About the Self-Stabilization of a Virtual Topology for Self-Organization in Ad Hoc Networks

Fabrice Theoleyre, Fabrice Valois

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

About the Self-Stabilization of a Virtual Topology for Self-Organization in Ad Hoc Networks

Fabrice Theoleyre — Fabrice Valois

N° 5650

August 2005

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--5650--FR+ENG

About the Self-Stabilization of a Virtual Topology for Self-Organization in Ad Hoc Networks

Fabrice Theoleyre * , Fabrice Valois * †

Thème COM — Systèmes communicants
Projet ARÈS

Rapport de recherche n° 5650 — August 2005 — 25 pages

Abstract: Ad hoc networks are spontaneous wireless networks without any wired infrastructure, composed of mobile terminals. We assume that nodes must collaborate to set up an efficient network, such a collaboration requiring a self-organization in the network. We proposed a virtual structure to organize the network: the backbone is a connected structure helping to optimize the control traffic flooding. Clusters form services area, hierarchizing the network, electing one leader per cluster. Since the ad hoc topology is volatile, the self-stabilization of the algorithms is vital. The algorithms for both the construction and the maintenance are analytically studied to prove the self-stabilization of the proposed virtual structure. Thus, the virtual structure is efficient and very scalable, a local topology change impacting only locally the virtual structure. Finally, simulations investigate the behavior and the performances of the virtual structure according to several paremeters.

Key-words: self-stabilization, self-organization , virtual structure, ad hoc networks

* Laboratoire CITI, INSA Lyon – INRIA ARES – Email: {firstname.lastname@insa-lyon.fr}

† This research report is the long version of the article *About the Self-Stabilization of a Virtual Topology for Self-Organization in Ad Hoc Networks* presented in the Self-Stabilization Symposium (SSS), 2005 [17]

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Sur l'auto-stabilisation d'une topologie virtuelle pour l'auto-organisation des réseaux ad hoc

Résumé : Les réseaux ad hoc sont des réseaux sans-fil spontanés sans infrastructure fixe, composés de terminaux mobiles. Nous supposons que ces noeuds doivent collaborer afin de mettre en place un réseau performant, une telle collaboration requérant une auto-organisation dans le réseau. Nous avons proposé une structure virtuelle pour organiser le réseau : un backbone est une structure connexe aidant à optimiser le flooding du trafic de contrôle. Des clusters forments des zones de service, hiérarchisant le réseau, en élisant un chef par cluter. Comme les topologies ad hoc sont volatiles, l'auto-stabilisation de ces algorithmes est vitale. Les algorithmes de construction et de maintenance sont ici étudiés analitiquement pour prouver l'auto-stabilisation de la structure virtuelle proposée. Ainsi, la structure virtuelle est performante et passe à l'échelle, un changement local de la topologie n'impactant que localement la topologie virtuelle. Finalement, des simulations permettent d'étudier le copoortement et les performances de la structure virtuelle selon de multiples paramètres.

Mots-clés : auto-stabilisation, auto-organisation, structure virtuelle, réseaux ad hoc

1 Introduction

MANet (Mobile Ad hoc NETworks) are spontaneous topologies of mobile nodes where each of them collaborate in order to give services like routing, localization, etc. It can be used to offer a spontaneous network infrastructure. Each terminal can communicate via wireless links without preconditioned fixed infrastructure [13]. The network must function autonomously, without any human intervention. To send packets from a source to a destination, either the destination is in the radio range of the source or intermediary nodes must help to forward the packets. To reach such a goal, the nodes must collaborate and exchange control information to set up routes in the network. Indeed, each node is both client and router. Because of the nodes mobility, radio links are created and deleted continuously leading to topology changes. And finally, routes are volatile. So, self-adaptation of the network to the dynamicity is a major issue of MANet. Ad hoc networks thanks to their flexibility are promised to a large spectrum of utilization. For example they can be useful to organize rescue operations when a natural cataclysm (earthquake, cyclone...) occurs and all fixed infrastructures are destroyed.

Ad hoc networks can be connected to the Internet, via a dedicated device, the wireless access point (AP). It is a gateway between the wired world and the ad hoc network. Such networks are often called *hybrid networks* constituting *wireless multihops cellular networks*. We think that hybrid networks constitute a natural evolution of access networks. Such networks could be used by telecommunications operator to extend without any additional cost the radio covering area of the cellular networks. Hybrid networks could also be intensively used in house automation networks, interconnecting personal services gateways at home to the Internet and their services providers.

Ad hoc and hybrid networks remain a large scientific domain to study. Classical networking solutions must be re-conceived because of the particular constraints of ad hoc networks: radio links implicate a low bandwidth, radio interferences, links instability creating rapid topology changes, a low reliability and packet losses. Moreover, ad hoc networks are mainly constituted by embedded terminals, presenting constraints in power-energy, CPU, memory... The network must collaborate to find a suitable power-energy saving policy. Several problems remain to be treated: addresses attribution, a solution to secure communications, an efficient interconnection to the Internet, the mobility management, a routing protocol presenting high performances with an acceptable overhead... Finally, the flooding in ad hoc networks presents important problems of reliability, transmissions redundancy and collisions: known as the *broadcast storm* problem [12].

In our point of view, self-organization can answer to the above key problems. Self-organization deals with virtual topologies in order to simplify ad hoc topologies. For example, virtual topologies can be constituted by a backbone [18], or a combination of a backbone and clusters [16]. The goal is to offer control on the MANet. Advantages of virtual topologies are:

- scalability: because MANet are constituted by many nodes, the clusters could group mobile nodes and the backbone could concentrate flooding packets to minimize the broadcast storm problem. So it is possible to provide a routing protocol taking into account such a decomposition: a local routing protocol restricted to the cluster and a global routing protocol using the backbone
- to facilitate the integration of a MANet in wireless/wired networks using the root of the virtual backbone. It can be viewed as a spontaneous wireless extension of wired infrastructures. It offers a natural way to manage hybrid networks
- to offer a framework to implement new services like mobility management, paging and localization services, native multicast support, etc
- to hide nodes neighborhood changes using a top-level view of MANet. A virtual topology stabilizes the neighborhood and simplifies the network topology
- taking into account heterogeneous nodes, a virtual topology can classify *better* nodes as dominators and others as dominatees. Dominators participate actively to the virtual topology in order to minimize the energy consumption for dominatees(e.g.). It gives a hierarchy of node contributions.

The mobility represents a key challenge in MANet. As each node is mobile, many radio links appear and disappear brutally, occurring many topology changes. Virtual structures must remain efficient along the time. Hence, it must be continuously maintained, such that structural constraints hold. The structure must reconstruct or repair itself with a minimal delay. Such a property conduct to the self-stabilization properties.

In this paper, we focus on the demonstration of self-stabilization properties of the virtual structure described in [16]. This article makes two main contributions to the understanding of ad hoc self-organized virtual structures. First, it proves theoretically self-stabilized properties of the backbone and clusters structure. Secondly, it proposes an evaluation of convergence time of the virtual structure construction and repair through simulations.

Next, we will expose related work about self-organized virtual structures in ad hoc networks. Backbone and clusters will be mainly presented. Section 3 presents a few details about the studied virtual structure. Section 4 presents the notations necessary to the comprehension of the article, and results about the complexity of the different algorithms. Section 5 presents an analytical study of the self-stabilized properties for the backbone, and section 6 is dedicated to the clusters. Results are given in section 7 in order to estimate through simulations the convergence time of the structure construction and reconstruction. Finally, we conclude the article and give some perspectives.

2 Related work

Self-organization structures help to organize the network in order to optimize floodings, to hierarchize the network and to structure it. This topic is currently well studied and several articles deal with such a problem.

2.1 Clusters

Clustering consists in grouping nodes geographically close. A clusterhead is often elected per cluster, managing its services area. Each node must be $k_{cluster}$ hops far at most from its clusterhead. Let $N_k(u)$ be the k-neighborhood of u, i.e. the set of nodes at most k hops far from u. Then, formally:

$$\forall u \in V, \exists c \in C \ / \ c \in N_k(u) \ (C = \bigcup Clusterheads) \tag{1}$$

[9] proposes to create overlapping clusters. A node gathers the current list of clusters from neighbors, and executes the algorithm construction. It adds all possible clusters and sort the clusters according to their cardinality. Then, it deletes redundant clusters, privileging clusters of higher size. Finally, it floods the new list in the network. However, the algorithm is quasi-centralized, and the network is supposed as perfectly synchronous. The problem of concurrent changes in the cluster list are not taken into account, creating inconsistences.

The algorithm [10] is the most used algorithm to construct clusters. In the first step, each node initiates a neighborhood discovering. Then, each node decides according to its neighborhood, to become clusterhead or not. The decision is propagated in the neighborhood so that each node which has not chosen any clusterhead yet takes the source as its new clusterhead. The decision could be based on several metrics (node identifier, mobility, location...).

Several maintenance procedures exist. If no clusterhead is required, only the diameter constraint must hold: when a node is 3 hops far or more, it initiates a cluster reconstruction [10]. The algorithm can choose to limit the number of new clusters, or to limit the number of nodes changing their cluster. A node must in consequence have a complete 2-neighborhood knowledge. Moreover, as the mobility is not predictive, an optimal maintenance is a difficult trade-off. When the clusterhead is required, two procedures exist. The first one proposes to maintain always the highest node of the cluster as the clusterhead. It occurs many cluster changes, and is in consequence not suitable. In the second approach [14], a clusterhead remains clusterhead as long as it can. Nodes which lose their connectivity toward their clusterhead choose another clusterhead or become themself their own clusterhead. Changes occur less frequently.

[2] proposes to construct k-clusters, i.e. each node is at most k hops far from its clusterhead. The first step allows to propagate highest id k hops far. k rounds are necessary. The second step allows a clusterhead to know that it has been elected by a node. k rounds are necessary to propagate k hops far the lowest ids of the first step. To form connected clusters, a node chooses as clusterhead the lowest node id heard in the second step. However, no maintenance is given, whereas it represents a key problem in volatile topology environments.

2.2 Backbones

A backbone could be well modeled with a *Minimum Connected Dominating Set* (MCDS): each node must be neighbor of at least one node of the MCDS, and the MCDS is a connected structure, with a minimal cardinality. Formally, in a *Connected Dominating Set* (CDS), the two following conditions hold:

$$\forall u \in V, \exists c \in CDS \ / \ c \in N_1(u) \tag{2}$$

$$CDS\ connected \tag{3}$$

A MCDS is defined by the following additional condition:

$$|MCDS| = min(|CDS|) \tag{4}$$

Usually, the members of an MCDS are called *dominators* and other nodes *dominatees*. We can extend this notion to a k-MCDS: each node is at most k hops far from a node of the k-MCDS, and the k-MCDS is a connected structure with a minimal cardinality.

The construction of an MCDS is a NP-hard problem [11]. Thus, several heuristics were proposed. [7] proposes two centralized algorithms. A dominator is colored black, a dominatee gray and any other node white. The first algorithm chooses to color the gray node which can color the highest number of white nodes. As a gray node is neighbor of a black node, it forms a connected dominating set. The second algorithm chooses the pair of nodes which can color the maximum number of white nodes. The pair must contain at least one gray node.

Many articles propose to construct distributively a CDS in 2 steps. First, a dominating set (DS) is created, where each node is neighbor of a node in the DS. Secondly, the DS is interconnected to form a connected structure. Usually, 4 states exist: dominator (in the CDS) / dominatee (not in the CDS) / active (in election) / idle (waits for the construction). In [3, 4, 1], a leader declares itself dominator and broadcasts its decision. The neighbors of a dominator become its dominatees. The neighbor of dominatees become active. The active nodes with the highest weight in their neighborhood become dominators, and the process keeps on.

With such a scheme for the DS construction, these algorithm construct a CDS where a dominator is at most 2 hops far from another dominator. In [4], the authors propose an iterative exploration. Each dominator which has already chosen a parent explores the dominatee with the highest number of dominator neighbors. This dominatee becomes dominator and explores one random neighbor which is a dominator without parent in the CDS. This explored neighbor chooses the source as new parent and continues the exploration. The exploration requires an important time to converge. [3] proposes a similar approach introducing a timer: a dominatee waits that all its neighbors are either dominatee or dominator. Hence, the cardinality of the CDS is reduced.

[5] follows a similar approach. A node becomes active if it has a dominatee-neighbor. Hence, a dominator is neighbor of a dominatee. A new dominator chooses the dominatee with the highest weight as parent. This parent is forced to become dominator and connects the CDS. However, such a scheme is not adaptable to a k-MCDS as the distance between two dominators exceeds two hops.

In the MIS constructed by [1], a dominator is at most 3 hops far from another dominator. A simple method, less efficient in cardinality, is proposed. Initially, the leader is the only connected dominator. Each connected dominator which has no idle or active node in its neighborhood sends an `invitation` with a TTL=3. A dominator not connected to the CDS sends a `join` to the source, forcing intermediary dominatees to become dominators. In the same way, the new connected dominator sends an `invitation`. The process stops when all the CDS is connected.

To the best of our knowledge, only [19] proposes a localized algorithm. A node is elected as a CDS member if it has 2 disconnected neighbors. Rules for a redundancy elimination are proposed: *a node with a set of 2 connected neighbors of higher id which cover its whole neighborhood becomes dominatee, else it becomes dominator.* This rule could be extended as: *a node with a set of neighbors of higher id forming a CDS and covering its whole neighborhood becomes dominatee, else it becomes dominator.* These rules elect distributively a CDS.

2.3 Self-Stabilization

Self stabilization was first defined by Dijkstra [6]: a system is self-stabilizing when "regardless of its initial state, it is guaranteed to arrive at a legitimate state in a finite number of steps." [15] presents bases of the self-stabilization in the fault tolerance domain. In ad hoc networks, topology changes occur frequently, and could be modeled as temporary faults. In consequence, the self-stabilization properties of an algorithm are essential in the ad hoc networks. Recently, [8] studied a multicast protocol in ad hoc networks. To the best of our knowledge, no prior work was done to study the self-stabilization properties of the Connected Dominating Sets structures in ad hoc networks.

3 The Virtual Structure for Self-Organization

We proposed in [16] a virtual structure for self-organization. This structure helps to structure the network, to optimize floodings, to create a hierarchy... It is constituted by a backbone and clusters. First, a k-neighborhood

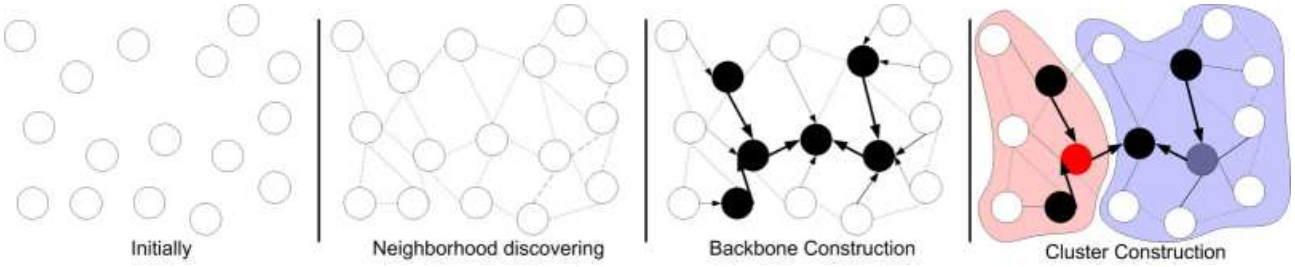

Figure 1: Virtual structure construction

discovering is initiated (fig. 1). Then, the algorithm constructs a k_{cds}-CDS in electing dominators and interconnecting them. Finally, some dominators are elected as clusterheads such than $k_{cluster}$-clusters are formed. We propose to study the self-stabilization properties of this structure.

3.1 Metric

A key property of virtual structures is their stability: for example, a backbone must be stable so that it constitutes an efficient routing cache, or allows power-energy saving for backbone clients. Hence, we proposed a weight combining several criteria to elect suitable backbone members or clusterheads:

- $\mathcal{M}$ (mobility): a node must have a low relative mobility so that its clients remain in its radio range. However, a GPS represents according to us an unacceptable cost and is inefficient for indoor environments. Hence, a node will simply monitor the changes in its neighborhood: it constitutes the single impact which is interesting
- $\mathcal{D}$ (degree): a node with a too high degree will constitute a bottleneck. Oppositely, a too low degree requires more backbone members. Hence, a difference to an optimal degree is computed
- $\mathcal{E}$ (energy): a backbone member with low energy batteries will die shortly. Hence, this metric discriminates nodes with too low energy reserves

These criteria are combined in a non-linear metric:

$$Weight = \mathcal{E}(\alpha \cdot \mathcal{M} + \beta \cdot \mathcal{D}) \qquad with\ \alpha \gg \beta$$

3.2 Backbone

A backbone could be very efficient in an ad hoc network:

- The virtual backbone constitutes a natural prolongation of the wired backbone
- The backbone helps to select privileged nodes to forward flooding packets
- The backbone helps to create a hierarchy: backbone members versus backbone clients

3.2.1 Construction

The following states exist: dominator / dominatee / active (in election) / idle (initial state). The Access Point (AP) acts as leader and becomes the first dominator. It propagates its new state k_{cds} hops far contained in an `hello` packet. The following rules are applied when a node receives an `hello`:

1. An active or idle node which receives an `hello` from a dominator D, k_{cds} hops far, becomes dominatee and chooses D as parent
2. An idle node which receives an `hello` from a dominatee D, k_{cds} hops far, becomes active and triggers a timer of $\Delta_{election}$ seconds. $\Delta_{election}$ is the maximal round-trip-time to a farthest k_{cds}-neighbor.
3. After the timer, a node which owns the highest weight among its active k_{cds}-neighbors becomes dominator. After $max_{election}$ unsuccessful elections, an active node becomes automatically dominator to overpass the problems of mobility or of inconsistency in the neighborhood tables. We can remark that a dominator has no parent during this phase.

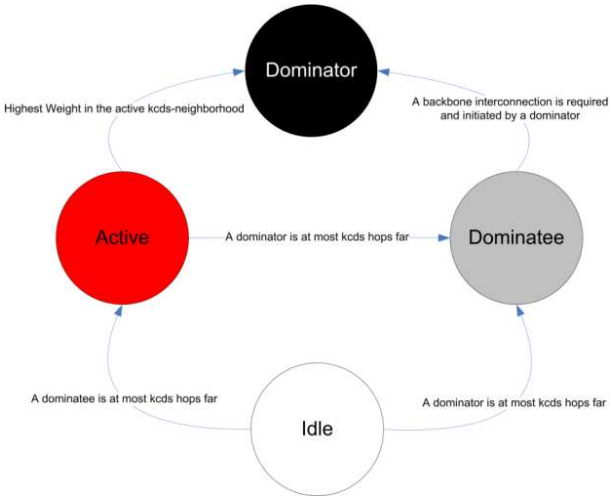

Figure 2: Possible state changes transitions for the construction of the backbone

This step constructs a k_{cds}-dominating set. The state changes follow the state-machine described in figure 2. The algorithm creates waves of elections: a node is elected dominator, its k_{cds}-neighbors become dominatees, and the k_{cds}-neighbors of dominatees become active. An election occurs, one or several active nodes are elected dominators. Waves stop when the network boundaries are reached.

The interconnection is inspired from [1]: the leader sends a `cds-invite`, with a TTL $2 \cdot k_{cds} + 1$. A dominator without parent chooses the source as new parent and sends a `cds-join` along the inverse route. Each intermediary dominatee becomes dominator and sets its parent as the next hop in the route. A dominator which sent a `cds-join` can send a `cds-invite` for other dominators in its $(2k_{cds} + 1)$-neighborhood. The dominators form finally a k_{cds}-CDS structure. The interconnection algorithm, starting from the root, expands a tree, spanning all dominators. In a normal execution, during the pseudo-round i, all dominators at most $i(2k_{cds} + 1)$ hops far from the root are interconnected to the tree.

3.2.2 Maintenance

A node sends periodically `hellos` containing its id, its weight, its cds-state, its parent in the CDS and the ids of its 1-neighbors. These `hellos` being forwarded k_{cds} hops along, each node has a complete knowledge of its k_{cds}-neighborhood. Hence, each dominatee can verify that its parent is still valid: it is at most k_{cds} hops far, is dominator, and there exists a dominatee neighbor having the same parent and being nearer of this parent (to force connectivity of the cds-dominance area).

The backbone must remain connected. Hence, the AP sends periodically `ap-hellos`, forwarded only by dominators. When a dominator receives an `ap-hello` from its parent, it considers itself connected and forwards the `ap-hello`. When a dominator misses several `ap-hellos`, it considers itself disconnected and engages a cds-reconnection procedure. It sends a `cds-reconnect` in broadcast with the id of the last `ap-hello` heard. This packet is forwarded in broadcast by its dominatee, and in unicast toward their parent for other dominatee. Any dominator with a valid parent and having received an `ap-hello` with an *ap-hello id* strictly superior to the id required by the source is authorized to send a `cds-reply`. Hence, we avoid loops in the reconstruction: a dominator can not reconnect itself to one of its descendant. The `cds-reply` is forwarded along the inverse route. Finally, the disconnected dominator chooses the best reconnection candidate (according to the path length, to the weight of the source of the `cds-reply`...). It sends a `cds-accept`, forcing intermediary dominatees to become dominators. The backbone is reconnected, potentially reconnecting the whole branch.

To avoid a constant growth in the size of the backbone, we propose a mechanism to eliminate redundancy. A dominator is *useless* if it has no dominatee at exactly k_{cds} hops and no dominator for which it is a parent. An useless dominator, U, sends a `useless-advertisement` in broadcast with a TTL of k_{cds} and becomes dominatee, without changing its parent. Its dominatees choose the parent of U as new parent, and forward the packet in broadcast, after having decremented the TTL of the packet. We can remark that the `useless-advertisement` are only flooded in the zone of dominatees which have chosen the useless dominator as parent. The overhead remains negligible.

If many reconnections occur in the backbone, the load on the radio medium could be important. Hence, many collisions occur, disturbing the reconnection process. A dominator which tries many unsuccessful `cds-reconnect` sends a `break` in broadcast and takes the *idle* state. When a node receives a `break` from its parent, it becomes *idle* and forwards the message. Finally, the whole branch becomes *idle*. The idle area waits for an extern solicitation to reconstruct locally the backbone. A dominator which has an *idle* k_{cds}-neighbor sends a `cds-invite`. The idle nodes receiving a `cds-invite` become *active* and store temporarily the packet to connect themself to the backbone if they are elected dominators. However, in the worst case, the idle area can be exactly $k_{cds}+1$ hops far from one dominator. Thus, a dominatee neighbor of its dominator and which has an idle neighbor exactly k_{cds} hops far, forces its dominator to send a `cds-invite` for the reconstruction of the idle area. The reconstruction acts in the same way as the construction.

3.3 Clusters

We propose to construct services areas, i.e. clusters, useful in many manners:

- Clusters structure the network in creating a hierarchy
- A clusterhead is elected to manage one services area
- Zones are useful for hierarchical routing, for localization...

3.3.1 Construction

As the backbone was constructed during the first phase, we use naturally it for the cluster construction. Only dominators participate to the election, reducing the overhead. Moreover, a clusterhead is forced to be dominator: a clusterhead will use further the backbone to optimize the floodings.

During the construction, each dominator begins to send periodically `cluster-hellos` when all its neighborhood has either the dominator or the dominatee state. `cluster-hellos` contain the address of the source and its weight. Theses packets are forwarded $k_{cluster} - k_{cds}$ hops along, uniquely by virtual neighbors. A virtual neighbor of N is either a parent of N in the CDS, or a child (a node for which N is a parent). A `cluster-hello` is forwarded only if it comes from a parent or a child in the CDS. A node is elected clusterhead if it has the highest weight among all its $k_{cluster} - k_{cds}$-virtual neighbors without clusterhead. An elected clusterhead sends a gratuitous `cluster-hello` to advertise its decision. A dominator without clusterhead chooses the source of the `cluster-hello` as clusterhead if the previous hop has also chosen this clusterhead, and if the clusterhead is at most $k_{cluster} - k_{cds}$ hops far. Such a condition forces the construction of connected clusters. Since dominatees are at most k_{cds} hops far from their parent, the algorithm constructs clusters of radius $k_{cluster}$.

3.3.2 Maintenance

`cluster-hellos` are not yet required for the maintenance. However, each node adds in its `hellos` its clusterhead, the relay and the distance toward it. Hence, each dominator can easily verify that its clusterhead is valid, i.e. a virtual neighbor has the same clusterhead and is at most $k_{cluster} - k_{cds} - 1$ hops far from its clusterhead.

If the relay of a node N announces a different clusterhead, at a distance at most $k_{cluster} - k_{cds} - 1$, N takes the clusterhead of its relay.

A node A must verify the validity of its clusterhead C:

- a neighbor B of A announces in its `hello` that it chose C as clusterhead, and that its distance to C is $\mathcal{H}$. To be valid, $\mathcal{H}$ must be inferior strictly to $k_{cluster} - k_{cds}$
- A will in consequence send in its next `hellos` its clusterhead id. C and its distance to its clusterhead $\mathcal{H}+1$. A inscribes B as its relay to its clusterhead.

If a node A loses its clusterhead C_{old}, i.e. C_{old} is no more valid, it searches a new candidate:

- A neighbor has a clusterhead C_{new} and its distance is at most $k_{cluster} - k_{cds} - 1$
- A neighbor B has the clusterhead C_{old}, its distance is at most $k_{cluster} - k_{cds} - 1$ and B is not the relay for A toward C_{old}. Such a case can seldom occur (e.g. when a backbone reconnection occurred, modifying the backbone topology and the virtual neighborhood) and avoids the count-to-infinity problem.

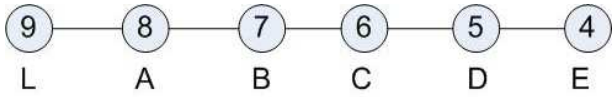

Figure 3: A linear network of strictly decreasing weight

When a node changes its clusterhead, it sends immediately a gratuitous `hello` to force other nodes to change potentially their own clusterhead. In this way, the convergence delay is reduced.

We propose a procedure to eliminate redundancy. If a clusterhead has no virtual neighbor having chosen it as clusterhead, the node is an useless clusterhead. Since a cluster is connected, no other node has a fortiori chosen it as clusterhead. A useless clusterhead tries to find a new valid clusterhead to become a *normal* node.

4 Preliminaries

4.1 Notations

To study the ad hoc networks, we use the graph theory: a node in the network is represented by a vertex, and there exists one edge from one vertex to another iif there exists a radio link between the two nodes. Since we use only bidirectional links, we study undirected graphs. We note G(V,E) the graph, V being the set of vertices and E the set of edges. We assume that the graph is connected. We use the following notations:

- n: the cardinality of the network ($= |V|$)
- D: the set of dominators: $|D|$ is the CDS cardinality
- $N_k(u)$: the k-neighborhood of u
- $\Delta_k(u)$: the number of k-neighbors ($\Delta_k(u) = |N_k(u)|$), i.e. the number of nodes at most k hops far. By convention, $\Delta_1(u) = \Delta(u)$
- $\Delta'_k(u)$: the number of k-virtual-neighbors. A virtual neighbor of N is either the parent or a child of N the CDS. We can remark that $\Delta'_k(u) \leq \Delta_k(u)$
- $w(u)$: the weight of the node u
- $d(u, v)$: the distance in hops from u to v
- h_T: the height of the tree T, the maximal distance from one node to the root of T
- $dominator(u)$: is the parent chosen by u, u being a dominatee. $dominator(u) \in N_{k_{cds}}(u)$
- $parent(u)$: is the parent chosen by u, u being a dominator. $parent(u) \in N(u)$

4.2 Complexity

4.2.1 Backbone

Construction To elect a k_{cds}-dominating set, any node changes at most 2 times its state:

- idle → active → dominator : any dominator being elected
- idle → dominatee : any node becoming dominatee, one of their k_{cds}-neighbor being dominator
- idle → active → dominatee : any node entering in election, but not becoming dominator since another k_{cds}-neighbor has an higher weight and has become dominator

For each state-change, a node sends a packet, forwarded by the $k_{cds} - 1$-neighbors. The message complexity of the first phase is $O(n \cdot \Delta_{k_{cds}-1}) = O(n^2)$

The time complexity is extracted from the linear network of strictly decreasing weight case (fig. 3), presenting the worst case. The leader L is the left extremity. As the network is not synchronized, the time required for an election is called a *pseudo-round*. During the first pseudo-round, the k_{cds}-neighbors of the leader become

dominatees and all the nodes between k_{cds}+1 and 2.k_{cds}+1 hops far from L become *active*. In the worst case, the node k_{cds}+1 hops far from the leader becomes dominator. During each pseudo-round, k_{cds} nodes become *dominatees* and one become *dominator*. Thus, the time complexity is $O(\left\lceil \frac{n}{k_{cds}+1} \right\rceil) = O(n))$

During the interconnection, each *dominator* sends one `join-invite` and one `join-reply`. In the same way, each *dominatee* forwards one `join-invite` with TTL=x if it already forwarded at most nb_{max}-1 `join-invite` with a TTL superior or equal to x. The maximum TTL of a `join-invite` is $2 \cdot k_{cds} + 1$. Thus, a *dominatee* forwards at most $nb_{max} \cdot (2 \cdot k_{cds}+1)$ `join-invites`. The message complexity is $O(n \cdot (nb_{max} \cdot (2 \cdot k_{cds}+1)) = O(n)$

The worst case for the time complexity is the linear network (fig. 3). During a pseudo-round, any dominators binds itself to the tree if possible, i.e. it is at most $(2 \cdot k_{cds} + 1)$ hops far from at least one connected *dominator*. In a linear network, only one dominator connects itself to the tree per pseudo-round. Dominators being at least $(k_{cds} + 1)$ hops interspaced, the time complexity is $O(\left\lceil \frac{n}{k_{cds}+1} \right\rceil) = O(n)$

Maintenance Details are not given here, but could be deducted from the section 3.2.2. The following complexities hold for the backbone maintenance :

	Messages	Time
`hellos`	$O(n \cdot \Delta_{k_{cds}-1}) = O(n^2)$	$O(k_{cds} - 1)$
`ap-hellos`	$O(D)$	$O(h_{CDS})$
Reconnection of a dominatee	$O(1)$	$O(1)$
Reconnection of a dominator	$O(3 \cdot \Delta_{2 \cdot k_{cds}+1})$	$O(k_{cds})$
Useless dominator	$O(\Delta_{k_{cds}-1})$	$O(k_{cds} - 1)$
break of a branch	$O(n)$	$O(h_{branch})$

4.2.2 Clusters

Construction Let C be the set of clusterheads and D be the set of dominators. Each dominator sends initially a `cluster-hello` to discover its neighborhood and one `cluster-hello` to notify its choice of clusterhead. Each of these packet is forwarded by all dominators $k_{cluster} - k_{cds}$ hops along the CDS. Thus, the message complexity is $O(\Delta'_{k_{cluster}-k_{cds}})$.

The construction time is highly dominated by election. The propagation and treatment time of a packet is negligible. The worst case is a linear chain of dominators with increasing weight. During a pseudo-round, only one dominator becomes clusterhead, and all the dominators at most $k_{cluster} - k_{cds}$ become its clients. Thus the time complexity is $O(\frac{|D|}{k_{cluster}-k_{cds}})$.

Maintenance The maintenance complexity is largely reduced compared to the construction complexity. Ad hoc networks presenting a volatile topology, the maintenance complexity is the more important part.

A disconnected dominator uses the information contained in the `hello`, and more specifically the fields *clusterhead*, *clusterhead-distance* and *clusterhead-relay*. Hence, the additional message complexity is null, integrated in the backbone maintenance. The time complexity is $O(1)$, the decision being immediate: either chooses a new clusterhead or becomes its own clusterhead.

5 Backbone Self-Stabilization

Ad hoc networks presenting a volatile topology, the virtual structure must adapt itself to changes. We present here and in the following section results about self-stabilization of the virtual structure presented in section 3. The construction algorithms converge in a finite time. In the same way, the maintenance algorithms form a valid virtual structure if the number of topology changes (edge/vertex addition or deletion) is finite and sufficiently inter-spaced. We assume that the graph associated to the ad hoc network is connected.

Hypothesis 1 *We assume that the radio topology is stable after a list of changes, constituted by a sum of elementary topology change (vertex/edge deletion/addition). The inter-changes time is sufficient to let the construction or maintenance algorithm converges.*

If during the construction, not enough time is sufficient to let the structure converge because of unknown reason, the algorithm will converge during the maintenance step. Effectively, the backbone is not required to be valid before the maintenance occurs. However, the convergence time will be longer. We can remark that an arbitrary very short time before two topology changes mean that the topology is highly volatile. In this case,

an algorithm has to change the virtual topology every x units of time, x being arbitrarily small. According to us, such an algorithm can't act distributively with an acceptable overhead.

The network being supposed unreliable, some packets can suffer from collisions. Each packet collision can be modeled as an edge deletion in the communication graph. However, the graph is required to be connected for the convergence of our algorithms. In consequence, as can be remarked through the proofs further, the unique condition on reliability for the convergence of the construction is that the graph, even through the edge deletion because of collisions, must remain connected. The condition for the maintenance algorithms is weaker. Effectively, as can be noted through the proofs of self-stabilization properties further, the maintenance procedures are periodical. Hence, a packet collision will just delay the convergence reconstruction. More time will be required, but the structure will be valid after a time proportional to the number of maximal possible successive collisions.

We propose here to demonstrate that the construction algorithm provides a k_{cds}-Connected Dominating Set (CDS). First we demonstrate that the backbone forms a k_{cds}-Dominating Set: each node is at most k_{cds} hops far from one backbone member. Then, the backbone is proven to form a connected structure, showing the construction of a k_{cds}-CDS. Finally, we prove in addition that the backbone forms a tree. Same conclusions are given for the maintenance algorithms.

5.1 Construction

5.1.1 Creation of a k_{cds}-dominating set

Theorem 1 *The algorithm of the first phase terminates and forms a k_{cds}-dominating set*

Lemma 1 *Every vertex has either the dominator or the dominatee state at the end of the first step.*

Proof: Let separate the problem in 2 cases:

• Let assume that an idle vertex I exists, and that there exists another not-idle vertex N in the connected component including N. let $c = \langle I, c_1, c_2, ..., c_k, N\rangle$ be a path from I to N. All the k_{cds}-neighbors of I are idle, else I would have change its state. Thus, $\{c_j\}_{j\in[1..k_{cds}]}$ are idle. In the same way, we obtain the recurrence formula:

$$\forall i, \{c_{i\cdot k_{cds}+j}\}_{j\in[1..k_{cds}]}\ idle \Rightarrow \{c_{(i+1)\cdot k_{cds}+j}\}_{j\in[1..k_{cds}]}\ idle$$

In consequence, N must be idle. The connected component is only constituted by idle vertices. However, at least the leader is not idle. This leads to a contradiction.

• Let assume that a vertex N is active. If a k_{cds}-neighbor is dominator, N would be dominatee. In the same way, if all the k_{cds}-neighbors are dominatees, N would be dominator. If N is the active node of highest weight in its k_{cds}-neighborhood, then N would be elected dominator after $\Delta_{election}$ seconds at most.So, there exists an active A_1, at most k_{cds} hops far, with an higher weight than N.

Let A_k be the graph so that its vertices are the active vertices of G during the k^{th} round, and so that there exists an edge from a vertex a_i to a vertex a_j if and only if $w(a_i) < w(a_j)$. A_k is acyclic and has a finite cardinality, inferior or equal to n. The second property is trivial, let demonstrate the first property. Let $c = \langle c_0, c_1, ..., c_k\rangle$ be a cycle in A_k. An edge exists from c_i to c_{i+1}, i.e. $w(c_i) < w(c_{i+1})$ with $i \in [1..k-1]$. Transitively, $w(c_0) < w(c_k)$. However, c is a cycle. Thus, an edge exists from c_k to c_0, and $w(c_k) < w(c_0)$, this leads to a contradiction.

The graph A_k contains at least a sink a_k, i.e. a vertex has a null outer degree. After $\Delta_{election}$ seconds, a_k will be elected and become dominator, its k_{cds}-neighbors becoming its dominatees. Let I_k be the set of idle vertices in G during the k^{th} round. During the round k, at least one vertex a_k becomes dominator. So, $a_k \notin A_{k+1} \cup I_{k+1}$. The k_{cds}-neighbors of a_k in $A_k \cup I_k$ become its dominatees. Simultaneously, some vertices are extracted from I_k and added to A_{k+1}. So:

$$\begin{aligned} & |I_k| + |A_k| \geq |I_{k+1}| + |A_{k+1}| + |\{a_k\}| \\ \Rightarrow\ & |I_k| + |A_k| \succ |I_{k+1}| + |A_{k+1}| \\ \Rightarrow\ & |A_n| = |I_n| = 0 \end{aligned}$$

In consequence, the algorithm will converge at the end of the first phase to a graph with no active vertex. ■

Remark 1 *If many topology changes occur (contradicting the hypothesis 1), inconsistent neighborhood tables can appear. In the same way, an hacker with an arbitrary high weight could block the election. In consequence, an active node will become automatically dominator if it was active for more than $\Delta_{election} \cdot max_{election}$ seconds. The algorithm will converge quicker, but more dominators will be elected.*

Lemma 2 *Every vertex is at most k_{cds} hops far from a dominator, or is itself a dominator, i.e. the graph of dominators forms a k_{cds}-dominating set.*

Proof: The proof comes directly from the lemma 1: at the end of the first phase, only dominatees and dominators exist:

- A dominatee changed its state because a dominator is at most k_{cds} hops far (by construction).
- A vertex elected dominator remains dominator.

■

5.1.2 Formation of a k_{cds}-CDS

Theorem 2 *The set of dominators forms at the end of the construction a connected set of k_{cds}-dominating, i.e. a k_{cds}-CDS.*

Property 1 *Let c be a path between 2 dominators D_1 and D_k. c follows the property 1 if it is composed by a set of i dominators, interspaced consecutively from each other by at most $2 \cdot k_{cds}$ dominatees: $\exists c = \langle D_1, d_1, ..., d_j, D_2, d_{j+1}, ..., d_k, D_3, d_{k+1}, ..., D_i \rangle$ such that d_l are dominatees, and such that $d_c(D_i, D_{i+1}) \leq 2 \cdot k_{cds} + 1$.*

Lemma 3 *A path c exists at the end of the first phase of the algorithm which follows the property 1, binding each dominator to the leader $\mathcal{L}$.*

Proof: Let D_k be the set of dominators elected during or before the k round. $D_0 = \{\mathcal{L}\}$. D_0 comprises only one dominator following trivially the property 1.

Let assume that D_k follows the property 1. At the end of the $k-1^{th}$ round, a set S_{k-1} of vertices was elected dominators, such that $S_{k-1} \cup D_{k-1} = D_k$ and $S_{k-1} \cap D_{k-1} = \emptyset$. A node N of S_{k-1} is active during the k-1^{th} round before being elected at the end of the round. Let $c_1 =< N, a_1, ...a_i, d >$ be the path from N to the nearest dominatee d during the round $k-1$. N being active, by construction, $|c_1| \leq k_{cds} + 1$. The $\{a_l\}$ are by definition not dominatees, and are by construction at most k_{cds} hops far from d, a dominatee. In consequence, $\{a_l\}$ are active. Since N will be elected dominator, $\{a_l\}$ will become its dominatees at the end of the round. Let $c_2 =< d, d_1, ...d_i, D >$ be the path from the dominatee d to its parent D. By definition, $D \in D_k$, $|c_2| \leq k_{cds}$, and d_l are dominatees. Since $D \in D_k$, let $c_3 =< D, ..., \mathcal{L} >$ be the path from D to the leader. c_3 follows the property 1. Clearly, the path concatenation $c_1.c_2.c_3$ follows the property 1 at the end of the first phase of the algorithm. ■

Lemma 4 *If the property 1 is respected at the end of the first phase, the algorithm will construct a connected k_{cds}-dominating set.*

Proof: Let $\mathcal{D}_i$ be the set of dominators such that for each dominator D from $\mathcal{D}_i$, the path c from D to the leader, following the property 1 has at most i dominators. $\mathcal{D}_0 = \{\mathcal{L}\}$. $\mathcal{D}_0$ forms a trivial connected k_{cds}-dominating set applied to the vertices dominated by $\mathcal{D}_0$. It will send, according to the construction algorithm, a `join-invite` with a TTL=$2 \cdot k_{cds} + 1$.

Let assume that the set $\mathcal{D}_i$ forms a connected k_{cds}-dominating set. Let a dominator $u \in \mathcal{D}_{i+1}$, and c be the path from u to the leader $\mathcal{L}$, respecting the property 1. $c = \langle u, v_1, ..., v_k, \mathcal{L} \rangle$. From the lemma 3, there exists a dominator v_i from c, at most $2k_{cds}+1$ hops far from u since c respects the property 1. v_i has a path $c' \subset c$ respecting the property 1. Moreover, $v_i \in \mathcal{D}_i$. Thus, v_i will send a `join-invite` with a TTL=$2k_{cds}+1$. u will receive the `join-invite`, and will connect itself to $\mathcal{D}_i$. In consequence, $\mathcal{D}_{i+1}$ forms a connected k_{cds}-dominating set. ■

5.1.3 Formation of a tree

Definition 1 *Let the CDS $\mathcal{G}_{CDS}$ containing all the vertices of G, and such that an edge exists from a vertex u to a vertex v iif v is the parent of u if u is a dominator, or iif v is the relay toward its dominator if u is a dominatee.*

Theorem 3 *$\mathcal{G}_{CDS}$ is a tree.*

Proof: According to the previous definition of $\mathcal{D}_i$, $\mathcal{D}_0 = \{\mathcal{L}\}$ is a trivial tree, formed by a singleton.

Let assume that $\mathcal{D}_i$ forms a tree. $\mathcal{D}_i$ has $|\mathcal{D}_i - 1|$ edges. Let $u \in \mathcal{D}_{i+1}/\mathcal{D}_i$. u will interconnect itself to the CDS thanks to a `join-invite` sent by a dominator from $\mathcal{D}_i$. Let v be this dominator. The path $c = \langle u, u_1, ..., u_k, v\rangle$ has only dominatees, else u choosing the nearest dominator, will not interconnect itself to v. In consequence, dominatees will become dominators. We add to $\mathcal{D}_i$ a branch of k dominatees and one dominator, with k edges from a dominatee to its new parent, and an edge from u to its new parent. Thus, $\mathcal{D}_i \cup \{u\} \cup \{u_i\}_{i\in[1..k]}$ has $|\mathcal{D}_i| - 1 + 1 + k$ edges, i.e. $\left|\mathcal{D}_i \cup \{u\} \cup \{u_i\}_{i\in[1..k]}\right| - 1$ edges. In consequence, $\mathcal{D}_{i+1}$ is a tree.

Let d_i the set of dominatees at at most i hops from their father. When a vertex d_0 to $\mathcal{D}$, the vertex and the edge toward its parent is added. Then, $d_0 \cup \mathcal{D}$ remains a tree.

Let $d_i \cup \mathcal{D}$ be a tree. Let $u \in d_{i+1}$ be a dominatee. u chooses a parent and a relay r toward this parent. r is one hop nearer from its parent, by construction. Thus, $r \in d_i$. Only one vertex and one edge are added. $d_i \cup \mathcal{D}$ is a tree. A dominatee being at most k_{cds} hops far from its dominator, $\bigcup_{i\in[1..k_{cds}]} d_i \cup \mathcal{D} = G$. In conclusion, the CDS forms a tree. ■

Remark 2 *A dominatee d is at most k_{cds} hops far from its dominator. A path $p = \langle d, d_1, ..., d_i, dominator(d)\rangle$ exists, such that all d_i are dominatees and have the same dominator $\forall i \in [1..k-1], dominator(d) = dominator(d_i)$.*

Proof: A dominatee d has a valid dominator $dominator(d)$ if a physical neighbor N exists, such that:

- $dominator(N)$ is in the neighborhood table of d
- N is at most $k_{cds} - 1$ hops far from $dominator(N)$ (this information being known via the `hello` packets)
- N is a bidirectionnal neighbor of d (redundant with the previous condition, but minimizing the impact of neighborhood tables incoherences)
- $dominator(d) = dominator(N)$

The remark follows trivially. ■

5.2 Maintenance

5.2.1 Dominating Set

Theorem 4 *A dominatee has always a dominator, at most k_{cds} hops far, i.e. the CDS forms a k_{cds}-dominating set.*

Proof: Dominatees with a dominator neighbor choose it as parent. This dominator is valid. Let assume that the set of dominatees at most i hops far from their parent have a valid parent. A dominatee at most $i+1$ hops far from its parent has chosen it since it is at most k_{cds} hops far, through another dominatee having chosen the same dominator, but at i hops, with $i < k_{cds}$. Thus, since the parent of dominatees at most i hops far from their parent is valid, each dominatee chooses a valid parent.

A dominatee can have no dominator candidate for reconnection in its neighborhood table, i.e. no neighbor exists having chosen a dominator at most k_{cds}-1 hops far. Such a dominatee becomes active. An active vertex becomes dominatee iif it finds a valid dominator as parent. Active vertices becoming dominators execute the maintenance reserved for dominators. Thus, each dominatee has a dominator at most k_{cds} hops far, and this dominator is reachable through a dominatee with the same dominator, one hop nearer from its parent. ■

5.2.2 Connectivity

Theorem 5 *The set of dominators forms a tree*

Lemma 5 *The set of dominators remains a (connected) tree when the radio topology is stable.*

Proof: Let assume that the topology is stable. Each dominator receives an `ap-hello`, maintaining the source as parent. Let $\mathcal{D}_i$ the set of dominators, i hops far via other dominators from the leader, the root of the CDS. $\mathcal{D}_i$ is supposed connected. The vertices of $\mathcal{D}_{i+1}/\mathcal{D}_i$ choose a parent in $\mathcal{D}_i$ since they receive the `ap-hello` from their parent, and so they are one hop farther from the leader. Thus, $\mathcal{D}_{i+1}$ is connected.

Let assume $\mathcal{D}_i$ has no cycle, E_i be the set of edges of $\mathcal{D}_i$, and V_i be the set of its vertices.We can establish that $|E_i| = |V_i - 1|$. For each vertex of $\mathcal{D}_{i+1}/\mathcal{D}_i$, we add one vertex in E_i and one edge in V_i. So :

$$|E_{i+1}| = |V_i| - 1 + [|V_{i+1}| - |V_i|] = |V_{i+1}| - 1$$

Thus $\mathcal{D}_{i+1}$ is connected, without any cycle. ■

Definition 2 *We consider a dominator u* connected *iif there exists an ascendant path directed from u to the leader $\mathcal{L}$, where the first edge is $(u, parent(u))$, and then constituted by the ascendant path $\langle parent(u), ..., \mathcal{L}\rangle$.*

Remark 3 *We assume that the displacement of a node is finite, and the occurred topology changes are propagated in a finite time among the k_{cds}-neighborhood, before a new change appears. It is clear that if a topology change occurs at the time t, all nodes will have an exact view of the topology at the time $t + \Delta_t$ (see hypothesis 1).*

Lemma 6 *When a dominator of a branch of the CDS reconnects itself, all its ascendants and descendants reconnect themself.*

Proof: If a dominator u reconnects itself, then there exists a valid path $\langle u, ..., \mathcal{L}\rangle$ to the leader. Besides, a descendant or an ascendant v of u has by definition a path $\langle u, ..., v\rangle$. Thus, v has a path $\langle u, ..., v\rangle \cup \langle u, ..., \mathcal{L}\rangle$ to the leader. However, all dominators must potentially change their parent to have a valid directed path to the leader. ■

Lemma 7 *When all dominator of a branch are disconnected, at least one dominator will reconnect itself.*

Proof: Every topology change could be decomposed by an elementary addition/deletion of edges. The addition of an edge in the graph cannot generate a disconnection in the CDS. Let assume that the edge (u, x) was deleted. After a finite time Δ_t, the whole branch, i.e. the descendants of u, will consider itself disconnected. A dominator considers itself disconnected when it missed all `ap-hellos` during Δ_t. Δ_t depends from the interval between two `ap-hellos` and the number of acceptable missed `ap-hellos`. Let v be a dominator descendant of u. u will not forward any `ap-hello` with an id superior to l, id of the last `ap-hello` forwarded before the edge (u, x) broke. Thus, the child of u cannot forward any `ap-hello` with an id superior to l. Recursively, v can neither receive nor forward any `ap-hello`. The dominators of the branch of root u consider themself disconnected, and try to reconnect themself via a dominator forwarding an `ap-hello` with an id superior to l.

At least one dominator finalizes its reconnection, and no cycle is created in the CDS, i.e. v cannot choose to reconnect itself to a descendant of u. Effectively, v asks for an `ap-hello` id higher than the last `ap-hello` forwarded by any descendant of u, as explained above. Let $\mathcal{D}$ be the set of descendant dominators of u, and their dominatees (the disconnected part). $\mathcal{D}$ is a connected component. Let $C = G/\mathcal{D}$. C is also a connected component: let $c \in L$ be a descendant of $\mathcal{L}$. $c \in C$ is by definition not descendant of u. Thus $u \notin \langle c, ..., \mathcal{L}\rangle$, in other words $\langle u, x\rangle$ and $\langle c, ..., \mathcal{L}\rangle$ are disjoint.

Let $\mathcal{N}$ be the set of vertices in C, neighbors of $\mathcal{A}$. $\mathcal{N} \neq \emptyset$: let $u \in \mathcal{D}$. The graph is assumed connected. Thus, a path $p = \langle u, u_1, ..., \mathcal{L}\rangle$ exists with $u \in \mathcal{D}$ and $\mathcal{L} \in C$. $u_i \in p$ exists such that $u_i \in C$ and $u_{i-1} \in \mathcal{D} \cap p$. By definition of $\mathcal{N}$, $u_i \in \mathcal{N}$. Moreover, $dominator(u_{i-1})$ is a dominator of the disconnected branch since u_{i-1} is in $\mathcal{D}$. $dominator(u_i)$ is connected, and is in C. $d(dominant(u_{i-1}), dominator(u_i)) \leq 2 \cdot k_{cds} + 1$. In consequence, a dominator exists in the disconnected branch such that it is at most $2k_{cds} + 1$ hops far from a connected dominator.

Finally, $dominator(u_{i-1})$ will reconnect itself to $dominator(u_i)$ thanks to a `cds-reconnect` with a TTL of $2k_{cds} + 1$. According to the lemma 6, each dominator of $\mathcal{D}$ will reconnect itself and choose a new valid parent with an `ap-hello`. ■

In consequence, we can conclude:

Theorem 6 *When an edge deletion implicates a disconnection in the CDS, the CDS will reconstruct itself and a valid CDS will be created.*

Lemma 8 *If a break of the CDS occurs, the branch is broken and then rebuilt.*

Proof: A dominator sends a `break` when no reconnection is successful. This packet is forwarded by descendants, until it reaches dominatees, leaves of the CDS. The whole branch becomes idle, waiting an exterior signal for the reconstruction.

The idle zone is a connected component of the graph. Since we consider the events as discretes, the set of not idle nodes forms also a connected component, comprising the leader $\mathcal{L}$. Let u be a vertex not idle, neighbor of the idle area. Such a vertex exists for the same reason as the lemma 7. Let i be in the idle zone, and neighbors of u. We have here two cases:

- u is dominator. If u is not connected, it will reconnect itself in a finite time according to the theorem 6. If u is a connected dominator, it will send a `cds-invite` with a TTL=$k_{cds}+1$. Clearly, i will receive this packet.
- u is dominatee. $dominator(u)$ will be in a finite time a connnected dominator for the same reason as above. Let $p = \langle i, u, d_1, ..., d_j, dominator(u)\rangle$ the path from i to $dominator(u)$. p is at most $k_{cds}+1$ hops long, since u is at most k_{cds} hops far from its dominator. All the $d_j, j \in [1..k_{cds}-1]$ are dominatees (remark 2). If p is strictly inferior to $k_{cds}+1$ hops, $dominator(u)$ will have i, an idle node, in its neighborhood table. In consequence, $dominator(u)$ will send a `cds-invite` which will reach i. If i is exactly $k_{cds}+1$ hops long, d_j is neighbor of its dominator and has an idle node i, in its neighborhood table. d_j will send a packet forcing its dominator to send a `cds-invite`, with a TTL=$k_{cds}+1$. i will receive this packet.

A `cds-invite` received by the idle node i triggers the reconstruction of the idle branch. i becomes the leader of the zone. The reconstruction leader i reconnects itself to $dominator(u)$ in sending a `cds-accept`. $dominator(u)$ being by definition connected itself to the leader, i is transitively connected. Following a proof similar to theorem 2, a CDS is reconstructed. ■

Remark 4 *A useless dominator becoming dominatee maintains a CDS being moreover a tree.*

Proof: if a dominator is a leaf of the CDS and does not have any dominatee exactly k_{cds} hops far, it becomes dominator and its parent becomes the new dominator of its dominatees. Thus, the CDS remains connected since the dominator has no descendant dominator in the CDS. In the same way, the CDS remains trivially a k_{cds}-dominating set since its dominatees are at most k_{cds} hops far from their new parent. ■

6 Cluster Self-Stabilization

We propose here to demonstrate that the construction algorithm provides a $k_{cluster}$-Dominating Set. A node is at most $k_{cluster}$ hops far from one central node: the clusterhead. Same conclusion are given for the maintenance algorithm.

6.1 Construction

Theorem 7 *The set of clusterheads constructs a $k_{cluster}$-dominating set, i.e. a $k_{cluster}$-clustering.*

Proof: If a dominator is not a clusterhead, then it chooses a dominator-clusterhead according to the process of neighborhood discovering on the CDS topology. `cluster hellos` are forwarded along the CDS-links, at most $k_{cluster} - k_{cds}$ hops far. So a dominator chooses a clusterhead at most $k_{cluster} - k_{cds} < k_{cluster}$ hops far. Moreover, a dominator chooses as clusterhead the source of a `cluster hellos` only if the previous hop chose also the source as clusterhead. Hence, the cluster is connected.

A dominatee has the same clusterhead as its dominator. Moreover, according to the lemma 2, it is at most k_{cds} hops far from its dominator, itself $k_{cluster} - k_{cds}$ hops far from its clusterhead. Transitively, a dominatee is at most $(k_{cluster} - k_{cds}) + k_{cds} = k_{cluster}$ hops far from its clusterhead. Since a dominatee is connected to its dominator through a path containing at most k_{cds} dominatees having chosen the same dominator, the cluster is connected.

According to the lemma 1, each vertex is either dominator or dominatee. In consequence, any vertex has a clusterhead, at most $k_{cluster}$ hops far. ■

Parameter	Default value
Number of nodes	40
Degree	10
Radio range	300m
Mobility	0 $m.s^{-1}$
$interval_{hello}$	4 seconds
$interval_{ap-hello}$	2 seconds

Table 1: Parameters

6.2 Maintenance

Theorem 8 *The maintenance algorithm maintains a set of clusterheads forming a $k_{cluster}$-dominating set of $\mathcal{G}_{CDS}$[1].*

Lemma 9 *All dominators have in $\mathcal{G}_{CDS}$ a clusterhead at most $k_{cluster}$ hops far.*

Proof: Let $\mathcal{G}'_{CDS}$ be the set of dominators having chosen the vertex C as clusterhead. There exists one edge in $\mathcal{G}'_{CDS}$ from u to v if v is the relay toward the clusterhead for u. We can remark that such edges and vertices own to $\mathcal{G}_{CDS}$. If v is at most H hops far from its clusterhead, u is at most $H+1$ hops far from its own clusterhead in $\mathcal{G}'_{CDS}$ and also in $\mathcal{G}_{CDS}$.

$\mathcal{G}'_{CDS}$ is a tree, i.e. no cycle exists in $\mathcal{G}'_{CDS}$. Let assume the existence of a cycle $\langle u_1, ..., u_k \rangle$. u_i with $i \in [1..k]$ is H_i hops far from C. u_1 is the relay toward the clusterhead of u_k. So $H_k = H_1 + 1$. In the same way, u_{j+1} being the relay of u_j for $j \in [1..k-1]$, $H_j = H_{j+1} + 1 \Rightarrow H_j < H_{j+1}$. Thus, $H_k = H_1 + 1$ and $H_1 < H_k$, this leads to a contradiction. $\mathcal{G}'_{CDS}$ is a tree, a dominator chooses a relay one hop nearer of the clusterhead C.

Let D_i be the set of dominators which set the field *distance to clusterhead* to i in their `hellos`. Any dominator of D_i chooses by construction a relay in D_{i-1}. Let assume that the vertices in D_{i-1} are $i-1$ hops far from C. Thus, the vertices of D_i are i hops far from C. Moreover, $D_0 = C$ and C is 0 hops far from itself. Finally, a dominator is allowed to choose a relay only if this relay is at most $k_{cluster} - kcds$ hops far from its clusterhead. Thus, $D_{k_{cluster}-k_{cds}} = \emptyset$. A dominator has either a clusterhead at most $k_{cluster} - k_{cds}$ hops far, or becomes its own clusterhead.

These results hold whatever topology changes occur. Thus, any dominator chooses a clusterhead at most $k_{cluster} - k_{cds}$ hops far. ■

Lemma 10 *Dominatees are at most $k_{cluster}$ hops far from their clusterhead in $\mathcal{G}_{CDS}$.*

Proof: This result holds for the same reasons as in the theorem 7. ■

7 Performance evaluation

We simulate our solution wit OPNET Modeler 8.1, with the WIFI standard model (300m radio range). The default parameters are 40 nodes and a degree of 10. To have representative results, we conduct series of 20 simulations. The 95% confidence intervals are systematically reported on the figures. We present general performances of the proposed virtual structure. Then, we investigate the behavior of the solution, particularly according to the initial convergence, and to the reaction following a temporary failure. The study on the impact of the mobility was isolated in the last simulations.

General performances Figure 4 presents the general performances of the CDS, without mobility. The cardinality of the backbone is almost stable according to the number of nodes. The virtual structure is stable and scalable according to the network cardinality. Less dominators are required when the number of nodes is lower than a threshold: boundary effects are not negligible. When we increase the maximal distance from one dominatee to its dominator, less dominators are required, which is the expected behavior. We could use an high k_{cds} when the network is quasi static, and a small k_{cds} when the topology is very volatile. We can remark that the CDS presents a high connectivity, higher than 99.5% even with 100 nodes. The connectivity is not 100% since collisions may occur. Then, we investigate the performances of the cluster structure (fig. 5). Same tendancies can be observed for clusters than for the CDS. The cardinality and connectivity are very scalable

[1] $\mathcal{G}_{CDS}$ is defined in def. 1

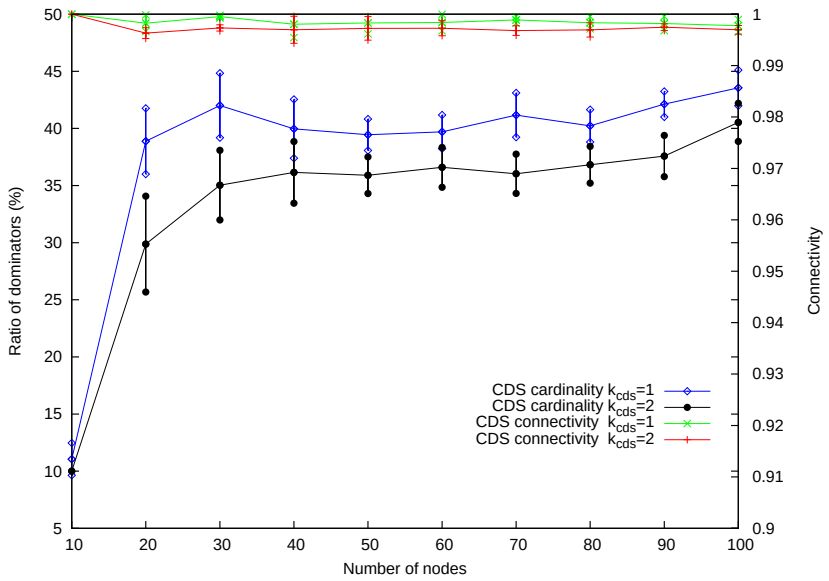

Figure 4: Cardinality and Connectivity of the CDS according to the number of nodes

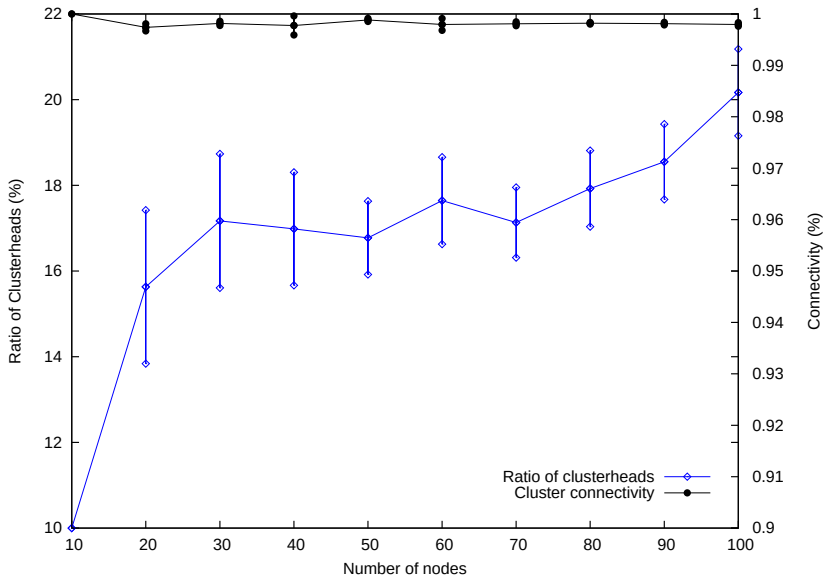

Figure 5: Cardinality and Connectivity of the clusters ($k_{cluster}$=3)

according to the number of nodes. Algorithms for both the CDS and the clusters seem present a good horizontal scalability, i.e. according to the network cardinality.

The impact of the degree was studied (fig. 6). We can remark that the backbone cardinality decreases when the degree increases: less backbone members are required, deserving more dominatees per backbone member. Less dominators are still required when $k_{cds} = 2$ than when $k_{cds} = 1$. The connectivity remains very high, whatever the degree is. For small degrees, the connectivity decreases (but remains over 98%): the network presents less redundancy, reconnections are more difficult. For degree lower than 7, the network being random, it is often disconnected.

Convergence of the construction algorithm We investigate the convergence time of the algorithm for the CDS construction. The clusters are always well-constructed before the end of the CDS construction. In consequence, the clusters are robust and don't represent the more sensitive part of the virtual structure. Thus, no simulation result about the convergence of clusters are given here, the convergence being too fast.

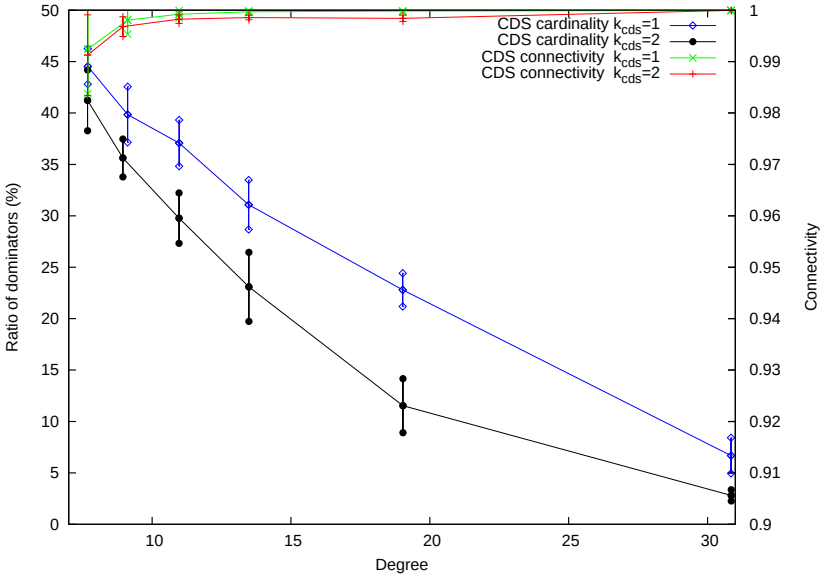

Figure 6: Cardinality and Connectivity of the CDS according to the degree

First, we set k_{cds}=1 and $k_{cluster}$=2 (fig. 7). Approximately 5 seconds are needed to have no idle node in the networks. Two supplementary seconds are necessary for the election, i.e. no active node remains in the network. Finally, less than 10 seconds are necessary to have a CDS *largely connected* or *strictly connected*. *Strictly connected* means that the tree relation (node→parent) creates a valid Connected Dominating Set. For a *largely connected* CDS, we take into account the redundant mesh structure of the CDS. More precisely, we add the following edges:

- $D_1 \rightarrow D_2$: if D_1 and D_2 are dominators and physical neighbors
- $d_1 \rightarrow d_2$: if d_1 and d_2 are dominatees, physical neighbors, and have the same dominator
- $d_1 \rightarrow D_2$: if d_1 is dominatee, D_2 dominator, d_1 and D_2 are physical neighbors, and D_2 is the dominator of d_1

The construction algorithms, executed in parallel for the first and second phases seem efficient: they converge quickly, forming in a few seconds an operational and self-organized ad hoc network. If we set k_{cds}=2 and $k_{cluster}$=3 (fig. 8), the convergence time is longer since the paths necessary to the connection of the Dominating Set are longer. However, a valid CDS is constructed in less than 14 seconds, even with 100 nodes.

In figure 9 is represented the number of state changes to have a valid CDS with k_{cds}=1 and $k_{cluster}$=2. More precisely, we measure the number of times a node changed on average its state before having a globally valid CDS. For example, with 100 nodes, 70% of the nodes become active, 80% dominatee and 35% dominators. During the first phase, among the active nodes, some nodes are elected dominators, and some other become dominatees. In the second phase, some dominatees become dominators to have a connected structure. In conclusion, a node changes its cds-state in average 2 times so that the structure becomes valid. Moreover, the number of cds-state changes per node is stable according to the number of participants. The algorithms are very scalable.

Finally, the behavior of the structure was studied during the time (fig. 10). With a network of 50 nodes, k_{cds}=1 and $k_{cluster}$=2, idle and active nodes are only present during the construction part, in the very first seconds. During the maintenance, no break of the CDS occur, and no dominatee nodes become isolated from the CDS. We can observe the initialization phase during the first seconds (fig. 11). Dominators are elected but, being redundant, they become dominatees after a few seconds. Moreover, the structure is very stable: the number of dominators and dominatees is almost the same all during the simulation. Slight variations appear because of packet collisions: some `hello` or `ap-hello` packets are lost. The nodes believe that a topology change occurs in the neighborhood: they try to reconstruct locally the backbone, some virtual topology changes can occur.

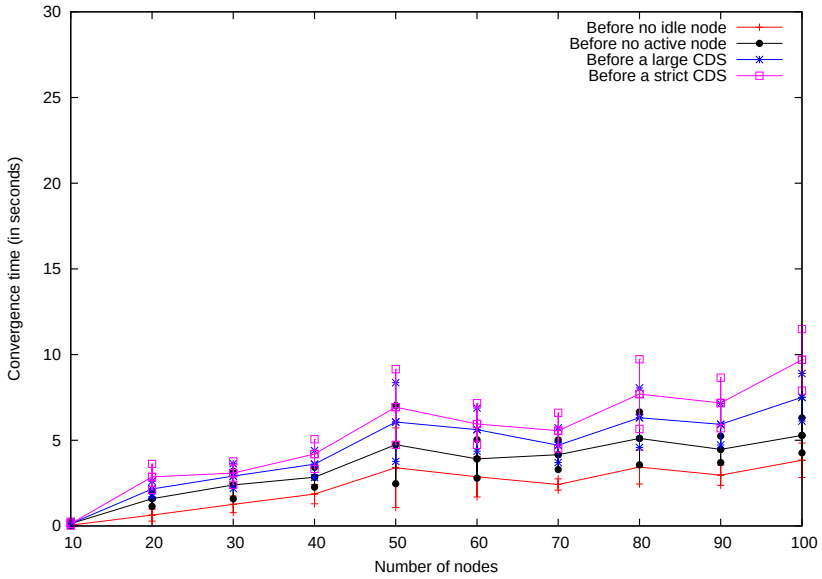

Figure 7: Convergence Time for a CDS (k_{cds}=1 / $k_{cluster}$=2)

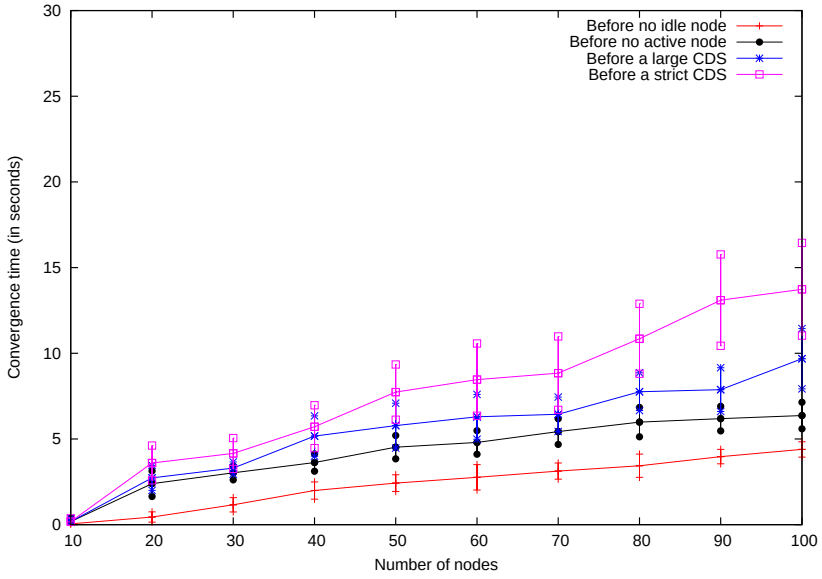

Figure 8: Convergence Time for a CDS (k_{cds}=2 / $k_{cluster}$=3)

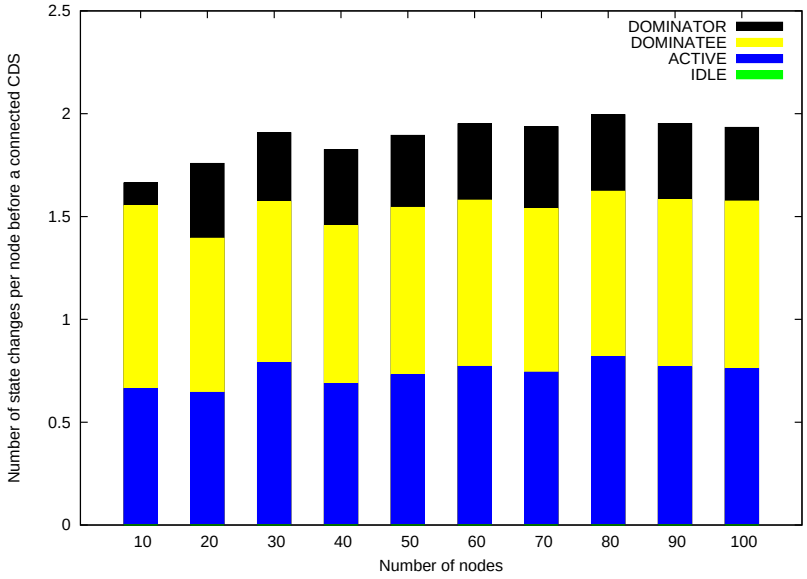

Figure 9: Ratio of the number of cds state changes and the number of nodes before having a connected CDS (k_{cds}=1 / $k_{cluster}$=2)

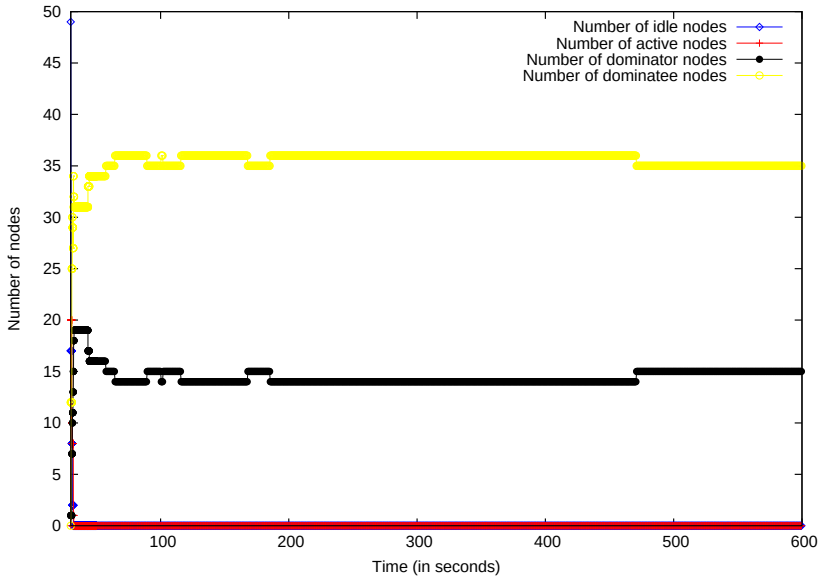

Figure 10: Number of idle/active/dominator/dominatee nodes during a 600s simulation (k_{cds}=1 / $k_{cluster}$=2 / 50 nodes)

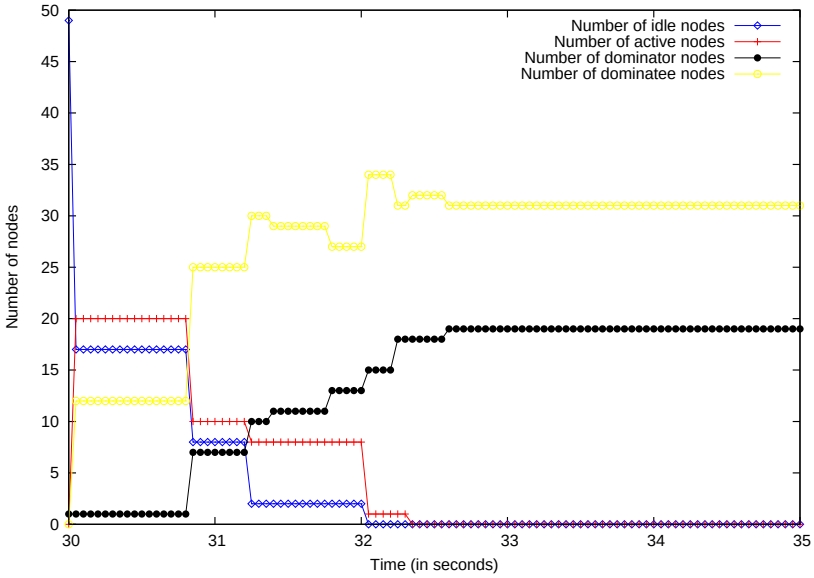

Figure 11: Zoom of the number of idle/active/dominator/dominatee nodes during the first seconds of simulation (k_{cds}=1 / $k_{cluster}$=2 / 50 nodes)

Temporary failure We simulate a temporary failure: a dominatee becomes arbitrarily dominator, or a dominator becomes arbitrarily dominatee (fig. 12). This simulates a node failure (after for example a power-off). We can remark that the convergence time if a dominatee becomes dominator is almost null. However, when a dominator becomes dominatee, a reconstruction could be required to have a connected tree. If we authorize the CDS to be *largely connected* (see the previous paragraph), the convergence time is greatly reduced. Less than 0.7 second is required to reconnect the CDS. If a random node is chosen for a temporary failure, the CDS is largely connected after 0.3 second, and strictly connected after at most 1.5 second, whatever the k_{cds} and $k_{cluster}$ are. Figure 13 presents the number of cds-state changes required to reconnect the CDS. Precisely, we measure the number of nodes in the network that changed their state so that the CDS is again globally valid. When a temporary failure occurred for a dominator, some dominatees can be out of the range from the backbone: an election must occur, creating idle nodes. However, the number of state changes is very limited. When a dominatee suffers from a temporary failure, neither election nor break of the CDS are required. Moreover, on average one node per temporary failure changes its state to maintain the CDS. Our maintenance algorithms are very efficient, and the stability of the structure is not impacted. A local topology change impacts only locally the CDS.

Mobility Finally, the impact of the mobility is studied. The random waypoint mobility model is used: a node chooses a random destination and a speed chosen uniformly between $[0, V_{MAX}]$. When the node reaches this destination, it chooses another destination and speed without pause. Firstly, the global behavior of the backbone according to the mobility was studied (fig. 14). We can see that the cardinality of the backbone is almost constant with the speed. The cardinality of the backbone with $k_{cds} = 2$ is logically smaller than for $k_{cds} = 1$ because the allowed distance from one node to the backbone is increased. The connectivity is smaller when $k_{cds} = 2$: the backbone reconnections are more complex to handle. The connectivity decreases when the mobility increases. However, even with a mobility of 30 m.s^{-1}, the connectivity remains over 99% with $k_{cds} = 1$ and over 95% with $k_{cds} = 2$.

Then, details of the impact of one topology change were given (fig. 16 & 15). A node is randomly picked up, and its new position is randomly chosen on the simulation area. Hence, all its old radio links will surely break. In consequence, a node must rediscover its neighborhood, determinate its new state, and perhaps must create an interconnection path to the existing backbone. Even with so many events to deal, the reconnection time remains under 5 seconds. The backbone reconnection requires more time when $k_{cds} = 2$. We can remark that the state of the node seems to have no impact on the convergence time.

We can see in figure 16 that changes are only local. Many nodes changed their neighborhood. However, only a few state changes are required (less than 4 state changes so that after the node *movement*, the backbone was reconnected). A dominatee seems to require more state changes for the reconnection: in almost all the cases,

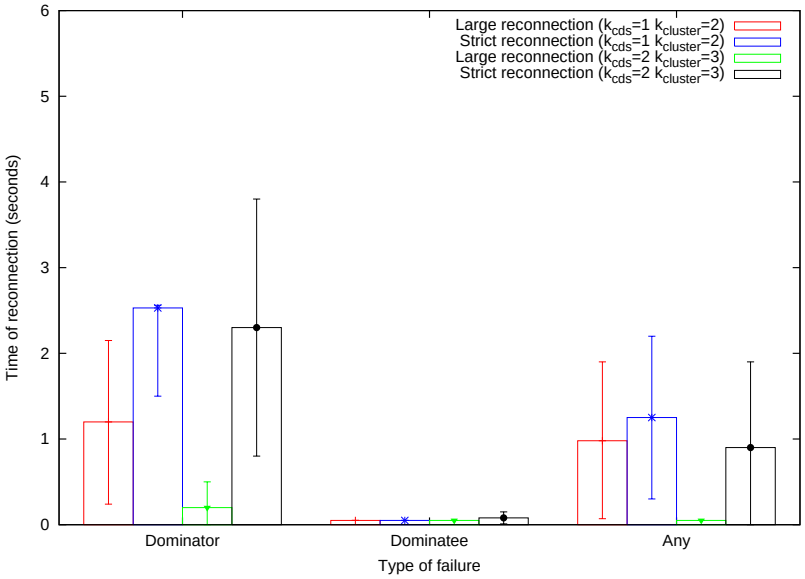

Figure 12: Reconnection Time after a temporary failure

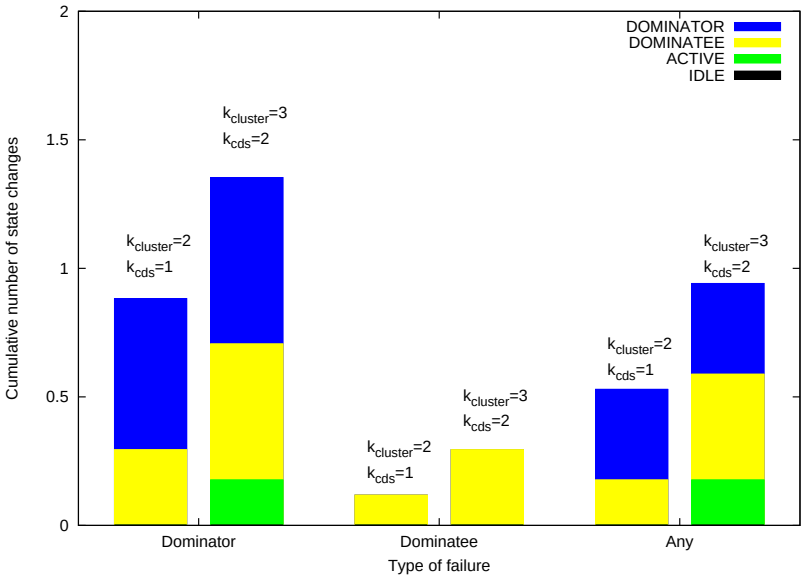

Figure 13: Number of changes of cds-states after a temporary failure

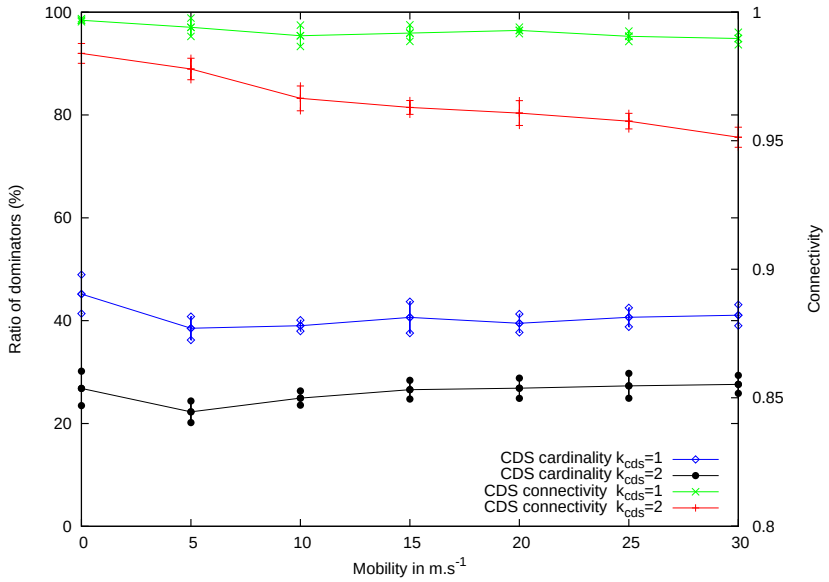

Figure 14: Cardinality and Connectivity of the CDS according to the mobility

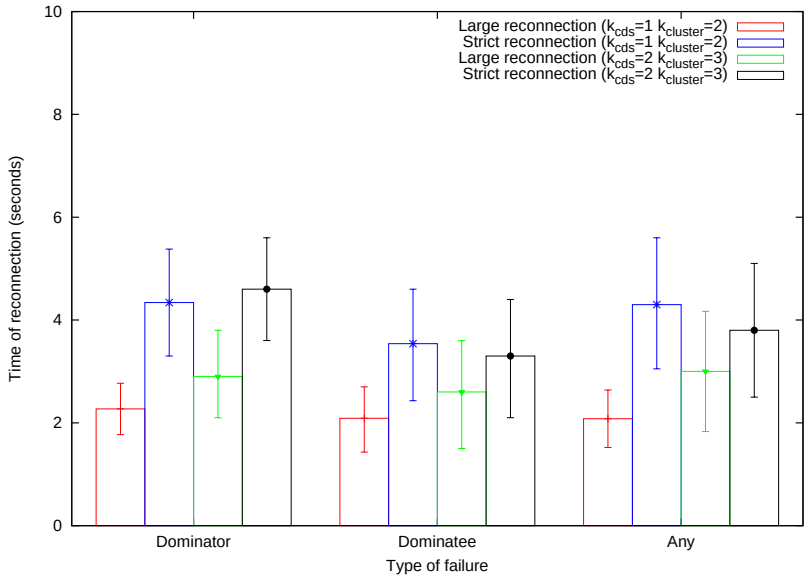

Figure 15: Reconnection Time after a discrete movement

the dominatee becomes active, triggers an election, and asks for the conversion of several dominatees. Because the backbone could be potentially broken, dominatees could become isolated, elections and reconnections are sometimes required (some nodes become idle). However, the backbone experiences only a few local changes. This explains why the backbone algorithms are very scalable according to the mobility.

8 Conclusion

In this article, we propose the construction and the maintenance of a virtual structure for the self-organization of ad hoc networks. A backbone helps to collect the traffic control and to distribute it efficiently in the network. Clusters create a hierarchical organization of the ad hoc networks, clusterheads managing their cluster, i.e. their services area. The construction algorithms are proven to construct a Connected Dominating Set and a clustering scheme in any ad hoc network. The complexity in messages is reduced, which represents a required property in wireless networks. Moreover, ad hoc networks present a very volatile topology. In consequence,

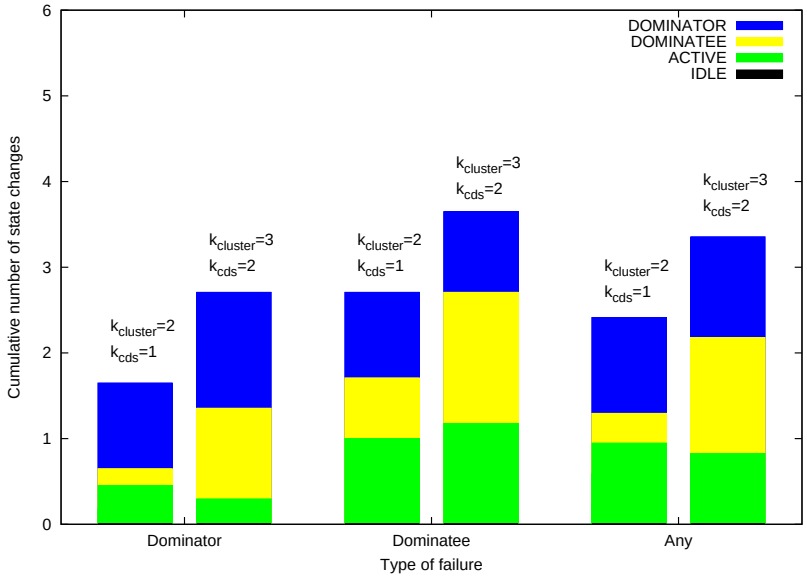

Figure 16: Number of changes of cds-states after a discrete movement

maintenance is vital. The proposed algorithms are proven to be self-stabilizing if topology changes occur in the network. Both the construction and the maintenance are time-bounded.

The virtual structure is studied trough simulations. The cardinality and connectivity of the structures remain very stable. Moreover, the structure is constructed efficiently and quickly. When a temporary failure occurs, the maintenance algorithms reconnect the structure in a very small delay, and only a few nodes are impacted by changes. A local topology change impacts only locally the structure. This explains the good scalability of the virtual structure.

The proposed virtual structure for self-organization is proven to be self-stabilized. In consequence, such a scheme is very flexible and totally parameterizable. It constitutes a genuine framework to deploy efficiently new services in ad hoc networks: routing could be deployed on this self-organization, taking into account the natural scalability of the virtual structure. More sophisticated services like localization, services discovering, hierarchical addressing should be studied and proposed.

References

[1] K. M. Alzoubi, P.-J. Wan, and O. Frieder. Distributed heuristics for connected dominating set in wireless ad hoc networks. *IEEE ComSoc/KICS Journal of Communications and Networks, Special Issue on Innovations in Ad Hoc Mobile Pervasive Networks*, 4(1):22–29, march 2002.

[2] Alan Amis, Ravi Prakash, Thai Vuong, and Dung Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *INFOCOM*, pages 32–41, Tel-Aviv, Israel, March 1999. IEEE.

[3] Sergiy Butenko, Xiuzhen Cheng, Ding-Zhu Du, and Panos M. Pardalos. On the construction of virtual backbone for ad hoc wireless networks. In *Cooperative Control: Models, Applications and Algorithms*, volume 1 of *Cooperative Systems*, chapter 3, pages 43–54. Kluwer Academic Publishers, January 2003.

[4] Mihaela Cardei, Xiaoyan Cheng, Xiuzhen Cheng, and Ding-Zhu Du. Connected domination in ad hoc wireless networks. In *International Conference on Computer Science and Informatics (CSI)*, North Carolina, USA, March 2002.

[5] Xiuzhen Cheng and Ding-zhu Du. Virtual backbone-based routing in multihop ad hoc wireless networks. Technical Report 02-002, University of Minnesota, Minnesota, USA, January 2002.

[6] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.

[7] Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. In *European Symposium on Algorithms (ESA)*, pages 179–193, Barcelona, Spain, Septembre 1996.

[8] Himanshu Gupta, Samir R. Das, and Quinyi Gu. Connected sensor cover: self-organization of sensor networks for efficient query execution. In *Interational Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 189–200, Annapolis, USA, June 2003. ACM.

[9] P. Krishna, Nitin Vaidya, Mainak Chatterjee, and Dhiraj Pradhan. A cluster-based approach for routing in dynamic networks. In *SIGCOMM*, pages 49–65, Cannes, France, April 1997. ACM.

[10] Chunhung Richard Lin and Mario Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7):1265–1275, 1997.

[11] Madhav V. Marathe, Heinz Breu, Harry B. Hunt III, S. S. Ravi, and Daniel J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, December 1995.

[12] S.Y. Ni, Y.C. Tseng, Y.S. Chen, and J.P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *International Conference on Mobile Computing and Networking (MOBICOM)*, pages 151–162, Seattle, USA, August 1999. ACM.

[13] Charles E. Perkins. *Ad hoc networking*, addison-wesley edition, 2001.

[14] Basu Prithwish, Khan Naved, and Thomas D. C. Little. A mobility based metric for clustering in mobile ad hoc networks. In ACM, editor, *International Conference on Mobile Computing and Networking (MOBICOM)*, pages 129–140, Roma, Italy, July 2001. ACM.

[15] Marco Schneider. Sef-stabilization. *ACM Computing Surveys*, 25(1):45–67, March 1993.

[16] Fabrice Theoleyre and Fabrice Valois. A virtual structure for mobility management in hybrid networks. In *Wireless Communications and Networking Conference (WCNC)*, volume 5 of *1*, pages 1035–1040, Atlanta, USA, March 2004. IEEE.

[17] Fabrice Theoleyre and Fabrice Valois. About the self-stabilization of a virtual topology for self-organization in ad hoc networks. In *Self-Stabilization Symposium*, Barcelona, Spain, October 2005.

[18] Jie Wu. An enhanced approach to determine a small forward node set based on multipoint relays. In *Vehicular Technology Conference (VTC Fall)*, pages 2774–2777, Orlando, USA, October 2003. IEEE.

[19] Jie Wu and Hailan Li. Dominating-set-based routing in ad hoc wireless networks. In *International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL'M)*, pages 7–14, Seattle, USA, August 1999. ACM.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399