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

Research Report RR-04-114

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

August 25th, 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

[1]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 relaying 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.

Index Terms

Mobile Ad Hoc Networks, Trajectory, Topology Control, Topology Management, Mobility Prediction, Prediction-based, Beacon-less, Dead-Reckoning.

Contents

List of Figures

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, nodes that are part of the backbone must be close to each other and connected. In graph theory, the Minimum Dominating Set 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 (MDS). 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. There exists many hierarchical approaches that are able to distributively create CDS, like DDR [2], TMPO [8], and MPR [10].

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 2-level hierarchical topology control approach denoted as Kinetic Adaptive Dynamic topology management for Energy efficient Routing (**KADER**). Indeed, 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, and to improve the overall delay. 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. The idea is to model nodes' positions as a piece-wise linear trajectory, as opposed to a fixed position for a single time instant, such that we can predict a node future position, and according to it, adapt the forest, therefore the dominating set. The protocol only requires minimal updating when a node changes course, since most of the links remain valid. Indeed, a node simply notifies its neighborhood upon an unpredicted event, such as trajectory changes, instead of periodically sending beacons. Therefore, the algorithm can be seen as a per-event approach. Since energy is not wasted on message transmissions, KADER reaches energy efficiency on topology management. To sum up, KADER combines the concepts of forest, zone and trajectory knowledge in order to achieve *low complexity, broadcasting efficency, and energy efficency*.

Similar to KST [3], TMPO [8], MPR [9, 10], or DDR [2], our approach seeks a CDS and does not need any negotiation round to obtain it. However, different from KST, our approach does not require a full topology knowledge to use the trajectory information, therefore restricting the complexity overhead of nodes' trajectory changes. KADER is further able to create shortest path routes between every node contained in a zone, and not to a single source as it is the case in KST. Unlike TMPO, 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. Contrary to MPR which requires global topology knowledge to build a CDS, KADER is only based on local information. Finally, what dissociates our approach from TMPO,MPR, and DDR, is that KADER uses trajectory knowledge, and consequently does not need periodic updates 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 topology control 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 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 [1] , thereafter called *mobility*. We describe in this section the way KADER is able to extract those trajectories. We further insist on the fact that KADER's neighbor discovery procedure distinguishes itself from regular protocols since it is neither periodically performed, nor initiated after some adaptive intervals depending on nodes mobility. It is rather triggered only when the neighborhood effectively changes.

Consider nodes moving in a mobile ad hoc network. In order to compute neighbors' trajectories, nodes must first exchange their respective mobility parameters. They do so by broadcasting their parameters to their immediate neighbors. Such messages are 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 [2], one can assume that each such node, say i, follows a linear trajectory. Its position as a function of time is then described by

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

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 velocity. Let us consider node j as a neighbor of i. In order to let node i 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 $a_{ij} \geq 0$, $c_{ij} \geq 0$. Consequently, a_{ij}, b_{ij}, c_{ij} are 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, is 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.

[1] We are considered moving in a two-dimensional plane.

[2] The time required to transmit a data packet is orders of magnitude shorter than the time the node is moving along a fi xed trajectory.

3 Kinetic Adaptive Dynamic Topology Management Algorithm

We propose to construct a self-adapting forest from an ordinary network that will consist of non-overlapping dynamic trees, thereafter called zones[3]. Each zone is kept connected with its neighboring zones through gateway nodes, thus making the whole network 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 non-periodic message communicating its new mobility parameters. Following this event, the forest will adapt itself to the new topology (see section 3.6).

The algorithm described hereafter 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.

3.1 Neighborhood Discovery

Basically, KADER's neighbor discovery procedure makes a node detect changes in its neighborhood without exchanges of periodical beacon messages. Upon completion of this discovery procedure, nodes in the network will have a fair knowledge of their neighborhood, and as long as their neighbors keep on moving along their initial linear trajectories, there will be no need to refresh it. In other words, no periodical beacon messages are required, and it is only when a node's neighborhood effectively changes that a new neighbor discovery procedure is initiated. Nodes keep their neighbor parameters in a so-called neighboring table.

In order to construct this table, each node broadcasts a single message to indicate its presence in the neighborhood, and to transmit its parameters. 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 invalid due to an unpredicted event (a trajectory change), the respective node spontaneously advertises its new parameters, refreshing the predictions in a event-driven way.

This table consists of several information related to a node's direct neighborhood: neighboring identities nid, neighboring mobility, such as x, y, dx, dy. It also contains neighboring stability parameters β and neighboring last trajectory change time t_i. As a node changes course, KADER considers it as a new node in the network. Since *nids* cannot be changed, and that we do not forward nodes trajectories in the network in order to distinguish a node from the same node that has changed its trajectory, we add a Trajectory Counter (*Tc*), mentioning the number of trajec-

[3]We will later use the term tree and zone interchangeably.

tory 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. For example in Figure 3, the neighboring table of node k has three neighbors, which are $< (f, x, y, dx, dy, \beta, t_f), (c, x, y, dx, dy, \beta, t_c), (d, x, y, dx, dy, \beta, t_d) >$.

3.2 Preferred Neighbor Election

A node's *Preferred Neighbor* is a dedicated neighbor through which a node sends, receives, or forwards packets. It therefore represents the link on which a node sends its traffic. The criterion determining this neighbor depends on the application needs. We are introducing in this section a link parameter that better represents the routing orientation of our topology management algorithm. As we will explain later, a zone in KADER is proactively maintained, and each node keeps a table that describes the path to reach any destination contained in its zone. Yet, we do not want these tables to hold the best path in term of hops, or of broadcast coverage, but in term of power needed to reach a destination. Energy consumption in manet being a critical issue, it is attractive to use this cost as the link parameter from a source to a destination such that we can extend the network lifetime.

To do so, we propose to use the *stochastic power function* reflecting a link energy cost, and its existence's uncertainty as the cost of a link between two nodes. Defining the *Quality of Connectivity* (QoC) as a function of this cost, 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 the Appendix 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

As explained above, we wish to obtain a criterion that is able to satisfy two objectives. The first one is to represent the energy needed to reach a neighbor. It is interesting to be able to reach a node using the an energy as low as possible. The second one is the neighbor's stability. A node's stability is the probability that it has to evolve as predicted, in other words, the probability of not having changed its initial trajectory. Since KADER does not periodically update its set of links, we want the links chosen by KADER to remain stable enough such that reorganization overheads could be kept low. Therefore, KADER will not elect the closest neighbor, but may decide to choose a further, yet more stable one.

The *power cost* function, required to transmit between nodes i and j at time t, is defined as

$$P_{ij}(t) = \kappa D_{ij}^{\alpha}(t) + \gamma$$

where $\alpha \geq 2$ and for some constants γ and κ. Without loss of generality, we assume $\alpha = 2$. The constant κ depicts the overhead due to MAC control messages

and also possible retransmission probability between nodes i and j. Since we do not consider a MAC layer at this stage, and assume a perfect channel without loss or collisions and where all sent packets are successfully received, we set $\kappa = 1$. Then, γ represents a constant charge for each transmission, including the energy needed for signal processing and internal computation. However, since KADER does not put any extra burden on any particular node[4], γ is common to all nodes and is not of great significance when comparing power costs. Therefore, without loss of generality, we assume $\gamma = 0$. [5] By choosing power as the cost, one obtains minimum power routes that help preserve battery life.

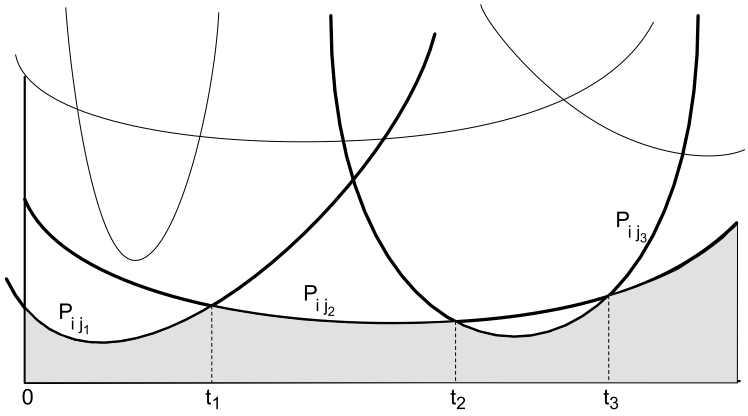


Figure 1: Graphical representation of the Power function at node i.

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*

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

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 $t = t_0 \to \infty$.

[4]KADER is based on a tree and not on a cluster hierarchy

[5]Therefore, Power and Distance will later be interchangeably used.

However, nodes keep their trajectories only during 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 remains valid a longer time). To deal with this problem, we bias the power function toward t_0 since as $t \to \infty$ nodes will have changed trajectory anyway.

We define

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

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. Figure 7 is an example of the evolution of $\frac{1}{\beta}$ versus nodes' maximum velocities in a Random Waypoint Mobility Model (RWM).

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*.

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

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 from electing 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

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

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

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

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

Now we have seven parameters a_{ij}, b_{ij}, c_{ij}, β_{ij}, t_{ij}, $timeout_j$ and $const$ describing $\widetilde{P}_{ij}(t)$ as criteria for a preferred link between two nodes. The constant parameter $const$, which controls the transition slop between 1 and 0, is in fact common to every node and 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*, $\widetilde{P}_i(t) = \min_t\{\widetilde{P}_{ij_1}(t), \widetilde{P}_{ij_2}(t), \widetilde{P}_{ij_3}(t)\}$, 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 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

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

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

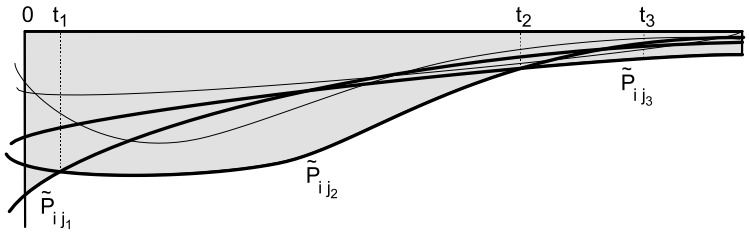


Figure 2: Graphical representation of the QoC function at node i.

3.2.2 Election Algorithm

In the previous section, we described a criterion based on which a node could obtain energy saving yet stable links. However, since our protocol predicts future

topology configurations, when a node finds such links, KADER needs to determine how long this link will remain optimal compared to other neighbors' links. In this section, we introduce a time interval, called *activation*, during which a link between two nodes remains optimal. Therefore, a link between a node and its neighbor will be *activated* over a certain period, then later *de-activated* when another link becomes optimal. The constructed forest will then be the set of active links at a certain time.

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 k other neighbors of i. The criterion is defined as

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

We further define a fixed time interval $[t_1, t_2]$, called activation, in which j remains i's PN. We say that j is *activated* by i from t_1 to t_2 , and during this time interval, 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

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

Therefore, the set (i, j,$act(i,j)[t_1,t_2]$) uniquely identifies a *preferred link* between node i and node j activated from t_1, and 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_set_i, regroups all actual and predicted future PNs of node i. This set is defined as :

$$\begin{aligned} \forall j \in nb_i \ j \in PN\_set_i \quad & iff \\ \exists \ t_1, t_2 \quad & s.th \quad act(i,j)[t_1,t_2] \neq \emptyset \end{aligned}$$

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

Finally, note that PN_set_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 from which activations are obtained, the set of activations contained in PN_set_i are mutually exclusive (see Figure 2 where $[0, t_1] \bigcap [t_1, t_2] = \emptyset$).

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 time interval, iff j is in the neighborhood of i and has the biggest QoC among its neighbors during this time 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 all activated links. In the Appendix, we prove that, whatever the network topology is, this approach always yields to a forest at every time instant.

3.3 Forest Construction

For every time instant, a forest is constructed by connecting each node to its PN. However, since every node knows in advance the set of its actual and future PNs, this process can be performed without any exchange of messages. Therefore, a self-adapting forest is built when, at each time instant, each node is connected to its activated PN. We depict hereafter the process of notifying neighboring nodes of their PN election.

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 $(act(i,j)[t_1,t_2] \bigcap act(i,k)[t_3,t_4]) = \emptyset$ and $(act(i,j)[t_1,t_2] \bigcup act(i,k)[t_3,t_4]) = [t_1,t_4]$. 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

KADER aims at regrouping closed-by nodes into a zone in order to provide energy efficient communications. In previous sections, we saw how nodes are able to elect and reach their PNs, and how the forest is constructed. At this phase, we illustrate nodes' attempt to expand their own view about the tree they belong to by completing an intra-zone table containing the best path to reach all nodes in their zone. This process creates and proactively maintains local minimal dominating sets (MDSs), each local MDS representing a zone.

When a node i gets elected by a neighbor j, it then locally notify all its neighbors of this election. To do so, i sends a so called *Learned_PN* message

$Learned\_PN_i = (i, \overline{ij}_{[t_1,t_2]} : \overline{jk}_{[t_3,t_4]} : \overline{jl}_{[t_5,t_6]}; ...; \overline{iu}_{[t_7,t_8]} : \overline{uv}_{[t_9,t_{10}]})$, 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 $(act(i,j)[t_1,t_2] \bigcap act(j,k)[t_3,t_4]) \neq \emptyset$.*

A direct consequence of Definition 1 is that i, j and k are belonging to the same tree over the activation $(act(i,j)[t_1,t_2] \bigcap act(j,k)[t3,t_4])$, while $act(j,k)[t_3,t_4] \setminus (act(i,j)[t_1,t_2] \bigcap act(j,k)[t_3,t_4])$ is considered as a separate tree connecting only j and k.

Upon reception of this message, each tree member updates its intra_zone table iff PNs learned through this message have valid activations, and re-advertises to its neighbors if it is not a leaf node[6]. 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 a time interval $[t_1,t_2]$, 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 $Learned\_PN_j = (j, \overline{ji}_{[t_1,t_2]})$. Then, the local view of k's tree is that, over $(act(i,j)[t_1,t_2] \bigcap (j,k)[t_3,t_4])$, i is reachable through j.

If node j is chosen as the PN of many nodes through a period of time, then j forwards their decisions encapsulated in a Learned_PN message, that is $Learned\_PN_j$ $=(j, \overline{ji}_{[t_1,t_2]} : \overline{ji'}_{[t_3,t_4]} : \overline{ji''}_{[t_5,t_6]} : ...)$. It indicates that node j is forwarding the new learned PNs or new tree members $\overline{ji}_{[t_1,t_2]} : \overline{ji'}_{[t_3,t_4]} : \overline{ji''}_{[t_5,t_6]} : ..$ on the tree. Other neighboring nodes of i add j to the Learned_PN field corresponding to i in their intra-zone table if node i already exists in their table over a valid activation. In this way, we say that j is learned to be the PN of i. Note that node i is also learned by the neighboring nodes of j. Node i creates a Learned_PN message if the set of PNs learned by i is non-empty. This set is denoted by $Learned\_PN_i$. Node i forwards the $Learned\_PN_i$, if it is not a leaf node. Hence, the intra-zone table is only updated if the information originates from PN members and not from $Learned\_PN$ members or an ordinary neighbor.

For example, in Figure 3, let us consider the scenario where node k wants to communicate to one of the nodes belonging to its tree during $[0,10]$. According to its *Intra_Zone* table (see Table 1(a)), node k can reach nodes a, b, q, w, r, j through node f over $[0,10]$, while other nodes d, c are directly reachable. Since node d is a

[6]A leaf node is a node which only has a single neighbor and which is never a PN.

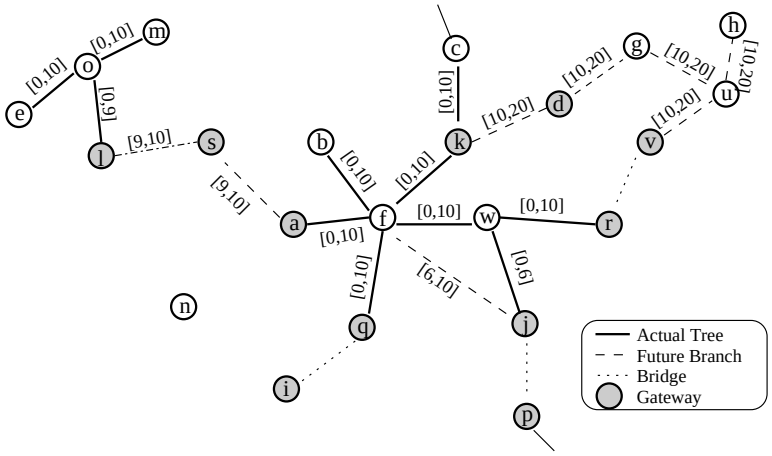


Figure 3: Constructed forest

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

(a) $Intra\_ZT_k$

PN	Learned_PN
$w_{[0,10]}$	$r_{[0,10]}$
	$j_{[0,6]}$
$\mathbf{k}_{[0,10]}$	$c_{[0,10]}, d_{[0,10]}$
$j_{[6,10]}$	-
$b_{[0,10]}$	-
$a_{[0,10]}$	$s_{[9,10]}, l_{[9,10]}$
$q_{[0,10]}$	-

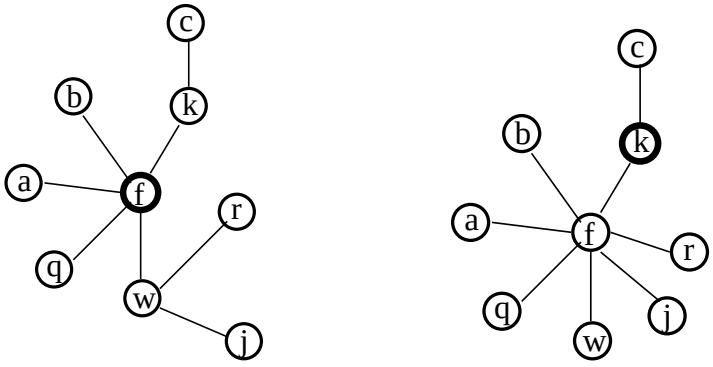
(b) $Intra\_ZT_f$

Table 1: Intra-zone table of nodes k and f regarding Figure 3

future PN of k (f being the actual one), it is not yet accessible to nodes in k's tree (at least not through k). So, regarding to $Intra\_ZT_k$, the next hop to reach nodes a, b, q, w, j, r is node f and not c, d. However, node k does not see the change in the topology that arises at $t = 6$ between nodes f,w, and j. f initially had j as Learned_PN, thus reachable through w from over $[0, 6]$, but then, node j switched and elected f as its PN from over $[6, 10]$. Note that this had been done without any messages exchanged, since this configuration had been predicted by the three nodes during the neighborhood discovery.

Then, as shown in Table 1(a) and 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) do not see this change in the topology, and do not receive any message about it. The topology reorganization scope in KADER is 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.



(a) Node f (b) Node k

Figure 4: Tree view of nodes k and f.

3.5 Self-Adaptive Inter-Zone Clustering

In the previous section, we explained how KADER is able to create and maintain MDSs. However, we did not describe how these sets are kept connected to each others. Besides, the objective of our algorithm is not only to create MDSs, but to connect them as well in order to obtain a Connected Dominating Set (CDS). In this part, we show how KADER uses a different table, called *Inter_Zone* table, that regroups the connections to different surrounding zones. Every node belonging to this table are named as *gateway* and the link connecting two zones as *bridge*. Until convergence, PN and Learned_PN messages are exchanged to fill *Intra_Zone* and *Inter_Zone* tables attached to every node. We further emphasize that these messages are not periodic, but sent only when new information about a zone or surrounding zones appears. Then the size of different zones will grow and shrink over time, which makes them self-adapting to the changing topology without message exchanges unless an unexpected topology change happens (trajectory change).

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 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 $\{[T_1, T_2] \backslash [t_1, t_2]\}$. 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 $\{[T_1, T_2] \backslash [t_1, t_2]\}$. 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, $\{(\bigcup acts_{Intra\_Zone}) \bigcup (\bigcup acts_{Inter\_Zone}\} = [T_1, T_2]$.

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 $[0, 9]$. 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 is a *gateway* and the link $\overline{as}$ is a *bridge*. In Figure 3, the inter_zone table of node a has one member activated from $[0, 9]$, $< s_{[0,9]} >$.

3.6 Impact of Trajectory Knowledge on Zone Maintenance

Zone maintenance consists of several tasks, such as removal of irrelevant links, acquisition of new neighbors, notification of link errors. Most of topology control algorithms perform these tasks periodically. What makes KADER unique is its ability to do these tasks in a complete per-event manner (i.e. non periodically). Since it predicts future topology configurations, KADER triggers a zone maintenance only when those predictions appear to have failed, in other words, when nodes changed their trajectories. In this section, we will show how a particular message, called *New Trajectory* (NT) message, is used to inform neighboring nodes of any topology changes, and to trigger a local maintenance process. Therefore,

KADER's zone maintenance can be seen as per-event based.

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 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.

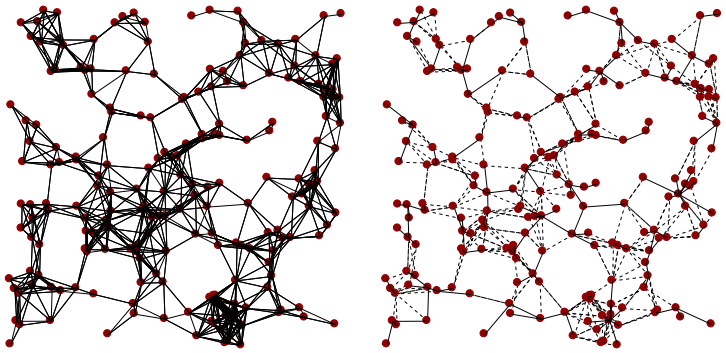
Upon reception of a NT message from j, node i updates its neighbor parameters and recomputes the Quality of Connectivity of the new node. All activations to j being invalid, i must notify its neighbors that the topology has changed. To do so, it sends a *Remove* (RM) message including the node that changed 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 do 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 is not forwarded, and we therefore say that a RM message has reached a *critical vertex*.

For example, in Figure 4, if w looses its link with f during the actual activation and reconnects with j, a change takes place in the view node f keeps 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)) does not change, thus it does 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 are adapted to the new topology and are valid until another node changes course, creating an 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 (see section 3.4), KADER builds *Minimal Dominating Sets*, and by keeping these zones connected with each others (see section 3.5), it obtains a *Connected Dominating Set* (CDS). In this section, we 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 are 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*.

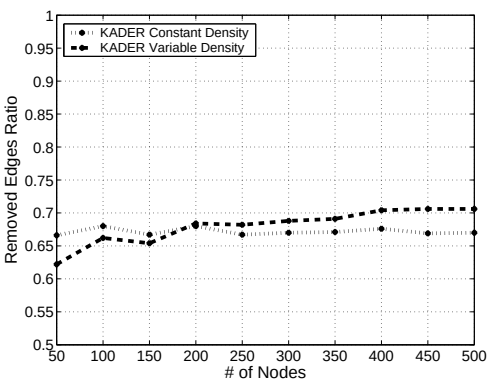
We begin by showing in Figure 5 the topology created by KADER from an arbitrary graph G (see Figure 5(a)) to a forest and trees (see Figure 5(b)), where solid lines are tree edges and dashed lines are bridges connecting different trees.



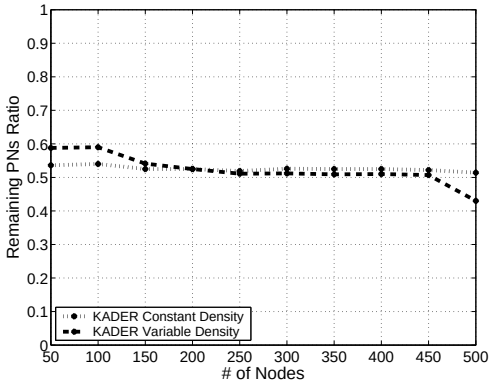
(a) An arbitrary graph (b) Constructed forest

Figure 5: KADER's Connected Dominating Set

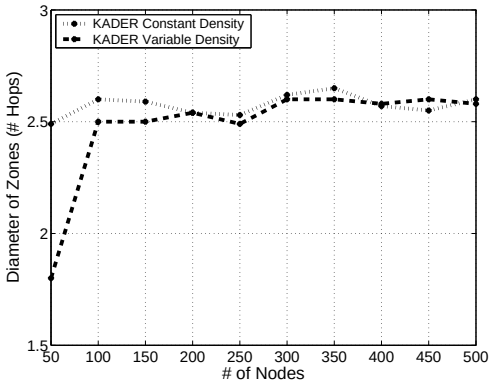
The graphs in Figure 6(a) and 6(b) show the edges' ratio the topology algorithm is able to remove versus the number of nodes and *preferred neighbors* ratio versus the number of nodes. We can see that KADER is able to remove 65% of the total number of edges. By getting rid of unnecessary links, KADER 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 maintenance is reduced. As seen in 6(b), KADER is able to remove 45% of PNs, which helps to reduce the broadcasting overhead in the network. Therefore, by removing 65% of unecessary links and by only keeping a backbone of 55% of router nodes, KADER



(a) Removed Edges Ratio



(b) Remaining PNs Ratio



(c) Diameter of Zones

Figure 6: Properties of KADER's Connected Dominating Set

proves to be broadcasting efficient.

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 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 tries to create a link between nearby neighbors, neither density nor the number of nodes have a big impact on the zone's diameter. Note that 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 only to the closest nodes, at a cost of extra overheads for intra_zone communications. Finally, this graph shows that the electing criterion gives an upperbounded zone length. This is important since KADER's complexity overhead is directly proportional to this length.

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 [7]. 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 τ be the rate of topology control generation, and p be the period of beaconing, or hello transmission. In the KADER case, τ is replaced by the trajectory change' rate of a node, $\frac{1}{\beta}$, which trigger the topology control. As mentioned before, we do not use beacons, therefore p does not have 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 number of generated forward messages becomes $(d-1)N$. In conclusion, the total amount of communication overhead for creating the topology produced by KADER is

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

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

[7] KADER being zone-wise proactive, it is able to find the intra-zone paths without any increase in complexity.

It can be shown that the complexity of KADER is always smaller than MPR, TMPO, or even DDR (see Fig 8). In MPR, the communication overhead is $S_{MPR} = pN + \tau R_N N \quad messages, \quad R_N \ll N$. 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 $S_{TMPO} = pN + \tau \cdot |MDS| \cdot 2 \cdot N \quad messages$[8], 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 larger than KADER. Finally, similar to KADER, DDR partitiones 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 $S_{DDR} = Np + \tau(d-1)N \quad messages$, 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.

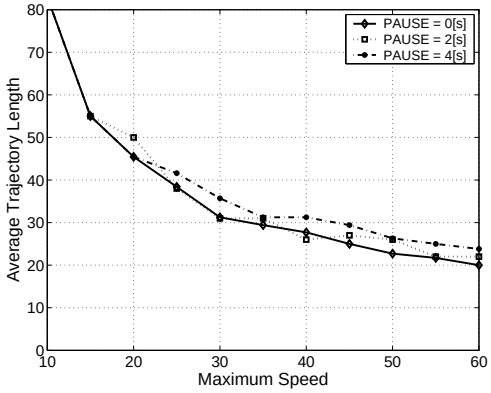


Figure 7: Average nodes' trajectory length ($\frac{1}{\beta}$) under the Random Waypoint Mobility Model

Figure 7 depicts results we obtained when attempting to have an insight of fair values for $\frac{1}{\beta}$. The benefits of KADER depending greatly on it, this might be interesting to try to fix a range for this parameter. We simulated with ns-2 a mobile ad hoc network moving according to a Random Waypoint Mobility Model. We obtained these values by computing the average time between the moment a node starts heading to its final destination and the time its reaches it. We simulated it with different PAUSE times, and with nodes' maximum velocities variating from 10m/s to 60m/s. The results are interesting since even with a maximum velocity of 60m/s, nodes have an average trajectory length of 20s.

Figure 8 shows the complexity overhead of KADER, TMPO, MPR and DDR versus the number of nodes. We can see that as the size of the network grows, so

[8]We are assuming a simplifi ed case, where we are not considering the complexity overhead triggered by gateways and doorways, which further increases TMPO complexity.

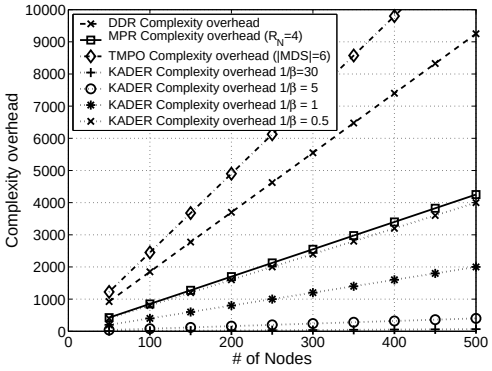


Figure 8: Comparision of KADER's complexity overhead with other topology control protocols

is the overhead. However, MPR, TMPO and DDR never reach the low complexity obtained by KADER. Since KADER's overhead is proportional to the factor $\frac{1}{\beta}$, we computed it for four different values of $\frac{1}{\beta}$: 0.5s, 1s, 5s and 30s. These values are smaller than what Figure 7 could suggest. We did this on purpose in order to consider penalizing scenarii for KADER. When $\frac{1}{\beta} = 0.5s$, this means that, every half a second, the CDS needs to be updated since a node changed its trajectory. In this configuration, we can see that KADER's complexity overhead is similar to MPR's using a periodic update of half a second too. However, as we increase $\frac{1}{\beta}$, therefore making use of KADER's predictions, we see that our protocol outclasses all other ones.

To conclude, we found that a fair value for $\frac{1}{\beta}$ could be established to 30s. This shows KADER's low complexity overhead compared to other topology control protocols, and further justifies our approach not to periodically update the constructed topology, but instead to predict trajectories.

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 guarantied. It then becomes interesting to analyze the benefit routing protocols can obtain from it.

6.1 Efficient Routing

KADER is able to group nodes into a set of zones, which proactively maintains routes between every node belonging to the same zone. Therefore, any routing protocol using KADER does not need any overhead for *Intra_Zone* routing. A reactive approach, such that AODV [13], takes great help of the CDS created in KADER by reducing the overhead of its *route discovery* procedure. This creates 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 is 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 elects 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 terms of energy data flow generated and forwarded by each node, further reducing the energy used for routing and increasing the channel capacity. Since KADER does not use beacons, routing protocols using KADER reach routing energy efficiency.

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 Efficency, and Energy Efficency*.

We showed that by using trajectory knowledge to predict nodes future positions, KADER is 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 "Unididirectional 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 Wirless 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-completness*, 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 of 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 $C = v_i, v_{i+1}, \ldots, v_{i-1}, v_i$, let us suppose the contrary, and let v_i be the vertex of C with the biggest QoC.

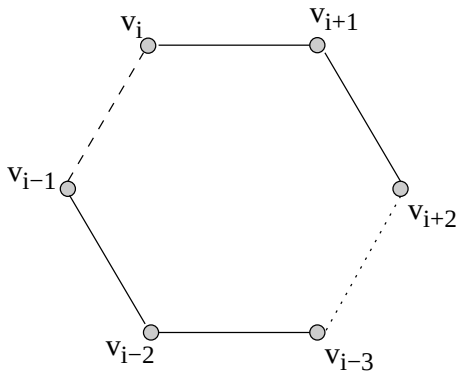


Figure 9: The proof of theorem 1

Let us consider two vertices of v_{i-1} and v_{i+1} adjacent to v_i in C (Figure 9). 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 chooses 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 its 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 $([0, t_1] \bigcap [0, t_2])$, 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 is 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.

Let us 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 $act(v_i, v_{i-1})[0, t_3],\ as\ t_3 <= t_1$. $[0, t_3]$ is the remaining activation after multiple prunning 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 is not possible, since $\forall t \in [0, t_3]$, we obtain 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 contain 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, let us 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 $[t_3, t_1] \bigcap [0, t_3] = \emptyset$, $\overline{v_{i+2}v_{i+3}}$ is then a branch 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 ($\bigcap (act(v, v_1), \ldots, act(v, v_n)) = \emptyset$), from Theorem 2, we can conclude that KADER always yield to a forest at every time instant. ■