# Handling the M in MANet: an algorithm to identify stable groups of peers using crosslayering information

Hoa Ha Duong
Institut Telecom, Telecom ParisTech, CNRS
UMR 5141
46 rue Barrault, 75013, Paris, France
hoa.haduong@telecom-paristech.fr

Isabelle Demeure
Institut Telecom, Telecom ParisTech, CNRS,
UMR 5141
46 rue Barrault, 75013, Paris, France
isabelle.demeure@telecom-paristech.fr

## ABSTRACT

This paper proposes an algorithm to identify groups of users connected to a mobile ad hoc network that remain stable over time. Several similar algorithms have been proposed to manage mobility either by predicting disconnections or by identifying groups of peers stable over time. They all rely on information such as GPS, signal strength or routes and result in message overhead. The algorithm proposed here uses information from the routing layer to detect groups that are stable over time. This algorithm is fully distributed and creates no message overhead as the result of using cross-layer information.

## 1. INTRODUCTION

A MANet, or Mobile Ad Hoc Network, is a network established spontaneously between mobile terminals with wireless capacities, that do not require preexisting network infrastructures [5]. It therefore allows people in geographical proximity to collaborate without paying for network infrastructures, or when infrastructures are absent or down.

In this paper, we present an algorithm to build groups of peers stable over time in order to enable collaboration over a MANet. The proposed algorithm uses crosslayering information acquired from the routing layer, in our case OLSR [4]. It does not require specific equipment (such as GPS device), and does not generate traffic overhead.

This algorithm can be useful as part of CSCW[1] applications for MANet. When collaborative applications users go out of their usual working environment, such as researchers meeting at a conference, or kids on a field trip, they want to be able to work as they usually would. and keep sharing files, doing collaborative edition, editing wiki, etc. This is made easy if they use MANets because they do not require any preexisting network infrastructure.

However, when using MANets, mobility of terminals may cause network partition, when the nodes initially connected split in connected groups isolated from each other. This creates new issues for collaborative users, and especially for collaborative services, such as:

- A peer P has been elected to provide a service such as hosting an index of all the documents available in the network. This way finding a document requires a unique message exchange. What will then happen when the network is partitioned?
- A user wants to read a document stored on a distant terminal T. Should it be accessed once and then discarded, in which case the user would not be able to access it if T disappears, or should it be replicated, creating coherence issues and additional network traffic?

The proposed algorithm can be coupled with other algorithms to maintain data availability and data coherence in a nomadic context: proactive data replication may then be enforced within the stable groups. More about these other algorithms is described in [7].

Note that in this proposal, we focus on the use of MANets by pedestrians gathering in groups to collaborate. This context allows us to make hypothesis about mobility and traffic. Traffic and data accesses are human generated and therefore sporadic (as opposed to sensors). Groups of users are stable over time but may split and merge (for example as researchers go from one conference room to another) and users may come and go.

The proposed stable group building algorithm therefore tries to build stability over time rather than looking for proximity between terminals, although this parameter can also be taken into account.

A key issue is that the target terminals for our algorithm are mobile devices. Such devices are often battery operated and are therefore limited in energy. The two main sources of energy consumptions in a mobile device are the screen and the wifi card. The extent of use before having to reload is therefore correlated to the network load. Hence, any algorithm intended for MANet should try to limit its communication needs. This is addressed in our proposed algorithm that does not create message overhead as a result of using cross-

[1]Computer Supported Cooperative Work

layer information.

Several similar algorithms have been proposed to manage mobility either by predicting disconnections or by identifying groups of peers stable in time. Such propositions will be surveyed in next section.They all rely on information such as GPS, signal strength or routes and result in message overhead. The algorithm proposed here uses information from the routing layer to detect groups that are stable over time.

The remainder of this paper is organized as follows: we first present existing solutions and their context of use. We then present our proposal, illustrate it with a simple example and discuss the choice of a few parameters. In section 4 we evaluate our algorithm; a validation protocol is presented, and several scenarios are tested.

## 2. EXISTING SOLUTIONS

Several solutions have already been proposed for dealing with mobility. They can be classified in three categories: those that predict when a partition will occur; those that detect groups of terminals moving along; and finally those that create clusters of terminals in dense networks, with no management of terminals volatility.

These solutions rely on different technologies and techniques: the first one is the use of absolute coordinates, such as those used by GPS. This is the most precise assessment but it necessitates that peers exchange information; another technique is to use relative distances, for example based on signal strength (this is less precise and allows only to evaluate distance between peers within reach); another possibility is the use of higher level information, such as the routing graph or capacities of the terminals, for example the battery level. Some solutions, such as [17] aim at creating clusters of terminals in dense networks. This is a context where a lot of peers are connected and nodes are grouped by proximity, for example to produce a hierarchical routing algorithm. While an interesting approach, this does not consider the partitioning problem and is therefore of little interest in our context.

Older solutions aimed at creating groups to achieve hierarchical routing and therefore limit network traffic. In [13], Lin proposes a simple algorithm to build 2-hop wide clusters. Peers broadcast a list of their one-hop neighbours, on one hop, and the cluster ID is set to the ID of the peer with the smallest identifier among its neighbours. The stability of the algorithm is evaluated by counting the number of nodes changing cluster in a 100 ms interval. This solution is clearly not aimed at detecting human user groups, working together over periods of several minutes. In [3], Basu proposed a distributed 2-hops clustering algorithm by computing relative mobility between terminals based on the frequency of a beacon signal. It aggregates terminals by 2 hops wide clusters, where the clusterhead has the lower mean relative mobility.

Other works try to predict partition in order to adequately replicate services. In [14], Su proposes such an algorithm, based on GPS and an absolute dating system (it comes with GPS devices). Velocity vectors are exchanged and the duration of a link is computed based on communication range, velocity and positions, which is then used to predict partitioning. In [8], Hauspie proposes to detect partitions by computing the 'robustness' of a path between two nodes. This is done by checking for redundancy in the routing graph. When the robustness decreases, the communication may be interrupted and a partition occurs. This solution, if distributed, requires a full reconstruction of the graph, and would likely cause high traffic overhead. In [6], De Rosa, proposes to predict partitions and to prevent them: a coordinator centralizes distance information between nodes, computed by using signal strength. It then predicts future nodes position, and therefore possible partitions. The coordinator asks nodes to fill the possible gaps to ensure full connectivity. While the idea of moving peers is interesting and acceptable in some context such as rescue or military operations, it does not suit our problem.

Finally, some proposals sort the nodes in mobility groups. In [15] and [16], Wang proposes to build groups based on mobility to replicate services in each group. In [15], two algorithms are proposed to build groups of peers with similar velocity. The first one, detailed in [16], is centralized, and relies on a server to collect velocity vectors and cluster them. The second algorithm is therefore fully distributed: every node computes a mean distance to each of its neighbours, and a standard deviation over time, and uses those criteria to build groups. In [11], Huang proposes to cluster peers based on GPS information. Each terminal broadcasts to n-hops a list of positions with a timestamp. The peer with the smallest IP address among the messages received becomes a zone-master, and organizes nodes based on the position vector, by computing velocities. Since it is fully distributed, this solution generates traffic each time the clusters are updated.

In [18], Zheng proposed a distributed clustering algorithm using a positioning device (GPS). Nodes exchange their positions, and $\alpha$-stable clusters are constructed, where $\alpha$ is a probability of keeping the connectivity at t+1 between 2 peers, knowing the velocity vector and the position of each peer at t. In an $\alpha$-stable cluster, every pair of peers is $\alpha$-stable. While presenting good stability results, this solution does not evaluate the network overload.

The validation of [3] and [14] assumes a maximum speed of 72km/h. These solutions are therefore probably meant for vehicular networks. The proposals [14],

[11], [18], and possibly [15] rely on GPS.

Our solution is closer to this third class of algorithm: we postulate that users are moving in groups and we want to create groups of terminals stable over time. Our proposal uses cross-layering information, obtained from the routing tables, and therefore creates no network overhead. We only require a proactive routing algorithm that maintains the routes, so that we always know which peers are reachable at any time. In our evaluation we use the OLSR routing algorithm [4]. Our algorithm is fully distributed and does not require the use of a GPS device. Hence, it should be compared to proposal such as [3], [15] and [8].

# 3. PROPOSAL FOR STABLE GROUPS

In this section we introduce our proposal. First of all we present our working hypotheses, and we propose a definition for the notion of stable group. We then present the proposed algorithm, both with pseudocode and an example. Our algorithm relies on several parameters: the refreshing rate, the stability threshold, and the number of tolerated sporadic absences. We see how these values have been chosen.

## 3.1 Asumptions

In our proposal, we make the following hypotheses. First of all, we assume pedestrian users, moving in groups, that can be modelled with the Reference Point Group Mobility Model (see section 4.3.2). We also assume that communication are symmetrical, with a maximal range of communication of 100m (the maximal range of 802.11b outdoor). Finally, the routing algorithm is proactive and we use OLSR.

## 3.2 Stable group: definition

We define a stable group as a group of peers able to continuously communicate over time. In other words, to become stable neighbours at time t, A and B must have been in contact for at least $\delta_p$ seconds. If they cannot establish contact for $\delta_f$ seconds, they stop being stable neighbours.

Therefore, we consider that peer A is able to continually communicate with peer B around time t if:

- B receives all broadcast messages sent by A since t - $\delta_p$
- B receives all broadcast message sent by A between t and t+$\delta_f$

If these conditions are true, B is a stable neighbour of A. Since communications are symmetrical by hypothesis, if B is a stable neighbour of A, A is a stable neighbour of B.

We define a stable group around peer P as the set G of peers where each pair of peers in G are stable neighbours of P.

## 3.3 OLSR

Our work is part of the Transhumance?? research project, a middleware for MANet. As designing a routing protocol was not part of the project, we wanted a solid routing protocol implementation. We settled on OLSR because of its active community and the availability of the uniK implementation [2].

In our algorithm, we acquire information about which peers are in view by looking at OLSR routing tables.

OLSR is a proactive routing algorithm for mobile ad hoc networks. Proactive routing algorithms maintains routes by periodically sending messages to check if routes still exist, and offers low latency, to the cost of maintaining the routes. Reactive routing algorithms create routes on demand, by flooding the network. Both approaches have their strong points and drawbacks and [12] shows that for sporadic traffic, proactive algorithms are better suited. This clustering algorithm is a building block for human manned collaborative applications, thus generating sporadic traffic.

## 3.4 The algorithm

Figure 1 presents the pseudocode of our algorithm.

In this algorithm we use a function called getRoutesFromRoutingLayer. It returns an associative array including all the nodes in view and the number of hops to reach it.

In a nutshell, our algorithm is a two-phase algorithm that behaves in this fashion:

- Observation phase : when a peer $P$ comes into view, we create a counter, called *Presence Counter*, or *PC*, and set it to 1. Each period, we check if $P$ is present. If it is, *PC* is increased (line 28) and when *PC* gets over a stability threshold, $P$ rejoins the stable group (l. 34). Else, *PC* is decreased (l. 25), and if it reaches 0, we stop observing $P$ (l. 23).
  To become part of a stable group, $P$ must be present for at least (stability threshold) periods, plus the number of periods when it was absent.
- Stability phase : when $P$ is in our stable group, we check at each period if P is present. If it is, *PC* increases, up to a second threshold (l. 28), else, *PC* is decreased. When *PC* gets under the stability threshold (l. 35), $P$ gets our of the stable group.
  A number of consecutive sporadic absences are thus tolerated before it is withdrawn from the group.

## 3.5 An Example

Before explaining the way our algorithm is parameterized, we present its functioning with an example, where of stable threshold = 5 and maxAb=3. We consider the value associated to the name of a peer in

```
1  PERIOD=2
2  stableThreshold = 120
3  maxAbs = 60
4
5  class Peer :
6     def __init__(self, group, filename):
7         self.group = group
8         self.heardOf =dict()
9         self.stableNeighbourhood=[]
10
11 def updateHeardOf(self, routes) :
12    heardOfSet= set(self.heardOf.keys())
13    routesSet= set(routes.keys())
14    absent = heardOfSet-routesSet
15    stillpresent= routesSet & heardOfSet
16    newcomers = routesSet-heardOfSet
17
18    for p in newcomers :
19        self.heardOf[p]= 1
20
21    for p in absent :
22        if self.heardOf[p] == 1 :
23            del self.heardOf[p]
24        else:
25           self.heardOf[p]= self.heardOf[p] - 1
26
27    for p in stillpresent :
28        if self.heardOf[p]<stableThreshold+maxAbs :
29            self.heardOf[p]=self.heardOf[p]+1
30
31 def buildStableGroup(self):
32    self.stableNeighbourhood=[]
33    for p in self.heardOf.keys():
34        if self.heardOf[p]>stableThreshold :
35            self.stableNeighbourhood.append(p)
36
37 def buildStableGroup(self) :
38    while true:
39        routes = getRoutesFromRoutingLayer()
40        self.updateHeardOf(routes)
41        self.buildStableGroup()
42        sleep(PERIOD)
43
```

**Figure 1: An algorithm to build stable groups**

*heardOf* to be its presence counter. In this table we see the evolution of the presence counter, PC, associated to peer P by peer A over time and the consequences on the stable group.

1. at t=0, A had never heard of P.
2. from t=0 to t=2, PC increases.
3. at t=3, P is absent so PC is decreased.
4. from t=4 to t=6 PC increases ; at t=6, P becomes part of peer A's stable group. We can see that to become part of a stable group, a peer has to be more present than absent.
5. from t=6 to t=7, PC increases.
6. at t=8, P is not reachable so its PC is decreased; it stays in the stable group : a sporadic absence can be tolerated.
7. from t=9 to t= 13, PC is increased to *stableThreshold+maxAbs*, and then stabilizes.
8. at t=15, P disappears; its PC is decreased but it stays in the stable group.
9. from t=14 to t=16, PC decreases.
10. at t=18, PC goes below *stableThreshold*, and so after *maxAbs*, it is withdrawn from the stable group.

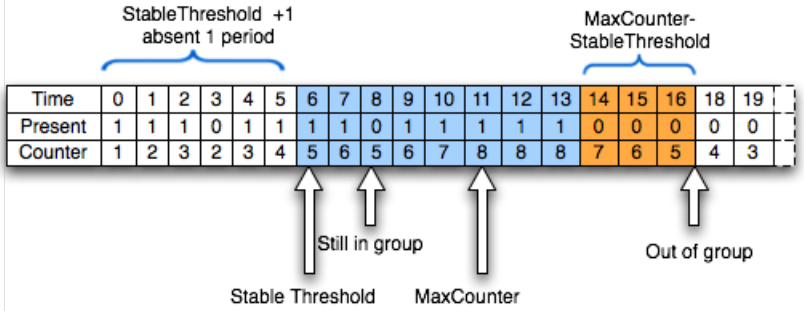


**Figure 2: An example**

## 3.6 Algorithm parameters

In this algorithm, three parameters can be adapted: The refreshing rate, represented by the PERIOD parameter (lines 1, 42) ; The length of the observation phase, represented by the stableThreshold parameter (lines 28,34) ; The number of absences tolerated in the stability phase ; represented by the parameter maxAbs (line 34).

In this section we discuss their significance, and how we chose their values.

### *3.6.1 Stable group refresh period*

Our algorithm periodically builds a stable group: it tries to predict which peers will be present between the present time, and the next time it is executed, based on past experience. This must be done frequently enough that the peers in the stable groups are always in reach between two refresh.

Since HELLO messages are sent every 2s, as proposed in the OLRS RFC [4], the routing tables are refreshed every 2 seconds. Therefore, a smaller period would not yield more information than a period of 2s. Therefore we decided on PERIOD_SEC=2s, and in this section, we evaluate this choice.

Consider a peer A in our stable group at time t. We would like to bound the probability that until the next refresh of the group, the peer is actually reachable. At t+2, the peer must still be seen.

We consider a communication range between $r_{min} = 30m$, and $r_{max} = 100m$. The peer velocity is v= 1ms. If two peers are walking in opposite directions, the distance between them will increase by $2 * period * v = 4m$. Therefore, as illustrated by fig. 3, if the peer is in the grey area at t, we are certain that we will still see the peer at t+Period. Therefore, to bound the probability that the peer is still seen at t+period, we compute

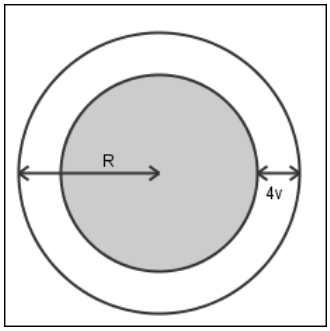


**Figure 3: Bounding the algorithm period**

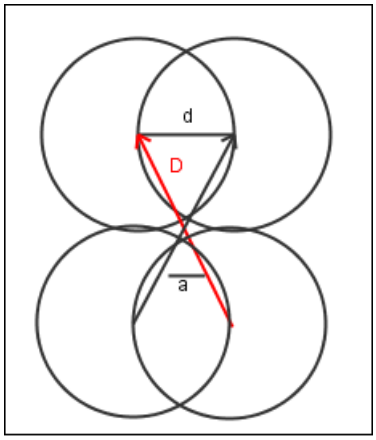


**Figure 4: Two groups crossing at angle a**

the probability for the peer to be in the grey area at t, knowing that it can be seen.

$$p_{see\ at\ t+2} = \frac{grey\ area}{whole\ area} = \frac{\pi * (R-4v)^2}{\pi * R^2} = (\frac{R-4v}{R})^2$$

For $r_{max}$, $p_{see\ at\ t+2} \approxeq 92\%$. For $r_{min}$, $p_{see\ at\ t+2} \approxeq 75\%$. For a mean value of r=65, $p_{see\ at\ t+2} \approxeq 88\%$.

Note that those values are lower bounds. Even if a peer is in the grey area at t, it will not automatically be out of reach at t+1 : it depends on its direction and speed.

### 3.6.2 Stability Threshold

The stability threshold parameter indicates for how many periods we expect a peer to be seen before it becomes part of our stable neighbourhood.

In the group definition given in section 3.2, this is an approximation of $\delta_p$, the number of seconds peers must continuously communicate before becoming stable neighbours. With our algorithm, two peers become stable neighbours after *(stableThreshold+the number of communication drops)* seconds. Therefore, the time between two peers making contact, and two peers becoming stable neighbours varies and is equal or superior to stableThreshold.

We want to set this threshold low enough that groups can be quickly formed, but high enough that our algorithm is able to discriminate between two groups crossing path and groups merging.

The time two groups crossing path stay in touch depends on the angle $a$ between their trajectories, as illustrated in fig. 4. In the best case, groups come from opposite direction. In the worst case, groups cross at a very small angle and their trajectories seem aligned.

We choose this threshold so that groups crossing at right angle ( 90°, or $\frac{\pi}{2}$ rad) will not be mixed.

We want to compute the time span $T$ during which

| Angle | 80 | 70 | 60 | 50 |
|---|---|---|---|---|
| Time | 1m2 | 2m17 | 4m40 | 5m53 |
| Angle | 40 | 30 | 20 | 10 |
| Time | 7m44 | 10m52 | 17min11 | 36m14 |

**Figure 5: Time to correct discrepancy, depending on the angle, in degrees**

two groups crossing at angle $\alpha$, with a maximum radius of d (this will be explain further in section 4.3.2, where we present our mobility model), will stay in contact. As illustrated by figure 4, $T$ is the time taken to walk D.

Hence, $T$ can be computed as:

$$\frac{0.5 * d}{0.5 * D} = sin(0,5 * \alpha) \implies D = \frac{d}{sin(0,5 * \alpha)}$$

as v=1m/s, T = d$\frac{}{sin(0,5*\alpha)}$

For $\alpha = 90°$, d=200, D = 282m. Since the velocity is 1m/s, it takes 282s between the time the two groups see each other and the time the two groups split. Since the algorithm is executed every 2 seconds, we set the stableThreshold to 140. Therefore, we consider a peer as part of our group if it is seen for at least 4 minutes 40.

With the stability threshold set to 140, the algorithm can distinguish between two groups crossing at an angle $\alpha = 90°$ or less. If the angle is smaller, our algorithm will momentarily detect one group instead of two before correcting itself.

For $\alpha < 90°$, we can compute $error\_span_\alpha$, a theoretical value for the time during which our algorithm erroneously detects one group instead of two.

At $t_{deb}$=0, groups make contact and the presence counter PC starts to increase. At $t_{grp}$=2*stableThreshold = 280, the peers are grouped. At $t_{end}$ =D, PC starts to decrease; at that point, we have two cases.

If $D > 2 * (stableThreshold + maxAbs)$ :

- PC =stableThreshold+maxAbs
- Mistake is corrected at $t = t_{end} + 2 * maxAbs$ sec.
- The algorithm was incorrect for
  $error\_span = t_{end} + 2 * maxAbs - t_{grp}$
  $error\_span = D + 2 * maxAbs - 280$

Else:

- $PC = \frac{D}{2}$
- Mistake corrected at $t = t_{end}+2*(\frac{D}{2}-stableThreshold)$
- The algorithm was incorrect for:
  $error\_span = t_{end}+2*(\frac{D}{2}-stableThreshold)-t_{grp}$
  $error\_span = 2 * D - 560$

Table 5 presents a few results for different angles.

Note that if $\alpha$ is small enough, the groups would appear to have identical trajectories to a human observer, who may think for a while that they, in fact, have merged. Our algorithm is misled for the same reason.

#### 3.6.3 Number of absences tolerated

The *maxAbs* parameter indicates for how many period at most we allow a peer in the stable group to be absent before it is removed from the stable group.

In the group definition given in section 3.2, this is an approximation of $\delta_f$, the number of seconds stable neighbours should be able to communicate. When two stable neighbours lose contact, their presence counter is comprised between *stableThreshold* and *stableThreshold+maxAbs*. It is steadily decreased until it gets under *stableThreshold* and the two peers stop being stable neighbours. Therefore, the time to determine that two peers are not stable neighbours anymore after they stop communicating is less or equal to *maxAbs*.

This parameter is used in two cases :

- When a peer part of our stable group leaves, we want it to be withdrawn from the stable group as soon as possible. The same problem arises when a group splits in new groups.
- When a peer part of our stable group is absent for a small period, we want our algorithm to tolerate this absence and keep it in the stable group, so *maxAbs* should be high enough.

The number or frequency of peers leaving have no influence on the choice of *maxAbs* value, except that it should be low enough.

The absences on the other hand do. For example, if we could establish that peers are never sporadically absent for more than 30 seconds, we could set *maxAbs* to 15. The frequency of absences can be influence by different parameters:

- a temporary obstacle, such as a wall, between two peers
- the network load creates a loss of information at OLSR level
- the terminal freezes for a few seconds

Since we have no control over those situations, and no way to predict those parameters, we chose a value for *maxAbs* based on the group definition: if a peer in my stable group at t is absent for more than 2 minutes, it should be withdrawn from the group.

Hence, maxAbs=60.

# 4. EVALUATION

In this section we present our algorithm evaluation.

First of all, we see how it compares to others in term of complexity. Then, we run it on a few typical scenarios, to see how it behaves. Since testing on MANets is difficult, we did this part on a simulator.

## 4.1 Usual metrics : distributed algorithm complexity

Since it uses information acquired from the routing tables, our algorithm do not need to exchange overhead messages to establish topological information. It is, in this respect, better than any other propositions.

Since each peer computes its neighbourhood independently, the algorithm is fully distributed and presents no centralization bottleneck. Therefore, our algorithm scales as much as the routing algorithm does.

We need another metric to evaluate the performance of our algorithm.

## 4.2 Proposition: an accuracy metric

Different groups are evolving in an area. Suppose an oracle who can set apart groups based on distance and motions, and creates ideal groups.

To computer the accuracy of our algorithm, we compare the composition of the *ideal* group, to that of the *computed* group:

$$accuracy(t) = \frac{|ideal|}{|ideal \cap computed(t)|}$$

Note that $|ideal \cap computed(t)|$ is never null, since both sets contain the peer itself.

This computes a value in $\mathbb{N}$, and an accurate algorithm keeps this metric to close to 1.

## 4.3 Evaluation by simulation

MANets are inconvenient to deploy: since each device is mobile, we would need human or robot operators able to move and reproduce mobility at will. They also are inconvenient for reproductibility: wireless communications may be perturbed by external signals upon which we have no control, and two experimentations with strictly identical mobility patterns may produce different results, unless we have a vast isolated space for experimentation.

Hence, we have validated our algorithm by simulation.

#### 4.3.1 Modus operandi

To validate our algorithm by simulation, we designed and used several tools:

1. We generate mobility traces compliant to the RPGM model (cf section 4.3.2).
2. These traces are then injected in the ns-3 simulator[9], configured with wifi 802.11b[1], and OLSR.
3. Within ns3, we dump the routing tables every 2 seconds on the disk.
4. We run our algorithm using this tables with different values for each parameter.
5. For each run of our algorithm, we compute an aggregated accuracy overtime.

#### 4.3.2 Mobility model

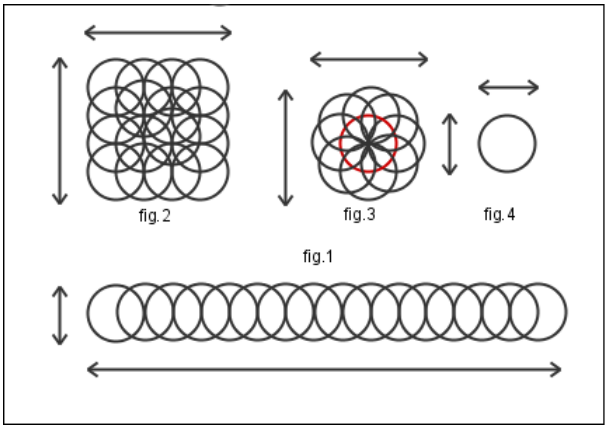


**Figure 6: Possible network configurations**

To generate mobility traces, we used the Reference Point Group Mobility (RPGM) Model [10].

In RPGM, nodes are organized in groups. Each group has a reference point, that is a logical centre : the reference point (RP) moves with a Random Waypoint pattern and other nodes move to stay within range of the reference point.

Since the upper bound of the communication range of IEEE 802.11b in open space is 100m, and since each peer is placed within communication range of RP, two peers will never be more than 200m apart. Therefore, the maximal radius of an RPGM group is 200m.

#### 4.3.3 *Node density, network topology, and influence on the accuracy measurement*

In a small enough simulated area, groups will interact even if we try to keep them apart because of the nodes density. Therefore, if the simulated area size is less than the group area size by the number of groups, the node density is too high to distinguish all the groups.

We have to calibrate our tests so that the accuracy is not perturbed by the closeness of the groups.

To do so we need to compare two values:

- S = the surface area of the simulated area;
- NG*GS = the number of groups * the maximal area covered by a group.

The maximal area GS covered by a group depends on its topology. For example, as seen in figure 6, the group covers most ground when it is organized as a line.

In the RPGM model, peers are organized so that each peer is within reach of the reference point. This corresponds to a network configuration as in fig. 6. A group should therefore occupy a circular area of 200m of radius.

In the subsequent scenario, we chose the size of the simulated area so that groups can be isolated.

### 4.4 Test scenarios

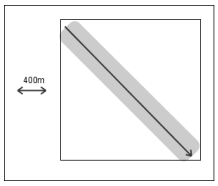


**Figure 7: 1 group**

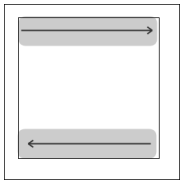

**Figure 8: 2 distinct groups**

In this section we examine how our algorithm behaves in characteristic situations. We see how it behave with no groups interfering, and what happens when two groups cross paths, merge, and when one group splits in two subgroups.

For the simulations presented below, the simulated area size is 2000mx2000m, and each group is made up of 10 peers. The mobility model is RPGM.

If not indicated otherwise, the *stable threshold* varies between 100 and 180 . The maximum of authorized periods of unavailability varies between 20 and 100.

For each case, we present the scenario, namely which mobility pattern, and which parameters are tested, and the theoretical result. Results are presented as graphs representing the evolution of the algorithm accuracy over time.

#### 4.4.1 *One group*

In this scenario, illustrated by fig.7, we want to verify the behaviour of our algorithm in a simple situation with one group of peers.

**Scenario:** A group of peers is walking from the upper left corner to the opposite corner. We want to test if our algorithm work when no disruption occurs, and which value of *stableThreshold* maximizes, in that case, the accuracy; *stableThreshold* varies between 100 and 180.

**Expected result:** We expect the computed group not to be accurate up to *stableThreshold* seconds, to stay accurate after. The lowest value of *stableThreshold* should maximize the accuracy.

**Observation:** Graph 9 represents evolution of the accuracy over time. Each curve represents the accuracy for a given value of *stableThreshold*. As we can see, the simulation validates our expected results.

#### 4.4.2 *Two groups with no interaction*

In this scenario, illustrated by fig.8, we want to verify that having two groups with no interaction does not alter the computation accuracy.

**Scenario:** Two groups follow opposite borders of the area. They never come in contact, and we want to verify that the algorithm accuracy behaves as it does for one group. The *stable threshold* varies between 100 and 180.

**Expected result:** As groups are not interacting, we

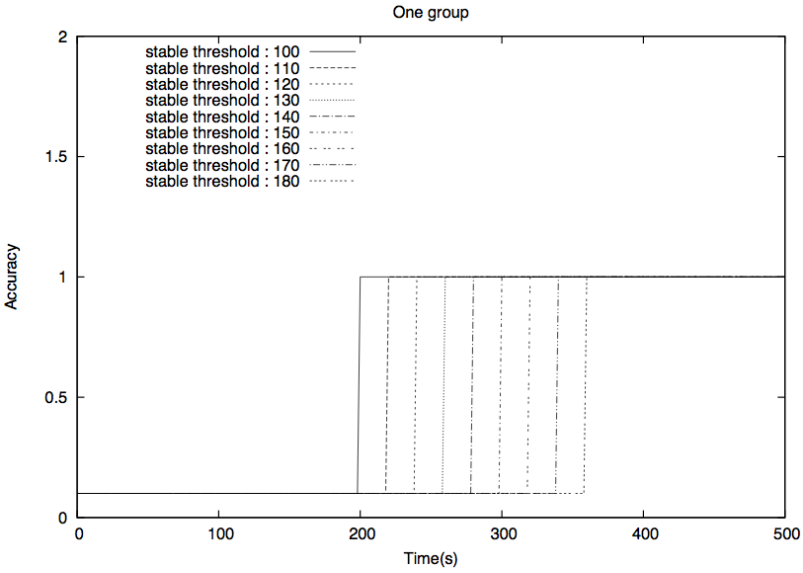


**Figure 9: One group, $\neq$ stable threshold**

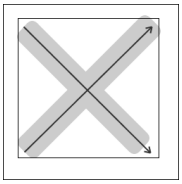

**Figure 10: Paths crossing**

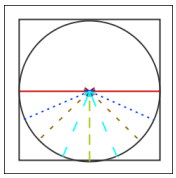

**Figure 11: Same distance, different angles**

expect the same curve as in the previous experiment.

**Observation:** Results are similar to fig. 9.

### *4.4.3 Two groups, crossing path*

In this scenario, illustrated by figure 10, we examine the behaviour of our algorithm when two distinct groups are crossing paths.

**Scenario 1:** Two groups start out of reach of each other, at upper left, and bottom left corners; both groups walk diagonally to reach the opposite corner of the area; at t= 1415 seconds, they cross path at right angle, at (0,0). Around t=1415, we want to verify if for the chosen value of *stableThreshold*, our algorithm discriminates between the two groups. The *stable threshold* varies between 100 and 180.

**Expected result 1:** We examine the behaviour of our algorithm around t= 1415. For a *stable threshold* under 140, no errors should be detected; as the stable threshold decreases, the results should worsen.

**Observation 1:** Graph 12 show that results are better than expected : the two groups stay distinct for a stable threshold under 120.

**Scenario 2:** Two groups start out of reach from each other and at t= 1415 sec, they cross path at (0,0); the

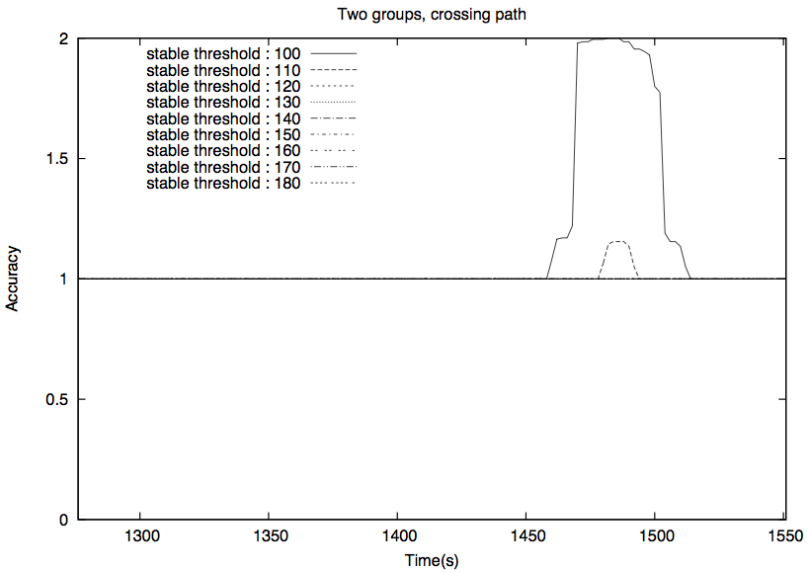


**Figure 12: 2 groups crossing, $\neq$ stable thresholds**

angle $\alpha$ formed by their trajectories varies between 0 rad and $\pi$ rad; their starting positions are chosen as illustrated by fig.11 so that all the groups cross paths at time t=1000. We want to verifie that the algorithm discriminates between groups crossing at angle larger than or equal to $\frac{\pi}{2}$ rad. The *stable threshold* is fixed at 140.

**Expected result 2:** If two groups are walking with a 0 rad angle, they effectively behave as one group, so in this case the accuracy should be 2 from t=1415 sec and on. For other values of $\alpha$, the errors should decrease as $\alpha$ grows, and for $\alpha > \frac{\pi}{2}$, no error should be detected.

**Observations 2:** In figure 4.4.3, each curve represents the evolution of the accuracy for a given angle. We can see that our prediction for $\alpha = 0$ and $\alpha > \frac{\pi}{2}$ is correct. The correctness of the algorithm is also increasing as $\alpha$ grows. We can also see that the recovery time for $\alpha < 90°$ is lower than expected. For example for $\alpha = 45°$, the error is 4m40, lower than the theoretical time of recovery for $\alpha = 50°$, 5m53, as computed in 3.6.2.

### *4.4.4 Two groups, fusion*

In this scenario, illustrated by figure 14 we measure how our algorithm behaves when two groups merge.

**Scenario:** Two groups start at the two bottom corners of the area; they both walk at right angle to the middle of the area, taking 1415 seconds; the groups merge and walk straight to the top. We want to verify that the algorithm detects the merging, and the time it takes depending on the *stability threshold*. The *stable threshold* varies between 100 and 180.

**Expected result:** We examine the behaviour of the grouping algorithm around t=566. The lesser the *stable*

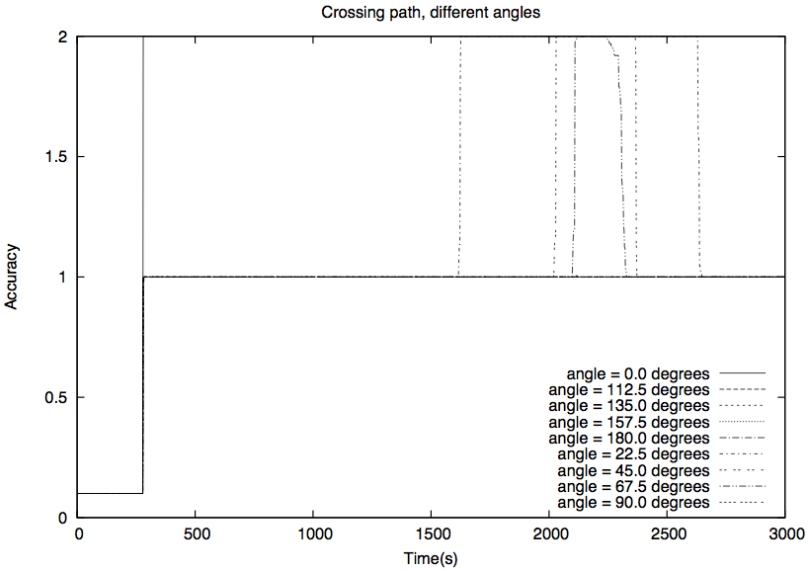


**Figure 13: 2 groups crossing, $\neq$ angles**

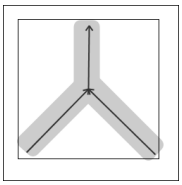

**Figure 14: Merging**

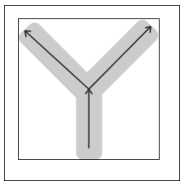

**Figure 15: Split**

*threshold* is, the faster groups are formed and the better the accuracy.

**Observation:** As expected, we can see on figure 16, that the algorithm recovers better with a lower *stable threshold*. Note that as trajectories are perpendicular, the peers starts communicating 142 sec before the merge. With a *stable threshold* of 70, no error would occur.

This experiment is a generalization of the case of a peer joining a group and therefore validates it.

### 4.4.5 *One group, splitting*

In this scenario, illustrated by figure 4.4.3, we measure how our algorithm detects a group splitting.

**Scenario:** The group starts at (0,1000) and walks to the center, taking 1000 sec; there, it splits in two; one half walks to the upper left corner while the other one walks to the upper right corner. We want to verify that the algorithm detects the split, and the time it takes depending on the maximum absences tolerated. The *stable threshold* is fixed to 140 and *maxAbs* varies between 20 and 90.

**Expected result:** We examine the behaviour of the group algorithm around t = 2000. In this scenario, the

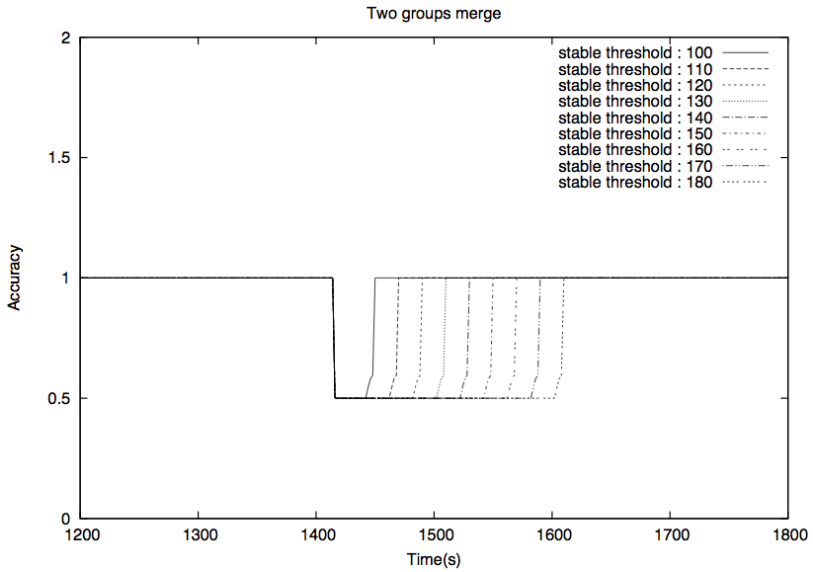


**Figure 16: Two group merge, $\neq$ stable thresholds**

algorithm cannot always be accurate, even with *maxAbs*=0, since when the groups split at t=1000, they can still communicate for 224 sec. Still, we expect that the lower *maxAbs* is, the quicker our algorithm will correct the groups.

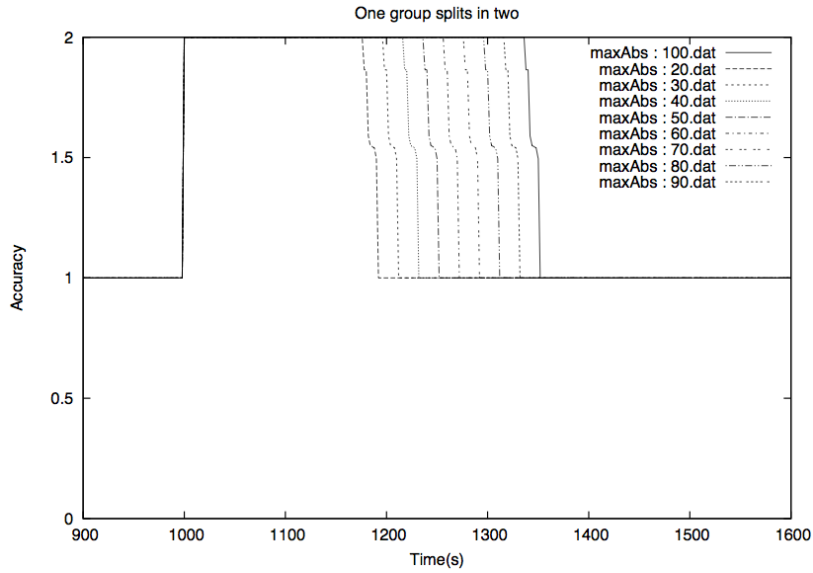


**Figure 17: One group splits, $\neq$ maximum absence**

**Observations:** In figure 17, each curve represents the evolution of the accuracy over time for a given value of *maxAbs*. It validates the expected result.

Note that this experiment is a generalization for the case of a peer leaving a group and validates it.

#### 4.4.6 Overall observations

Experiments 4.4.1, 4.4.2 and 4.4.4 show that the lower the *stability threshold* is, the quicker a stable group can be formed, while experiment 4.4.3 shows that if the *stability threshold* is too low, the algorithm cannot differentiate between a group, and two groups crossing path.

Experiment 4.4.5 shows that the lowest *maxAbs* allows a better detection of group splitting. However, while the experiment is not shown here due to the lack of space, *maxAbs* allows for tolerating transient faults and therefore should not be 0.

Overall, these experiments validate the expected behaviour of our algorithm, and in the case of two groups crossing, shows even better results than expected.

## 5. CONCLUSION

In this paper we proposed an algorithm to build groups stable over time. This algorithm relies on cross-layering information, namely routing information maintained by a proactive routing algorithm, to establish which other peers are reachable for long enough to be considered stable neighbours.

The two strong points of this algorithm, compared to existing proposal, are the lack of need for a positioning equipment, such as GPS, and the lack of network overhead. Both are sources of energy use; energy is a critical resource in a MANet context where terminals are mostly handheld and battery-operated.

We presented an evaluation of our algorithm, with a theoretical model and by simulation.

In term of distributed complexity, our algorithm is better than any existing proposal as it does not create network overload. We also validated our theoretical model by simulation.

Further works would be to implement existing proposals to test them again our accuracy metric, in order to provide further comparison.

## 6. REFERENCES


[1] http://standards.ieee.org/getieee802/802.11.html.

[2] http://www.olsr.org/.

[3] P. Basu, N. Khan, and T. D. Little. A mobility based metric for clustering in mobile ad hoc networks. In *In International Workshop on Wireless Networks and Mobile Computing (WNMC2001*, pages 413–418, 2001.

[4] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), Oct. 2003.

[5] S. Corson and J. Macker. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501 (Informational), Jan. 1999.

[6] F. De Rosa, A. Malizia, and M. Mecella. Disconnection prediction in mobile ad hoc networks for supporting cooperative work. *IEEE Pervasive Computing*, 4(3):62–70, 2005.

[7] H. H. Duong and I. M. Demeure. Proactive data replication using semantic information within mobility groups in manet. In J.-M. Bonnin, C. Giannelli, and T. Magedanz, editors, *MOBILWARE*, volume 7 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 129–143. Springer, 2009.

[8] M. Hauspie, D. Simplot, and J. Carle. Partition detection in mobile ad hoc networks, 2003.

[9] T. Henderson, , T. R. Henderson, and S. Roy. ns-3 project goals.

[10] X. Hong, M. Gerla, G. Pei, and C.-C. Chiang. A group mobility model for ad hoc wireless networks. In *MSWiM '99: Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 53–60, New York, NY, USA, 1999. ACM.

[11] J.-L. Huang, M.-S. Chen, and W.-C. Peng. Exploring group mobility for replica data allocation in a mobile environment. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 161–168, New York, NY, USA, 2003. ACM.

[12] A. Huhtonen. Comparing AODV and OLSR routing protocols, May 27 2004.

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

[14] W. Su, S.-J. Lee, and M. Gerla. Mobility prediction and routing in ad hoc wireless networks. *Int. J. Netw. Manag.*, 11(1):3–30, 2001.

[15] K. H. Wang and B. Li. Efficient and guaranteed service coverage in partitionable mobile ad-hoc networks, 2002.

[16] K. H. Wang and B. Li. Group mobility and partition prediction in wireless ad-hoc networks, May 29 2002.

[17] H. Yu, H. Hassanein, and P. Martin. Cluster-based replication for large-scale mobile ad-hoc networks. In *International Conference on Wireless Networks, Communications in Computing*, pages 552–557, June 2005.

[18] J. Zheng, J. Su, and X. Lu. A clustering-based data replication algorithm in mobile ad hoc networks for improving data availability. In *Proceedings of 2nd International Symposium on Parallel and Distributed Processing and Applications (ISPA 2004)*, pages 399–409, 2004.