



Institut Eurécom  
Department of Mobile Communications  
2229, route des Crêtes  
B.P. 193  
06904 Sophia-Antipolis  
FRANCE

Research Report RR-04-112

**Trajectory Knowledge for Improving Topology  
Management in Mobile Ad-Hoc Networks**

April 22<sup>nd</sup>, 2004

Jerome Haerri, Navid Nikaein and Christian Bonnet

Tel : (+33) 4 93 00 26 26

Fax : (+33) 4 93 00 26 27

Email : {Jerome.Haerri,Navid.Nikaein,Christian.Bonnet}@eurecom.fr

---

<sup>1</sup>Institut Eurécom's research is partially supported by its industrial members: Bouygues Télécom, Fondation d'entreprise Groupe Cegetel, Fondation Hasler, France Télécom, Hitachi, ST Microelectronics, Swisscom, Texas Instruments, Thales

# Trajectory Knowledge for Improving Topology Management in Mobile Ad-Hoc Networks

Jerome Haerri, Navid Nikaein and Christian Bonnet

## Abstract

This paper presents a novel approach to topology management in mobile ad-hoc networks. We are proposing an algorithm that is able to construct and maintain a Connected Dominating Set (CDS), without using periodic beacons. By knowing the positions and velocities of its neighbors, a node is able to extract their linear trajectories. Based on this information, it obtains a local prediction of its neighborhood's evolution, and can thereafter proactively adapt the dominating set without relying on periodic beacons. Maintenance will be driven as a per-event basis; it is only when a node changes course that messages are to be exchanged to adapt the CDS. The correctness of the algorithm is proven, and the complexity is compared with other topology management heuristics. Our approach is able to keep a stable time-changing CDS, and a low broadcasting overhead while having a lower complexity than other approaches.

# 1 Introduction

A mobile ad-hoc network (Manet) consists of a collection of mobile nodes forming a dynamic autonomous network through a fully mobile infrastructure. Nodes communicate with each other without the intervention of centralized access points or base stations. In such a network, each node acts as a host, and may act as a router. Due to limited transmission range of wireless network interfaces, multiple *hops* may be needed to exchange data between nodes in the network, which is why the literature often uses the term of *multi-hop* network in Manet. The *topology* of a multi-hop network is the set of communication links between nodes used by routing mechanisms. Removing redundant and unnecessary topology information is usually called *topology management*. The topology management plays a key role in the performance of a routing protocol simply because the wrong topology information can considerably reduce the capacity, increase the end-to-end delay and routing control overhead, and decrease the robustness to node failure.

There are two approaches to topology management in mobile ad-hoc networks: power control and hierarchical topology organization. Power control mechanisms adjust the power on a per-node basis, so that one-hop neighbor connectivity is balanced and overall network connectivity is ensured. However, topologies derived from power control schemes often result in unidirectional links that create harmful interferences due to different transmission ranges among one-hop neighbors [6]. The hierarchical approach selects a subset of nodes, called *cluster-heads*, to serve as the network backbone over which essential network control functions are supported [7]. Every node is then associated with a cluster-head, and cluster-heads are connected with each others via gateway nodes. Therefore, the union of cluster-heads and gateways constitute a connected backbone. For clustering to be effective, links and nodes that are part of the backbone must be close to the minimum, and must be connected. In graph theory, the Minimum Dominating Set (MDS) problem and the relevant Minimum Connected Dominating Set (MCDS) problem best describe the clustering approach to topology management. MDS consists of finding a subset of nodes with the following property: each node is either in the dominating set, or adjacent to a node in the dominating set. MCDS consists of obtaining a minimum subset of nodes in the original graph, such that it composes a Dominating Set (DS), and the induced subgraph of an MCDS has the same number of connected components than the original graph. Unfortunately, in graph theory, computing the MDS is *NP-hard* [11, 12], and MCDS is a well known *NP-complete* problem, even if the complete topology is available. In heuristics proposed in the past, cluster-heads are equivalent to a sub-optimal dominating set, that can be improved through non-deterministic negotiations or by applying deterministic criteria, and reach a *Minimal* Dominating Set. The union of selected cluster-heads and gateways forms a connected dominating set (CDS), which is a sub-optimal solution to the MCDS problem.

Node mobility can cause frequent unpredictable topology changes. Hence topology management is a non trivial task. Many hierarchical approaches have

been proposed, including DDR [2], TMPO [8], and MPR [10]. MPR creates a topology based Multi-Point Relay nodes (MPR), where each node determines its own MDS (set of MPR nodes), *MPR selector set*, covering up to its two hops neighbors. Then, a node retransmits a broadcast message if it belongs to the sender's *MPR selector set*. This approach is however suboptimal, because the set of MPR nodes at each node overlaps those of neighboring nodes. Therefore, considering the complete network, MPR is only able to create a global DS and not a global MDS. In order to obtain such MDS, MPR is improved (used by OLSR [9]) by exchanging MPR sets through the network, such that an optimality in the number of MPR nodes is reached.<sup>1</sup> TMPO [8], on the other hand, is a cluster-based topology management protocol, in which a Connected Dominating Set (CDS) is introduced, that keeps the same number of connected components as the original topology, yet with a lower reorganization complexity. TMPO uses a "willingness" value of a node, which represents the probability a node gets to be elected in the MDS, given its mobility and its battery life. Finally, DDR [2] uses forests and zones abstractions to obtain a fully connected graph, while keeping a low topology management complexity. Zones are proactively maintained, and as TMPO, negotiation is not required. Besides, DDR uses nodes' degree as an electing criterion, such that the broadcast tree is optimized.

In this paper, we are focusing on a novel approach, in which nodes are able to predict neighbors' future position, hence getting rid of periodic beacons. We are adapting this concept to a hierarchical topology control approach denoted as Kinetic Adaptive Dynamic topology management for Energy efficient Routing (**KADER**). We are proposing a distributed self-maintained topology management strategy based on deterministic criteria, where changes in the topology (trajectory changes) are announced by the respective nodes in a per-event basis. We want to construct a self-adaptive forest from a network topology in a distributed way, yet without using periodic beacons. Each tree in the forest forms a zone and each zone is proactively maintained. While forest seeks to reduce broadcasting overhead, proactive zones are used to reach high scalability. Besides, since the energy cost of increased computation is much less than the cost of increased message transmission [1], we want to reduce as much as possible the use of topology management messages. To do so, the idea is, by modeling nodes' positions as a piece-wise linear trajectory, as opposed to a fixed position for a single time instant, we can predict a node future position, and according to it, adapt the dominating set. The protocol does not need periodic beaconing and only requires minimal updating when a node changes course, since most of the links remain valid. Moreover, it is a node's responsibility to announce its trajectory change. Therefore, the algorithm can be seen as an per-event approach. And since energy is not wasted on message transmissions, KADER reaches energy efficiency on topology management.

Similar to TMPO [8], MPR [9, 10], and DDR [2], our approach seeks a CDS and does not need any negotiation round to obtain it. Nevertheless, unlike TMPO,

---

<sup>1</sup>We will thereafter consider the improved version when mentioning MPR.

the topology created with KADER is fully distributed. The algorithm does not put any additional burden on the selected nodes, while others are in sleeping mode. The overall topology maintenance is fully distributed along the network, further suppressing the need to balance the cluster-heads' role. Moreover, in highly mobile networks, cluster-heads are usually weak points in a topology management, since their loss has a major influence on the global network connectivity. Besides, contrary to MPR, KADER does not need to propagate the topology messages on the entire network. Topology management messages in KADER are not forwarded outside a zone. Finally, what dissociates our approach from all these topology management algorithms, is that KADER uses trajectory knowledge, and consequently does not need any periodic update to maintain the coherence of the created topology.

The rest of the paper is organized as follows. Section 2, outlines the method used to obtain nodes trajectory knowledge. In Section 3, we present in detail the different parts of the algorithm. Section 4 characterizes KADER's Connected Dominating Set. In Section 5 we show the low overhead complexity of KADER. Section 6 highlights the benefit of KADER on routing protocols. Finally, we draw concluding remarks and highlight future work.

## 2 Trajectory Knowledge

The term "Kinetic" in KADER reflects the motion aspect of our algorithm, which computes a node's future trajectory, based on its Location Information [3]. Such location information may be provided by the Global Positioning System (GPS) or other solutions exposed in [4] or [5]. Therefore, we assume a global synchronization between nodes in the network and define  $x, y, dx, dy$  as the four parameters defining a node's position and instant velocity<sup>2</sup>, thereafter called *mobility*. Then, consider the nodes in a mobile ad-hoc network. In order to compute neighbors' trajectories, nodes must first exchange their respective mobility parameters. They do so by, only once, broadcasting their parameters to their immediate neighbors. Such message will not be forwarded, therefore a node will receive the parameters  $x, y, dx, dy$  of only its one-hop neighbors.

Over a relatively short period of time<sup>3</sup>, one can assume that each such node, say  $i$ , follows a linear trajectory. Its position as a function of time is then described by

$$\mathbf{Pos}_i(t) = \begin{bmatrix} x_i + dx_i \cdot t \\ y_i + dy_i \cdot t \end{bmatrix}, \quad (1)$$

where  $Pos_i(t)$  represents the position of node  $i$  at time  $t$ , the vector  $[x_i, y_i]^T$  denotes the initial position of node  $i$ , and vector  $[dx_i, dy_i]^T$  its initial instantaneous

---

<sup>2</sup>We are considered moving in a two-dimensional plane.

<sup>3</sup>The time required to transmit a data packet is orders of magnitude shorter than the time the node is moving along a fixed trajectory.

velocity. Lets consider node  $j$  a neighbor of  $i$ . In order for node  $i$  is able to compute node  $j$ 's trajectory, let us define the squared distance between nodes  $i$  and  $j$  as

$$\begin{aligned}
 D_{ij}^2(t) &= D_{ji}^2(t) = \|\mathbf{Pos}_j(t) - \mathbf{Pos}_i(t)\|_2^2 \\
 &= \left( \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix} + \begin{bmatrix} dx_j - dx_i \\ dy_j - dy_i \end{bmatrix} \cdot t \right)^2 \\
 &= a_{ij}t^2 + b_{ij}t + c_{ij},
 \end{aligned} \tag{2}$$

where  $a \geq 0$ ,  $c \geq 0$ . Consequently,  $a_{ij}$ ,  $b_{ij}$ ,  $c_{ij}$  will be defined as the three parameters describing nodes  $i$  and  $j$  mutual trajectories, and  $D_{ij}^2(t) = a_{ij}t^2 + b_{ij}t + c_{ij}$ , representing  $j$ 's relative distance to node  $i$ , will be denoted as  $j$ 's linear relative trajectory to  $i$ . Consequently, thanks to Eq. 1, a node is able to compute one of its neighbor's future position, and by using Eq. 2, it is able to extract its neighbor's future relative distance to it.

### 3 Kinetic Adaptive Dynamic Topology Management Algorithm

The algorithm consists of six cyclic time-ordered phases: *neighborhood discovery*, *preferred neighbor election*, *self-adaptive intra-zone clustering*, *self-adaptive inter-zone clustering*, and *event-oriented zone maintenance*. These phases are carried out based on trajectories and stability information exchanged through specific non periodic event-oriented messages between nodes. The main idea of KADER is to model node positions as a piece-wise linear trajectory, as opposed to a fixed position for a single time instant, and to predict nodes' future positions [3]. It is only when a node changes its initial trajectory that the algorithm needs to be run again. And to adapt this idea to topology management, we want to construct a self-adapting forest from an ordinary network. Each such self-adapting tree in the constructed forest forms a zone.<sup>4</sup> Then, the network is partitioned into a set of non-overlapping dynamic zones. Each zone is connected via nodes that are not in the same zone but are in direct transmission range of each other. Such nodes are called *gateways*. So the whole network can be seen as a set of connected zones. The size of a zone will increase or decrease dynamically without any need of periodic maintenance. Unexpected topological changes are announced by the respective nodes, through a specific message communicating its new mobility parameters. Following this event, the forest will adapt itself to the new topology.

#### 3.1 Neighborhood Discovery

Basically a neighboring table is a table through which a node detects changes to its neighborhood. This table consists of several information related to a node's direct neighborhood: neighboring identities *nid*, neighboring mobility, such as

---

<sup>4</sup>We will later use the term tree and zone interchangeably.

$x, y, dx, dy$ . It also contains neighboring stability parameters  $\beta$  and neighboring last trajectory change time  $t_i$ . As a node changes course, KADER will consider it as a new node in the network. Since  $nids$  cannot be changed, and that we do not forward nodes trajectories in the network as a way to distinguish a node from the same node that has changed its trajectory, we will add a Trajectory Counter ( $Tc$ ), which mentions the number of trajectory changes a node experienced. Thus, a unique identifier of node  $i$  and its trajectory is defined by the pair  $(nid_i, Tc_i)$ , and will thereafter simply be referred as  $i$ .

In order to construct this table, each node broadcasts a single message to indicate its presence in the neighborhood, and to transmit its parameters  $(i, x, y, dx, dy, \beta, t_i)$  as well. Upon receiving this kind of message, a node can make local predictions about its neighborhood, like neighbors' expected future positions, or expected connection time (before being out of range). Since such predictions will be considered valid for all time, there will be no need to periodically refresh them. If they happened to be invalidated by an unpredicted event (a trajectory change), the respective node will spontaneously advertise its new parameters, refreshing the predictions in a event-driven way. For example in Figure 3, the neighboring table of node  $k$  has three neighbors, which are  $\langle (f, x, y, dx, dy, \beta, t_f), (c, x, y, dx, dy, \beta, t_c), (d, x, y, dx, dy, \beta, t_d) \rangle$ .

### 3.2 Preferred Neighbor Election

Instead of a node degree or a node priority, we are introducing a novel link parameter that will better represent the routing orientation of our topology management algorithm. A zone in KADER being proactively maintained, each node has to keep a `zone_table` that describes the path to reach any destination contained in its zone. Yet, we do not want these `zone_tables` to hold the best path in term of hops, or broadcasting coverage, but in term of power needed to reach a destination. To do so, we are introducing a *modified power function* computed between a node and its neighbors. However, due to trajectory changes, all links a node keeps become sooner or later invalid. Therefore, in order to increase the life of a link, we need to create links between nodes whose trajectories will hold as long as possible. Consequently, we are introducing a *stability function* which is representative of a node's probability to keep its trajectory. Besides, to reflect a node finite range, we will ensure that the criterion used to elect a node will fall to zero as soon as a node becomes unreachable. A *sigmoid function*, with a dropping time equal to the time at which a node becomes unreachable, will be used in order to refrain a node to create a link with a node that is out of range. Thus, we define *Quality of Connectivity* as a function of those three criteria. Then, the algorithm selects for each node, a neighbor that has the max *QoC* during a given interval. Such neighbor will be thereafter denoted as *Preferred Neighbor* (PN). We define a *Preferred Link* a link created by connecting a node with its *Preferred Neighbor*. We will prove in A that whatever the network topology is, connecting each *preferred link* always yields to a forest at every time instant.

### 3.2.1 An Energy Based Election Criterion:

The *power cost* as a function of time, required to transmit between nodes  $i$  and  $j$  at time  $t$ , is defined as  $P_{ij}(t) = P_{ji}(t) = \kappa D_{ij}^2(t)$ , for some constant  $\kappa$ ; without loss of generality, we assume  $\kappa = 1$ .<sup>5</sup> By choosing power as the cost, one obtains minimum power routes that help preserve battery life.

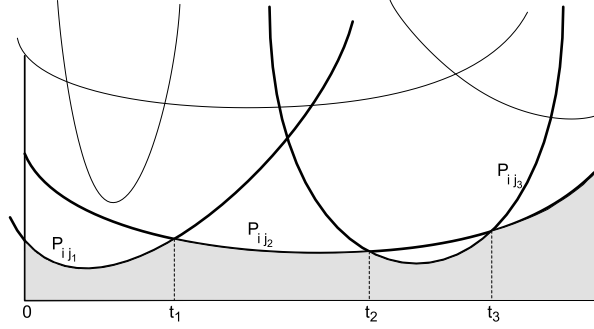


Figure 1: Graphical representation of the Power function at node  $i$ .

This deterministic criterion translates to a routing table at each node, whose entries are the minimum-power paths to reach a neighbor as a function of time. A graphical representation of such table at node  $i$  appears in Figure 1. The shaded area indicates its *composite power cost*  $P_i(t) = \min_t \{P_{ij_1}(t), P_{ij_2}(t), P_{ij_3}(t)\}$  which is required to reach  $j_1, j_2$ , and  $j_3$ . One sees that at times  $t_1, t_2$ , and  $t_3$ , a new path becomes optimal. To achieve minimum power routing, node  $i$  forwards to node  $j_1$  for  $0 \leq t < t_1$ , to node  $j_2$  for  $t_1 \leq t < t_2$ , to node  $j_3$  for  $t_2 \leq t < t_3$ , and to node  $j_2$  for  $t_3 \leq t$ .

This criterion effectively minimizes at node  $i$  the *energy function*

$$F_i = \int_{t_0}^{\infty} P_i(t) dt \quad (3)$$

in a distributed manner, where  $t_0$  denotes the execution time and where the *energy function*  $F_i$  denotes the total energy needed to reach node  $i$  from  $t = t_0 \rightarrow \infty$ .

However, nodes keep their trajectories only a short period of time. Therefore, the topology configuration obtained in Figure 1 might be invalid even before node  $i$  decides to forward to  $j_3$ . Even worse, a node which happens to be sub-optimal in term of power may become highly attractive through its high stability (meaning that even if the link is not optimal, it will remain valid a longer time). To deal with this problem, we bias the power function toward  $t_0$  since as  $t \rightarrow \infty$  nodes will have changed trajectory anyway.

<sup>5</sup>Therefore, Power and Distance will later be interchangeably used.



We define

$$p_i(t) = e^{-\beta_i(t-t_i)} \quad (4)$$

as the probability that a node  $i$  is continuing on its present trajectory, where the Poisson parameter  $\frac{1}{\beta_i}$  indicates the average time the node follows a course, and  $t_i$  the time its current trajectory began.

Assuming independent node trajectories,  $p_{ij}(t) = p_i(t) \cdot p_j(t)$  describes the probability that nodes  $i$  and  $j$  are continuing on their respective courses at time  $t$ , which will be considered as the *stability* of link  $\overline{ij}$ . The modified power cost below probabilistically weights the power cost  $P_{ij}(t)$  to reflect the link's *stability*.

$$\tilde{P}_{ij}(t) = -\frac{p_{ij}(t)}{P_{ij}(t)} \quad (5)$$

A low modified power cost favors a low power cost with high stability.

Finally, since we are getting rid of beacons, a node that will shortly leave the neighborhood must be automatically removed from the neighboring table. We do so by computing a *timeout* counter, given a neighbor relative position and velocity. Upon expiration, it will remove the corresponding neighbor from the table. Besides, we still need to refrain nodes to elect a PN that, either will soon leave, or will have left when we will need to reach it. To represent the node's finite range, we introduce the inverse sigmoid function

$$Sigm_i(t) = \frac{1}{1 + e^{a \cdot (t - timeout_i)}} \quad (6)$$

whose value is equal to 1 until  $t = timeout$  and drops to 0 when the node runs out of range. And  $timeout_i$  is the neighbor's time before being unreachable to  $i$ , which can be computed given the neighbor relative position, velocity, and transmission range. Functions 4 and 6 readily substitute in the algorithm described before:

$$\tilde{P}_{ij}(t) = -\frac{p_{ij}(t)}{P_{ij}(t)} \cdot Sigm_{ij}(t) \quad (7)$$

$$\begin{aligned} &= -\frac{e^{-(\beta_{ij})(t-t_{ij})}}{a_{ij}t^2 + b_{ij}t + c_{ij}} \cdot \frac{1}{1 + e^{a \cdot (t - timeout_{ij})}} \\ &= -\frac{e^{-(\beta_i + \beta_j)(t - \frac{t_i\beta_i + t_j\beta_j}{\beta_i + \beta_j})}}{a_{ij}t^2 + b_{ij}t + c_{ij}} \cdot \frac{1}{1 + e^{a \cdot (t - \min_{k \in \{ij\}} \{timeout_k\})}} \end{aligned} \quad (8)$$

Now we have seven parameters  $a_{ij}(t)$ ,  $b_{ij}(t)$ ,  $c_{ij}(t)$ ,  $\beta_{ij}(t)$ ,  $t_{ij}(t)$ ,  $timeout_j$  and  $a$  describing  $\tilde{P}_{ij}(t)$  as criteria for a preferred link between two nodes. The parameter  $a$ , which control the transition slop between 1 and 0 will be in fact common to every node and be fixed at the beginning of the simulation. Figure 2 is an example of a typical quality of connectivity table using the modified power

cost. The shaded area still represents the evolution of the *modified power cost*,  $\tilde{P}_i(t) = \min_t \{\tilde{P}_{ij_1}(t), \tilde{P}_{ij_2}(t), \tilde{P}_{ij_3}(t)\}$ , between next-hop nodes  $j_1$ ,  $j_2$  and  $j_3$ ; Node  $i$  forwards to node  $j_1$  for  $0 \leq t \leq t_1$ , to node  $j_2$  for  $t_1 \leq t \leq t_2$ , and to node  $j_3$  for  $t_2 \leq t$ .

Finally, since we are minimizing the *modified power cost* in order to obtain the best link between two nodes for all time, we define the *Quality of Connectivity* as

$$QoC(t) = -\frac{1}{\tilde{P}_{ij}}(t) \quad (9)$$

Then, by minimizing a link's *modified power cost*, we are in fact maximizing its *QoC*.

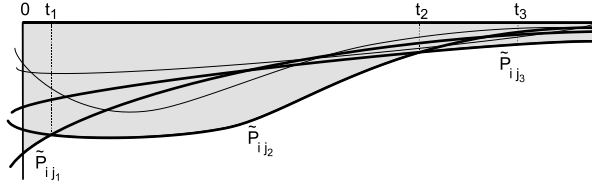


Figure 2: Graphical representation of the QoC function at node  $i$ .

### 3.2.2 Election Algorithm:

Based on the information provided by the neighboring table, a node  $i$  can determine its *preferred neighbor*  $j$  at time  $t_1$ , which represents the time at which  $j$  has the biggest *QoC* over all other neighbors of  $i$ . The criterion is defined as

$$PN_i(t_1) = j \quad \text{iff} \quad QoC_{ij}(t_1) = \max_{k \in nb_i} (QoC_{ik}(t_1)) \quad (10)$$

We further define a fixed interval  $[t_1, t_2]$ , called *activation*, in which  $j$  remains  $i$ 's PN. We say that  $j$  is *activated* by  $i$  over  $[t_1, t_2]$ , and during this activation,  $j$  has the biggest *QoC* among  $i$ 's neighbors. An activation between node  $i$  and node  $j$  over an interval  $[t_1, t_2]$  is defined as

$$act(i, j)[t_1, t_2] = \begin{cases} \min t_1 \text{ s.th. } PN_i(t_1) = j \\ \max t_2 \in (t_1; \infty) \text{ s.th. } \begin{cases} \forall t \in [t_1, t_2] \\ PN_i(t) = j \end{cases} \end{cases} \quad (11)$$

Therefore, the set  $(i, j, act(i, j)[t_1, t_2])$  uniquely identifies a *preferred link* between node  $i$  and node  $j$  over  $[t_1, t_2]$  and will thereafter mentioned as  $\overline{ij}_{[t_1, t_2]}$ .

Although a node can only have one PN at a time (see Section 3.2.1), it can have several PNs over time, with mutually exclusive set of activations. This means that a node can locally predict its actual and future PNs, conditioned over the lack of any unexpected topology changes. As time goes on, nodes will switch from PNs to PNs, always maximizing the *QoC*. The set of  $i$ 's PNs, denoted  $PN_i$ , regroups all actual and predicted future PNs. This set is defined as :

$$j \in PN_i \text{ iff } \exists t_1, t_2 \text{ s.th } act(i, j)[t_1, t_2] \neq \emptyset$$

Note that  $PN_i(t)$  uniquely defines a PN of node  $i$  at time  $t$ .

We distinguish three cases:

- **No PN**— if the set is empty, then node  $i$  has no PN which means it has no neighbors. In Figure 3, node  $n$  has no neighbor and consequently no PN;
- **Single PN**— if the  $PN_i$  has only one member, then this member is the elected PN at every time, until further notified (unexpected topology changes).
- **Multiple PN**— the set  $PN_i$  can have more than one member. In that case, node  $i$  will automatically switch to the new PN, as soon as its actual one becomes sub-optimal. This procedure does not require any extra overhead, since this situation has been predicted in a distributed way.

Finally, note that  $PN_i$  is kept valid until further notified, meaning that, unless  $i$  receives a new trajectory message from one of its neighbors, this set stays optimal. Upon reception of such messages,  $i$  is allowed to revise its previous decision, so that the set remains optimal (see Section 3.6). Furthermore, thanks to the intrinsic property of the QoC graph which activations are obtained from, the set of activations contained in  $PN_i$  are mutually exclusive (see Figure 2 where  $[0, t_1] \cap [t_1, t_2] = \emptyset$ ).

Consequently the algorithm selects, for each node  $n$  in the network topology and at each time  $t$ , a neighbor that has the biggest QoC in the neighborhood. We say that node  $i$  is the *preferred neighbor* of node  $j$  during a defined interval, if  $j$  is in the neighborhood of  $i$  and has the biggest QoC among its neighbors during this interval. Therefore, each node elects exactly one PN at each time and can be chosen as the PN of many nodes. Thus, the way in which a node is elected follows a monotonic increasing function depending only on its QoC. A time adapting forest is built after connecting each node to its PN during their respective activations. In Section A, we will prove that, whatever the network topology is, this approach always yields to a forest at every time.

### 3.3 Forest Construction

In order to construct *preferred links* and consequently the forest, each node generates a table called *Intra-Zone table*. Indeed, as soon as node  $i$  determines the set of its PNs, it must notify its neighbors, especially its PNs, of its decision.

Therefore, node  $i$  sends a PN message  $PN_i = (i, \overline{ij}_{[t_1, t_2]}; \overline{ik}_{[t_3, t_4]})$ . It then updates its *Intra-Zone table* regarding its PNs. This message indicates that node  $i$  is electing node  $j$  as its PN with the activation  $act(i, j)[t_1, t_2]$  and node  $k$  as its future PN with the activation  $act(i, k)[t_3, t_4]$ . Note that  $(act(i, j)[t_1, t_2] \cap act(i, k)[t_3, t_4]) = \emptyset$  and  $(act(i, j)[t_1, t_2] \cup act(i, k)[t_3, t_4]) = [t_1, t_4]$ . Upon reception of  $i$ 's message, node  $j$  check whether it has been chosen as the PN of  $i$ . If so, it also updates its intra-zone table regarding  $i$ . This means that a tree branch is built between node  $i$  and its preferred neighbor  $j$  during  $[t_1, t_2]$ , and a future tree branch between  $i$  and  $k$  is also created over  $[t_3, t_4]$ . Besides, the preferred links  $\overline{ij}_{[t_1, t_2]}$  and  $\overline{ik}_{[t_3, t_4]}$  belong to a different tree, since  $act(i, j)[t_1, t_2]$  and  $act(i, k)[t_3, t_4]$  are mutually exclusive. Therefore, those edges become a preferred link, and the set of preferred links in each neighborhood generates the set of preferred paths in the network.

### 3.4 Self-Adaptive Intra-Zone Clustering

At this phase, nodes attempt to expand their own view about the tree they belong to by completing their intra-zone table. Indeed, when a node  $i$  gets elected by a neighbor, it then locally advertises the new PN learned. To do so,  $i$  sends a so called *Learned\_PN* message  $Learned\_PN_i = (i, \overline{ij}_{[t_1, t_2]} : \overline{jk}_{[t_3, t_4]} : \overline{jl}_{[t_5, t_6]}; \dots; \overline{iu}_{[t_7, t_8]} : \overline{uv}_{[t_9, t_{10}]})$ , indicating that node  $j$  with  $act(i, j)[t_1, t_2]$  has nodes  $k$  with  $act(j, k)[t_3, t_4]$  and  $l$  with  $act(j, l)[t_5, t_6]$  as its PNs, and node  $u$  with  $act(i, u)[t_7, t_8]$  has node  $v$  with  $act(u, v)[t_9, t_{10}]$  as its PN.

**Definition 1** *Let node  $k$  be a Preferred Neighbor of node  $j$ , and a Learned Preferred Neighbor of node  $i$ . We formally define an activation  $act(j, k)[t_3, t_4]$  as **valid** with respect to an activation  $act(i, j)[t_1, t_2]$ , iff  $(act(i, j)[t_1, t_2] \cap act(j, k)[t_3, t_4]) \neq \emptyset$ .*

A direct consequence of Definition 1 is that  $i, j, k$  are belonging to the same tree over the activation  $(act(i, j)[t_1, t_2] \cap act(j, k)[t_3, t_4])$ , while  $act(j, k)[t_3, t_4] \setminus (act(i, j)[t_1, t_2] \cap act(j, k)[t_3, t_4])$  is considered as a separate tree connecting only  $j, k$ .

Upon reception of this message, and if PNs learned by it have valid activations, each tree member updates its *intra\_zone* table, and re-advertises to their neighbors if it is not a leaf node<sup>6</sup>. For this purpose, each node generates another field in its *intra\_zone* table called *Learned Preferred Neighbor (Learned\_PN)* in order to keep nodes that have been learned to be a tree member. We mention again that node  $j$  is chosen to be the PN of  $i$  over an interval of  $[t_1, t_2]$ , and  $i$  has sent a PN message to inform its neighborhood of its elected PN. Among the neighboring nodes of  $i$ , the PN  $j$  forwards  $i$ 's decision to nodes that hold a tree edge with  $j$ , say  $k$ , activated over  $[t_3, t_4]$ , by setting its *Learned\_PN* message to  $LPN_j = (j, \overline{ji}_{[t_1, t_2]})$ . Then, the

---

<sup>6</sup>A leaf node is a node which only has a single neighbor and which is never a PN.



PN	Learned_PN	PN	Learned_PN
$\mathbf{f}_{[0,10]}$	$a_{[0,10]}, b_{[0,10]}$	$w_{[0,10]}$	$r_{[0,10]}$
	$q_{[0,10]}, w_{[0,10]}$		$J_{[0,6]}$
	$r_{[0,10]}, j_{[0,6],[6,10]}$	$\mathbf{k}_{[0,10]}$	$c_{[0,10]}, d_{[0,10]}$
	$l_{[9,10]}, s_{[9,10]}$	$j_{[6,10]}$	-
$\mathbf{d}_{[10,20]}$	$g_{[10,20]}, u_{[10,20]}$	$b_{[0,10]}$	-
	$v_{[10,20]}, h_{[10,20]}$	$a_{[0,10]}$	$s_{[9,10]}, l_{[9,10]}$
$c_{[0,10]}$	-	$q_{[0,10]}$	-

(a)  $Intra\_ZT_k$ (b)  $Intra\_ZT_f$ Table 1: Intra-zone table of nodes  $k$  and  $f$  regarding Figure 3

nodes during the neighborhood discovery.

Then, as shown in Table 1(a) & 1(b), the view of a node, say  $i$ , about its tree consists of two levels:  $PN$ ,  $Learned\_PN$ . The  $PN$  level contains the nodes holding tree-edges with node  $i$ . The second level,  $Learned\_PN$ , contains nodes that are learned by the  $PN$  level. In fact, node  $i$  can reach them via their associated  $PN$  in its intra-zone table. Therefore, node  $i$  only knows the next hop for its second level nodes (see Figure 4). Actually, each entry in  $Intra\_ZT_i$  can be seen as a branch of  $i$ , that is  $NID \rightarrow Learned\_PN$ . Thus, each node obtains a partial view of its tree in the sense that it does not know the detailed structure of its tree.

For example, in Figure 3, where node  $k$  is only able to see one route to reach  $j$ , node  $f$  sees two different and exclusive paths to reach  $j$ , one through  $w$  and another directly. Nevertheless, this does not have any influence on the Learned.PNs of  $f$ , since they still need to route through it anyway. Therefore,  $f$  is called *critical vertex*, since all up-link nodes (up-link with respect to node  $j$ ) will not see this change in the topology, and will not receive any message about it. The topology reorganization scope in KADER will be limited to these *critical vertices*, such that the complexity remains acceptable. Finally, note that from  $t \in [9, 10]$ , the tree, which node  $k$  belongs to, is expanded to reach nodes  $s$  and  $l$ , such that node  $k$  will learn about  $l$  and  $s$  existence for  $t \in [9, 10]$ . Again, this procedure does not need any message, as it had already been predicted during the initial topology creation.

### 3.5 Self-Adaptive Inter-Zone Clustering

A different routing table, called *Inter-Zone* table is created to represent nodes that does not belong to the tree over activations different from those in the *Intra-Zone*. Every node belonging to this table over some activations are considered part of a different tree and will be called a *gateway*. The link between two gateways belonging to two different trees will be called a *bridge*.

At the beginning, every neighbors of a node  $i$  are put in its *Inter-Zone* table during their full initial activation, say  $[T_1, T_2]$ , which is defined as the connection

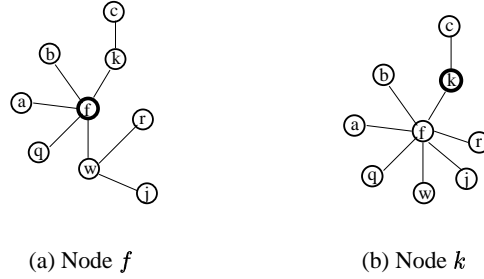


Figure 4: Tree view of nodes  $k$  and  $f$ .

life between the two nodes, or the time two nodes remain direct neighbors. Then, as a node  $i$  succeeds to add some neighbor  $j$  to its tree and updates its *intra\_zone* table over an activation  $[t_1, t_2]$ , it therefore prunes  $j$ 's initial activation. The remaining activation is then  $\{[T_1, T_2] \setminus [t_1, t_2]\}$ . During this time, node  $j$  is still not considered belonging to the same tree as  $i$ . Node  $j$  then appears in the *Intra\_Zone* table over  $[t_1, t_2]$  and in the *Inter\_Zone* table over  $\{[T_1, T_2] \setminus [t_1, t_2]\}$ . Every time  $i$  receives PN or Learned\_PN information about  $j$  over some valid activation, it will further prune its *Inter\_Zone* activations. Note that whatever the configuration is,  $\{(\bigcup acts_{Intra\_Zone}) \cup (\bigcup acts_{Inter\_Zone})\} = [T_1, T_2]$ .

Until convergence, PN and Learned\_PN messages will be exchanged to fill *Intra\_Zone* and *Inter\_Zone* tables at every node. Note that the behavior would be the same would consider a snapshot at a particular time. Then, the size of different zones will grow and shrink with time, self-adapting to the changing topology, without message exchanges, unless an unexpected topology change happens (trajectory change).

For example, in Figure 3, node  $d$  belongs to the inter-zone table of node  $k$  over activation  $[10, 20] \not\subset [0, 10]$ , and node  $s$  is in the inter\_zone of node  $a$  over the activation  $[0, 9]$ , but at time 9, both  $s$  and  $l$  will be reachable from  $a$ , thus are Learned\_PNs to  $a$ , further increasing node  $k$ 's zone. We say that for  $t \in [0, 9]$ , node  $s$  will be a *gateway* and the link  $\overline{as}$  will be a *bridge*. In Figure 3, the inter\_zone table of node  $a$  has one member activated from  $[0, 9]$ ,  $\langle s_{[0,9]} \rangle$ .

### 3.6 Impact of Trajectory Knowledge on Zone Maintenance

As mentioned before, node trajectories information are only valid during a short period of time. Then, since a node is unable to predict the time its neighbors will change their trajectories, it biases the *QoC* to reflect the decreasing probability of the link existence. Yet, we still consider this link valid as long as not notified otherwise. Consequently, when a node is changing its trajectory, it must inform its neighbors about the induced topology change. All links it formerly had with its direct neighbors are invalidated. To do so, it sends a *New Trajectory* (NT) message to all its neighbors, and piggybacks its new coordinates and velocity. Therefore,

its neighbors are able to adapt their trees to this event. Eventually, the algorithm carries out the PN election phase again.

Upon reception of a NT message from  $j$ , node  $i$  updates its neighbor parameters and recomputes the Quality of Connectivity of the new node. All activations to  $j$  being invalid,  $i$  must notify its neighbors that the topology has changed. To do so, it sends a *Remove* (RM) message including the node that changed its trajectory as well as all Learned\_PNs depending to it. To do so, node  $i$  prunes its actual and future trees by cutting the actual and future branches connecting  $i$  to  $j$ , which consequently cuts all preferred links belonging to the pruned branches as well. Then, in order to re-connect themselves to a tree,  $j$  and  $i$  run a new election round. Even if  $i$  did not change course, it is then allowed to reconsider its previous decision given the new topology so that it always keeps an optimal tree. As mentioned before, we call *critical vertex* a vertex on which down-link topology changes does not truly affect the view up-link nodes have about the tree. Hence, the topology reorganization scope in KADER is limited to these *critical vertices*. When a RM message does not have any influence on the view up-link nodes have about their trees, it will not be forwarded, and we will therefore say that a RM message has reached a *critical vertex*.

For example, in Figure 4, if  $w$  loses its link with  $f$  during the actual activation, and reconnects with  $j$ , a change will take place in the the view node  $f$  has of its tree (see Figure 4(a)), therefore  $f$  will forward the RM message to its neighbors. But the view node  $k$  has of its tree (see Figure 4(b)) will not change, thus it will not forward the message to  $c$ . By doing so, we only remove links that really are affecting the view of the topology a node has. In fact, we limit the maintenance's scope. The size of the trees as well as their configurations will be adapted to the new topology and will be valid until another node changes course, creating a event-driven zone maintenance. Finally, we emphasize that by keeping the trajectory change's rate of a node below the beaconing rate of ordinary topology management protocols, KADER ensures a decrease in the topology management messages, thus keeping more resources for payload traffic.

## 4 KADER's Connected Dominating Set

By creating zones as mentioned in section 3.4, KADER builds **Dominating Sets**, and by keeping these zones connected with each others as described in section 3.5, it obtains a **Connected Dominating Set**. In this chapter, we propose to study KADER's computed CDS with parameters such as average zone diameter (i.e. in term of number of hops), average number of zones in the network, average ratio of remaining edges, average ratio of PNs in the network. The following results were obtained by measuring the metrics after the population of mobile nodes was distributed uniformly on a grid of 2000mx2000m with each node having a transmission range of 250m. Moreover, each node has a different stability value, but nodes' average stability is  $1/\beta = 30s$ . We will compare KADER in two different



cases : *variable density* and *constant density*.

We begin by showing in Figure 5 the topology created by KADER from an arbitrary graph  $G$  (see Figure 5(a)) to a forest and trees (see Figure 5(b)), where full lines are tree edges and dashed lines are bridges connecting different trees.

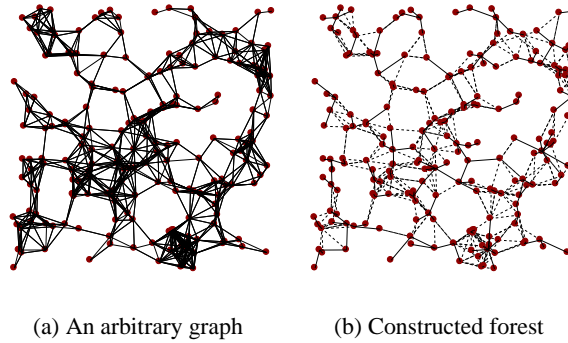


Figure 5: KADER's Connected Dominating Set

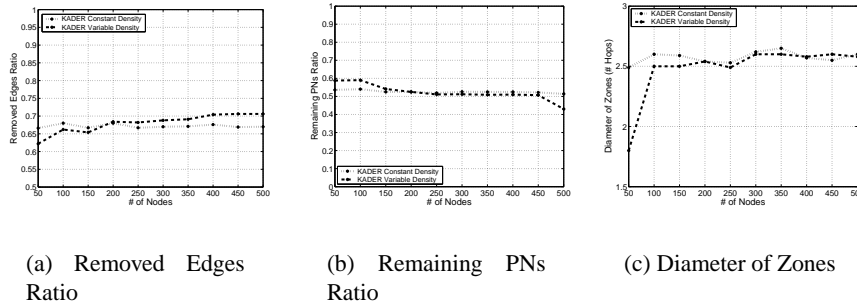


Figure 6: Properties of KADER's Connected Dominating Set

Then, the graph in Figure 6(c) shows the diameter of a zone versus the number of nodes in the network. The diameter of a zone is defined as the length of the path that has the longest hop count. If the zone diameter is fixed, then we can place an upper bound on the Intra\_zone end-to-end delay. We can clearly see in Figure 6(c) that zones in KADER are relatively stable, in both a variable and a constant density. This comes from its distance parameter in the electing criterion. Since KADER always try to create a link between nearby neighbors, neither density, nor the number of nodes have a big influence. Furthermore, the longer the zone diameter becomes, the more proactive the protocol is, and vice versa. As a consequence, KADER tends to reduce its reorganization complexity by limiting its scope to closest nodes, at a cost of extra overheads between zones. Finally, this graph shows that KADER has an upperbounded zone length. This is important since its complexity is directly proportional to this length.

The graph in Figure 6(a) shows the edges' ratio the topology algorithm is able to remove versus the number of nodes. We can see that KADER is able to remove 65% of the total number of edges. By getting rid of unnecessary links, KADER helps reducing the broadcast burden in the network, and the scalability of routing protocols. Besides, unlikely to other approaches, the objective of KADER is not necessary to obtain the minimum number of edges, since intra-zone routing is obtained at no extra cost. It is rather designed to create stable zones, such that its proactive maintainance is reduced. The conclusion is the same in Figure 6(b), which shows the remaining *preferred neighbors* ratio versus the number of nodes. KADER is able to remove 45% of PNs, which helps to reduce the broadcasting overhead in the network. Therefore, by removing 65% of unnecessary links and by only keeping a backbone of 55% of router nodes, KADER proves to be broadcasting efficient.

## 5 Overhead Complexity

It is important to compare the communication complexity of that algorithm for topology creation. The communication complexity describes the average number of messages required to perform a protocol operation. Note that this comparison does not include the complexity of route discovery.<sup>7</sup> This issue is not covered in this paper. We have selected three related protocols: MPR, TMP and DDR. We consider a network with  $N$  nodes. For the three protocols, let  $\tau$  be the rate of topology control generation, and  $p$  be the period of beaconing, or hello transmission. In the KADER case,  $\tau$  is replaced by the trajectory change' rate of a node,  $\frac{1}{\beta}$ , which trigger the topology control. As mention before, we do not use beacons, therefore  $p$  does not any significance in our algorithm.

In KADER, the network is partitioned into  $M$  zones, and each zone will have  $\frac{N}{M}$  nodes. The amount of communication overhead to build and maintain the forest is  $N$  since sending a PN election message, a forest will be constructed. To construct a zone, each node generates  $(d-1)$  messages to forward the learned PN or removed PN, where  $d$  is the hop-wise zone diameter. Therefore, each zone generates  $(d-1)\frac{N}{M}$  messages. Since there exists  $M$  zones in the network, the overall generated forward messages becomes  $(d-1)N$ . In conclusion, the total amount of communication overhead for creating the topology produced by KADER is

$$S_{KADER} = \frac{N}{\beta} (d + 1) \quad \text{messages}$$

where  $d < 3$  (see Figure 6(c)).

It can be shown that the complexity of KADER is always smaller than MPR, TMPO, or even DDR. In MPR, the communication overhead is  $S_{MPR} = pN +$

---

<sup>7</sup>KADER being zone-wise proactive, it is able to find the intra-zone paths without any increase in complexity.

$\tau R_N N$  messages,  $R_N \ll N$ .  $R_N$  is the average number of retransmissions in an MPR flooding and is proportional to the number of nodes, whereas in KADER,  $d$  reaches a threshold. TMPO is creating a communication overhead of  $S_{TMPO} = pN + \tau \cdot |MDS| \cdot 2 \cdot N$  messages<sup>8</sup>, where  $|MDS|$  is the average number of cluster-heads in the MDS. We can see that, similar to MPR, the complexity of TMPO is always bigger than KADER. Finally, similar to KADER, DDR partitionnes the network  $N$  in  $M$  zones of length  $d$ , and each zone generates  $(d - 1) \frac{N}{M}$  messages. However, DDR needs periodical messages. Therefore, the total communication overhead in DDR is  $S_{DDR} = Np + \tau(d - 1)N$  messages, where  $d < 8$ . Since  $d_{KADER} < d_{DDR}$ , KADER has a smaller communication overhead than DDR. And by removing periodic messages (which saves  $pN$  overhead messages) the communication overhead of KADER is even further reduced.

## 6 Benefit of KADER on Routing Algorithms

KADER is able to derive the most stable links from a network topology such that full connectivity is always guaranteed. It then becomes interesting to analyze the benefit routing protocols can extract from it.

### 6.1 Efficient Routing

KADER is able to group nodes into a set of zones, which proactively maintains routes between every nodes belonging to the same zone. Therefore, any routing protocol using KADER would not need any overhead for *Intra\_Zone* routing. A reactive approach, such that AODV [13], would take great help of the CDS created in KADER by reducing the overhead of its *route discovery* procedure. This would create an hybrid routing protocol, using proactive intra-zone routing, and on-demand zone-level routing. On the other hand, similar to OLSR [10] using MPR [9], a proactive protocol would be able to benefit from KADER broadcasting efficiency by improving its scalability and its end-to-end delay.

### 6.2 Energy Efficiency

In KADER, during the construction of the forest, every node elected its *Preferred Neighbor* partly depending on the energy needed to reach it. Indeed, the transmission range of the *Intra\_zone* routing is always adapted to reach only the desired PN. Hence, KADER makes proactive *Intra\_Zone* routes optimal in term of energy data flow generated and forwarded by each node, further reducing the energy used for routing and increasing the channel capacity. And since KADER does not use beacons, routing protocols using KADER could reach routing energy efficiency.

---

<sup>8</sup>We are assuming a simplified case, where we are not considering the complexity overhead triggered by gateways and doorways, which further increases TMPO complexity.

## 7 Conclusion

In this paper, we have presented a novel approach to topology management algorithms, called Kinetic Adaptive Dynamic topology management for Energy efficient Routing (KADER). It employs nodes' trajectory knowledge to get rid of periodic beacons. The major properties of KADER are *Low Complexity, Broadcasting Efficiency, and Energy Efficiency*.

We showed that by using trajectory knowledge to predict nodes future positions, KADER was able to dynamically create and maintain a connected dominating set without using periodic beacons. Results pointed out that the CDS created and maintained by KADER was composed of only 45% of nodes, and 65% of links composing the original network, while always being able to keep a full connectivity between every node. Then, using a similar denomination than those in routing protocols, KADER can be classified as a reactive topology management protocol, since it is only triggered when an event occurs. Therefore, by initiating topology maintenance only when a node is changing course and not periodically, KADER reaches energy efficiency for topology management. Moreover, KADER is able to obtain a lower maintenance complexity than other topology management algorithms.

In our future work, we will evaluate the performance of KADER with different routing protocols under various traffic load and mobility rate.

## References

- [1] G. J. Pottie, "Wireless sensor networks", *Proc. IEEE Information Theory Workshop*, Killarney, Ireland, pp. 139-140, June 1998.
- [2] Navid Nikaein, Houda Labiod, and Christian Bonnet, "Distributed Dynamic Routing Algorithm for Mobile Ad-Hoc Networks," *Proc. MobiHOC 2000*, USA/Boston.
- [3] C. Gentile, J. Haerri, and R. E. Van Dyck, "Kinetic minimum-power routing and clustering in mobile ad-hoc networks," *IEEE Proc. Vehicular Technology Conf. Fall 2002*, pp. 1328-1332, Sept. 2002.
- [4] R.J. Fontana and S.J. Gunderson, "Ultra-wideband precision asset location system", *IEEE Conf. on Ultra Wideband Systems and Technologies*, pp. 147-150, 2002.
- [5] C. Gentile and Luke Klein-Berndt, "Robust Location using System Dynamics and Motion Constraints", To be presented to the *IEEE International Conference on Communications*, Paris, 20-24 June, 2004.
- [6] R. Prakash "Unidirectional links prove costly in Wireless Ad-Hoc Networks", *In Proc. of the Discrete Algorithm and Methods for Mobile Computing Computing and Communications (DialM)*, Seattle, WA, August, 1999.
- [7] P. Krishna *et al*, "A cluster-based approach for routing in dynamic networks" *ACM SIGCOMM Computer Communication Review*, pp. 49-65, Apr. 1997.
- [8] L. Bao and J.J. Garcia-Luna-Aceves, "Topology Management in Ad-Hoc Networks", *Proc. MobiHOC 2003*, USA/Annapolis.
- [9] A. Laouiti *et al*, "Multipoint Relaying: An Efficient Technique for Flooding in Mobile Wireless Networks", *35th Annual Hawaii International Conference on System Sciences (HICSS'2001)*, Hawaii, USA, 2001.
- [10] T. Clausen *et al*, "Optimized Link State Routing Protocol", *IEEE INMIC*, Pakistan, 2001.
- [11] A. Amis *et al*, "MaxMin D-Cluster Formation in Wireless Ad-Hoc Networks", *In Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, Mar. 1999.
- [12] M.R. Garey and D.S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*, Freeman, Oxford, UK, 1979.
- [13] C.E. Perkins *et al*. "Ad Hoc On-Demand Distance Vector Routing", draft-ietf-manet-aodv-13, 2003.

## A Correctness

**Definition 2** An arbitrary undirected time dependent graph  $G(t)$  is defined as  $G(t) = (V, E(t))$ , where  $V$  is the set vertices, and  $E(t)$  is the set of edges at time  $t$ .

**Theorem 1** For any graph  $G(t)$ , let  $G'(t) = (V, E'(t))$  be the subgraph obtained by connecting each vertex  $V$  to its preferred links  $E'(t)$ . Then  $G'(t)$  is a forest.

*Proof:* Let  $G(t)$  be the original graph at time  $t$ , and let  $G'(t)$  be the graph obtained by executing the KADER algorithm for each vertex  $v \in V$  at time  $t$ . We first recall that the main idea is to select, for each node  $v \in G(t)$ , a neighbor that has the maximum  $QoC$ . In order to prove that  $G'(t)$  does not contain any cycle  $C = v_i, v_{i+1}, \dots, v_{i-1}, v_i$ , lets suppose the contrary, and let  $v_i$  be the vertex of  $C$  with the biggest  $QoC$ .

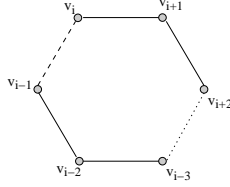


Figure 7: The proof of theorem 1

Let us consider two vertices of  $v_{i-1}$  and  $v_{i+1}$  adjacent to  $v_i$  in  $C$  (Figure 7). Without loss of generality, assume that the algorithm on  $v_i$  chosen an adjacent vertex  $v_{i+1}$  (if neither  $v_{i-1}$  nor  $v_{i+1}$  had been chosen,  $C$  is not a cycle). Consider now the execution of the algorithm on  $v_{i-1}$ . We will show that such node will not choose  $v_i$ , thus implying that  $C$  is not a cycle.

Lets define  $f(v_i, v_{i+1})$  as the  $QoC$  function between  $v_i$  and  $v_{i+1}$ . Since  $v_i$  chose  $v_{i+1}$ ,  $f(v_i, v_{i+1}) > f(v_i, v_{i-1})$ . And by  $v_{i-2}$ 's decision to choose  $v_{i-1}$ ,  $f(v_{i-2}, v_{i-1}) > f(v_{i-2}, v_{i-3}) > f(v_i, v_{i+1})$ ,  $f$  being an monotone increasing function. Therefore,  $f(v_i, v_{i-1}) < f(v_{i-1}, v_{i-2})$ . This proves that  $v_{i-1}$  will not choose  $v_i$  as PN, and  $C$  will not be a cycle. ■

**Theorem 2** For any PN activation  $act(v_i, v_{i+1})[t_1, t_2]$ , and any graph  $G'(t_1, t_2) = (V, E'(t_1, t_2))$  obtained by connecting each vertex to is preferred links  $E'(t_1, t_2)$  activated during  $[t_1, t_2]$ ,  $G'(t_1, t_2)$  is then always a forest at every time instant included in  $[t_1, t_2]$ .

*Proof:* When a node  $v_i$  elects a PN  $v_{i+1}$  during an activation  $act(v_i, v_{i+1})[t_1, t_2]$ , it means that  $\forall t \in [t_1, t_2]$ ,  $f(v_i, v_{i+1})(t) > f(v_i, v_k)(t)$ ,  $\forall k$ . Since nodes share

a common clock, all their current left activation are equal to the current time and will thereafter be considered as 0, past activations being irrelevant.

If a node  $v_i$  elects a PN  $v_{i+1}$  during an activation  $act(v_i, v_{i+1})[0, t_1]$ , without loss of generality,  $v_{i+1}$  can elect a node  $v_{i+2}$  as PN during an activation  $act(v_{i+1}, v_{i+2})[0, t_2]$ . Since the algorithm prunes the activation between  $v_i$  and  $v_{i+2}$  as  $([0, t_1] \cap [0, t_2])$ , we must consider two cases. In the first case, the initial activation  $act(v_i, v_{i+1})[0, t_1]$  is less or equal than  $act(v_{i+1}, v_{i+2})[0, t_2]$ , thus it will be kept unaltered during the forwarding steps. In the second case, the algorithm prunes the initial activation. The forwarded activations are two separated and mutually exclusive activations.

Lets consider  $t_2$  smaller or equal to  $t_1$ . Then, following the development in the proof of Theorem 1, at some point, node  $v_{i-1}$  could elect node  $v_i$  during an activation  $act(v_i, v_{i-1})[0, t_3]$ , as  $t_3 \leq t_1$ .  $[0, t_3]$  is the remaining activation after multiple pruning at each node in the path. Then it means that  $\forall t \in [0, t_3]$ ,  $v_{i-1}$  could elect  $v_i$  as PN, thus creating a cycle during this time. Theorem 1 prove that this situation was not possible, since  $\forall t \in [0, t_3]$ , we obtained a stable tree which is not a function of  $t$ . Then, during the activation  $act(v_i, v_{i-1})[0, t_3]$ ,  $C$  does not contains any cycle.

Since the initial activation has been pruned, we still need to consider the case of the remaining activation  $([t_3, t_1])$ . Without loss of generality, lets consider that this activation has been pruned at a single node  $v_{i+2}$ . This node has the possibility to elect  $v_{i+1}$  as PN (mutual election), updating the mutual activation as the union of their respective ones. Note that this case does not create a cycle.  $v_{i+2}$  can otherwise elect another node, say  $v_{i+3}$ . Since  $[t_3, t_1] \cap [0, t_3] = \emptyset$ ,  $\overline{v_{i+2}v_{i+3}}$  is then a branche of a different and independant tree and the situation is independant to the previous one. Therefore, this neither creates a cycle, which concludes the proof. ■

**Theorem 3**  $\forall G$ , let  $G'$  be the subgraph obtained by connecting each node to its preferred links during their respective activations. Then  $G'$  is a forest at every time instant.

*Proof:*  $\forall$  node  $v_i$ , since all its PNs activation intervals are mutually exclusive ( $\bigcap (act(v, v_1), \dots, act(v, v_n)) = \emptyset$ ), from Theorem 2, we can conclude that KADER always yield to a forest at every time instant. ■