_i$.	$\forall j, in_i[j] = \emptyset$
$out_i[]$	$out_i[j]$ represents the credit sent from CR_i to CR_j.	$\forall j, out_i[j] = \emptyset$
$session_i$	The current session of the computation at CR_i.	0
$initiator\_id$	The initiator of the current session of the computation at CR_i.	0
At the chief executive node, C_E		
$PU_{affected}[]$	$PU_{affected}[j]$ represents the identity of an affected node, CR_j.	$\forall j, PU_{affected}[j] = \emptyset$
$C\_PU_{affected}[]$	$C\_PU_{affected}[j]$ represents the credit at an affected node, CR_j.	$\forall j, C\_PU_{affected}[j] = \emptyset$

Table 3: Data structures used in the *T-CRAN* protocol.

In addition, this protocol always satisfies the three requirements: (*i*) at any time, the sum of credits held by nodes, activation messages, and the oracle is 1, (*ii*) when a node is active, it holds a credit $C > 0$, and (*iii*) an activation message, which is in-transit, holds a credit $C > 0$.

The limitations of Mattern's protocol [31] is the existence of a fixed oracle to announce termination that increases the waiting time for termination announcement. Also, this protocol is assumed to work in static networks, where the nodes communicate using fixed communication links. However, unlike Mattern's protocol [31], our protocol is designed for dynamic *CRN*s that have multiple heterogeneous channels. Also, we do not assume the existence of a fixed oracle. Further details of the proposed protocol are presented in the next section.

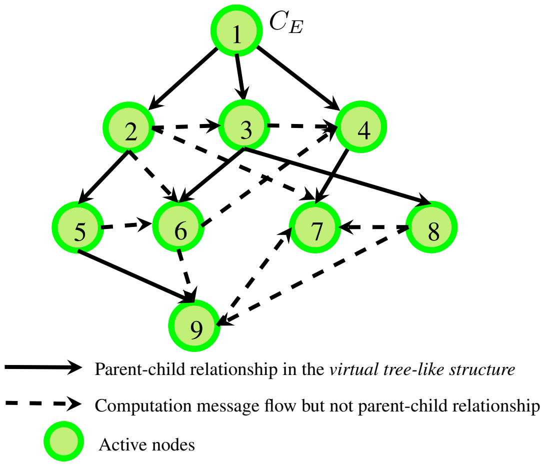

C_E
1
2
3
4
5
6
7
8
9
Parent-child relationship in the *virtual tree-like structure*
Computation message flow but not parent-child relationship
Active nodes

Figure 2: The *virtual tree-like structure*.

4 The *T-CRAN* Protocol

We present our credit distribution and aggregation based termination detection protocol, called *T-CRAN* (see Figures 3 and 4). The initiation of *T-CRAN* protocol is marked by the distribution of a fixed credit value, C, and when a node receives back the same credit value, C, it announces termination.

4.1 High level description of the *T-CRAN* protocol

A node initiates a computation and the *T-CRAN* protocol[2] with a *fixed* credit value, C, and such a node is called the *chief executive node*, C_E. C_E may distribute the computation among its neighboring nodes, called the *child nodes* (of C_E), with non-zero credit values, and C_E becomes a *parent node* of its *child nodes*. The *child nodes* can further distribute the computation like their *parent node*. In this manner, the credit distribution phase creates an illusion of a logical tree among the CRs that are executing an identical computation. We call it the *virtual tree-like structure*, henceforth (see Figure 2). Note that the sum of credits in the network (including the nodes and in-transit messages) must be equal to C.

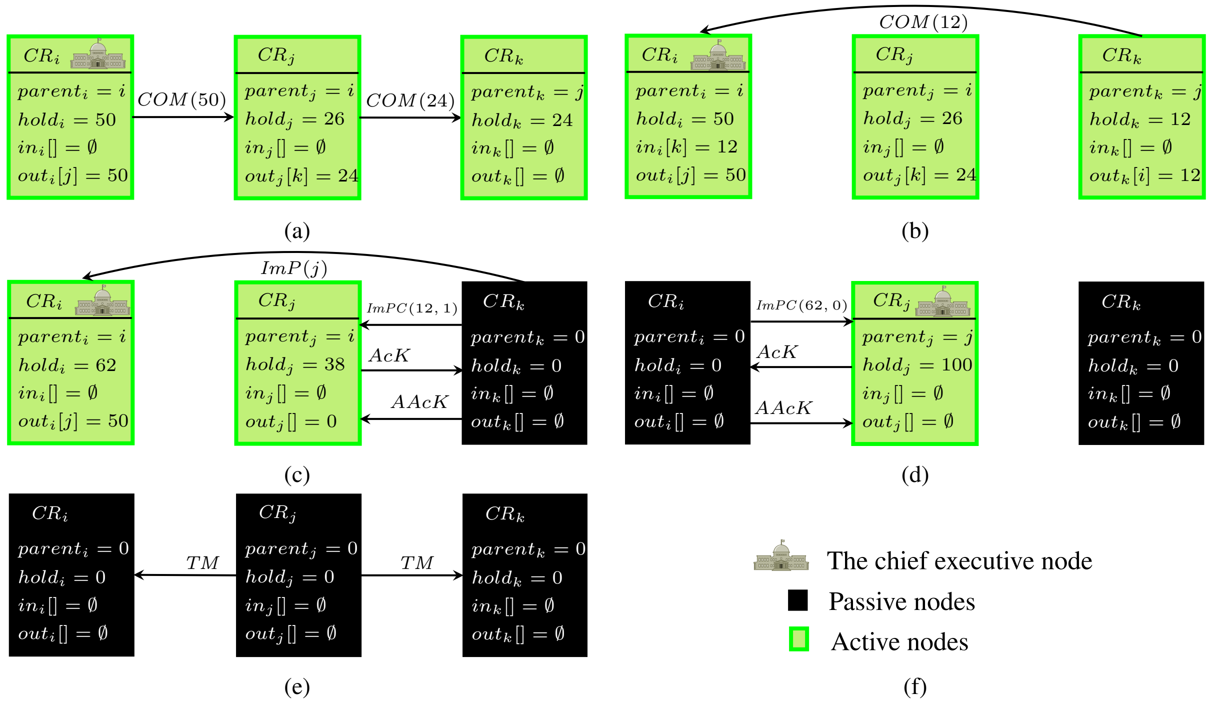
When a node finishes its computation, the node's state becomes passive, and the node surrenders its credit. In the *virtual tree-like structure*, a node surrenders its credit to either (*i*) its parent node if the parent node is active, (*ii*) any node whose state is active and that had sent credit to the node previously, or (*iii*) any node that is executing the same computation, whose state is active.[3] As PUs appear, the neighboring nodes[4] of the affected nodes inform C_E about the affected nodes. Once the affected nodes become non-affected nodes, they inform about their recovery to C_E and their neighboring nodes, whose states are still active. However, C_E waits for a reasonable amount of time[5] for the transition of affected nodes to non-affected nodes before the announcement of termination. Once C_E receives back the credit C (the same amount of credit that was distributed at the initiation of the computation) or a timeout occurs, it announces termination.

[2] Recall that the *T-CRAN* protocol is modeled as another layer on top of the computation; hence, it executes concurrently with the computation.

[3] Such a credit surrender process decreases the waiting time for any node, especially for parent nodes and C_E, if they have finished their computation earlier than their child nodes. Following that it is clear that the parent nodes are also allowed to surrender their credit to any of their child nodes if they are active or to any active neighboring node that is executing the identical computation.

[4] A preference is given to neighboring nodes whose states are active.

[5] The time may be based on the size of the network, message transmission time, and criticality of the computation.



(a) (b) (c) (d) (e) (f)

Figures 3a and 3b show the *credit distribution phase* (STEPS 1-3, STEPS 1-6 are given in Section 4.2). In Figure 3a (STEPS 1-2), CR_i initiates a computation and the *T-CRAN* protocol; hence, CR_i becomes the chief executive node. CR_j receives a *COMputation message* from CR_i with credit 50, holds a credit value ($hold_j = 26$), and further distributes the computation to CR_k. In Figure 3b (STEP 3), CR_k distributes the computation to CR_i, where CR_i does not initialize CR_k as its parent node.
Figures 3c and 3d show the *credit aggregation phase* (STEPS 4-5). In Figure 3c (STEPS 4-5), CR_k becomes passive and surrenders its credit to its parent CR_j (using an *I am Passive with Credit message*, $ImPC(C, z)$). CR_k also sends an *I am Passive message*, $ImP(p)$, to CR_i, where the child node of CR_k (*i.e.*, CR_i) has not terminated its computation. Also, CR_j and CR_k do a three way handshake to ensure a lossless delivery of the credit at CR_j (using an *AcKnowledgement message*, AcK, and an *Acknowledgement of AcK message*, $AAcK$). In Figure 3d, the chief executive node, CR_i, surrenders its credit to CR_j, and CR_i, CR_j do a three way handshake. Now, CR_j becomes the new chief executive node.
Figure 3e shows termination announcement (STEP 6), where the chief executive node, CR_j, announces termination using *Termination Messages*, TM, when CR_j becomes passive.

Figure 3: The termination detection protocol, *T-CRAN*, in the absence of primary users.

Comparison with conventional credit distribution and aggregation protocols. The difference between the conventional credit distribution and aggregation protocols [31, 23, 39, 24] and our protocol lies in the credit surrender process when a node completes its computation.

The conventional credit distribution and aggregation protocols use a logical tree structure, where a *fixed* root node announces termination; thus, it is mandatory that the root node stays in active state till the end of the computation.

Our credit distribution and aggregation based protocol, *T-CRAN*, uses a logical structure, called the *virtual tree-like structure*, where a *non-fixed* C_E may become passive on completion of its computation, and may send its credit, arbitrarily, to one of its child node that becomes a new C_E. The new C_E is responsible for termination declaration, and thus, the existence of an identical C_E till the end of the computation is not desired in the *virtual tree-like structure*.

4.2 Details of the *T-CRAN* Protocol

Now, we first provide details of credit distribution and aggregation phases in the absence of PUs. Later in Section 4.3, we will consider the presence of PUs too.

Credit distribution. In our protocol, initially, all the nodes are in passive state, and a computation starts by a single message from outside world. The credit distribution phase creates a *virtual tree-like structure* (see Figure 2) and consists of three steps (see Figures 3a and 3b), as follows:

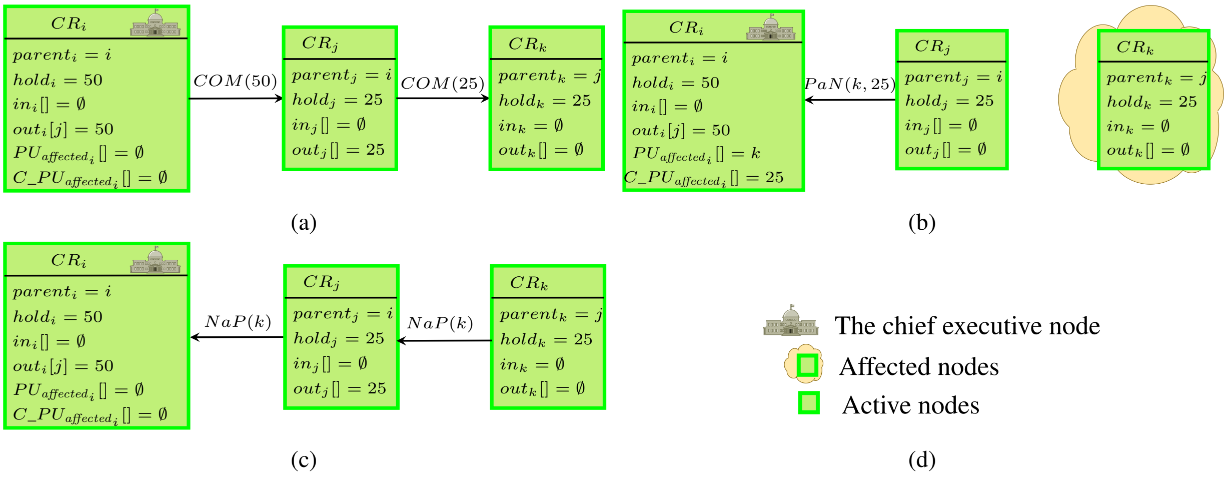


Figure 4a shows the *credit distribution phase* (STEPS 1-2), like in Figure 3a, and CR_i becomes the chief executive node, C_E.
Figure 4b shows the presence of a PU (STEP 7, STEPS 7-8 are given in Section 4.3). CR_k becomes an affected node that does not possess any available channel. However, CR_i and CR_j continues the computation because of some available channels. CR_j sends a *Primary user affected Nodes message* ($PaN(k, 25)$) to C_E to inform CR_k's unavailability in the computation.
Figure 4c shows actions when a PU disappears (STEP 8), and hence, CR_k again becomes a non-affected node. CR_k informs its neighbor, CR_j, and C_E about recovery using *Nodes released by Primary user messages*, NaP.

Figure 4: The termination detection protocol, *T-CRAN*, in the presence of primary users.

STEP 1: Initiation and distribution of a computation and the *T-CRAN* protocol. The computation and the *T-CRAN* protocol is initiated by a CR node, with a fixed credit value C (that is stored in variable $hold_{C_E}$), called the *chief executive node*, C_E. The node C_E may distribute the computation among its q ($q \leq N$) neighboring CRs with non-zero credit values, say $C_1, C_2, \ldots, C_q$, using q different *COMputation messages* ($COM(C_1), COM(C_2), \ldots, COM(C_q)$). Note that once the credit distribution is over, the total credit in the network must be C, *i.e.*, $hold_{C_E} + C_1 + C_2 + \ldots + C_q = C$. Also, we assume that the division of any credit value do not result in a floating point problem, which may result in fractional loss of credits.

STEP 2: Reception of a *COMputation message* at a passive node. The reception of a $COM(C_j)$ at a passive node, say CR_j, causes CR_j to become active and initiate the computation. In addition, CR_j holds credit C_j (in $hold_j$ variable). CR_j may also distribute the computation among neighboring node(s) with non-zero credit values (following the procedure similar to C_E). Secondly, on reception of the first *COMputation message* from any node, say CR_x, at a node, say CR_y, the node CR_y designates the node CR_x as its parent node and becomes a child of CR_x.

STEP 3: Reception of *COMputation messages* at an active node. An active node, CR_j, may receive further *COMputation messages* from the nodes other than its parent, say from CR_k. In such a situation, CR_k does not become the parent node of CR_j. Also, the newly received credit value does not increase credit that CR_j holds (*i.e.*, $hold_j$). However, CR_j performs the corresponding computation and keeps the received credit in an array $in_j[i]$.

Credit aggregation. At the end of the credit distribution phase, the recipients of non-zero credits, become part of an interaction graph, $\mathcal{IG}$. When the nodes complete their computation, the credit aggregation phase is initiated. The credit aggregation phase (see Figures 3c and 3d) consists of two steps, as follows:

STEP 4: Credit surrendering by active nodes. Once CR_j finishes its computation, it surrenders its credits to the corresponding nodes, whose states are active and had sent some credits to CR_j previously. Unlike [31, 23, 39, 24], the *T-CRAN* protocol elevates the credit surrender process at any node by allowing the node to surrender its credits, after the completion of its computation, to the corresponding sender nodes that are active (or vice versa). Moreover, CR_j may surrender its credit to any node whose state is active and that is executing the identical computation, in case, its parent node and all the child nodes have become passive.

The credit surrender process reduces the waiting time for any parent node (or child nodes) that wants to terminate its computation. In fact, C_E can also surrender its credit to any of its child nodes, say CR_x, and CR_x becomes the new C_E. In addition, C_E also transfers its data structures, namely $PU_{affected}[]$ and $C\_PU_{affected}[]$ (see Table 3),

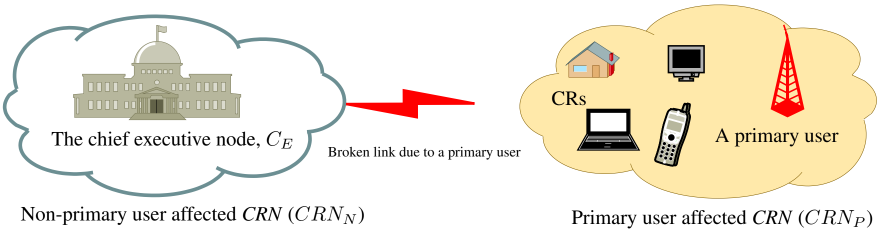

The chief executive node, C_E
Broken link due to a primary user
CRs
A primary user
Non-primary user affected *CRN* (CRN_N)
Primary user affected *CRN* (CRN_P)

Figure 5: The abstract view of a primary user's appearance in cognitive radio networks.

to CR_x at the time of credit surrender.

STEP 5: **Three-way handshake.** In the absence of failures, CR_i acknowledges CR_j (to inform CR_j that CR_i has received the credit back from CR_j) using an *AcKnowledgement message* (AcK). CR_j also acknowledges the reception of the AcK to CR_i using an *Acknowledgement of AcK message* ($AAcK$). Such a mechanism provides a *three-way handshake* and ensures safe delivery of credits. However, the non-reception of an AcK at CR_j, after a timeout, causes CR_j to surrender its credit to another node whose state is active. The three-way handshake can be avoided, if there is a guarantee of message delivery at the receiving node, which is neither an affected node nor a crashed node (using an algorithm suggested in [12]).

Termination declaration. In the beginning, any node may initiate the termination detection protocol and becomes C_E. However, once initiated, any node may take charge as C_E (according to STEP 4), and the new C_E declares the final termination (see Figure 3e). In fact, between initiation and termination of any computation, there could be multiple chief executive charge handovers in the network.

STEP 6: **Termination announcement.** When C_E holds credit C and it has completed its computation, C_E informs all the other nodes in the interaction graph, $\mathcal{IG}$, about the termination of the computation using *Termination Messages* (TM). However, in any case, only a single node (*i.e.*, C_E) can announce termination (when it holds credit C). We also relax the termination detection criteria in Section 4.4.

4.3 Detection of primary user(s)

The appearance of a PU perturbs the working of the CRs (as well as the termination detection protocol) and forces them to tune to another available channel in their LCS. Specifically, the appearance of a PU can be visualized similar to the network partitioning, as it partitions the network into two parts, as explained below:

- Primary user(s) affected *CRN* (CRN_P): It is a part of the network that consists of affected nodes, which cannot send or receive any message.
- Non-primary user(s) affected *CRN* (CRN_N): It consists of all the CRs that have completed their computation and surrendered their credit to the respective senders (of the credit).

In Figure 5, we show these two *CRN*s, namely CRN_P and CRN_N, where, the total credit C is the sum of credits at CRN_P and credit at CRN_N. Further, C_E waits for credits of affected nodes. Intuitively, the appearance of a PU can be interpreted as follows:

- When a PU never ever leaves the channel and the nodes are unable to tune to another available channel, the state of the affected nodes can be interpreted as a crash (that is a permanent failure).
- When a PU persists in the network for a very long time, the computation at the affected nodes can be interpreted as excessively slowed down due to PUs.

However, both the above situations are indistinguishable for other nodes in the network. Thus, we develop an approach that is useful to declare termination even in the presence of primary users. Note that the available hardware approaches —- match filter, energy filter, feature filter, inference temperature management [7] and spectrum sensing techniques [40, 45, 43] — are capable enough to detect the presence of PUs (by any CR). Specifically, the appearance of a PU may turn a non-affected node to an affected node (see STEP 7 and Figure 4b), and when the PU leaves the channel, an affected node becomes a non-affected node (see STEP 8 and Figure 4c), as follows:

STEP 7: **Node failure due to the appearance of PUs.** An affected node, say CR_j, is detected by all the neighboring nodes, whose states are active (and these neighboring nodes may be the parent node or child nodes of CR_j). All the neighboring nodes of CR_j, whose states are active, inform C_E about such a situation using a *Primary user affected Nodes message* (PaN). Each PaN holds the identity of CR_j, and the credit sent (received) to (from) CR_j. On reception of each PaN, C_E enlists CR_j and CR_j's credit in the corresponding data structures (namely, $PU_{affected}[]$ and $C\_PU_{affected}[]$).

Further, all the senders of PaN messages remove credit information about CR_j from their $in[]$ or $out[]$, whichever the case may be. Note that, like any CR, C_E can also detect its affected neighboring nodes, and it does the same as on receiving a PaN and removes them from $in_{C_E}[]$ or $out_{C_E}[]$. However, other PaN messages, about the same affected nodes require the identical processing at C_E.

Moreover, the reception of PaN messages is sufficient to declare termination of the computation, after a reasonable amount of time, in case the following equation 1 holds true:

$$out_{C_E}[] = \emptyset \wedge in_{C_E}[] = \emptyset \wedge (hold_{C_E} + C\_PU_{affected_{C_E}}[] = C) \tag{1}$$

Such a termination detection is called *weak termination* (see Section 4.4 for details, and the equation 1 will be proved in Section B.2).

STEP 8: Recovery of the affected nodes. Once an affected node, say CR_j, becomes a non-affected node, CR_j informs C_E and all its neighboring nodes, whose states are active, using *Nodes released by Primary user messages* (NaP). On reception of a NaP message at C_E, C_E first checks whether the computation has terminated. If not, then C_E informs CR_j about the ongoing computation (with CR_j's credit value). On the other hand, if the neighboring nodes of CR_j have not completed the computation, they hold the credit back in their respective data structures, $in[]$ or $out[]$, whichever the case may be. In this manner, the total credit remains identical as it was at the time of computation initiation.

4.4 Termination declaration

Any kind of node failure (*e.g.*, non-availability of a channel in LCS, mobility of the nodes, and crash, given in Failure Model, Section 2) may lead to either temporary or permanent disconnection of the CRs from the network as well as discontinuity of the computation. However, the failures are quite common in *CRN*. It is not surprising that such a disconnection may leave C_E to starve to collect the necessary credits for termination declaration. Hence, in order to avoid the endless waiting at C_E, C_E may declare either kind of termination, as defined below:

- *Strong termination* infers passive state of all the CRs that were executing an identical computation and the absence of in-transit messages. It is also called *correct and safe termination* announcement.
- *Weak termination* refers to CRN_P and CRN_N, where the state of all the non-affected CRs is passive, and there is no in-transit message. Also, the disappearance of PUs (or recovery from mobility) results in the strong termination. The main advantage of weak termination is the detection of affected nodes and to avoid endless waiting to announce strong termination. However, our approach announces weak termination if equation 1 holds true.

  The significance of weak termination can be figured out by the following example: suppose, we start a leader election (LE) protocol [36] in *CRN* that consists of 100 nodes, initially. During the execution of the LE protocol, say 20 nodes became affected nodes. Thus, it is impractical to wait for strong termination, because a leader may also be elected out of 80 nodes. After recovery from PUs, the remaining 20 nodes may join the network. Such a scenario reduces the waiting time for the announcement of a leader, maintains computation continuity, and enhances resource utilization. However, the weak termination losses its significance, when it is unable to satisfy the safety requirements in the computation, *e.g.*, mutual exclusion and consensus.

Two more criteria for termination in the network are also defined, as follows:

- *Local termination* represents termination of the computation at a node. Further, the node has surrendered its credit to its parent, any neighbor, or any node that is executing the identical computation.
- *Global termination* represents termination of the computation at all the nodes. In other words, the local termination at all the nodes may lead to the global termination, in case, no message is in-transit.

More specifically, two possible outcomes of termination are considerable as: global weak termination or global strong termination.

5 The *T-CRAN* Protocol as Guarded Actions

We specify our *T-CRAN* protocol as guarded actions. A guarded action is written as: $\langle Guard \rangle \rightarrow \langle Action(s) \rangle$. A guard (or predicate) of actions (or rules) is a Boolean expression, and if a guard is true, then all the actions, corresponding to that guard, are executed in an atomic manner. At some point of time more than one guard may be true. The *T-CRAN* protocol as guarded actions is given in Tables 4, 5, and 6. For the sake of simplicity, we assume that all the guards are related to an identical session of a computation.

Notations: $SEND_i(m, j)$: CR_i sends a message m to CR_j, $STATE(i)$: current state of CR_i, *i.e.*, either active or passive, t_e: value of timeout. All the data structures have usual meanings (see Table 3 for details of the data structures).

```
A_1. Computation and protocol initiation. $CR_i$ receives a message $M$ from outside world $\rightarrow$
  $parent_i := i, hold_i := C, in_i[] := \emptyset, out_i[] := \emptyset, session_i := x$

A_2. Distribution of credits. $CR_i$ sends COMputation messages to its $q$ ($q \leq N$) neighboring nodes$\rightarrow$
  $parent_i := i, hold_i := C_i, in_i[] := \emptyset, out_i[1, 2, \ldots, q] := \{C_1, C_2, \ldots, C_q\}, session_i := x$,
  for all $q$ neighboring nodes $SEND_i(COM(C_p), p)$

A_3. Reception of COMputation messages. $CR_j$ receives a COMputation message ($COM(C_j)$) from $CR_i$ $\rightarrow$
  $\llbracket parent_j = \emptyset \wedge STATE(j) = PASSIVE \rightarrow parent_j := i, hold_j := C_j, in_j[] := \emptyset, out_j[] := \emptyset, session_j := x$
  $\llbracket parent_j = CR_a \wedge STATE(j) = ACTIVE \wedge hold_j := C_a \rightarrow parent_j := CR_a, hold_j := C_a, in_j[i] := C_j$,
    $out_j[] := \emptyset, session_j := x$

A_4. Credit surrender. $CR_j$ becomes idle $\rightarrow$
  $\forall k : k \in in_j[] \wedge STATE(k) = ACTIVE \rightarrow SEND_i(ImPC(in_i[k], 0), k)$, Three-wayHandshake(),
  $\llbracket parent_j = j \rightarrow$ (//A case to show when $CR_j$ is the chief executive node)
    $SEND_j(ImP(z), y)$, (//where $y$ are the total nodes in $out_j[] \setminus z$, whose states are active, and $z$ is the new $C_E$ that also belongs to $out_j[]$)
    $SEND_j(ImPC(hold_j, b), z)$, (//$b$ represents the total child nodes of $CR_j$, whose states are active, except $z$)
    Three-wayHandshake(),

  $\llbracket parent_j \neq j \wedge STATE(parent_j) = ACTIVE \rightarrow$ (//A case to show when $CR_j$ is not the chief executive node)
    $SEND_j(ImPC(hold_j, k'), parent_j))$, Three-wayHandshake(),
    $\forall k' : k' \in out_j[] \wedge STATE(k') = ACTIVE \rightarrow$ ($k'$ represents child nodes of $CR_j$ whose states are active)
      $SEND_j(ImP(parent_j), k')$,

  $\llbracket parent_j \neq j \wedge STATE(parent_j) = PASSIVE \wedge (\exists! k' : k' \in out_j[] \vee in_j[]) \wedge STATE(k') = ACTIVE \rightarrow$
   (//A case to show when $CR_j$ is not $C_E$, $parent_j$ is passive, and there is at least one child node of $CR_j$)
    $SEND_j(ImPC(hold_j, k''), k')$, Three-wayHandshake(), $SEND_j(ImP(k'), k'')$, (// $k'' \in out_j[] \setminus k'$)

  $\llbracket parent_j \neq j \wedge STATE(parent_j) = PASSIVE \wedge (\nexists! k' : k' \in out_j[] \vee in_j[]) \wedge STATE(k') = ACTIVE \rightarrow$
   (//A case to show when $CR_j$ is not $C_E$, $parent_j$ is passive, and there is no child node of $CR_j$)
    $SEND_j(ImPC(hold_j, 0), z')$, Three-wayHandshake(), (//where $z'$ is a node that is executing the same computation as $CR_j$ did)

  $parent_j = 0, hold_j := 0, inj[i] := \emptyset, outj[] := \emptyset, session_j := x$

A_5. Reception of $ImPC(C, b)$ messages. $CR_j$ receives $ImPC(C, b)$ from $CR_i$ $\rightarrow$
  $\llbracket parent_i = j \wedge b = 0 \rightarrow hold_j := hold_j + C + in_j[i], out_j[i] := \emptyset, in_j[i] := \emptyset$
  $\llbracket parent_i = j \wedge b \neq 0 \rightarrow hold_j := hold_j + C + in_j[i], out_j[i] := \emptyset, out_j[] = out_j[] \cup out_i[], in_j[i] := \emptyset$
  $\llbracket parent_j = i \rightarrow parent_j := j, hold_j := hold_j + C + in_j[i], out_j[] = out_j[] \cup out_i[], in_j[i] := \emptyset$
  $\llbracket parent_j \neq i \wedge parent_i \neq j \wedge b = 0 \rightarrow hold_j := hold_j + C$
  $SEND_j(AcK, i)$,
  Wait for a $t_e$ or an $AAcK$ from $CR_i$,
  if $t_e \wedge \neg AcK$ then
    send a special message to $C_E$ (This special message avoids multiple credit surrender by $CR_i$ to different nodes.)

A_6. Reception of $ImP(p)$ messages. $CR_j$ receives $ImP(p)$ from $CR_i$ $\rightarrow$
  $\llbracket parent_j = i \rightarrow parent_j := p$
  $\llbracket parent_j \neq i \rightarrow hold_j := hold_j + in_j[i], in_j[i] := \emptyset$

Function Three-wayHandshake(){
  Wait for a $t_e$ or $AcK$,
  if $t_e \wedge \neg AcK$ then $SEND_j(ImPC(hold_j, b), z')$, (//where $z'$ is the new $C_E$)
  else $SEND_j(AAcK, z)$}
```

Table 4: The credit distribution and aggregation phases.

5.1 The credit distribution and aggregation phases

The following actions A_1 through A_6 define the credit distribution and aggregation phases, refer to Table 4.

A_1 **Computation and protocol initiation.** A_1 is executed on reception of a message M from outside world, and then, a node initiates a computation and the *T-CRAN* protocol.

A_2 **Distribution of credits.** In A_2, CR_i distributes the computation among its q ($q \leq N$) neighboring nodes using q different *COMputation messages*. Note that the *T-CRAN* protocol executes concurrently over the computation.

A_3 **Reception of *COMputation messages*.** The first guard shows that CR_j, whose state is passive, receives a *COMputation message* for the first time from CR_i. Hence, CR_i becomes the parent node of CR_j, and CR_j keeps credit, C_j, in variable $hold_j$. The second guard shows that CR_j, whose state is active, receives a *COMputation message* from CR_i. Hence, CR_j does not assign CR_i as its parent, and CR_j keeps credit, C_j, in $in_j[i]$.

A_4 **Credit surrender.** A_4 is executed when a node finishes its computation and consequently becomes passive. The first statement shows that CR_j sends *I am Passive with Credit messages* to all the k nodes whose states are active and they had sent some credits to CR_j.

The first guard becomes true when C_E becomes passive. C_E sends *I am Passive messages* ($ImP(p)$) to all its child nodes with the information of the new C_E. In addition, the old C_E sends an *I am Passive with Credit message* to the new C_E and executes the function *Three-wayHandshake()* to ensure delivery of its credit ($hold_{C_E}$) to the new C_E.

The second guard becomes true when CR_j becomes passive and CR_j is not the chief executive node. CR_j informs all its child nodes about the new parent node using $Imp(p)$ messages. Also, CR_j sends an *I am Passive with Credit message* to its parent node with its credit ($hold_j$) and executes the function *Three-wayHandshake()* to ensure delivery of its credit.

The third guard becomes true when CR_j becomes passive and its parent node is also passive. However, there is at least a node $CR_{k'}$ in $out_j[]$ or $in_j[]$ whose state is active. CR_j sends its credit to $CR_{k'}$ using an *I am Passive with Credit message* and executes the function *Three-wayHandshake()*. Also, CR_j sends $ImP(p)$ messages to all the nodes whose states are active and in $out_j[]$.

The forth guard becomes true when CR_j becomes passive, its parent node is also passive, and there is no node in $out_j[]$ or $in_j[]$ whose state is active. CR_j sends its credit to $CR_{z'}$ using an *I am Passive with Credit message* and executes the function *Three-wayHandshake()*. When $CR_{z'}$ is selected, a priority is given to a node that is executing the same computation as CR_j.

A_5 **Reception of $ImPC(C,b)$ messages.** The reception of *I am Passive with Credit messages* at CR_j is shown in A_5. The first guard becomes true when CR_j receives $ImPC(C,b)$ messages from its child nodes that do not have any child node. The second guard becomes true when CR_j receives $ImPC(C,b)$ messages from its child nodes that have some child nodes. The third guard becomes true when CR_j receives an $ImPC(C,b)$ from its parent node. The forth guard becomes true when CR_j receives $ImPC(C,b)$ messages from any node that is executing the same computation as CR_j is executing. In all the cases, CR_j also sends an AcK to CR_i, and if CR_j does not receive an $AAcK$ from CR_i, it sends a special message to C_E that is used to balance the credit in the system. (The working of this special message is shown in Figure 10).

A_6 **Reception of $ImP(p)$ messages.** The reception of *I am Passive messages* at CR_j is given in A_6. The first guard is related to the arrival of an $ImP(p)$ at a child node CR_j from its parent node, and the second guard is related to the arrival of an $ImP(p)$ from any node other than its parent node.

5.2 The appearance and disappearance of primary users

The following actions B_1 through B_4 are related to the appearance and disappearance of primary users on channels, refer to Table 5.

B_1 **Appearance of a PU.** B_1 becomes true when a PU appears on a channel. The first guard becomes true when CR_i has some channels in its LCS and tunes to one of them. The second guard becomes true when CR_i's LCS is empty; hence, CR_i becomes an affected node. In addition, all the k neighbors, whose states are active, of the affected node CR_i send PaN messages to C_E.

B_2 **Reception of PaN messages.** When C_E receives PaN messages, it places the affected nodes and their credits, which are received with PaN messages, in the respective data structures.

B_3 **Disappearance of a PU.** When a PU leaves the channel, all the affected nodes become non-affected nodes, and they send NaP messages to C_E and all its neighboring nodes, whose states are active.

B_4 **Reception of NaP messages from CR_j.** The reception of a NaP at C_E, from any node, notifies the absence of PU(s). The first guard becomes true when C_E receives NaP messages for the current computation. C_E removes the sender of the NaP (that is CR_j) from its $PU_{affected}[]$ and informs CR_j about the ongoing computation. The second guard becomes true when the neighboring nodes of CR_i receive NaP messages. All the receiver nodes send

Notations: $SEND_i(m, j)$: CR_i sends a message m to CR_j, $STATE(i)$: current state of CR_i, *i.e.*, either active or passive, $Neighbor_i[]$: neighboring nodes of CR_i that are executing the same computation as CR_i. All the data structures have usual meanings (see Table 3 for details of the data structures).

B_1. **Appearance of a PU.** A primary user appears on channels $\rightarrow$
$[]\, LCS_i \neq \emptyset \rightarrow CR_i$ leaves the channel and tune to another available channel
$[]\, LCS_i = \emptyset \rightarrow CR_i$ becomes an affected nodes and is not allowed to send and receive messages,
$\forall k \in Neighbor_i[] \wedge STATE(k) = ACTIVE \rightarrow SEND_k(PaN, C_E)$, where a PaN holds $\langle CR_i, in_k[i], out_k[i] \rangle$,
$in_k[i] = \emptyset, out_k[i] = \emptyset$

B_2. **Reception of PaN messages.** C_E receives a PaN that holds $\langle CR_i, in_k[i], out_k[i] \rangle \rightarrow$
$PU_{affected_{C_E}}[i] := CR_i, C\_PU_{affected_{C_E}}[i] := \langle in_k[i], out_k[i] \rangle$

B_3. **Disappearance of a PU.** A primary users disappear from channels $\rightarrow$
$LCS_i \neq \emptyset$, $SEND_i(NaP, C_E)$, $\forall k \in Neighbor_i[] \wedge STATE(k) = ACTIVE \rightarrow SEND_i(NaP, k)$

B_4. **Reception of NaP messages from CR_j.**
$[]\, i = C_E \wedge session_{C_E} = x \wedge STATE(C_E) = ACTIVE \rightarrow$
$PU_{affected_{C_E}}[j] := \emptyset, C\_PU_{affected_{C_E}}[j] := \emptyset, SEND_{C_E}(m, CR_j)$
$[]\, i \neq C_E \wedge j \in Neighbor_i[] \wedge session_i = x \wedge STATE(i) = ACTIVE \rightarrow SEND_i(m', C_E)$

Table 5: The appearance and disappearance of primary users.

C_1. **Global weak termination.**
$out_{C_E}[] = \emptyset \wedge in_{C_E}[] = \emptyset \wedge (hold_{C_E} + C\_PU_{affected_{C_E}}[] = C) \wedge STATE(C_E) = PASSIVE \rightarrow$
Announce global weak termination

C_2. **Global strong termination.**
$out_{C_E}[] = \emptyset \wedge in_{C_E}[] = \emptyset \wedge C\_PU_{affected_{C_E}}[] = \emptyset \wedge hold_{C_E} = C \wedge STATE(C_E) = PASSIVE \rightarrow$
Announce global strong termination

Table 6: The termination announcement.

a special message m' to C_E that is a request to receive back the credit that they had sent earlier when CR_i was an affected node.

5.3 The termination announcement

The actions C_1 and C_2 represent termination detection in the networks, refer to Table 6.

C_1 **Global weak termination.** C_1 announces the global weak termination when only a single node (that is C_E) has the credit C, which was distributed at the time of computation initiation, in its $hold_{C_E}$ and $C\_PU_{affected}[]$.

C_2 **Global strong termination.** C_2 announces the global strong termination when C_E contains the total credit C, which was distributed at the time of computation initiation, in its $hold_{C_E}$, no credit in $C\_PU_{affected}[]$, and C_E has become passive.

6 The Working of the *T-CRAN* protocol

In Figures 6 and 7, a sample execution of the *T-CRAN* protocol is represented in the absence and presence of a primary user, respectively. The local channel set for every node is given in Table 7, where boldface characters show the currently tuned channel at the respective nodes. Actions A_1 to A_6, actions B_1 to B_4, and actions C_1, C_2 are given in Tables 4, 5, and 6. Also, all the nodes are in the transmission range of each other.

In Figure 6a, cognitive radio node 1 initiates an assigned computation and the *T-CRAN* protocol, hence, behaves as the chief executive node, C_E. Node 1 sends a *COMputation message* ($COM(0.9)$) to node 2, and node 1 holds a credit value 0.1. Further, nodes 2, 3, 4, and 5 also distribute the computation with some credit values and initialize their respective data structures (A_1, A_2, and first guard of A_3). Note that, $in_i[]$ array of all the nodes is empty initially, and the sum of credits at nodes 1-6 equals 1.

Nodes	Local Channel Sets (LCS)
1	$2, 3, \mathbf{5}$
2	$3, \mathbf{5}, 6, 9$
3	$\mathbf{5}$
4	$\mathbf{5}$
5	$\mathbf{5}, 7, 9$
6	$\mathbf{5}, 9$

Table 7: Local channel sets of all the nodes.

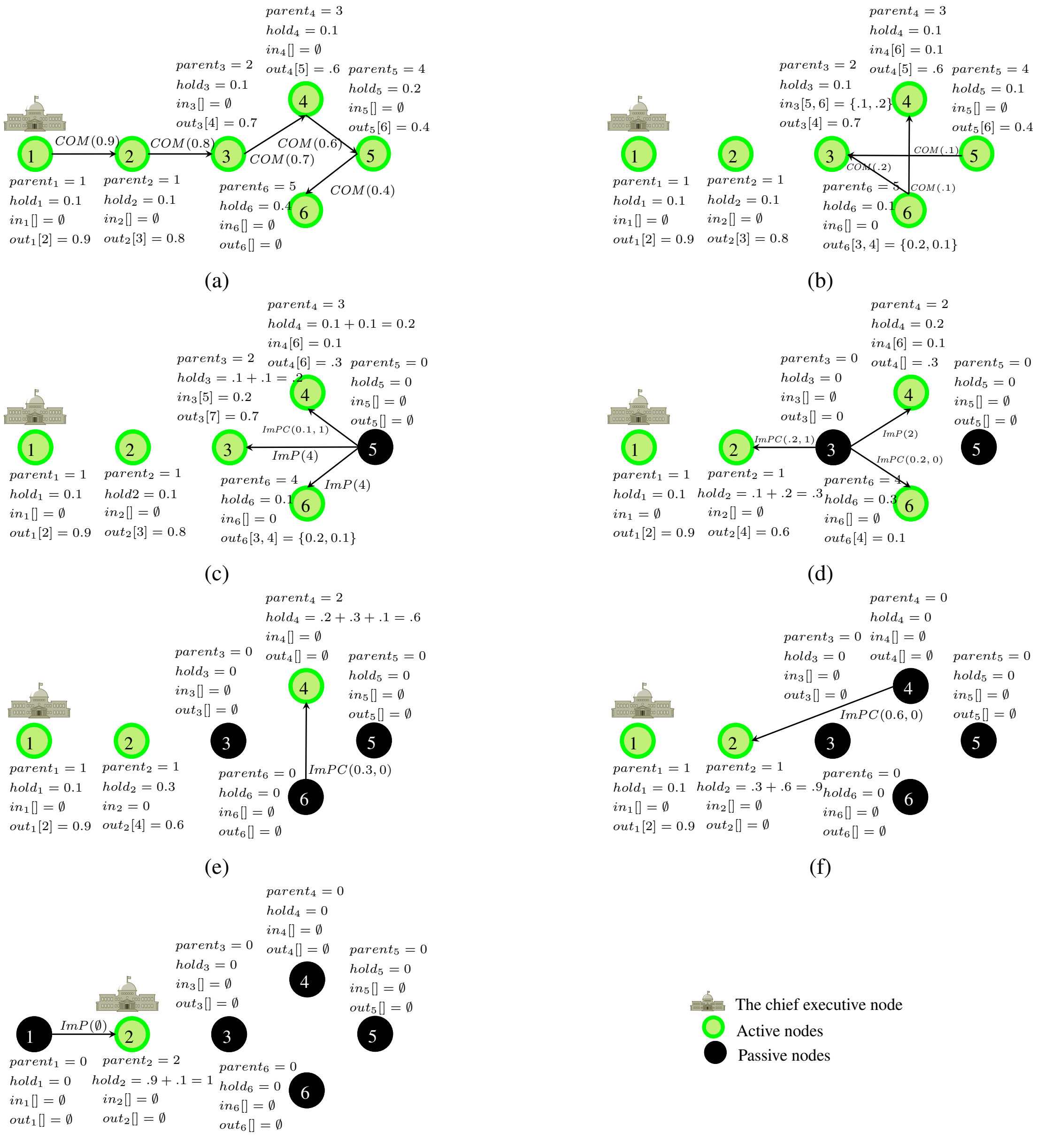


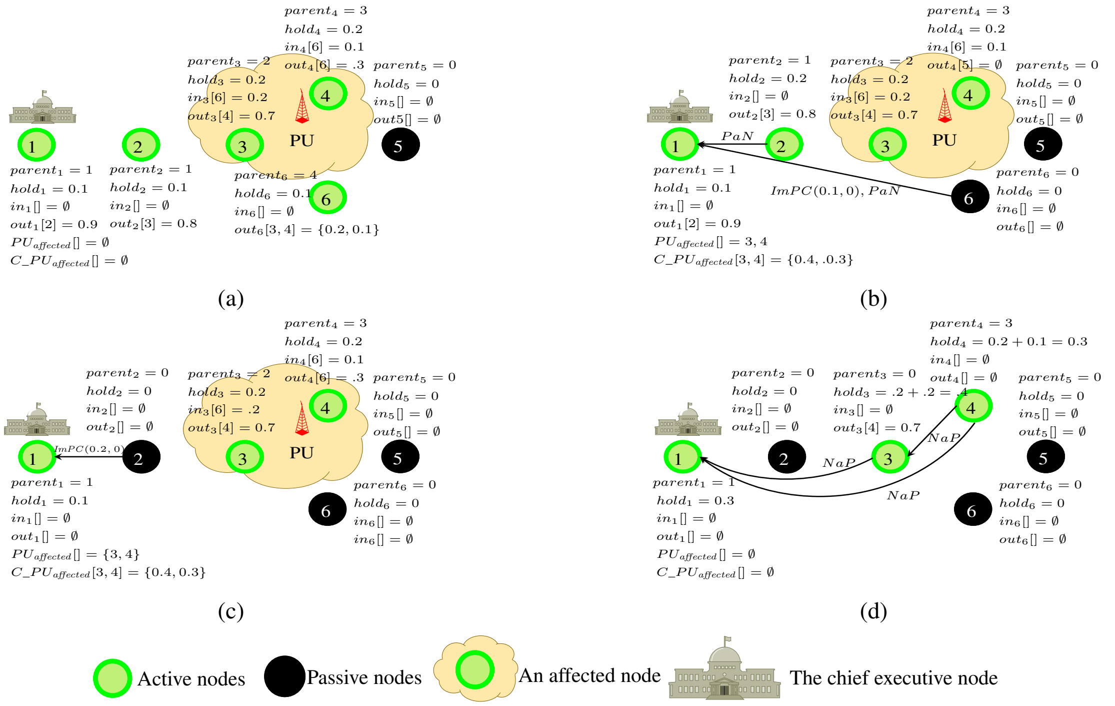
Figure 6: The Working of the *T-CRAN* protocol in the absence of PUs.

In Figure 6b, node 5 sends a *COMputation message* ($COM(0.1)$) to node 3. The reception of $COM(0.1)$ initializes $in_3[5]$ array variable at node 3 and corresponding entry in $out_5[3]$ (second guard of A_3). Also, node 6 sends two *COMputation message*s to nodes 4 and 3.

In Figure 6c, node 5 becomes passive and surrenders its credit (contained in variable $hold_5$) to its parent node 4 via an *I am Passive with Credit message*, $ImPC(0.1, 1)$ (second guard of A_4). In addition, node 5 sends two *I am Passive message*s ($ImP(4)$) to its child nodes 3 and 6. These *I am Passive message*s hold information of the new parent node 4. The reception of $ImPC(0.1, 1)$ at node 4 triggers second guard of A_5. The reception of $ImP(4)$ triggers first guard of A_6 at node 6 so that node 6 assigns node 4 as its parent node, and second guard of A_6 at node 3 so that node 3 holds 0.2 credits. Note that we are not showing the three way handshake for clarity of figures, interested readers can look ahead to Figure 10 to see the working of the three-way handshake.

In Figure 6d, node 3 becomes passive and sends: (*i*) an $ImPC(0.2, 0)$ to node 6 (the first line of A_4), and node 6 holds back 0.3 credits (first guard of A_5), (*ii*) an $ImPC(0.1, 1)$ to its parent node 2 (second guard of A_4), and node 2 holds 0.3 credits now (second guard of A_5), (*iii*) an $ImP(2)$ to node 4 (second guard of A_4), and node 4 assigns node 2 as its parent node (first guard of A_5).

In Figure 6e, node 6 becomes passive and surrenders its credit to node 4 (second guard of A_4). Node 4 holds 0.6 credits now (first guard of A_5). In Figure 6f, node 4 becomes passive and surrenders its credit to node 2 via an $ImPC(0.6, 0)$ (second guard of A_4).



(a)

(b)

(c)

(d)

(e) Notations

Figure 7: The Working of the *T-CRAN* protocol in the presence of PUs.

In Figure 6g, the chief executive node 1 becomes passive and surrenders its credit to node 2 (first guard of A_4), and node 2 now becomes the new chief executive node, holds credit 1 (third guard of A_5). Once the node 2 finishes its computation, node 2 announces the global strong termination (according to C_2).

Figure 7 shows the presence of a PU. Figure 6c, where there is no PU, turns to Figure 7a in the presence of a PU, where nodes 3 and 4 are affected nodes (second guard of B_1). In Figure 7b, nodes 6 and 2 detect affected nodes 4 and 3, and inform node 1 using PaN messages (second guard of B_1), and node 1 places information of the affected nodes 3, 4 in the respective data structures (according to B_2). In addition, node 6 surrenders its credit to node 2 (forth guard of A_4).

In Figure 7c, node 2 becomes passive and surrenders its credit to node 1 (second guard of A_2). Now, node 1 holds the credit value 1, which was distributed at the time of computation's initiation, in $hold_1$ and $C\_PU_{affected}[]$, and this condition is sufficient to announce the global weak termination after a timeout (according to C_1). In Figure 7d, the primary user disappears, and nodes 3, 4 inform node 1 (C_E) and ask about the ongoing computation (according to B_3). Node 1 informs them and deletes their entry from $PU_{affected}$ and $C\_PU_{affected}[]$, and then, the sum of credits at node 1, 3, and 4 equal to 1.

7 Conclusion

A termination detection protocol, *T-CRAN*, for an asynchronous multi-hop cognitive radio networks is presented. The *T-CRAN* protocol is capable enough to work on heterogeneous channels, and it can also handle multiple computations simultaneously. The *T-CRAN* protocol is based on credit distribution and aggregation approach. The proposed protocol uses a new kind of logical structure, called the *virtual tree-like structure*. In the *virtual tree-like structure*, a node may surrender its credit to any node (not necessarily to its parent node) that is executing the identical computation. This credit surrender approach significantly reduces the waiting time to announce termination. Further, it is not mandatory for the initiator of the protocol (*i.e.*, the first root node of the *virtual tree-like structure*) to stay involved until the termination of the computation. Hence, the protocol may witness different root nodes at different time instants, during the course of termination announcement.

The proposed protocol can also be implemented in dynamic networks, *e.g.*, cellular, mobile ad hoc networks, and vehicular ad hoc networks. The proposed *virtual tree-like structure* is also able to decrease the waiting time to announce termination in dynamic networks, which is a desirable requirement in dynamic networks, due to its flexible credit surrender approach. Moreover, the *virtual tree-like structure* can substitute the conventional tree structures in various distributed computations, *e.g.*, snapshot, global-state, leader election, message ordering, and group communication.

Acknowledgements

Authors thank anonymous referees for detailed comments that helped to improve the quality of the paper.

References

[1] Available at: `http://research.microsoft.com/en-us/projects/spectrum/fcc_dynamic_spectrum_access_noi_comments.pdf`.

[2] M. K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Computing*, 13(2):99–125, 2000.

[3] I. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty. A survey on spectrum management in cognitive radio networks. *Communications Magazine, IEEE*, 46(4):40–48, April 2008.

[4] I. F. Akyildiz, W.-Y. Lee, and K. R. Chowdhury. CRAHNs: Cognitive radio ad hoc networks. *Ad Hoc Networks*, 7(5):810–836, 2009.

[5] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey. *Computer Networks*, 50(13):2127–2159, 2006.

[6] T. Bansal, N. Mittal, and S. Venkatesan. Leader election algorithms for multi-channel wireless networks. In *WASA*, pages 310–321, 2008.

[7] D. Cabric, S. Mishra, and R. Brodersen. Implementation issues in spectrum sensing for cognitive radios. In *Signals, systems and computers, 2004. Conference record of the thirty-eighth Asilomar conference on*, volume 1, pages 772–776, 2004.

[8] M. Cesana, F. Cuomo, and E. Ekici. Routing in cognitive radio networks: Challenges and solutions. *Ad Hoc Networks*, 9(3):228–248, 2011.

[9] S. Chandrasekaran and S. Venkatesan. A message-optimal algorithm for distributed termination detection. *J. Parallel Distrib. Comput.*, 8(3):245–252, 1990.

[10] R. F. DeMara, Y. Tseng, and A. Ejnioui. Tiered algorithm for distributed process quiescence and termination detection. *IEEE Trans. Parallel Distrib. Syst.*, 18(11):1529–1538, 2007.

[11] E. W. Dijkstra and C. S. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

[12] S. Dolev, S. Dubois, M. Potop-Butucaru, and S. Tixeuil. Stabilizing data-link over non-FIFO channels with optimal fault-resilience. volume 111, pages 912–920, 2011.

[13] K. Erciyes and G. Marshall. A cluster based hierarchical routing protocol for mobile networks. In *Computational Science and Its Applications–ICCSA 2004*, pages 528–537. Springer, 2004.

[14] Federal Communications Commission, 445 12th Street, SW Washington, DC 20554. *MOBILE BROADBAND: THE BENEFITS OF ADDITIONAL SPECTRUM*, OCTOBER 2010.

[15] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[16] C. Fortuna and M. Mohorcic. Trends in the development of communication networks: Cognitive networks. *Computer Networks*, 53(9):1354–1376, 2009.

[17] N. Francez. Distributed termination. *ACM Trans. Program. Lang. Syst.*, 2(1):42–55, Jan. 1980.

[18] S. Ghosh. *Distributed Systems: An Algorithmic Approach*. Chapman & Hall/CRC Computer & Information Science Series. Taylor & Francis, 2010.

[19] E. Godard, Y. Métivier, M. Mosbah, and A. Sellami. Termination detection of distributed algorithms by graph relabelling systems. In *ICGT*, pages 106–119, 2002.

[20] S. Haykin. Cognitive radio: brain-empowered wireless communications. *Selected Areas in Communications, IEEE Journal on*, 23(2):201–220, 2005.

[21] E. Hossain, D. Niyato, and Z. Han. *Dynamic spectrum access and management in cognitive radio networks*. Cambridge University Press, 2009.

[22] S.-T. Huang. Detecting termination of distributed computations by external agents. In *ICDCS*, pages 79–84, 1989.

[23] S.-T. Huang and P.-W. Kao. Detecting termination of distributed computations by external agents. *J. Inf. Sci. Eng.*, 7(2):187–201, 1991.

[24] P. Johnson and N. Mittal. A distributed termination detection algorithm for dynamic asynchronous systems. In *ICDCS*, pages 343–351, 2009.

[25] S. Katiyar and S. Karmakar. A simple scheme for termination detection in delay tolerant networks. In *ICCSN*, pages 478–482, 2011.

[26] A. D. Kshemkalyani and M. Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[27] H. Kurian, A. Rakshit, and G. Singh. Detecting termination in pervasive sensor networks. In *ISADS*, pages 323–332, 2009.

[28] A. Liotta. The cognitive NET is coming. *Spectrum, IEEE*, 50(8):26–31, August 2013.

[29] Q. Mahmoud. *Cognitive Networks: Towards Self-Aware Networks*. Wiley-Interscience, 2007.

[30] J. Matocha and T. Camp. A taxonomy of distributed termination detection algorithms. *Journal of Systems and Software*, 43(3):207–221, 1998.

[31] F. Mattern. Global quiescence detection based on credit distribution and recovery. *Inf. Process. Lett.*, 30(4):195–200, 1989.

[32] J. Misra and K. M. Chandy. Termination detection of diffusing computations in communicating sequential processes. *ACM Trans. Program. Lang. Syst.*, 4(1):37–43, 1982.

[33] N. Mittal, F. C. Freiling, S. Venkatesan, and L. D. Penso. On termination detection in crash-prone distributed systems with failure detectors. *J. Parallel Distrib. Comput.*, 68(6):855–875, 2008.

[34] N. Mittal, S. Krishnamurthy, R. Chandrasekaran, S. Venkatesan, and Y. Zeng. On neighbor discovery in cognitive radio networks. *J. Parallel Distrib. Comput.*, 69(7):623–637, 2009.

[35] S. Sharma and A. K. Singh. On detecting termination in cognitive radio networks. In *PRDC*, pages 71–78, 2011.

[36] A. K. Singh and S. Sharma. Elite leader finding algorithm for MANETs. In *ISPDC*, pages 125–132, 2011.

[37] R. W. Topor. Termination detection for distributed computations. *Inf. Process. Lett.*, 18(1):33–36, 1984.

[38] Y.-C. Tseng. Detecting termination by weight-throwing in a faulty distributed system. *J. Parallel Distrib. Comput.*, 25(1):7–15, 1995.

[39] Y.-C. Tseng and C.-C. Tan. Termination detection protocols for mobile distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):558–566, 2001.

[40] R. Urgaonkar and M. J. Neely. Opportunistic scheduling with reliability guarantees in cognitive radio networks. *IEEE Trans. Mob. Comput.*, 8(6):766–777, 2009.

[41] B. Wang and K. Liu. Advances in cognitive radio networks: A survey. *Selected Topics in Signal Processing, IEEE Journal of*, 5(1):5–23, 2011.

[42] J. Wang, M. Ghosh, and K. S. Challapali. Emerging cognitive radio applications: A survey. *IEEE Communications Magazine*, 49(3):74–81, 2011.

[43] W. Wang, H. Li, Y. L. Sun, and Z. Han. Securing collaborative spectrum sensing against untrustworthy secondary users in cognitive radio networks. *EURASIP J. Adv. Sig. Proc.*, 2010, 2010.

[44] A. M. Wyglinski, M. Nekovee, and Y. T. Hou. *Cognitive Radio Communications and Networks: Principles and Practice*. Academic Press, 2009.

[45] Y. Zeng, Y.-C. Liang, A. T. Hoang, and R. Zhang. A review on spectrum sensing for cognitive radio: Challenges and solutions. *EURASIP J. Adv. Sig. Proc.*, 2010, 2010.

[46] Y. Zeng, N. Mittal, S. Venkatesan, and R. Chandrasekaran. Fast neighbor discovery with lightweight termination detection in heterogeneous cognitive radio networks. In *ISPDC*, pages 149–156, 2010.

A Complexity Analysis

We analyze our protocol in terms of *message complexity* and *time complexity*. Message complexity is defined in terms of the total number of control messages that are used in our protocol. Time complexity is defined in terms of the time elapsed between the initiation of the protocol and the announcement of termination. Notations used to analyze our protocol are given in Table 8.

A.1 Message complexity

We are using eight types of messages in our protocol (see Table 2). We analyze each message separately, except the *Termination Message* (TM). The TM is sent by C_E (when C_E receives back credit C that was used at the time of credit distribution) to all the nodes of the interaction graph to declare termination of the computation. Since TM can be broadcasted to all the nodes in a unit time, we ignore to analyze this message. The message complexity of each message is also given in Table 9.

- *COMputation message* ($COM(C)$): A node may send $COM(C)$ messages to all its neighboring nodes to distribute the computation. Since there are N nodes in *CRN* and the maximum allowable degree of a node is $\triangle$, $\mathcal{O}(N \times \triangle)$ *COMputation message*s can be exchanged in the protocol.
- *I am Passive with Credit message* ($ImPC(C, b)$): A node sends an $ImPC(C, b)$ to its parent node, if the parent node is active, and to all the neighboring nodes, whose states are active and had sent credits to the node previously. Since at most $N_{neighbor}$ nodes of a node may execute the same protocol and at most N_{leave} nodes may leave the network, the message complexity of $ImPC(C, b)$ is $\mathcal{O}(N_{neighbor} \times N_{leave})$.

Notations	Description
N	The total number of the cognitive radio nodes in the network.
$\triangle$	Maximum degree of a nodes in the network.
$Height$	Maximum height of the *virtual tree-like structure*.
N_{leave}	The total number of nodes that can leave the network during the protocol execution.
$N_{affected}$	Maximum number of nodes that are affected during the protocol execution.
$N_{neighbor}$	Maximum number of neighboring nodes that are executing the same computation.

Table 8: Notations used in the complexity analysis of the *T-CRAN* protocol.

Messages	Complexity
COMputation message	$\mathcal{O}(N \times \triangle)$
I am Passive with Credit message	$\mathcal{O}(N_{neighbor} \times N_{leave})$
I am Passive message	$\mathcal{O}((N_{neighbor} - 1) \times N_{leave})$
Primary user affected Nodes message	$\mathcal{O}(N_{neighbor} \times N_{affected})$
Nodes released by Primary user message	$\mathcal{O}((N_{neighbor} + 1) \times N_{affected})$
AcKnowledgement message	$\mathcal{O}(N_{neighbor} \times N_{leave})$
Acknowledgement of AcK message	$\mathcal{O}(N_{neighbor} \times N_{leave})$

Table 9: Message complexity of the *T-CRAN* protocol.

- *I am Passive message* ($ImP(p)$): A node sends $ImP(p)$ messages to all its child nodes, whose states are active. Since a single node sends at most $(N_{neighbor} - 1)$ $ImP(p)$ messages and at most N_{leave} nodes may leave the network, the message complexity of $ImP(p)$ is $\mathcal{O}((N_{neighbor} - 1) \times N_{leave})$.
- *AcKnowledgement message* (AcK) and *Acknowledgement of AcK message* ($AAcK$): An AcK and an $AAcK$ is generated in response to an $ImPC(C, b)$; hence, $\mathcal{O}(N_{neighbor} \times N_{leave})$ AcK and $AAcK$ messages can be generated in the protocol.

We also present the total number of non-control messages, as follows:

- *Primary user affected Nodes message* (PaN): All the neighboring nodes of an affected node, whose states are active, send PaN messages to C_E. Since at most $N_{neighbor}$ nodes of at most $N_{affected}$ nodes may send PaN messages, the message complexity of PaN is $\mathcal{O}(N_{neighbor} \times N_{affected})$.
- *Nodes released by Primary user message* (NaP): After recovery, the affected node sends PaN messages to its neighboring node and C_E. Since at most $N_{neighbor}$ and C_E receive PaN messages from $N_{affected}$ nodes, the message complexity of NaP is $\mathcal{O}((N_{neighbor} + 1) \times N_{affected})$.

A.2 Time complexity

There are three types of nodes in the network: (*i*) nodes whose $out[] = \emptyset$, (*ii*) C_E, and (*iii*) node whose $out[] \neq \emptyset$ or $in[] \neq \emptyset$ and they are not the chief executive node. The nodes with $out[] = \emptyset$ may leave the network when they finish their computation by sending $ImPC(C, b)$ messages to at most $N_{neighbor}$ nodes. Similarly, C_E may also leave by sending $ImPC(C, b)$ or $ImP(p)$ messages to at most $N_{neighbor}$ nodes. Also, the node other than C_E that has $out[] \neq \emptyset$ or $in[] \neq \emptyset$ exchanges at most $N_{neighbor}$ messages before leaving the network. We assume that all the messages are delivered in a unit time. Hence, in the failure-free network, all the nodes of the network take $\mathcal{O}(Height)$ time to leave the network that results in global strong termination declaration. However, the presence of PUs increases termination latency. In such scenarios, the declaration of global weak termination would be delayed according to the value of timeout.

B Correctness Proof

We first provide the system invariants; afterward, we prove the safety and liveness properties of the *T-CRAN* protocol. We also prove an impossibility result that the appearance of a primary user on a single channel may defy termination forever.

B.1 System invariants

Invariant 1 *Let, $STATE(i)$ represents the state of CR_i, which may be active or passive. For CR_i, $hold_i = 0$ indicates passive state of CR_i and vice versa. Also, $hold_i \neq 0$ indicates active state of CR_i and vice versa.*

$$\forall i : hold_i = 0 \Leftrightarrow STATE(i) = PASSIVE,\ \forall i : hold_i \neq 0 \Leftrightarrow STATE(i) = ACTIVE$$

Invariant 2 *In CRN, the sum of credits at the nodes and credits associated with in-transit messages must be C.*

$$\forall i, j \in v : hold_i + hold_j + in_i[] + in_j[] + SEND_i(m, j) + SEND_j(m), i) = C$$

where, m can be a COMputation message or an I am Passive with Credit message.

Invariant 3 *The global strong termination can be declared, in case, there is no PUs in CRN. Thus, only a single CR_i contains credit value C if the node is the chief executive node and there is no in-transit message, m, in the global channel set, GCS.*

$$\exists i, \forall j : j \in n, i \in j :: hold_i = C \Leftrightarrow i = C_E \wedge STATE(j) = PASSIVE \wedge m \notin GCS$$

Invariant 4 *For the global weak termination, the total credit value C is known to C_E. However, C is distributed among C_E and the affected nodes.*

$$out_{C_E}[] = \emptyset \wedge in_{C_E}[] = \emptyset \wedge (hold_{C_E} + C\_PU_{affected_{C_E}}[] = C)$$

B.2 Safety property

The safety property ensures that in no case a node other than C_E announces termination if the computation has indeed terminated. In order to prove the safety property, we consider all the possible cases that may negate the system invariants and violate the safety requirements, as follows:

1. The incorrect recovery from any failure (*e.g.*, the appearance of PUs, mobility, and crash) may temporarily falsify Invariants 2, 3, and 4. Lemma 1 and Lemma 2 assert that the incorrect recovery from any failure does not violate the safety requirements.
2. Before reaching the actual termination, the value of $hold_{C_E} = C$ or $hold_{C_E} > C$, then Invariant 3 or Invariant 4 are violated. Lemma 3 and Lemma 4 ensure that C_E holds credit C in case of global strong termination and the credit less than C in case of global weak termination.

The proofs of Lemmas 1- 4 guarantee the safety requirements of the *T-CRAN* protocol. The following Lemma 1 and Lemma 2 prove that the nodes do not violate the safety requirements on their recovery.

Lemma 1 *The reception of stale messages, $m_{\langle session,*\rangle}$, at the nodes do not increase credit value C forever, which violates the safety requirements of the T-CRAN protocol.*

Proof. Assume that on recovery,[6] CR_i receives stale messages, $m_{\langle session,*\rangle} = m_{\langle x,*\rangle}$, from unreliable channels or other recovered nodes. The reception of $m_{\langle x,*\rangle}$ at CR_i is able to execute the computation and transmission of $m_{\langle x,*\rangle}$, in case $session_i \leq x$. For the contrary, we assume that a node receives a stale message, $m_{\langle x,*\rangle}$, executes the computation and propagates $m_{\langle x,*\rangle}$. We now prove that the reception of stale messages does not violate the safety requirements, as follows:

CR_i can further distribute the computation or surrender credit after completion of its computation among its neighboring nodes, in response to $m_{\langle x,*\rangle}$. The neighboring node CR_j of CR_i may be a recovered node or unaware of the just terminated computation whose $session = x$. Hence, the recipient CR_j can also behave similar to CR_i. However, one of the nodes in the network or C_E terminates the flow of $m_{\langle x,*\rangle}$ due to $session_{C_E} \neq x$ (Action A_4 in Table 4).

Hence, the system maintains Invariant 2, and once the credit is greater than C, it is detected by some nodes; thus stale messages cannot violate the safety requirements of the *T-CRAN* protocol. ■

The following assumptions help us to prove Lemma 2: we use four different time instants α, β, γ, and δ such that $\alpha < \beta < \gamma$ (all the other lemmas will also use these time instances) and three nodes CR_i, CR_j and CR_k that are neighbors of each other. CR_i initiates the *T-CRAN* protocol at time α among CR_j and CR_k with $session = x$. Under a fault-free scenario, at time γ, CR_i announces global strong termination. Suppose, CR_k becomes an-affected node at time β.

Lemma 2 *On recovery, the initiation of a node in active or passive state does not result in false termination.*

Proof. We first mention all the possible situations that may exist at the time of transition of a node from an affected node to a non-affected node or vice versa, which may announce false termination. Afterward, we prove that none of these situations can lead to the violation of the safety requirements in our protocol.

CASE 1 CR_k is not able to recover, *i.e.*, CR_k is an affected node for a very long time.
CASE 2 CR_k recovers, due to availability of another available channel in LCS_k or disappearance of the PU, at time δ, where $\delta < \gamma$.

[6] We assume that the recovery process takes non-zero time.

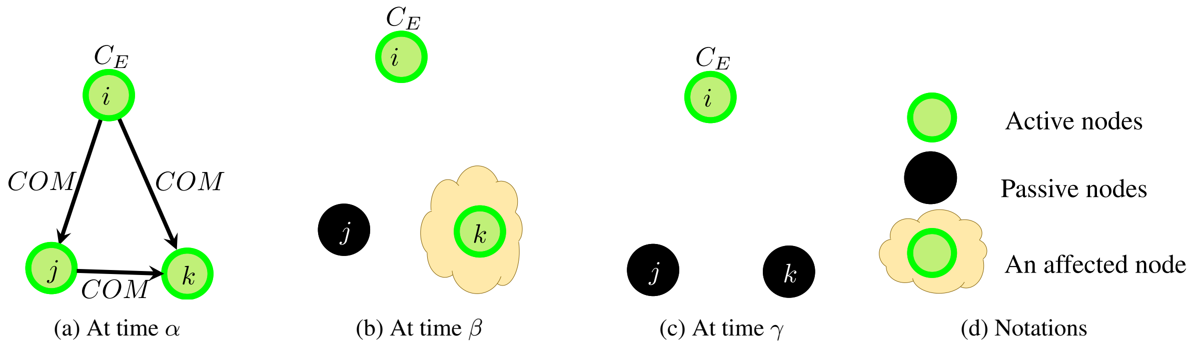


Figure 8: Illustration for the proof of Lemma 2.

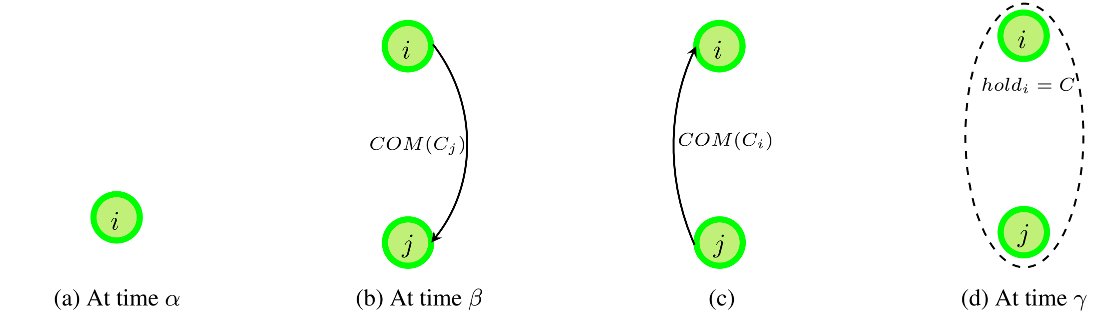


Figure 9: Illustration for the proof of Lemma 3.

Case 3 CR_k recovers, due to availability of another available channel in LCS_k or disappearance of the PU, at time δ, where $\delta > \gamma$.

The Case 1 results in permanent failure of CR_k, *i.e.*, CR_k is a crashed node. Hence, the global strong termination is defied forever, and the protocol announces global weak termination of the computation at CR_i and CR_j.

The Case 2 results in the global strong termination, when $session_k = session_{C_E}(= session_i)$ at the time of recovery of CR_k in active state. However, passive state of the node is irrelevant here, because passive state of CR_k indicates that CR_k has already surrendered its credit before the transition from a non-affected node to an affection node.

In Case 3, C_E has already declared global weak termination before recovery of CR_k. Specifically, Case 1 and Case 3 are almost similar and do not affect C_E, because $session_{C_E} \neq session_k$. In addition, the recovery of CR_k in active state may cause to propagate messages, $m_{\langle session,*\rangle} = m_{\langle x,*\rangle}$, to CR_i or CR_j. However, according to Lemma 1, CR_i, which is C_E, discard $m_{\langle x,*\rangle}$ eventually because $seesion_{C_E} \neq x$. (For a better understanding, readers may refer to Figure 8)

Thus, on recovery, the nodes' state do not affect the correct termination, and Invariants 3 and 4 holds. ■

The credit aggregated at C_E never ever becomes equal to C before the global strong termination is reached. This fact can be justified with the help of Lemma 3 and Lemma 4, as follows:

Lemma 3 *Under no condition the credit aggregated at C_E equals to C except in the case of global strong termination.*

Proof. Suppose, only two processors CR_i and CR_j are executing a computation, and CR_i is the chief executive node. CR_i sends credit C_j to CR_j, and again, CR_j sends credit C_i to CR_i. Thus, according to Invariant 2, the following equation 2 holds true:

$$hold_i + SEND_i(COM(C_j), j) + hold_j + SEND_j(COM(C_i), i) = C \tag{2}$$

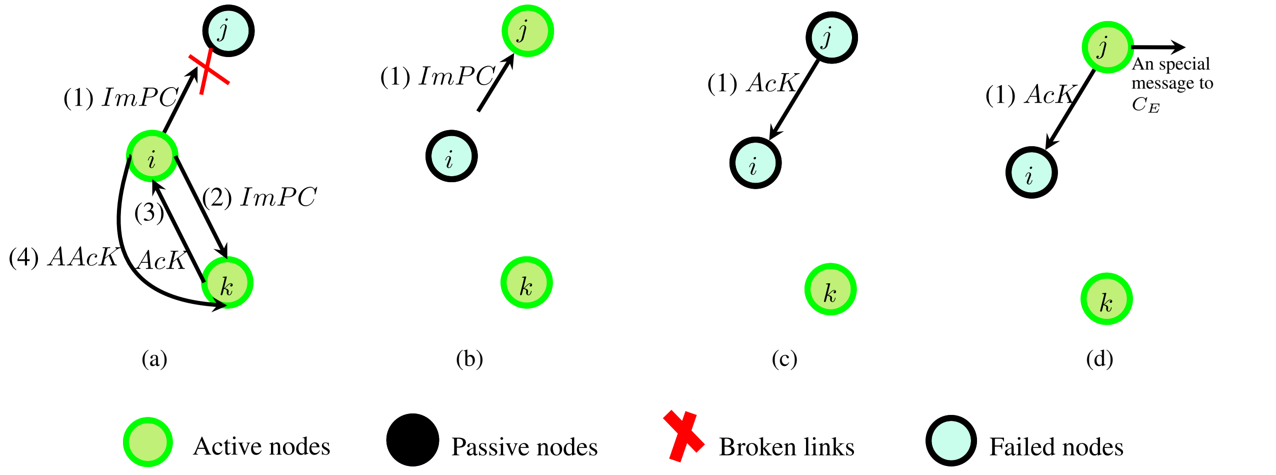
Suppose at time α, CR_i becomes active. Thus, $SEND_i(COM(C_j), j) = 0$. However, at time β, the following equation 3 holds true:

$$hold_i + SEND_i(COM(C_j), j) + hold_j = C \tag{3}$$

The above equation 3 indicates credit distribution using a *COMputation message* from CR_i to CR_j. However, once CR_j receives the *COMputation message*, then $SEND_i(COM(C_j), j) = 0$. Thus, the following equation 4 holds ture:

$$hold_i + hold_j = C \tag{4}$$

Assume the contrary, at a later time γ, $hold_i = C$, and CR_i, which is the chief executive node, declares global strong termination, while $STATE(CR_i) = STATE(CR_j) = ACTIVE$. It is possible only when CR_i and CR_j are two



(a) (b) (c) (d)

(e) Notations

Figure 10: Illustration for the proof of Lemma 4.

processes at an identical node, *i.e.*, $CR_i = CR_j$. However, the chief executive node, CR_i, never declares global strong termination despite $hold_i = C$, unless $STATE(CR_i) = PASSIVE$ (Action C_2, Table 6). Therefore, the protocol never declares termination unless all the nodes are passive, and $hold_{C_E} = C$. (For a better understanding, readers may refer to Figure 9.)

The above proof can be generalized for any number of participating nodes. Thus, the protocol always aggregates correct credit in case of the global strong termination. ■

The credit aggregated at C_E never becomes (*i*) greater than or equals to C, in case of the global weak termination and, (*ii*) greater than C, in case of the global strong termination. We call the *wrong credit aggregation* when $hold_{C_E} \geq C$ in case of the global weak termination and $hold_{C_E} > C$ in case of the global strong termination. Both the above facts are proved by Lemma 4, as follows:

Lemma 4 *No message transmission ever results in wrong credit aggregation in weak or strong termination declaration.*

Proof. The conditions that may lead to wrong credit collection at C_E are the aggregation of an identical credit at more than one node, and the stale messages in the network. However, we have proved that the stale messages are eventually discarded (Lemma 1). Hence, we consider the aggregation of an identical credit at more than one node.

We first present a scenario that may lead to multiple times credit surrender of an identical credit at two different nodes. Note that duplicate message reception at a node is handled in the protocol; hence, we not consider it. We present two possible cases that may lead to wrong credit aggregation, and following that we prove by contradiction that these cases never arise in the protocol.

Suppose, an ongoing computation with $session = x$, where CR_i completes its computation and sends an $ImPC(hold_i, b)$ to CR_j. In the meantime, suppose CR_j becomes an affected or a failed node (see Failure Model, Section 2). Thus, CR_j cannot send an AcK to CR_i. Due to non-reception of an AcK from CR_j, CR_i sends an $ImPC(hold_i, b)$ to another node, say CR_k. However, the recovery of CR_j and the reception of an $ImPC(hold_i, b)$ at CR_j signify that $hold_i$ is surrendered at two different nodes. However, in the protocol, there are only two possible cases of the *imperfect credit surrender*, as follows:

CASE 1 CR_j is an affected or a failed node after sending an AcK to CR_i, and CR_i is already an affected or a failed node after the transmission of an $ImPC(hold_i, b)$.

CASE 2 CR_i sends an $ImPC(hold_i, b)$ to CR_j, and due to the absence of an AcK from CR_j, CR_i sends an $ImPC(hold_i, b)$ to CR_k. After sending $ImPC(hold_i, b)$ messages to two different nodes, CR_i becomes an affected node. Also, CR_j receives the $ImPC(hold_i, b)$.

The reception of an AcK and an $AAcK$ is assumed to be an atomic operation, *i.e.*, CR_j is not allowed to move to a different location before the reception of an $AAcK$ or a timeout, and also, CR_j receives an $AAcK$ in the presence of PUs. Therefore, CASE 1 never holds true.

The CASE 2 is essentially an outcome of the CASE 1. CR_j receives an $ImPC(hold_i, b)$ and sends an AcK to CR_i. As the affected node CR_i cannot receive an AcK from CR_j; after a timeout value, CR_j recognizes that CR_i is an affected node. Thus, CR_j sends a special message, m, with credit $hold_i$, to C_E. Eventually, C_E subtracts $hold_i$ from $hold_{C_E}$.

On recovery, CR_i again surrender its credit (Lemma 2). On the other hand, CR_i receives an AcK from CR_j though it has already surrendered its credit to CR_k earlier; thus, CR_i behaves as an affected node. Therefore, the credit of CR_i remains a constant in the network. (For a better understanding, readers may refer to Figure 10.)

Thus, Invariants 3 and 4 are preserved, and an imperfect surrender of a credit is infeasible in the *T-CRAN* protocol. ■

B.3 Liveness property

In the *T-CRAN* protocol, C_E eventually announces termination in finite time. In order to prove the liveness, we show:

1. The *virtual tree-like structure* does not grow infinitely.
2. The height of the *virtual tree-like structure* eventually reduces, to (*i*) one, in case of the global strong termination, and (*ii*) two, in case of the global weak termination.

We show that the *virtual tree-like structure* grows and has a finite height. C_E, *i.e.*, the root of the *virtual tree-like structure*, expands the computation among N nodes (using A_2 and A_3, Table 4). The distribution of the computation and credit increases the height of the *virtual tree-like structure*, and the maximum height of the *virtual tree-like structure* can be N. However, the joining of new nodes in the network during the computation execution may increase the total number of CRs and the maximum height to $N + k, k > 0$. In this manner, the *virtual tree-like structure* grows infinitely. However, once the nodes stop to join the network, then the *virtual tree-like structure* does not grow infinitely.

We now show that the height of the *virtual tree-like structure* eventually reduces. The nodes are not allowed to delay the computation for an infinite time. Hence, once all the nodes (except C_E), whose heights are identical, complete their computation, they send $ImPC(C, b)$ and $ImP(p)$ messages to their parent node or to any number of neighboring nodes, if they are active (Action A_4, Table 4). Thus, the transmission of $ImPC(C, b)$ and $ImP(p)$ messages by all the nodes (except C_E), whose heights are identical (not necessary at an identical time), results in reduction of the height of the *virtual tree-like structure* by at least 1.

We now show that when the *virtual tree-like structure* has height 1 after credit aggregation, it is a sufficient condition to announce the global strong termination. From the previous facts, it is clear that the height of the *virtual tree-like structure* reduces by at least 1 when all the nodes (except C_E), whose heights are identical, send $ImPC(C, b)$ and $ImP(p)$ messages. Hence, when all the nodes of all the height levels (except C_E), send $ImPC(C, b)$ and $ImP(p)$ messages that result in the height of the *virtual tree-like structure* to be 1, eventually, and only a single node, C_E, holds the complete credit (that is equal to the credit that was distributed at the time of initiation). This fact is enough to show that at the time of the global strong termination the *virtual tree-like structure* has height 1.

We now show that when the *virtual tree-like structure* has height 2 after credit aggregation, it is a sufficient condition to announce the global weak termination. From the previous facts, it is clear that the height of the *virtual tree-like structure* reduces when the non-affected nodes sends $ImPC(C, b)$ and $ImP(p)$ messages. In addition, only a single non-affected node, C_E, holds the credit of all the non-affected nodes eventually. Since the affected nodes cannot send $ImPC(C, b)$ and $ImP(p)$ messages, the network is divided into two partitions as: CRN_P and CRN_N (Figure 5). Hence, the *virtual tree-like structure* has height 2, and it is sufficient for the global weak termination.

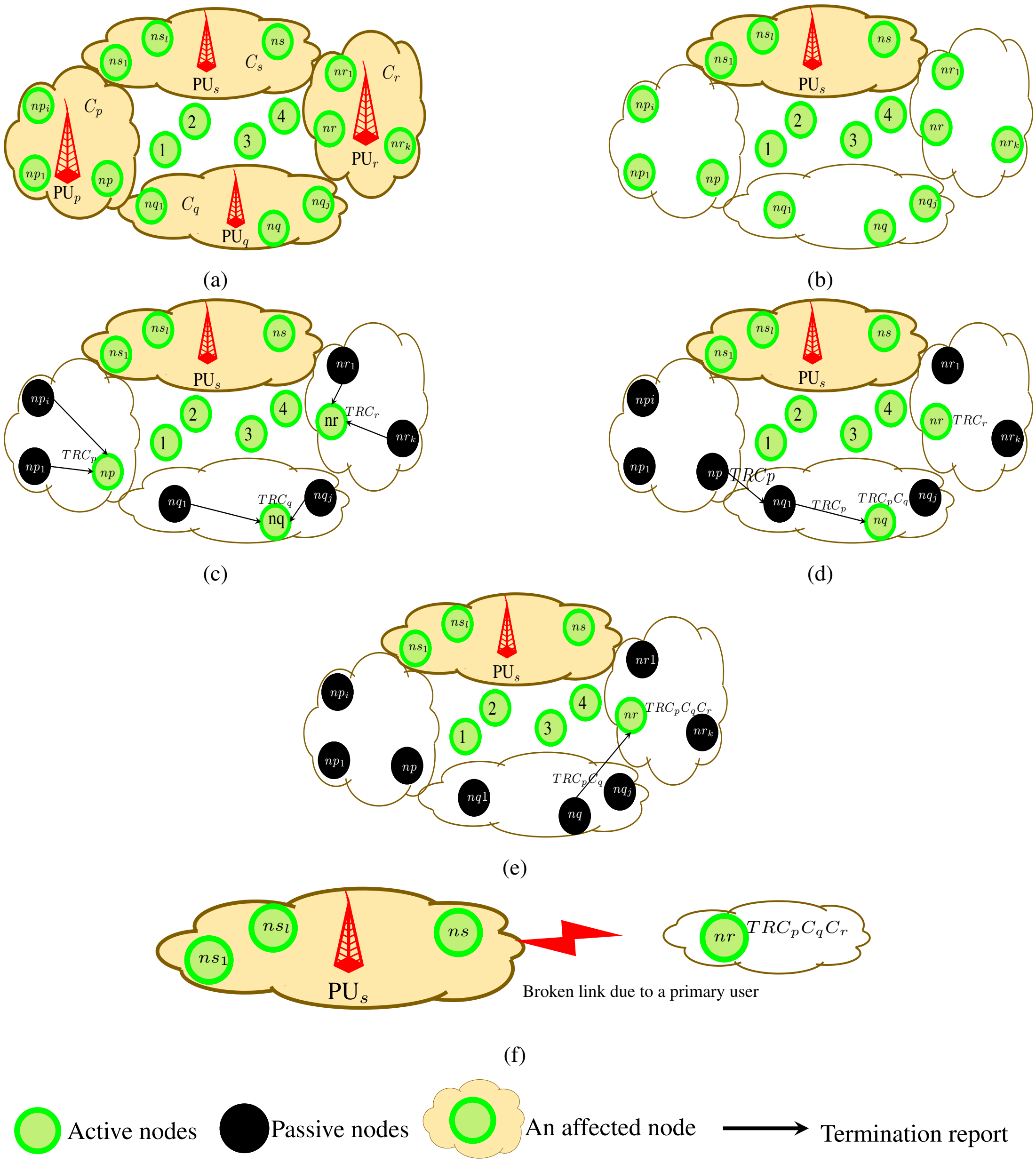
B.4 The impossibility of termination

We provide an abstract view to show the impossibility of the global strong termination in the presence of a single primary user. By this abstract view, it will be clear that the appearance of a primary user is difficult to handle than mobility and crash of nodes. (Note that in a purely asynchronous *CRN*, the global strong termination is impossible [15] to detect even if a single primary user exists in the network.)

We consider a *CRN* as a connected communication graph that has nodes $np, np_1, \ldots, np_i$, $nq, nq_1, \ldots, nq_j$, $nr, nr_1, \ldots, nr_k$, $ns, ns_1, \ldots, ns_l$; and no node is assumed to be special. (The node ids are selected in a special way to help readers to understand the abstract view, which will be clear soon.) Also, we assume four PUs, namely PU$_p$, PU$_q$, PU$_r$, and PU$_s$ that affect all the nodes. The *virtual clustering* is performed by considering np, nq, nr, and ns as fixed centers [13] (or cluster heads) that partition the communication graph into four virtual clusters, say C_p, C_q, C_r, and C_s, see Figure 11a.

Now, assume that three PUs, namely PU$_p$, PU$_q$, and PU$_r$, disappear from the network. Thus, the nodes, namely $np, np_1, \ldots, np_i, nq, nq_1, \ldots, nq_j, nr, nr_1, \ldots, nr_k$, become non-affected nodes, Figure 11b. All the non-affected nodes of each cluster send $ImPC(C, b)$ messages to the respective cluster heads. Hence, each cluster head holds a termination report (or credit) of its cluster, namely np has TRC_p, nq has TRC_q, and nr has TRC_r, Figure 11c.

In order to announce the global strong (or weak) termination in the network, it is required to aggregate all the termination reports (or credits) of each cluster head. Thus, np sends TRC_p to its neighboring virtual cluster head nq, and nq aggregates the received credit as: $TRC_pC_q = TRC_p \cup TRC_q$, Figure 11d. This state of the network is equivalent



(a) (b)

(c) (d)

(e)

(f)

(g) Notations

Figure 11: Illustrating the impossibility of termination in cognitive radio networks.

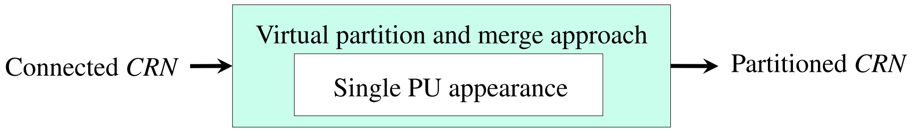


Figure 12: The virtual partition and merge approach.

to the *virtual merging* of both the virtual clusters C_p and C_q in one virtual cluster, where C_q is a virtual cluster head. Similarly, TRC_pC_q is aggregated with TRC_r at nr as: $TRC_pC_qC_r = TRC_pC_q \cup TRC_r$, Figure 11e.

Since a PU, PU$_s$, persists in the network and $ns, ns_1, \ldots, ns_l$ are affected node, an aggregated termination report (or credit) of the network cannot be generated. The network is now partitioned into two parts, where the first part holds credit $TRC_pC_qC_r$ at nr and the remaining credit is distributed among the affected nodes, $ns, ns_1, \ldots, ns_l$. This state of the network is equivalent to the *virtual partitioning* of the network into two virtual clusters: one virtual cluster, where C_p, C_q, and C_r are virtually merged and nr is a virtual cluster head, and the another virtual cluster C_s with ns as a virtual cluster head, Figure 11f.

Therefore, it is shown that the existence of a single PU results in two isolated sub-networks, which is sufficient to prevent the protocol to announce the global termination. Without loss of generality, the above *virtual partition-merge* technique (Figure 12) can be applied to a network of arbitrary size and arbitrary number of PUs.

Note: When the chief executive node, C_E, becomes an affected node, neither strong nor weak termination detection is possible.

Authors' Biographies

Shantanu Sharma is a PhD student at Ben-Gurion University of the Negev, Israel. He received his MTech degree in Computer Science from National Institute of Technology, Kurukshetra, India, in 2011. His research interests include designing models for MapReduce computations, distributed algorithms, and mobile computing.

Awadhesh Kumar Singh received his BTech degree in Computer Science from Gorakhpur University, Gorakhpur, India, in 1988, and his MTech and PhD degrees in Computer Science from Jadavpur University, Kolkata, India, in 1998 and 2004, respectively. He joined the Department of Computer Engineering at National Institute of Technology, Kurukshetra, India, in 1991, where he is Professor at present and earlier he also served as the chairman of Computer Engineering Department during 2007-2009. His research interests include distributed algorithms, mobile computing, and fault tolerance.