Distributed Node Location in clustered multi-hop wireless networks

Nathalie Mitton, Eric Fleury

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Distributed Node Location in clustered multi-hop wireless networks.

Nathalie Mitton - Eric Fleury

N° 5723

.

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--5723--FR+ENG

Distributed Node Location in clustered multi-hop wireless networks.

Nathalie Mitton - Eric Fleury

Thème COM — Systèmes communicants
Projet ARES

Rapport de recherche n° 5723 — . — 30 pages

Abstract: *Ad hoc* routing protocols proposed in the MANET working group are all flat routing protocols and are thus not suitable for large scale or very dense networks because of bandwidth and processing overheads they generate. A common solution to this scalability problem is to gather terminals into clusters and then to apply a hierarchical routing, which means, in most of the literature, using a proactive routing protocol inside the clusters and a reactive one between the clusters. We previously introduced a cluster organization to allow a hierarchical routing and scalability, which have shown very good properties. Nevertheless, it provides a constant number of clusters when the intensity of nodes increases. Therefore we apply a reactive routing protocol inside the clusters and a proactive routing protocol between the clusters. In this way, each cluster has $O(1)$ routes to maintain toward other ones. When applying such a routing policy, a node u also needs to locate its correspondent v in order to pro-actively route toward the cluster owning v. In this paper, we propose a localization scheme, based on Distributed Hashed Tables and Interval Routing which takes advantage of the underlying clustering structure. It only requires $O(1)$ memory space size on each node.

Key-words: localization, DHT, ad hoc, sensors, wireless, self-organization, clusters, scalability, interval routing

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

Localisation distribuée des noeuds dans les réseaux sans fils auto-organisés.

Résumé : Les protocoles de routage proposés pour les réseaux *Ad hoc* par le groupe de travail MANET ne sont que des protocoles "à plat", sans hiérarchie. Ils ne sont par conséquent pas adaptés à des environnements plus larges ou plus denses du fait des ressources mémoire et radio (bande passante) qu'ils sollicitent. Une solution pour permettre le passage à l'échelle de tels réseeaux est d'introduire un routage hiérarchique en groupant les stations sans fils en *clusters*. Cela permet ensuite d'appliquer différentes politiques de routage à plus petite échelle entre clusters et dans un cluster. Nous avons précédement introduit un algorithme de *clustering* [19] qui présente d'intéressantes propriétés et qui s'est révélé plus efficaces sur différents aspects que les protocoles existants. Dans la plupart des cas rencontrés dans la littérature, un routage hiérarchique implique un routage proactif dans les clusters et un routage réactif entre les clusters. Cependant, du fait des caractéristiques de notre algorithme de clustering, nous proposons ici l'approche inverse, *c.a.d* un routage réactif dans les clusters et proactif entre clusters. Nous avons un nombre de clusters faible et constant quel que soit le nombre de nœuds dans le réseau donc, de cette façon, chaque nœud n'aura que $O(1)$ routes à maintenir vers les autres clusters. En partant de là, un nœud a maintenant besoin de savoir à quel cluster appartient son correspondant. Pour cela, nous proposons de tirer avantage de la structure sous-jascente en arbres de nos clusters pour appliquer un algorithme de localisation basé sur des tables de hâchage distribuées (DHT) et un routage par intervalle qui ne nécessite que $O(1)$ taille mémoire sur chaque nœud.

Mots-clés : localisation, DHT, ad hoc, senseurs, auto-organisation, clusters, passage à l'échelle, routage par intervalle

1 Introduction

Wireless multi-hop networks such *ad hoc* or sensor networks are composed of terminals that communicate via wireless interfaces. They require no fixed infrastructure and no human intervention. Even if every mobile can move everywhere, and thus can disappear or appear in the network at any time, the network manages the changes in topology and provides the connectivity between any pair of terminals. If actual wireless cards allow to establish a radio communication between mobiles that are in communication range, a routing protocol is required to provide the full connectivity in the network. As there are no dedicated devices in the network, all the terminals are potential routers. Multi-hop wireless networks have been becoming very popular due to their easiness of use. Their applications range from the network extension when cabling is not possible or too expensive, to spontaneous networks, in case of natural disaster where the infrastructure has been totally destroyed, by going through the monitoring and the collect of data with wireless sensor networks.

Due to the dynamics of wireless networks (terminal mobility and/or instability of the wireless medium), the routing protocols for fixed networks do not efficiently fit. *Ad hoc* routing protocols proposed in the MANET working group at IETF[1] are all flat routing protocols: that means that there is no hierarchy and all the terminals have the same role. If flat protocols are quite effective on small and medium size networks, they are not suitable for large scale or very dense networks because of bandwidth and processing overheads they generate [33]. A common solution to this scalability problem is to introduce a hierarchical routing. Hierarchical routing often relies on a specific partition of the network, called *clustering*: the terminals are gathered into clusters according to some criteria, each cluster is identified by a special node called *cluster-head*, and specific routing protocols are used within the clusters and between the clusters. In addition to its scalable feature, such an organization also presents numerous advantages as to synchronize mobiles in a cluster or to attribute new service zones. In most of the literature, a hierarchical routing means using a proactive[2] routing protocol inside the clusters and a reactive[3] one between the clusters [10, 14, 25, 26]. In this way, nodes store full information concerning nodes in their cluster and only partial information about other nodes.

We previously introduced a clustering algorithm [19]. Its builds clusters by locally constructing trees. Every node of a same tree belong to the same cluster, the tree root is the cluster-head. This algorithm has already been well studied by simulation and theoretical analysis. It has shown to outperform some other existing clustering schemes regarding structure and behavior over node mobility and link failure. It may also be used to perform efficient broadcasting operations [21]. Nevertheless, it provides a constant number of clusters when the node intensity increases. Thus, there still are $O(n)$ nodes per cluster and using a proactive routing scheme in each cluster as in a classical hierarchical routing, would imply that each node still stores $O(n)$ routes, which is not more scalable than flat routing. Therefore, we propose to use the reverse approach, *i.e.*, applying a reactive routing protocol inside the clusters and a proactive routing protocol between the clusters. Indeed, as the number of clusters is constant, each cluster has only $O(1)$ routes to maintain toward other ones. As far as we know, only the SAFARI project [31] has proposed such an approach, even if most of the clustering

[1] http://www.ietf.org/html.charters/manet-charter.html
[2] Nodes permanently keep a view of the topology. All routes are available as soon as needed.
[3] Routes are searched on-demand. Only active routes are maintained.

schemes [2, 25] present an increasing number of nodes per cluster with an increasing node intensity and still claim to apply a proactive routing scheme inside the clusters. When applying such a routing policy, a node u first needs to locate the node v with which it wants to communicate, *i.e.*, it needs to know in which cluster node v is. Once it gets this information, u is able either to pro-actively route toward this cluster if it is not the same than its, or to request a route toward node v inside its cluster otherwise. So, a localization function which returns for any node u, the name of the cluster to which it belongs is needed. In this paper, we introduce our localization scheme which takes advantage of the tree/cluster structure. It is based on Distributed Hashed Tables (DHT) and Interval Routing. DHT are known to be scalable and allow to each node u to register its cluster identity over the network on several rendezvous points which will be contacted by every node looking for node u. Interval routing is already known to be a highly memory efficient routing for communication in distributed systems. In addition, this scheme also takes advantage of the broadcasting feature of the wireless communications to reduce even more the memory size. Yet, each node only requires a memory size in $O(1)$.

The remaining of this report is organized as follows. Section 2 briefly describes the system model and the notations we use henceforth. The description of the initial clustering algorithm as well as interesting features for the localization algorithm are given in Section 3. Then, we describe in Section 4 how we propose to take advantage of the underlying structure of our organization to perform our localization algorithm. Then, we describe in Section 5 our proposition. Lastly, in Section 6, we discuss some improvements and future works.

2 Model and Notations

We classically model a wireless multi-hop network by a random geometric graph $G = (V, E)$ where V is the set of mobile nodes ($|V| = n$) and $e = (u, v) \in E$ represents a bidirectional wireless link between a pair of nodes u and v. We use the disk graph model, *i.e.*, if $dist(u, v)$ is the Euclidean distance between nodes u and v, then u and v share a bidirectional link ($\exists (u, v) \in E$) if and only if $dist(u, v) \leq R$, R being the transmission range. If $d(u, v)$ is the distance in the graph between nodes u and v (minimum number of hops needed to reach v from u), we note $\Gamma_k(u)$ the set of nodes v such that $d(u, v) = k$. Note that node u does not belong to $\Gamma_k(u) \forall k$. $\delta(u) = |\Gamma_1(u)|$ is called the *degree* of u. We note $\mathcal{C}(u)$ the cluster owning u and $\mathcal{H}(u)$ the cluster-head of this cluster. Let $\mathcal{P}(u)$ denote the parent node of node u in a tree and $\mathcal{C}h(u)$ the set of children of u, *i.e.*, the set of nodes v such that $\mathcal{P}(v) = u$. Note that u is a leaf iff $\mathcal{C}h(u) = \emptyset$. Moreover, we note $s\mathcal{T}(u)$ the subtree rooted in node u. We say that $v \in s\mathcal{T}(u)$ if $v = u$ or if u is the parent of node v ($\mathcal{P}(v) = u$) or if the parent of node v is in the subtree rooted in u ($\mathcal{P}(v) \in s\mathcal{T}(u)$).

$$\{v \in s\mathcal{T}(u) \cap \Gamma_1(u)\} \Leftrightarrow \{v \in \mathcal{C}h(u)\} \text{ or } \left\{v \in s\mathcal{T}(u) \cap \bar{\Gamma_1}(u) \setminus \{u\}\right\} \Leftrightarrow \{\mathcal{P}(v) \in s\mathcal{T}(u)\}$$

3 Our cluster organization

In this section, we summarize our previous clustering work on which we apply our localization scheme. Only basis and features which lead our motivations or which are relevant for localization and routing are evoked here. For more details or other characteristics of our clustering heuristic, please refer to [19, 20, 22].

3.1 The clustering heuristic

3.1.1 The link density metric

Our initial goal was to propose a way to use multi-hop wireless networks over large scale. As flat routing and managing protocols are not suitable for it because of the process and message overheads they induce [33], a well known solution is to introduce a hierarchy. So, we proposed a clustering algorithm motivated by the fact that in a multi-hop wireless environment, the less information exchanged or/and stored, the better. First, we wanted a cluster organization with no-overlapping clusters with a flexible radius (Many clustering schemes [9, 16] have a radius of 1, in [2, 12] the radius is set a priori.) and able to adapt to the different topologies. Second, we wanted the nodes to be able to compute the heuristic from local information, only using their 2-neighborhood knowledge. (In [2], if the cluster radius is set to d, the nodes need to gather information up to d hops away before taking any decision.) Finally, we desired an organization robust and stable over node mobility, *i.e.*, which do not need to be recomputed at each single change in the topology. For it, we introduced a new metric called *density* [19]. The notion of density characterizes the "relative" importance of a node in the network and within its neighborhood. This link density (noted $\rho(u)$) smooths local changes down in $\Gamma_1(u)$ by considering the ratio between the number of links and the number of nodes in $\Gamma_1(u)$.

Definition 1 (density) *The density of a node $u \in V$ is:*

$$\rho(u) = \frac{|\{e = (v, w) \in E \mid w \in \{u\} \cup \Gamma_1(u) \; and \; v \in \Gamma_1(u)\}|}{\delta(u)}$$

3.1.2 The cluster formation

On a regular basis, each node locally computes its density value and regularly locally broadcasts it to its 1-neighbors (*e.g.*, using `Hello` packets). Each node is thus able to compare its density value to its 1-neighbors' and decides by itself whether it joins one of them (the one with the highest density value) or it wins and elects itself as cluster-head. In case of ties, the node with the lowest Id wins. In this way, two neighbors can not be both cluster-heads. As every node joins exactly one of its 1-neighbors, we actually draw a tree $T' = (V, E')$ which is a subgraph of G, such that $E' \subset E$. T' is actually a directed acyclic graph (DAG). A DAG is a directed graph that contains no cycles, *i.e.*, a directed tree. The node which density value is the highest within its neighborhood becomes the root of the tree and thus the cluster-head of the cluster. If node u has joined node w, we can say that w is node u's parent (noted $\mathcal{P}(u) = w$)in the clustering tree and that node u is a child of node w

(noted $u \in \mathcal{C}h(w)$). A node's parent can also have joined another node and so on. A cluster then extends itself until it reaches another cluster. If none of the nodes has joined a node u ($\mathcal{C}h(u) = \emptyset$), u becomes a leaf. All the nodes belonging to a same tree belong to the same cluster. We then build the clusters by building a spanning forest of the network in a distributed and local way as the decision criterion is only based on information regarding the 2-neighborhood of a node.

For a node $v \in V$, for all node $u \in \Gamma_1(v)$, let define $Age(u)$ as the number of successive rounds that node u is elected by node v as its parent. We define $\prec$ as a binary total order such that, if $u, v \in V^2$, $u \prec v$ if and only if $\{\rho(u) < \rho(v)\}$ or $\{\rho(u) = \rho(v) \wedge Age(u) < Age(v)\}$ or $\{\rho(u) = \rho(v) \wedge Age(u) = Age(v) \wedge$ *Id(v)* $<$ *Id(u)*$\}$.

This algorithm stabilizes when every node knows its *correct* cluster-head value.

Algorithm 1 cluster-head selection

For all node $u \in V$

▷ *Variable initialization, only when node* u *appears.*

$\mathcal{H}(u) = \mathcal{P}(u) = -1$

$\forall v \in \Gamma_1(u), Age(v) = 0$

while ($\mathcal{H}(u) = -1$) or ($\mathcal{H}(u) \neq \mathcal{H}_{old}(u)$))

▷ *Runs until stabilization*

$\mathcal{H}_{old}(u) = \mathcal{H}(u)$

▷ *Checking the neighborhood*

Gather $\Gamma_2(u)$

Compute $\rho(u)$

Locally broadcast $\rho(u)$

▷ *This local broadcasting can be done by piggybacking* $\rho(u)$ *in* HELLO *packets.*

▷ *At this point, node* u *is aware of all its 1-neighbors' density value and can elect its parent.*

if ($\forall v \in \Gamma_1(u),\ v \prec u$) **then** $\mathcal{H}(u) = u$ ▷ u *is promoted cluster-head.*

else

▷ $\exists w \in \Gamma_1(u)$ *s.t.* $\forall v \in \{u\} \cup \Gamma_1(u),\ \ v \prec w$

$\mathcal{P}(u) = w$

$\mathcal{H}(u) = \mathcal{H}(w)$

▷ *Either* $\mathcal{P}(w) = \mathcal{H}(w) = w$ *and* u *is directly linked to its cluster-head, or* w *has joined another node* x *(*$\exists\, x \in \Gamma_1(w) \mid \mathcal{P}(w) = x$*) and recursively* $\mathcal{H}(u) = \mathcal{H}(w) = \mathcal{H}(x)$.

end

if ($\mathcal{H}(u) = u$ and $\exists v \in \Gamma_1(u) \mid \mathcal{P}(v) \neq u$) **then**

▷ *Node* u *is a cluster-head, however all its* 1*-neighbors have not joined it.*

if ($\mathcal{P}(v) = \mathcal{H}(v)$) **then**

▷ *At least two cluster-heads (*$\mathcal{H}(u)$ *and* $\mathcal{H}(v)$*) have a common neighbor* v*. As* $\mathcal{P}(v) = \mathcal{H}(v)$*, we have* $u \prec \mathcal{H}(v)$*.* u *is not a cluster-head anymore and elects* v *as its parent (and clusters merge).*

$\mathcal{P}(u) = v$ and $\mathcal{H}(u) = \mathcal{H}(v)$

Age(v)++ and $\forall w \in \Gamma_1(u)\ Age(w) = 0$.

end

end

if ($\exists v \in \Gamma_1(u)$ s.t. $\{\mathcal{H}(v) = v$ and $\mathcal{H}(\mathcal{P}(u)) \neq \mathcal{H}((u)\}$)

▷ u *is at at least* 2 *hops away from its cluster-head (its parent is not a cluster-head) and it has another cluster-head among its neighbors. It will join this one instead.*

$\mathcal{P}(u) = v$ and $\mathcal{H}(u) = v$

$Age(v)$++ and $\forall w \in \Gamma_1(u)\, Age(w) = 0$.
end
Locally broadcast $\mathcal{P}(u)$ and $\mathcal{H}(u)$
end

As proved in [22], at the end of three message exchange rounds, each node is aware of its parent in the tree, at the end of four message rounds, it knows the parent of each of its neighbors and thus is able to determine whether one of them has elected it as parent and thus learns its condition in the tree (root, leaf, regular node). A node is a leaf if no other node has chosen it as its parent; a node is a cluster-head if it has chosen itself as parent and all its 1-neighbors have joined it; a node is a regular node otherwise. It has also been proved that in an expected constant and bounded time, every node is also aware of its cluster-head identity and of the cluster-head identity of its neighbors. It thus knows whether it is a frontier node. A node is a frontier node if at least one of its neighbors does not belong to the same cluster than itself. Table 1 summarizes what a node is able to learn and compute after each message exchange round.

	What node u learns from its neighbors	What it can then compute
Round 1	$\Gamma_1(u)$	
Round 2	$\Gamma_1(u)$ + $\Gamma_2(u)$	$\rho(u)$
Round 3	$\Gamma_1(u)$ + $\Gamma_2(u)$ + $\rho(v) \forall v \in \Gamma_1(u)$	$\rho(u)$ + $\mathcal{P}(u)$
Round 4	$\Gamma_1(u)$ + $\Gamma_2(u)$ + $\rho(v) \forall v \in \Gamma_1(u)$ + its condition	$\rho(u)$ + $\mathcal{P}(u)$

Table 1: Sum up of exchanged information

3.1.3 Example

To illustrate this heuristic, let's run Algorithm 1 over the graph plotted on Figure 1. In its 1-neighborhood topology, node a has two 1-neighbors ($\Gamma_1(a) = \{d, i\}$) and two links ($\{(a, d), (a, i)\}$), thus $\rho(a) = 1$; node b has 4 1-neighbors ($\Gamma_1(b) = \{c, d, h, i\}$) and five links ($\{(b, c), (b, d), (b, h), (b, i), (h, i)\}$), thus $\rho(b) = \frac{5}{4}$. Table 2 shows the final results of density values computed for all nodes.

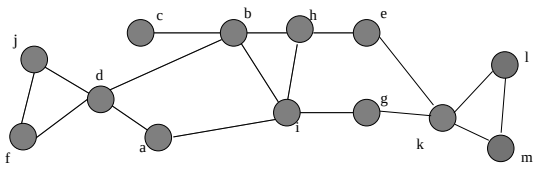


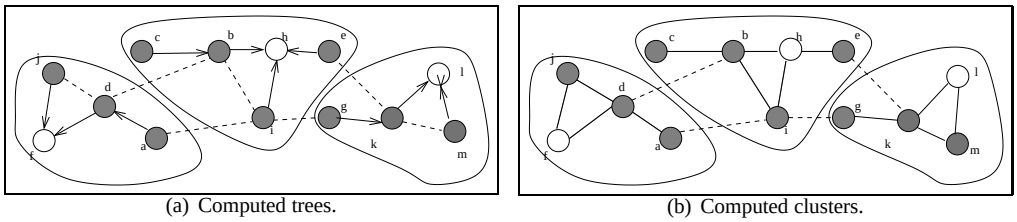
Figure 1: Example.

In our example, node c joins its neighbor node b ($\mathcal{P}(c) = b$) which density is the highest in $\Gamma_1(c)$ ($\forall v \in \Gamma_1(c) \cup \{c\}\, v \prec b$). The node with the highest density in node b's neighborhood is h. Thus, $\mathcal{P}(b) = h$. As node h has the highest density in its own neighborhood, it becomes its own

Nodes	a	b	c	d	e	f	g	h	i	j	k	l	m
Degree	2	4	1	4	2	2	2	3	4	2	4	2	2
# Links	2	5	1	5	2	3	2	4	5	3	5	3	3
Density	1	1.25	1	1.25	1	1.5	1	1.33	1.25	1.5	1.25	1.5	1.5

Table 2: Results of our heuristic run over the graph potted in Figure 1.

cluster-head: $\mathcal{H}(h) = h$ and thus: $\mathcal{H}(c) = \mathcal{H}(b) = \mathcal{H}(h) = h$. To sum up, node c joins b which joins h and all three of them belong to the same cluster/tree which cluster-head/root is h. Note that $\rho(j) = \rho(f)$. We thus use the Id to decide between both nodes. Let's assume that node f has the lowest Id ($j \prec f$): $\mathcal{P}(j) = f$ and $\mathcal{P}(f) = f$ so $\mathcal{H}(f) = \mathcal{H}(j) = f$. No node has chosen nodes a, j, c, e, i, g and m as parent: they become leaves. Finally, we obtain a spanning forest of the network, composed of three trees rooted at h, l and f (See Figure 2(a)), which give birth to three clusters organized around three cluster-heads: h, l and f (See Figure 2(b)).



(a) Computed trees. (b) Computed clusters.

Figure 2: Example of trees (a) and clusters (b) computed with the density-based algorithm over the graph potted in Figure 1 (cluster-heads/roots appear in white).

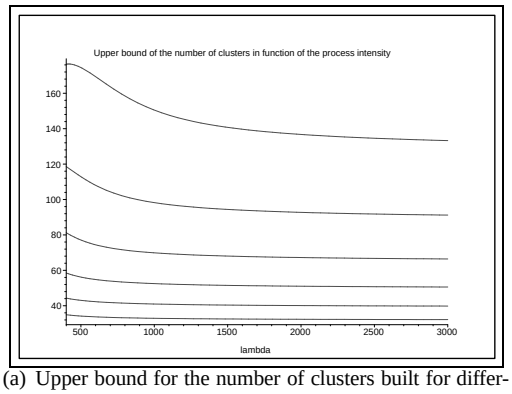
Every node needs 3 message exchange rounds to know its parent and its condition within the tree (leaf, root or internal node) (See Table 1). Moreover, in this example, nodes need at most 2 more exchange rounds to know their cluster-head identity *i.e*, the maximum tree depth.

3.2 Some characteristics of our clustering algorithm

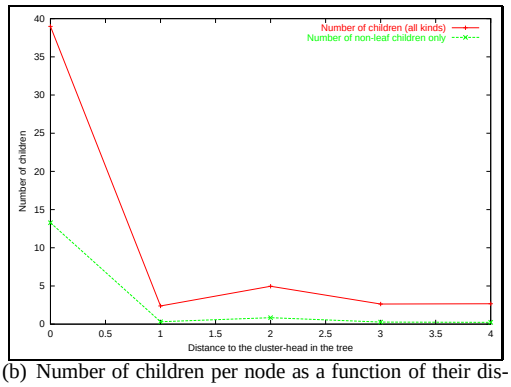
Algorithm 1 stabilizes when every node knows its *correct* cluster-head value. In [22], it has been proved by theory and simulation to self-stabilize within a low, constant and bounded time. It also has been proved that a cluster-head is aware of an information sent by a frontier node in a low, constant and bounded time since the tree depth is bounded.

The number of clusters built by this heuristic has been studied analytically and by simulation. The upper bound of the number of clusters drawn on Figure 3(a) shows that the number of clusters tend toward a constant asymptote when the number of nodes in the network increases. Moreover, the clusters built are homogeneous. Compared to other clustering schemes as DDR [25] or Max-Min d cluster [2], our cluster organization has revealed to be more stable over node mobility and arrivals and to offer a better behavior over non-uniform topologies (see [19]). Moreover, our algorithm

presents a lesser complexity in time and messages as it only needs information regarding the 2-hop neighborhood of a node while Max-Min needs information to d hops away and DDR nodes need to store information about all the nodes belonging to the same cluster than themselves.



(a) Upper bound for the number of clusters built for different values of R (from the bottom to the top $R = 0.1, 0.09, 0.08, 0.07, 0.06, 0.05$ m).



(b) Number of children per node as a function of their distance to the cluster-head in the clustering tree.

Figure 3: Some cluster and tree characteristics

Other interesting features for routing and locating obtained by simulations are gathered in Table 3. These characteristics illustrate some of our motivations for our proposition as explained later in Section 4.

	500 nodes	600 nodes	700 nodes	800 nodes	900 nodes	1000nodes
# clusters/trees	11.76	11.51	11.45	11.32	11.02	10.80
Cluster diameter	4.99	5.52	5.5	5.65	6.34	6.1
Cluster-head eccentricity	3.01	3.09	3.37	3.17	3.19	3.23
Node eccentricity	3.70	3.75	3.84	3.84	3.84	3.84
Tree depth	3.27	3.34	3.33	3.34	3.43	3.51
Degree in the tree of non-leaves	3.82	3.99	4.19	4.36	4.51	4.62
% leaves	73,48%	74,96%	76,14%	76,81%	77,71%	78,23%

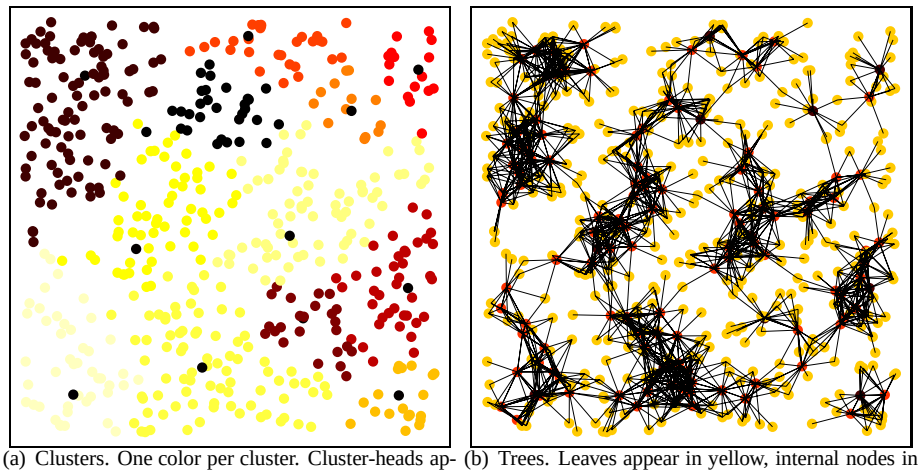
Table 3: Some cluster and clustering trees characteristics.

The eccentricity of a node is the greater distance in number of hops between it and any other node in its cluster. We can see in Table 3 that the tree depth is low and close to the optimal (cluster-head eccentricity). That means that the routes in the trees from the cluster-head to any other node within its cluster are close to the shortest paths in the network.

Clustering trees present some interesting properties and a great proportion of leaves and a small amount of children per non-leaf node (See Figure 3(b)). These features and the *"Degree in the tree*

of non-leaf nodes" entry in Table 3 show that, in average, an internal node does not have a lot of children.

Figure 4 plots an instance of clusters and trees obtained over a 500-node network. Cluster-head appear in black. As we can note, leaves are numerous and pretty well distributed over the space. The same kid of shape is observed for other node intensities.



(a) Clusters. One color per cluster. Cluster-heads appear in black.

(b) Trees. Leaves appear in yellow, internal nodes in red.

Figure 4: Example of clusters (a) and trees (b) over a 500-node network.

4 Basic ideas

In fixed networks, routing information is embedded into the topological-dependent node address. For instance, an IP address both identifies a node and locates it, since the network prefix is included in the IP node address. In wireless networks, nodes may arbitrarily move, appear or disappear at any time. So, the permanent node identifier can not include any dynamic location information and thus, it has to be independent from the topology, which implies an indirect routing between nodes. A routing operation is referred as indirect when it is performed in two steps: *(i)* first **locate** the target and then *(ii)* directly **communicate** with the target. This allows the network to dissociate the location of a node from the location itself. With this approach, the routing information can be totally distributed, which is important for achieving scalability in large scale networks. Figure 5 illustrates such a routing process. Such routing is used for instance in the cell phone GSM technology[4] or by Mobile IP[5], where the position of the destination is primarily asked respectively to a HLR (Home

[4] http://www.gsm.org

[5] http://www.ietf.org/rfc/rfc2002.txt

Location Register) or a Home Agent before the communication being established directly between the sender and the final destination. Nevertheless, as in multi-hop wireless networks, every node may move, even the home agent, this can not be directly applied.

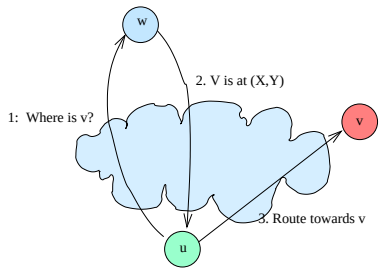


Figure 5: Indirect Routing: Node u needs to communicate with node v but does not have any clue about its location. It thus first needs to ask a third entity (here node w) where v is. Node w knows where node v is since v regularly registers its position on w and can answer node u. u is then able to communicate with v.

We propose to use such an indirect routing scheme for routing in large scale multi-hop wireless networks. We are motivated by the fact that we want a very scalable solution, thus our proposition aims to store as less information on nodes as possible. We also want to avoid situations where the distance between the source/requester (node u in Figure 5) and the rendezvous node (node w) is much greater than the distance between the source (node u) and the destination/searched node (node v). Indeed, if the request has to cross twice the whole network before two nodes be able to directly communicate, we waste much bandwidth and latency.

Distributed Hash Tables (DHT) are a basis for indirect routing. Implementation of DHT in wireless networks raise new issues than from wired networks [28]. But, as shown in [11], such a dynamic addressing is a promising approach for achieving scalability in large scale wireless multi-hop networks. DHT provide a general mapping between any information and a location. They use a virtual addressing space $\mathcal{V}$. Partitions of this virtual space are assigned to nodes in the network. The idea is to use a $hash$ function to first distribute node location information among rendezvous points. This same $hash$ function is known by every node and may then be used by a source to identify the rendezvous point which stores the position of the target node. Each information is *hashed* into a key ($hash(v) = key_v \in \mathcal{V}$) of the virtual addressing space $\mathcal{V}$ and is then stored on the node(s) responsible for the partition of the virtual space this key belongs to. Then, in Figure 5, we have $hash(v) = Position\_node\_v \in I \subset \mathcal{V}$ and node w responsible for interval I. By knowing I, nodes v and u are able to find node w either to register their own location (for node v) or to ask for the location of v (for node u). Note that nodes u and v do not need to know the real identity of node w for it, they only need its virtual address in $\mathcal{V}$. This operation in DHT-based systems which returns the node(s) responsible for a wanted *key* is called the *lookup* operation. For more details about the lookup operation, please refer to [4].

In the literature, DHT have been applied at two different levels: at the application level and at the network level. Applying DHT at the application layer is widespread in peer-to-peer networks. The information hashed in such file-sharing systems is the identity of a file. The node responsible for the $key = hash(file)$ stores the identifier of the nodes which detain that file. DHT nodes form an overlay network on which the lookup queries are routed. The main difference among the many proposals is the geometry of this overlay which ranges from a ring (Chord [35]) to a de Bruijn graph (D2B [13]) through trees (Tapestry [41], Kademlia [18]), d-dimensional spaces (CAN [30]), butterfly structures [17] or some hybrid trees-rings (Pastry [32]).

At the network layer, DHT are applied to distribute node location information throughout the topology and are used to identify a node which is responsible for storing a required node location. This is the way we intend to use DHT. When a node u needs to send an information to a node v, it first has to know where v is. To get this information, it first asks a node w in charge of the key $k = hash(v)$ and thus knowing where v is.

Two schemes can be found: DHT-dependent and DHT independent routing. In DHT-independent routing schemes, *i.e.*, the virtual address is not used for the routing operation. The nodes generally know their geographic coordinates, either absolute (by using a GPS for example) or relative (as in [7]), which is the location information they associate to the key. By performing $hash$(target), a node u gets the geographical coordinates of a rendezvous area $\mathcal{A}$. u then applies a geographic routing protocol to join a node v laying in $\mathcal{A}$ and which is aware of the geographical coordinates of the target node. From it, node u is able to reach the destination by performing a geographical routing again. This is the case for instance in [3, 23, 24], in the Terminodes project [5, 6] or in the Grid project [15]. As we do not want our nodes to depend on a positioning system, we can not apply that DHT utilization. In DHT-dependent routing schemes, the virtual space of the DHT is used not only for locating but at the same time for routing toward the destination. The virtual address is dependent of the location. In this way, the coherency of the routing protocols relies on the coherent sharing of the virtual addressing space among all nodes in the network. The routing is performed over the virtual structure. In such scenarii, a node u performing $hash(w)$ gets the virtual address of the rendezvous point. From it, u routes in the virtual space to v which gives it the virtual address of w. u thus is able to reach w by routing in the virtual space again. The routing scheme used is generally a greedy routing: "Forward to the neighbor in the virtual space whose virtual address is the closest to the virtual address of the destination". This is for instance the case of Tribe [38, 39] or L+ [8] on which is based SAFARI [31]. The main challenge here is to disseminate the partitions of the virtual space in such a manner that the paths in the virtual space are not much longer than the physical routes.

Thus, in DHT-based systems, we can consider two phases of routing: *(i)* a routing toward the rendezvous node which stores the needed information (Arrow 1 on Figure 5) and *(ii)* a routing toward the final destination node which location had been obtained by the lookup operation (Arrow 3 on Figure 5). In all proposals cited above, both routing phases are performed in the same way, either in the physical network (for DHT-independent routing proposals), or in the virtual one (for DHT-dependent routing proposals). In the approach we propose, the two routing steps are completed in two different manners. The first routing step is performed by using the virtual address of the

rendezvous point (DHT-dependent) whereas the routing toward the final destination is performed over the physical network (DHT-independent).

In our proposal, every node already gets an information about its relative location: its cluster identity. As we suppose that the nodes do not have any geographical information neither absolute nor relative, this is the only information they have and can use. Thus, nodes register their cluster Id as location information. As seen in Section 3, we have a tree structure. We propose to partition the virtual space $\mathcal{V}$ over each tree and that each node registers in each cluster, in order to add redundancy. In this way, when a node v looks for a node u, it just has to search the information in its cluster. As the node eccentricity is low (Section 3), the latency is reduced. Moreover, partitioning the virtual space in each cluster rather than once in the whole network avoids situations where the distance between the source and the rendezvous point is much greater than the distance between the source and the destination, as in this way, the source and the rendezvous point always are in the same cluster whereas the destination may be anywhere in the network.

To distribute the partitions of the virtual space of the DHT in such a way that, given a virtual address, a node u is able to find the node responsible for without any additional information, we use a tree Interval Labeling Scheme to then allow an Interval Routing in the virtual space.

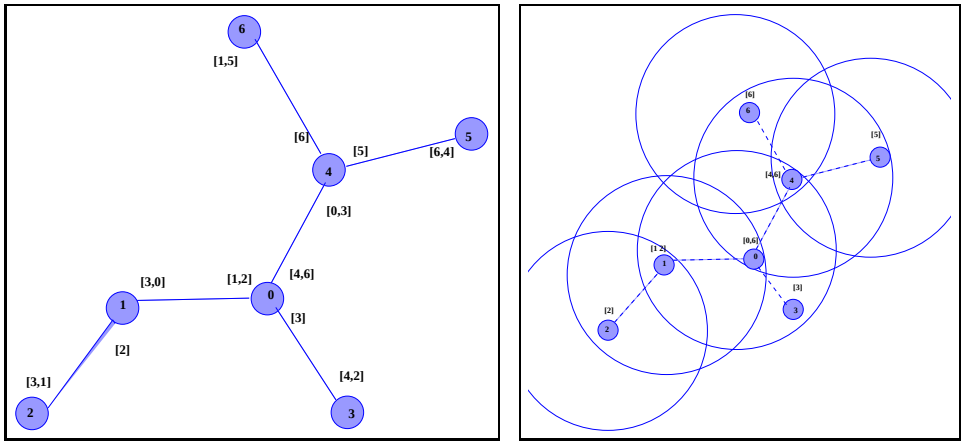
Interval Routing is really attractive by its simplicity. It was introduced in wired networks by Santoro and Khatib in [34] to reduce the size of the routing tables. It is based on representing the routing table stored at each node in a compact manner, by grouping the set of destination addresses that use the same output port into intervals of consecutive addresses. The main advantage of this scheme is the low memory requirements to store the routing on each node u: $O(\delta(u))$. The routing is computed in a distributed way with the following algorithm: at each intermediate node x, the routing process ends if the destination y corresponds to x, otherwise, it is forwarded with the message through an edge labeled by a set I such that $y \in I$.

The Interval Labeling Scheme (ILS) is the manner to assign the intervals to the edges of each node, in order to perform an efficient interval routing with routes as short as possible. Yet, the authors of [37] have shown that undirected trees can support an Interval Routing Scheme with shortest paths (in the tree) and with only one interval per output port when performing a Depth-First-Search ILS.

In wired networks, nodes have to store an interval for each of its output edge. Therefore the size of the routing table is in $O(\delta(u))$. But in wireless environments, a transmission by each node can reach all nodes within radius distance from it. Edges actually are hyper-edges (see Figure 6). Thus the problematic is a bit different as querying unicast transmission actually are broadcast transmission. Indeed, as from a node u, there is only one hyper-edge, nodes can store only one interval for it and thus for all their neighbors. In our proposal, nodes only store the interval for which their subtree is responsible (and the intervals of each of their neighbors). This gives a table routing size in $O(1)$. When a query is sent, as all neighbors receive it in any way, only the one(s) concerned by it answer(s).

4.1 Summary and complexity analysis

To sum up, we propose to apply an indirect routing scheme over our clustered network by using a DHT which associates each node identifier to a virtual address of a logical space $\mathcal{V}$. The set of virtual



(a) Routing in wired environments. Nodes store one interval per output edge.

(b) Routing in wireless environments. Nodes store only one interval (one per hyper-edge).

Figure 6: Edges in wired networks (a) Vs hyper-edges in wireless networks (b).

addresses $\mathcal{V}$ is partitioned over the nodes of each cluster. As the number of clusters is constant and clusters are homogeneous, each node finally stores $O(1)$ location information.

When a node u wants to communicate with a node v, it first uses the DHT to find out the virtual address of v: $hash(v) = key_v \in \mathcal{V}$. Then, by using an Interval Routing over the virtual space $\mathcal{V}$ of its own tree, it reaches at least one node in its cluster responsible for storing the location of v, *i.e.*, $\mathcal{C}(v)$. As the intervals of the neighbors are not stored on the node, this one only stores its own interval and the size of its routing table is in $O(1)$.

As the number of clusters is constant when the intensity of nodes increases, each cluster has $O(1)$ routes to maintain toward other ones for the proactive routing phase.

5 Our proposition

5.1 Preliminaries

Each node u has two identifiers:

- An universal address $Id(u)$ (also note u in the following). This address is unique in the network and never changes. It is the "real" name of node u.
- A logical address $i(u) \in \mathcal{V}$. This address is unique in a cluster. It may change at each distribution of the partitions of $\mathcal{V}$ over the nodes of a cluster. It identifies node u in the logical space.

Let $I(u)$ be the partition of $\mathcal{V}$ node u is assigned.
$I(u)$ is such that $I(u) = [i(u), ...[$ such that $i(u) \neq hash(u)$ where $i(u)$ is the logical address of node u in $\mathcal{V}$. $i(u)$ thus depends on $I(u)$. We note $I_{tree}(s\mathcal{T}(u)) = \bigcup_{v \in s\mathcal{T}(u)} I(v)$ the interval/partition of $\mathcal{V}$ for which the subtree of node u is responsible. $|I|$ is used to refer to the size of interval I.

5.2 Virtual space partitioning

As mentioned in Section 4, an optimal Interval Labeling Scheme in a tree is obtained via a Depth-First Search (DFS) Labeling Scheme. Every node is assigned a partition of a logical addressing space $\mathcal{V}$. In this section, we present the way we distribute the partitions of $\mathcal{V}$ based on such an ILS.

As in a traditional DFS Interval Labeling Scheme, the partitions of $\mathcal{V}$ are distributed in such a way that, for every node $u \in V$:

- The intervals of the nodes in $s\mathcal{T}(u)$ form a contiguous interval.
- The size of the interval a subtree is in charge of is proportional to its size: $|I_{tree}(s\mathcal{T}(u))| \propto |s\mathcal{T}(u)|$. We thus have, for every node $v \in s\mathcal{T}(u)$, $|I_{tree}(s\mathcal{T}(u))| \geq |I_{tree}(s\mathcal{T}(v))|$.
- $\mathcal{V}$ is completely shared among the nodes of the cluster: $\mathcal{V} = \bigcup_{v \in \mathcal{C}(u)} I(v)$.
- The control regions are mutual exclusive: $\forall v \in \mathcal{C}(u), \forall w \in \mathcal{C}(u), v \neq w \ I(v) \cap I(w) = \emptyset$

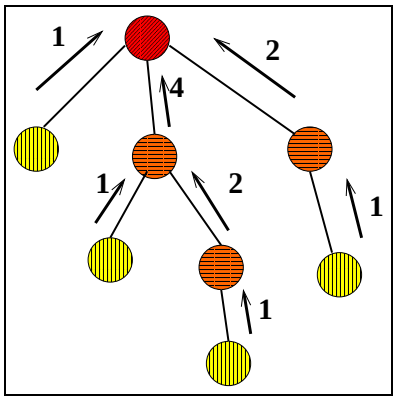
We propose a parallel interval distribution over the different branches of each tree. This distribution can be qualified of quasi-local according to the taxonomy established in [40] as each node u needs information up to $d_{tree}(u, \mathcal{H}(u))$ only, where $d_{tree}(u, \mathcal{H}(u))$ is the number of hops in the tree between u and its cluster-head $\mathcal{H}(u)$ in the tree. Yet, the tree depth is bounded by a low constant (see Section 3). Our algorithm runs in two steps: a step up the tree (from the nodes to the cluster-head) and a step down the tree (from the cluster-head to the nodes). Figure 7 plots an example using $\mathcal{V} = [1, 17[$ to illustrate this partitioning. The distribution is performed in parallel over each branch of each tree, so each step has a time complexity of $O(Tree\_depth)$. So, the global complexity in time of the distribution algorithm for a cluster/tree is $2 \times (Tree\_depth)$. As the tree depth is bounded by a constant, the complexity in time is $O(1)$.

Step 1**.** As seen in Section 3, every node u might be aware in an expected bounded and low time, of the parent of each of its neighbors. It then is able to determine whether one of them has elected it as parent and thus learns its condition in the tree (root, leaf, regular node). Thus, in a low and constant time, each internal node in the tree is aware of the number of its children. If every node sends its parent the size of its subtree ($|s\mathcal{T}(u)| = |u \bigcup_{v \in \mathcal{C}h(u)} s\mathcal{T}(v)| = 1 + \sum_{v \in \mathcal{C}h(u)} |s\mathcal{T}(v)|$) up to the cluster-head, each node is expected to know the size of the subtree of each of its children in a low time and so the cluster-head.

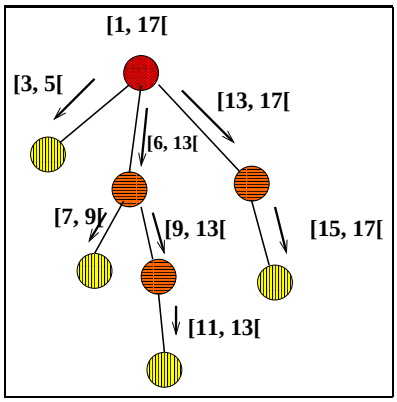
For instance, on Figure 7(a), every node sends the size of its subtree. The leaves (yellow vertical nodes) are the only node of their subtree, they sends 1. Internal nodes (orange horizontal nodes) collect informations from all their children and sum the size of the subtree of each of them. And so on till reaching the root/cluster-head (red oblique node).

Step 2**.** Once the cluster-head is aware of the size of the subtree of each of its neighbors/children, it shares $\mathcal{V}$ between itself and its children. Each node v is assigned a partition of $\mathcal{V}$: $I_{tree}(s\mathcal{T}(v))$ proportional to the size of its subtree. Each internal node then re-distributes the partition its parent assigned it, between itself and its own children, and so on, till reaching the terminal leaves.

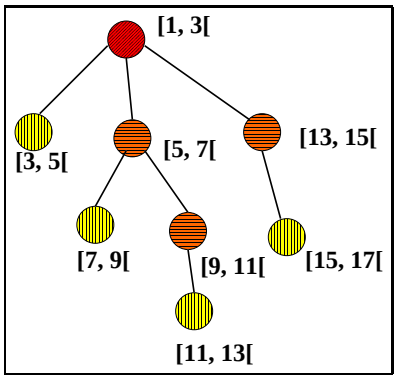
For instance, on Figure 7(b), every internal node share the interval it has been aside between itself and its children, starting from the cluster-head to the leaves. Intervals in arrows are the ones a node is given by its parent for its subtree.



(a) Step 1: Every node sends its parent the size of its subtree, starting from the leaves.



(b) Step 2: Every internal node shares the interval given by its parent between itself and its children proportionally to the size of each subtree, starting from the cluster-head.



(c) Result: Every node has been assigned an interval to be in charge of.

Figure 7: Virtual Space Partitioning.

Then, after both steps have been completed, every node has been equitably assigned a partition of $\mathcal{V}$. All partitions are of same size. Figure 7(c) shows the resulting assignment of $\mathcal{V}$ on the example graph. A node u which has been assigned an interval $I(u)$ is responsible for storing location information of all nodes v such that $hash(v) \in I(u)$. Note that, as $\mathcal{V}$ is much smaller than the domain of the node identifiers, they are several nodes v such that $hash(v_i) = hash(v_j)$. Node u has thus to store information relative to all these nodes v.

Each internal node v also keeps in memory $I_{tree}(s\mathcal{T}(v))$, the interval for which its subtree is responsible but it does not store the intervals for which each subtree of its children is in charge of.

As we saw in Section 3, each internal node only has few children to which distribute the partitions of the virtual space. Thus, this operation does not include a lot of computing on nodes.

5.3 Registration

In order to be located afterward, each node u has to register its position (cluster identity) on each node responsible for $hash(u)$ in its cluster and in all other clusters. To register in its own cluster, it only has to send a Registration Request (see Subsection 5.6). To register in other clusters, u has to reach a node v_i in each cluster $\mathcal{C}_i$. Then each v_i sends a Registration Request for node u in its own cluster $\mathcal{C}_i$. To find nodes v_i, node u only has to pro-actively route toward $\mathcal{C}_i$ as described later in Subsection 5.7.

Nodes register their position every $\Delta(t)$ time units. We reserve for future works the study of the registration frequency, *i.e.*, the value of $\Delta(t)$, which may depend on how far a cluster is from the registering node.

5.4 Departures and arrivals

When a node arrives in a tree, it is responsible for none interval for a while.

When a node leaves, the information for which it was responsible is lost (but is still expected to be found in other clusters). Each internal node u is aware of the departures and arrivals of its children. If it sees too many changes among its children, it can locally decide to re-distribute $I_{tree}(s\mathcal{T}(u))$ among itself and its children thanks to the use of Hello Packets. The closer an internal node is to its cluster-head, the more important the logical space partition re-assignments are, since once an internal node is assigned a new partition, it has to re-attribute it over it subtree an so its children, up to the final leaves. Moreover, a new distribution of the partitions of $\mathcal{V}$ implies a new distribution of logical addresses of nodes. But, as shown in [20], the closer to the cluster-head an internal node is, the more stable its neighborhood is. Thus, the amount of re-assignments is likely to be low for such nodes.

When intervals are re-assigned, every node which previously was responsible for interval I_{old} and which is assigned a new partition I_{new} only keeps the latter information concerning $I_{old} \cap I_{new}$. In order to maintain the previous total information stored by the nodes and not to systematically loose it, when partitions are re-assigned, every node sends a register request all locations of nodes for which it is not responsible anymore *i.e.*, $\forall u \in V$ s. t.$hash(u) \in I_{old} \setminus I_{new}$.

5.5 Adding redundancy

As mentioned in Section 5.4, when a node collapses or leaves the network, the information it was responsible for is lost in a cluster till the next registrations. In order to overcome this drawback, a node may register its location d times in each cluster, d being a constant. For it, the virtual space $\mathcal{V}$ has to be distributed d times over the nodes of each cluster, each node being assigned of d non-overlapping partitions of $\mathcal{V}$. Each node has d different logical addresses. As d is a constant, the memory size required on nodes is still in $O(1)$ as, in average, each node would have $d * c$ locations to store instead of c (c and d are both constants). Yet, even when some nodes disappear, a node location is still expected to be found in cluster. However, this induces more messages when routing in the virtual space as requests have to follow each of the d partitionings of $\mathcal{V}$. Thus, in the worst case, routing in virtual space will lead to a cluster broadcasting. But, as studied in [21], such a cluster broadcasting can be performed in such a cluster topology in a very efficient way with low cost and latency.

5.6 Routing in the virtual space

In this section, we detail how the Interval Routing is performed in a tree. Routing in the virtual space is needed for registering a node location and locating nodes.

In our model, each node u has a unique identifier $Id(u)$. As in every DHT scheme, we assume that every node knows a specific function $hash$ which associates each node identifier to a value in the logical space $\mathcal{V}$: $hash : I\!R \rightarrow \mathcal{V}$, $Id(u) \rightarrow hash(u)$. As $\mathcal{V}$ is much smaller than the domain of the node identifiers $I\!R$, several nodes may have the same value returned by the $hash$ function. We propose to use the following tuple as a key for a node x: $\{hash(x), id(x)\}$.

A node u routes in the virtual space to reach a node v responsible for a given key key. For it, it needs three kinds of requests according to the purpose of its routing. In every case, u is looking for a node v such that $key \in I(v)$.

- u **wants to register a position:**
  u may need to register its own position or, if it is a border node, it may have been asked to register the position of a node u' standing in another cluster. In this case, u is looking for the node responsible for its own virtual address: $hash(u)$ or for the virtual address of u': $hash(u')$.
  $key = \{hash(u), u\}$ (resp. $key' = \{hash(u'), u'\}$)
  u then sends a Registration Request (RR) $\langle RR, key, \mathcal{C}(u), flag\rangle$ (resp. $\langle RR, key', \mathcal{C}(u'), flag\rangle$).

- u **needs to locate** x**:**
  In this case, u is looking for the node responsible for the virtual address of x: $hash(x)$.
  u sends a Location request (LR) $\langle LR, key = \{hash(x), x\}, i(u), flag\rangle$. The logical address of u, $i(u)$, will then be used to reply to node u.

- u **needs to answer a location request for a key it is responsible for (**$key \in I(u)$**):**
  In this case, node u has received a Location Request such that $\langle LR, key = \{hash(x), x\}, i(w), flag\rangle$ initiated by node w such that $key \in I(u)$. It has to answer to node w.
  It sends a Location Reply (Reply) $\langle Reply, key = \{i(w), -1\}, \mathcal{C}(x), flag\rangle$. Note that in this case, the key does not have to contain the real identity of w and node u only knows $i(w)$.

In every case, the value $flag$ is set to 1 by the node forwarding the message if key belongs to the interval its subtree is responsible for, it is set to 0 otherwise. As detailed later, it is useful for the routing decisions.

Remark that node u already knows the location (cluster Id) of its neighbors, of its cluster-head and of the nodes it is responsible for. Thus, if node v is such that $v \in \mathcal{H}(u) \cup \Gamma_1(u)$ or $hash(v) \in I(u)$, node u directly routes toward node v, skipping the localization steps (skipping steps 1 and 2 on Figure 5). It only performs the second phase of the indirect routing.

Due to the broadcast feature of wireless communications, nodes may receive requests which do not concern them. Indeed, a node can hear messages sent by all its neighbors in the network, even from the ones which are not its neighbors in the tree (neither a child or its parent). When receiving a request, a node has thus to decide either to forward it or to ignore it, basing on the key and the flag contained in it.

The routing process is the same whatever the kind of message (LR, RR or Reply) as the routing decision is based on the key and the flag. The field $flag$ is set at each forwarding step. Algorithm 2 describes the routing operation.

Upon reception of a message M (RR, LR or Reply) containing the key $\{hash(x), x\}$ coming from node u ($u \in \Gamma_1(v)$), node v end routing if it is responsible for the key ($key \in I(v)$) or if it is the wanted node ($key = \{hash(v), v\}$). This second case is pretty rare as it can only happen when the message initiator node x is in the same cluster than the node v it is looking for ($\mathcal{C}(x) = \mathcal{C}(v)$) and if v is on the path of the request in the tree to reach the node responsible for $hash(v)$.

Note that node u may just forward itself the message and is not necessarily the request initiator.

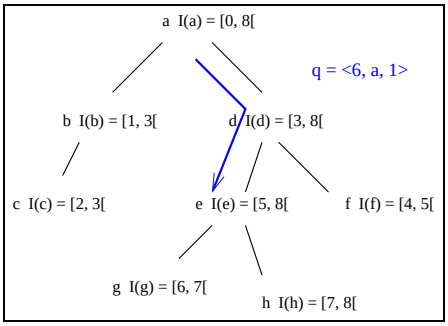
When a RR message $\langle RR, key = \{hash(u), u\}, \mathcal{C}(u), flag\rangle$ reaches its final destination v, v updates the location of u in its table. When a Reply message $\langle Reply, key = \{i(u), -1\}, \mathcal{C}(x), flag\rangle$ reaches its final destination u, u is thus able to route to $\mathcal{C}(x)$ using the hierarchical routing. When a LR message $\langle LR, key = \{hash(x), x\}, i(w), flag\rangle$ reaches its final destination v (in charge of $hash(x)$), v answers the sender by initiating a Reply message $\langle Reply, \{i(w), -1\}, \mathcal{C}(x), flag\rangle$.

If $key \neq \{hash(v), v\}$, node v forwards M in three cases:

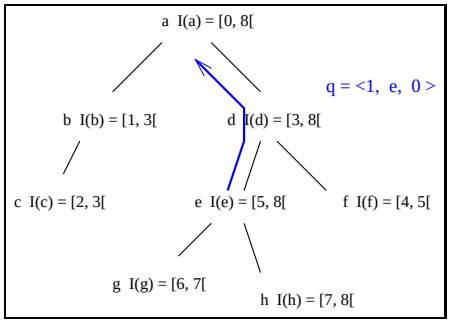
- If $u = \mathcal{P}(v)$ and $key \in I_{tree}(s\mathcal{T}(v))$ (the message is coming from node v's parent and the key is in its subtree's interval). See Figure 8(a).
- If $u \in \mathcal{C}h(v)$ (the message is coming from a child of node v), v forwards M:
  - if $key \notin I_{tree}(s\mathcal{T}(v))$ (the key is not in its subtree, and obviously neither in the subtree of its child v): the message has to follow its way up the tree. See Figure 8(b).
  - if $key \in I_{tree}(s\mathcal{T}(v))$ and $key \notin I_{tree}(s\mathcal{T}(u))$ ($flag = 0$) (the key is in its subtree but not in the subtree of its child from which it has received the request): the message has to be forwarded down its subtree to another child. See Figure 8(c).

And node v discards M in all other cases, which means:

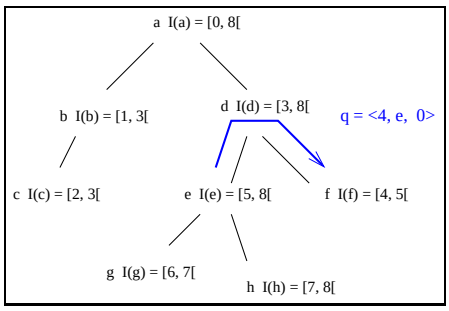
- If $u \notin \mathcal{P}(v) \cup \mathcal{C}h(v)$ (the message is coming from a node which is neither the parent nor a child of node v). See Figure 9(a). Even if node v is concerned by the request, it can not answer it right now. It has to wait till the request arrives from a branch of the tree. For instance, in Figure 9(a), node d has to wait till a forwards the request.



(a) Case 1: The message is going down the tree. a is looking for the node in charge of 6.



(b) Case 2: The message is going up the tree. e is looking for the node in charge of 1.



(c) Case 3: The message is going up and down the tree. e is looking for the node in charge of 4.

Figure 8: Different cases of figures of when a message received at node d is forwarded.

- If $u \in \mathcal{C}h(v)$ and $key \in I_{tree}(s\mathcal{T}(u))$ ($flag = 1$) (the message is coming from a child which subtree is responsible for the key). Thanks to the flag, u knows that the subtree of the sender node is in charge of the key and that it does not need to forward the request. The message goes up and down the tree via its child v. See by Figure 9(b).
- If $u = \mathcal{P}(v)$ and $key \notin I_{tree}(s\mathcal{T}(v))$ (M is coming from v's parent but the subtree of v is not responsible of the key). u is not concerned by the request, it does not forward. One of its siblings is in charge of the key and thus will forward or answer the request. See Figure 9(c).

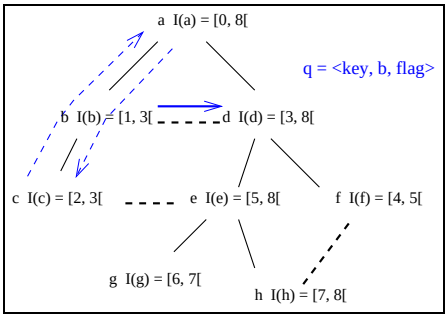
Algorithm 2 Query Forwarding

For all node u**, upon reception of a message** $\langle Type, key = \{hash(x), x\}, X, flag\rangle$**,** X **depending on the kind of message coming from a node** $v \in \Gamma_1(u)$ **and initiated at a node** y:

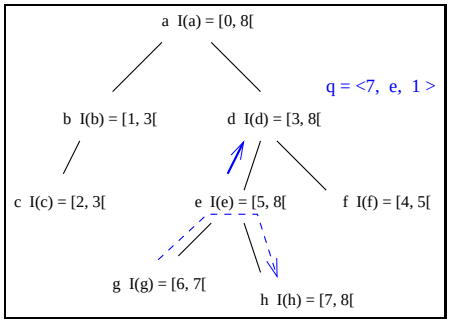
if $(u = x)$ **then** Reply sending $\langle Reply, \{X = i(y), -1\}, \mathcal{C}(u), flag\rangle$ and Exit **end**

▷ *u is the wanted node. It can answer node y.*

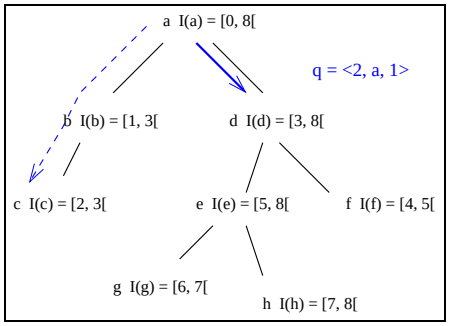
if $(key \in I(u))$ **then**



(a) Case 1: The message is coming from a neighbor of node d but does not concern node d.



(b) Case 2: e is looking for the node in charge of 4. The message is going up and down the tree on node e but is heard by node d. dashed links represent the links of the graph G which are not in the tree T.



(c) Case 3: a is looking for the node in charge of 7. Node d is not in charge of the researched key. One of its sibling will forward.

Figure 9: Different cases of figures when a message received on node d is discarded on d. The dashed arrows represent the possible paths followed by the message.

▷ *u is responsible for storing the wanted key. The message has reached its final destination.*
if (Type = LR) **then** Reply sending $\langle Reply, \{X = i(y), -1\}, \mathcal{C}(x), flag\rangle$ and Exit **end**
if (Type = RR) **then** Register the location of node x and Exit **end**
if (Type = Reply) **then** Route toward the destination cluster X and Exit **end**
end
if ($v = \mathcal{P}(u)$) **then**
▷ *The message is going down the tree.*
if ($key \in I_{tree}(s\mathcal{T}(u))$) **then** Set $flag$ to 1 and Forward.
▷ *$\exists w \in s\mathcal{T}(u)$ such that $key \in I(w)$. See Figure 8(a).*
else Discard. ▷ *See Figure 9(b).*
end

else
if ($v \in \mathcal{C}h(u)$) **then**
▷ *The query is coming up the tree from a child of node* u.
if ($key \notin I_{tree}(s\mathcal{T}(u))$) **then** Set $flag$ to 0 and Forward.
▷ *The query is forwarded up the tree. See Figure 8(b).*
else
▷ $\exists w \in s\mathcal{T}(u) \setminus \{u, v\}$ *such that* $key \in I(w)$.
if ($flag = 0$) **then** Set $flag$ to 1 and Forward.
▷ $key \notin I_{tree}(s\mathcal{T}(v))$ *but as* $key \in I_{tree}(s\mathcal{T}(u))$, u *has to forward the query to its other children. The query goes up and down via* u. *See Figure 8(c).*
else Discard. ▷ *The query goes up and down via* v. *See Figure 9(c).*
end
end
else Discard. ▷ *See Figure 9(a).*
end
end

5.7 Routing in the physical network

As mentioned in Section 4, we propose to apply a hierarchical routing by using the cluster topology. As the number of clusters is constant when the intensity of nodes increases, the number of nodes per cluster increases but the mean node eccentricity inside a cluster remains low and constant (between 3 and 4 hops). We thus suggest to use a reactive routing protocol inside the clusters and a proactive routing protocol between the clusters. As the number of clusters is constant, the number of routes to store per cluster is also constant ($O(1)$ routes to maintain). As the mean node eccentricity is low, the latency is expected to be low as well as the overhead induced by a reactive routing in a cluster.

A proactive routing scheme between the clusters implies for a node u to know the sequence of clusters to go through from its own cluster $\mathcal{C}(u)$ toward any other one. The first question to address is: who has this information? All nodes in $\mathcal{C}(u)$ or only the cluster-head $\mathcal{H}(u)$? We suggest that this parameter depends on the frequency of messages sent toward nodes in other clusters. Indeed, if a node rarely needs to communicate with a node in another cluster, it can ask its cluster-head for the route at each communication to minimize the information to store on each node. Otherwise, if communications between clusters are frequent, the information is distributed over all nodes of the cluster by the cluster-head. This broadcasting task may be performed in an efficient and quick way as claimed in [21]. We reserve for future works a deeper analysis of the nodes detaining the inter-cluster routing tables. From now, we just suppose that a node knows the following cluster the message has to go through to reach the final destination, whatever the way it gets it.

Suppose node u needs to reach node v. If $v \in \Gamma_1(u) \cup \{\mathcal{H}(u)\}$, then node u already knows the location of v and how to reach it. Otherwise, node u performs a lookup operation to learn $\mathcal{C}(v)$ before applying the routing rules. The routing process is illustrated on Figure 10. If $\mathcal{C}(v) = \mathcal{C}(u)$, then u initiates a reactive routing within its cluster to reach v. Otherwise, it looks at its routing table for the next cluster $\mathcal{C}(w)$ on the route toward $\mathcal{C}(v)$ and initiates a reactive routing in its cluster to look for a node $x \in \mathcal{C}(u)$ which is a frontier node of $\mathcal{C}(w)$ (*i.e.*, x is such that $x \in \mathcal{C}(u)$ and $\exists y \in \Gamma_1(x) \cap \bar{\mathcal{C}(u)}$).

The message is then sent to x which forwards it to one of its neighbors y in the neighboring cluster $\mathcal{C}(w)$. The routing process is thus reiterated at node y and so on till reaching the final destination.

As the reactive routing step is confined in clusters which have low diameters, the induced flooding and its undesirable aspects are limited. Note that it can also be enhanced as in [21].

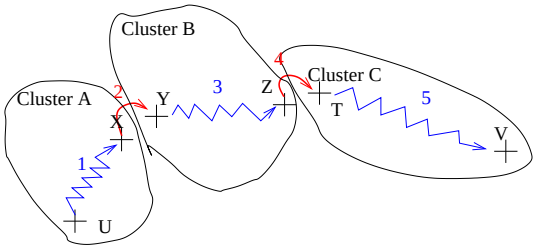


Figure 10: Node u needs to communicate with node v. Thanks to the inter-cluster pro-active routing algorithm, it knows that $\mathcal{C}(v) = C$ and that the next cluster to reach cluster C is cluster B. It uses a reactive routing protocol to reach a node neighboring B: node x (Arrow 1). Node x forwards the message to one of its neighbors in B: node y (Arrow 2). y knows that clusters B and C are neighbors, it forwards the message to a node in B frontier node for the cluster C: node z (Arrow 3). z forwards the message to one of its neighbors in C: node t (Arrow 4). Finally, t performs a reactive routing in C to find v (Arrow 5).

Algorithm 3 presents this routing scheme. We use $Next\_Hop(cluster_1, cluster_2)$ to denote the function of the pro-active routing protocol which returns the next cluster on the route from $cluster_1$ to $cluster_2$. This function is known by every node.

Algorithm 3 Routing

For a message M **sent by node** $x \in \mathcal{C}(x)$ **to node** $y \in \mathcal{C}(y)$ **do**
  $\mathcal{C}_{current} = \mathcal{C}(x)$
  $\mathcal{C}_{next} = \mathcal{C}(x)$
  while ($\mathcal{C}_{next} \neq \mathcal{C}(y)$) do
    $\mathcal{C}_{next} = Next\_Hop(\mathcal{C}_{current}, \mathcal{C}(y))$
    Reactively route M toward node $u \in \mathcal{C}_{current}$ such that $\exists v \in \Gamma_1(u) \cap \mathcal{C}_{next}$.
    Node u sends M to node v.
    $\mathcal{C}_{current} = \mathcal{C}_{next}$
  end do
  ▷ *The message has reached the cluster of the destination node.*
  Reactively route M toward destination node y.
end do

Registering node location. If node u only wants to register its position in another cluster, it uses the same scheme. Let's say u wants to register in cluster C. It already has the inter-cluster path to reach it and thus does not need to perform any *lookup* operation. It runs Algorithm 3 till reaching any node in cluster C (node t in our example). Node t then sends the following Registration Request $\langle RR, key = \{hash(u), u\}, A, flag\rangle$ in cluster C. Note that when forwarding the request of node u

to node z, node y can also send a Registration Request for node u to register it in cluster B at the same time.

5.8 Stretch Factor

The *stretch factor* is the difference between the cost (or length) of the route provided by the routing protocol and the optimal one. Flat routing protocols generally provide optimal routes.
Plaxton, Rajaraman and Richa were the first to provide a lookup protocol with analytical bounds of the stretch factor in [27]. As far as we know, none of the indirect routing proposals for wireless multi-hop networks has provided an analysis of it, unlike protocols in Peer-to-Peer applications. Indeed, many DHT applications in Peer-to-Peer proposals provide an analysis of this stretch factor and try to minimize it, as in Pastry [32] or Tapestry [41]. All of them provide an expected stretch in finding target which is a multiplicative constant: the cost is thus proportional to the distance in the graph between the source and the destination.
Only the LAND protocol [1] proposes an indirect routing that for any $\epsilon > 0$ has worst case stretch factor bounded by $(1+\epsilon)$. ϵ is tunable. To respect it, the protocol has to collect information concerning m nodes which may be several hops away, m depending on ϵ. For a fixed ϵ, the authors of LAND prove that m is in $0(log(n))$ where n is the total number of nodes in the network. Nevertheless, as studied in [29], the underlying network has to fit a hard assumption which is a bounded growth. And, if m effectively is proportional to $log(n)$ under this assumption, it actually is proportional to $c \times log(n)$ where c is a very high constant ($c \approx 10^6$ in the best case), which is not mentioned in the paper.

The length of the routes provided by our proposed routing scheme is equal to twice the length of the route in the tree for locating the target node (Step 1 of the indirect routing), more the length of the final route to the target (Step 2 of the indirect routing). As the routing process of the second step is performed over the physical topology with the use of flat protocols, the stretch factor of this step is close to 1. Thus, only the stretch factor induced by the first step (routing in the virtual space) is noticeable. Our global stretch factor s is such that $s = 1 + 2 \times l(u, v)$ where $l(u, v)$ is the length of the path in the virtual space (tree) from node u to node v, v being the rendezvous point storing the information needed by u. As this routing scheme is performed within a cluster only, we can bound $l(u, v)$ by the length of the longer path in the tree: $l(u, v) \leq 2 \times Tree\_depth$. Finally, we have $s \leq 1 + 4 \times Tree\_depth$. As already mentioned, $Tree\_depth$ is a low constant (between 3 and 4 hops). Thus, routes provided by our locating/routing proposition may be costly in term of number of hops only for small paths in the graph since, as we evolute in a large scale environment, this constant can be expected negligible in front of most of the path lengths. It is not the case only for some routes within a cluster. Moreover, as we have already seen in Section 4 and unlike the LAND protocol, we only need information concerning few nodes and up to few hops away.

Simulation environment. All simulations we performed and which are mentioned in this section, follow the same model. We use a simulator we developed. Nodes are randomly deployed using a Poisson Point Process in a $(1 + 2R) \times (1 + 2R)$ square with various levels of intensity λ. In such a process, λ represents the mean number of nodes per surface unit. The communication range R is set

to 0.1 in all tests. In each case, each statistic is the average over 1000 simulations. Only the points within the square w of size 1×1 are taken into account to estimate the different quantities (mean degree, mean density, etc.). But in order to avoid edge effects, the samples of the Point Process are generated in a larger window W. For instance, if we estimate the mean degree of the nodes ($\overline{\delta}$), we take the degree of the points in w to compute $\overline{\delta}$. If we did not consider the points of W, the points close to the edge within w would have a degree less than points close to the center introducing a bias in the estimation. Both windows are shown in Figure 11. This technique is called "minus-sampling", a more detailed description can be found in [36] page 132.

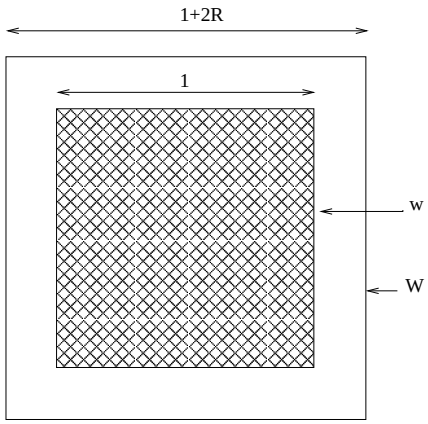


Figure 11: Only points of w are taken into account to estimate the different quantities, but the Point Process is generated in the square W in order to avoid the edge effects.

We consider every pair of nodes (u, v) in a same cluster ($\mathcal{C}(u) = \mathcal{C}(v)$) and we compared the length of the path $d(u, v)$ in the graph (physical topology) to the length of the path $d_{tree}(u, v)$ in the tree (logical topology) in order to estimate the cost for routing in the tree. Results are shown in Figure 12. As we can note, path lengths are quite constant when the number of nodes increase and remains low (between 3 and 4 hops). To reach a rendezvous node v, node u has to use the tree (as it needs to perform the interval routing in the logical space). As Table 4 shows, using the tree only adds 1 hop in average than using the physical graph (so 2 hops for a round trip) and induce much less traffic overhead and memory requirements than using a classical routing in a cluster.

Nevertheless, we remind that the total stretch factor for a node u looking for a node v, actually is equal to the length of the round trip path from node u to node w responsible for $hash(v)$. It does thus not exactly correspond to this additional hop, but as the direct routing which is used in the second routing step has a negligible stretch factor, it is not far from it. So, we can say that the locating/routing approach we propose does not add much latency, except when a node tries to locate a node in the same cluster than its. However, as we are dealing with a great amount of nodes, the proportion of such communications is low.

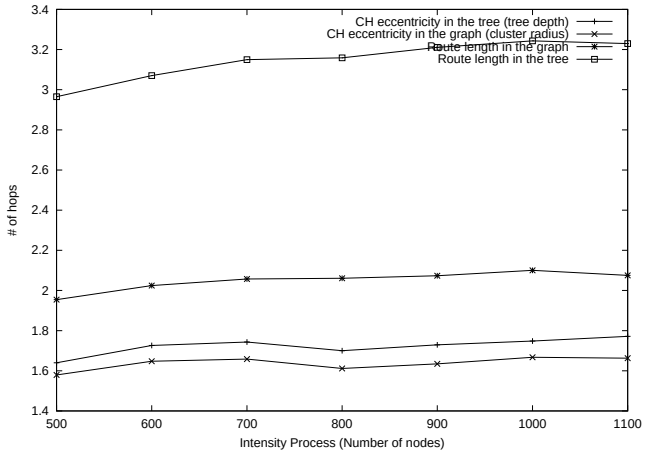


Figure 12: Comparisons of distances in the logical and physical topologies.

	500 nodes	600 nodes	700 nodes	800 nodes	900 nodes	1000nodes
Graph Vs Tree (# of hops)	1.01	1.05	1.10	1.10	1.14	1.14

Table 4: Difference between path length in the tree and path length in the graph between two nodes of the same cluster.

However, even if the second step of our indirect routing is close to the optimal, it is not indeed at every time. Indeed, inter-cluster pro-active routing will consider the shortest sequences of clusters for each path. However, a longer sequence of clusters does not necessary mean a longer number of hops as some cluster may be crossed with only one hop. We reserve for future works the analysis of the inter-cluster paths. This issue is similar to the one in the "BGP"[6] protocol where sequences of AS (Autonomous System) do not always lead to the optimal paths in number of hops between routers.

6 Conclusion

In this report, we have proposed a way for locating a node and routing toward it in a large scale clustered wireless multi-hop network. This scheme lies on a hierarchical and indirect routing which only requires $O(1)$ memory size on nodes and generates low traffic overhead and low latency. It takes advantage of the underlying tree structure to perform an efficient Interval Routing within clusters allowing a quick lookup. Then, unlike what is usually proposed in the literature, we use a pro-active routing approach between clusters and a reactive one inside the clusters. In future works, we intend

[6] www.freesoft.org/CIE/RFC/1772/

to analyze deeper our algorithm concerning refreshing periods of node location registrations and re-assignments of the partitions of the logical space over the different trees. We also intend to study the impact of the mobility over the whole locating scheme and more precisely over these refreshing periods. As the underlying topology has presented a good behavior over node mobility and that the localization algorithm is quasi-local, we expect it to be pretty robust towards node mobility as well. Then, we plan to compare our proposition to other existing ones as for instance SAFARI [31] which also uses this reverse approach for the hierarchical routing. Some primary works make us expect our proposition to generate less message flooding over the network and thus to be less costly.

References

[1] I. Abraham, D. Malkhi, and O. Dobzinski. LAND: Locality Aware Networks for Distributed Hash Tables. In *ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, New Orleans, LA, USA, 2004.

[2] A. Amis, R. Prakash, T. Vuong, and D. Huynh. Max-Min d-cluster formation in wireless ad hoc networks. In *Proceedings of the IEEE INFOCOM*, Tel-Aviv, Israël, March 2000. IEEE.

[3] F. Araujo, L. Rodrigues, J. Kaiser, L. Changling, and C. Mitidieri. CHR: A Distributed Hash Table for Wireless Ad Hoc Networks. In *4th International Workshop on Distributed Event-based Systems (DEBS'05)*, Columbus, Ohio, USA, June 2005.

[4] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, February 2003.

[5] L. Blazevic, L. Buttyan, S. Capkun, S. Giordano, and J.-Y. Le Boudec. Self-organization in mobile ad-hoc networks: the approach of Terminodes. *IEEE Communications Magazine*, 39(6):166–174, June 2001.

[6] L. Blazevic, S. Giordano, and J.-Y. Le Boudec. Self-organized Terminode routing. *Journal of Cluster Computing*, 5(2), April 2002.

[7] S. Capkun, M. Hamdi, and J.-P. Hubaux. GPS-free positioning in mobile ad-hoc networks. In *Proceedings of The 34th Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii, USA, January 2001.

[8] B. Chen and R. Morris. L+: Scalable landmark routing and address lookup for multi-hop wireless networks. Mit lcs technical report 837, MIT, March 2002.

[9] G. Chen, F. Garcia, J. Solano, and I. Stojmenovic. Connectivity-based k-hop clustering in wireless networks. In *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, Hawaii, USA, January 2002.

[10] Y. P. Chen, A. L. Liestman, and J. Liu. Clustering algorithms for ad hoc wireless networks. *Ad Hoc and Sensor Networks*, 2004.

[11] J. Eriksson, M. Faloutsos, and S. Krishnamurthy. Scalable ad hoc routing : The case for dynamic addressing. In *INFOCOM*, Hong Kong, China, March 2004.

[12] Y. Fernandess and D. Malkhi. k-clustering in wireless ad hoc networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, Toulouse, France, 2002. ACM Workshop On Principles Of Mobile Computing.

[13] P. Fraigniaud and P. Gauron. An overview of the content-addressable network D2B. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC'03)*.

[14] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster based approach for routing in dynamic networks. In *ACM SIGCOMM*, pages 49–65. ACM, April 1997.

[15] J. Li, R. Morris, J. Jannotti, D. S. Decouto, and D. R. Karger. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (Mobicom'00)*, pages 120 – 130. ACM, August 2000.

[16] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7):1265–1275, 1997.

[17] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*, 2002.

[18] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, MIT Faculty Club, Cambridge, MA, USA, March 2002.

[19] N. Mitton, A. Busson, and E. Fleury. Self-organization in large scale ad hoc networks. In *The Third Annual Mediterranean Ad Hoc Networking Workshop, MED-HOC-NET 04*, Bodrum, Turkey, June 2004.

[20] N. Mitton, A. Busson, and E. Fleury. Broadcasting in self-organizing wireless multi-hop network. Research report RR-5487, INRIA, February 2005.

[21] N. Mitton and E. Fleury. Efficient broadcasting in self-organizing multi-hop wireless network. In *4th International Conference on AD-HOC Networks & Wireless (Ad Hoc Now'05)*, Cancun, Mexico, October 2005.

[22] N. Mitton, E. Fleury, I. Guérin-Lassous, and S. Tixeuil. Self-stabilization in self-organized multihop wireless networks. In *2nd International Workshop on Wireless Ad Hoc Networking (WWAN'05)*, Columbus, Ohio, USA, June 2005.

[23] E. T. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFOCOM*, New-York, USA, June 2002.

[24] D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *Proceedings of GLOBECOM'01*, November 2001.

[25] N. Nikaein, H. Labiod, and C. Bonnet. DDR-distributed dynamic routing algorithm for mobile ad hoc networks. In *1st ACM international symposium on mobile ad hoc routing and computing*, Boston, MA, USA, November, 20th 2000. ACM.

[26] C. Perkins. *Ad hoc networking.* Addison-Wesley, 2001.

[27] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'97)*, pages 311–320, June 1997.

[28] H. Pucha, S. M. Das, and Y. C. Hu. Ekta: An efficient DHT substrate for distributed applications in mobile ad hoc networks. In *WMCSA*, pages 163–173, 2004.

[29] H. Radigois and C. Oros Blanch. Application d'un protocole de localisation P2P à des réseaux ad hoc grande échelle. Master Report, INSA Lyon, June 2004.

[30] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[31] R. Riedi, P. Druschel, Y. C. Hu, D. B. Johnson, and R. Baraniuk. SAFARI: A self-organizing hierarchical architecture for scalable ad hoc networking networking. Research report TR04-433, Rice University, February 2005.

[32] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001.

[33] C. Santivanez, B. McDonald, I. Stavrakakis, and R. R. Ramanathan. On the scalability of ad hoc routing protocols. In *INFOCOM*, New-York, USA, June 2002.

[34] N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The computer Journal*, 28:5–8, 1985.

[35] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (Sigcomm'01)*, pages 149–160. ACM Press, 2001.

[36] D. Stoyan, S. Kendall, and J. Mecke. *Stochastic geometry and its applications, second edition*. John Wiley & Sons, 1995.

[37] J. Van Leeuven and R. Tan. Interval routing. *The computer Journal*, 30:298–307, 1987.

[38] A. C. Viana, M. Dias de Armorim, S. Fdida, and J. Ferreira de Rezende. Indirect routing using distributed location information. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, page 224, Washington, DC, USA, 2003. IEEE Computer Society.

[39] A. C. Viana, M. Dias de Armorim, S. Fdida, and J. Ferreira de Rezende. Self-organization in spontaneous networks: the approach of DHT-based routing protocols. *Ad Hoc Networks Journal*, 2005.

[40] J. Wu and W. Lou. Forward node set based broadcast in clustered mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 3(2):141–154, 2003.

[41] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), January 2004.

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

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

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