A Synchronous Tiered Based Clustering Algorithm for large-scale Ad hoc Networks

Imen Jemili[1,2]**, Abdelfatteh Belghith**[1]**, and Mohamed Mosbah**[2]

HANA Lab.
N. School of Computer Sciences, Tunisia , e-mail: imen.jemili@cristal.rnu.tn,abdelfattah.belghith@ensi.rnu.tn

LaBRI
University of Bordeaux I, Talence, France e-mail: {jemili,mosbah}@labri.fr

Abstract Relaying on a virtual backbone formed by the induced hierarchy of a connected dominating set is a widespread solution in ad hoc networks. However, the majority of existing approaches require gathering neighborhood information without considering the loss of information due to collision and the effect of wireless interference and its impact on the accuracy of information used during clustering process. In this paper, we present an improved version of our clustering algorithm, TBCA, operating in layered manner and exploiting the eventual collision to accelerate the clustering process. We prove through simulations that our mechanism outperforms other ones, in terms of effectiveness throughput and per-packet sojourn delay. Conducted simulations show also that it copes better with mobility.

1 Introduction

Clustering is considered as a promising approach to face one of the most critical problems confronting the ad hoc networks: the scalability. Indeed, the dynamic topology, limited mobile node capability and limited link bandwidth pose a challenge for the scalability of these networks.

The clustering process aims to build a hierarchy among the nodes, by organizing nodes into smaller groups called clusters simpler to manage. A virtual backbone can be formed by clusterheads and sometimes gateway nodes in a connected dominating set. A node set is a dominating set (DS) if every node in the network is either in the set or a neighbor of a node in the set. The DS is called a connected dominating set (CDS) if any two nodes in a DS can be connected through intermediate nodes from the DS. The virtual backbone formed by the induced hierarchy of a CDS plays a very important role in routing, broadcasting and connectivity management in wireless ad hoc networks [1, 2]. Many CDS formation algorithms exist in the literature. However, constructing a virtual backbone comes at a cost in terms of the time it takes for CDS building and also the overheads incurred in the additional network traffic generated. In fact, most of the proposed approaches require the knowledge of

the h-hop neighborhood information only with a small h. However, the lack of structure is a characteristic of wireless multi-hop radio networks when being deployed. Every node is unable to know its neighbors or the number of its neighbors without exchanging control messages with its neighborhood. Thus, a discovery phase is required preceding the execution of the algorithm aiming to construct a set of connected dominating nodes. Besides, over time, the dominating set must be updated to reflect the changes occurring in the network topology due to node mobility. The maintenance phase should also require as little communication between mobile nodes as possible. The majority of the proposed approaches gather neighborhood information periodically to maintain the CDS; collecting information takes time and there is a high likelihood of changes in network connectivity before the termination of this gathering phase required for maintenance. The use of explicit message exchange among neighbors in periodic basis can greatly affect the performance of the upper-layers protocols in large scale networks. Further, these algorithms do not take into account the presence of message losses due to eventual collision and the effect of wireless interference.

In this paper, we present an improved version of DCAWNK [3, 4], entitled TBCA (Tiered Based Clustering Algorithm). The main contribution of DCAWNK is that our mechanism does not necessitate any type of neighborhood knowledge, trying to alleviate the network from some control messages exchanged during the clustering and maintenance process. Consequently more bandwidth is left for transporting data traffic. The new version guarantees the construction of a dominating node set. So, we detail the three overlapping phases followed to construct a connected dominating set. The two first phases allow us to select the clusterheads and then gateways. While the third one allows us to verify that all clusterheads are connected through gateway neighbors. Improvements concerning rules for selecting the appropriate candidate for clusterhead and gateway roles are also presented. The rest of the paper is organized as follows. Section 2 reviews the related works; we focus specifically on clustering algorithms based on graph domination. Then, we give an exhaustive description of our clustering algorithm in section 3. We confirm the effectiveness of our algorithm through exhaustive simulations in section 4. In section 5, we conclude.

2 Related works

Many approaches have been proposed to construct a connected dominating set. Wu [5] proposed a distributed algorithm to find a CDS in order to design efficient routing schemes. The first phase allows nodes with at least two unconnected neighbors to be marked as dominator. Then some extension rules are implemented to reduce the size of a CDS generated from the marking process. Wu and Li algorithm was improved in term of message overhead in [6]. An extension of Wu and Li's distributed algorithm has been presented in [7], where the connected dominating set selection is based on the node degree and the energy level of each host. In [2], the authors

proceed in the same way. A periodic exchange of hello permits nodes to gather two hop information allowing nodes to verify their eligibility for coordinator role. In [8], a dominating set is firstly constructed, then, the second step permits to connect the dominating set to form a CDS. The authors adopt in [9] the same way, a leader node triggers the construction of a Maximal Independent Set (MIS) by sending its hello message. Selection of others dominators of the MIS is based on the maximum weight in a given neighborhood. In [10], the authors propose a fast interference-aware distributed algorithm for wireless networks which constructs only a dominating set in an asynchronous environment. They assume that the network nodes have no information about their local neighborhood and do not possess a reliable collision detection mechanism. In [11], the authors present local control algorithmic techniques for CDS construction in wireless ad hoc networks which take into account message losses due to collisions from interfering transmissions. One of the described distributed algorithms requires each node to know their three-hop topology, while the second requires the knowledge of the maximum degree and the size of the network. This latest necessitates gathering two hops neighbor information.

Most of previous works have focused on selecting a small virtual backbone for high efficiency. A CDS with small size reduces the number of nodes involved in routing activities which contribute to deplete their energy quickly. It is also important to maintain a certain degree of redundancy in the virtual backbone for fault tolerance and routing flexibility. Besides, all these propositions rely on gathering h-hop neighborhood information, only with a small h. In many mechanisms, no mechanism was specified how to proceed when information loss occurs due to collision between data packets and the control packets or due to effect of interference. Moreover, over time, the CDS must change to reflect the changes in the network topology as nodes move. Many CDS schemes employ explicit message exchange among neighbors in periodic basis for maintaining the backbone structure. They apply local repairs, with the target of enhancing as much as possible the quality and stability of the hierarchy organization, by avoiding any unnecessary structure changes. However, in highly dynamic ad hoc net-works, local repairs are insufficient, since they are based on gathering information, which can be not up-to-date when being used by nodes. Algorithms requiring two hops or more neighborhood knowledge face this drawback, since the propagation of this information necessitate time.

3 Connected Dominating Set construction

Our clustering algorithm DCAWNK is based on the presence of a specific node called the Designated Node (DN). Electing the DN node is out of the scope of this paper. The basic idea is to organize the clustering process into layered stages in order to limit the number of participating nodes at the clustering process at a given instant. In this way, we reduce the eventual collisions between the control messages dedicated to the clustering phase. Through the improved version, TBCA, we guarantee

the construction of a connected dominating set formed by elected clusterheads and gateways. Thus, we add a checking phase assuring that all clusterheads can communicate through gateways. Besides, we define new rules for clusterhead selection and gateway declaration in order to allow the most appropriate nodes to gain clusterhead or gateway role. We aim also to minimize the total number of nodes elected as clusterhead or as gateway. At the start of the clustering process, all nodes are unaffected, marked to N. We assume that all the nodes are synchronized thanks to an external agent.

3.1 Clusterhead and member declaration

The DN node triggers the clustering process by sending a cbeacon message exactly at TBTT[1]. The latter is the first declared clusterhead and remains a clusterhead forever. Its immediate neighbors will join its cluster as member nodes (M) and reply after a SIFS [2] by an ACK_BT. To avoid signal attenuation especially in large-scale ad hoc networks, we impose to nodes with odd identity to send their ACK_BT after a SIFS sensing period, while, nodes with even identity will transmit their ACK_BT after 2 SIFS periods. Simultaneous sending of ACK_BT generates hence a collision, considered like a 'Busy Tone' (BT). The busy tone will be intercepted by the two hop neighbors of the DN informing them that they are able to compete to become clusterheads. A node which has heard a cbeacon message is not allowed to participate into the clusterhead election. A node, marked to N, which intercepts a BT collision or an ACK_BT is an eligible candidate for the clusterhead role (candCH). The eligible nodes can concur to gain the clusterhead status in a randomized manner. To compute the backoff time, we prefer to use local information in order to avoid the required collecting information phase which is prone to collisions. So, each candCH node i computes a new boundary D_1^i for the backoff computing delay taking into account the energy factor, as follows:

$$D_1^i = \lceil ((\frac{(D_{1min} - D_{1max})}{E_{max}}) * E_i) + D_{1max} \rceil \quad (1)$$

Where:

D_{1max}, D_{1min} : The maximal and minimal boundaries imposed for the backoff calculation by eligible candCH nodes

E_{max}, E_i : The maximal and remaining energies at node i

The personalized boundary D_1^i favors the declaration of nodes having sufficient energy, since clusterheads will have to assume additional tasks like routing. Every eligible node will calculate a random delay d_1 uniformly distributed in the range between zero and the new calculated boundary D_1^i. Our aim is to reduce the collision probability between multiple candCHs accessing the medium for announcing their

[1] TBTT : Target Beacon Transmission Time

[2] SIFS : Short InterFrame Space

cbeacon. Thus, we fix a minimal boundary different from zero. The random delay d_1 timer is decremented according to the defined CSMA/CA backoff algorithm. A candCH cancels the remaining random delay and the pending cbeacon transmission, if a cbeacon packet arrives before the random delay d_1 timer has expired. So, such a node will be assigned member status while joining the cluster of the cbeacon sender. Otherwise, it declares itself a clusterhead after transmitting its own cbeacon packet. Each node, which receives a cbeacon, sends an ACK_BT in order to inform two hops clusterhead neighbors that are able to candidate to clusterhead role. In this way, the clustering process progress in a layered manner.

To be able to differentiate between collisions which happened between cbeacon messages from collisions which happened between a cbeacon and ACK_BT messages, we force the layering to be centered at the DN. We impose the termination of the clustering process in a specific layer before allowing nodes in the subsequent layer to start the clustering process. Furthermore, nodes in the current layer that have heard cbeacon(s) transmit their ACK_BT at the same time upon the termination of the clusterhead election process at this layer. So, we divided the time in a succession of periods of time, denoted by T_i, i=1..N, where N represents the number of layers needed to cover the entire network. Depending on the context, we consider that T_i denotes also the instant at which period T_i ends. Period T_i defines the time duration required by the execution of the clustering process for layer i. It includes a sub period for the announcements of cbeacon messages followed by a second sub period for the simultaneous transmission of ACK_BT. We denote by T_{iBT} the instant at which this second sub period starts; as shown in Figure 1. Collisions occurring among cbeacons messages (only possible during the first sub period) are called CH collision, and the collisions between ACK_BT messages (happening during the second sub period) are called BT collision.

Let us now express explicitly the T_i, i = 1..N. We define the following quantities:
Txcbeacon : The transmission time of a cbeacon
TxBT : The transmission time of a ACK_BT
SlotTime : The slot time
CWsize : the contention window which is slot count
We get the following expressions for T_i, for i=1...N:

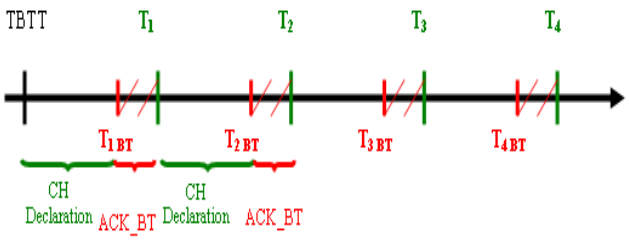

Fig. 1 Time division for clustering process

$$T_1 = SIFS + Txcbeacon + (2 * SIFS) + TxBT \tag{2}$$

$$T_N = T_1 + (N-1) * (SIFS + CWsize + Txcbeacon + (2 * SIFS) + TxBT) \tag{3}$$

And the following expressions for T_{iBT}, i=1.. N:

$$T_{1BT} = SIFS + Txcbeacon \tag{4}$$

$$T_{NBT} = T_{N-1} + (SIFS + CWsize + Txcbeacon) \tag{5}$$

Since the beginning of the clustering process, all nodes will be passive, except the DN node. Each node will estimate continuously the instants T_i and T_{iBT} while waiting for clustering process to reach its level. We note that upon the first reception of a cbeacon or an ACK_BT or the detection of a CH collision or BT collision, a node realizes that the current clustering process concerns its layer.

3.2 Gateway declaration

3.2.1 Normal candGW declaration

During the clustering process, a node within layer i, that hears a correct cbeacon message, modifies its status to a Member node and adds the sender node identity to its clusterhead list (CH list). A member node that receives two or more correct cbeacon messages during T_i or T_{i+1} becomes a candidate gateway (candGW), called a normal candGW. This node is eligible to ensure lately the role of a gateway between its clusterhead neighbors. Immediate declaration of a candGW will cause troubles for next layer nodes busy with the clustering process in progress. Thus, a candGW node within layer i must wait a period of time, denoted by TG_i, in such a way to not perturb the clustering process traffic of the layers i and i+1. To this end, computing the period TG_i by a normal candGW at level i is done as follows for i = 1..N:

$$TG_i = T_{i+2} - t \tag{6}$$

Where, t denote the instant of node declaration as candGW.

In figure 2, the node k is a candGW for the clusterheads i, j and m. Waiting for a period TG_i aims to eliminate all risk of collision with the cbeacon messages that can be sent by candCH nodes within the next layer, such as node m. The period TG_i was also computed in such a way to minimize the risk of interference able to prevent the good receipt of the cbeacons sent by the candCH at i+2 level, like the node o. After waiting the TG_i period, all normal candGW, belonging the same level i, are authorized to compete in order to acquire the gateway status. Each candGW selects a random backoff delay d_2. To favor candGW nodes closer to many clusterheads, each candGW i computes a personalized boundary D_2^i as follows:

$$D_2^i = (\frac{D_2}{CH\_Neighbor\_Number}) \quad (7)$$

Where D_2 is the maximal boundary used

So, each candGW chooses a random slot number in the range of zero to D_2^i. We note that the personalized boundary D_2^i can't exceed the threshold $D_2/2$, since a node must hear at least two correct cbeacons to obtain the candGW status. Upon expiration of d_2 backoff delay, a candGW is authorized to transmit its declaration to be an effective gateway for the clusterheads in its CH list, if it is steal eligible for this role. An eligible node for gateway role is a candGW with at least one uncovered CH neighbor. Otherwise, it becomes a member node. In fact, at the receipt of other gateway declaration, a candGW is able to verify if its own clusterheads are covered or not. Besides, every normal candGW or gateway renounces to the gateway role if it hears another GW declaring itself for a set of clusterheads that covers its own, the latest is called a dominating gateway. We note that during checking eligibility operation, a candGW or a GW must take into account the declaration of other GWs within the same layer. With this rule, we ensure that enough gateways will be elected to assure communication between clusterheads within adjacent layers.

3.2.2 Declaration of candGW with anonymous CHs

During the step of candCH declarations within a layer i, we can't eliminate completely the risk of collision occurrence. CandGWs with anonymous CHs gather nodes which intercept a CH collision due to a simultaneous transmission of two or more cbeacons from candCH nodes during T_i or T_{i+1}. In this case, the node adds an anonymous clusterhead to its CH list to represent the existence of two or more unknown clusterheads. However, such a node will not be assigned candGW status in all situations.

1) Detection of a CH collision by a node within level i during the first sub period of T_i

During the clusterhead election at level i, a node can be assigned one of these states when intercepting a CH collision: candCH, N node, M node or a candGW node. A candCH node or node, initially marked to N, acquires only member sta-

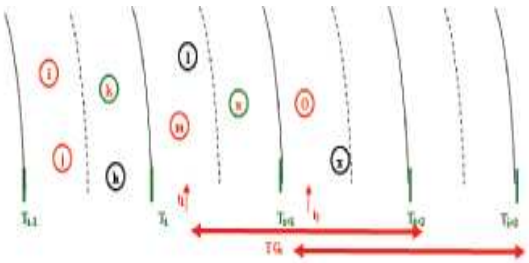

Fig. 2 Waiting period TG_i

tus despite the fact that it is closer to many clusterheads. While a member node conserves also its member status. In such a situation, a candGW node is already an eligible candidate for assuming gateway role. Subsequently, this candGW must indicate the existence of anonymous clusterheads in its list and specify the time of collision occurrence.

We remind that our main objective is to minimize the whole gateways number while guarantying the declaration of enough gateways in order to assure communication inter layers. Thus, communication between clusterheads within the same layer can be done easily through gateways of the previous layer i-1. Moreover, in case of dense ad hoc networks, the collision probability between multiple candCHs accessing the medium is higher. Such collisions may be detected by many nodes within the same layer, like nodes o, m and h (see figure 3). Authorizing all these nodes to acquire candGW status will increase considerably the number of candGW. Besides, no rule is able to differentiate efficiently between such candGWs, since they do not know all their neighboring clusterheads.

2)Detection of a CH collision by a node within level i during the first sub period of T_{i+1}

When the clusterhead election at layer i+1 starts, the clustering process at level i is already stopped. So, only member or candGW nodes, within a layer i, are able to detect a CH collision during T_{i+1}. Such a candGW adds an anonymous clusterhead to its CH list and memorizes to time of collision occurrence, like the node j (see figure 3). Same actions are undertaken by a member node and it declares itself as candGW. However, we must allow normal candGW to candidate for gateway role before authorizing candGWs having an anonymous clusterhead in their CH list also to compete. So, after the waiting period TG_i, such candGWs select a random delay d_2 in the range $D_2/2$ to the maximal boundary D_2. Each candGW must indicate the number of anonymous clusterheads and the times of collision detection. We note that we associate an anonymous CH for each CH collision. Thanks to these indications, other similar candGWs can verify their eligibility to assure gateway role. A such candGW compares the instants of collision detection indicated in the received declaration to their own in order to estimate if they are close to the same clusterheads. Thus, a steal eligible candGW announces its declaration whenever the

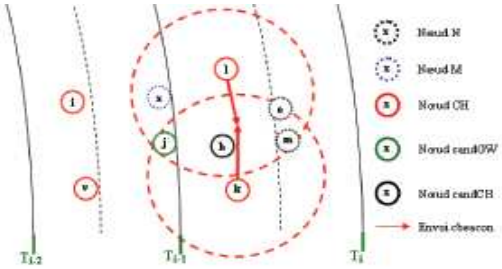

Fig. 3 Detection of a CH collision

backoff delay expires. Otherwise, it cancels the gateway declaration transmission and modifies its status to member (M).

3.3 Checking phase

Thanks to clusterheads and gateways declaration, we aim to form a connected dominating set. In fact, every node will either a member node attached to a clusterhead, or a dominating node assuming a clusterhead or gateway role. In order to avoid any negative impact on the other protocol exploiting the virtual backbone resulting from the clustering process, we must insure that the set formed by gateways and clusterheads is connected. However, the gateway declaration step following clusterhead election one is insufficient in some cases due to information loss. Indeed, the risk of packet loss due to possible cbeacon collision can prevent the member nodes from being aware of clusterhead existence in their neighborhood. While the eventual collision during the gateway election step avoid the correct receipt of gateway declaration by neighbor clusterheads. We note that an eventual mistake when verifying the eligibility for gateway role by a candGW with anonymous clusterhead can lead to a case of disconnected clusterheads within adjacent layers.

To avoid the case of disconnected clusterheads belonging to adjacent levels, we require a checking phase. This step will be realized by all clusterhead nodes, each one shall verify if it has at least one gateway allowing it to communicate with previous layer's clusterheads. Each clusterhead within a layer i should wait the period required for gateways declaration before being able to verify connectivity. Thus, upon sending its cbeacon, each clusterhead estimates the waiting period P_attente. This period P_attente must take into account the periods TG_{i-1} and TG_i required by any candGW within layers i and i-1 before being able to candidate for gateway role. Moreover, the period P_attente must include the necessary time for the declaration of the maximum number of gateways able to be close to a clusterhead. These gateways may belong the layers i or i-1. We aim to estimate the waiting period in terms of period T_i. Thanks to this, every node will maintain a landmark to initiate a given step during the clustering process. Thus, we force a clusterhead to do this checking step in the beginning of a T_i period. Computing the number of period T_i to wait is done as follows:

$$Nbr\_TiGW = \lceil (CWsize + (NumberMax\_gw/CH * TcandGW)/T_i) \rceil \quad (8)$$

$$T_i = SIFS + CWsize + Txcbeacon + (2 * SIFS) + TxBT \quad (9)$$

where:

TcandGW : Time for transmitting a gateway declaration

NumbreMax_gw/CH : Maximum number of gateways able to be close the a given clusterhead

Nbr_TiGW : Number of periods T_i necessary for election step of gateways close to the given clusterhead

Consequently, a clusterhead within layer i determines the waiting period P_attente following the formula 10, if it belongs to an odd level. While, other ones within an even level will adopt the formula (11). We try to avoid the problems of collision and interference able to occur due simultaneous cbeacon sending by two disconnected CH within adjacent layers.

$$P\_attente = (T_{i+2} - t) + (Nbr\_TiGW * T_i) \tag{10}$$

$$P\_attente = (T_{i+2} - t) + (Nbr\_TiGW * T_i) + (3 * T_i) \tag{11}$$

Where t represents the instant of the given CH election

Upon the P_attente period expires, every CH verifies its connectivity with clusterheads within the previous layer. A a clusterhead, with no intermediate gateway from the previous layer, has to retransmit its cbeacon after a random delay backoff to oblige their neighbours from the previous layer to react. In the beginning of the following period T_i, gateways from the previous layer, which receive the cbeacon, must send again their declaration. They defer their transmission by choosing a random delay selected in the range of 0 to $D_2/2$. Member nodes from the previous layer gain the candGW status. But, they select their random delay in the range of $D_2/2$ to D_2. In this way, we allow the gateways to manifest first minimizing consequently the number of additional gateway declaration. This gateway election step is submitted to the same rules as the normal gateway declaration phase. The checking period Tphase_verif includes a period T_i for the cbeacon retransmission by disconnected clusterheads and a period for gateways and additional candGW declarations. Details are omitted here.

3.4 Maintenance

Providing rapidly a connected dominating set able to be deployed by others upper-protocols is one of the main characteristics of TBCA. This advantage allows us to suspend sending data traffic before starting the clustering process. This temporary data traffic pause allows us to quickly finalize the clustering process. In this way, we avoid also eventual collisions between data packets and control packets exchanged during the clustering process. These eventual collisions can cause loss of information and useless retransmissions and prevent the good progress of the clustering process. We note that any node which has already participate in the clustering process is able to determine the number of T_i period to wait before processing the suspending data packets. Details are omitted here for length limitation.

In order to take onto account up-to-date information, we relinquish the clustering algorithm periodically in order to reconstruct a new CDS. To avoid the construct the overall CDS from scratch and to enhance stability of the clustering infrastructure, we add another stability mechanism. This latest aims to permit re-election of older clusterheads, when they have yet enough resources to assure additional tasks. Besides, we limit the maximal number of consecutive clustering periods for cluster-

head re-election in order to rotate the clusterhead role equitably between all nodes. In this context, we define two other boundaries D_{1pmin} and D_{1npmin}. During the first execution of the clustering process, all candCH nodes have the same opportunity to become clusterhead, since they use the rule (1). During subsequent clustering phases, a candCH node computes its personalized boundary D_1^i following the rule below, if it was a clusterhead during the previous clustering period and it does not exceed the maximal number of clustering period as clusterhead :

$$D_1^i = \lceil ((\frac{(D_{1pmin} - D_{1max})}{E_{max}}) * E_i) + D_{1max} \rceil \quad (12)$$

Other candCH nodes use the rule (1) and they must ensure that d_1 is greater or equal to the new boundary D_{1pmin}. In this way, we give the advantage to older clusterheads able to assure coordinator responsibilities. We guarantee also a certain degree of stability in the CDS structure. To rotate the responsibility among all nodes fairly, we limit the number of successive clustering periods for a node as clusterhead. In this context, we impose to an older clusterhead which has lost this privilege to use the following rule to compute D_1^i, the value of d_1 must be greater or equal to the boundary D_{1min}.

$$D_1^i = \lceil ((\frac{(D_{1npmin} - D_{1max})}{E_{max}}) * E_i) + D_{1max} \rceil \quad (13)$$

4 Performance evaluation

Constructing a CDS with low communication and computation costs is the main advantage of our new algorithm TBCA. Besides, it provides rapidly a virtual infrastructure able to support other communication algorithms. To show the effectiveness of our algorithm, we conduct exhaustive simulations over several static and mobile topologies. We compare performances of our algorithm against SPAN performances [2]. Similar to our algorithm, SPAN operates at MAC level in a synchronous environment. But, it relies on a periodic exchange of hello message for maintaining the connected dominating set. The two mechanisms were implemented under the J-SIM simulator. During the evaluation phase, we focus on three essential factors which have a direct impact on the performance of the clustering algorithm and consequently on the network performance. The first criterion is the size of the generated connected dominating. Then, we tried to show the impact of periodic control packet exchange on network performance. Finally, we evaluate the impact of mobility on the performances of the two algorithms. All simulations are allowed to run for 300 seconds and the results are averaged over several simulation runs. During all conducted simulations, we assume that every node has a maximum transmission range, which is the same for all nodes in the network. The transmission range is fixed to 250 meters. Any two nodes are considered as neighbors if there are within the maximum transmission range of each other.

a) The size of the connected dominating set

During this first phase, we compare the size of the connected dominating set produced by SPAN and TBCA while varying the node density and the size of the simulation area. The exchange of hello messages is done every 10 seconds in SPAN mechanism. The same interlude is adopted by TBCA for relinquishing periodically the clustering algorithm. In the first step, simulations are done with different topologies chosen randomly in an area of $500 \times 500m$. Figure (4.a) shows that the size of the dominating set generated by TBCA is smaller when of the maximal boundaries D_1 and D_2 are equal to 31. We recall that these boundaries are used during the clusterhead election phase and the gateway declaration phase. When these boundaries are large, the probability of collision occurrence between cbeacon messages and gateway declarations decreases, since the probability of choosing the same backoff by candidate nodes decreases consequently. We notice also a little difference in the size of generated CDS by the two mechanisms. Our mechanism TBCA produced small CDS, despite the lack of neighborhood knowledge.

During the second step, we choose different topologies with variable size in an area of $1000 \times 1000m$. In this stage, we fix the boundaries D_1 and D_2 to 31, since we deal with large-scale topologies and the probability of collision occurrence increases with density. The results exposed in figure (4.b) confirm that TBCA outperforms by far SPAN.

In this phase, we proved by simulations that density has no impact on the performance of our mechanism, since it keeps the size of the dominating set as low as possible without requiring any periodic control message exchange for gathering neighborhood information. Involving more nodes in routing contribute to deplete their energy, this can cause network partitioning.

b) Impact of the additional overhead dedicated to clustering

To illustrate the impact of the additional overhead introduced for clustering maintenance purpose on the network performance, we opt for geographic forwarding. We assume that nodes can obtain the geographic position of others neighbors through a low-power Global Position System (GPS) receiver or through some other ways. The source verifies if the destination is in its neighborhood. In such case, the source sends directly the data packet to the destination. Otherwise, it determines the closest

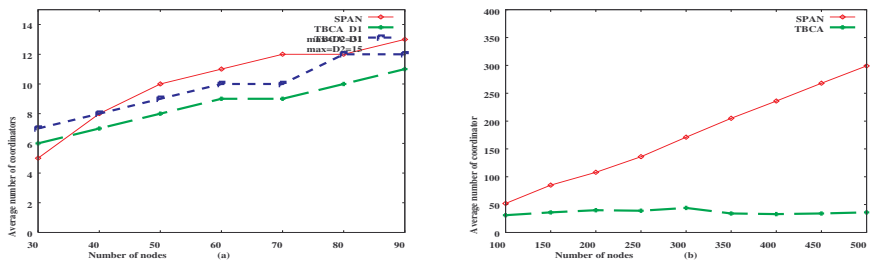

Fig. 4 Average number of coordinator

neighbor coordinator to the destination and sends to it the packet. The coordinator also operates in the same manner. We note here, that forwarding is done through coordinator nodes only. We choose ten sources which send packets to ten destinations. Sources and destinations are chosen at the boundary of the simulation area in order to allow multi hop communication. Each CBR flow includes 512 bytes packets.

During the first step, we choose different random topologies of 100 nodes confined in a square area of $500 \times 500m$. The periodicity for clustering update is fixed to 5 beacon intervals. We vary the traffic load in order to evaluate the impact of the periodic exchange of control messages on data traffic. Figure (5. a) illustrates that the number of delivered packets using our clustering mechanism is higher than the number measured when using SPAN, since we do not require periodic control messages which throttle the actual data traffic. The figure (5. b) exposes the average sojourn delay measured during all simulations, we notice that the sojourn delay increases with the data traffic load for both mechanisms. However, the sojourn delay measured for SPAN is always greater than our algorithm delay, since the periodic hello messages contend with data traffic.

The figure (6.a) consolidates this result, since it exposes the number of delivered packets while varying the updating interval used for periodic exchange of hello

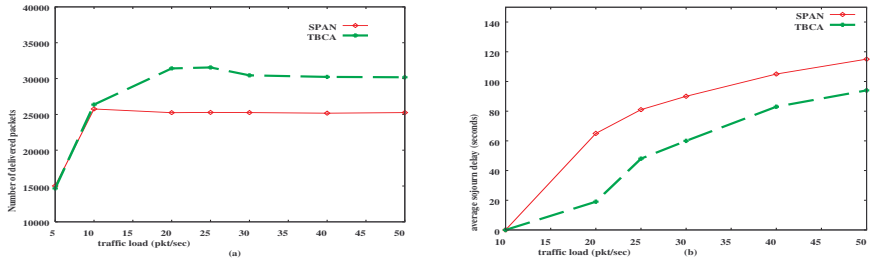

Fig. 5 (a) Average number of delivered packets regarding variable traffic load, (b) Average sojourn delay regarding variable traffic load

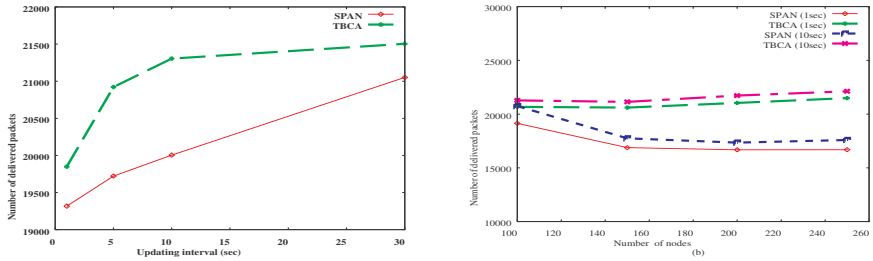

Fig. 6 (a) Average number of delivered packets regarding variable updating interval, (b) Average number of delivered packets regarding variable updating interval and variable number of nodes

message. In these simulations, every source injects 20 packets per second during the 200 seconds. We remark that performance of SPAN improves as the updating interval becomes longer. However, our algorithm still outperforms it.

During the second step, we simulate different topologies with variable size in a fixed square region of $500 \times 500m$ to evaluate the impact of density on performance of the network. Every CBR flow sends 50 packets per second. In these simulations, we vary also the updating interval for exchanging hello messages for SPAN and relinquishing clustering process for TBCA. We notice through results illustrated in figure (6. b) that TBCA is unaffected by the variation of density or updating interval. While the performances of SPAN degrade when the number of nodes increases, since the control overhead of hello messages increases also.

c) Impact of mobility

In the second phase, we consider random topologies of 100 mobiles nodes. Only the sources and destinations are stationary. To evaluate the impact of mobility, we measure the number of delivered packets under different speeds. The figure (7.a) shows that the performances of SPAN degrade when the updating interval for clustering maintenance increases. In fact, gathering 2 hop neighborhood information takes time, so the collected information may be not up to date when be used by nodes. We remark also that the performances of SPAN are badly in highly dynamic environment, particularly when the updating interval is large, since routing information used for routing is not up to date. In the figure (7. b), we expose the results provided by our mechanism. Despite the decrease in the number of delivered packets, we still outperform the SPAN mechanism. In fact, at the beginning of the updating interval, we relinquish the clustering process in order to build a new connected dominating set based on the actual state of the network (position, energy). Thanks to this, other upper-layer protocols can be implemented efficiently on top of the virtual backbone.

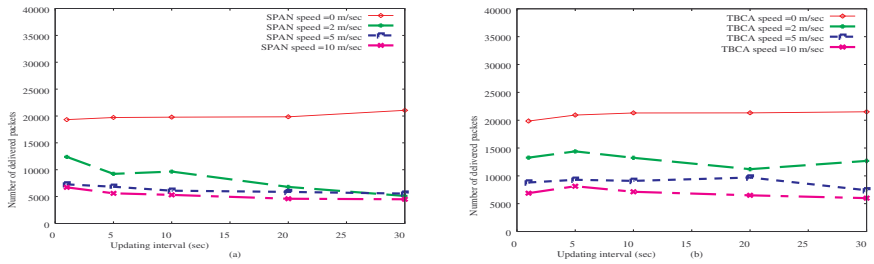

Fig. 7 (a) Average number of delivered packets regarding variable updating interval for SPAN, (b) Average number of delivered packets regarding variable updating interval for TBCA

5 Conclusion

In this paper, we detail the three phases of our clustering algorithm TBCA. Our goal is to construct a connected dominating set while taking into consideration the eventual occurrence of collision. Besides, we exploit this situation for doing the clustering process in an organized layered manner. No neighborhood knowledge is needed for node to be able to decide locally on the role to take. Conducted simulations show first that our algorithm outperforms by far other clustering techniques for CDS construction in terms of the average number of delivered packets and per-packet sojourn delay, especially in case of heavy loaded networks. Moreover, we notice that it copes better with mobility.

References

1. Das, B., Sivakumar, R., Bhargavan, V.: Routing in ad hoc networks using a spine. Proceedings of ICCCN. pp. 1-20 (1997).
2. Chen, B., Jamieson, K. Balakrishnan, H., Morris, R.: Span: an energy efficient coordination algorithm for topology maintenance in ad hoc wireless networks. ACM Wireless Networks J. 8 (5). pp. 481-494 (2002).
3. Belghith, A. Jemili, I., Mosbah, M. : A Distributed Clustering Algorithm without an Explicit Neighborhood Knowledge. International Journal of Computing and Information Sciences, Vol. 5, number 1, pp. 24-34 (2007).
4. Jemili, I., Belghith, A., Mosbah, M. : Algorithme Distribu de Clusterisation sans connaissance du voisinage : principe et valuation. Proceeding NOTERE'2007, Marrakech, June (2007).
5. J. Wu, H..L. Li (1999) On calculating connected dominating set for efficient routing in ad hoc wireless networks. Proceedings of the 3rd ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications. PP 7-14.
6. I. Stojmenovic, M. Seddigh and J. Zunic (2001) Dominating sets and neighbor elimination-based broadcastings in wireless networks. IEEE Transactions on Parallel and Distributed Systems, 13(1):14-25, January 2001.
7. J. Wu, M. Gao, I. Stojmenovic (2002) On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. Journal of Communication and Networks, March 2002.
8. Wan, P.-J., Alzoubi, K., Frieder, O. : Distributed construction of connected dominating set in wireless ad hoc networks. In IEEE INFOCOM (2002).
9. Theoleyre, F., Valois, F. : A self-organization structure for hybrid networks, Ad Hoc. Netw. (2007). doi:10.1016/j.adhoc.2007.02.013
10. Kuhn, F., Moscibroda, T., Wattenhofer, R. : Initializing newly deployed ad hoc and sensor networks, in: 10th Annual International Conference on Mobile Computing and Networking (MOBICOM), (2004).
11. Gandhi, R., Parthasarathy, S. : Distributed algorithms for connected domination in wireless networks. Journal of Parallel and Distributed Computing, Volume 67, Issue 7, pp. 848-862 July (2007).