Computing (2012) 94:811–832
DOI 10.1007/s00607-012-0202-3

ASH: tackling node mobility in large-scale networks

Andrei Pruteanu · Stefan Dulman

Received: 15 November 2011 / Accepted: 12 June 2012 / Published online: 9 August 2012

Abstract With the increased adoption of technologies like wireless sensor networks by real-world applications, dynamic network topologies are becoming the rule rather than the exception. Node mobility, however, introduces a range of problems (communication interference, path uncertainty, low quality of service and information loss, etc.) that are not handled well by periodically refreshing state information, as algorithms designed for static networks typically do. To address specifically this problem, the main contribution of this paper is the introduction of a novel mechanism (called *ASH*) for the creation of a quasi-static overlay on top of a mobile topology. It is powered by simple, local interactions between nodes and exhibits self-healing and self-organization capabilities with respect to failures and node mobility. We show that the overlay mechanism works without assumptions about position, orientation, speed, motion correlation, and trajectory prediction of the nodes. A preliminary evaluation by means of simulation shows that *ASH* succeeds in tackling node mobility, while consuming only minimal resources.

Keywords Static overlay · Mobile networks · Clustering algorithm · Self-adaptive networks · Self-configuring networks

Mathematics Subject Classification 68M14 - Distributed systems

1 Introduction and motivation

Recent years have seen a significant increase in the number and the diversity of the devices that form the wireless networks around us. The number of devices per

A. Pruteanu (✉) · S. Dulman
Mekelweg 4, Delft, The Netherlands
e-mail: andrei.pruteanu@gmail.com; a.s.pruteanu@tudelft.nl

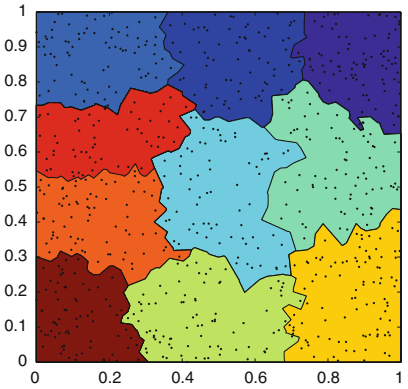
S. Dulman
e-mail: s.o.dulman@tudelft.nl

network has grown substantially, and research domains such as *mobile ad-hoc networks* (MANETs) and *wireless sensor networks* (WSNs) have studied the corresponding scalability issues, for example, by providing theoretical boundaries [15,16]. However, large collections of networked devices also bring in the problem of mobility; the larger the network, the higher the probability that individual or groups of devices become mobile. Examples range from networks in which mobility occurs relatively rarely (e.g., static networks with occasional node relocation due to maintenance operations) to highly dynamic networks (e.g., monitoring freight in transport and logistics applications).

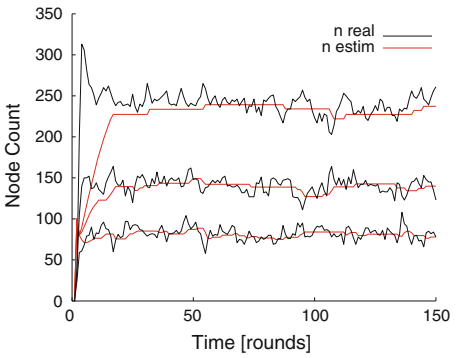
For networks that exhibit low mobility, algorithms developed for static topologies perform reasonably well. Usually, changes of the device positions are detected and the network topology is repaired—either via a dedicated mechanism within the protocol (as in the DSR algorithm [21]), or by simply refreshing state information periodically. For networks that exhibit high mobility, however, new solutions are needed as the required rate of adaptation induces way too much (communication) overhead. Recent research projects targeting the development of large-scale cyber-physical systems, including *programmable matter* [14], swarms of tiny robots [22] and *amorphous computing* [2], take mobility as a default assumption. A common approach to tackle mobility has not materialized yet; most of the research efforts are focused on the scalability aspect, in particular, the need to program the network as a whole rather than as an individual set of nodes [10,18,31].

In this paper we propose a novel mechanism (called *ASH*) for handling mobility in large-scale networks; in essence we "slow down" the network by creating a quasi-static overlay on top of the highly mobile network. A unique feature of *ASH* is that it is based on the execution of local rules only: there is no knowledge of the global structure of the network and there is no usage of additional information related to position, speed and direction of nodes. A key idea behind *ASH* is that nodes are not addressed individually, but rather that the network is composed of a set of domains—groups of nodes—whose membership constantly changes (see Fig. 1a). By observing their neighbors, nodes can decide themselves which domain they currently belong to; the decision policy is tuned to lead to domains whose centers of gravity hover around slowly, effectively providing a "quasi-static" overlay. The name *ASH* was inspired by the metaphor of an ash cloud where tiny particles are floating around in the air. The cloud is slowly traversing the sky, while the contained particles move around each other randomly and fast. This natural phenomenon resembles the dynamics of our mechanism; the cloud can be compared to the *domains (clusters)*, while the volcanic ash particles resemble the moving individual nodes.

ASH can be used directly as an efficient overlay mechanism, for example, by assigning different application-level functionality to the different domains. Alternatively, *ASH* can be used as a clustering protocol by adding a leader election mechanism (see Sect. 4). *ASH* is—by design—very robust to node and link failures. It is based on a combination of gossiping, which is topology agnostic, with a periodic adjustment procedure that reconstructs local state based on the actual neighborhood. Between state updates, message loss and node failures are simply regarded and treated as nodes leaving the network (domain). Simulations show that *ASH* succeeds in providing a



(a) ASH Overlay (colors mark different domain IDs); node positions superimposed (black dots).



(b) Domain Size Evolution (green (continuous) = real number of nodes in a domain; red (dotted) = estimated domain size).

Fig. 1 ASH geometry and time evolution

robust overlay mechanism at low cost, that is, with minimal message exchange. This approach makes *ASH* extremely robust in contrast to many other existing protocols.

The domains defined by *ASH* are quasi-static with respect to the deployment area. Their mobility does not increase with the increase of the speed of the nodes. From this perspective, if applications are targeted at the domains *ASH* provides, rather than at the individual nodes, then employing algorithms for static networks on top of the overlay becomes possible.

The remainder of this paper is organized as follows. In Sect. 2 we present an overview of *ASH*, while in Sect. 4 we introduce a clustering mechanism as an application example. The convergence of the algorithm is shown in Sect. 3. We analyze the performance of *ASH* in Sect. 5. In Sect. 6 we describe related work. Finally, Sect. 7 concludes the paper.

2 The *ASH* algorithm

Since we noted that most existing networking protocols cannot handle large-scale networks exhibiting high node mobility, we set out to design an architecture that would be capable of handling these networks of the future. The result is a fully decentralized mechanism (*ASH*) that makes use only of local nodes interaction, to create a *quasi-static overlay*, a virtual partitioning of the network into domains. For the sake of clarity, we define a *domain* as a group of neighboring nodes that share the same identifier (domain ID). Each domain usually spans over *multiple communication hops* and has a tendency to maintain a convex shape—see Fig. 1a.

The domains in *ASH* can be thought of in analogy with a number of gas balloons filling a fixed physical space (i.e., a box). Due to disturbances, the shape of the balloons

and their relative positions may change. The amount (mass) of gas in each balloon is nevertheless constant, although the pressure in each balloon may fluctuate. Despite the random movement of air molecules, the system will converge to a stable state. *ASH* works on the same principle: each domain has a total *mass* $\mathcal{M}$ distributed over the nodes in that domain. The share of mass a node i holds is denoted by m_i. In a domain $\mathcal{S}$, we have $\sum_{i \in \mathcal{S}} m_i = \mathcal{M}$. p_i represents the local pressure of each node and is a function of the total mass $\mathcal{M}$ and *the number of nodes* in a domain.

For a domain containing a large number of nodes, the share of the mass variable on each node will be small (we say that the domain has a low pressure). Neighboring domains containing a smaller number of nodes (i.e., having a higher pressure) will extend, by pushing the boundaries of the first domain until the number of nodes in each domain will be approximately equal—pressures will equalize.

For a static network, the system will converge to an equilibrium at the borders between the domains. In a dynamic network, the mobile nodes continuously fluctuate the position of the borders, although the resulting macro-mobility of the domains does not increase when node mobility is higher (this is somewhat similar to immiscible fluids interaction modeling with cellular automata [5]).

In order to explain *ASH* in detail, we consider the abstraction of network communication occurring in *rounds*—similarly to the work presented in [19]. Rounds are fixed-length time intervals with each node broadcasting once every round. This model does not reduce the generality of the solution as the rounds *do not need to be synchronized*, avoiding cumbersome clock synchronization between nodes. In practice, this translates to nodes performing actions approximately periodically, such that, when averaging over a large period of time, all nodes perform equal number of actions.

During each round, each node has to perform the following three phases (see Algorithm 1):

1. *domain ID selection*—nodes will decide their domain ID based on the domain IDs and local pressures of their neighbors (see Sect. 2.2);
2. *residual mass return*—nodes that just changed their domain ID will distribute their mass value to neighboring nodes from the old domain, if any (see Sect. 2.3);
3. *diffusion of mass*—nodes will attempt to equally distribute the mass in each domain (i.e., equalize the pressure in each domain) by means of gossiping (see Sect. 2.4).

The second action (residual mass return) can lead to mass loss in the domains ($\sum_{i \in \mathcal{S}} m_i < \mathcal{M}$); nodes belonging to one domain can move out so fast that they do not have the chance to return their share of the mass back to the old domain. This phenomenon leads to a steady drop of mass in the domains over time—see Sect. 2.5 for a solution.

2.1 Initialization

ASH considers a fixed number of domains at start ($\mathcal{N}_\mathcal{S}$). In the initialization part, the algorithm starts by randomly assigning domain id k ($k = 1..\mathcal{N}_\mathcal{S}$) and mass $\mathcal{M}$ to a $\mathcal{N}_\mathcal{S}$ number of nodes. They act as "seeds" from which the domains will "grow". Thus, each domain starts out as a single node, and will expand until the whole deployment

Algorithm 1 *ASH* algorithm.

```
1: function ASH(N_S, n_r, M)
            N_S – number of domains
            n_r – number of rounds
            M – initial mass for all domains
2:     for all N_S domains k do                          ▷ initialization
3:         Pick random unassigned node j
4:         node j ← domain k
5:         node j ← mass M
6:     end for
7:     for n_r rounds do                                 ▷ main algorithm
8:         for all nodes i do                            ▷ algorithm phase 1
9:             node i updates its domain ID
10:        end for
11:                                                      ▷ algorithm phase 2
12:        for all domain leader nodes j do
13:            node j runs pressure correction phase
14:        end for
15:        for all nodes i do                            ▷ algorithm phase 3
16:            node i runs diffusion phase
17:        end for
18:    end for
19: end function
```

area is covered by domains. These special nodes play a role solely during the initialization phase. They do not need to act seeds after the clusters have converged to a stable structure.

When compared to "classic" clustering algorithms (although *ASH* is a more general framework!), the fixed number of domains might seem like a limitation. Almost all existing competing algorithms dynamically adapt the number of clusters. *ASH* can be easily extended to comply with this behavior, by allowing domains to be dynamically created at run-time. For example, when the pressure inside a domain is very low (meaning that the domain is made up of a large number of nodes), a new domain can be spawned. A similar rule based on checking the pressure level can be used to remove unwanted domains.

Based on the current approach, we are enabling a new class of applications. The constant number of multihop domains allows the so-called functional partitioning of a network. Each domain will be assigned a different functionality. As individual nodes randomly roam through domains, a scheduling policy at high level, of how many nodes should perform a certain functionality at each given moment, is easily implementable.

2.2 Domain ID selection

Each node will decide its domain ID ("domain color" in Fig. 1a) based on a weighted combination between *majority voting*—dominant domain ID of its neighbors—and the *pressure difference* between neighboring domains (via a weight $\eta \in [0, 1]$).

Let us assume a node i has in its vicinity nodes from D_i distinct domains. The number of neighbors for node i, including itself, belonging to a domain k (where

$k = 1..|D_i|$) is $n_{i,k}$. The *average pressure* of the surrounding nodes in domain k is $p_{i,k} = \frac{m_{i,k}}{\omega_{i,k}}$ (See Push-Sum algorithm [23]). The node i will compute a series of values $\theta_{i,k}$:

$$\theta_{i,k} = (1 - \eta) \cdot \frac{n_{i,k}}{\sum_{t \in D_i} n_{i,t}} + \eta \cdot \frac{p_{i,k}}{\sum_{t \in D_i} p_{i,t}}. \quad (1)$$

Let $\tilde{k}$ be the id of the domain corresponding to the maximum $\theta_{i,k}$ for node i ($\tilde{k} = \arg\max_k \theta_{i,k}$). The node i will consider switching its domain ID to the domain $\tilde{k}$. Let k_0 be the previous domain id of the node i. To allow for a smooth functioning of the network, the switch to a new domain is subject to a threshold mechanism: node i will switch its domain only if $|\theta_{i,k_0} - \theta_{i,\tilde{k}}| > \Delta$ with Δ being a predefined threshold.

Since the domain selection process is carried out independently by all nodes, it can happen that a small domain simply disappears when all members join a neighboring domain. If not prevented, this effect will carry through and cause all domains to eventually collapse into a single one, spanning the entire network. This undesirable effect is removed by introducing a *domain leader* (see Sect. 4), which will be prevented from changing its domain ID. This step is only needed at the initialization part.

2.3 Residual mass return

At this phase, nodes that decided to change their domain ID need to adjust their mass by sending it back to the old domain and enter the new domain with mass 0. The diffusion phase that follows ensures that mass redistributes equally in both the old and the new domain.

The simplest way a node can return mass to the old domain is to select one or more of its neighbors from the old domain and send them its mass. This approach works in most cases, with one exception though. It can happen, due to various dynamics, that a node finds itself in a situation in which it has no neighbors from the old domain anymore. In this case, the mass on the node will actually be lost, unless a mechanism such as routing is in place. We decided to use the simple solution of discarding the mass in this particular case, and repairing the loss later. The reason is that we avoid the complexity of routing in a dynamic network topology, and the repair mechanism described in Sect. 2.5 is easy to implement.

2.4 Diffusion

At this phase of the *ASH* algorithm nodes diffuse the mass via gossiping in order to flatten-out the pressure distribution. Since the nodes are *not* synchronized, and may deploy sleep schedules to conserve energy (behavior common to sensor networks), there is no guarantee that all neighbors are ready to communicate when a node enters the diffusion phase. This situation is aggravated by nodes moving in and out of range, as well as errors on the wireless communication channel. To handle the resulting volatility *ASH* employs a gossiping style of communication on top of a periodic mechanism of

neighborhood discovery (the diffusion phase may actually consist of several gossiping rounds—see Sect. 5).

Periodic neighborhood discovery is done by nodes sending short "Hello" messages containing a tuple ⟨*node ID*, *domain ID*, *local pressure*⟩. Periodic neighborhood discovery is a common mechanism in mobile networks, and the functionality is provided by the *media access control* (MAC) layer. For the implementation of the gossiping mechanism, we also consider acknowledgments. This needs not be perfect due to the fact that the pressure correction mechanism described in Sect. 2.5 compensates the effects of message loss.

Similarly to the *Push-Sum gossiping algorithm* [23], each node i needs to store the following local variables: the local mass (m_i), a weight factor (ω_i) and the *domain ID* of the node (d_i). Local pressure is computed as $p_i = \frac{m_i}{\omega_i}$ via the averaging mechanism described in the Push-Sum algorithm.

In short, the gossiping protocol works as follows. Assume a node i has the values $m_{i,t}$ and $\omega_{i,t}$ at the beginning of communication round t. Node i randomly picks a neighbor from the same domain, and sends it *and* itself the set $\{\frac{m_{i,t}}{2}, \frac{\omega_{i,t}}{2}\}$. During that round t, the node receives updates $\{m^r_{j,t}, \omega^r_{j,t}\}$ from a set $\mathcal{S}_0$ of n_i neighbors, including itself ($j \in \mathcal{S}_0$). The node updates its mass value and weight, for the communication round $t+1$, as follows: $m_{i,t+1} = \sum_{j\in\mathcal{S}_0} m^r_{j,t}$ and $\omega_{i,t+1} = \sum_{j\in\mathcal{S}_0} \omega^r_{j,t}$. As shown in Proposition 2.2 in [23], the sum $\sum_i m_{i,t}$ remains constant at each moment in time.

For fixed infrastructures, standard gossiping has a convergence time for computing an average value across the network within accuracy e that requires $\Theta(n^2 \log e^{-1})$ messages. The solution of constructing a spanning tree and flooding back the average in an ad-hoc network introduces a lot of overhead that leads to high message complexity. It has been proven that any kind of mobility is beneficial, especially the fully-random one as is the case with our scenarios [28]. Different mobility patterns can have significantly different effects on the convergence of distributed algorithms such as gossiping [13]. If m nodes have full mobility and the others are fixed, the convergence time drops to $\Theta(n^2/m \log e^{-1})$.

To ensure that gossiping has enough time to converge, the diffusion phase may actually consist of several gossiping rounds. The number of rounds that are required depends on the application, more exactly, on the average speed of the nodes and their density, as well as the desired domain stability. We evaluate this dependency via simulation and present the results in Sect. 5.

2.5 Mass correction

In practice, domains lose mass over time. This happens primarily as an effect of node mobility; when a node suddenly finds itself surrounded by neighbors all belonging to a different domain than itself, it must switch domain ID, but cannot hand its residual mass back to the originating domain. Simulation results show that the mass loss is small even for high mobility, across all domains. Nonetheless, if no measures are taken, the mass in each domain will constantly drop towards 0. For the case of real networks, mass loss may also occur due to various failures. For example, nodes that suddenly reset or messages lost in the diffusion phase lead also to additional mass loss.

Thus, providing a mechanism for solving this problem, leads not only to a solution for the problem of nodes having no neighbors from the same domain, but also constitutes a self-healing mechanism for two of the most common failures met in mobile wireless networks.

Simplistic approaches for solving the mass loss issue may rely on knowing the statistical characteristics of the network: average density and flux of nodes in and out of domains. Based on these estimations, the mass could be periodically increased in each domain with a precomputed amount. This mechanism, however, cannot guarantee that the average mass across all domains is stable (may diverge to either infinities) due to the lack of a feed-back mechanism. We propose a solution for keeping the average mass level constant in the domains that assumes the existence of a leader in each domain (similar to a conventional *cluster head*). The basic idea is that a diffusion-based mechanism (called the *ASH-NetSize algorithm*) is used to estimate the number of nodes inside a domain. By multiplying this estimate with the average mass value obtained in the previous round, a domain leader can determine the total mass in its domain. If it drops bellow a threshold, the leader can inject additional mass into the domain to compensate for the loss.

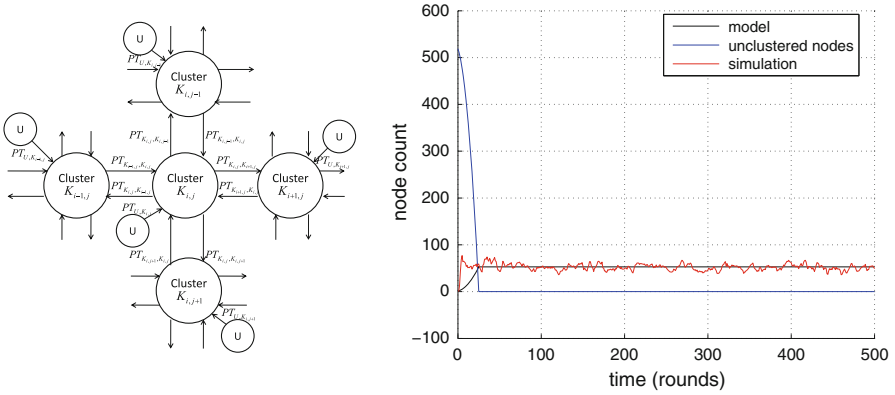
ASH-NetSize is making use of the gossiping mechanism [28] to estimate the domain size. One of the aggregates that we can compute via gossiping is the mean value of some shared variable ϕ through all n nodes of domain ($\overline{\phi} = \frac{\sum_i \phi_i}{n}$ with $i = \overline{1..n}$). Let us assume that all nodes have a value of $\phi_i = 1$. The domain leader (node k) estimates the size of the network to be n_e and subtracts this value locally ($\phi_k \leftarrow 1 - n_e$). Via gossiping, after a number of rounds, the set ϕ_i converges to a new set $\phi'_i = \overline{\phi'}$—the new average of the distributed variable. If the network size was correctly estimated, then $\overline{\phi'_i} = 0,\ \forall i$. If not, then the sign and value of $\overline{\phi'_i}$ gives an indication on how the estimation of the network size needs to be updated (if $\overline{\phi'_i} < 0$ then n_e was overestimated, else it was underestimated). By constantly updating it, n_e will follow the variations in the network size.

Due to space limitations, the *ASH-NetSize* algorithm cannot be presented in full detail, but we do provide information on its accuracy. Figure 1b shows that the estimation algorithm is able to follow the fluctuations of the domains quite closely, smoothing out the "noise". The traffic overhead associated with *ASH-NetSize* is minimal, since the correction information is piggy-backed through the already existing mechanism of diffusion (see Sect. 2.4). As a result, the average mass in the domains will be close to the desired value.

3 ASH convergence

The creation of *ASH* clusters depends solely on local rules. At equilibrium, they converge to an equilibrium of equal node count. To prove convergence, we will first consider a general case, in which we analyze the variation of the number of nodes N_i and N_j in two neighboring clusters i and j.

The amount of nodes that switch between the two clusters i and j is proportional to the contact border $\phi_{i,j}$ between them. Let the node density in cluster i be $\rho_i = \frac{N_i}{S_i}$,



(a) Model State Diagram. **(b)** Cluster-size convergence.

Fig. 2 ASH modeling

where N_i is the number of nodes in cluster i and the cluster surface is S_i. Let $r_i = \sqrt{\frac{N_i}{\pi \cdot \rho_i}}$ be the radius of the cluster i, given that it has a convex shape. Let the length of the contact border between the two clusters i and j be

$$L_{i,j} = f_{i,j} \cdot 2\pi r_i = 2 f_{i,j} \sqrt{\frac{\pi N_i}{\rho_i}} \tag{2}$$

where $f_{i,j}$ is the fraction of cluster i border that is "touching" cluster j. We can approximate $L_{i,j}$ to:

$$L_{i,j} \approx K \cdot \sqrt{N_i} \tag{3}$$

K is a constant resulting from the fact that ρ_i is also constant at equilibrium. Since we are dealing with a spatial problem, the distance between the domains also plays an important role. We capture it via function $\alpha(d_{i,j})$ that models the relation between domains i and j distance, and *the node exchange rate* between domains.

A sketch of the state transitions that models the system is shown in Fig. 2a. $PT_{U,i}$ denotes the probability of unclustered nodes to be part of the cluster i. $PT_{i,j}$ shows the probability that a node will transition from cluster i to cluster j.

$$PT_{U,i} \approx K \cdot \sqrt{N_i} \tag{4}$$

$$PT_{i,j} \approx Sign(N_i - N_j) \cdot K \sqrt{N_i} \cdot \alpha(d_{i,j}) \tag{5}$$

We approximate $\alpha(d_{i,j})$ with an exponential distribution to show that the rate of transfer between neighboring domains decreases exponentially fast with the distance between them. Although in practice $\alpha(d_{i,j})$ has a very abrupt profile, being actually equal to 0 after a distance threshold, the exact form of the function does not influence

the convergence proof (although it certainly influences the convergence rate).

$$\alpha(d_{i,j}) = e^{-\psi \cdot d_{i,j}} H(d_{i,j}) \tag{6}$$

where ψ is a constant and H is a modified version of the heaviside step function, defined as following:

$$H(d_{i,j}) = \begin{cases} 0 & \text{if } d_{i,j} > 0.5 \\ 1 & \text{if } d_{i,j} \leq 0.5 \end{cases}$$

From a node flow perspective, the number of nodes in a domain j varies as following:

$$\frac{dN_j}{dt} = \sum_{i=1, i \neq j}^{N} Sign(N_j - N_i) \cdot K\sqrt{N_j} \cdot \alpha(d_{i,j}) \tag{7}$$

We check the model by comparing it with simulation results as shown in Fig. 2b. As parameters, we chose ten clusters and node mobility following a Random Walk model. The maximum node speed was set to 20 meters per second [mps]. The average neighborhood size was 15. For the constants, we used $K = 0.1$ and $\psi = 3$.

As shown by Fig. 2b, the number of un-clustered nodes converges fast to 0. The results in the simulation closely match the model. In order to compute the equilibrium point, we need to solve the system given by $\frac{dN_j}{dt} = 0$, for all clusters j in the setup. If for all clusters j, we sum up Eq. 7 and reduce K and $\alpha_{i,j}$, we have:

$$\begin{aligned} 0 = &\sqrt{N_1}(Sign(N_2 - N_1) + \cdots + Sign(N_K - N_1) \\ &+\sqrt{N_2}(Sign(N_1 - N_2) + \cdots + Sign(N_K - N_2)) \\ &\cdots \\ &+\sqrt{N_K}(Sign(N_1 - N_K) + \cdots + Sign(N_{K-1} - N_K)) \\ = &\, f(N_{i=1..K}) \end{aligned} \tag{8}$$

Since the number of nodes in each cluster is strictly positive, at equilibrium, $(Sign(N_2 - N_1) + \cdots + Sign(N_K - N_1) = 0$, $(Sign(N_1 - N_2) + \cdots + Sign(N_K - N_2) = 0$, etc. For the general case, the only equilibrium point of the dynamic system occurs only when $N_1 = N_2 = N_3 = \cdots = N_K = \frac{N}{K}$, as expected.

4 Application example—*ASH-cluster*

In this section we show how the domains built by *ASH* can be used by various applications. We revert to the example of clustering, for which a large number of algorithms have been surveyed and compared in works such as [1,33]. The performance of these clustering schemes is evaluated with a multitude of metrics: communication overhead, power balancing, re-clustering ripple effect, cluster formation time, etc. The large

Algorithm 2 *ASH* cluster head election

1: **function** ASH- CLUSTERHEAD(c_{t-1}) **returns** c_t
  local node variables:
    g_i – local gradient
    n_i – nr. neighbors of i
    $n_{o,i}$ – nr. neighbors of i from other domains
  notations:
    i – node identifier
    c_r – clusterhead node ID at round r
    $\mathcal{N}_i$ – set of neighbors of i in the same domain
2: **for all** nodes in domain **do** ▷ algorithm phase 1
3: $g_i \leftarrow \frac{n_{o,i}}{n_i+1}$
4: **end for**
5: **for all** nodes in domain **do** ▷ algorithm phase 2
6: **if** $i == c_{t-1}$ **then**
7: $c_t \leftarrow \arg\min g_i \in \mathcal{N}_i$
8: **end if**
9: **end for**
10: **end function**

majority of these algorithms target static networks—as soon as mobility is involved the clustering problem becomes increasingly more difficult to solve although possible alternatives have been proposed [32].

The domains defined by *ASH* can be readily used as clusters, hence, *ASH-Cluster* (see Algorithm 2) was developed as a natural algorithm on top of the overlay. *ASH-Cluster* provides multihop clustering for mobile networks by solving the problem of re-clustering ripple effect in an elegant way, while keeping the communication overhead at a low value. The key is that *ASH-Cluster* creates the domains (clusters) independent of the election of a cluster head. This makes it superior to the large majority of existing algorithms, in the sense that mobility of a cluster head does not trigger re-clustering. Actually, the cluster head election is a mechanism implemented in *ASH-Cluster after* the clusters have been created.

Assume node i belongs to domain D_i. To establish a gradient on node i (see Fig. 3a), we use the ratio between the number of neighboring nodes belonging to other domains ($\sum_{j \in D_i, j \neq i} n_{i,j}$) and the number of all neighboring nodes ($\sum_{j \in D_i} n_{i,j}$).

Figure 3a shows the gradient in colors, blue indicating the center of the domains. The probability of nodes "hosting" the cluster head agent is smallest in the red regions and highest in the dark blue ones.

Routing of information takes place in a unidirectional way, in the sense that nodes can send data towards the cluster head (fitting the data-collection type of applications) as shown is [3], via the gradient mechanism described below. The communication between the cluster heads is similar to the one used by the LEACH protocol [17]. The cluster heads form a backbone network (a spanning tree routed at the gateway) and make use of an increased transmit power to communicate to each-other. The cluster head is, in our case, a software agent that "jumps" to different nodes to perform the data collection and communication with other agents. It is usually located in the minimum gradient area, and also uses the gradient to restrict the search area (ideally to the

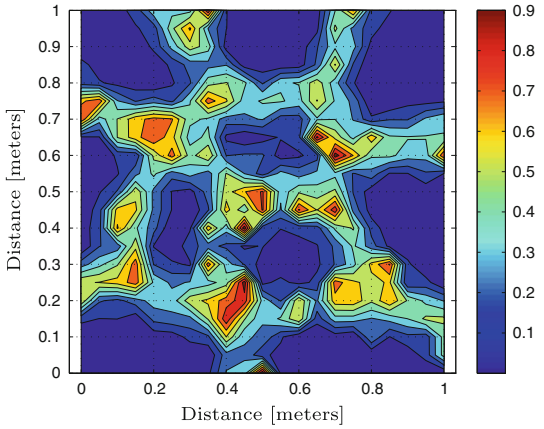


Fig. 3 ASH gradient

center of the domain) when looking for a new candidate to "jump to". Nodes route data towards the minimum gradient point, where it will be met by the cluster head agent. The second phase of Algorithm 2, has a very low overhead. When a given cluster-head finds in its neighborhood a node with a lower gradient value it passes the token to it. The rule is only performed by the clusterhead node. Additionally, *ASH-cluster* algorithm avoids routing deadlocks caused by the uncertainty of the cluster-head assignments. It works such that a node will send packets in a multi-hop manner via the gradient. There is no need for the routing tables to maintain paths towards the cluster-head.

5 *ASH* algorithm analysis

To evaluate the stability of *ASH*, we need some means of characterizing the shapes of the domains as well as their fluctuation in size (number of nodes). We introduce four metrics to measure the stability of domains over time: the motion of the centroid of the node positions per unit of time (*motion metric*), the domain shape variation per unit of time (*variation metric*), the standard deviation of the domain sizes (*Std. Dev. Domain Size metric*) and the ratio between the largest domain size and the smallest domain size (*ratio metric*). The domains "hover" around, somewhat similarly to Brownian motion, at a speed significantly smaller than the average speed of the nodes in the network. At the same time, the shapes of the domains fluctuate around a stable circular-alike perimeter. The *motion metric* captures the actual mobility of the domains per unit of time, and the *variation metric* captures the fluctuations in surface size per unit of time. The *stdSize metric* tracks the variation of domain sizes through time and the *ratio metric* shows the imbalance between the largest domain size (measured in number of nodes) and the smallest.

The *motion metric* is equal to the distance traveled by the centroid of the node positions, in a domain. The *centroid* is defined as $(x_c, y_c) = (\frac{1}{n}\sum_i x_i, \frac{1}{n}\sum_i y_i),\ i = 1..n$, where x_i and y_i are the coordinates of the nodes. The motion metric is defined as $m_1 = ||(x_{c,t+1}; y_{c,t+1}), (x_{c,t}; y_{c,t})||$.

Let $\overline{d}_t$ be the mean value of the distances between the centroid of a domain and all the nodes in the domain, at time round t. The *variation metric* is equal to the difference: $m_2 = \overline{d}_{t+1} - \overline{d}_t$. This metric will show increasing values with the fluctuations of the sizes of the domains.

We simulated *ASH* and *ASH-Cluster* using Matlab. We considered mobile nodes deployed in a square deployment area with an edge length of 1,000 m. The transmission range of the nodes is set to 100 m based on a disk communication model. We set *ASH* to operate with ten domains in our simulations (cf. Fig. 1a). One might argue that such a high-level simulation is not a true representation of an actual deployment since a lot of problems occur from unexpected places (software bugs, hardware failures, communication interference, scalability issues etc). The scope of the paper is to present a mechanism that is agnostic to the lower layers such as MAC and PHY. To preserve the generality, we model all low-level errors as mass loss, with the implication that these simplified assumptions do not affect the overall stability of *ASH*. The high-level, more abstract nature of our work does not presume that a complete implementation would not be affected by various engineering problems. On the contrary, we acknowledge this limitation and claim that, due to the nature of the systems we simulate (highly mobile), an implementation on an actual embedded platform is expensive and difficult to realise and, most of the times, it is affected by potential problems caused by various implementation choices. This fact can potentially invalidate some high-level assumptions. On the other hand, algorithmic wise, it would not bring a tremendous insight for our work. Since our modeling of failures is straightforward, it is highly unlikely that this would happen. We considered Matlab due to its built-in analytic capabilities and easy of usage.

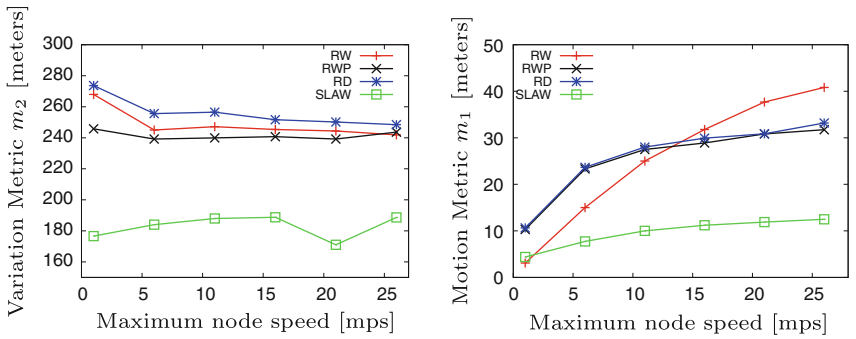
Node travel with a speeds ranging from a minimum of 10 m/round (0.5 mps) to a maximum of 100 m/round (20 mps). In our simulations, we use four mobility models: *Random Walk* [6], *Random Direction* [27], *Random Waypoint* [6] and *Slaw* [25]. The choice was made such that they cover a wide range of behaviors, from nodes traveling all over the deployment area (Random Waypoint) to nodes moving in a localized manner (Random Walk) to realistic human mobility (SLAW). Each experiment consisted of simulations running for 500 time $rounds$, for each mobility case.

For the case of the Random Walk model [6], also named Markovian Mobility model, nodes move freely anywhere in the simulation area. The direction of the movement φ is taken from a uniform distribution in the interval $[0..2\pi]$. The speed values ϑ follow a uniform distribution. Once the node reaches a destination, it chooses a new direction and starts moving toward it after a randomly chosen time interval, taken from an exponential distribution.

The Random Direction model [27] operates similarly to the random walk, except that nodes continue to travel until they are within some distance of the simulation space boundary. Then they stop and choose new, random destinations.

In the Random Waypoint model [6] a node randomly chooses a destination point in the deployment area, moves with constant speed υ (chosen uniformly from a given interval) on a straight line then pauses for a random time before it again chooses a new destination.

SLAW mobility model [25] describes social contexts that occur among people sharing common interests or those in a single community such as university campus,



(a) Variation Metric for Varying Node Speed.

(b) Motion Metric for Various Node Speeds.

Fig. 4 Node speed influence. Network size: 477 nodes

companies and theme parks. It expresses the mobility patterns involving these contexts by fractal waypoints and heavy-tail flights on top of the waypoints.

5.1 Influence of network mobility

The Random Waypoint (RWP) mobility leads to the formation of more stable domains when compared to other mobility patterns (see Fig. 4a, b). The results are similar to well known experiments, such as those presented in [7]. This is caused by the position distribution of the nodes, which is higher in the center of the deployment area, when compared to the other two models.

The Random Waypoint mobility model is used in many prominent simulation studies for ad-hoc network protocols. Although its ability to produce realistic mobility patterns is debatable, the flexibility of the model determines its adoption by a lot of simulation scenarios. The Random Direction model provides a uniform distribution of nodes over the deployment space. As seen in our simulation results, this mobility model leads to comparable results with the Random Waypoint model. The Random Walk model causes *ASH* to perform the worst due to the lack of correlation between neighboring nodes motion. The SLAW model produces to human-alike mobility and, as such, creates clusters that are more stable (reduced speed around specific points). *ASH* performs best for this model.

Figure 4 shows that the increase in node speed has basically no influence on the overall stability of *ASH* for all mobility models except *Random Walk*. In fact, both variation and motion metrics exhibit a similar behavior. This is the most important characteristic of our algorithm. For the Random Walk model, on the other hand, *ASH* shows a steadily increasing degradation in performance with the increase of speed. This happens for a high node speed due to the fact that the second term in the domain selection formula leads to irregular shapes of the domains, affecting the stability of the algorithm. Above 12 meters per second [mps], the clusters are less stable for the case of Random Walk model. Network size was set to 477 nodes in order to maintain an average node density of 15. The authors of [9] showed that the idea of a critical radius

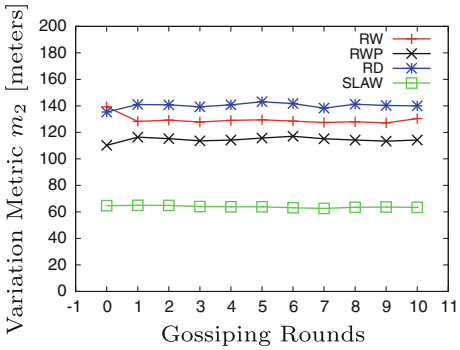


Fig. 5 The influence of gossiping on the stability of *ASH*. Network size: 477 nodes. Mobility model: RWP

(smallest possible transmission radius, to minimize the amount of consumed energy for transmission, without compromising connectivity) being determined solely on the given node density is not accurate. For the case of uniform mobility models [20], it is expressed as a function of the node velocity, as well. The benefits of the uniform node density and resulting connected graph dependence on the node velocity parameters are greatly influencing the performance of the algorithms.

Figure 4b shows the influence of average node speed on the the average domain movement (the motion metric). The average speed of the center of mass of the domains is not increasing with the average node speed.

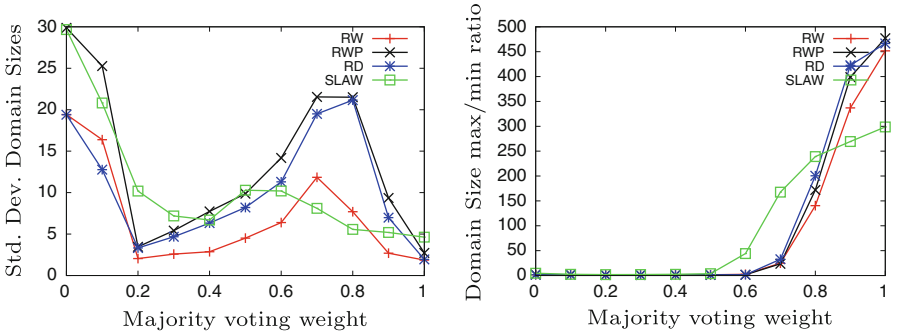
5.2 *ASH* communication mechanism

Transmission-wise, wireless communication is inherently a *broadcast* medium. This property has some advantages on the sending side, since there is no need to transmit multiple packets towards individual nodes in the one-hop vicinity when disseminating information. The usage of broadcasting, leads to faster diffusion of information.

The costs of information exchange and time synchronization are important problems in wireless sensor networks. For this reason, we do *not* assume that all nodes are awake or that they have any sort of time synchronization. Still, at every diffusion step (equal spaced in time) the nodes are sending "Hello" messages. At all times, we assume that only a subset of the one-hop neighbors receive the packets. To compensate for this reduction in diffusion speed, several gossiping rounds are performed. As seen in Fig. 5, when all the nodes are listening and are perfectly synchronized with the senders, the domains become more stable (corresponds to the value 0 on the X axis). Otherwise, *ASH* compensates the reduction of diffusion speed with more gossiping steps. As seen in Fig. 5, the performance achieved when using a perfect broadcasting mechanism is equaled by performing five or more rounds of gossiping steps.

5.3 Domain selection stability

The domain selection mechanism (see Sect. 2.2) combines majority voting and the difference between "pressure" levels of the domains. The two terms have different



(a) Std. Dev. Domain Sizes for a Varying η.

(b) Domain Size max/min ratio for Varying η.

Fig. 6 *ASH* Sensitivity to η. Network size: 477 nodes

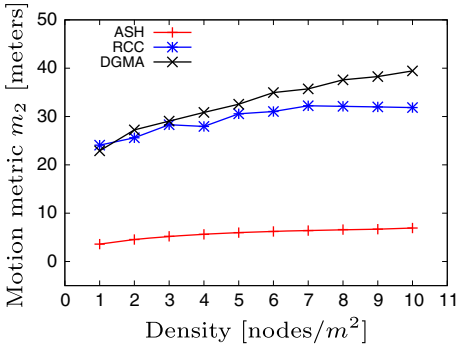
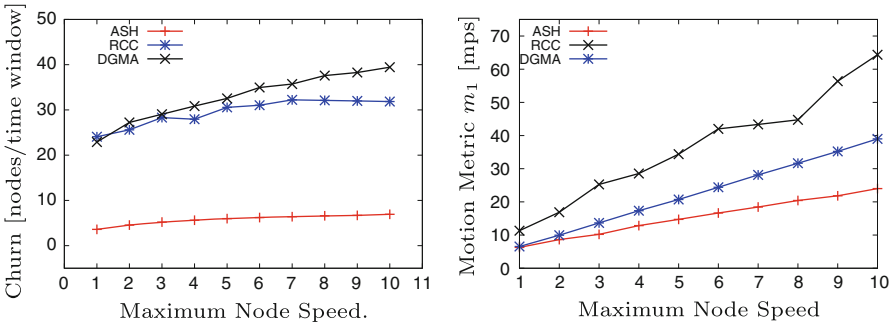


Fig. 7 *ASH* Sensitivity to different mobility models for various node mobility levels and node density

influences. Majority voting smooths out the edges at the borders of the domain, while the difference of pressure mechanism ensures the dynamic equilibrium between the domains such that all of them will converge to the same size.

We checked the stability of the static overlay mechanism by varying the weight η, see Fig. 6. We found that if majority voting dominates ($\eta \to 0$), one or more domains may disappear due to the lack of "pressure" influence on the decision. That is, a domain may keep enlarging since there is no feed-back mechanism to limit its growth, while the others will gradually shrink—see Fig. 6b. On the other hand, if the difference of pressure mechanism dominates ($\eta \to 1$), the edges are more rough and the shapes of the domains are highly irregular and keep changing frequently. This time, although the domains are balancing each other, any small difference in the pressure levels at the borders can trigger frequent changing of the domain ID values for the nodes at the border. Figure 6a confirms this by showing great fluctuations in the standard deviation of the domain sizes for $\eta \notin (0.2, 0.6)$ and stability for $\eta \in (0.2, 0.6)$.

Another important factor of the stability analysis is the network density. As seen in Fig. 7a it has an important influence on the overall stability of *ASH*. Given a fixed transmission range of 100 m, at lower node densities, the variation metric shows a



(a) Average Cluster Size Variation for Various Node Speed.

(b) Cluster Head Motion for Various Node Speed.

Fig. 8 Algorithm comparison for various node speed. Mobility model: RWP

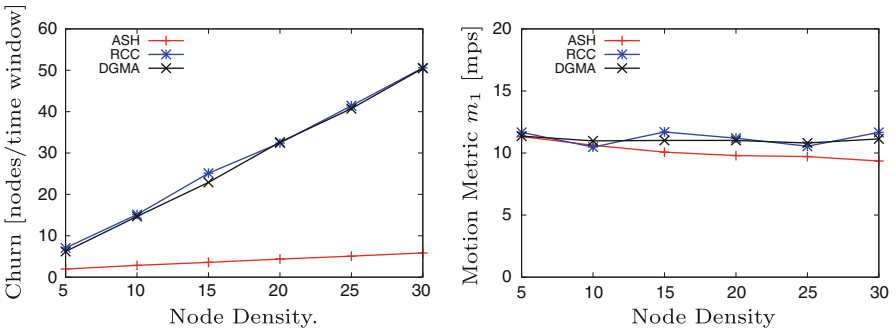
high fluctuation of the domain sizes. Only after the node density is higher than ten nodes per transmission radius we see a stable behavior.

As shown previously, for fixed infrastructures, standard gossiping has a convergence time for computing an average value across the network within accuracy e that requires $\Theta(n^2 \log e^{-1})$ messages. As expected, our simulations indicate that given enough gossiping time, the global variables converge faster for the case of mobile networks (see Fig. 4a).

5.4 Comparison to similar mechanisms

Although there has been a lot of research on clustering algorithms for wireless networks [1], the emphasis on resilience to node mobility has been more or less neglected due to various causes such as difficulty of the problem, experimental limitations such as the lack of an easy way to use/deploy mobile test-beds, scarcity of envisioned applications for civilian use etc. We selected two clustering algorithms that were specifically designed to cope with node mobility, to compare ASH with. The first one is called RCC (Random Competition based Clustering) [30]. The nodes use timers before broadcasting recruit messages in order to reduce contention. If multiple recruit broadcast messages are received, the rule of the lower-id takes place—the node with a lower node id becomes cluster head. RCC is a multi-hop clustering scheme, so the chances of getting two cluster-heads within communication range with each other is lower compared to one-hop clusters. The other scheme that we chose to compare ASH with is the Distributed Group Mobility Adaptive Clustering Algorithm for Mobile Ad Hoc Networks (DGMA) [34]. Compared to RCC, it only forms one-hop clusters so it is more prone to be affected by re-shuffling due to topology dynamics. An important assumption it makes is that nodes use position information.

As seen in Fig. 8b, the influence of node mobility is tremendous for RCC and DGMA. *ASH* is almost not influenced at all by the increase of the the node mobility. This is a tremendous achievement since the re-clustering effect is one of the most detrimental effects.



(a) Average Cluster Size Variation for Various Node Speed.

(b) Cluster Head Motion for Various Node Speed.

Fig. 9 Algorithm comparison for various node density. Mobility model: RWP

An important aspect is to look at the number of nodes that change cluster assignment due to re-shuffling caused by node mobility. As seen in Fig. 8a, due to the fact that both RCC and DGMA produce much less stable clusters and member nodes change their assignment much more frequently, the *churn* metric shows that *ASH* is around five times more stable. One would expect node speed to affect the same metric more than the node density. In Fig. 9a we notice a much more pronounced influence of an increased node density compared to an increased node speed. Again, *ASH* is significantly more stable. The motion metric though shows no significant difference between the competing algorithms for an increased node density (Fig. 9b).

When searching for competing algorithms, we had a hard time finding one that targets mobile networks. Additionally, most of the algorithms are either simplistic or make use of strong assumptions such as having knowledge of node positions. The reason we think both competing algorithms perform considerably worst compared to *ASH* is due to the underlying assumption that the network is always mobile. There is no time for them to converge to a stable overlay then slowly adapt to small changes in topology.

Both *RCC* and *DGMA* algorithms have $O(N)$ message complexity. It means they require a low message complexity for the control traffic needed to maintain the clusters. An important property of *ASH* is that all algorithmic steps are piggybacked on one underlying gossiping scheme. The creation of the gradient, the election of the cluster-head, the return of mass between clusters etc. are not adding-up to message complexity. On the other hand, when considering broadcasting, the number of messages that are received by each node is a function of the local node density ρ_i. The message complexity for *ASH* is $O(N)$ when no messages are lost. For the case of multicasting (it models the broadcast with message loss), the number of messages that are received by each node is a fraction of node density $K \cdot \rho_i$ where K is the fraction of nodes that receive the messages at one gossiping step.

5.5 Discussion

ASH shows a number of attractive features making it a good choice for the design of networking algorithms. First of all, *ASH* uses no node position or movement infor-

mation. Secondly, the scheme scales well with the size of the network. There are no limitations in terms of upper network size (our simulations involve thousands of nodes already). Thirdly, communication failures are modeled as nodes that travel with high speed and do not need special handling.

On the other hand, the only assumption that causes an overhead is the periodic neighborhood discovery protocol. From a practical point of view (i.e., implementing *ASH* on a real mobile network), a deployment will have to consider the analysis and the tweaking of several parameters such as weights (majority voting, gradient, round lengths, etc.), as well as their correlation with the deployment conditions (maximum node speed and transmission range). Also, of particular importance is the choice of the MAC protocol and the associated transmission scheme (TDMA, CSMA/CA, etc.) that best suits the application requirements in terms of latency, bandwidth, and tolerance to possible congestions and information loss.

A possible drawback comes from the fact that the protocol is based on local rules only, is that nobody keeps track of the routes towards other nodes. This leads to directionality in the communication, since data can only be sent from the nodes towards the cluster heads in a multi-hop fashion, following the gradient mechanism. The cluster head that forwards data towards a particular node needs to find a route to that node first. We plan to address this issue in the future.

Based on these considerations, a number of applications can be deployed on top of *ASH*. They target mainly data collection protocols but can be easily extended to localization protocols (given that a domain is fixed to some geographical region or close to some 'event'), clustering protocols (see Sect. 4) and further on to routing, etc. Additionally, one could develop a functional partitioning of the network, where nodes in different domains perform different actions (nodes in one domain surrounding an event can collect data, while nodes in another one can perform in-network processing of the data).

All these possibilities can be achieved by virtue of the stability exhibited by the *ASH* algorithm on top of the raw network mobility. The two metrics used for analysis of the stability of *ASH* (the mobility and variation metric) showed that it is insensitive to variations in the maximum speed of the nodes (cf. Fig. 4b). In terms of the values needed for weights, the domain are stable and do not increase the fluctuate of the shapes with the increase of node speed (see Sect. 5.3).

The motion metric does not describe the motion of the cluster heads. It describes the motion of the centroids for each cluster. It it a measure of the aggregate movement of the clusters per unit of time (seconds in our scenarios). The stabilizing properties of *ASH* are more evident for high node speed where the motion of the centroids is follow a logarithmic like curve.

6 Related work

In *ASH*, individual mobile nodes are using simple behavioral rules (i.e., periodic local exchange of a set of variables) to generate a pattern at the collective level (i.e., a static overlay) that is more intricate than the simple, one-hop interactions from which it emerges. From this perspective, *ASH* resembles algorithms met in the area of

complex systems [26], such as the techniques inspired by biological systems, which are based on simple local interactions and are fully decentralized. For example, the *firefly-inspired synchronization* [29] has several striking features that make it attractive for large-scale networks. To synchronize with each other, nodes execute very simple computations and interact in a simple manner, maintaining no internal state regarding neighbors or network topology. The synchronism provably emerges in a completely decentralized manner, without any explicit leaders and regardless of the starting state. The algorithm is very robust to network topology changes. On the other hand, *desynchronization* is the logical opposite of synchronization; instead of nodes attempting to perform periodic tasks at the same time, nodes perform their tasks at moments in time equally spaced apart from each other. Desync [11] is such a self-maintaining desynchronization primitive and achieves desynchronization in a single-hop network. Other types of emergent algorithms, similar to *ASH*, exist as well. In the MIT *amorphous computing project* [2], researchers used the *peer pressure* algorithm to regularize the regions of a surface covered by *smart paint*, smooth the edges and fill in the surface holes. Similar to the assumptions *ASH* makes, the myriad of computational elements are uniform-randomly distributed throughout the smart paint.

ASH uses *gossiping* as the basis for its communication mechanism. Gossiping (also known as *epidemic algorithms*) is a simple randomized procedure, finding its use in disseminating information in large-scale networks. It was firstly introduced to maintain consistency for distributed databases when performing updates [8], offering a resource-efficient and robust alternative to complex deterministic algorithms. From a communication perspective, the underlying communication mechanism of *ASH* is related to the work presented in [24], where epidemic algorithms were proposed to forward information in mobile networks with intermittent communication links. Similar mechanisms, such as *random walks*, are explored in more recent work [4]. The focus in these approaches is on algorithms that reliably spread information in large-scale networks, while minimizing the energy usage [36].

When dealing with the challenges introduced by network mobility, there exist few alternative techniques to constructing a static overlay (as *ASH* proposes) or a network hierarchy in general. The simplest one is *flooding*: flooding small-sized packets in the network is a common practice in routing algorithms such as DSR [21], but induces big overheads for large networks. A second alternative is using knowledge on the geographical position of the nodes—as is the case of geographical routing algorithms [12,35]. Geographical routing algorithms have the default assumption that sensor nodes have a means to determine their locations and usually come with the overhead that the position of the final destination of a message is explicitly included in the message. Unfortunately, in the case of wireless sensor networks, location information acquired through GPS is usually expensive energy-wise and unavailable for indoor applications.

7 Conclusions

The problem of node mobility is one of the most difficult topics in wireless networks research. There are many approaches toward solving it, unfortunately with rather poor results in terms of assumptions and performance.

In this paper, we introduced *ASH*, a mechanism for constructing a quasi-static overlay network on top of a mobile infrastructure. By the means of information diffusion and a mechanism inspired by the equilibrium of gases inside a container we are able to partition the network into stable domains that "hide" the node mobility for certain classes of applications. Additionally, we introduced a design example in the form of the *ASH-Cluster* clustering mechanism. Algorithms developed for static wireless networks can be applied on top of the overlay *ASH* builds, even if the support network is large-scale and highly-mobile. This is an important property, as most of the algorithms developed for wireless sensor networks do not work properly for high network mobility.

ASH provides two major improvements with respect to other approaches. First, *ASH* is a fully decentralized scheme as it makes use only of local interactions between nodes. Secondly, it uses no strong assumptions like knowledge about absolute or relative node positions (e.g., through GPS or any other estimator), correlated (group) motion, and motion prediction. As a result we can apply *ASH* to totally random motion scenarios. Besides the practical application that we showcase, *ASH* is an example of applying self-organizing, self-assembling principles to algorithms in mobile wireless networks.

The current paper is presents a mechanism that makes use of node mobility for the creation of quasi-static overlays in wireless networks. We plan to address some of the short-comings that we encountered: mass loss estimation, increased resistance to node failures and extend our work toward localizing some of the virtual domains around fixed geographical positions and/or events.

References

1. Abbasi AA, Younis M (2007) A survey on clustering algorithms for wireless sensor networks. Comput Commun 30(14–15):2826–2841
2. Abelson H, Allen D, Coore D, Hanson C, Homsy G, Knight TF Jr, Nagpal R, Rauch E, Sussman GJ, Weiss R (2000) Amorphous computing. Commun ACM 43(5):74–82
3. Akyildiz IF, Estrin D, Culler DE, Srivastava MB (eds) (2003) In: Proceedings of the 1st international conference on embedded networked sensor systems, SenSys 2003, Los Angeles, California, USA, November 5–7, 2003. ACM
4. Avin C, Brito C (2004) Efficient and robust query processing in dynamic environments using random walk techniques. In: IPSN 2004, pp 277–286
5. Bastien C, Michel D (2005) Cellular automata modeling of physical systems. Cambridge University Press, Cambridge
6. Bettstetter C (2001) International workshop on modeling analysis and simulation of wireless and mobile systems. In: MSWIM 2001, pp 19–27
7. Camp T, Boleng J, Davies V (2002) A survey of mobility models for ad hoc network research. Wirel Commun Mobile Comput 2(5):483–502
8. Chaintreau A, Le Boudec J-Y, Ristanovic N (2009) The age of gossip: spatial mean field regime. In: SIGMETRICS/Performance 2009, pp 109–120
9. Chu T, Nikolaidis I (2004) Node density and connectivity properties of the random waypoint model. Comput Commun 27(10):914–922 (Protocol engineering for wired and wireless networks)

10. De Rosa M, Goldstein S, Lee P, Pillai P, Campbell J (2008) Programming modular robots with locally distributed predicates. In Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on pages 3156–3162. IEEE
11. Degesys J, Rose I, Patel A, Nagpal R (2007) Desync: self-organizing desynchronization and tdma on wireless sensor networks. In: IPSN 2007
12. Dhurandher SK, Singh GV (2005) Weight based adaptive clustering in wireless ad hoc networks. In: ICPWC 2005, pp 95–100
13. Dimakis A, Rabbat M (2005) Gossip and message-passing algorithms for sensor networks. IEEE Trans Inf Theory 58(3):1731–1742
14. Goldstein Seth C, Campbell JD, Mowry TC (2005) Programmable matter. IEEE Comput 38(6):99–101
15. Grossglauser M, Tse David NC (2002) Mobility increases the capacity of ad hoc wireless networks. IEEE/ACM Trans Netw 10(4):477–486
16. Gupta P, Kumar PR (2000) The capacity of wireless networks. IEEE Trans Inf Theory 46(2):388–404
17. Heinzelman WR, Ch A, Balakrishnan H (2000) Energy-efficient communication protocol for wireless microsensor networks. In: HICSS 2000, pp 3005–3014
18. Hnat TW, Sookoor TI, Hooimeijer P, Weimer W, Whitehouse K (2008) Macrolab: a vector-based macroprogramming framework for cyber-physical systems. In: SenSys 2008
19. Iwanicki K, van Steen M (2010) Gossip-based self-management of a recursive area hierarchy for large wireless sensornets. IEEE Trans Parallel Distrib Syst 21(4):562–576
20. Jardosh A, Belding-Royer EM, Almeroth KC, Suri S (2003) Towards realistic mobility models for mobile ad hoc networks. In: MobiCom 2003
21. Johnson DB, Maltz DA (1996) Dynamic source routing in ad hoc wireless networks. In: Mobile Computing 1996
22. Karpelson M, Wei G-Y, Wood RJ (2009) Milligram-scale high-voltage power electronics for piezoelectric microrobots. In: ICRA 2009
23. Kempe D, Dobra A, Gehrke J (2003) Gossip-based computation of aggregate information. In: FOCS 2003
24. Khelil A, Becker C, Tian J, Rothermel K (2002) An epidemic model for information diffusion in manets. In: MSWiM 2002, pp 54–60
25. Lee K, Hong S, Kim SJ, Rhee I, Chong S (2009) Slaw: a mobility model for human walks. In: INFOCOM 2009, IEEE, pp 855–863
26. Mitchell M (2009) Complexity: a guided tour. Oxford University Press, Inc., New York
27. Royer EM, Melliar-Smith PM, Moser LE (2001) An analysis of the optimum node density for ad hoc mobile networks. In: ICC 2001, pp 857–861
28. Sarwate AD, Dimakis AG (2009) The impact of mobility on gossip algorithms. In: INFOCOM 2009, IEEE 10
29. Werner-Allen G, Tewari G, Patel A, Welsh M, Nagpal R (2005) Firefly-inspired sensor network synchronicity with realistic radio effects. In: SENSYS 2005
30. Xu K, Hong X, Gerla M (2002) An ad hoc network with mobile backbones. In: Communications, 2002. ICC 2002. IEEE international conference on vol 5. IEEE, pp 3138–3143
31. Yamins D (2008) A theory of local-to-global algorithms for one-dimensional spatial multi-agent systems. PhD thesis, Harvard, Cambridge, MA, USA, Adviser-Nagpal, Radhika
32. Youssef MA, Youssef A, Younis MF (2009) Overlapping multihop clustering for wireless sensor networks. In: IEEE transactions on parallel and distributed systems, pp 1844–1856
33. Yu JY, Chong PHJ (2005) A survey of clustering schemes for mobile ad hoc networks. IEEE Commun Surv Tutor 7(1):32–48 (qtr. 2005)
34. Zhang Y, Ng JM, Low CP (2009) A distributed group mobility adaptive clustering algorithm for mobile ad hoc networks. Comput Commun 32(1):189–202
35. Zorzi M, Rao RR (2003) Geographic random forwarding (geraf) for ad hoc and sensor networks: energy and latency performance. In: IEEE Transactions on Mobile Computing, 2003
36. Zuniga M, Avin C, Hauswirth M (2010) Querying dynamic wireless sensor networks with non-revisiting random walks. In: EWSN 2010, pp 49–64