Computer Networks 52 (2008) 1797–1824

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

# Clustering in mobile ad hoc networks through neighborhood stability-based mobility prediction ☆

Charalampos Konstantopoulos [a,*], Damianos Gavalas [b], Grammati Pantziou [c]

[a] Research Academic Computer Technology Institute and Department of Computer Engineering and Informatics, University of Patras, Greece
[b] Department of Cultural Technology and Communication, University of the Aegean, Mytilene, Greece
[c] Department of Informatics, Technological Educational Institution of Athens, Athens, Greece

ARTICLE INFO

*Article history:*
Received 17 March 2007
Received in revised form 3 December 2007
Accepted 30 January 2008
Available online 16 March 2008

Responsible Editor: V.R. Syrotiuk

*Keywords:*
Mobile ad hoc networks
Clustering
Routing
Information theory
Distributed algorithms

ABSTRACT

Clustering for mobile ad hoc networks (MANETs) offers a kind of hierarchical organization by partitioning mobile hosts into disjoint groups of hosts (clusters). However, the problem of changing topology is recurring and the main challenge in this technique is to build stable clusters despite the host mobility. In this paper, we present a novel clustering algorithm, which guarantees longer lifetime of the clustering structure in comparison to other techniques proposed in the literature. The basis of our algorithm is a scheme that accurately predicts the mobility of each mobile host based on the stability of its neighborhood (i.e., how different is its neighborhood over time). This information is then used for creating each cluster from hosts that will remain neighbors for sufficiently long time, ensuring the formation of clusters that are highly resistant to host mobility. For estimating the future host mobility, we use provably good information theoretic techniques, which allow on-line learning of a reliable probabilistic model for the existing host mobility.



## 1. Introduction

Clustering [1] is a promising approach for enhancing the scalability of mobile ad hoc networks (MANETs) in the face of frequent topology changes mainly due to the host mobility. Clustering not only makes a large MANET to appear smaller, but more importantly, it makes a highly dynamic topology to appear less dynamic [3]. In clustering, a representative of each cluster is 'elected' as a cluster head (CH) and a mobile host (MH), which serves as intermediate for inter-cluster communication, is called a gateway. Remaining members are called ordinary MHs. CHs hold routing and topology information while the boundaries of a cluster are defined by the transmission area of its CH.

The feasibility of a clustering method is determined by the stability of the cluster structure that it creates, despite network topology changes. Otherwise, frequent reclustering is required thereby creating a large volume of control messages which in turn consume considerable bandwidth and drain MHs' energy quickly. As the main cause for topology changes in MANET is the host mobility, an efficient clustering method should seriously take the movements of MHs into account in order to form clustering structures resistant to the host mobility.

☆ This work is co-funded by 75% from EU and 25% from the Greek Government under the framework of the Education and Initial Vocational Training II, programme "Archimedes".

* Corresponding author. Tel.: +30 210 5385312; fax: +30 210 5910975.
*E-mail address:* konstant@cti.gr (C. Konstantopoulos).

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.01.018

Many researchers [3–6,9–12] have acknowledged the importance of host mobility estimation for building clustering schemes more stable and less reactive to topological changes of ad hoc networks. Authors in [3] propose the $(a, t)$ clustering scheme, where MHs form clusters according to a path availability criterion. The network is partitioned into clusters of MHs, that are mutually reachable along cluster internal paths which are expected to be available for a period of time $t$ with a probability of at least $a$. The parameters of this model are predefined. In addition, it is assumed that the movement of each MH is random and entirely independent of the movements of other MHs. However, this random walk model cannot always capture some host mobility patterns occurring in practice in MANETs.

MOBIC in [4] elects as CHs the MHs which exhibit the lowest mobility in their neighborhood. Each MH compares the receiving signal strength from its neighbors over the time and uses the variance in these values as an indication of how fast this MH is moving in relation to the neighboring MHs. MOBIC uses only the current mobility to determine the most suitable MHs for CHs. As an extension of MOBIC, MobDHop [5,6] also uses the variability in receiving signal strength as a hint of neighborhood mobility and builds variable-diameter clusters. It uses more samples of receiving signal than MOBIC to estimate the predicted mobility but again the prediction model is rather simple since it is based on the assumption that the future mobility patterns of MHs will be exactly the same as those of the recent past.

MAPLE [7] is another clustering algorithm which also infers host mobility from measurements of the received signal strength. In particular, each MH belonging to a cluster can estimate its distance from its CH by using the well-known formula [8] of the signal attenuation versus travelled distance. Then, based on past measurements, MHs use a linear model for estimating their future distance from their CH. This helps MHs proactively join another cluster if they are going to leave their current cluster. However, MAPLE does not take host mobility into account during CH election. Specifically, MHs contend for free frames (i.e. time slots) in a single shared broadcast channel during the cluster formation phase and MHs that first reserve the available frames in this phase become CHs. Thus, the election of CHs is mostly a random procedure and it is not based on some CH suitability criteria. In addition, the algorithm sets an upper bound on the number of clusters in the network as well as on the number of MHs per cluster.

DMAC in [9] and GDMAC in [10] proposed by Basagni are generic weight-based clustering schemes, where MHs with the highest weight among neighboring ones are elected as CHs. Basagni suggested to use the inverse of the speed of MHs as a weight in its scheme. WCA in [11] is also a weight-based clustering technique which extends the work in [9,10]. The weight in this scheme is determined by considering various factors that affect the suitability of a MH as a CH. Among these factors is host mobility. Specifically, each MH measures its average speed by sampling its position coordinates at regular time intervals. This method of measurement requires the use of a GPS device on each MH, which is not always feasible. Furthermore, this method fails to capture the correlation that may exist among the movements of neighboring MHs as in the case of group movement.

Information theory-based techniques for host mobility prediction have been first employed in [14], where the authors focused on the problem of mobile tracking and localization in cellular networks. Later, Sivavakeesar et al. [12] used the basic technique of [14] in their cluster formation algorithm for MANETs. A basic assumption in their work is that a geographical area is divided into circular-shaped regions named virtual clusters and each MH knows the virtual cluster where it is currently in. So, the ad hoc network in their technique is very much like a cellular one and the ideas in [14] can be applied.

In this paper, we propose a novel mobility-aware technique for cluster formation and maintenance. The main idea in our technique is to estimate the future mobility of MHs so as to select CHs that will exhibit the lowest predicted mobility in comparison to the other MHs. As a measure of host mobility rate, we use the probability of a MH having the same MHs in its neighborhood for sufficiently long time. A high probability value for a MH indicates a relatively immobile host or the existence of a group of MHs around this particular MH that exhibits the same mobility pattern. Whatever the case is, this MH is apparently a good candidate for a CH, because in all probability, it will serve the same neighbors for a long time. For estimating the predicted mobility of a MH, we make the realistic assumption for most MANETs that the movements of MHs are not random but demonstrate a regular pattern [15], which can be predicted provided that enough "historic" information has been gathered for the movements of each MH.

For the organization of the historic record and the estimation of future mobility based on this record, we borrow prediction techniques from the field of data compression. Specifically, we reduce the problem of estimating the future neighborhoods of a MH to one of predicting the next characters in a text given that we have already seen a particular text context. Then, by using context-modelling techniques [16], we can reliably estimate the probability of stable neighborhood around a MH. The most important characteristic of these methods is the on-line learning of the probability model which these methods use for prediction of the next character/neighborhood. This is essential in the case of ad hoc networks because the movements of individual MHs as well as the strong correlation that exists in the movements of these hosts cannot be easily described by predefined random models.

Note also that we do not make any use of a fixed geographical partition in contrast to previous work [14,12] and thus the notion of cells is irrelevant to our technique.

Besides the stability of the clustering structure, an important objective in cluster creation is to keep the number of elected CHs relatively low so that the virtual backbone built over these MHs will be of correspondingly small-size and hence routing update protocols could be efficiently ran on this backbone. The well-known highest connectivity (degree) algorithm [2] promises the election of relatively few CHs. In this paper, we propose a new clustering algorithm named MobHiD, which combines the highest degree technique with our mobility prediction scheme and ensures a relatively small as well as stable

virtual backbone despite host mobility. The performance of our technique was verified via simulation experiments, which compared our algorithm with other competitive techniques of the literature.

Note that our mobility prediction technique is of independent interest and may be combined with other clustering algorithms to enhance the stability of the derived clustering structure in the presence of frequent topology changes.

The paper is organized as follows. In Section 2, we discuss our mobility prediction method. In Section 3, we present our MobHiD clustering algorithm which uses the mobility prediction method. Section 4 addresses the details of the distributed implementation of MobHiD and then Section 5 discusses the simulation results about the performance of our clustering technique. Finally, Section 6 concludes the paper by summarizing the main contribution of our work.

Preliminary results of this work have been presented in [20].

## 2. Our mobility prediction method

A MH is considered a good candidate for CH if its neighborhood is relatively stable in comparison to the neighborhoods of other candidate hosts. Let $ngh_{i,t} = \{h_0, h_1, h_2, \ldots, h_{n_t-1}\}$ be the $n_t$ neighboring MHs of MH $i$ at time step $t$. Somehow, we have to estimate the probability of the stability of this neighborhood, i.e., the probability $P(ngh_{i,t})$ that this neighborhood will remain the same in the following time steps, if possible, forever. By making the simplified assumption that the presence of a MH among the neighbors of MH $i$ in the future is independent of the presence of any other host, we can equivalently write the probability $P(ngh_{i,t})$ as follows:

$$P(ngh_{i,t}) = P(h_0)P(h_1)\cdots P(h_{n_{t-1}}), \tag{1}$$

where $P(h_j)$ is the probability of MH $h_j$ being constantly present in the neighborhood of MH $i$ from then on.

To estimate the above probability for each MH, we need a way to predict the movements of MHs after any given moment. This prediction should not involve complex calculations because this computation should be carried out on MHs with limited battery power and rudimentary processing capability. To this end, we can use prediction techniques, such as those that have been successfully employed in the field of data compression. If we consider each neighborhood of a MH as a symbol of an alphabet, we can use this kind of technique to predict the next symbol/neighborhood. Compression and decompression algorithms for images and video are routinely included in the software of new mobile phones and devices as their implementation does not have a prohibitive cost. Most successful compression methods use context-modelling techniques, which estimate the appearance probability of the next symbol in the text given that a substring (context) has already been seen. A digital trie structure is typically used for organizing the contexts we meet as we are parsing the text. For updating this structure, we use an heuristic similar to that used in the LZ78 algorithm [17]. Specifically, a dictionary of common substrings found in the text is organized using a digital trie structure. Updating proceeds as follows:

1. Initially the dictionary is empty.
2. Examine the remaining input stream and search for the longest prefix which has appeared in the dictionary.
3. Add the prefix followed by the next symbol in the input stream to the dictionary.
4. Go to Step 2.

In our method, for a MH $i$, each symbol is the set of MHs that are neighbors of MH $i$ at any given moment and the input stream is the sequence of neighborhoods of MH $i$ over time. Each MH learns about its neighbors through a periodic exchange of HELLO messages. Specifically, a MH consider as current neighbors all MHs from which it has received a HELLO message in the last $2 \cdot BI$ seconds, where $BI$ is the broadcast interval of HELLO messages.

Algorithm 1 shows the use the digital trie structure in our method. The structure is accessed every $BI$ seconds. Each trie node holds a neighborhood and a counter which measures the number of times this particular node has been met in total during periodic visits. Initially, the trie contains only one node, the root node. A current node is also maintained which shows the node we have reached in the last access of the trie. In the next visit to the trie, we search for the child node of the current node which contains the current neighborhood. If this child is found, the counter of the node is increased and the current node is updated in order to show this node now. Otherwise, the current neighborhood is inserted as new child of the current node and the counter of the new node is set to 1. We also return to the root of the trie (line 14). Essentially, the sequence of neighborhoods from the root up to the parent of the newly inserted leaf is the longest prefix of the remaining stream of neighborhoods that could be found in the trie structure (step 2 of the above updating procedure).

Clearly, at each visit to the trie, we compare the current neighborhood to the neighborhoods of all the children of the current node in the worst case. Assuming that the neighbors inside each neighborhood are kept sorted according to their id, the comparison of two neighbors requires $2\Delta - 1$ comparisons at most,[1] where $\Delta$ is the maximum number of neighbors that a MH can have at any time. So, finding the child holding the current neighborhood (if any) requires $(2\Delta - 1) \cdot c_{\max}$ comparisons at most where $c_{\max}$ is the maximum number of children of a node in the trie.

[1] The technique for achieving this complexity is similar to the technique used for merging two sorted lists [18, p. 48].

**Algorithm 1.** Periodic access of the MH trie

```
1:  {Initially, the trie has only one node, the root node.}
2:  cur_node = root
3:  while true do
4:    wait for BI seconds {The trie is accessed every BI seconds.}
5:    Let h_0, h_1, ..., h_{l-1} be the neighbors seen in the last 2 · BI seconds.
6:    Find the child node c of the cur_node with c.neighborhood = {h_0, h_1, ..., h_{l-1}}
7:    if c exists then
8:      cur_node = c
9:      cur_node.count = cur_node.count + 1
10:   else
11:     add c as a new child of the cur_node
12:     c.neighborhood = {h_0, h_1, ..., h_{l-1}}
13:     c.count = 1
14:     cur_node = root
15:   end if
16: end while
```

An example of trie update is illustrated in Fig. 1. The sequence of neighborhoods in the upper left of the figure are the current neighborhoods of MH 0 in successive intervals of *BI* seconds. The dashed lines show the points at which we cannot go down further in the trie and so we have to return to the root of the trie (line 14 of Algorithm 1). Each neighborhood just before a line is a new neighborhood to be inserted in the trie and so at these points the trie grows by one node. From Fig. 1, it can been seen that each trie node not only corresponds to a particular neighborhood but also is associated with a neighborhood subsequence that start from the root of the trie and ends at this node. Under this meaning, the counter of a node also shows how many times we have met the corresponding subsequence during periodic visits to the trie. For instance, the trie node labelled ({1,2,3,4},6) shows that the sequence of neighborhoods {1,2,3}, {1,2,3,4} has been met 6 times in total. After leaving an initial period for the trie to grow, node counters in the structure can be used for reliable conditional probability estimates. Specifically, given that we have met a particular neighborhood subsequence and so we are at a particular node in

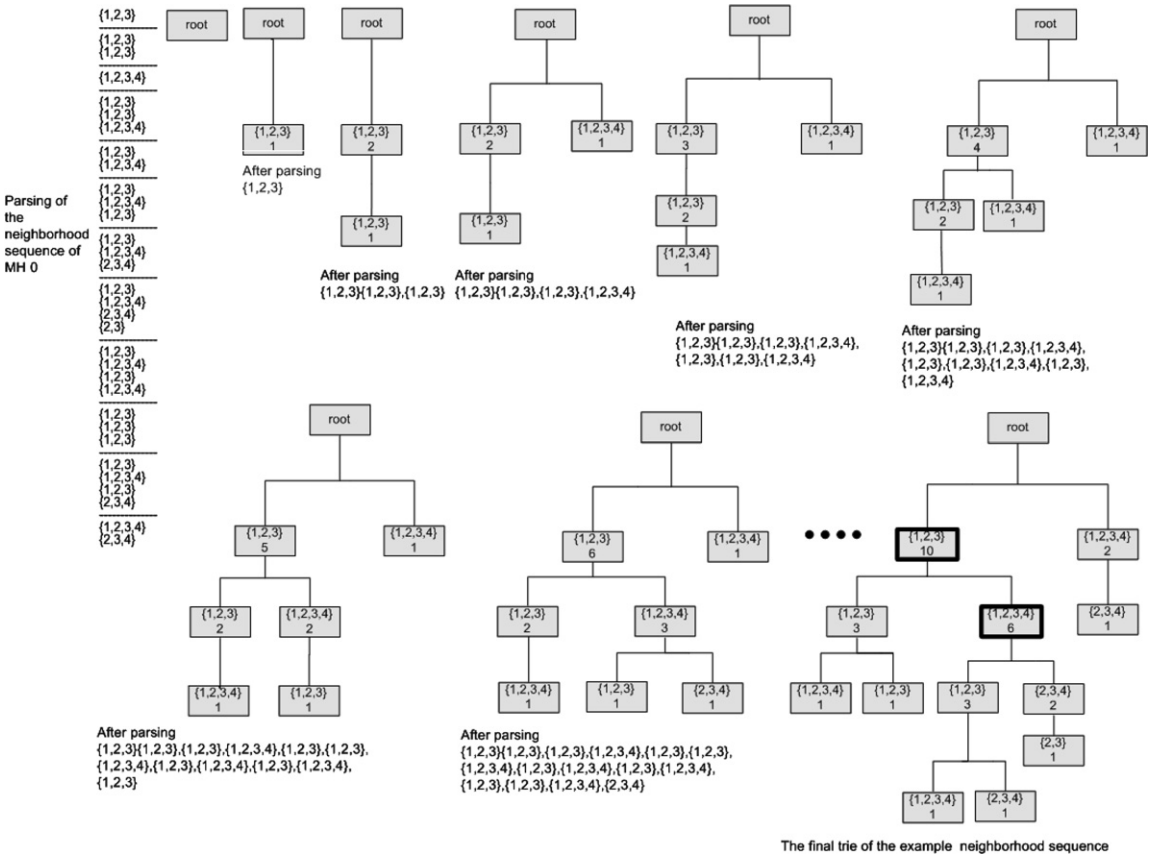



**Fig. 1.** An example of trie construction.

the trie, the counter of each of the children of this node is a measure of the conditional probability of what neighborhood appears next. Clearly, the more skew the probability distribution of the next possible neighborhoods, the better the prediction of what really follows.

So, we can use the information stored in the trie in order to estimate the probabilities in Eq. (1). Given that the $m$ most recent neighborhoods of a MH are the neighborhoods $ngh_0, ngh_1, \ldots ngh_{m-1}$, Algorithm 2 estimates the probability of the current neighborhood $ngh_{m-1}$ of being constantly present from that point on. In case that this sequence does not appear as a whole in the trie, i.e. there does not exist the corresponding path in the trie, we find the largest suffix of this sequence appearing as a tree path starting from the root of the trie (line 2 in the algorithm). In this way, we ensure that the conditional probability estimation will always be based on the most recent available past history.

Next, Algorithm 2 visits all the nodes in the subtree of trie node $c_{m-l-1}$, which holds the neighborhood $ngh_{m-1}$, and then count for each neighbor $h_i$ belonging to $ngh_{m-1}$ the number of times this host appears in the neighborhoods of the subtree (lines 7–9). It also sums all the counters of the nodes of the subtree. Now, the probability of host $h_i$ of being constantly present in the neighborhood of the specific MH is assumed to be the ratio of the first number to the second one (line 15). Finally, by applying Eq. (1) we can get the probability of the current neighborhood $ngh_{m-1}$ of being constantly present in the future (line 16). If $s$ is the size of the subtree of the trie node $c_{m-l-1}$ then the number of additions for finding the probabilities of the hosts in $ngh_{m-1}$ is $s \cdot q + s$ at most. Apparently, we also need $q - 1$ multiplications and $q$ divisions for finding the probability that neighborhood $ngh_{m-1}$ will be unceasingly present from then on.

**Algorithm 2.** Probability estimation for the current neighborhood

1: Let $ngh_0, ngh_1, \ldots, ngh_{m-1}$ be the sequence of the $m$ last seen neighborhoods with $ngh_{m-1}$ being the current neighborhood.
2: Find the minimum $l$ such that there are $m-l$ trie nodes $c_i (i = 0, \ldots, m-l-1)$ with $c_i.neighborhood = ngh_{l+i}$ $(i = 0, \ldots, m-l-1)$ and $c_0.parent = root$, $c_i.parent = c_{i-1}$ $(i = 1, \ldots, m-l-1)$.
3: Let $h_0, h_1, \ldots, h_{q-1}$ be the hosts in the neighborhood $ngh_{m-1}$ and $cnt_0, cnt_1, \ldots, cnt_{q-1}$ be $q$ counters one for each neighbor.
4: Set $total\_count = 0$ and $cnt_i = 0$ for $i = 0, \ldots, q-1$.
5: **for** each node $c$ in the subtree of $c_{m-l-1}$ **do** {root $c_{m-l-1}$ excluded}
6: $total\_count = total\_count + c.count$
7: **for** $i = 0, \ldots, q-1$ **do**
8: **if** $h_i$ appears in $c.neighborhood$ **then**
9: $cnt_i = cnt_i + c.count$
10: **end if**
11: **end for**
12: **end for**
13: $Prob = 1$
14: **for** $i = 0, \ldots, q-1$ **do**
15: $P_i = cnt_i / total\_count$
16: $Prob = Prob \cdot P_i$
17: **end for**
18: return $Prob$

An example of probability estimation is illustrated in Fig. 1. Given that the last two neighborhoods of MH 0 are the neighborhoods {1,2,3}, {1,2,3,4} (and so the current neighborhood is the {1,2,3,4}) we can easily see that the probability of MH 1 and 4 being a neighbor of MH 0 from then on is $\frac{3+1}{2+3+1+1+1} (= \frac{4}{8})$ and $\frac{2+1+1}{2+3+1+1+1} (= \frac{4}{8})$, respectively, while the same probability for MHs 2 and 3 is 1. So, the probability that MH 0 will keep having the current neighborhood $(1, 2, 3, 4)$ from then on is the product of the appearance probabilities of each MH, that is $\frac{4^2}{8^2}$.

An alternative to the proposed prediction technique would be each MH keeping a separate trie structure for each of its neighbors. In this way, it would store the presence/absence patterns of each neighbor separately. In this case also, each trie node would have only two children nodes since now there are only two possibilities, that is either a neighboring MH is present or not. Although, this solution would be possibly attractive because each individual trie would be a simple binary tree, we think that this approach fails to capture most of the existing correlations in the movement patterns of neighbors. Specifically, it records only the movement pattern of each neighbor of a MH relevant only to that MH while in our approach, storing sets of neighbors additionally reveals possible correlations among movements of neighboring MHs, e.g. as in the case of group mobility. This also leads to more accurate predictions for the stability of the current neighborhood at least in principle.

From an opposite point of view, the basic technique of storing sets of neighbors in the trie structure may not automatically guarantee high neighborhood predictability. In other words, the main question is how predictable the neighborhoods formed at each step actually are. The formation of a particular neighborhood around a MH is not the result of the behavior of only this MH but it is the cumulative effect of the behaviors of a number of MHs at the same time. So the probability of the next neighborhood at a particular trie node may not be as biased as we would wish. Another difficulty is that now each symbol in the trie of a MH is a set of the ids of hosts that are neighbors of the MH. This could result in a large number of trie nodes and in addition many of the counts in the trie nodes may not have accumulated enough to be considered as statistically significant.

In order to alleviate these problems, we present a modification to the basic scheme in the following section which drastically reduces the number of trie nodes and also increases the confidence in the estimation of predicted neighborhoods.

### 2.1. An improvement on the basic scheme – Neighborhood merging

We introduce the slack variable $k$ and also allow each trie node to store a number of neighborhoods, where each neighborhood in this set differs in at most $k$ MHs from all other neighborhoods in this set. Formally, neighborhoods $neigh_i$ and $neigh_j$ are stored in the same node only if

$$|ngh_i - ngh_j| + |ngh_j - ngh_i| \leqslant k, \tag{2}$$

where "−" is the set difference operator. Now, as we go down the trie, the new neighborhood encountered at each step is checked against each of the possible children of the current trie node. We insert the new neighborhood to the first child for which the condition above is not violated after that insertion. If none of the children satisfies the condition above, we create a new trie node as a child of the current trie node. This newly created trie node will contain only the new neighborhood. Since the host ids in each neighborhood are kept sorted, the computation of (2), requires $2\Delta - 1$ comparisons at most, where again $\Delta$ is the maximum number of hosts that can be found in a neighborhood. So, if $n_{\max}$ is the maximum number of neighborhoods that a trie node stores and $c'_{\max}$ is now the maximum number of children of a trie node, going down one-level in a trie requires $(2\Delta - 1)n_{\max}c'_{\max}$ comparisons at most. Note that in comparison to our original technique, there exists a kind of trade-off. Specifically, $c'_{\max}$ is expected to be smaller than $c_{\max}$, i.e. the maximum number of children in our original technique. But now we also have that $n_{\max} \geqslant 1$ while in the first technique each trie node stores only one neighborhood.

Another important detail of this scheme is that we separately count the number of appearances of each individual neighborhood inside each trie node so that the neighborhood appearance probabilities can be accurately computed. We can easily deduce that the estimation of the probability of the current neighborhood being constantly present from then on requires $s \cdot \Delta + s$ additions at most where now $s$ is the total number of neighborhoods stored in the subtree on which the probability estimation takes place. Again, we also need $\Delta - 1$ multiplications and $\Delta$ divisions at most.

In conclusion, with this scheme of neighborhood merging, we drastically reduce the number of trie nodes. This in turn allows the build-up of statistically significant counter values, which can be used reliably for the estimation of the appearance probability of MHs in the neighborhood of a particular MH.

Now, it is interesting to provide an estimation of the number of children each trie node will have after this modification. Consider again a MH $i$ and its neighborhood $ngh_{i,t} = \{h_0, h_1, h_2, \ldots, h_{n_t-1}\}$ at time step $t$. Let also $N$ be the total number of different nodes that were or will be neighbors of MH $i$ and let also $ngh_{\text{all}}$ be the corresponding set containing all these nodes. For example, if $ngh_{i,1} = \{1,2\},\ ngh_{i,2} = \{2,3,4\},\ ngh_{i,3} = \{2,5\}$, $ngh_{i,4} = \{1,2\},\ ngh_{i,5} = \{2,3\}$, then for these five time steps we have $N = 5$ and $ngh_{\text{all}} = \{1,2,3,4,5\}$. Normally, the parameter $N$ will not have high values, since each MH is not expected to have many different neighbors. Clearly also, each of the neighborhoods $ngh_{i,t}$ of MH $i$ are subsets of the $ngh_{\text{all}}$ and so they can be represented with a codeword $c_{i,t}$ of length $N$ where each bit in this representation shows the presence or absence of the corresponding MH in the neighborhood $ngh_{i,t}$. In the example above, $c_{i1} = 11000,\ c_{i2} = 01110,\ c_{i3} = 01001,\ c_{i4} = 11000,\ c_{i5} = 01100$.

Under this interpretation, we can revisit the condition of neighborhood inclusion in a trie node and write the inequality (2) as $|c_{it} \oplus c_{jt}| \leqslant k$, where $\oplus$ is the bit-XOR of the codewords $c_{it},\ c_{jt}$. The result of this operation is also known as the hamming distance between two codewords and so essentially the hamming distance of the codewords of any two neighborhoods stored in the same trie node should be not more than $k$.

Now, we give a useful definition from the coding theory field [21]:

**Definition 1.** A $\rho$-covering code, $\rho - C_N$, is a set of $N$-bit codewords with the property that every other $N$-bit word has hamming distance $\rho$ or less from at least one codeword in $\rho - C_N$.

Using this definition, we can state the following proposition:

**Proposition 2.** *There is a direct relation between the number of the children of a trie node and the size of a $k/2 - C_N$ covering code.*

First, note that the Hamming distance belongs to the class of metric distances. As a consequence, all codewords of neighborhoods stored in a trie node form a "spherical" cluster around a particular codeword. In other words, the Hamming distance of this centric word from the codewords of the trie node is at most $\frac{k}{2}$. Note that a codeword at the center of such a "spherical" cluster does not necessarily correspond to a real neighborhood that has been stored in the trie node. Now, it is clear that if a trie node has $m$ children after applying the technique of neighborhood merging, then there will be $m$ centric codewords $cc_i$ $(i = 1, \ldots, m)$ whose spheres "cover" all the other codewords. This simply means that all codewords of neighborhoods occurring immediately after a particular trie node have Hamming distance at most $\frac{k}{2}$ from at least one of the $m$ codewords $cc_i$. Codewords $cc_i$ should cover $2^N$ codewords (all possible subsets of $neigh_{\text{all}}$) at the worst case and hence constitute a $k/2 - C_N$ covering code.

However, in practice we need to cover far less than $2^N$ codewords, since the possible neighborhoods of a MH are not that many.

As the neighborhoods of a MH $i$ are stored in the children of a trie node in a first-fit fashion, that is each neighborhood is stored in the first trie node where relation (2) is satisfied for all the neighborhoods of that node, the "spherical" clusters of the corresponding codewords are formed basically at random. So, the question about the number of children of a trie node

after applying neighborhood merging can be recast as follows: if we randomly select codewords $cc_i$, how many such words should we select so that all the other codewords ($2^N$ in the worst case) will be at (Hamming) distance at most $k/2$ from at least one of the codewords $cc_i$?

A well known lower bound for the number $m$ of the codewords in a $k/2 - C_N$ covering code is the so called *sphere covering bound* [21, Theorem 11.1.4, p. 434] which adjusted to our case is the following:

$$m \geqslant \frac{2^N}{\sum_{i=0}^{\frac{k}{2}} \binom{N}{i}}. \tag{3}$$

The denominator of the expression, $\sum_{i=0}^{\frac{k}{2}} \binom{N}{i}$, is the number of codewords differing in at most $\frac{k}{2}$ bits from a particular codeword. So, each "spherical" cluster around each codeword $cc_i$ $(i = 1, \ldots, m)$ contains that many codewords and at the same time the $m$ clusters should cover all $2^N$ codewords. Apparently, the number $m$ of clusters satisfying this requirement also obeys the inequality above.

Although, a closed expression for the sum $\sum_{i=0}^{r} \binom{N}{i}$ does not exist, an asymptotic formula is provided for this sum in [22, p. 598]. Specifically, if $\alpha$ is a constant in the range $0 < \alpha < \frac{1}{2}$ and $r = \alpha N$, then

$$\sum_{i=0}^{r} \binom{N}{i} = 2^{NH(\alpha) - \frac{1}{2}\log N + O(1)}, \tag{4}$$

where $H(\alpha) = -\alpha \log_2 \alpha - (1-a)\log_2(1-a)$. Note that the $H$ function is a binary entropy function and its value is near 0 when $\alpha$ tends to 0 and 1 when $\alpha$ takes values close to $\frac{1}{2}$. In our case, we have $\alpha = \frac{k}{2N}$ and the lower bound (3) can also be written as $m = \Omega(2^{N\cdot(1-H(k/2N))})$.

With regard to the upper bound of the number $m$ of codewords $cc_i$ needed for covering $2^N$ codewords, we can prove an upper bound with high probability by using a known fact from the coding theory [23, Theorem 12.1.2, p. 320][24, Lemma 4, p. 74]. Specifically, we can prove that if we select $m = N \cdot 2^N / \sum_{i=0}^{\frac{k}{2}} \binom{N}{i}$ codewords $cc_i$ at random $(i = 1, \ldots, m)$, these codewords cover all possible $2^N$ codewords with probability at least $1 - 2^N \cdot e^{-N}$. Note that this probability tends to 1 when $N \to \infty$. So, we can achieve the lower bound (3) up to a factor $N$ with high probability.

So, we have determined a lower and a close upper bound for the number $m$ of covering codewords $cc_i$ and hence for the number of children under a trie node after applying neighborhood merging. Now, we can easily prove the following lemma.

**Lemma 3.** *By reasonably assuming that the slack variable k grows proportionally to N, i.e. $\alpha = k/2N$ is a constant with $a \in (0, \frac{1}{2})$, the reduction in the number of children nodes of any trie node after neighborhood merging is $\Omega(N^{-\frac{3}{2}} \cdot 2^{N\cdot H(\alpha)})$ at least with high probability, that is the reduction is almost exponential[2] in N.*

**Proof.** By the asymptotic formula (4), the upper bound for $m$ can also been written as $m = O(N^{\frac{3}{2}} \cdot 2^{N\cdot(1-H(\alpha))})$. Before the use of the slack variable $k$, a trie node could have $2^N$ children in the worst case. So, after neighborhood merging, the reduction in the number of children nodes is $2^N/m = \Omega(N^{-\frac{3}{2}} \cdot 2^{N\cdot H(\alpha)})$ at least. □

It should be noted again that the whole analysis above is a worst case analysis. In practice, the number of possible neighborhoods that may exist immediately after a particular history of neighborhoods will be much less than $2^N$. Accordingly, the codewords $cc_i$ will actually cover considerably smaller number of codewords than $2^N$ and thus in practice $m$ is expected to be much smaller than the value predicted in our worst case analysis.

## 3. Mobility-aware highest degree (MobHiD) technique

The distributed implementation of our technique should create a stable clustering structure with minimal control overhead. As CHs and gateways will form the virtual backbone through which messages will be routed on the ad hoc network, the size of this backbone should be kept as small as possible so that the delay of message routing is correspondingly small. Also, we opt for one-hop clusters where each MH is one-hop away from its CH. In this way, the routing decisions inside each cluster are straightforward and there is no need for involved routing update protocols within each cluster.

The highest degree (HD) clustering algorithm [2] is a clustering scheme that creates a relatively small number of one-hop clusters and thus a small-size routing backbone. In this technique, each MH having the highest degree among all its neighbors is elected as CH. The degree of each MH is the number of one-hop adjacent MHs. The main weakness of the technique is the frequent CH changes due to host mobility. However, by combining the HD technique with our mobility prediction scheme, we substantially eliminate the instable behavior of the technique. More precisely, we define the weight $w_i$ for each MH $i$ as follows:

$$w_i = a_1 \cdot P(neigh_i) \cdot d_i + a_2 \cdot avg\_d_i, \tag{5}$$

[2] *Note that* $2^{H(\alpha)} > 1$.

where $P(neigh_i)$ has been defined in the previous section and is the probability that the current neighborhood $neigh_i$ of MH $i$ will remain the same from then on. Also, $d_i$ is the degree of MH $i$, i.e., the number of neighbors in $neigh_i$, and $avg\_d_i$ is the average degree of the future neighborhoods of the MH $i$, which can be easily computed from the information contained in the sub-trie having root the current neighborhood $neigh_i$. Finally, the coefficients $a_1$ and $a_2$ are used to give more weight to the first or the second term in the expression above.

A large value of weight $w_i$ practically means that MH $i$ is surrounded by many neighbors that will remain in the vicinity for a long time with high probability. In addition, due to the second term in the weight expression, it is very likely that MH $i$ will continue to be surrounded by a large number of neighbors in the future too. So, by electing as CHs hosts that have the largest weight value in their neighborhood, we can obtain a small-size virtual backbone, which will remain stable despite host mobility.

## 4. Distributed implementation

In the proposed mobility prediction method, each MH $i$ should compute its weight $w_i$ according to the weight formula (5). Therefore, each MH should know its neighbors and how its neighborhood changes over time. This implies a periodic exchange of HELLO messages, namely messages HELLO($clusterhead?, w_i$) so that each MH $i$ can inform its neighbors about its presence, whether it is a clusterhead or not and about its weight. The information carried by the two fields of the HELLO message proves useful when a MH wishes to affiliate with another cluster or during reclustering.

### 4.1. Cluster formation

With regard to the cluster formation, the distributed implementation should tolerate possible topology changes while cluster creation is in progress. Our method for cluster formation adopts some ideas of the distributed implementation of the DMAC clustering algorithm[9]. However, in our scheme the host weights are determined from the weight formula (5). Note that on the system startup, MHs have not yet gathered statistics in their trie structure and hence the initial CH election is carried out essentially according to the HD technique, i.e., in formula (1) we set $w_i = d_i$.

Cluster formation is done as follows: Each MH $u$ that has the highest weight among its neighbors broadcasts the message CLUSTERHEAD($u$) to its neighbors, thus declaring its decision of being CH. If the MH $u$ does not possess the largest weight in its neighborhood, first it waits for the decision of all the MHs having larger weight than its own weight and then it decides its own role (CH or ordinary MH). More precisely, there are two cases. First, if MH $u$ has received at least one CLUSTERHEAD message, it joins the cluster of the CH, say CH $v$, which has the highest probability of still being neighbor in the future in comparison to other CHs that sent CLUSTERHEAD messages. This information can be easily obtained from the trie structure of $u$. Then, MH $u$ broadcasts the message JOIN($u, v$) to communicate its decision to its neighbors. However, if MH $u$ received only JOIN messages from all neighboring MHs with larger weights, this simply means that all these MHs have deferred to other CHs. Now, MH $u$ is free to become a CH and thus it broadcasts the CLUSTERHEAD message to its neighbors.

### 4.2. Cluster maintenance

Our method for cluster maintenance eliminates the problem of frequent CH changes, by allowing a MH to become a CH or to affiliate with a new cluster without starting a reclustering process. In addition, our method does not suffer from the chain reaction effect [19] where local changes in clusterhead roles may propagate over the network. Indeed, in case of reclustering, the size of the affected area is effectively controlled by our method. In the following, we give a high-level description of the Cluster Maintenance. For further details, the pseudocode of this phase can be found in Appendix A.1.

First, if an ordinary MH $u$ cannot connect to its CH anymore, it tries to find another neighboring CH $v$ and then affiliates to the corresponding cluster by sending the message JOIN($u, v$) to $v$. If more than one CH exist in the neighborhood of $u$, it connects to the CH having the highest probability of still being a neighbor in the future in comparison to other neighboring CHs. In the case that there is no CH in its vicinity, then $u$ becomes a CH.

Reclustering may be initiated only by a CH. Specifically, reclustering is triggered, only when a CH realizes that the number of CHs having gathered in its neighborhood is above a particular threshold $L$. Due to the periodic exchange of HELLO messages, each CH can easily and rapidly check if the condition above for reclustering actually holds. As said before, a difficult issue about reclustering is the extent of the area that will be affected by this reclustering. It is possible for a single local change in topology to trigger global reclustering with considerable control overhead. Also, the group of CHs after reclustering may be quite different from the previous CH group and thus a lot of routing information must be either transferred from the old CHs to the new ones or acquired again from scratch via routing protocols.

#### 4.2.1. Avoiding global reclustering

To avoid global reclustering in our scheme, we restrict reclustering locally around the CH that triggers the new reclustering. We introduce the parameter *cl_extent*, which determines the clusters around the triggering CH that will participate in the reclustering. Specifically, when a CH $i$ triggers a reclustering, it broadcasts the message RECLUSTER($i, cl\_extent$) to its neighbors. After receiving this message, each of these neighbors leaves from the idle state, INACTIVE state, and moves to

the ACTIVE state where it is ready to participate in the reclustering procedure. Then, these nodes relay the message further to their neighbors, which change their state accordingly.

When a CH receives the RECLUSTER message, it first decreases by one the value of *cl_extent* and then sends the message. The last CH that reduces the value of *cl_extent* to 0 sends the message to its neighbors and the receiving MHs stop flooding the message. Note that the first field of the RECLUSTER message is not altered by the relaying hosts and it is used for avoiding duplicate RECLUSTER messages. Specifically, if a RECLUSTER message has already reached a MH, this host ignores any other RECLUSTER messages which arrive later with the same value in the first field. Essentially, the id of the triggering CH passed in the first field of the RECLUSTER message serves as an identification of the current reclustering process and thus MHs ignore any further notifications for the current reclustering.

After having received the RECLUSTER message, MHs start executing the clustering algorithm. MHs which still remain in the state INACTIVE ignore any message CLUSTERHEAD or JOIN that arrives from neighboring MHs. However, all the MHs that will participate in a reclustering are not activated at the same time because the RECLUSTER message does not reach all relevant MHs instantly. So each MH waits for a while before starting the clustering algorithm so as to ensure that all relevant MHs have been activated. Specifically, we can prove the following lemma.

**Lemma 4.** *If a MH has been activated by the receipt of the message RECLUSTER* $(i, cl\_extent)$*, then MH needs to wait for* $3 \cdot (cl\_extent + 1) \cdot T$ *seconds at most before running the clustering algorithm, where T is an upper bound of the message transfer delay over a single link.*

**Proof.** Fig. 2 shows the longest path that a RECLUSTER message may follow. Notice that no MH on this path is within the transmission range of both CHs of adjacent clusters. In this case, two MHs, one in each cluster, should serve as gateways and thus any two successive CHs on the path of the RECLUSTER message are three hops away. Therefore, when a CH receives a message RECLUSTER$(i, cl_extent)$, it should wait for $(3 \cdot cl\_extent + 1) \cdot T$ seconds in order that the message can reach the last CH on the path and then the remaining MHs of the last cluster. Note also, after RECLUSTER message leaves a CH, the value of the *cl_extent* field has been decreased by one, and this value remains constant as the RECLUSTER message passes through the two gateways up until the next CH. The first of the two intervening gateways should wait for $3 \cdot (cl\_extent + 1) \cdot T$ seconds and other for $(3 \cdot cl\_extent + 2) \cdot T$ seconds where *cl_extent* now is the new value of the field after decreasing by one in the last CH. Overall, a MH should wait for $3 \cdot (cl\_extent + 1) \cdot T$ seconds at most. □

If a MH receives a new message RECLUSTER$(j, cl_extent')$ while waiting for starting the clustering algorithm, it extends its wait for another $3 \cdot (cl\_extent' + 1) \cdot T$ seconds at most in order to accommodate the new reclustering request coming from CH *j*. After this waiting period expires, the MH starts executing the clustering maintenance algorithm. Now, each MH that does not have the largest weight among its neighbors should wait for a message, either CLUSTERHEAD or JOIN, from each active neighbor with larger weight than its weight in order to take a decision, i.e., whether it becomes a CH or not. However, due to host mobility, a MH may not be sure about which of its neighbors are actually active and hence whether it should wait for them before reaching a decision. For this reason, each active MH investigates the state of each neighboring MH with larger weight by sending the message INVITE. On receipt of such message, each active MH sends the message ACCEPT as an answer. Otherwise, when the MH is in the state INACTIVE, it replies with the message DECLINE. So, a MH that receives one of these replies from another MH knows whether it should wait for the decision of this particular MH or not. Of course, if a MH happens to receive a CLUSTERHEAD or JOIN message from another MH before sending the INVITE message to this particular MH or while waiting for ACCEPT or DECLINE message from this MH, it can proceed without waiting for its reply, because it now knows indirectly that the MH is already active.

Note that an active MH returns to the INACTIVE state only after having reached its decision. Before its final decision, a MH accepts any invitation for joining a reclustering process. Also, an invitation could also come from a MH executing a different

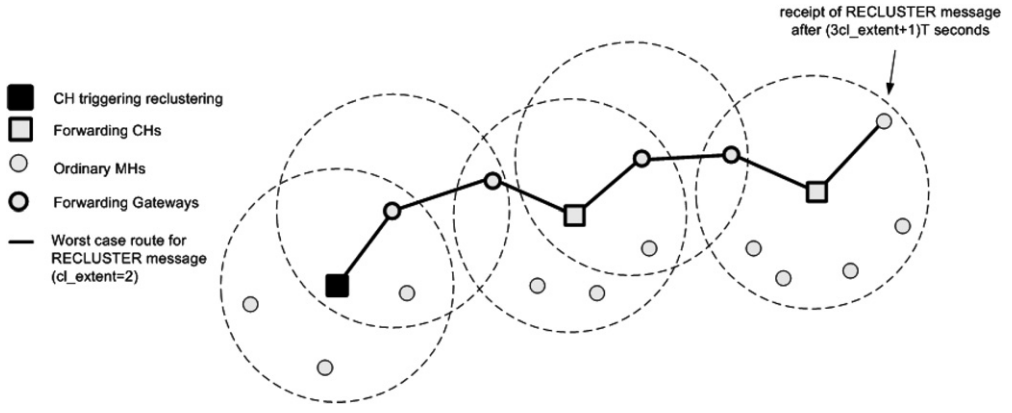



**Fig. 2.** Worst case scenario for RECLUSTER message route.

instance of reclustering. So, if more than one reclustering processes are concurrently in progress, the exchange of INVITE/ACCEPT messages enables the fusion of these processes into one. Now, the total affected network area is the union of the affected areas of the initial concurrent processes.

A possible concern about the restricted reclustering is what happens on the boundary of the affected area. It may be the case that a number of CHs near the boundary have been elected and thus a CH outside the affected area may trigger again a reclustering. This chain reaction effect can be avoided by properly setting the $L$ parameter which determines the number of CHs that can be adjacent to a CH before triggering reclustering. Note also that if the transmission range of each host is $R$, all the CHs that have been elected in the just finished reclustering are now at distance greater than $R$. Now, by using arguments similar to those in [25], we can prove the following lemma.

**Lemma 5.** *In an area of radius R, there can exist at most five MHs whose mutual distance is greater than R and none of them occupies the center of the area.*

**Proof.** First, we give an example of five hosts that satisfy the claim of the lemma. Specifically, Fig. 3 shows five hosts $H_i,\ i = 0,\ldots,4$, located on the vertices of a regular pentagon inscribed inside a circle of radius $R$. We can easily check that the distance between any two hosts is greater than $R$.

Now we will prove that there cannot exist a placement of six hosts inside an area of radius $R$ which will satisfy the requirements of Lemma 5. In Fig. 4a, we see six hosts on the vertices of a regular hexagon which marginally do not satisfy the requirements of the lemma since the distance for some pairs of hosts is exactly $R$ and for other are greater than $R$. Note also that successive vertices of the hexagon form a 60° angle with the center of the area.

Now assume that we can find a placement of six hosts that satisfy the requirements of Lemma 5, e.g., the placement of hosts $H_i, i = 0,\ldots,5$ in Fig. 4b. In this figure, $H'_i (i = 0,\ldots,5)$ denotes the intersection with the circle of the ray starting from the center $C$ of the area and passing through the corresponding host $H_i$. Note that there must be at least one value of $i$ such as $|H'_iH'_{(i+1)mod6}| \leqslant R$. Otherwise, all the angles $\angle H'_iCH'_{(i+1)mod6}$ will be greater than 60° and hence their sum will be larger than 360°, which is impossible. So, let $H'_2H'_3$ be a line segment which is shorter than or equal to $R$. By inspection of the Fig. 4b, we can see that the distance between the hosts $H_2$ and $H_3$ cannot be greater than $R$. So, there is a contradiction with our assumption that each host is at distance greater than $R$ from all other hosts. □

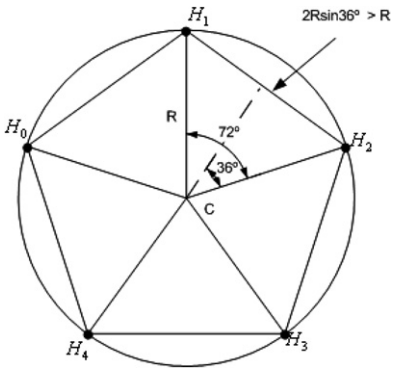



**Fig. 3.** An example of 5 points satisfying the claim of Lemma 5.

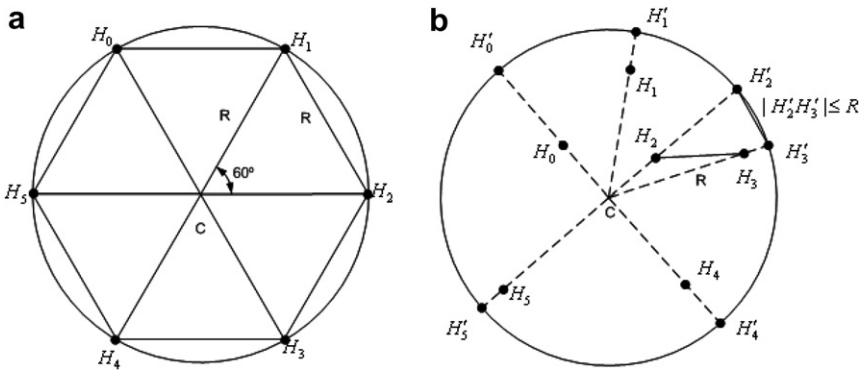



**Fig. 4.** No existence of 6 points satisfying the claim of Lemma 5.

It is clear now that at most five CHs inside the affected area can be neighbors with a CH which is outside the affected area. By setting $L > 5$ and assuming that only this specific reclustering event happens over the whole network, we can ensure that CHs outside the affected area do not unnecessarily trigger the reclustering process again.

### 4.3. Message and time complexity of MobHiD algorithm

In determining the performance of a distributed algorithm, message and time complexity are always two important metrics to consider. In our algorithm, we first examine the number of messages exchanged and time required for completing the initial cluster formation and then estimate the same metrics for the cluster maintenance phase.

#### 4.3.1. Cluster formation

In this phase, the distributed implementation of the MobHiD algorithm is based on the DMAC algorithm. Through a periodic exchange of HELLO messages, each MH learns the weights of their neighbors and then should wait for the decision of the neighbors with higher weights before reaching a decision for its own status (clusterhead or ordinary host). After that, MHs send either a JOIN or a CLUSTERHEAD message, informing neighbors about their final decision. Each MH receives at most $\Delta$ messages from their higher weighted neighbors, where $\Delta$ is the maximum number of neighbors a MH can have. Overall, during the cluster formation phase, exactly one message is sent and most $\Delta$ are received per node. Here, we assume that a message sent from a MH is received by all its neighbors.

With regard to the total time required for the initial cluster formation, this time is largely determined by the time spent by MHs waiting for the final decision of neighbors with larger weights. More precisely, let $MH_{i_0}, MH_{i_1}, \ldots, MH_{i_{l-1}}$ be a path with monotonically decreasing weights ($w_{i_j} > w_{i_{j+1}}$, $j = 0, \ldots, l-1$) where for $j = 1, \ldots, l-1$, $MH_{i_{j-1}}$ is the last host which $MH_{i_j}$ is waiting for before reaching its final decision. For convenience, we call such a path as a *chain path*. Also, we are mainly interested for maximal chain paths. From maximality assumption, $MH_{i_0}$ has the lowest weight in its neighborhood. It is also clear that all neighbors of the last host $MH_{i_{l-1}}$ different from $MH_{i_{l-2}}$ either have larger weights than $MH_{i_{l-1}}$ or have decided earlier or should wait further for another host to decide. For example, if $MH_{i_{l-1}}$ have decided to join a cluster as an ordinary node, a neighbor of $MH_{i_{l-1}}$, say $MH_{i_l}$, with weight lower than $w_{i_{l-1}}$ may have to wait for another host with weight between $w_{i_{l-1}}$ and $w_{i_l}$ in order to decide. Thus, $MH_{i_{l-1}}$ is not the last host that has decided before the final decision of $MH_{i_l}$.

Clearly, the total time required for completing the initial cluster formation phase is equal to $l_{\max} \cdot T$ where $l_{\max}$ is the length of the longest chain path.

Unfortunately, it is very difficult to make an analytical estimation of $l_{\max}$. However, in Appendix A.2 we give an upper bound for the value of this length, useful only when the ad hoc network is very sparse.

#### 4.3.2. Cluster maintenance

When a CH decides reclustering, it first sends a RECLUSTER message which travels for $3 \cdot cl\_extent + 1$ hops at most in all directions around the initiating CH. The CHs within this local region relay the message to their cluster members while gateways pass it to the next neighboring gateway or CH. MHs also ignore any duplicate RECLUSTER message that they receive during this process. So, each MH sends at most one message and receives at most $\Delta$ messages since RECLUSTER messages could arrive via multiple disjoint paths with starting point the CH that initiated reclustering.

With regard to the time required for this step, recall that each MH that receives a RECLUSTER message may have to wait for all other MHs receiving this message too. From Lemma 4, the waiting time is $3 \cdot (cl\_extent + 1)T$ seconds at most.

After all MHs in the area around the triggering clusterhead have been informed for the pending reclustering procedure, each MH sends one INVITE message to their neighbors. The recipient of the message will be all neighbors with higher weights than that of the sending MH. Normally, each MH should send a separate reply (ACCEPT or DECLINE) to each INVITE message it receives. However, with a small modification in our algorithm we can reduce the reply messages sent by each MH. Specifically, since each MH taking part in the reclustering will send the INVITE message at about the same time, immediately after the initial $3 \cdot (cl\_extent + 1)T$ seconds, a MH that receives the first INVITE message can delay its reply for a short time in order to receive more INVITE messages from the other participating neighbors. So, with high probability it will finally send only one REPLY or DECLINE message to its inquiring neighbors. The additional delay a MH should wait for before replying back could be small constant $c(\leqslant 2)$ times $T$.

On the other hand, each MH that sent an INVITE message will receive $\Delta$ reply messages at most, one from each of its neighbors in the worst case. Clearly also, this round of message exchanges (INVITE and then ACCEPT or DECLINE message) can be completed in $(2 + c)T$ time.

After this step, MHs are ready to execute the cluster formation algorithm. The number of messages sent or received per MH is the same as in the initial cluster formation phase. The time for completing this phase is $l'_{\max}T$ where now $l'_{\max}$ is the length of the longest chain path that can be formed inside the local region around the triggering CH.

## 5. Simulation results

The performance of the MobHiD algorithm was tested through a series of simulation experiments on the ns2 simulator [26]. For comparison, we also simulated four other one-hop clustering algorithms, namely the Lowest ID (LI) [2], Highest

Degree (HD), GDMAC as well as MOBIC. First, we measured how the number of created clusters/CHs of each technique varies with the total number of MHs. Then, we studied the variation of lifetime (duration) of elected CHs in each of these methods against the maximum host speed. We also ran tests where the duration of clusters created by the MobHiD algorithm is compared for various values of the slack variable $k$ as well as for different lengths of the neighborhood context used in our mobility prediction method.

In another set of tests, we compared the number of reaffiliations observed in each algorithm when the maximum transmission range or the speed of each MH varies. We also measured the message control overhead of the five algorithms for various values of maximum MH speed. We also ran simulations for the local memory requirements in each MH, that is the size of local trie built during simulation and specifically the rate of increase in the size of this structure. This performance metric is important since besides showing the feasibility of method in practice, a slow increase in trie size is also indicative of the good prediction capability of the structure; when the trie structure grows slowly, this simply means that every time a new neighborhood is inserted into the trie, a large neighborhood sequence has been accurately predicted just before.

The last set of simulations concerns the computational overhead of the MobHiD algorithm. Specifically, we measured the arithmetic operations per second required in our trie-based mobility prediction technique. Together with low local memory requirements, a low instruction count proves the feasibility of the method again.

With regard to host mobility, we assumed two different mobility models in our simulation tests. First, we ran tests assuming the hosts are moving according to the random waypoint (RW) model [13]. Then, we measured the performance of the algorithms under the reference point group mobility model (RPGM) [28]. The scenarios for these two mobility models were generated with BonnMotion [29], a Java-based tool for generation and analysis of mobility scenarios.

Notice also that the RW model forms the worst case scenario for the MobHiD algorithm since the basic assumption of correlated host movements breaks down under this model. MHs are moving independently of each other and the only correlation that does exist in this model comes from the fact that two MHs which are neighbors at a certain time, they will continue to be neighbors for some more time till they move out of the transmission range of each other.

In contrast, RPGM is the "ideal" setting for MobHiD since hosts are moving in groups and thus their movements are correlated and more easily predicted.

### 5.1. Random waypoint model

First, we examine the performance of the clustering algorithms under the RW model. Specifically, the hosts are moving according to the RW model [13] with zero pause time in a terrain of $1000 \times 1000\ \text{m}^2$. The speed of each host was selected randomly between a minimum and a maximum value. Following the suggestions in [27], we set the minimum node speed to a positive value (1 km/h in our case) throughout our simulations. The maximum value was set a value between 1 and 80 km/h.

Moreover, a number of parameters relevant to each technique were fixed before running the experiments. First, for the MobHiD we set the $a_1,\ a_2$ coefficients to 0.7 and 0.3, respectively. We also set $k = 3,\ L = 6$ and $cl\_extent = 3$ and the broadcast interval of HELLO messages equal to 1 s. As a context for determining the conditional host appearance probabilities in our mobility prediction technique, we used a maximum context length of 10 successive neighborhoods, i.e. we set $m = 10$ in Algorithm 2.

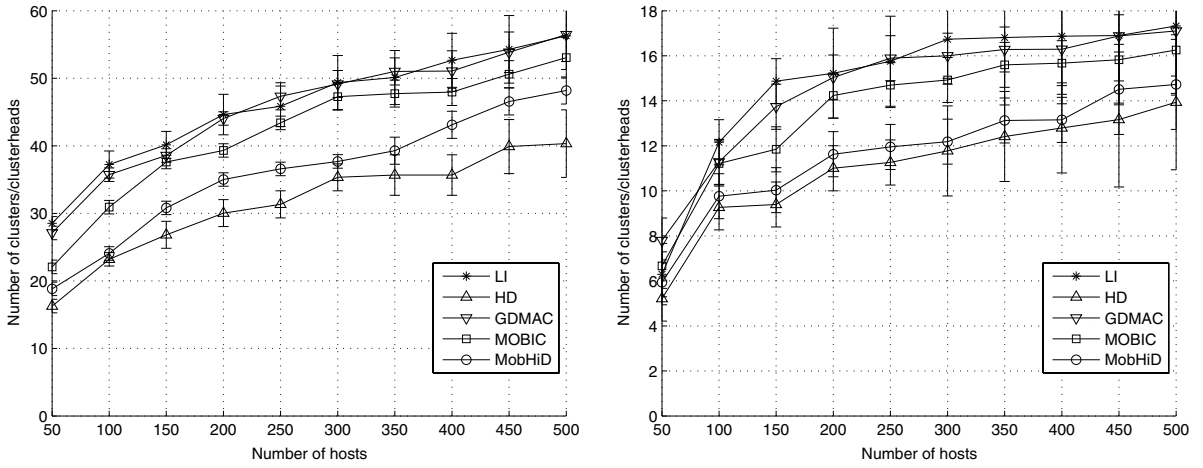
For GDMAC, by following the suggestion in [10], we set the weight of each host equal to $81 - speed$ where $speed$ is the velocity of the host. We also used $k = 6$ where $k$ is the maximum number of neighboring clusters before triggering a reclustering event. We also set the parameter $h$ equal to 30. This parameter in the algorithm controls the frequency of reaffiliations e.g. how often MHs change clusters. Finally for the MOBIC algorithm, we set the clusterhead contention interval (CCI) parameter to 4 s. The CCI determines the maximum period during which two clusterheads can be neighbors. If after that interval these CHs are still neighbors, one of them should give up its role as a CH.

Unless stated otherwise, in simulation tests for the RPGM model the same values are also assumed for the above algorithmic parameters.

#### 5.1.1. Number of clusters

In this set of simulations (Fig. 5), we studied the number of the formed clusters/CHs versus the total number of hosts for two values of transmission range. For the same parameter settings, we ran the simulation 50 times. The maximum speed of each host in these simulation runs was set to 70 km/h. Also, each simulation ran for 25 min and we allowed a warm-up period of 10 min before collecting results in order that the system could reach a steady state [27]. After that initial period, we sampled the number of CHs every 30 s. Then, we averaged these sample values for each simulation run. The final values in Fig. 5 are the mean of these values over the 50 simulation runs. It should be mentioned that similar procedures for collecting results has also been followed in the other sets of simulations as well. Also, in most of figures 95% confidence intervals have been drawn around each point.

The LI technique, the MOBIC as well as GDMAC algorithm produce a higher number of CHs in comparison to MobHiD and HD algorithm. This is because the first three algorithms are not optimized for minimizing the number of resulting CHs in contrast to the last two techniques. Note also that MobHiD is close to the performance of HD. It is also clear that the number of CHs is much lower when the transmission range is relatively high. Indeed, in the second case mobile hosts are connected




(a) transmission range=100m (b) transmission range=200m

**Fig. 5.** Average number of clusters/CHs versus total number of MHs.

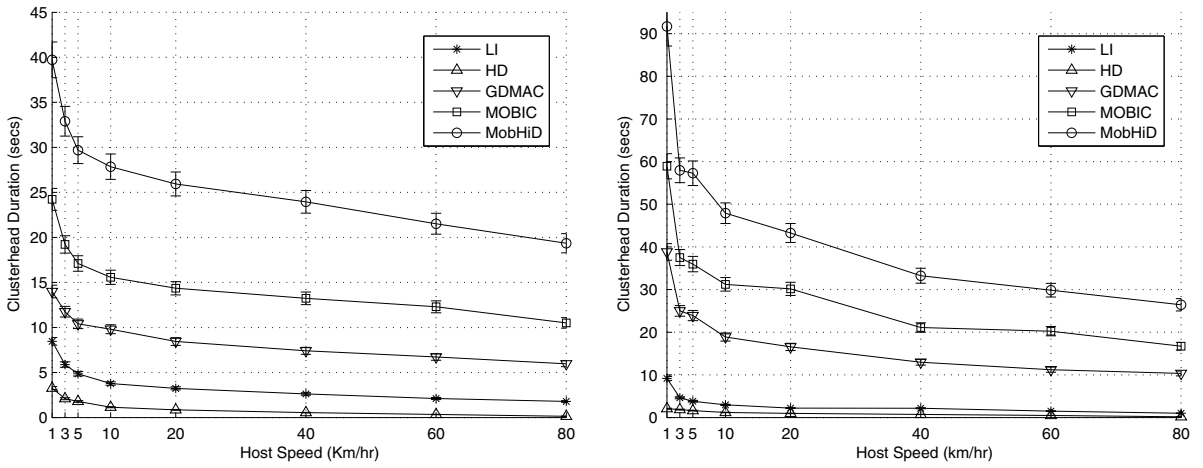
with an increased number of peers and thus more hosts gather in each cluster. This in turn results in much lower number of clusters and hence CHs.

*5.1.2. CH duration*

Now, we compare the techniques with respect to the duration of the elected CHs when the maximum speed of MHs increases (Fig. 6). By doing so, we assessed the stability of the derived clustering structure with increasing host mobility. For these experiments, we used 50 hosts and ran each simulation with the same parameters 50 times. The CH duration for a single simulation run was the average over all the CHs. Then we estimated the average of these values over the 50 simulation runs.

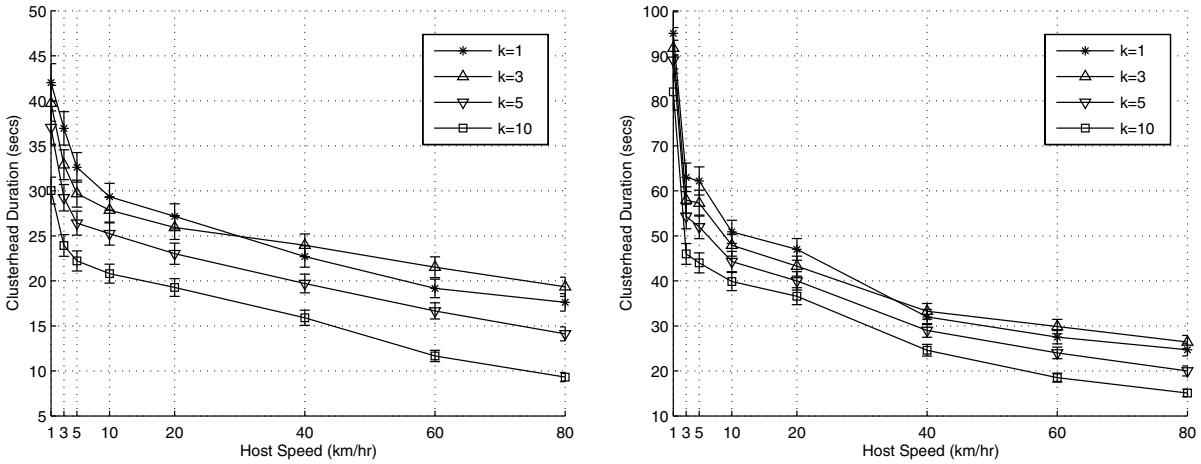
From the experimental results, we can see that MobHiD performs much better than the other techniques in the face of host mobility. By adopting a more reliable prediction model, our technique ensures more stable clustering structure in comparison to other mobility-aware techniques. The HD and LI techniques do not take into account the mobility of hosts in the CH election procedure and hence their poor performance when the maximum attainable speed of hosts increases. Again with larger transmission range, the connectivity graph of hosts is denser and the adjacency relations of hosts change less frequently despite the increased host mobility. As a result, each CH serves for a longer time period before giving up its role.

We also conducted another set of simulations, where we evaluated the impact of the slack variable $k$ and that of the maximum neighborhood context length on the performance of the MobHiD algorithm. In Fig. 7, we see the average CH duration of the MobHiD algorithm with respect to the maximum MH speed for various values of the variable $k$. In this set of the




(a) transmission range=200m (b) transmission range=300m

**Fig. 6.** Average CH duration versus maximum MH speed.

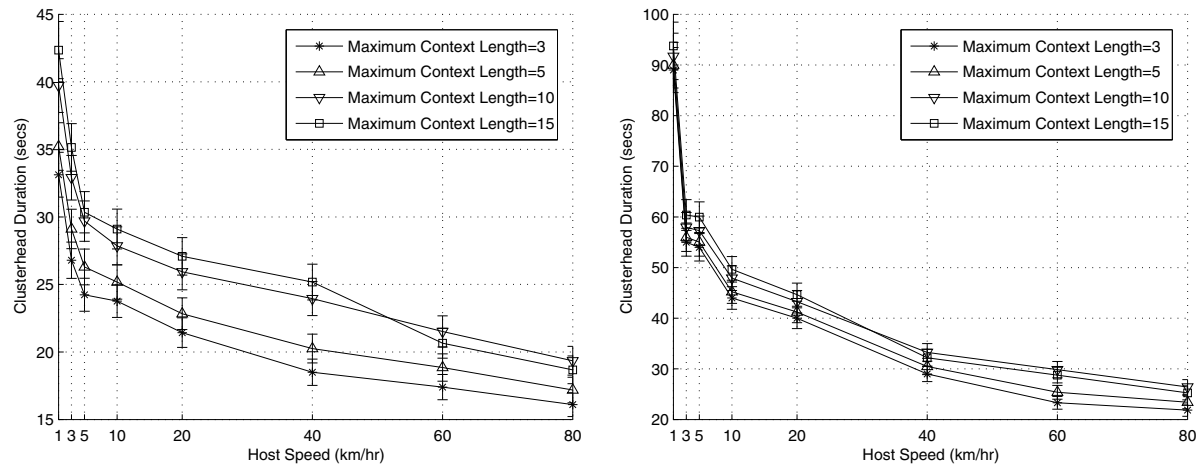



(a) transmission range=200m (b) transmission range=300m

**Fig. 7.** Average CH duration versus maximum MH speed for various values of $k$.

experiments, the maximum context length is set again to 10. From the results of Fig. 7, it is clear that the lifetime length of CHs drops as the value of $k$ increases. This is quite reasonable since when the value of $k$ is relatively large many quite different neighborhoods are considered the same and hence the MobHiD algorithm loses some of its predictive capability. This difference in performance is conspicuous in high MH speed values as then the neighborhood of each MH is changing faster and thus it is harder to predict without sufficient stored information. However, the high discrimination among patterns of neighborhood offered in the case $k = 1$ cannot always easily been exploited and thus the MobHiD algorithm has lower performance in the case $k = 1$ compared with the case $k = 3$ when MHs are fast moving. As explained in Section 2, the lower performance can be attributed to the fact that when keeping all possible neighborhoods in detail in the trie structure, it is more difficult to obtain reliable statistical evidence since the counters of trie nodes have not accumulated enough and have low and hence statistically insignificant values. This is not a serious problem when the MH speed is low since the neighborhoods of MHs are easily predicted in this case but in the range of high speed values, the problem of not sufficient statistical data is getting worse and hence the lower performance for $k = 1$.

Fig. 8 shows the performance of the MobHiD algorithm for various values of maximum context length used for estimating the future appearance probability of the current neighborhood. For this set of experiments, $k$ is equal to 3. We can easily see that the clusterhead duration is larger when using longer neighborhood contexts for making probability calculations. Basically, the same discussion as above holds in this case too. Longer contexts correspond to higher order statistical Markov models [16] and hence they possess higher discriminative as well as predictive power in comparison to shorter contexts. Again, for high values of MH speed, an excessively long context (context length = 15) suffer from unreliable statistical data. As result, for this range of speed values, the context of length 15 does not yield the highest performance.




(a) transmission range=200m (b) transmission range=300m

**Fig. 8.** Average CH duration versus maximum MH speed for various neighborhood context lengths.

*5.1.3. Reaffiliation rate*

Fig. 9 shows the rate of reaffiliations encountered in the five algorithms as a function of the maximum MH speed (Fig. 9a) and then as a function of the maximum transmission rate of each MH (Fig. 9b). Again, in these tests we assume 50 hosts moving in an area of $1000 \times 1000$ m$^2$.

With increasing MH speed, increase in reaffiliation rate should be expected as high mobility spoils the existing clustering structure sooner. It is more likely that MHs will move away form their clusterheads and reaffiliate to a different cluster. In comparison to the other algorithms, MobHiD demonstrates the lowest reaffiliation rate. In our algorithm, a MH does not change cluster even when there are other clusterheads in the vicinity. The only event that could trigger reaffiliation of a node to another cluster is the loss of contact with the current clusterhead. However, during cluster formation, each MH selects the clusterhead with the highest probability of being its neighbor for a long time and thus the probability of MH reaffiliation is minimized accordingly.

In Fig. 9b, an increase in reaffiliation rate is evident in the mid-range of values of transmission range especially for LI, HD and GDMAC algorithm. When the transmission range of MHs is relatively small, the neighborhood of each MH is relatively sparse and thus there is a low chance of reaffiliating to another clusterhead. On the other hand, when the transmission range is large, a high percent of MHs are neighbors with each other and the mobility of MHs does not affect the neighborhood of each MH to a large extent. In contrast, for middle values of transmission range, the neighborhood of each MH is changing more frequently and thus reaffiliations are highly possible.

However, this trend is largely attenuated in MobHiD algorithm since, as is explained above, each MH selects as its clusterhead the clusterhead with the highest probability of being in the neighborhood of MH for a long time. Hence, the event of a MH losing contact with its clusterhead does not happen frequently and thus the reaffiliation rate does not rise greatly in the mid-range of the values of transmission range despite the fact that the neighborhoods of MHs are highly volatile for these values of transmission range. To a lesser extent, the same good feature is observed in MOBIC algorithm. Similarly to MobHiD, this algorithm takes into account the mobility of neighbors around each MH in order to decide the most suitable clusterheads. In contrast, the three other algorithms are not based on the relative mobility of MHs, i.e. the mobility of a MH with respect to other MHs, when deciding clusterheads and hence their low effectiveness when the network topology is highly volatile.

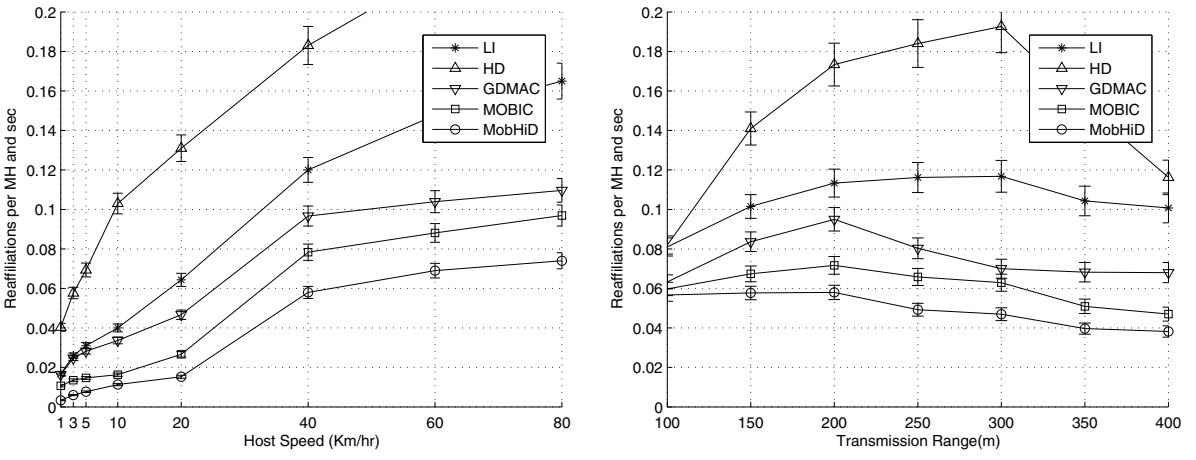
*5.1.4. Control message overhead*

In another set of experiments, we evaluated the control message overhead of the clustering algorithms for various values of MH maximum speed (Fig. 10). In our experiments, HELLO messages were not considered as the exchange of these messages between MHs for discovering their neighbors is almost standard in all clustering algorithms. We tested the algorithms for two values of transmission range and we also assumed that 50 hosts are moving over a terrain of a $1000 \times 1000$ m$^2$.

Clearly, MobHiD exhibits the lowest number of control messages sent from each host per second in comparison to the other techniques. This is mainly due to the fact that MobHiD builds stable clusters and so reclustering events which produce the main volume of control messages are relatively sparse. In addition, limiting reclustering in a certain area of the network further reduces the control message sent or received during reclustering. On the other hand, the heavy control message overhead of HD and LI techniques is another piece of evidence for their rather unstable clusters.

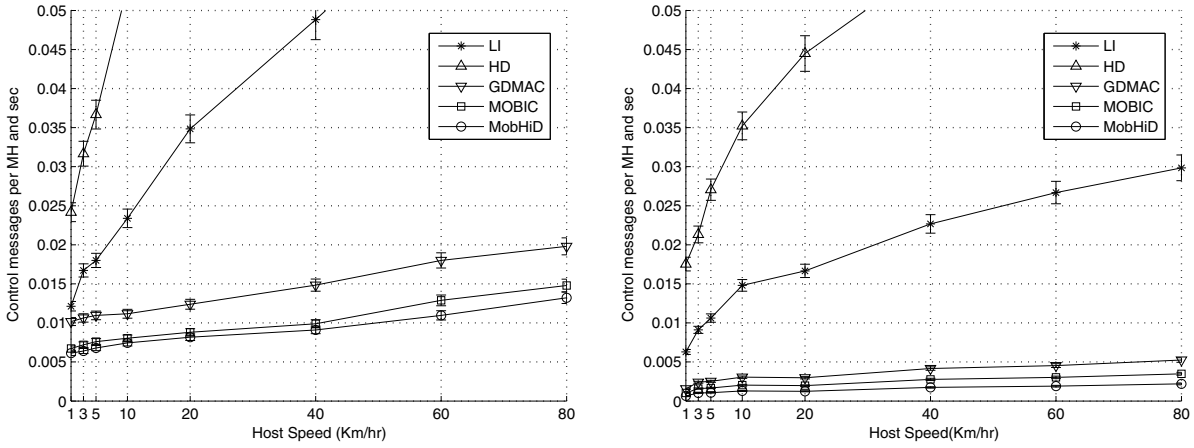
*5.1.5. Local memory requirements*

In this set of tests (Figs. 11–14), we examined the growth of the trie structure in each MH over all simulation time. Specifically, we measure the number of the nodes in the trie as well as the number of neighborhoods in the structure. Recall that



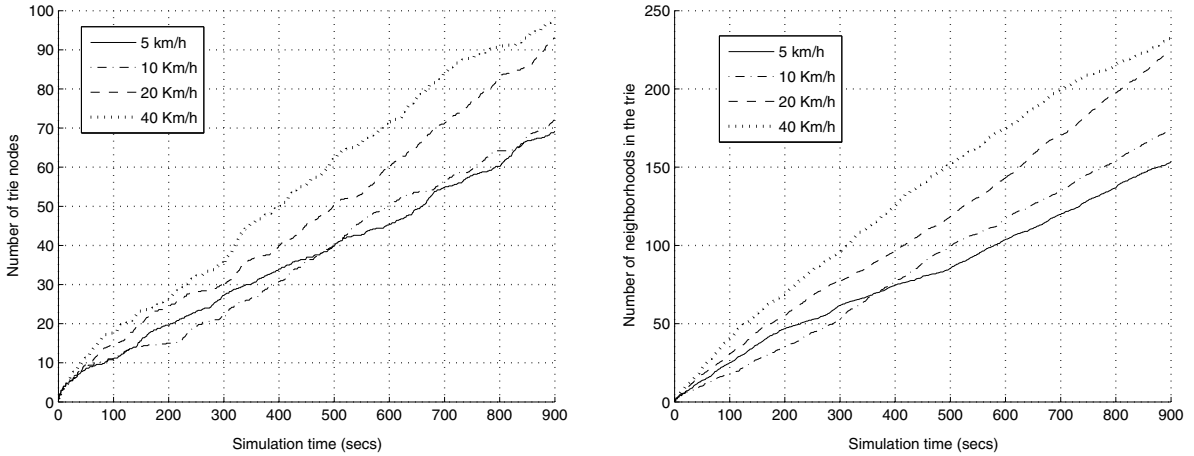

(a) transmission range=200m (b) max speed=40km/h

**Fig. 9.** Average number of reaffiliations per MH and second versus maximum MH speed and MH transmission range.



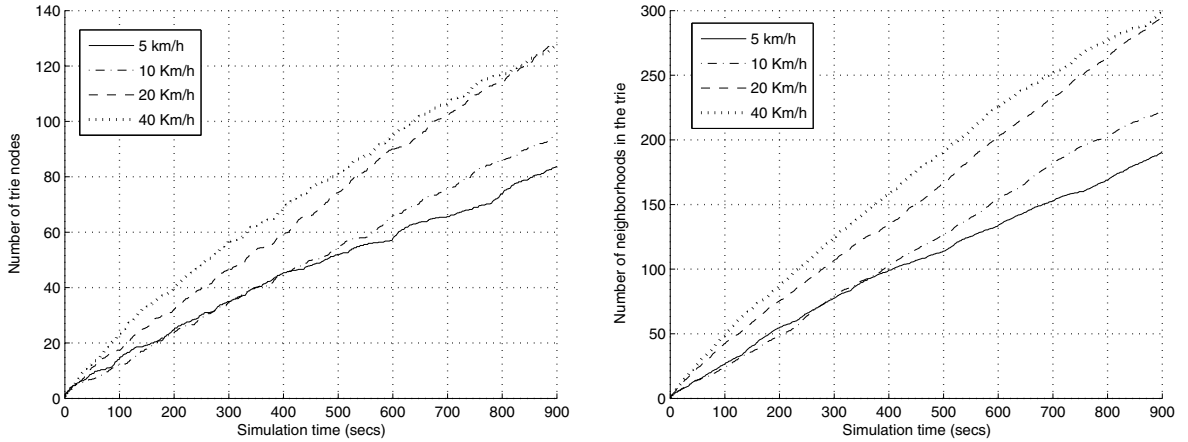

(a) transmission range=200m (b) transmission range=300m

**Fig. 10.** Average number of control messages per MH and second versus maximum MH speed.



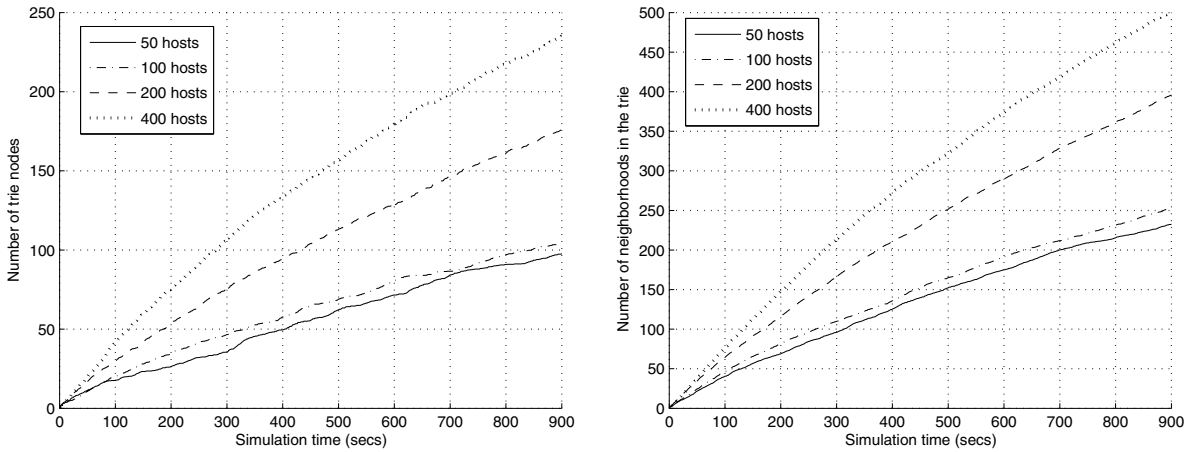

(a) Number of trie nodes (b) Number of neighborhoods in the trie

**Fig. 11.** Growth of trie structure over simulation time for transmission range at 100 m and 50 MHs.



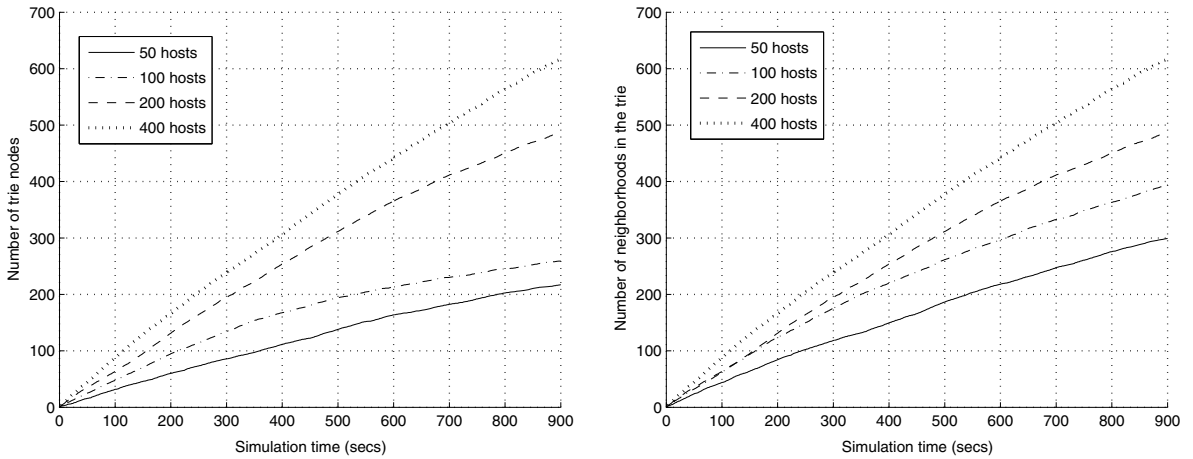

(a) Number of trie nodes (b) Number of neighborhoods in the trie

**Fig. 12.** Growth of trie structure over simulation time for transmission range at 200 m and 50 MHs.




(a) Number of trie nodes (b) Number of neighborhoods in the trie

**Fig. 13.** Growth of trie structure over simulation time for transmission range at 100 m and maximum host speed at 40 km/h.




(a) Number of trie nodes (b) Number of neighborhoods in the trie

**Fig. 14.** Growth of trie structure over simulation time for transmission range at 200 m and maximum host speed at 40 km/h.

with neighborhood merging technique, more than one neighborhoods can be stored in the same trie node. Again, we set the slack variable $k$ equal to 3 and thus each trie node can store only neighborhoods that differ in at most three neighbors. Apparently, the number of neighborhoods in the trie is always at least the number of nodes in the trie. This fact is clearly depicted in all figures of this section.

Note also that the number of neighborhoods in the trie is an indication of the storage savings coming from the use of trie structure. More precisely, since the actual simulation time is 15 min (=900 s) and each MH learns about its neighbors every second through the periodic exchange of HELLO messages, the total number of neighborhoods seen by a MH will be 900. Clearly, if we used a simple list for storing this history of neighborhoods, the length of the historical record would be 900. If, instead, we use a trie structure for the organization of this record, the number of stored neighborhoods is only a fraction of the total number of neighborhoods a MH has seen during simulation. This is because the trie structure stores only once recurring patterns in the neighborhood history. Obviously, a lower bound of the number of neighborhoods stored in the trie is the number of different neighborhoods seen during simulation.

In the figures of this section, we have not drawn the confidence intervals around each point. Since, 900 points should be drawn in these graphs, the resulting figures would be cluttered if confidence intervals were included. For 95% confidence intervals, the maximum variation from the average value that has been observed was 5.1% of the average value.

Fig. 11 shows the growth of trie structure as simulation proceeds for various values of maximum MH speed and when the transmission range of each MH is 100 m and the number of MHs is 50. In this figure as well as in all figures of this section, each point in graphs is the average of the structure size over all MHs. As expected, the size of the trie is increasing as the speed of MHs is increasing. This is because in a scenario of fast moving hosts, the neighborhoods of MHs are changing

frequently and the recurring patterns in neighborhood history are more sparse. However, even in the case that MHs are moving with maximum speed at 40 km/h, the number of neighborhoods in the trie is only a small fraction of the total number of neighborhoods that a MH has seen. This is also indicative of the good predictive capability of the trie structure.

Fig. 12 shows results for the same simulation scenario except that now the transmission range of each MH is 200 m. We can notice an increase in the size of the trie in comparison to the size of the trie in Fig. 11. Now, more MHs can be neighbors of a MH and thus longer neighborhoods are stored in the trie structure. Since it is less likely that two long neighborhoods are exactly the same, fewer recurring patterns are arising in the neighborhood history. As a result, an increased number of neighborhoods should be stored in the trie. However, notice that even at the worst case scenario where MHs are moving with maximum speed of 40 km/h, only one third of the total number of neighborhoods has been stored in the trie. This again shows the high predictive capability of our technique.

Figs. 13 and 14 show the rate of increase in the size of the trie for various numbers of MHs, for maximum host speed at 40 km/h and for two values of transmission range, namely 100 m and 200 m. In this scenario, MHs are fast moving since their movements are taking place in a terrain of $1000 \times 1000$ m$^2$. In addition, the terrain is densely populated, specially in the scenarios with 200 or 400 MHs. Taking also into account that MHs are moving according to RW model where there is weak correlation among the movements of MHs, we can easily conclude that the scenario in Figs. 13 and 14 is an adverse scenario for our method. So, for 400 MHs and transmission range at 200 m, we can see that the number of neighborhoods stored in the trie rises to about 2/3 of the total number of neighborhoods seen by a MH. However, for lower number of MHs, this size is much smaller and thus we can see that for reasonable host density, our method fares very well.

#### 5.1.6. Computational overhead

In this set of simulation tests, we measure the computational overhead of the MobHiD algorithm. Specifically, we measure the instructions required for accessing the trie structure and potentially inserting a new neighborhood in the structure. We also count the number of instructions needed for estimating neighborhood probabilities when reclustering is required.

As has been explained in Section 2, the trie structure is periodically accessed every *BI* seconds where *BI* is the broadcast interval of HELLO messages. The current neighborhood is searched in the children of the current node of the trie and if it is not found, it is inserted a new child of the node. Apparently, the main operation executed in this periodic access of the trie structure is the comparison between two neighborhoods. Recall that when the ids of neighbors are kept sorted in each neighborhood, testing whether two neighborhoods contain the same MHs or not requires $l + m - 1$ integer comparisons at most where $l$ and $m$ is the number of MHs in these two neighborhoods, respectively. Note that the worst case is always occurs when the two neighborhoods are the same since in this case all the hosts in these two neighborhoods should be compared in order to ascertain equality. In contrast, when neighborhoods are not the same, we may find out difference earlier before examining the neighborhoods in their entirety.

When reclustering should be performed, each MH calculates the probability of its current neighborhood being constantly present from then on. This computation is carried out in a subtree of the trie. There are two kinds of operations. First, a number of integer comparisons are required in order to find in which neighborhoods in the subtree each MH of the current neighborhood appears. then, we need a number of arithmetic operations (mainly additions and some multiplications) in order to estimate the above probability. Apparently, the computational overhead from these operations is largely determined by the frequency of reclustering events as well as the size of the sub-trie used in these calculations.

So, in this set of experiments we have measured the above operations executed during the whole simulation. We have averaged this number over all MHs and finally divided this average with the total duration of each simulation test in order to find the number of instructions executed per second in our experiments. As in most experiments concerning RW model, we use a maximum context length of 10 successive neighborhoods again.

In the first experiment (Fig. 15), we assume 50 MHs moving on a terrain of $1000 \times 1000$ m$^2$ and we measure the number of instructions per second as the maximum host speed is increasing and for different values of host transmission range. The main observation here is that the computational overhead is increasing with higher MH speed as well as with higher values of transmission range. As host speed is increasing, reclustering is occurring with higher frequency and hence probability calculations should be executed more frequently accordingly. Also, when the transmission range is large, each neighborhood may contain a lot of neighbors and thus neighborhood comparison is more expensive to execute. This explains the increased number of instructions for higher values of transmission range.

In the second set of experiments (Fig. 16), we measure the number of instructions per second with increasing number of hosts and for various values of transmission range when the maximum host speed is at 40 km/h. Again with higher values of transmission range, neighborhood configurations are getting longer and hence neighborhood comparisons are taking longer to execute. The same effect is also observed when the number of MHs is increasing since in this case too, neighborhoods may include a large number of MHs. In addition, when the number of hosts is increasing reclustering events are more frequent. As a net result, the number of instructions executed per second is a monotonically increasing function of the number of MHs moving on the terrain.

### 5.2. Reference point group mobility model

In the second set of simulations, we assume that the MHs are moving according to the reference point group mobility (RPGM) model [28]. In this model, MHs are separated in groups, and all hosts belonging to the same group are moving ran-

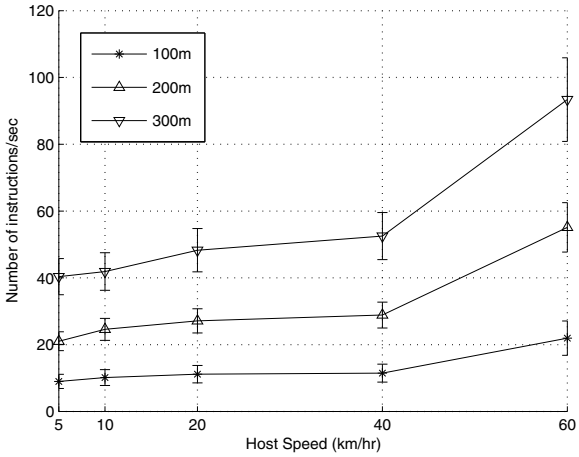



**Fig. 15.** Number of instructions versus maximum MH speed for different values of transmission range.

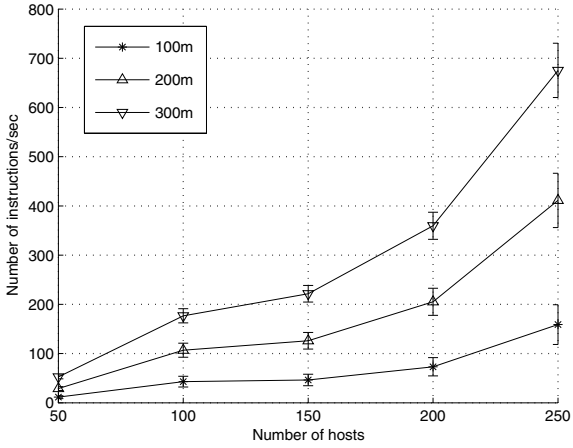


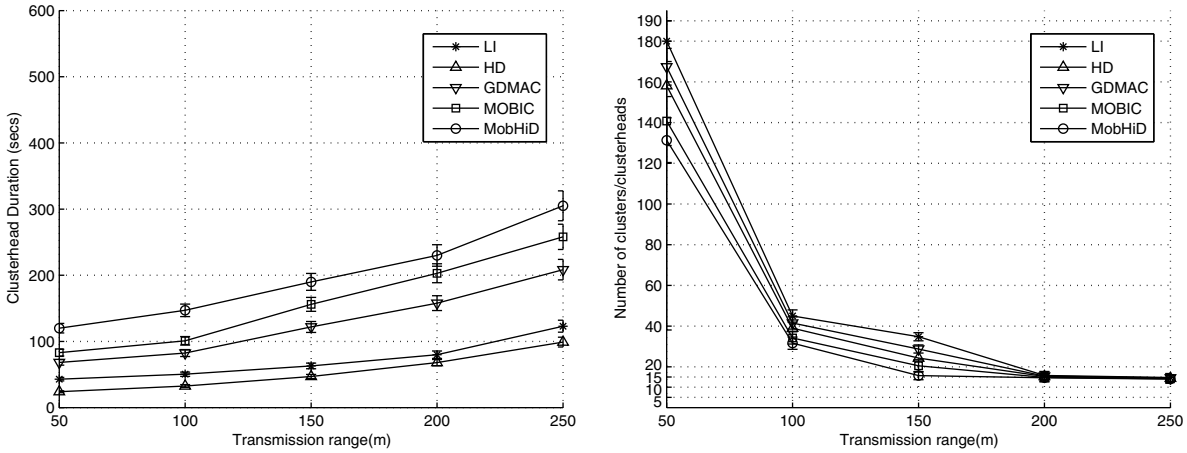

**Fig. 16.** Number of instructions versus number of MHs for different values of transmission range.

domly within a certain distance of a reference point. Each group has a different reference point and in our simulation setting we assume that these points are moving according to the RW model with maximum speed at 40 km/h. So, the final position of each MH is determined from the current position of its reference point as well as from the random position of the MH around its reference point. In our tests, we assume that each MH is 150 m at most away from its reference point (*group radius* = 150 m). We also assume that the groups are moving on a terrain of size $5000 \times 5000$ m$^2$.

*5.2.1. Number of clusters and cluster lifetime*

Fig. 17 shows the duration of elected clusterheads and the number of clusters/clusterheads for various values of transmission range and for a scenario where 750 MHs are organized in 15 groups with 50 hosts per group. Fig. 17a clearly shows that MobHiD creates the most stable clusters in comparison to other techniques. It is apparent also that the clusterhead duration observed for all techniques is much longer than the duration of clusterheads in RW model since movements now are more predictable and the local network topology around each MH does not present frequent changes.

Fig. 17b shows that the number of clusters is decreasing as the transmission range of MHs is increasing. When transmission range is lower than group radius (=150 m), not all the MHs of a group are direct neighbors and so more than one clusters per group should be created. When the transmission range is 150 m, MobHiD almost attains the "ideal" number of clusters, 15, which simply means one cluster per group. When the transmission range of each MH assumes values higher than group radius, MHs of nearby groups are becoming neighbors and clusters including MHs from different groups might be created. In this case, the total number of clusters may be lower than the number of groups. However, this kind of clustering aroused only few times in our tests. Thus for relatively high values of transmission range, the average number of clusters that has been obtained from our tests is just under the value of 15 clusters for almost all the clustering algorithms. In any case, the number of clusters formed by MobHiD for various values of transmission range is always close to the minimum number of clusters necessary for covering the MHs of all groups.



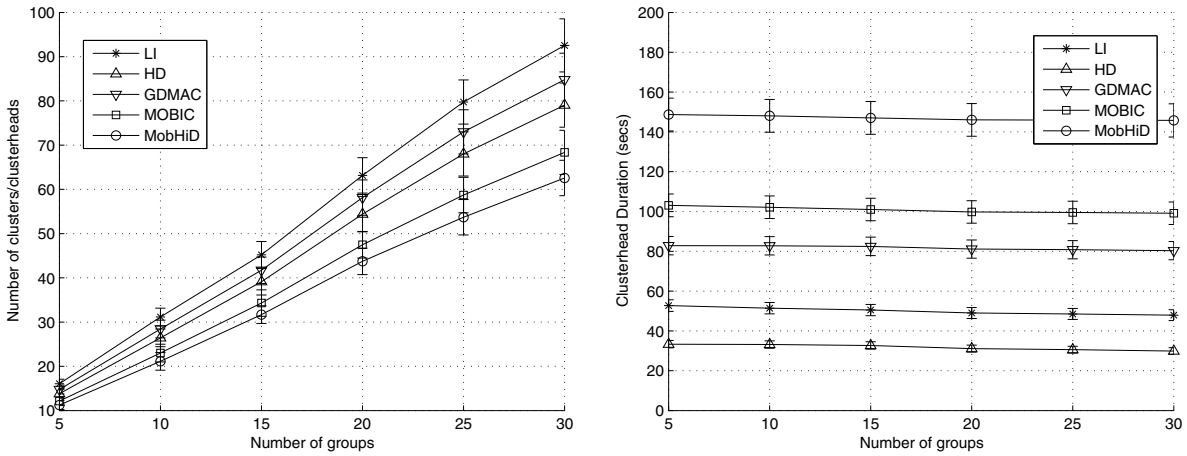

(a) Clusterhead duration (b) Number of clusters

**Fig. 17.** Performance comparison for increasing transmission range under the RPGM model.

This fact is also clearly depicted in Fig. 18a, where the numbers of clusters formed by each clustering algorithm are compared for a varying number of MH groups. Note that transmission range of MHs is now 100 m, i.e. lower than the group radius. Thus, more than one clusters may be needed for covering the MHs of a group. For different numbers of groups, MobHiD creates the lowest number of clusters and always close to the minimum. MOBIC yields the next lower number of clusters. Like MobHID, MOBIC takes into account the stability of the neighborhood of each MH in order to elect clusterheads. Within a group, MHs that maintain their neighbors for a long time are the hosts located near the reference point of the group. The corresponding clusters cover a large number of hosts especially when compared with clusters near the periphery of the group. Thus, election of clusterheads near group reference points leads also to a small number of clusters overall.

In contrast, LI and GDMAC techniques form much higher number of clusters on average. The main reason for this is that the random placement of MHs inside each group around the reference point may create many local minimums or maximums in the clustering criterion of LI and GDMAC algorithm and correspondingly many clusterheads and hence clusters. The same problem also arises in HD technique. However, the criterion used for clusterhead election in HD algorithm helps reduce the number of clusters in comparison to the LI and GDMAC algorithm.

Finally, in Fig. 18b the duration of elected CHs for each algorithm is drawn versus the number of MH groups. Now, the transmission range of MHs is set at 100 m. For this value of transmission range, the groups of MHs are well separated most of the time and thus the clusterhead duration in each algorithm is mainly determined by the movements of MHs within each group. Thus, in all algorithms the clusterhead duration varies only a little as the number of groups is increasing. Specifically, the duration of clusterheads is slightly decreasing with increasing number of groups. With many groups of MHs moving around, the probability that some groups are overlapping is increasing and thus the local network topology around each




(a) Number of clusters (b) Clusterhead duration

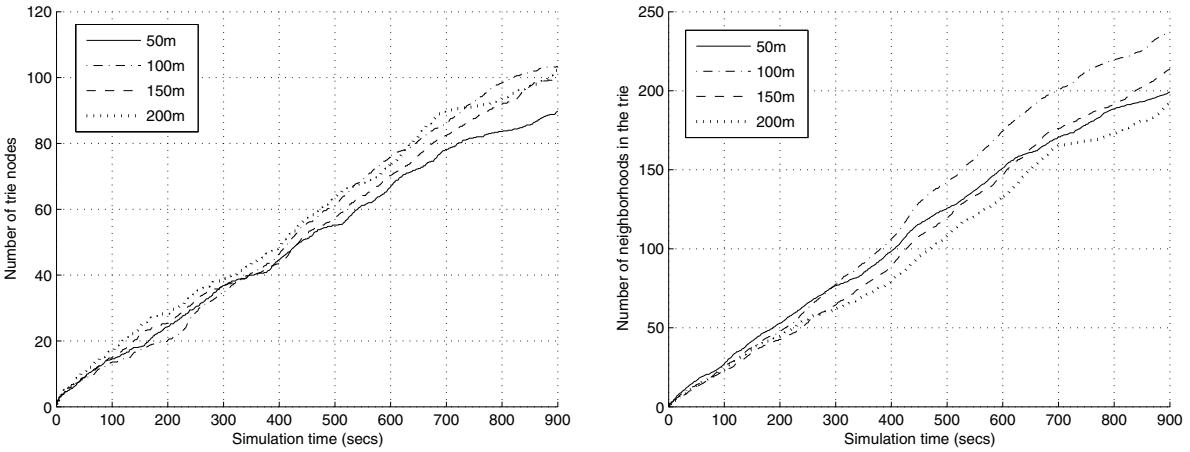
**Fig. 18.** Performance comparison for increasing number of groups under the RPGM model.

MH is getting less stable. However, as Fig. 18b shows, MobHiD forms the most stable clusters in comparison to other techniques.

*5.2.2. Local memory requirements*

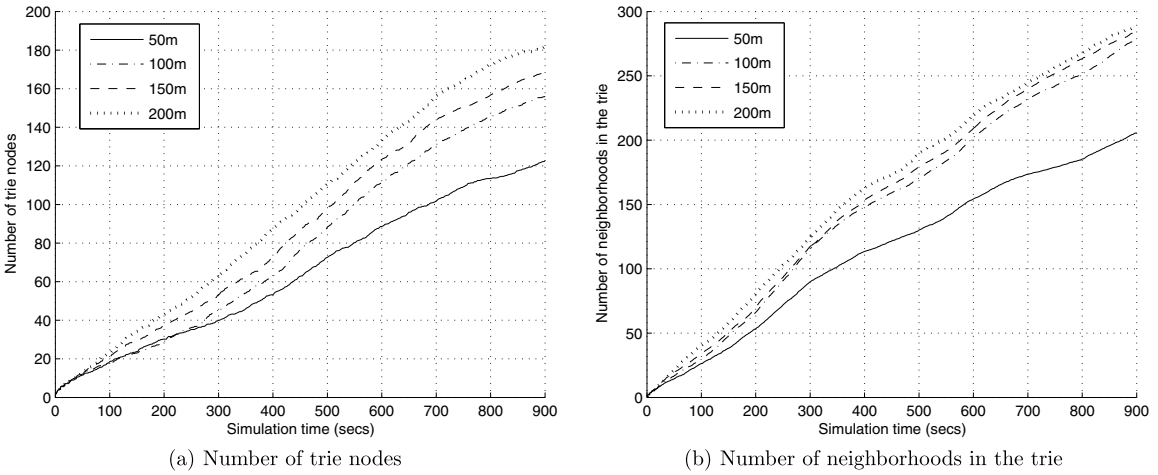
In this section, we present results about the growth of the trie structure during simulation when MHs are moving according to the RPGM model. Specifically, we consider two scenarios with 15 and 30 groups of MHs, respectively, and with each group moving with maximum speed at 40 km/h. Again, the deployed terrain is of size $5000 \times 5000$ m$^2$, each group consists of 50 MHs and the maximum distance of a MH from its corresponding group reference point is 150 m. For the same reasons as those mentioned in Section 5.1.5, we have not included confidence intervals in the graphs so as to keep figures uncluttered. For 95% confidence intervals, the maximum variation around the average value that has been measured in simulations is about 3.9% of the average value, that is the variation now is lower than that of the RW model.

In Figs. 19 and 20, we have plotted the number of nodes as well the number of neighborhoods stored in the trie during simulation for various values of transmission range of MHs. Comparing the results in the two figures, we can see that the size of the structure increases similarly for the two different numbers of groups. This is something to be expected because most of time the neighbors of a MH are MHs from its own group. Different groups only occasionally are met and at these moments new neighbors may appear around a MH. As a result, new neighborhoods should be inserted in the trie since these specific neighborhoods have not probably met previously. Apparently, in the case of 30 groups, the probability of two groups meeting each other is higher than in the case of 15 groups. That is why we see an slightly increased number of nodes and neighborhoods in the trie in the scenario of Fig. 20 where 30 groups are moving over the terrain.




(a) Number of trie nodes (b) Number of neighborhoods in the trie

**Fig. 19.** Growth of trie structure over simulation time for 15 groups and maximum group speed at 40 km/h.




(a) Number of trie nodes (b) Number of neighborhoods in the trie

**Fig. 20.** Growth of trie structure over simulation time for 30 groups and maximum group speed at 40 km/h.

Figs. 19 and 20 also show that the number of neighborhoods stored in the trie is a small fraction of the total number, 900, of neighborhoods seen by a MH during simulation. Note that the lowest number of neighborhoods in the trie in the scenario of 30 moving groups results when the transmission range is 50 m. This is explained by the fact that for this small transmission range, a MH does not have a lot of neighbors and thus a match of a new neighborhood with a previously stored neighborhood in the trie is more likely than in longer neighborhood configurations arising in higher transmission ranges. In addition, in the scenario of 30 groups, a very large number of MHs, namely 1500 MHs, is moving. Thus, the probability of group overlapping is not small in this case, in particular if we consider that the group radius is 150 m. Therefore, for higher values of transmission range, group overlapping has as a result the higher variability of neighborhoods and hence the increased number of neighborhoods in the trie.

Finally, in all the graphs of the two figures, the number of neighborhoods is about twice the number of nodes in the trie. This simply means that each trie node stores two neighborhoods on average. Since the number of nodes and neighborhoods in the trie is much lower than the number of neighborhoods a MH has seen in total, this low number of neighborhoods per trie node shows the high predictive capability of our technique when RPGM model is assumed. This could also be a suggestion for lowering the value of slack variable $k(=3)$ of our neighborhood merging technique without incurring any performance loss.

### 5.2.3. Computational overhead

In this section, we assess the computational overhead of the MobHiD algorithm as we did for the RW model. In Fig. 21, we plot the number of instructions executed in our algorithm with increasing maximum group speed and for various values of transmission range. We also assume that there are 750 MHs organized in 15 groups. Now, we can notice an increase in the number of instructions both when the transmission range is increasing and when groups are moving with higher speed. As has been explained earlier for the RW model, when transmission range is increasing, the neighborhood of a MH may include more hosts and thus the operation of neighborhood comparison required in trie computations is getting more expensive.

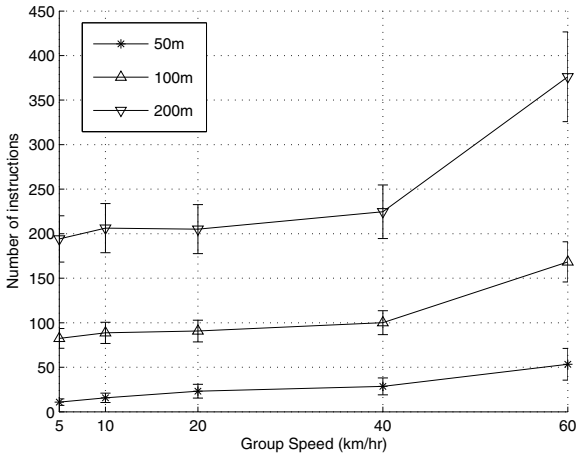



**Fig. 21.** Number of instructions versus maximum group speed for different values of transmission range.

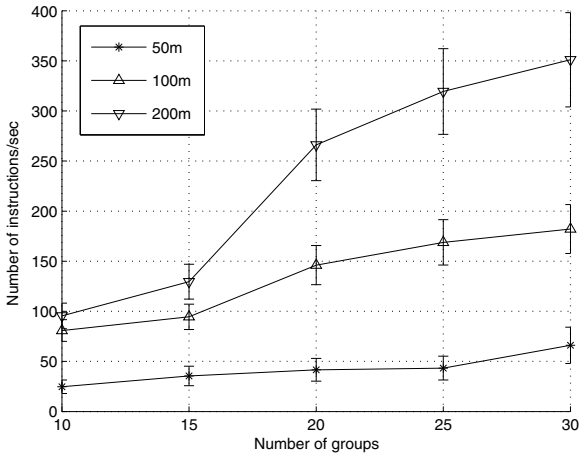



**Fig. 22.** Number of instructions versus number of groups for different values of transmission range.

In the second set of tests in Fig. 22, we measure the computational load of the MobHiD algorithm for an increasing number of groups. Again each group contains 50 hosts and it is also assumed that each group is moving with a maximum speed of 40 km/h. Clearly, computational load is getting higher as the number of groups is increasing. The reason is twofold. First, a large number of groups implies also a large number of MHs in total and thus longer neighborhoods for each MH. Also, with higher number of groups, group overlapping is more likely and hence the condition for reclustering may arise more frequently since clusters are more easily disturbed in this case.

As for the increase in the number of instructions when the transmission range is increasing, we can repeat the argument of the first paragraph. Higher transmission range has as a result "longer" neighborhoods and hence more expensive neighborhood comparisons in terms of execution time.

## 6. Conclusions

In this paper, we presented a mobility-aware clustering scheme which uses well known information theoretic techniques for reliably estimating the future mobility of MHs. The right prediction enables the formation of clusters that are highly resistant to the topological changes of the ad hoc network due to host mobility. For measuring the mobility, we do not use special purpose hardware such as GPS but the mobility of each host is inferred from how different is the neighborhood of the MH over time. In this way, we take into account the strong correlation that usually exists among the movements of neighboring MHs, thereby achieving accurate estimation of future host mobility. Then, by combining our mobility prediction scheme with the highest degree clustering technique, we proposed a distributed algorithm that builds a small and stable virtual backbone over the ad hoc network. Our results have been verified through simulation experiments, which showed the high performance of our technique in practice.

## Appendix A

### A.1. Pseudocode of cluster maintenance

{Cluster maintenance for MH $u$}
$L$ = the maximum number of CHs in the neighborhood before triggering reclustering
*cl_extent* = the radius of the affected area in terms of cluster hops
$T$ = the upper bound of the message transfer delay over a single link
*rcl_events* = the set of the current reclustering events

**while** TRUE **do**
  *state* = INACTIVE
  *clhd_nghs* = the set of CHs in the neighborhood of $u$
  $N$ = the set of neighbors of $u$
  *LWN* = the set of neighbors that have larger weight than that of $u$
  *accept_msg_sndrs* $= \{\}$ {the ids of MHs sending ACCEPT messages to $u$}
  *decline_msg_sndrs* $= \{\}$ {the ids of MHs sending DECLINE messages to $u$}
  *clhd_msg_sndrs* $= \{\}$ {the ids of MHs sending CLUSTERHEAD messages to $u$}
  *join_msg_sndrs* $= \{\}$ {the ids of MHs sending JOIN messages to $u$}
  *rcl_events* $= \{\}$ {no reclustering in progress}
  **if** $u$ is a CH **then**
    *cl_id* $= u$
    *cluster_members* = the group of the MHs in the cluster of $u$
  **else**
    *cl_id* = the CH of $u$
    *cluster_members* $= \{\}$
  **end if**
  *waiting_period* = FALSE {TRUE when $u$ waits for the RECLUSTER message to reach all the affected clusters}
  **while** state of $u$ is INACTIVE **do**
    **if** *cl_id* $= u$ **then**
      {u is a clusterhead}
      **if** $|$ *clhd_nghs* $| > L$ **then**
        send RECLUSTER($u$, *cl_extent*) to the neighbors of $\{u\}$
        *rcl_events* $=$ *rcl_events* $\cup u$
        *state* = ACTIVE
        set *waiting_period* = TRUE and wait for $3 \cdot (\textit{cl\_extent} + 1) \cdot T$ seconds
        *waiting_period* = FALSE
      **end if**

```
        else
            {ordinary MH}
            if u can't communicate with its CH then
                if | clhd_nhbs | > 0 then
                    {affiliate with another cluster}
                    cl_id = v {v is the CH in clhd_nhbs with the highest probability of being neighbor of u in the future}
                    send JOIN(u,v) to v
                else
                    cl_id = u {u becomes a CH}
                end if
            end if
        end if
    end while
    cl_id = NULL {start reclustering}
    cluster_members = {}
    if | LWN | ≠ 0 then
        if | clhd_msg_sndrs ∪ join_msg_sndrs | < | LWN | then
            send INVITE to all nodes in LWN − clhd_msg_sndrs − join_msg_sndrs
        end if
        wait until accept_msg_sndrs ∪ clhd_msg_sndrs ∪ join_msg_sndrs ∪ decline_msg_sndrs = LWN
        wait until clhd_msg_sndrs ∪ join_msg_sndrs = LWN − decline_msg_sndrs
        if | clhd_msg_sndrs | > 0 then
            cl_id = v {v is the CH in clhd_msg_sndrs with the highest probability of being neighbor of u in the future}
            send JOIN(u,v) to all neighbors
        else
            cl_id = u
            send CLUSTERHEAD(u) to all neighbors
        end if
    else
        {| LWN | = 0}
        cl_id = u
        send CLUSTERHEAD(u) to all neighbors
    end if
end while


{Callback functions}
on receiving RECLUSTER(w, cl_extent) from MH v
if w ∉ rcl_events then
    {It is the first notification for the reclustering initiated by CH w}
    rcl_events = rcl_events ∪ {w}
    if state = INACTIVE or waiting_period then
        if cl_extent ≠ 0 then
            if cl_id = u then
                {u is a CH}
                send RECLUSTER(w, cl_extent − 1) to all MHs in N − {v}
            else
                send RECLUSTER(w, cl_extent) to all MHs in N − {v}
            end if
            state = ACTIVE
            if waiting_period then
                wait_secs = the remaining waiting time in seconds
            else
                wait_secs = 0
            end if
            set waiting_period = TRUE and wait for max(3 · (cl_extent + 1) · T, wait_secs) seconds
            waiting_period = FALSE
        end if
        if cl_extent = 0 and cl_id = v then
            {u is an ordinary MH receiving RECLUSTER message from its CH}
            set state = ACTIVE
```

```
            waiting_period = FALSE
        end if
    end if
end if
endon

on receiving INVITE message from MH v
if state = ACTIVE then
  send ACCEPT message to MH v
else
  send DECLINE message to MH v
end if
endon

on receiving ACCEPT message from MH v
if state = ACTIVE and v in LWN − clhd_msg_sndrs − join_msg_sndrs then
  accept_msg_sndrs = accept_msg_sndrs ∪ {v}
end if
endon

on receiving DECLINE message from MH v
if state = ACTIVE and v in LWN − clhd_msg_sndrs − join_msg_sndrs then
  decline_msg_sndrs = decline_msg_sndrs ∪ {v}
end if
endon

on receiving CLUSTERHEAD(v) message from MH v
if state = ACTIVE and v in LWN − join_msg_sndrs then
  clhd_msg_sndrs = clhd_msg_sndrs ∪ {v}
end if
endon

on receiving JOIN(v,w) message from MH v
if state = ACTIVE and v in LWN − clhd_msg_sndrs then
  join_msg_sndrs = join_msg_sndrs ∪ {v}
end if
if cl_id = u and w = u then
   {u may receive JOIN message during reclustering and when a single MH wishes to affiliate with the cluster of CH u}
   cluster_members = cluster_members ∪ {v}
end if
endon
```

### A.2. Average length of the longest chain path

In this section, we give a loose upper bound of $E(l_{\max})$, that is the average value of the maximum length of a chain path. We first prove the following lemma.

**Lemma 6.** *Let $A, B, C, D$ be four MHs successive on a chain path. MH D cannot be within the transmission range of MH A.*

**Proof.** Suppose that all three MHs, $B, C, D$, are within the transmission range of MH $A$. There are two cases to consider. First, MH $A$ decides to become a clusterhead and so it sends the CLUSTERHEAD message to all its neighbors (Fig. A.1a). In this case, node $D$ can decide immediately after and it joins the cluster of MH $A$. Consequently, it does not wait for the decision of MH $C$ and thus the path $A, B, C, D$ is not a chain path.

In the second case, MH $A$ decides to affiliate with a cluster and then sends a JOIN message to all its neighbors. From the assumption of lemma, node $B$ can decide immediately that it can become CLUSTERHEAD and so it sends a CLUSTERHEAD message to all its neighbors. So, MH $D$ does not have to wait for the decision of node $C$, that is again path $A, B, C, D$ cannot be a chain path.

So, we have proved that MH $D$ cannot be within the transmission range of MH $A$ or in other words, at most tree successive MHs on a chain path can be within the transmission range of each other.

Now, let $MH_1, MH_2, \ldots MH_k$ be $k$ MHs with monotonically decreasing (MD) weights. Let $E_i$ be the event that the first $i$ MHs form a chain path. Now, we have that

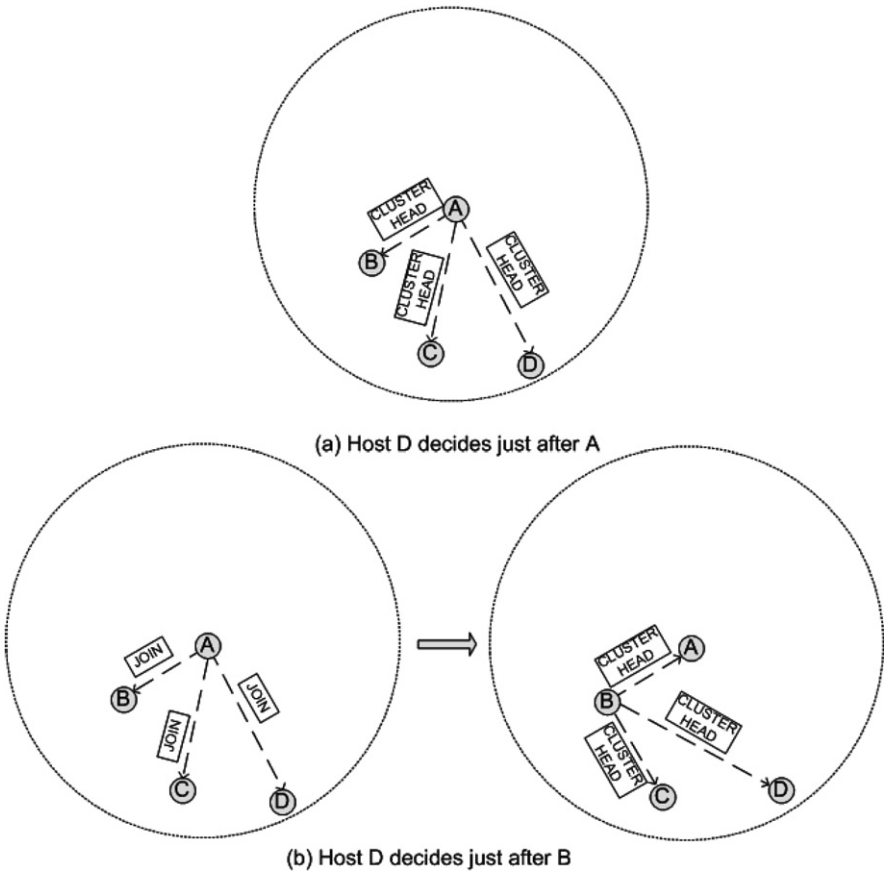



**Fig. A.1.** MH $D$ cannot be within the transmission range of MH $A$.

$$P(E_k) = P\left(\bigcap_{i=1}^{k} E_i\right) \Rightarrow$$
$$P(E_k) = P(E_1) \cdot P(E_2 \,|\, E_1) \cdot P(E_3 \,|\, E_1, E_2) \cdots P(E_k \,|\, \cap_{i=1}^{k-1} E_i).$$

However, it holds that $P(E_k \,|\, \cap_{i=1}^{k-1} E_i) \leqslant P(E_j \,|\, E_{j-1}, E_{j-2}, E_{j-3})$ for $(j = 4, \ldots, k)$. So, we get

$$P(E_k) = P(E_1) \cdot P(E_2|E_1) \cdot P(E_3|E_1, E_2) \cdot \prod_{j=3}^{k} P(E_j|E_{j-1}, E_{j-2}, E_{j-3}).$$

Since, from Lemma 6, there is not any restriction in the placement of three successive MHs except that each MH should be within the transmission range of its predecessor, we can easily see that $P(E_1) = 1,\ P(E_2|E_1) = \pi R^2/D$ and $P(E_3|E_1, E_2) = \pi R^2/D$ where $D$ is the total deployment area and $R$ is the transmission range of each MH. However, again from Lemma 6 for every four successive MHs on a chain path, the fourth MH cannot be within the transmission range of the first one. So, we can deduce that

$$P(E_j \,|\, E_{j-1}, E_{j-2}, E_{j-3}) = \frac{1}{D^4} \int_{C(\mathbf{0},R)} \int_{C(\mathbf{x},R)} \int_{C(\mathbf{y},R)} (C(\mathbf{z},R) - C(\mathbf{0},R)) \mathrm{d}\mathbf{z}\, \mathrm{d}\mathbf{y}\, \mathrm{d}\mathbf{x},$$

where the integration is carried out over all the area of the circles $C(\mathbf{0},R), C(\mathbf{x},R), C(\mathbf{y},R)$ and $C(\mathbf{z},R) - C(\mathbf{0},R)$ is the area of circle $C(\mathbf{z},R)$ not covered by $C(\mathbf{0},R)$. Apparently, it is very difficult to analytically estimate the above expression. So, for convenience we set $p = P(E_j \,|\, E_{j-1}, E_{j-2}, E_{j-3})$.

Now, we have that

$$P\left(\bigcap_{i=1}^{k} E_i\right) \leqslant (\pi R^2/D)^2 p^{k-3}.$$

By assuming distinct weights for MHs, the number of chain paths with $k$ MHs is $\binom{n}{k}$ at most since for each set of $k$ MHs, there is only one permutation of MHs where weights are monotonically decreasing. So, we have that

$$
\begin{aligned}
&E(\text{Number of chain paths with } k \text{ hosts})\\
&\leqslant \binom{n}{k}(\pi R^2/D)^2 p^{k-3} \quad \text{for } k = 3,\ldots,n-1,\\
&\leqslant \binom{n}{2}\pi R^2/D \quad \text{for} \quad k = 2,\\
&\leqslant n \quad \text{for } k = 1.
\end{aligned}
$$

Let $A_k$ the event that there is a chain path with $k$ hosts. By Markov inequality,[3] we have that

$$
\begin{aligned}
&P(A_k) = P(\text{Number of chain paths with } k \text{ hosts} \geqslant 1) \Rightarrow\\
&P(A_k) \leqslant E(\text{Number of chain paths with } k \text{ hosts}) \Rightarrow\\
&P(A_k) \leqslant \binom{n}{k}(\pi R^2/D)^2 p^{k-3} \quad \text{for } k = 3,\ldots,n-1\\
&\qquad \leqslant \binom{n}{2}\pi R^2/D \quad \text{for } k = 2\\
&\qquad \leqslant 1 \quad \text{for } k = 1.
\end{aligned}
$$

Now, we can easily see that

$$
\begin{aligned}
&P(\text{The longest chain path has } k \text{ hosts}) = P(A_k \cap \neg A_{k+1}) \Rightarrow\\
&P(\text{The longest chain path has } k \text{ hosts}) \leqslant P(A_k).
\end{aligned}
$$

Now, if $X$ the number of hosts on a longest chain path, we have that

$$
\begin{aligned}
&E(X) = \sum_{k=1}^{n} k \cdot P(\text{The longest chain path has } k \text{ hosts}) \Rightarrow\\
&E(X) \leqslant \sum_{k=1}^{n} k \cdot P(A_k) \Rightarrow\\
&E(X) \leqslant 1 + 2\binom{n}{2}\pi R^2/D + \sum_{k=3}^{n} k \cdot \binom{n}{k}(\pi R^2/D)^2 p^{k-3} \Rightarrow\\
&E(X) \leqslant 1 + 2\binom{n}{2}\pi R^2/D + p^{-3}(\pi R^2/D)^2 \cdot \sum_{k=3}^{n} k \cdot \binom{n}{k} p^k \Rightarrow\\
&E(X) \leqslant 1 + 2\binom{n}{2}\pi R^2/D + p^{-3}(\pi R^2/D)^2 \cdot \left(\sum_{k=1}^{n} k \cdot \binom{n}{k} p^k - 2\binom{n}{2}p^2 - \binom{n}{1}p^1\right) \Rightarrow\\
&E(X) \leqslant 1 + n(n-1)\pi R^2/D + p^{-3}(\pi R^2/D)^2 \cdot (np(1+p)^{n-1} - n(n-1)p^2 - np) \Rightarrow\\
&E(X) \leqslant 1 + n(n-1)\pi R^2/D(1 - \pi R^2/Dp) + (\pi R^2/Dp)^2 \cdot n((1+p)^{n-1} - 1) \Rightarrow\\
&E(X) \leqslant 1 + n(n-1)\pi R^2/D(1 - \pi R^2/Dp) + (\pi R^2/Dp)^2 \cdot n((1+np/n)^n/(1+p) - 1) \Rightarrow\\
&E(X) \leqslant 1 + n(n-1)\pi R^2/D(1 - \pi R^2/Dp) + (\pi R^2/Dp)^2 \cdot n(e^{np} - 1).
\end{aligned}
$$

Note that the second term in the above expression is negative since we have that $p \leqslant \pi R^2/D$. We can get an upper bound of the above expression by setting $p = \pi R^2/D$ and $n = \lambda D$, where $\lambda$ is the host density. So,

$$E(X) \leqslant 1 + n(e^{\lambda \cdot \pi R^2} - 1)$$

Finally, for $l_{\max}$, that is the length of the longest chain path, we have that $l_{\max} = X - 1$ and finally

$$E(l_{\max}) \leqslant n(e^{\lambda \cdot \pi R^2} - 1).$$

Note that the above bound is only meaningful for very sparse networks that is for very small values of $\lambda$.

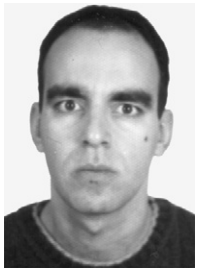
## References


[1] J. Yu, P. Chong, A survey of clustering schemes for mobile ad hoc networks, IEEE Communications Survey 7 (1) (2005) 32–48.
[2] M. Gerla, J. Tsai, Multicluster, mobile, multimedia radio network, ACM-Baltzer Journal of Wireless Network 1 (3) (1995) 255–265.
[3] A. McDonald, T. Znatti, A mobility-based framework for adaptive clustering in wireless ad hoc networks, IEEE Journal on Selected Areas in Communications 17 (8) (1999) 1466–1487.
[4] P. Basu, N. Khan, T. Little, A mobility based metric for clustering in mobile ad hoc networks, in: Proceedings of the 21st International Conference on Distributed Computing Systems Workshops (ICDCSW'01), 2001, pp. 413–418.
[5] I. Er, W. Seah, Mobility-based d-hop clustering algorithm for mobile ad hoc networks, in: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2004), vol. 4, pp. 2359–2364.
[6] I. Er, W. Seah, Performance analysis of mobility-based d-hop (MobDHop) clustering algorithm for mobile ad hoc networks, Computer Networks 50 (2006) 3375–3399.
[7] R. Palit, E. Hossain, P. Thulasiraman, MAPLE: a framework for mobility-aware pro-active low energy clustering in ad-hoc mobile wireless networks, Wireless Communications and Mobile Computing 6 (2006) 773–789.


[3] $P(X \geqslant a) \leqslant E(X)/a$ with $a = 1$.

[8] M. Schwartz, Mobile Wireless Communications, Cambridge Univeristy Press, 2005.
[9] S. Basagni, Distributed clustering for ad-hoc networks, in: Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN'99), June 1999, pp. 310–315.
[10] S. Basagni, Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks, in: Proceedings of IEEE International Vehicular Technology Conference, September 1999, pp. 889–893.
[11] M. Chatterjee, S. Das, D. Turgut, WCA: a weighted clustering algorithm for mobile ad hoc networks, Cluster Computing 5 (2002) 193–204.
[12] S. Sivavakeesar, G. Pavlou, A. Liotta, Stable clustering through mobility prediction for large-scale multihop ad hoc networks, in: Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'2004), March 2004.
[13] D. Johnson, D. Maltz, Dynamic source routing in ad hoc wireless networks, in: T. Imelinsky, H. Korth (Eds.), Mobile Computing, Kluwer Academic Publishers, 1996, pp. 153–181.
[14] Amiya Bhattacharya, Sajal K. Das, LeZi-Update: an information-theoretic framework for personal mobility tracking in pcs networks, Wireless Networks 8 (2002) 121–135.
[15] T. Camp, J. Boleng, V. Davies, A survey of mobility models for ad hoc network research, in: Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications, vol. 2, no. 5, 2002, pp. 483–502.
[16] T.C. Bell, J.G. Cleary, I.H. Witten, Text Compression, Prentice Hall, 1990.
[17] J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE Transactions on Information Theory 24 (5) (1978) 530–536.
[18] J. Kleinberg, E. Tardos, Algorithm Design, Addison Wesley, 2006.
[19] M. Gerla, T.J. Kwon, G. Pei, On demand routing in large ad hoc wireless networks with passive clustering, in: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC'2000), September 2000.
[20] C. Konstantopoulos, D. Gavalas, G. Pantziou, A mobility aware technique for clustering on mobile ad-hoc networks, in: Proceedings of the Eighth International Conference on Distributed Computing and Networking ICDCN 2006, LNCS, vol. 4308, Springer Verlag, 2006, pp. 397–408.
[21] W.C. Huffman, V. Pless, Fundamentals of Error-Correcting Codes, Cambridge University Press, 2003.
[22] R. Graham, D. Knuth, O. Patashnik, Concrete Mathematics, Addison-Wesley Publishing Company, 1994.
[23] G. Cohen, I. Honkala, S. Litsyn, A. Lobstein, Covering Codes, Mathematical Library, vol. 54, Elsevier, Amsterdam, 1997.
[24] E. Dantsin, A. Goerdt, E. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, Uwe Schoning, A deterministic $(2-2/(k+1))^n$ algorithm for $k$-SAT based on local search, Theoretical Computer Science 289 (2002) 69–83.
[25] V. Marathe, H. Breu, H.B. Hunt III, S.S. Ravi, D.J. Rosenkrantz, Simple heuristics for unit disk graphs, Networks 25 (1995) 59–68.
[26] Network Simulator – NS-2. <http://www.isi.edu/nsnam/ns/>.
[27] J. Yoon, M. Liu, B. Noble, Random waypoint considered harmful, in: Proceedings of IEEE INFOCOM, 2003.
[28] X. Hong, M. Gerla, G. Pei, C. Chiang, A group mobility model for ad hoc wireless networks, in: Proceedings of the Second ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 1999, pp. 53–60.
[29] <http://web.informatik.uni-bonn.de/IV/Mitarbeiter/dewaal/BonnMotion/>.

**Charalampos Konstantopoulos** received his Diploma in computer engineering from the Department of Computer Engineering and Informatics at University of Patras, Greece (1993). He also received his Ph.D., degree in Computer Science from the same department in 2000. Currently, he is a Ph.D., researcher at the Research Academic Computer Technology Institute, Patras, Greece. His research interests include parallel and distributed algorithms/architectures, mobile computing and multimedia systems.



**Damianos Gavalas** received his BSc degree in Informatics from University of Athens, Greece, in 1995 and his MSc and Ph.D., degree in electronic engineering from University of Essex, UK, in 1997 and 2001, respectively. He is currently Assistant Professor in the Department of Cultural Technology and Communication, University of the Aegean, Greece. His research interests include distributed computing, mobile code, network and systems management, network design, e-commerce, m-commerce, mobile ad hoc and wireless sensor networks.

**Grammati Pantziou** received the Diploma in Mathematics and her Ph.D. degree in Computer Science from the University of Patras, Greece, in 1984 and 1991, respectively. She was a Post-Doctoral Research and Teaching Fellow at the University of Patras (1991–1992), a Research Assistant Professor at Dartmouth College, Hanover, NH, USA (1992–1994), an Assistant Professor at the University of Central Florida, Orlando, FL, USA (1994–1995) and a Senior Researcher at the Computer Technology Institute, Patras (1995–1998). Since September 1998, she is a Professor at the Department of Informatics of the Technological Educational Institution of Athens, Greece. Her current research interests are in the areas of parallel computing, design and analysis of algorithms, distributed and mobile computing and multimedia systems.