ASH: Tackling Node Mobility in Large-Scale Networks

Andrei Pruteanu, Stefan Dulman, Koen Langendoen
Delft University of Technology, The Netherlands
Emails: {a.s.pruteanu,s.o.dulman,k.g.langendoen}@tudelft.nl

***Abstract*—With an 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. 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.**

***Index Terms*—static overlay, mobile networks, clustering algorithm, self-adaptive networks, self-configuring networks**

I. 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 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 [1], [2]. 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 [3]), 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* [4], swarms of tiny robots [5] and *amorphous computing* [6], 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 [7]–[9].

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 Figure 1). 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 together 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 our *domains*, 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 Section IV). *ASH* is – by design – very robust to node and link failures. It is based on a combination of gossiping, which is topology agnostic, and a periodic adjustment procedure that reconstructs local state based on the actual neighborhood. In 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 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 exhibits speeds orders of magnitude lower than the average speed of the nodes. From this perspective, if applications are targeted at the

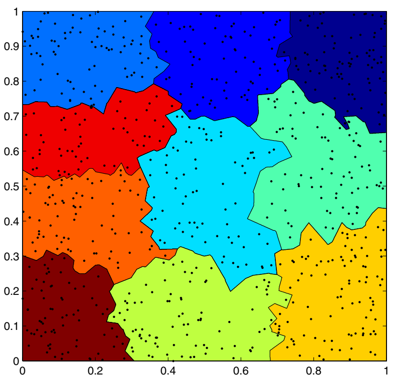

Fig. 1. ASH overlay (colors mark different domain IDs); node positions superimposed (black dots).

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 Section II we describe related work. In Section III we present an overview of *ASH*, while in Section IV we introduce a clustering mechanism as an application example. We analyze the performance of *ASH* in Section V. Finally, Section VI concludes the paper.

II. 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* [10], 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* [11] 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 [12] 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* [6], 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 [13], 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 [14], 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 [15]. The focus in these approaches is on algorithms that reliably spread information in large-scale networks, while minimizing the energy usage [16].

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 [3], 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 [17], [18]. 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.

III. 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 interactions between nodes, 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 Figure 1.

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 is orders of magnitude lower than the mobility of the nodes themselves (this is somewhat similar to immiscible fluids interaction modeling with cellular automata [19]).

In order to explain *ASH* in detail, we consider the abstraction of network communication occurring in *rounds* – similarly to the work presented in [20]. Rounds are fixed-length time intervals with each node acting 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 numbers 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 Section III-B);
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 Section III-C);
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 Section III-D).

The second action (residual mass return) can lead to mass loss in the domains ($\sum_{i \in \mathcal{S}} m_i < M$); 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 Section III-E for a solution.

Algorithm 1 *ASH* algorithm.

```
 1: function ASH($\mathcal{N_S}$, $n_r$, $\mathcal{M}$)
          $\mathcal{N_S}$ – number of domains
          $n_r$ – number of rounds
          $\mathcal{M}$ – initial mass for all domains
 2:    for all $\mathcal{N_S}$ domains $k$ do                    ▷ initialization
 3:        Pick random unassigned node $j$
 4:        node $j \leftarrow$ domain $k$
 5:        node $j \leftarrow$ mass $\mathcal{M}$
 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
```

A. Initialization

ASH considers a fixed number of domains at start ($\mathcal{N_S}$). In the initialization part, the algorithm starts by randomly assigning domain id k ($k = 1..\mathcal{N_S}$) and mass $\mathcal{M}$ to a $\mathcal{N_S}$ 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 area is covered by domains.

If we consider the clustering algorithms for comparison (although *ASH* is a more general framework!), the fixed number of domains might seem a limitation. Almost all existing clustering algorithms dynamically decide on the number of clusters in the network. This is because clustering is being used as a hierarchical way of controlling a large network by aggregating information. *ASH* can be easily extended to comply with this behavior, by allowing domains to be dynamically created at run-time: when the pressure of 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.

With the current approach, we are also covering a class of applications less often addressed: having a constant number of multihop domains leads to the so called functional partitioning of a network. Each domain will be associated 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, without requiring keeping track of each node separately.

B. Domain ID Selection

Each node will decide its domain ID ("domain color" in Figure 1) 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 $p_{i,k}$. The node i will compute a series of values $\theta_{i,k}$:

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

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 dissolves 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 complete network. This undesirable effect is disabled by introducing a *domain leader* (see Section IV), which will be prevented from changing its domain ID.

C. Residual Mass Return

This phase involves all nodes that decided, in the previous algorithmic phase, to change their domain ID. The nodes crossing to a different domain need to adjust their mass variable: they need to return the current mass value to the old domain and enter the new domain with mass 0. The diffusion phase that follows will make sure that mass redistributes equally in both old and 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 hand them its mass. This approach works in most of the 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 and being used. 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 volatile network, and the repair mechanism described in Section III-E is easy to implement.

D. Diffusion

The purpose of this phase in the *ASH* algorithm is to rearrange the mass allocated to each node in a domain such that all nodes share the same view on what the value of the pressure in the domain is. As 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 Section V).

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, with its functionality being assured by the *media access control* (MAC) mechanism. For the implementation of the gossiping mechanism, an acknowledgment mechanism for the messages being sent is assumed in place. This needs not be perfect as the pressure correction mechanism described in Section III-E compensates the effects of message loss.

Similarly to the *Push-Sum gossiping algorithm* [21], 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 $\left\{\frac{m_{i,t}}{2}, \frac{\omega_{i,t}}{2}\right\}$. During that round t, the node receives updates $\left\{m^r_{j,t}, \omega^r_{j,t}\right\}$ 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 [21], 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 and complexity. It has been proved that any kind of mobility is beneficial, especially the fully-random one as is the case with our scenarios [22]. Different mobility patterns can have significantly different effects on the convergence of distributed algorithms such as gossiping [23]. 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 information is spread across the complete domain, the diffusion phase may actually consist of several gossiping rounds. The right number of rounds depends on the application, more exactly, on the average speed of the nodes and their density, and the desired domain stability. We evaluate this dependency via simulation and present the results in Section V. On the other hand, one is not limited to gossiping alone. Broadcasting each message might come as a natural solution in many wireless networks as well. From the perspective of dissemination speed, gossiping presents the worst case scenario and we chose to use it to characterize the lower limit of the performance of our algorithm.

E. 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

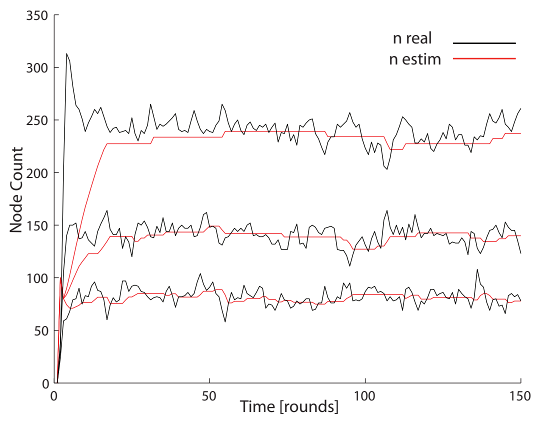

Fig. 2. Domain size evolution (green = real number of nodes in a domain; red = estimated domain size).

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 kept at small levels even for high mobility, across all domains. Nonetheless, if no measures are taken, the mass in each domain will constantly drop towards $-\infty$. In the case of real networks mass loss may also occur due to failures. For example, nodes that suddenly crash or messages getting lost in the diffusion phase lead to additional mass loss. Thus, providing a mechanism for solving the mass loss issue, leads not only to a solution to 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 numbers, 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 the loss.

ASH-NetSize is making use of the gossiping mechanism [22] to estimate the domain size at a moment in time. The idea behind gossiping algorithms is that they are able to compute a mean value of some shared variable ϕ through all the n nodes of the 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 ($\phi_k \leftarrow 1 - n_e$). By using gossiping, after a number of rounds in time, 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 exactly 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 2 shows that the estimation algorithm is able to follow the fluctuations of the domains quite closely, smoothing out temporary "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 Section III-D). As a result the average mass in the domains will stay around a desired value, steadily decreasing with time as an effect of nodes leaving and periodically increasing due to the domain-leaders injecting mass.

IV. Application Example – *ASH-Cluster*

In this section we show how the quasi-static domains of *ASH* can be used by applications. We revert to the example of clustering, for which a large number of algorithms have been surveyed and compared in works such as [24], [25]. The performances of these clustering algorithms are compared on a multitude of metrics: communication overhead, power balancing, re-clustering ripple effect, cluster formation time, etc. The large 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 [26].

The domains defined by *ASH* can be readily used as clusters, hence, *ASH-Cluster* 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* determines the domains (clusters) independently of the decision of electing a cluster head. This makes it superior to the large majority of existing clustering algorithms, in the sense that the 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 Figure 3), 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 ($\sum_{j \in D_i} n_{i,j}$). A low pass filter is also being applied. Figure 3 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.

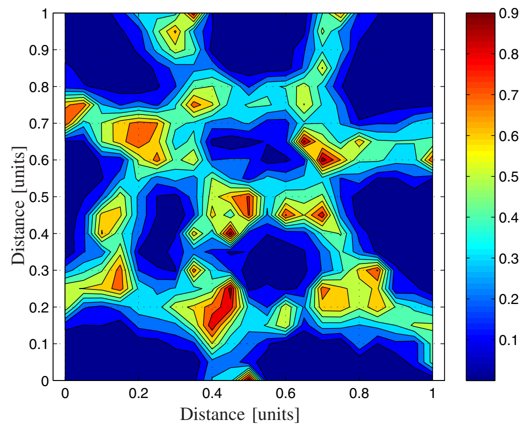

Fig. 3. ASH gradient.

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 in [27], via the gradient mechanism described below. The communication between the cluster heads is similar to the one used by the LEACH protocol [28]. The cluster heads form a network backbone (a spanning tree routed at the gateway) and can 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 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.

V. *ASH* Algorithm Analysis

To evaluate the stability of *ASH*, we need some means of characterizing the shapes of the domains. We introduce two metrics to measure the stability of domains over time: the motion of the centroid of the node positions (*motion metric*) and the domain shape variation (*variation metric*). The domains in *ASH* "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, and the variation metric captures the fluctuations in size.

The *motion metric* is equal to the distance traveled by the centroid of the node positions, in a domain. The *centroid* is defined as $(x_c, y_c) = \left(\frac{1}{n}\sum_i x_i, \frac{1}{n}\sum_i y_i\right)$, $i = 1..n$, 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:

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
```

$m_2 = \overline{d}_{t+1} - \overline{d}_t$. 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 1000 mobile nodes deployed in a square space with the edge of 1 $unit$. The transmission range of the nodes is set to 0.1 $units$ and a circular disk communication model is assumed. We configure *ASH* to operate with 10 domains in our simulations (cf. Figure 1). 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 that are being used. To preserve the generality, we model all low-level errors as mass loss, with the implication that the simplified assumptions do not affect the overall stability.

A node moves through space with a speed ranging from a minimum of 0.01 $units/round$ to a maximum of 0.1 $units/round$. In our simulations, we use three mobility models: *Random Walk* [29], *Random Direction* [30] and *Random Waypoint* [29]. 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). Each experiment consisted of simulations running for 500 time $rounds$, for each mobility case. The maximum speed was varied across simulations to achieve different characteristics for mobility.

In the Random Walk model [29], 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 on the interval $[0..2\pi]$. The speed values ϑ follow a uniform distribution. Once the node reaches a destination,

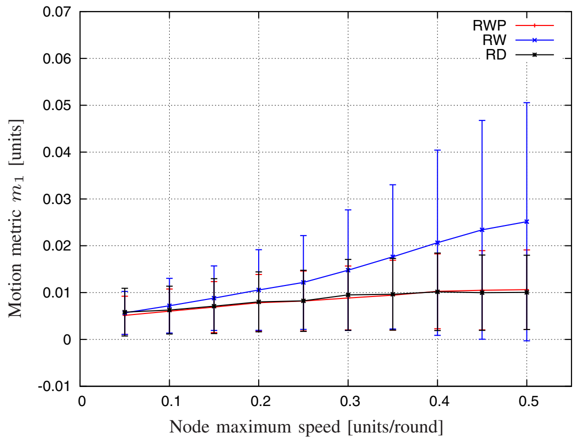

Fig. 4. Motion metric for different speeds.

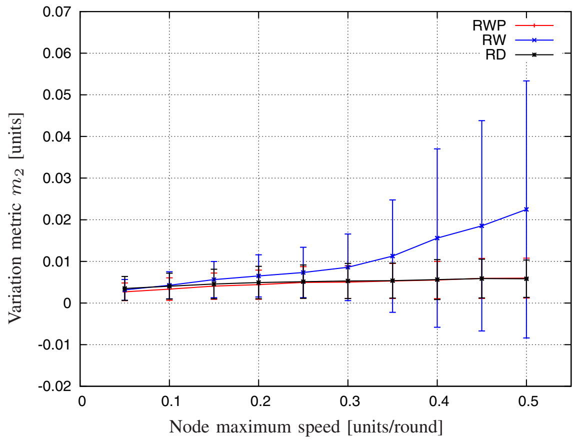

Fig. 5. Variation metric for different speeds.

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 [30] 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 [29] a node randomly chooses a destination point in the deployment area, moves with constant speed v (chosen uniformly between v_{min} and v_{max}) on a straight line then pauses for a random time before it again chooses a new destination.

A. *Influence of Network Mobility*

The Random Waypoint (RWP) mobility leads to the formation of more stable domains when compared to other mobility patterns (see Figure 4 and Figure 5). The results are similar to well known experiments, such as those presented in [31]. 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 more 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.

Figure 4 and Figure 5 show that the maximum speed of the nodes has basically no influence on the overall stability of *ASH* for the first two mobility models. In fact, both metrics exhibit a similar behavior. This is one of the most interesting characteristics 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 is because, for the case of a high maximum speed, the second term in the domain selection formula leads to domains having irregular shapes, affecting the stability of the algorithm. Up to a speed of 0.2 $units/round$, the metrics are relatively constant for all models. Above this limit, the system loses stability in the case of Random Walk and some domains grow, hence the larger values for the motion and variation metrics. The authors of [32] showed that the idea of a critical radius (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. In uniform mobility models [33], 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.

B. 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 for individual nodes in the one-hop vicinity when disseminating common information. On the receiving side however, this is only possible if all nodes are always listening, or synchronized to the sender. Hence the reduction in energy consumption at the sender due to fewer transmissions is leading to more energy consumption at receivers, along with high chances of packet collisions and information loss. Still, 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, in every diffusion phase (equal for all the nodes and not synchronized) the nodes are sending a “Hello” message. At all times, we assume that only a subset of the one-hop neighbors will receive the packets. To compensate for this reduction in diffusion speed, several gossiping rounds are performed. As seen in Figure 7, when all the nodes are listening the domains become more stable: the dashed lines. Otherwise, *ASH* compensates the reduction

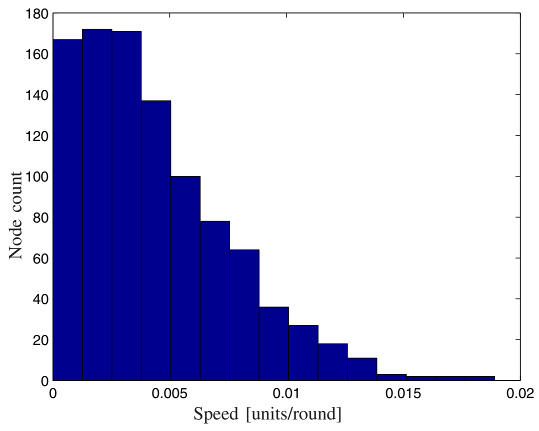

Fig. 6. Histogram center of mass delta average speed

of diffusion speed with more rounds of information exchange with randomly picked nodes to achieve the same domain stability. With 5 gossiping rounds we are able to achieve the same results as with broadcasting.

Figure 6 shows the distribution of domains average movement (the m_1 metric) through the simulation. The average speed of the center of mass of the domains is more than one order of magnitude smaller than the average speed of a node.

Another aspect of interest is the diffusion speed – especially because *ASH* uses gossiping for the dissemination of information. To compensate for the high mobility of the nodes and the imperfections of the gossiping mechanism, several gossiping rounds are performed every diffusion phase. As seen in Figure 7, the performance achieved when using a perfect broadcasting mechanism is equaled by performing five or more rounds of gossiping during the diffusion round.

C. Domain Selection Stability

The domain selection mechanism (see Section III-B) combines majority voting and the difference between pressure levels. The two terms have different influences. Majority voting smooths out the edges at the border, while the difference of pressure mechanism ensures the dynamic equilibrium between the domains such that all of them will converge to similar sizes.

We determined the stability of the static overlay mechanism by varying the weight η, see Figure 8. 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. 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. Figure 8 confirms this by showing great

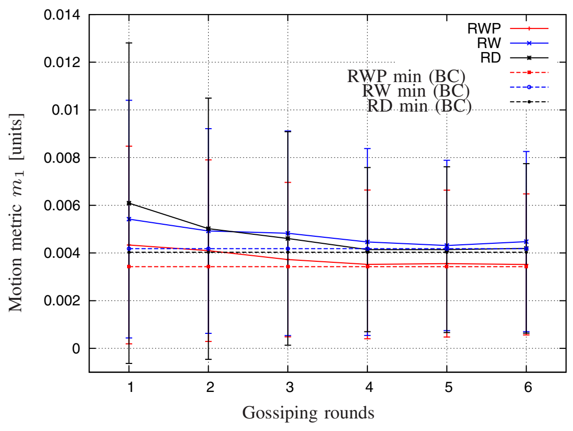

Fig. 7. Motion metric for several diffusion rounds.

fluctuations in the variation metric for $\eta \notin (0.3, 0.7)$ and stability for $\eta \in (0.3, 0.7)$.

Another important factor of the stability analysis is the network density. As seen in Figure 9 it has an important influence on the overall stability of *ASH*. Given a fixed transmission range of 0.1 $units$, at lower node densities, the variation metric shows a high fluctuation of the domain sizes. Only after the total number of nodes is higher than 250 we see a stable behavior. Although the domains do not disappear due to the restriction that we imposed to the domain leader software agent, *ASH* is becoming stable only after the average number of nodes is higher than $2 - 2.5$. An average neighborhood higher than 5 is not bringing a significant stability increase. In other words, the mechanism does not require a high network density to achieve stability.

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. Our simulations indicate that given enough gossiping the global variables converge to consistent values even for the case of mobile networks. *ASH-cluster* algorithm avoids potential problems caused by the uncertainty of the cluster-head assignments. Our algorithm works such that a node will send packets in a multi-hop manner via the gradient. There is no need to track the path towards the cluster-head or to follow it.

D. Comparison to Similar Mechanisms

Although they are not from the same class of applications, algorithms such as *Firefly* [11], *Desync* [12], *ACE* [34] and *ASH* use similar approaches (e.g., complex system behavior through simple node interactions, lack of network-level data structures, reduced local state, lack of time synchronicity, etc.) leading to robustness to the disappearance of nodes and links, reduced overhead and increased scalability.

Firefly and *Desync* are using broadcast messages to indicate firing events, while the cluster heads in *ACE* poll for cluster members. In the same manner, the nodes in the network running *ASH* send "Hello" messages every time round to

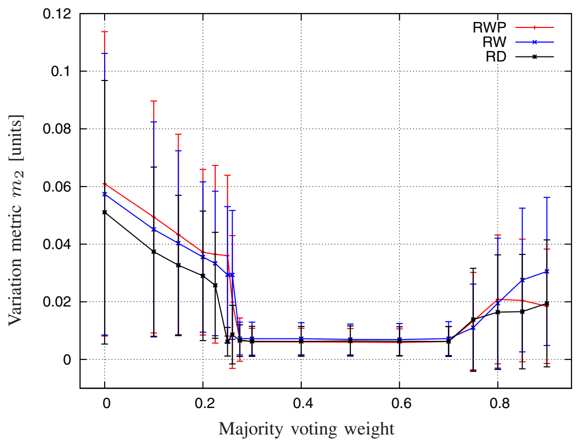

Fig. 8. Variation metric function of η.

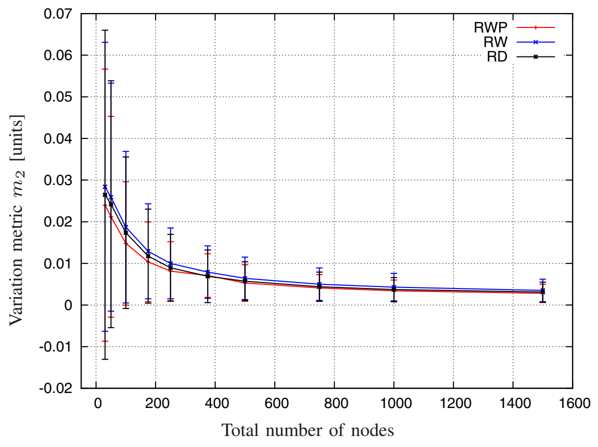

Fig. 9. Variation metric function of total number of nodes.

update neighborhood information. When comparing to *Firefly* and *Desync*, which do not maintain per-link and per-state information, *ACE* and *ASH* need to track the neighboring nodes (for *ACE* the "followers" subset); for *ASH* their count and their domain IDs. Like the other algorithms, *ASH* only considers the one-hop vicinity to reduce the amount of state information.

In all mentioned techniques time synchronization is not required. Still, except *Desync*, all the others use the concept of *time rounds* or *time iterations* to perform sequential operations such as state information refreshing. In terms of traffic overhead, *ACE* performs the worst. Nodes need to broadcast messages in several situations: cluster heads polling for followers, nodes promoting a new cluster head or its abdication, etc. Since the protocol was not designed for mobile nodes, the amount of traffic increases exponentially with the increase in the speed of the nodes. *ASH* solves this by means of gossiping, as described in Section III-D.

E. 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 position or movement information. 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). Third, communication failures are modeled as nodes that travel with high speed and do not need special handling.

On the other hand, the only assumption leading to 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 MAC protocol and associated transmission scheme (TDMA, CSMA/CA, etc.) that best suits the application requirements in terms of latency, bandwidth, and tolerance to possible congestion and information loss.

A possible drawback coming from the fact that the protocol is based on local rules only, is that nobody keeps track of the nodes positions. 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 addressing a particular node meets the problem of locating the node in the network first. We plan to address this issue as a next research step.

Based on these consideration, a number of applications can be envisaged to run 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 Section IV) 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 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. Figure 4). In terms of the values needed for weights, the domains shape is stable and slowly moving, for quite large ranges (see Section V-C).

VI. Conclusions

The problem of network 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 a first step in the direction of tackling high node mobility for the case of wireless sensor networks communication. 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] P. Gupta and P. Kumar, "The capacity of wireless networks," *Information Theory, IEEE Transactions on*, vol. 46, no. 2, pp. 388–404, mar 2000.

[2] M. Grossglauser and D. N. C. Tse, "Mobility increases the capacity of ad hoc wireless networks," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 477–486, 2002.

[3] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*, 1996.

[4] S. C. Goldstein, J. D. Campbell, and T. C. Mowry, "Programmable matter," *IEEE Computer*, vol. 38, no. 6, pp. 99–101, June 2005. [Online]. Available: http://www.cs.cmu.edu/ claytronics/papers/goldstein-computer05.pdf

[5] M. Karpelson, G.-Y. Wei, and R. Wood, "Milligram-scale high-voltage power electronics for piezoelectric microrobots," in *ICRA 2009*, 2009.

[6] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, Jr., R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss, "Amorphous computing," *Commun. ACM*, vol. 43, no. 5, pp. 74–82, 2000.

[7] D. Yamins, "A theory of local-to-global algorithms for one-dimensional spatial multi-agent systems," Ph.D. dissertation, Harvard, Cambridge, MA, USA, 2008, adviser-Nagpal, Radhika.

[8] T. W. Hnat, T. I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse, "Macrolab: a vector-based macroprogramming framework for cyber-physical systems," in *SenSys 2008*, 2008.

[9] M. De Rosa, S. C. Goldstein, P. Lee, J. D. Campbell, and P. Pillai, "Programming modular robots with locally distributed predicates," in *ICRA 2008*, 2008. [Online]. Available: http://www.cs.cmu.edu/ claytronics/papers/derosa-icra08.pdf

[10] M. Mitchell, *Complexity: A Guided Tour*. New York, NY, USA: Oxford University Press, Inc., 2009.

[11] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *SENSYS 2005*, 2005. [Online]. Available: http://www.eecs.harvard.edu/ssr/papers/sensys05-werner.pdf

[12] J. Degesys, I. Rose, A. Patel, and R. Nagpal, "Desync: self-organizing desynchronization and tdma on wireless sensor networks," in *IPSN 2007*, 2007. [Online]. Available: http://www.eecs.harvard.edu/ssr/papers/ipsn07-degesys.pdf

[13] A. Chaintreau, J.-Y. L. Boudec, and N. Ristanovic, "The age of gossip: spatial mean field regime," in *SIGMETRICS/Performance 2009*, 2009, pp. 109–120.

[14] A. Khelil, C. Becker, J. Tian, and K. Rothermel, "An epidemic model for information diffusion in manets," in *MSWiM 2002*, 2002, pp. 54–60.

[15] C. Avin and C. Brito, "Efficient and robust query processing in dynamic environments using random walk techniques," in *IPSN 2004*, 2004, pp. 277 – 286.

[16] M. Zuniga, C. Avin, and M. Hauswirth, "Querying dynamic wireless sensor networks with non-revisiting random walks," in *EWSN 2010*, feb 2010, pp. 49–64.

[17] M. Zorzi and R. Rao, "Geographic random forwarding (geraf) for ad hoc and sensor networks: energy and latency performance," in *Mobile Computing, IEEE Transactions on*, 2003.

[18] S. Dhurandher and G. Singh, "Weight based adaptive clustering in wireless ad hoc networks," in *ICPWC 2005*, 2005, pp. 95–100.

[19] C. Bastien and D. Michel, *Cellular Automata Modeling of Physical Systems*. Cambridge University Press (June 30, 2005), 2005.

[20] K. Iwanicki and M. van Steen, "Gossip-based self-management of a recursive area hierarchy for large wireless sensornets," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 562–576, 2010.

[21] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *FOCS 2003*, 2003.

[22] A. Sarwate and A. Dimakis, "The impact of mobility on gossip algorithms," in *INFOCOM 2009, IEEE*, 10 2009.

[23] A. Dimakis and M. Rabbat, "Gossip and message-passing algorithms for sensor networks." [Online]. Available: http://ewsn2010.uc.pt/files/tutorial2.pdf

[24] A. A. Abbasi and M. Younis, "A survey on clustering algorithms for wireless sensor networks," *Comput. Commun.*, vol. 30, no. 14-15, pp. 2826–2841, 2007.

[25] J. Yu and P. Chong, "A survey of clustering schemes for mobile ad hoc networks," *Communications Surveys Tutorials, IEEE*, vol. 7, no. 1, pp. 32 – 48, qtr. 2005.

[26] M. A. Youssef, A. Youssef, and M. F. Younis, "Overlapping multihop clustering for wireless sensor networks," in *IEEE Transactions on Parallel and Distributed Systems*, 2009, pp. 1844–1856.

[27] I. F. Akyildiz, D. Estrin, D. E. Culler, and M. B. Srivastava, Eds., *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003, Los Angeles, California, USA, November 5-7, 2003*. ACM, 2003.

[28] W. R. Heinzelman, A. Ch, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *HICSS 2000*, 2000, pp. 3005–3014.

[29] C. Bettstetter, "International workshop on modeling analysis and simulation of wireless and mobile systems," in *MSWIM 2001*, 2001, pp. 19 – 27.

[30] E. M. Royer, P. M. Melliar-Smith, and L. E. Moser, "An analysis of the optimum node density for ad hoc mobile networks," in *ICC 2001*, 2001, pp. 857–861.

[31] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483–502, 2002.

[32] T. Chu and I. Nikolaidis, "Node density and connectivity properties of the random waypoint model," *Computer Communications*, vol. 27, no. 10, pp. 914 – 922, 2004, protocol Engineering for Wired and Wireless Networks.

[33] A. Jardosh, E. M. Belding-Royer, K. C. Almeroth, and S. Suri, "Towards realistic mobility models for mobile ad hoc networks," in *MobiCom 2003*, 2003.

[34] H. Chan and A. Perrig, "ACE: An emergent algorithm for highly uniform cluster formation," in *EWSN 2004*, 2004.