Distributed Node Location in Clustered Multi-hop Wireless Networks

Nathalie Mitton, Eric Fleury

Distributed Node Location in clustered multi-hop wireless networks.

Nathalie Mitton and Eric Fleury

INRIA/ARES - INSA de Lyon - 69621 VILLEURBANNE Cedex
tel: 33-(0)472-436-415
firstname.lastname@insa-lyon.fr

Abstract. Wireless routing protocols 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 describe our 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.
keywords: wireless networks, localization, DHT, interval routing.

1 Introduction

Wireless multi-hop networks such *ad hoc* or sensor networks are mobile networks of mobile wireless nodes, requiring no fixed infrastructure. Every mobile can independently move, disappear or appear at any time. A routing protocol is thus required to establish routes between terminals which are not in transmission range one from each other. Due to the dynamics of such 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 are all flat routing protocols: there is no hierarchy, all the terminals have the same role and are potential routers. 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 [22]. 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*. In most of the literature, a hierarchical routing means using a proactive[1] routing protocol inside the

[1] Nodes permanently keep a view of the topology. All routes are available as soon as needed.

clusters and a reactive[2] one between the clusters [6,9,19,20]. 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 [13]. 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 [15]. 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 [21] has proposed such an approach, even if most of the clustering schemes [1,19] 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 paper is organized as follows. The description of the initial clustering algorithm as well as interesting features for the localization algorithm are given in Section 2. Then, we describe in Section 3 how we propose to take advantage of the underlying structure of our organization to perform our localization algorithm. Then, we describe in Section 4 our proposition. Lastly, in Section 5, we discuss some improvements and future works.

2 Our cluster organization

In this section, we summarize our previous clustering work on which we apply our localization scheme. Only basic features which are relevant for localization and routing are mentioned here. For more details, please refer to [13,14,16].
Let's first introduce some notations. We classically model a wireless multi-hop network

[2] Routes are searched on-demand. Only active routes are maintained.

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. If $dist(u, v)$ is the Euclidean distance between nodes u and v, then $\exists (u, v) \in E$ iff $dist(u, v) \leq R$. R is thus 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. 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 ($\mathcal{P}(v) \in s\mathcal{T}(u)$): $\{v \in s\mathcal{T}(u) \cap \Gamma_1(u)\} \Leftrightarrow \{v \in \mathcal{C}h(u)\}$ or $\left\{v \in s\mathcal{T}(u) \cap \bar{\Gamma}_1(u) \setminus \{u\}\right\} \Leftrightarrow \{\mathcal{P}(v) \in s\mathcal{T}(u)\}$

2.1 The clustering heuristic

Our initial goal was to propose a way to use multi-hop wireless networks over large scales. 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 [5,11] have a radius of 1, in [1,7] the radius is set a priori.), 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 [1], 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* [13]. The notion of density of a node u (noted $\rho(u)$) characterizes the "relative" importance of u in the network and within its neighborhood. This link density 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:* $\rho(u) = \frac{|\{e=(v,w) \in E \mid w \in \{u\} \cup \Gamma_1(u) \text{ and } v \in \Gamma_1(u)\}|}{\delta(u)}$

Because of page restrictions, we only give here a sketch of the cluster/tree formation, but the algorithm and an example can be found in [13]. 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. The node Id are used to break ties. In this way, two neighbors can not be both cluster-heads. 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 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 thus build the clusters by building a spanning forest of the network in a distributed and local way. As proved in [16], 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 border node. A node is a frontier node if at least one of its neighbors does not belong to the same cluster than itself.

2.2 Some characteristics of our clustering algorithm

The cluster formation algorithm stabilizes when every node knows its *correct* cluster-head value. In [16], 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 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. It has shown to be upper bounded by a constant asymptote when the number of nodes in the network increases. Compared to other clustering schemes as DDR [19] or Max-Min d cluster [1], 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 [13]). Moreover, our algorithm presents a smaller 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.

	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 1. Some cluster and clustering trees characteristics.

Other interesting features for routing and locating obtained by simulations are gathered in Table 1. These characteristics illustrate some of our motivations for our proposition as explained later in Section 3. The eccentricity of a node is the greater distance in number of hops between itself and any other node in its cluster. We can see in Table 1 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 as a great proportion of leaves and a small amount of non-leaf children per node. This feature and the *"Degree in the tree of non-leaf nodes"* entry in Table 1 show that, in average, an internal node does not have a lot of children.

3 Basic ideas

In fixed networks, routing information is embedded into the topological-dependent node address. For instance, an IP address both identify a node and locate 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 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)* **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 1 illustrates such a routing process.

We propose to use such an indirect routing scheme for routing in large scale multihop 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 1) and the rendezvous node (node w) is much greater than the distance between the source (node u) and the destination (node v). Indeed, if the request has to cross twice the whole network before two nodes are able to directly communicate, we waste bandwidth and latency. Distributed Hash Tables (DHT) are a basis for indirect routing.They 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 this 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.

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 [24,8,12]. 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

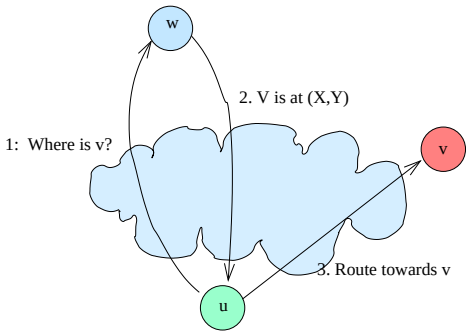


Fig. 1. Indirect Routing: Node u needs to ask w where v is.

knowing where v is. 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, 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 [2,17,18], in the Terminodes project [3] or in the Grid project [10]. 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 performing a routing again in the virtual space. 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 [26] or L+ [4] on which is based SAFARI [21]. 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.

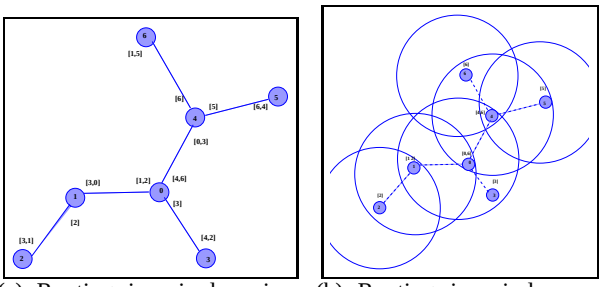
Thus, in DHT-based systems, we can consider two phases of routing: *(i)* a routing toward the rendezvous node which stores the needed information (Arrow 1 on Figure 1) and *(ii)* a routing toward the final destination node which location had been obtained by the lookup operation (Arrow 3 on Figure 1). 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). Indeed, as we propose to distribute a virtual space over each cluster, routing to the destination in the virtual space is not possible as the destination node may not be in the same cluster as the sender node and thus be in a different virtual space.

In our proposal, we propose nodes register their cluster Id as location information. As seen in Section 2, we have a tree structure. We propose to partition the virtual space $\mathcal{V}$ each tree and that each node registers on 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 2), 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 on the virtual space. Interval Routing is was introduced in wired networks by Santoro and Khatib in [23] 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 [25] showed 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 2). 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).

Summary and complexity analysis. To sum up, we propose to apply an indirect routing scheme over our clustered network by using DHT which associate each node identifier to a virtual address of a space $\mathcal{V}$. The set of virtual addresses $\mathcal{V}$ is partitioned d times over the nodes of each cluster. As the number of cluster is constant and clusters are homogeneous, each node finally stores $O(1)$ location information.



(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).

Fig. 2. Edges in wired networks (a) Vs hyper-edges in wireless networks (b).

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.

4 Our proposition

In this section, we describe how we wish to use DHT and Interval Routing over our cluster organization. However, because of page restriction, several details have been eluded but can be found in [14].

Virtual space partitioning. In this section, we present the way we distribute the partitions of $\mathcal{V}$, which leads to a Depth-First Search (DFS) ILS, *i.e.*, the optimal ILS for a tree . Let $I(u)$ be the partition of $\mathcal{V}$ node u is assigned. $i(u)$ is the first element of $I(u)$: $I(u) = [i(u), ...[|i(u) \neq hash(u)$. $i(u)$ is used as the virtual identifier of node u in the virtual space $\mathcal{V}$. Let $I_{tree}(s\mathcal{T}(u)) = \bigcup_{v \in s\mathcal{T}(u)} I(v)$ be the interval/partition of $\mathcal{V}$ of which the subtree of node u is in charge. $|I|$ is used to refer to the size of interval I. 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: $\mathcal{V} = \bigcup_{v \in \mathcal{C}(u)} I(v)$.
- 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 [27] 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. 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). The complexity in time of our 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 $O(1)$. Each step has a time complexity of $O(Treedepth)$. 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 $hash(v_i) = hash(v_j)$. As each internal node only has few children to which distribute the partitions of the virtual space (Section 2), this ILS does not include a lot of computing on nodes.

Step 1**.** As seen in Section 2, every node u might be aware in an expected bounded time, of the parent of each its neighbors. It 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). 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 neighbors in a low time and so the cluster-head.

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. Once an internal node u has assigned partitions of the virtual space among its children, it only stores the interval $I_{tree}(s\mathcal{T}(v))$ for which its subtree is responsible (and not intervals each of its children is in charge of). Then, it stores location information for nodes which key is in $I(u)$ only (and not for all keys in $I_{tree}(s\mathcal{T}(u))$).

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. When it sees too many changes among its children, it locally re-distributes $I_{tree}(s\mathcal{T}(u))$ among itself and its children. When intervals are re-assigned, in order to maintain the previous information stored by the nodes and not to loose it, every node keeps the latter information it was responsible for, in addition to the new one, for a period time $\Delta(t)$, $\Delta(t)$ being the period at which nodes register their location.

Routing in the virtual space. In this section, we detail how the Interval Routing is performed in a tree. The routing is performed till reaching the node responsible for this key. 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 use the following tuple as a key for a node x: $\{hash(x), id(x)\}$. In the following, we may use only x instead of $Id(x)$. A node u uses that kind of routing when it needs to reach a node responsible for a given key, which can happen for three reasons:

- u **wants to register a position:** u may need to register its position. In this case, u is looking for the node responsible for its own virtual address: $hash(u)$. u then sends a Registration Request (RR) $\langle RR, key = \{hash(u), u\}, \mathcal{C}(u), flag\rangle$.
- 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) $\langle LR, key = \{hash(x), x\}, i(u), flag\rangle$. $i(u)$, which is the identifier of u in the virtual space, 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 $\langle LR, key = \{hash(x), x\}, i(v), flag\rangle$ initiated on node v such that $key \in I(u)$. It has to answer to node v by sending a Location Reply (Reply) $\langle Reply, key = \{i(v), -1\}, \mathcal{C}(x), flag\rangle$.

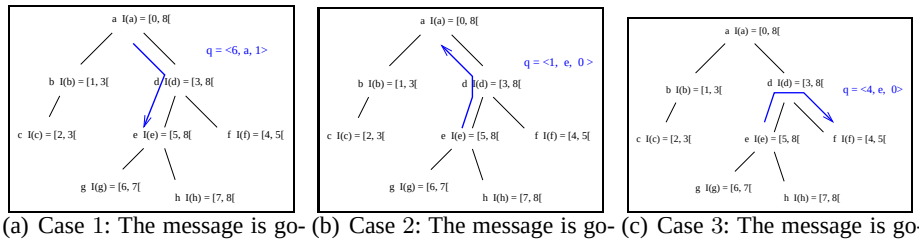
The routing process is the same whatever the kind of message (LR, RR or Reply) as the routing decision is only based on the key. In every case, the value $flag$ is set to 1 by the node forwarding the message if key belongs to the interval its subtree is responsible for, it is set to 0 otherwise. As detailed 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 1).

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 decides to end routing, forward M or discard M. Note that node u may just forward itself the message and is not necessarily the request initiator. The routing ends when M reaches a node v which either is responsible for the wanted key ($key \in I(v)$) or is the wanted node ($key = \{hash(v), v\}$).
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 3(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 3(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 3(c).

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

- If $u \notin \mathcal{P}(v) \cup \mathcal{C}h(v)$ (M is coming from a node which is neither the parent nor a child of node v). Figure 4(a).



(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.

Fig. 3. Different cases of figures of when a message received on node d is forwarded.

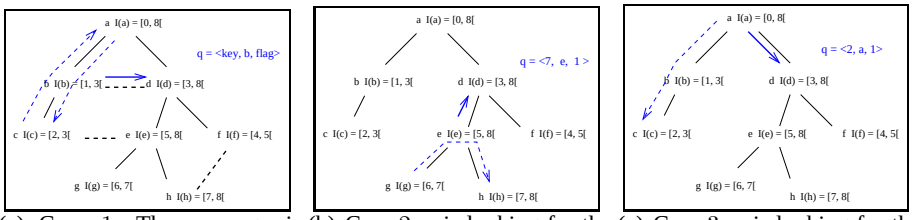
- If $u \in \mathcal{C}h(v)$ and $key \in I_{tree}s\mathcal{T}(u))$ $(flag = 1)$ (M is coming from a child v which subtree is responsible for the key). Thanks to the flag, u knows it does not need to forward as the message goes up and down the tree via v. Figure 4(b).
- If $u \in \mathcal{P}(v)$ and $key \notin I_{tree}(s\mathcal{T}(v))$ (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 will. Figure 4(c).

Algorithm 1 describes the routing operation. 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 $\langle Reply, \{i(w), -1\}, \mathcal{C}(x), flag\rangle$.

Algorithm 1 Query Forwarding

For all node u**, upon reception of a message** $\langle Type, key = \{hash(x), x\}, X, flag\rangle$**,** $X \in \{RR, Reply, LR\}$ **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**
▷ *u is responsible for storing the key. The message has reached its final destination.*
if (Type = LR) **then** Send $\langle Reply, \{X = i(y), -1\}, \mathcal{C}(x), flag\rangle$ 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, 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 3(a).*
else Discard. ▷ *See Figure 4(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 3(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. Figure 3(c).*
else Discard. ▷ *The query goes up and down via* v. *See Figure 4(c).*
end
end
else Discard. ▷ *See Figure 4(a).*
end
end



(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 key 4. The message is going up and down the tree on node e but is heard by node d.

(c) Case 3: a is looking for the key 2. Node d is not in charge of the researched key. One of its sibling will forward.

Fig. 4. 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.

Routing in the physical network. In this section, we detail how we perform our hierarchical routing over our cluster topology by using a reactive routing protocol inside the clusters and a proactive routing protocol between the clusters. Such a proactive routing scheme 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. Algorithm 2 describes our hierarchical routing (We use This function is known by every node as the routing between clusters is proactive.).

Suppose node u needs to reach node v. If node u does not already know how to reach v, it uses the $hash$ function to learn $\mathcal{C}(v)$ before applying the routing rules. The routing process is illustrated on Figure 5. 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 $\mathcal{C}(w)$ (x such that $x \in \mathcal{C}(u)$ and $\exists y \in \Gamma_1(x) \cap \mathcal{C}(u)$). Note that $\mathcal{C}(u)$ and $\mathcal{C}(w)$ may be two neighboring clusters, in this case $\mathcal{C}(v) = \mathcal{C}(w)$. The message is thus 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 on 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 [15].

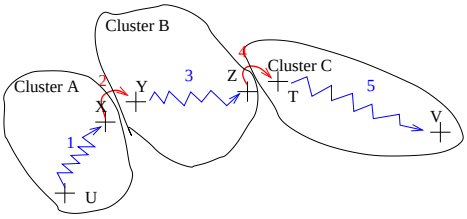


Fig. 5. Node u needs to communicate with node v. It uses successive reactive routing protocols to cross all clusters on its routes toward $\mathcal{C}(v)$ and then to reach node v.

Algorithm 2 Routing

For a message M **sent by node** $x \in \mathcal{C}(x)$ **to node** $y \in \mathcal{C}(y)$
  $\mathcal{C}_{current} = \mathcal{C}(x)$
  while $(\mathcal{C}_{next} \neq \mathcal{C}(y))$
    $\mathcal{C}_{next} = Next\_Hop(\mathcal{C}_{current}, \mathcal{C}(y))$
    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
  ▷ *The message has reached the cluster of the destination node.*
  Reactively route M toward destination node y.

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. 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). In our proposition, as, the routing process of the second step is performed over the physical topology with 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 soring 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 path 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.

We evaluated the length and the stretch factor of paths of our indirect routing first step by simulation. We used a simulator we developed. Nodes are randomly deployed using a Poisson point process in a 1×1 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. We compared the length of paths between two nodes of the cluster in the graph (physical topology) and in the tree (logical topology) to know the cost for routing in the tree. Results are shown in Figure 6. 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). 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 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.

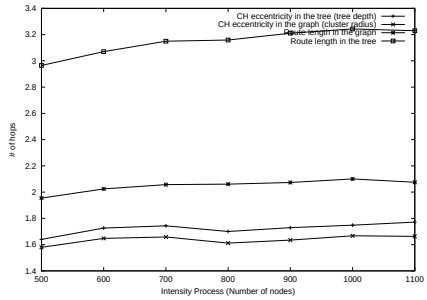


Fig. 6. Comparisons of distances in the logical and physical topologies.(CH eccentricity $+$: in the tree(tree depth) and $\times$: in the graph (cluster radius); Route length $*$:in the graph and $\square$: in the tree;)

5 Conclusion

In this paper, 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 needs $O(1)$ memory size and generates low traffic overhead and latency. It takes advantages of the underlying tree structure to perform an efficient Interval Routing within clusters. 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 to analyze deeper our algorithm concerning refreshing periods of node location registration and re-assignment of the partitions of the logical space over different trees. Then, we plan to compare our proposition to other existing ones as for instance SAFARI [21] which also uses this reverse approach for the hierarchical routing.

References

1. A. Amis, R. Prakash, T. Vuong, and D. Huynh. Max-Min d-cluster formation in wireless ad hoc networks. In *INFOCOM*, Tel-Aviv, Israel, 2000.

2. F. Araujo, L. Rodrigues, J. Kaiser, L. Changling, and C. Mitidieri. CHR: A Distributed Hash Table for Wireless Ad Hoc Networks. In *DEBS'05*, Columbus, USA, 2005.
3. L. Blazevic, S. Giordano, and J.-Y. Le Boudec. Self-organized Terminode routing. *Journal of Cluster Computing*, 5(2), April 2002.
4. B. Chen and R. Morris. L+: Scalable landmark routing and address lookup for multi-hop wireless networks. Mit lcs technical report 837, March 2002.
5. G. Chen, F. Garcia, J. Solano, and I. Stojmenovic. Connectivity-based k-hop clustering in wireless networks. In *HICSS'02*, Hawaii, USA, 2002.
6. Y. P. Chen, A. L. Liestman, and J. Liu. Clustering algorithms for ad hoc wireless networks. *Ad Hoc and Sensor Networks*, 2004.
7. Y. Fernandess and D. Malkhi. k-clustering in wireless ad hoc networks. In *POMC'02*, Toulouse, France, 2002.
8. P. Fraigniaud and P. Gauron. An overview of the content-addressable network D2B. In *PODC'03*, July 2003.
9. 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, April 1997.
10. J. Li, R. Morris, J. Jannotti, D. S. Decouto, and D. R. Karger. A scalable location service for geographic ad hoc routing. In *Mobicom'00*, pages 120 – 130, 2000.
11. C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7):1265–1275, 1997.
12. P. Maymounkov and D. Mazires. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS '02*, MIT Faculty Club, Cambridge, USA, 2002.
13. N. Mitton, A. Busson, and E. Fleury. Self-organization in large scale ad hoc networks. In *MED-HOC-NET 04*, Bodrum, Turkey, 2004.
14. N. Mitton and E. Fleury. Distributed node location in clustered multi-hop wireless networks. Technical Report In proceed, INRIA, 2005.
15. N. Mitton and E. Fleury. Efficient broadcasting in self-organizing multi-hop wireless network. In *Ad Hoc Now'05*, Cancun, Mexico, 2005.
16. N. Mitton, E. Fleury, I. Gurin-Lassous, and S. Tixeuil. Self-stabilization in self-organized multihop wireless networks. In *WWAN'05*, Columbus, USA, 2005.
17. E. T. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFOCOM*, New-York, USA, 2002.
18. D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *GLOBECOM'01*, 2001.
19. N. Nikaein, H. Labiod, and C. Bonnet. DDR-distributed dynamic routing algorithm for mobile ad hoc networks. In *MobiHoc*, Boston, USA, 2000.
20. C. Perkins. *Ad hoc networking.* Addison-Wesley, 2001.
21. 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. Technical Report TR04-433, Rice University, February 2005.
22. C. Santivanez, B. McDonald, I. Stavrakakis, and R. R. Ramanathan. On the scalability of ad hoc routing protocols. In *INFOCOM*, New-York, USA, 2002.
23. N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The computer Journal*, 28:5–8, 1985.
24. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *SIGCOMM'01)*, 2001.
25. J. Van Leeuven and R. Tan. Interval routing. *The computer Journal*, 30:298–307, 1987.
26. 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.
27. 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.