Wireless Netw (2012) 18:381–399
DOI 10.1007/s11276-011-0406-z

P&P: an asynchronous and distributed protocol for mobile sensor deployment

Novella Bartolini · Annalisa Massini · Simone Silvestri

Published online: 24 December 2011

Abstract The use of wireless mobile sensors is of great relevance for a number of strategic applications devoted to monitoring critical areas where sensors can not be deployed manually. Mobile sensors can adapt their position on the basis of a local evaluation of coverage, thus permitting an autonomous deployment. Several algorithms have been proposed to deploy mobile sensors over an area of interest. The applicability of these approaches largely depends on a proper formalization of rigorous rules to coordinate sensor movements, solve local conflicts and manage possible failures of communications and devices. In this paper we introduce P&P, a communication protocol that permits a correct and efficient coordination of sensor movements in agreement with the PUSH & PULL algorithm. We deeply investigate and solve the problems that may occur when coordinating asynchronous local decisions in the presence of an unreliable transmission medium and possibly faulty devices such as in the typical working scenario of mobile sensor networks. Simulation results show the performance of our protocol under a range of operative settings, including conflict situations and irregularly shaped target areas. Furthermore, a performance comparison between the P&P protocol and one of the best solutions based on the virtual force approach, shows the superiority of our proposal in terms of deployment time, message exchanges and energy consumption.

N. Bartolini · A. Massini · S. Silvestri (✉)
Department of Computer Science, "Sapienza" University of Rome, Rome, Italy
e-mail: simone.silvestri@di.uniroma1.it

N. Bartolini
e-mail: bartolini@di.uniroma1.it

A. Massini
e-mail: massini@di.uniroma1.it

Keywords Wireless sensor networks · Mobile sensors deployment · Distributed coordination protocol

1 Introduction

Recent advances in micro-electro-mechanical systems, wireless communications and digital electronics have enabled the realization of *mobile sensors*, namely devices with sensing, communication, computation and locomotion capabilities. Such devices are able to perform complex tasks such as cooperatively monitoring environmental conditions, processing local data, communicating with other devices, and coordinating their movements over the Area of Interest (AoI) to meet specific requirements.

The coordination and locomotion capabilities make sensor networks particularly appealing in a wide range of applications. Differently from static devices, mobile sensors can be used to monitor inaccessible, unknown, hazardous or even hostile environments. Applications of mobile sensor networks include forest fire detection, pollutants dispersion tracking, volcano monitoring, urban search and rescue operations and target detection and localization.

The inaccessible and often hazardous scenarios typical of mobile sensor network applications, impede manual sensor positioning. Hence, typical initial sensor deployment methods include, depending on the operative setting, air-dropping from an aircraft, delivering in an artillery shell, throwing by a catapult or sending from a safe-location. The network initial configuration is unlikely to meet the application requirements, thus mobile sensors can exploit the locomotion capabilities in order to achieve a satisfactory deployment. Such a self-deployment phase of the network brings up the need of distributed algorithms and protocols for the coordination of sensor movements.

The critical conditions in which mobile sensors operate together with the sensor hardware characteristics, pose several challenges and requirements to the design of algorithms and protocols for mobile sensor deployment. On the one hand, the sensor power availability is limited, and the critical conditions in which mobile sensors operate do not permit the replacement of power supplies. On the other hand, for several applications, such as forest fire detection and urban search and rescue missions, responsiveness is of primary importance. As a result, the final deployment must be achieved in a short time while minimizing the energy consumption. Furthermore, mobile sensor networks are generally composed by hundreds or even thousands of devices. Deployment algorithms and protocols must show a good scalability, being able to achieve good performance even when a high number of sensor is deployed.

Finally, a challenge is also constituted by the commonly high rate of device failures in sensor networks. Deployment algorithms and protocols must show a high degree of fault tolerance, achieving an acceptable performance even if several nodes in the network cease to work. Notice that the device mobility can be exploited to create networks with self-healing capabilities, as sensors can coordinate themselves and move to replace faulty nodes, without resorting to human intervention.

This paper introduces for the first time the specification of a protocol, named P&P, that implements a deployment algorithm Push & Pull previously proposed in [3]. P&P defines the rules to deploy mobile sensors according to Push & Pull, giving particular emphasis to the decision problems that occur in the execution of distributed deployment algorithms aiming at positioning sensors according to a regular pattern.

Indeed, under the execution of the Push & Pull algorithm, several types of conflicts may occur as several sensors often compete to cover the same position. Sensors should be capable to solve such conflicts by means of only local interactions. Differently from the work proposed in [3], we explicitly investigate and solve the problems that may occur when coordinating asynchronous local decisions in the presence of an unreliable transmission medium and possibly faulty devices that characterizes the typical working scenario of mobile sensor networks. Furthermore we modified the pull activity proposed in [3] to make it more efficient and equally solid in terms of algorithm termination and coverage capabilities.

The dynamic hierarchy of roles determined by the execution of the protocol P&P is a novel technique which significantly differs from any previously proposed approach. Previous approaches fall into one of two main categories, as they are either inspired by molecular physics [5, 6, 8–10, 13, 14, 22] or by computational geometry [12, 15–18]. Unlike those, the P&P protocol aims at spreading sensors according to precise coordinated movements along the tiles of a regular grid. The deployment of sensors over the AoI is performed by either letting sensors position themselves over grid points, also called snap positions, or making them act as slave of sensors previously snapped, and moving them towards uncovered areas.

Furthermore, differently from previous approaches which are tipically round based, P&P works in a completely asynchronous manner. It is based on the autonomic computing paradigm. It completely delegates to the single sensors every decision regarding movements and action coordination. This way self-organization emerges without the need of external coordination or human intervention as the sensors adapt their position on the basis of their local view of the surrounding scenario.

Given the absence of a centralized coordination unit, and the lack of synchronization, sensors have a primary role in the realization of the algorithm actions. Therefore, the design of the related coordination protocol, which is the aspect we address in this paper, is particularly challenging.

Simulation results show the performance of our protocol under a range of operative settings, including conflict situations, irregularly shaped target areas, and node failures. Furthermore, we compare the P&P protocol with one of the best solutions based on the virtual forces approach proposed in [11]. The results show the superiority of the P&P protocol in terms of deployment time, number of message exchanges and energy consumption.

2 Related work

The problem of mobile sensor deployment has been largely studied in the literature. Previously proposed approaches can be roughly classified in algorithms based on the *virtual force model*, algorithms inspired by *computational geometry models* and algorithms that aim at creating *pattern* based deployment.

Algorithms based on virtual forces [5, 7–9, 11, 14, 22] are inspired by the molecular interaction of particles. As in the case of particle interactions, each sensor exerts a virtual force on the others, that can be either attractive or repulsive, depending on the distance. By modeling the virtual force that acts on each sensor and moving sensors accordingly, the network is spread on the AoI so as to expand and improve the initial deployment. Despite its simplicity, this approach has several drawbacks. For example, most of the proposals making use of virtual force models are not able to achieve a final stable deployment, since the sensors do not find a stable position in which all the forces are balanced. Furthermore, the definition of the virtual forces requires the manual tuning of several key parameters which strongly affect the algorithm performance. Differently from the other

solutions based on virtual forces, the algorithm proposed in [11] has a guaranteed termination, even if it requires the manual tuning of several key parameters. We use this algorithm for performance comparisons with our protocol in Sect. 9.

The approaches inspired to models of the computational geometry typically resort to geometrical constructions, such as the *Voronoi diagrams* and the *Delaunay triangulations*, in order to guide sensor movements through the AoI. In particular, Voronoi diagrams [17, 18] are used to determine the coverage responsibility of each sensor, and to locally determine possible coverage holes. Likewise, Delaunay triangulations [12], are also used to guide the device movements according to the position of their Dealunay neighbors. Such approaches, although are able to guide sensor movements according to coverage considerations, they cannot be adopted in the presence of irregular (non convex) AoIs and they do not give any guarantees of coverage completeness and uniformity.

Pattern based approaches [15, 19–21] aim at deploying sensors according to a predefined pattern. Sensors move from their initial locations to some key positions in a pattern. These approaches try to combine the advantages of well known patterns, proposed for static sensor deployment, with the flexibility of mobile sensors networks. Nevertheless, most of the pattern based algorithms proposed in the literature, rely on the presence of a central unit [15, 19] which is responsible for the calculation of the pattern positions and, in some cases, even coordinates sensor movements. Few works investigate the possibility of realizing a pattern based deployment by means of a completely distributed algorithm [21], but they typically need global sensor synchronization which results in long deployment time. Finally, the solutions proposed in [2] and [3] provide asynchronous density driven distributed algorithms that uniformly deploy sensors according to a regular grid pattern.

The design of asynchronous distributed deployment algorithm for mobile sensor networks requires the proper formalization of rigorous rules to coordinate sensor movements, solve local conflicts and manage possible failures of communications and devices. Differently from previous proposals which only focus on the design of the coordination algorithm without entering the details of the communication scheme, in this paper we propose a communication and movement coordination protocol for the algorithm proposed in [3] called P&P. This protocol is completely distributed, it does not require sensor synchronization and it is able to achieve the desired sensor deployment even in presence of faulty sensors. Moreover, by fulfilling the requirements of the algorithm Push & Pull it provides a guaranteed termination and a complete coverage, if a sufficient number of sensors is available.

3 The Push & Pull algorithm

The purpose of Push & Pull is to let sensors form a hexagonal tiling that constitutes a complete coverage of the AoI and a connected network deployment. Notice that the hexagonal tiling corresponds to a triangular lattice arrangement, that is the one that guarantees at the same time network connectivity, optimal coverage extension and density, as discussed in [4]. The design of Push & Pull is based on the idea to make some sensors stick to the hexagonal grid points and let the others uniformly distribute over the whole AoI. According to the Push & Pull algorithm, sensors are involved in four basic activities executed in an interleaved manner: (1) **Snap**, described in Sect. 5, which makes the sensors move and stick to the grid points of the hexagonal tiling, (2) **Push**, described in Sect. 6, which allows the flow of non-snapped sensors towards low density areas, (3) **Pull**, described in Sect. 7, which attracts non-snapped sensors toward coverage holes, and (4) **Merge**, described in Sect. 8, which makes several grid portions merge into a unique regular hexagonal tiling. A fifth activity, **role exchange**, described in Sect. 6.3, is introduced to balance the energy consumption among the available sensors. Note that the P&P protocol we propose in this paper implements these activities without the need of global synchronization among sensor, as it will be explained in the next sections. More details on the activities at the basis of the Push & Pull algorithm can be found in [3].

For the sake of clearness, in Fig. 1 we give an example of the protocol execution over a rectangular AoI. In the figure the values of the sensor IDs are shown. Different kinds of arrows are associated to the different protocol activities. We refer to this figure throughout the paper to describe the main activities implemented by our protocol.

4 The P&P protocol

The implementation of the Push & Pull algorithm requires the definition of a protocol for the local coordination of the sensor activities.

The coordination protocol provides the rules to solve contentions that may happen in several cases. For example, two or more snapped sensors can decide to issue a snap command to different sensors towards the same hexagon tile or the same low density hexagon can be selected by several snapped sensors as candidate for receiving redundant sensors. These contentions are solved by properly scheduling actions according to message time-stamps and by advertising related decisions as soon as they are made.

The P&P protocol is designed to minimize energy consumption entailing a small number of message exchanges,

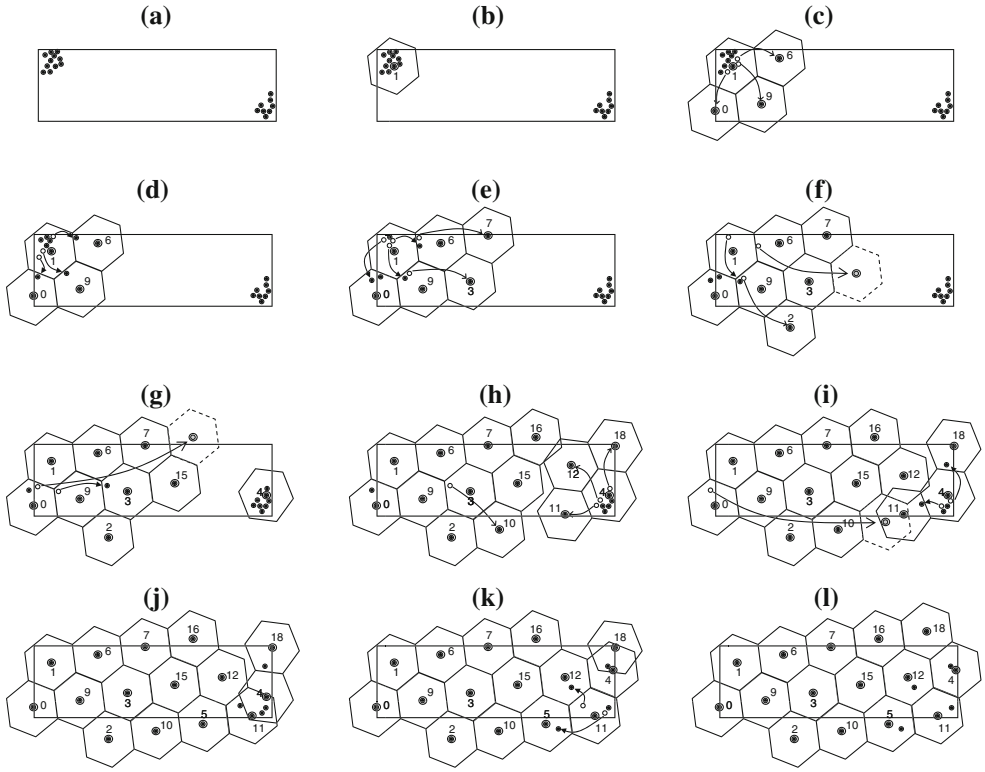

Fig. 1 Algorithm execution: an example

which is possible because the algorithm decisions are only based on a small amount of local information. Furthermore, we assume that P&P works over a communication protocol stack which handles possible transmission errors and message losses by means of timeout and retransmission mechanisms. Therefore the treatment of occasional message losses at the underlying protocol level implies the occurrence of delays in the corresponding messages at the P&P level that are dealt by P&P with proper timeout mechanisms.

Before we enter the details of the protocol P&P we introduce some definitions. We consider a set of equal sensors endowed with location determination, boolean sensing and isotropic communication capabilities. Notice that location awareness (usually obtained by means of GPS devices) is only necessary in the case of sensor deployment over a specific target area. If sensors are to be deployed in an open environment, the assumption of location determination capability can be removed, as in other works in the area [6].

The deployment consists in realizing a hexagonal grid with side length l_h less or equal to the *sensing radius* R_s. This setting guarantees both coverage and connectivity when the *transmission radius* R_{tx} is such that $R_{tx} \geq \sqrt{3}R_s$.

A sensor which is deployed at the center of a hexagonal tile is called *snapped*. *Hex*(p) is the hexagonal region whose center is covered by the snapped sensor p. All the other sensors lying in *Hex*(p) are called *slaves of* p and compose the set $S(p)$. All the sensors that are neither snapped nor slaves are called *free*. The set composed by the free sensors located in radio proximity to p and by its slaves is denoted by $L(p)$. The set VP(p) of vacant positions detected by sensor p contains the centers of hexagons adjacent to *Hex*(p) that are not yet occupied by any snapped sensor.

A summary of the message types used by protocol P&P during the sensor deployment is presented in Table 1.

5 Snap activity

The purpose of the snap activity is to give start to the creation of hexagonal grid portions and to extend the boundary of existing ones. In order to describe the snap activity, we need to distinguish three cases, according to the role of the involved sensor. Indeed the actions undertaken by the starter sensors, the already snapped sensors and the sensors being snapped, are substantially different.

5.1 Starter sensor behavior

At the beginning, any sensor p may give start to the creation of a tile portion by snapping itself to its present position in an instant of time $t_{start}(p)$ randomly selected over a time interval of length R_{tx}/v, where v is the sensor movement speed. If at the instant $t_{start}(p)$, sensor p has not yet received any message, it elects its position as the center

Table 1 Summary of the P&P messages

Message name	Message fields
IAS	ID, coordinates, starter timestamp
InfoSnapped	ID, coordinates, cardinality
InfoSlave	ID, coordinates, energy level
InfoFree	ID, coordinates
SIP	ID, receiver ID, target position coordinates
AckSIP	ID, receiver ID
ClaimPosition	ID, coordinates, timestamp
PositionTaken	ID, coordinates
InfoStopped	ID, coordinates
IAYS	ID, receiver ID
CardinalityInfo	ID, cardinality
Offer	ID, receiver ID, cardinality, transaction ID
AckOffer	ID, receiver ID
AckInfoArrived	ID, receiver ID
MoveTo	ID, receiver ID, dest. coord., dest. snapped sensor ID, trans. ID
InfoArrived	ID, receiver ID, transaction ID, energy level
Invitation	ID, hop counter, hole coordinates
InvitationAcceptance	ID, coordinate, receiver ID
SlaveSelected	ID, selected slave ID, hop counter
Subst	ID, receiver ID, energy level, destination coordinates
AckSubst	ID, receiver ID
SubstArrival	ID, receiver ID
ProfilePacket	ID, receiver ID, neighborhood information
MoveToSubst	ID, receiver ID, neighborhood information
Retirement	ID, hole coordinates

of the first hexagon and establish the orientation of its tile portion. At this point p executes the snap actions under the role of snapped sensor, as described in the following paragraph.

5.2 Snapped sensor behavior

5.2.1 Neighbor discovery

A snapped sensor p broadcasts an IAS (I Am Snapped) message to perform a neighbor discovery. Such message contains the ID of the sender snapped sensor, its geographic coordinates and the timestamp of the starter action. All sensors located in radio proximity to p (with the exception of those slaves located in different hexagons) reply to its IAS, with a message containing role dependent information: the snapped sensors reply with an InfoSnapped message, while the slave and the free sensors reply with an InfoSlave and an InfoFree message, respectively. All three types of replies contain the ID and geographic coordinates of the replying sensors. In addition, the InfoSnapped message includes also the *virtual cardinality* of the replying snapped sensors, that is the number of slave sensors located inside the hexagon of the sender as if all the movements related to its supervision or to its hexagon were already concluded. The InfoSlave message includes the energy level of the replying slave sensor.

Thanks to the execution of the neighbor discovery phase, a snapped sensor p is informed regarding the presence of vacant positions, i.e. it can determine the composition of VP(p), and is able to build the set $L(p)$. Notice that in order to determine the presence of an adjacent vacant position a snapped sensor only needs to evaluate the presence of snapped sensors in adjacent tiles.

5.2.2 Snap into position

A snapped sensor p selects the sensor in $L(p)$ closest to each vacant position and sends it a SIP (Snap Into Position) message. This message contains the target position of the correspondent snap action, and the ID of the selected sensor. The sensor p then starts a timeout waiting for the AckSIP message from the selected sensor.

When a sensor receives a SIP, and is available to fill the vacant position, it replies with an AckSIP message. This message contains the ID of the sensor that received the SIP, necessary for p to discriminate among the several sensors to which it sent SIP messages. If a sensor receives a SIP when it is not available to fill the vacant position (e.g it has already been contacted by another sensor), it does not reply to the SIP message of p and lets the AckSIP timeout expire. This way p will be capable to select a new sensor to snap in such still vacant position.

After the transmission of the SIP messages and the reception of the related AckSIP, p updates its local information, i.e. the number of free sensors located within its transmission range and its virtual cardinality. This way it keeps into account the departure of some sensors from either its transmission range or its hexagon.

In order to update the information related to the snapped neighbors, p waits for the reception of the corresponding IAS messages, to be sure that position conflicts are solved (see 5.3.3).

Five cases may occur during the snap into position phase. Let p be the sensor that is performing the snap action and let q be the one to which p sent a SIP message for the position x.

1. Sensor p receives both the AckSIP and the IAS message from q. This means that the snap action performed by p was successful, therefore p can update the local information regarding the snapped neighbor q and the hexagon $Hex(q)$.
2. Sensor p receives the AckSIP from q acknowledging its availability to fill position x, but a conflict occurs solved in favor of another sensor r, which reaches position x before sensor q. Hence p receives an AckSIP from q and an IAS from r for the same position x. Thus p can update the local information regarding the snapped neighbor r.
3. Sensor p receives the AckSIP from q acknowledging its availability to fill position x, but a failure occurred and the IAS timeout expires. If p detects the availability of another sensor in $L(p)$ that can be snapped to position x, it retries the snap action. If such sensor is not available, p starts the pull activity.
4. Sensor p does not receive the AckSIP from q, but receives an IAS message for position x from another sensor r, before the expiration of the AckSIP timeout. Sensor p can update the local information regarding the snapped neighbor r.
5. Sensor p does not receive the AckSIP from q nor the IAS from any other sensor within the AckSIP timeout. If p detects the availability of another sensor in $L(p)$ that can be snapped to position x, it retries the snap action. If such sensor is not available, p starts the pull activity.

At the end of the snap activity, a snapped sensor p sends a CardinalityInfo message to its neighborhood. This message contains the ID and the virtual cardinality of p.

5.3 Behavior of the sensors being snapped

5.3.1 Sensor localization

A free sensor q which receives an IAS message, coming from a snapped sensor p, replies with either an InfoFree or an InfoSlave message depending on its position with respect to p. If q is located outside the hexagon of p, it remains in the free state and replies to p with an InfoFree message. If instead q is located inside the hexagon of p, it switches its state to slave and replies to p with an InfoSlave message. In both cases q becomes part of the set $L(p)$, that is the set of sensors that p can snap to its adjacent vacant positions. Notice that if q is a slave, there is only one snapped sensor p such that $q \in L(p)$, thus slaves belonging to already snapped sensors do not reply to the IAS message of p. If instead q is a free sensor, it may belong to several sets $L(\cdot)$, for different snapped sensors located in radio proximity from q itself.

5.3.2 Snap into position

Sensor q, be it free or slave, at a certain time, may receive a SIP message coming from a snapped sensor. Slaves reply only to SIP messages coming from their related snapped sensor, while free sensors may receive SIP messages from any snapped sensor in radio proximity, but only reply to the first SIP message they receive.

After sending the AckSIP reply, sensor q travels towards the snapping destination until it reaches a distance d from it. Distance d is set small enough to guarantee the radio connectivity within the circular disk of radius d and the inclusion of such disk into the hexagonal tile. Therefore $d \leq \sqrt{3} l_h/2$.

At this point sensor q stops and broadcasts a ClaimPosition message containing a timestamp and waits for the expiration of a timeout to evaluate if other sensors are trying to snap in the same position and in case to resolve the related contention. At the timeout expiration, if no conflicts occurred or if a conflict was solved in its favor, q switches its state to snapped, sends a PositionTaken message and proceeds towards the destination. After being successfully snapped, sensor q starts its own snap activity.

5.3.3 Resolution of snap position contention

Three events may occur when one or more sensors are engaged in a conflict with sensor q due to the contention for the same snap position:

1. The sensor q receives a ClaimPosition or a PositionTaken before reaching distance d from the destination,
2. The sensor q receives a ClaimPosition after the arrival at distance d from the destination and before the expiration of the related timeout,
3. The sensor q receives a PositionTaken as a response to its ClaimPosition. This case may happen if q started traveling toward the destination when it was too far to perceive the previous ClaimPosition and PositionTaken messages.

In the first case, q stops moving and sends an InfoStopped message, to advertise its new position to the neighborhood, and starts a timeout. Snapped sensor receiving an InfoStopped message, verifies if the sender is inside its hexagon and in this case replies with a IAYS message (I Am Your Snapped), containing the sender and the receiver ID. If the stopped sensor receives a IAYS reply within the timeout, it sets its status to slave. Otherwise, if the timeout expires, it sets its status to free, not belonging to any hexagon.

In the second case, sensor q compares its timestamp with the one included in the `ClaimPosition` message. The sensor with lower timestamp wins the competition for the destination and proceeds its travel, sending a `PositionTaken` message, while the other sensor waits for the arrival of the `IAS` message of the new snapped sensor to switch its status to slave.

In the third case, sensor q sets its state to slave of the newly snapped sensor. Notice that this timestamp based conflict resolution is designed to avoid redundant replies to `ClaimPosition` messages and does not require global synchronization.

Figure 2 shows a typical conflict resolution scenario, where two sensors r and q receive a `SIP` message for the same position x from two different snapped sensors. Both r and q start traveling towards the destination x. Sensor q reaches distance d from the destination before sensor r, and sends a `ClaimPosition` message, with its timestamp. Sensor r receives such message while traveling, and consequently stops because the contention for position x was won by sensor q. Sensor r sends an `InfoStopped` message to alert its neighborhood of its new position and starts a timeout. In the case depicted in Fig. 2, r stops inside the hexagon centered in position x. For this reason, no snapped sensor replies to the `InfoStopped` message, thus after the timeout expiration, sensor r switches its status to free. After the expiration of the contention timeout,

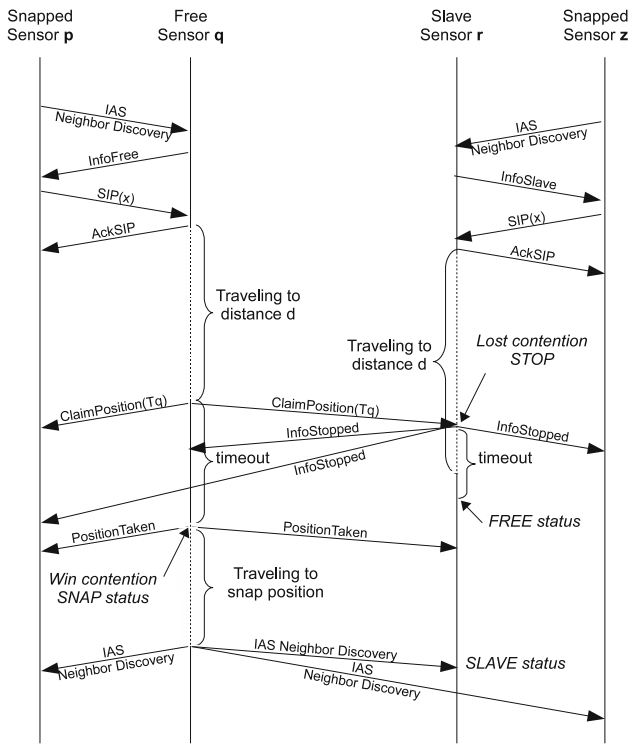

Fig. 2 A typical scenario of snap position conflict

sensor q broadcasts a `PositionTaken` message and switches to the snap status while definitely traveling to position x. When q reaches position x, it starts a neighbor discovery by sending an `IAS` message, in consequence of which, r switches its status to slave.

In order to show an example of the snap activity execution, we refer to Fig. 1. Figure 1(a) and (b) show that, at the beginning of our example of P&P execution, only one sensor assumes the starter role. In Fig. 1(c), this sensor snaps three of its slaves. In a second time, see Fig. 1(g), another node acts as starter and initiates the formation of a second grid portion by snapping three of its slaves as shown in Fig. 1(h).

6 Push activity

The push activity aims at moving slave sensors from high density hexagons to low density hexagons. To describe the push activity we distinguish the behavior of snapped and slave sensors and illustrate the role exchange mechanism introduced to uniform the energy consumption.

6.1 Behavior of snapped sensors

6.1.1 Push proposal

As soon as a snapped sensor p terminates the snap activity, it sends a `CardinalityInfo` message to its neighborhood. Such message contains its ID and its virtual cardinality. Neighbor snapped sensors that receive this message update their information regarding sensor p and evaluate the opportunity to move slave sensors to its hexagon.

Even sensor p evaluates the opportunity to move some of its slaves to adjacent hexagons to uniform the distribution of redundant sensors. To this end, it uses its information regarding the neighbor snapped sensors, collected in the neighbor discovery phase. In particular, the sensor p verifies the following condition, called the *Moving Condition* [3], for each adjacent snapped sensor q:

$$|S(p)| > |S(q)| + 1 \vee \big(|S(p)| = |S(q)| + 1 \wedge \mathrm{ID}(p) > \mathrm{ID}(q)\big)$$

Sensor p looks for neighbor snapped sensors whose hexagons verify the Moving Condition and have minimal cardinality. Among these, it selects the closest, to which it sends an `Offer` message containing its virtual cardinality, defined in Sect. 5.2, and an identifier of the current transaction (transaction ID). If no sensor verifies the Moving Condition with p, sensor p waits for further events.

6.1.2 Push agreement

The snapped sensor q that receives an `Offer` message from p, verifies the validity of the Moving Condition as it

could have more updated information than p, in particular it can have an updated value of its virtual cardinality. This way the responsibility of the slave movement is held by the receiver, thus ensuring that the move only happens when the Moving Condition is actually valid. This is particularly important to guarantee the algorithm termination.

Two cases may occur after the sensor p sends the `Offer` message to q: q accepts the offer it received from p, or q leaves the offer unreplied.

In the first case, q replies to p with an `AckOffer` message, containing only the recipient and sender ID. The sensor q updates its cardinality value, advertising the new value to its snapped neighbors, with a `CardinalityInfo` message. This way q can participate in further operations of distribution of redundant sensors with updated information. It also precludes other snapped sensors from sending unnecessary offers. When q accepts an offer, it starts a timeout identified by the transaction ID received in the `Offer` message. If q does not receive any message regarding the arrival of the new slave, containing the related transaction ID, within the timeout, it decreases its cardinality and advertises this change with a new `CardinalityInfo` message. If, otherwise, sensor q receives an `InfoArrived` message related to the current transaction, it replies with an `AckInfoArrived`, containing its ID and the receiver ID. This way the protocol is robust to possible node failures during the push activity.

The sensor p selects a slave r to be pushed and sends it a `MoveTo` message containing the sender and receiver ID, the position and the ID of the destination snapped node (the sensor q), and the transaction ID. This selection is based on an energy saving criterion. The sensor p selects the slave sensor r that will remain with the highest energy after the completion of the entire movement.

6.2 Behavior of a slave sensor

The slave sensor r selected by the sensor p receives a `MoveTo` message and starts moving towards the hexagon of the sensor q. As soon as the sensor r crosses the boundary of the hexagon of q, it sends an `InfoArrived` message, stops moving and waits for the related `AckInfoArrived` message. The `InfoArrived` message contains the sender and receiver ID, the transaction ID, and the energy level of the sender. If the `AckInfoArrived` message is not received within a timeout, the sensor r assumes that sensor q is not there anymore. As an example, this case can happen if the sensor q has moved to a different grid portion during the merge activity. Thus it tries to snap in the snapping position of q, as if it would have received a SIP message for that position.

6.3 Role exchange

The PUSH & PULL algorithm provides that slaves and snapped sensors may exchange their roles in order to balance the energy consumption over the set of available sensors. Any time a slave r has to make a movement across a hexagon as a consequence of a push action, it sends a role exchange proposal consisting in a `Subst` message to the snapped sensor p of the hexagon it is traversing, and starts a substitution timeout. `Subst` messages contain the ID of sender and receiver, the energy level of the sender and the destination coordinates. The snapped sensor p uses the energy level value of r to decide if a role exchange may be of benefit in balancing the overall energy consumption between the two sensors. In this case, p replies with an `AckSubst` message.

If sensor r receives an `AckSubst` message within the substitution timeout, it travels toward the snap position held by sensor p, while p waits for the arrival of sensor r before starting to travel towards the destination initially targeted by r. The sensor r advertises its arrival to sensor p with a `SubstArrival` message containing the same fields of the `AckSubst` message. The sensor p replies to r with a `ProfilePacket` message containing the sender and receiver ID, and the neighborhood information. This is necessary to enable a complete role exchange and starts traveling towards the destination.

If the sensor r does not receive an `AckSubst` message within the substitution timeout, it continues its travel towards the destination.

Slave and snapped sensor substitutions may also occur at the beginning of the slave travel. In this case the substitution is started by the snapped sensor itself which already has all the available information to evaluate the opportunity to perform the role exchange. Under these circumstances, the snapped sensor p sends a `MoveToSubst` message containing the profile information necessary to perform the substitution. As soon as sensor r arrives in proximity to the snap position held by p, it sends the `SubstArrival` message described before, after which p starts traveling towards the destination.

6.4 An example

Figure 3 depicts a typical scenario of the push activity. The snapped sensor q broadcasts its virtual cardinality with a `CardinalityInfo` message. The snapped sensors p and z receive this message and verify the Moving Condition with the updated information received from q. As both p and z satisfy the condition, they send an `Offer` message to q. Notice that the `Offer` message always contains an updated value of the virtual cardinality of the sender. Since each node can offer at most one sensor at a time the virtual cardinality does not change until the offer timeout expires,

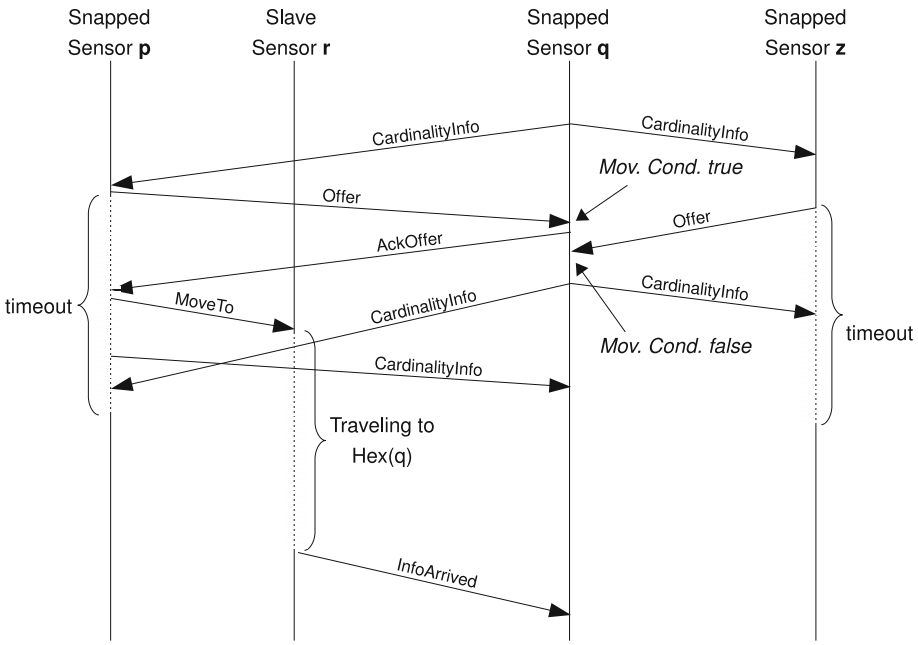

Fig. 3 A typical scenario of the push activity

or the receiver replies with an `AckOffer` message. Sensor q receives the `Offer` message from p before the one sent from sensor z. It verifies the validity of the Moving Condition with the updated virtual cardinality of p, received in the `Offer` message. As the Moving Condition is still satisfied, q replies with an `AckOffer` message, incrementing its virtual cardinality and broadcasting a `CardinalityInfo` message.

When node q receives the `Offer` message from z it verifies the Moving Condition again. Note that z sent this message on the basis of an old value of the virtual cardinality of q. Thus q finds that, as a consequence of the transaction just concluded with sensor p, the Moving Condition is unsatisfied with respect to sensor z, and consequently it does not reply to the offerer. Sensor z waits until the expiration of the offer timeout, after which it is able to be engaged in other push actions.

Sensor p receives an `AckOffer` message from q, thus it selects r within its slaves, and send it a `MoveTo` message. Sensor r moves towards the hexagon of q, and sends an `InfoArrived` message as soon as it arrives. Sensor p sends a `CardinalityInfo` message containing the decreased value of its virtual cardinality.

An example of the push activity execution can be found in Fig. 1. Figure 1(d) shows that the snapped node 1 has some slave sensors in its hexagon, and therefore starts the push activity towards its three adjacent hexagons. In Fig. 1(e) the snapped nodes 6 and 9 perform the snap activity. Notice that the snapped sensor 0 does not perform any snap action as it does not have any hole around its hexagon. It also does not execute any push action as the Moving Condition is not satisfied. In the Fig. 1(e) and (f), the snapped node 1 continues its push activity while the node 9 performs a snap of its slave. Figure 1(g)–(k) show other examples of the push activity execution.

7 Pull activity

The pull activity is intended to attract non snapped sensors toward coverage holes. We underline that the pull activity described in this section is slightly different from the one introduced in [3]. In [3] the pull action is based on the gradual and temporary modification of the IDs advertised by the sensors, so as to temporarily alter the flow of push actions. Nevertheless, we experimentally noticed that such an activity is very slow and can be performed more efficiently by means of an invitation mechanism, similar to the one introduced by [15].

In such an invitation mechanism, the sensors may play different roles. A first role is played by the sensor detecting a coverage hole in a neighbor location. This sensor starts the pull activity by sending an invitation that will attract slave sensors from nearby regions to fill the hole. The sensors having this role are hereafter called *Inviters*.

The second role is performed by the slave sensors which receive the aforementioned invitation messages. These sensors, hereby called *Invited*, reply to the inviter to communicate their availability to move and fill the hole.

If available slave sensors are in the radio proximity of the inviter, the invitation mechanism has an immediate effect. By contrast, when there are no available slave sensors, the

inviter proceeds inviting new slaves by means of a limited distance broadcast message at larger and larger distances.

Therefore a third role is performed by snapped sensors which receive the broadcast invitation messages and act as forwarder of the invitation, in order to reach hexagons with redundant slaves that can fill the holes. These sensors will be named *Forwarders*.

Our modified pull mechanism also has the beneficial effect of avoiding the case of too many invited sensors moving to the same destination which was possible with the pull technique proposed in [3].

In the following we summarize the invitation mechanism.

1. A snapped sensor detecting a hole sends an invitation to advertise the presence of a vacant position in its neighborhood. This message traverses the network according to a limited distance broadcast, whose extent is increased in the time slots of successive attempts. The inviter does not extend the invitation any further as soon as it receives an acknowledgment from at least one invited slave sensor.
2. A slave sensor collects the invitations received in a given time interval. If it receives multiple invitations, it stores them in a priority based queue, where the priority is the distance to the destination. The lower the distance, the higher the priority.
   An invited slave acknowledges only the highest priority invitation, so as to notify the inviter that it is available to fill the vacant position.
3. The inviter that receives such an acknowledgment from one or more slaves, selects the one that is closer to the hole, and notifies the others that an agreement to cover the hole was already made with another invited slave.
4. The selected invited slave that receives an acknowledgment from the inviter starts moving to the destination. On the contrary, the other non selected invited slaves listening the acknowledgment, process the next element in their priority queue.

Notice that it may occur that the timeout according to which an inviter snapped sensor s decides to extend the invitation to the next hop expires before receiving an invitation acceptance from the closest available slave z. It can also happen that the closest slave sensor z is not immediately available to fulfill a pull request because it is temporarily involved in another pull negotiation with another snapped sensor q. In both the described situations, such a slave sensor z may still become available for the pull action performed by s if the previous negotiation with q fails. Nevertheless this may occur after the expiration of the timeout of the snapped sensor s, which in the meantime could have already extended the invitation to the next hop. In all these cases, if the vacant position has not been covered yet, the available slave sensor z will be able to respond to s and guarantee the coverage of the vacant position.

7.1 Behavior of the inviter sensors

A snapped sensor p, located in proximity of some vacant positions (i.e. $\mathrm{VP}(p) \neq \emptyset$), terminates the snap activity when no more sensors are available in $L(p)$. Before starting the pull activity, the sensor p verifies if there is the possibility to receive slave sensors from its snapped neighbors performing the push activity. To this purpose, p checks the validity of the Moving Condition with respect to all of its snapped neighbors.

If the Moving Condition is not verified for any of its snapped neighbors, that is p can not receive any sensor by means of the push activity, it starts the pull activity. To this purpose p broadcasts an `Invitation` message containing its ID, a hop counter h, and the vacant position coordinates.

The hop counter h represents the forwarding horizon of the `Invitation` message. Initially h is set to zero, thus the snapped sensors receiving an `Invitation` do not forward this message. After a given timeout, if p does not receive any acknowledgment, it increases the hop counter and sends a new `Invitation` message.

On the contrary, before the expiration of the timeout, the inviter p may receive some `InvitationAcceptance` messages from one or more available slaves. This message contains the sender and receiver ID and the coordinates of the sender. After the expiration of the timeout, the inviter p selects the sensor s^*, among the sensors that replied to the invitation. In particular, the selected sensor s^* is the closest to the vacant position. The sensor p sends a message `SlaveSelected` (multicast message at maximum h-hop) to all the sensors which accepted the invitation. The message `SlaveSelected` contains the sender ID, the ID of s^* and the destination coordinates.

Figure 4 illustrates the pull action performed by sensor p as described above.

7.2 Behavior of the invited sensors

When a slave sensor s receives an `Invitation` message, and has not already committed to fill a different hole, it inserts such invitation message in a priority queue, containing all the invitation received in a given time interval, according to a priority based on the distance from the invited slave to the destination. This time interval starts as soon as a slave sensor receives the first invitation. At the end of this time interval, the invited sensors processes the elements in the priority queue one by one. If another slave sensor has already accepted the invitation to fill the hole being considered (this event corresponds to s receiving a `SlaveSelected` message containing the same hole

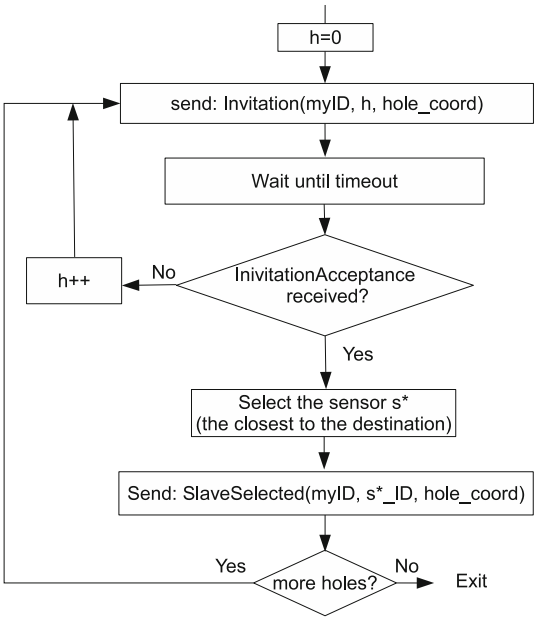

Fig. 4 Behavior of a sensor detecting a hole

coordinates but another invited ID), *s* proceeds by considering the next element in its priority queue.

The sensor *s* sends an `InvitationAcceptance` to the inviter related to the highest priority element in the queue, for which no previous agreement has been notified. As soon as *s* receives a `SlaveSelected` message regarding the hole considered in the `InvitationAcceptance` messages, it verifies to be the selected slave and, in this case, it starts moving towards the destination. Otherwise, *s* drops the record on the considered hole, and processes the next element in the queue.

7.3 Behavior of forwarder snapped sensors

When a snapped sensor *p* receives an `Invitation` message it participates in the pull activity by forwarding this message when necessary. The snapped sensor *p* forwards the invitation only if $h > 0$ and decreases by one the hop counter associated to the forwarded invitation.

In order to reduce the number of broadcast messages each snapped sensor forwards `Invitation` messages only if it has not already broadcast a message for the same hole with a greater or equal value of *h*.

Notice that, whether the transmission radius $R_{tx} \gg R_s$, the hop counter *h* refers to the communication hops. In this case, the forwarder sensors are only the snapped sensors located at the boundary of the communication range.

7.4 An example

Figure 5 shows a typical workflow of the pull activity in the scenario depicted in the sample grid shown in the lower part of the figure. The snapped sensor *p* detects a coverage hole in an adjacent position. Since *p* has no slaves in its hexagon and the Moving Condition with respect to its neighbors is unsatisfied, it starts the pull activity broadcasting an `Invitation` message with null hop counter. Since this invitation does not reach any slave sensor, at the expiration of the timeout, sensor *p* broadcasts another `Invitation` message increasing the hop counter. Sensor *q* evaluates the hop counter of the `Invitation` message it received from *p* and forwards such a message with decreased hop counter. Once again the timeout set by *p* expires because there is no available slave in proximity, thus the procedure is repeated until the `Invitation` message, reaches the slave sensor *s* which starts a timeout. The sensor *s*, after the expiration of the timeout, during which it has not received any other `Invitation` message, sends an `InvitationAcceptance` message to the inviter sensor *p*. Since the sensor *p* has not received any other `InvitationAcceptance` message during its timeout, it sends a `SlaveSelected` message to the sensor *s*. Upon receiving such a message, the sensor *s* starts traveling towards the coverage hole and snaps itself.

Figure 1 shows the interleaved execution of the pull activity with the other algorithm activities. In particular, Fig. 1(f) shows the pull activity started by node 7, which does not have any slave to perform the snap activity. This activity is started by node 7 in order to fill the coverage hole that it detects in its adjacent snapping positions. In agreement with the pull activity, the closest slave sensor is selected from the hexagon of the snapped sensor 6, to move towards the hole. Other examples of the pull activity execution are depicted in Fig. 1(g) and (i) where one can notice that the forwarding horizon has been extended in order to attract the available slave sensors. Figure 1(f) shows an example of the concurrent execution of the the snap, push and pull activities highlighted by three different kind of arrows.

8 Merge activity

The fact that many sensors act as starters implies the generation of several tiling portions with different orientations. The aim of the algorithm is to cover the AoI with a unique regular tiling thus minimizing overlaps of the sensing disks and enabling a complete and uniform coverage. Hence, the P&P protocol provides a merge mechanism to be executed whenever a sensor *p* receives a neighbor discovery message (`IAS`) from a snapped sensor *q* belonging to another tiling portion.

In this case, sensor *p* chooses to join the oldest grid portion. In order to discriminate which is the oldest grid, the sensor *p* evaluates the timestamp of the starter action, attached to any `IAS` message.

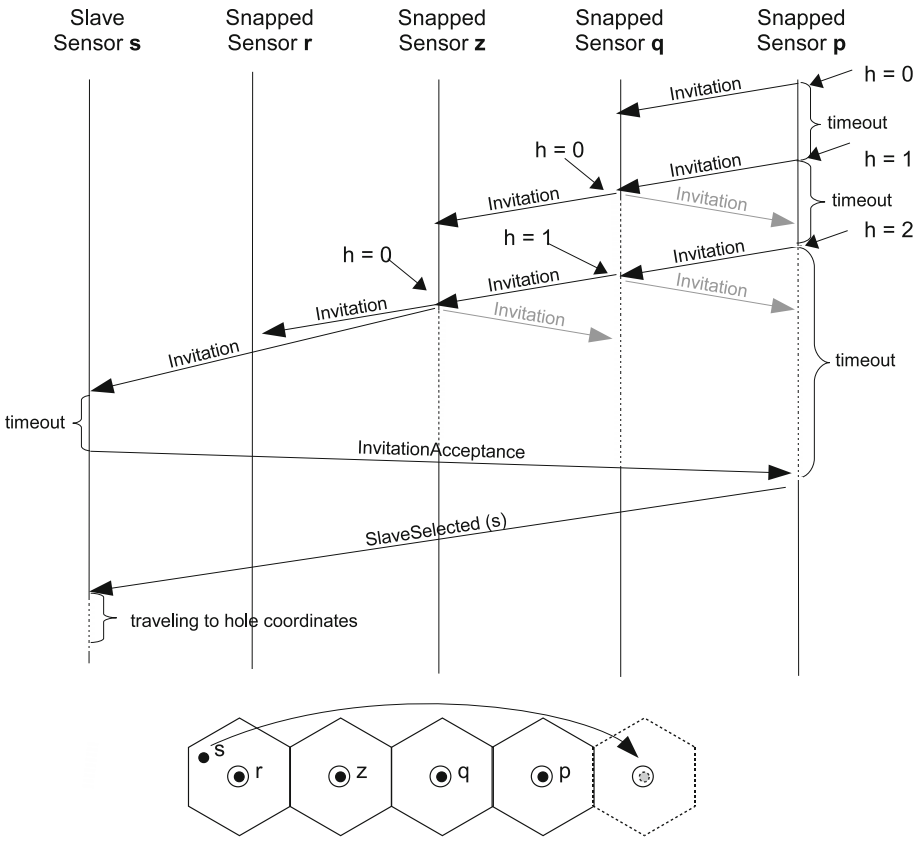

Fig. 5 A typical scenario of the pull activity

Notice that the detection of the sole neighbor discovery messages is sufficient to ignite the tiling merge activity because such messages are sent after any tiling expansion and, if two tiling portions come in radio proximity to each other, at least one of them is increasing its extension.

In order to explain the grid merge activity, we refer again to Fig. 1. Figure 1(h) shows the presence of two grid portions in radio proximity with each other. As a consequence of this reciprocal detection, the two grid portions start the tiling merge activity. In particular, the sensor 12 initially belonging to the right grid, is snapped in a vacant position of the left grid by node 16, as shown in Fig. 1(i). In Fig. 1(l) the tiling merge activity is concluded and a unique grid is built.

In the following we give the details on the protocol implementation of the grid merge activity. We call G_{old} and G_{new} the tiling portions with lower and higher timestamp, respectively. We distinguish three possible cases, depending on the role of the sensor p with respect to the two grids: (1) p belongs to G_{old}, (2) p belongs to G_{new} or (3) p is a free sensor.

1. The sensor p belongs to G_{new} and receives an `IAS` message from q belonging to G_{old}. If sensor p is a slave, it switches its state to free or to slave of the sensor q depending on their mutual distance and acquires the starter timestamp of the sensor q. Sensor p proactively communicates its new state to its neighborhood by sending either an `InfoFree` or an `InfoSlave` message. From now on p honors only messages from G_{old} and ignores those from G_{new}.
This proactive communication of the new state of p is needed to advertise the presence of G_{new} when there is no message activity within G_{new} that is perceivable by the sensors in G_{old}. In this way, the snapped sensor which p belonged to can properly update its slave set.
If p is instead a snapped sensor, it can not immediately switch to its new state because of its leading role inside G_{new} (e.g. it leads the slave sensors in $S(p)$ and performs push and pull activities). Hence p temporarily assumes a hybrid role: it advertises itself as free/slave to the nodes of G_{old} with an `InfoFree`/`InfoSlave` message and, at the same time, keeps on behaving as snapped node in G_{new} until it receives a movement command (`SIP` or `MoveTo` message) coming from G_{old}.
When p receives a `SIP` or a `MoveTo` command, p moves to a new position electing one of its slave in G_{new} as a substitute with a `MoveToSubst` message. The selected slave should reply with a `SubstArrival` upon arrival to the snap position, within a given timeout. If p receives a `SubstArrival` on time, it ceases its snapped role in G_{new} and honors the

commands issued by the snapped node in G_{old}. Otherwise, if this timeout expires before the reception of the `SubstArrival` message, p selects a new slave to snap. The process goes on until no more slaves are available. In this case p ceases its snapped role inside G_{new} advertising its departure to its neighbors in G_{new}, broadcasting a `Retirement` message. Upon the reception of a `Retirement` message the snapped neighbors that were located in positions adjacent to the one that p just freed, keep into account the new vacant position starting new snap activities.
Notice that a sensor q that is neighbor of the retiring sensor p could lie in a position that is out of the AoI (the hexagonal tile has an intersection with the AoI, but the center of the hexagon is outside of it). In this case, it may occur that, due to the merge activity, the sensor q remains disconnected from the two grids G_{new} and G_{old}. To avoid such a situation, when q receives a `Retirement` message from any of its adjacent snapped neighbors, it repositions itself inside the AoI, where it will either be involved in the activity of one grid or will become free and behave accordingly.

2. The sensor p belongs to G_{old} and receives an `IAS` message from q belonging to G_{new}: if p is a slave it ignores all messages from G_{new}. If p is snapped, it performs a neighbor discovery sending an `IAS` message, ignores all messages coming from G_{new}, apart from the neighbor discovery replies, and honors only messages from G_{old}. Observe that the neighbor discovery is necessary to ignite the merge mechanism and allows each snapped sensor in G_{old} to collect complete information on nearby sensors that previously belonged to G_{new}.
3. The sensor p is free: sensor p honors only messages from G_{old} and ignores those from G_{new}.

9 Experimental results

In this section we study the performance of the P&P protocol. To this end we consider different initial scenarios and compare P&P to a previously proposed virtual force based algorithm, through simulations. We consider the virtual force approach for our analysis because it is one of the most popular method proposed for the mobile sensor deployment. In order to perform the comparison, we developed a simulator based on the wireless module of the Opnet software [23]. Before introducing the simulation results, in the following we briefly describe the virtual force based algorithm named Parallel and Distributed Network Dynamics (PDND) [11], used for the comparisons. We choose this algorithm as its force model allows to obtain very good performance and, differently from several other solutions in the literature, it has a guaranteed termination.

9.1 Parallel and distributed network dynamics (PDND)

The Parallel and Distributed Network Dynamics (PDND) algorithm [11] is a virtual force based approach according to which the force exerted by the sensor s_i on the sensor s_j is modeled as a piecewise linear function. It is repulsive when the distance between s_i and s_j is lower than an arbitrarily tuned parameter r^*, it is attractive when the distance is larger, and it vanishes at another arbitrarily set distance. In order to ensure the convergence of PDND, the formulation of this force must respect the condition of Lipschitz continuity. To obtain this feature, the single sensor movement is limited by a maximum moving distance threshold which guarantees that the potential energy is always decreasing.

The PDND algorithm is a round based algorithm. At each round the sensors initially exchange their position information, calculate the resulting force exerted by the sensors in the neighborhood and then move accordingly. The algorithm is proved to achieve a final stable configuration in which all sensors stop moving provided that a positive minimum moving distance threshold is set. Note that, although this algorithm is one of the best proposals based on virtual forces, it presents the common drawbacks of the proposals designed following this approach. In particular, several threshold values must be manually tuned in order to make the algorithm work at its best.

9.2 Simulation results

In this Section we describe the performance comparisons between P&P and PDND. The parameter setting used in the experimental activity is as follows: the transmission radius R_{tx} is set to 11 m, the sensing radius R_s is set to 5 m, the sensor speed is 1 m/sec, and the AoI is a square with size 80 m $\times$ 80 m. For the PDND algorithm, the round length is set to 1 sec, the threshold r^* to $2R_s$ while the minimum moving distance is set to 0.1 m, as in [11].

Before giving a quantitative evaluation of the protocol performance, we show some examples of the execution of P&P and PDND. In Fig. 6(a)–(d) we show an example of the P&P protocol execution starting from a random initial distribution of 150 sensors over the AoI, whereas in Fig. 7(a)–(d) we show the execution of PDND starting from the same initial configuration. In Figs. 8 and 9 we show some snapshots of the execution of P&P and PDND, respectively, starting from a distribution where 150 sensors are densely deployed at the center of the AoI.

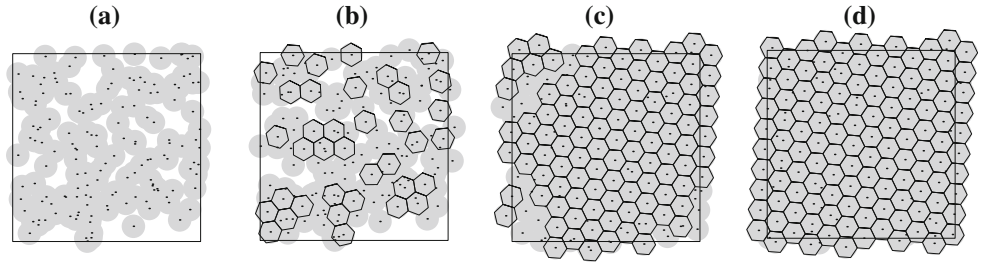

Fig. 6 Deployment with random distribution under P&P

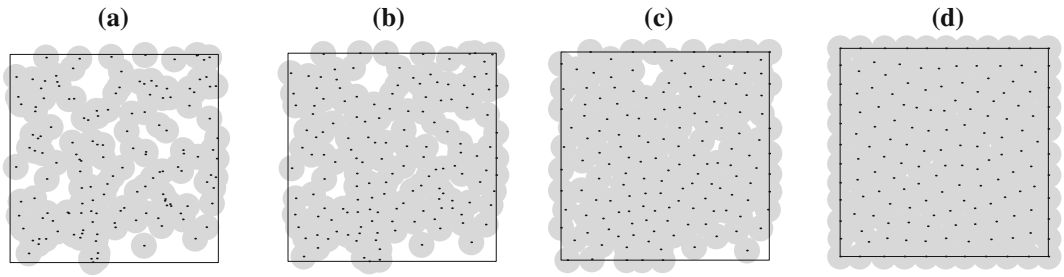

Fig. 7 Deployment with random distribution under PDND

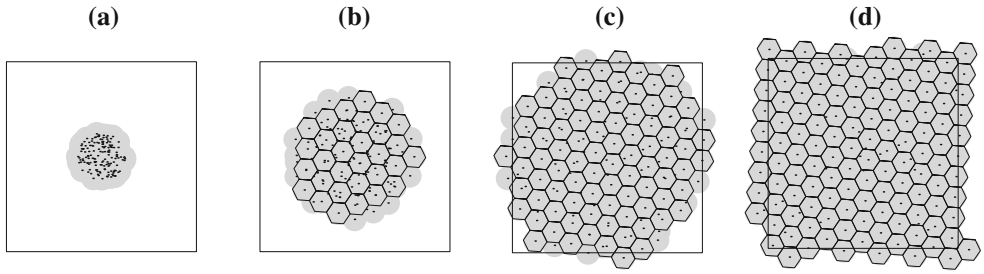

Fig. 8 Deployment with central distribution under P&P

The protocol P&P is able to achieve a complete coverage of the AoI in both scenarios. It is to notice that, in the case of randomly deployed sensors, several grid portions are created at the beginning of the algorithm execution (Fig. 6(b)), due to the random election of the starters. As the grids portions keep growing, after a certain time, they arrive in radio proximity to each other, and the merge activity starts. The older portions, which are likely composed by a higher number of nodes (Fig. 6(c)), include the newer ones step by step. At the end of the algorithm execution, only one tiling portions has remained which entirely covers the AoI (Fig. 6(d)). On the contrary, when the initial deployment is dense as the one shown in Fig. 8(a), every node can communicate with all the other nodes, thus only one grid is created, as illustrated in Fig. 8(b), (c).

The PDND algorithm is able to cover the AoI in the two considered scenarios as well. Nevertheless, as we will show in the following, it requires much longer time to achieve the final deployment and consumes a higher amount of energy with respect to P&P.

Furthermore, P&P is able to achieve a complete coverage in complex shaped AoIs. In Fig. 10 we show a synthetic representation of how the sensor deployment evolves under P&P when 150 sensors are sent from a high density region in an AoI composed by two squared rooms connected by a narrows.

In order to compare the performance of our protocol and PDND, we run two sets of experiments, starting from two different initial sensor deployments, by varying the number of deployed sensors. In the experiments we increase the number of deployed sensors ranging from 150 to 550. We do not show the graph of the coverage achieved by the two algorithms as, in the considered interval of available sensors, both of them always reach a complete coverage.

In the first set of experiments we considered the random initial deployment depicted in Fig. 8(a). Figure 11 refers to P&P and represents the number of conflicting snap actions, averaged over the number of snap positions, and the number of push conflicts, averaged over the number of slave sensors involved in a push action. A snap conflict

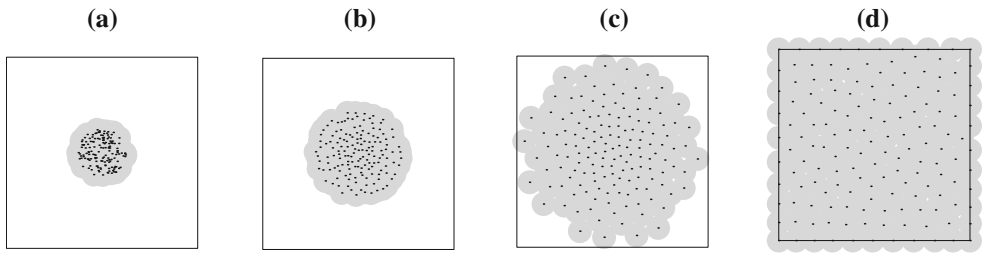

Fig. 9 Deployment with central distribution under PDND

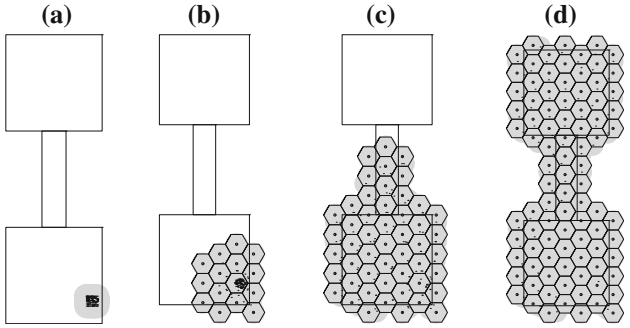

Fig. 10 Coverage of an irregular AoI

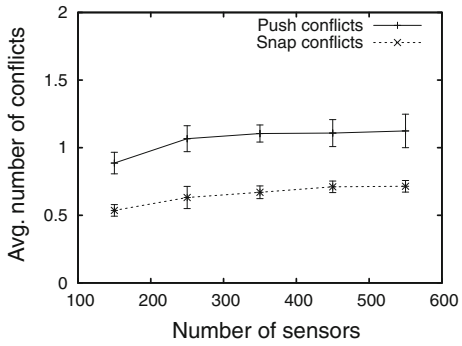

Fig. 11 Snap and Push conflicts

occurs whenever the same snap position is contended by two or more sensors being snapped, whereas a push conflict happens when a push offer made by one sensor becomes obsolete because of the push actions performed by other sensors.

The asynchronous behavior of P&P guarantees the resolution of the few snap/push conflicts that arise as a consequence of its distributed execution. Although the average number of snap conflicts grows with the number of available sensors, it remains significantly smaller than 1, meaning that, in the considered scenarios, no more than one conflict happens per snap position. Similarly, when the number of sensors is larger than the minimum to guarantee the coverage completeness, the average number of push conflicts per slave sensor becomes almost stable at about 1.2 push conflicts per slave sensor.

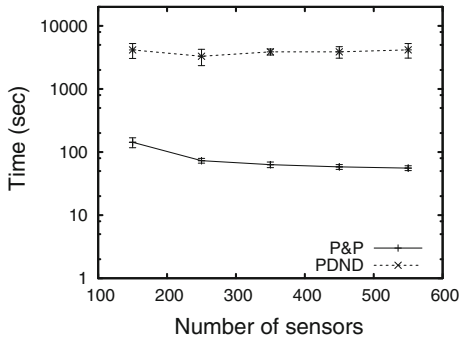

Fig. 12 Termination and coverage time

Figure 12 shows the termination time of P&P and PDND, i.e. the time at which all sensors stop moving since a stable configuration is reached. According to the protocol P&P this situation occurs when all the activities are terminated. In particular, the snap and pull activity terminate as soon as all the snapping position are occupied. The push activity terminates when the moving condition is not satisfied among all the adjacent snap sensors. Finally, the merge activity ends as soon as all the created grid portions are absorbed by the oldest grid. We have formally proved in [3] that such a final stable configuration is always reached by the PUSH&PULL algorithm.

Even the PDND algorithm is proved to converge to a stable configuration, in which the forces acting on all the sensors are balanced. Nevertheless, as Fig. 12 points out, PDND requires a time that is one order of magnitude larger than the one needed by P&P to terminate. The P&P protocol completes the deployment in such a moderate time because it is able to coordinate distributed decisions and solve local conflicts. On the contrary, the PDND algorithm requires a so long time to terminate because the sensors are allowed to traverse only very short distances at each round. This limitation on the traversed distance is necessary to guarantee that each movement contributes to a decrease in the overall potential energy of the system.

Both the algorithms require a longer time to terminate when 150 sensors are deployed. This is due to the fact that this value is close to the minimum number of sensors required to entirely cover the AoI. The results shown in

Fig. 12 highlight the good scalability of P&P, in fact the time needed to terminate slightly decreases as the number of available sensors increases.

The next figures detail the performance evaluation of the two protocols in terms of energy consumption. The protocol activities having the major impact on the energy consumption are: movements, starting/braking actions and message exchanges.

Figure 13 shows the average moving distance per sensor. It is worth noting that under both algorithms the traversed distance decreases when the number of available sensors increases. This is due to the initial random distribution which ensures an even density over the AoI that helps to achieve the final configuration with few movements. However, P&P let sensors traverse shorter distances with respect to PDND.

An important contribution to the overall energy consumption is due to the starting/braking actions performed by the moving sensors [18]. Figure 14 shows that the PDND algorithm performs at least two orders of magnitude of starting/braking actions more than P&P. Indeed, according to PDND, in order to guarantee that the potential energy of the system decreases with time, sensors are allowed to move only for a very short distance at each round, resulting in a very high number of moving actions. On the contrary, P&P makes precise movements, drastically reducing the number of starting/braking actions. Moreover, as Fig. 14 shows, the number of moving actions decreases when the number of sensors increases, thus evidencing a good scalability of the proposed approach.

The last term of the overall energy consumption is the number of message exchanges, shown in Fig. 15. At each round of the PDND algorithm the sensors have to exchange their position information. Since the algorithm requires a very high number of rounds to terminate, a very high number of messages is exchanged. On the contrary, under P&P, the number of exchanged messages remains almost stable even when the number of sensors increases significantly, as shown in Fig. 15. It is worth noting that the energy consumption related to transmitting and receiving messages is affected by the sensor density too.

Indeed, the higher the sensor density, the higher the contribution to the overall energy consumption due to message receiving actions. This trend is made evident in Fig. 16, where we

analyze the overall energy consumption. In this figure we utilize a unified energy consumption metric obtained as the sum of the contributions given by movements, starting/braking actions and communications. The energy spent by

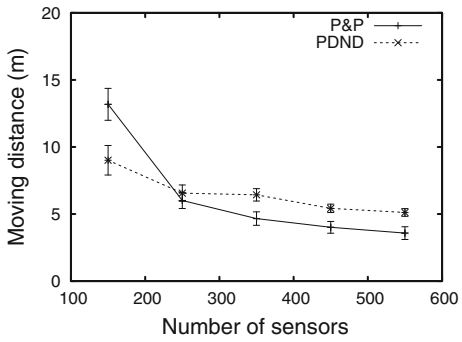

Fig. 13 Traversed distance

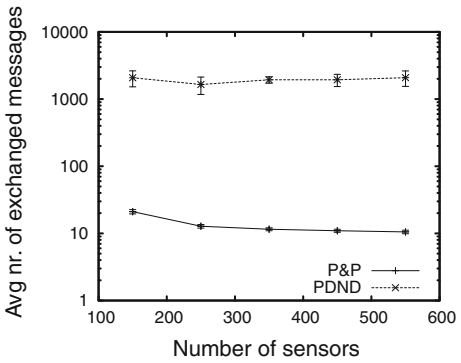

Fig. 15 Exchanged messages

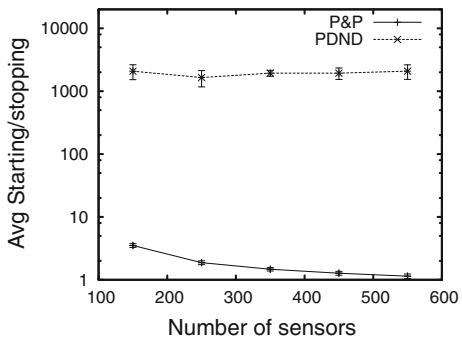

Fig. 14 Starting/braking

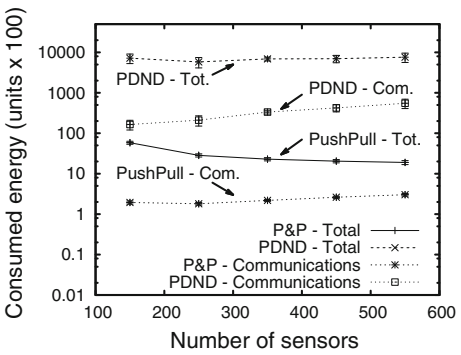

Fig. 16 Consumed energy

sensors for communications and movements is expressed in energy units. The reception of one message corresponds to one energy unit, a single transmission costs the same as 1.125 receptions [1], a 1 m movement costs as much as 300 transmissions [18] and a starting/braking action costs the same as 1 meter movement [18].

As expected by the above comparisons between P&P and PDND of the energy consuming activities, P&P shows an overall energy consumption which is two orders of magnitude less than the one of PDND. Notice that the overall energy consumption of P&P is even lower than the one required by PDND only for the communications.

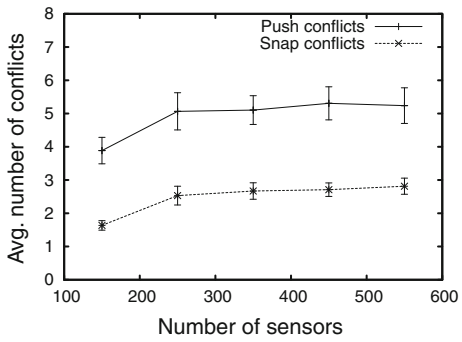

Fig. 17 Snap and Push conflicts

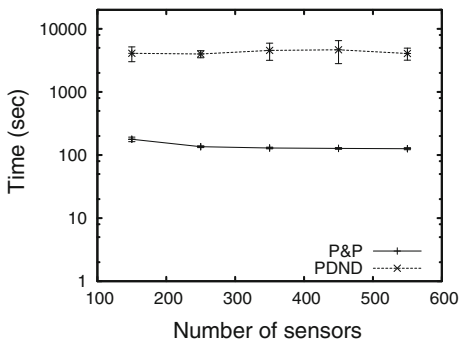

Fig. 18 Termination and coverage time

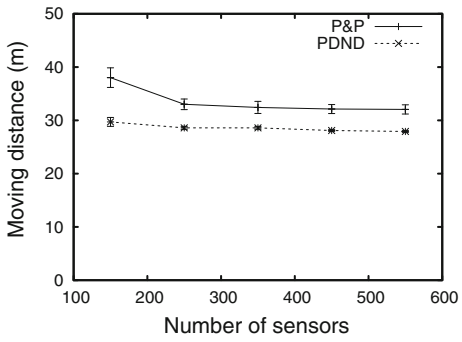

Fig. 19 Traversed distance

In the second set of experiments we compare the performance of the two algorithms starting from the initial configuration shown in Fig. 8 where the sensors are densely deployed at the center of the AoI. The results are shown in the Figs. 17, 18, 19, 20, 21 and 22.

It is to notice that, with respect to the previous experimental setting, the P&P algorithm shows a higher number of conflicting actions (Fig. 17). Indeed, by starting from a denser initial deployment, more sensors compete for the same snapping positions and, similarly, more sensors are pushed towards the same hexagons. For this reason, the termination time and the number of exchanged messages increase with respect to the case of random initial

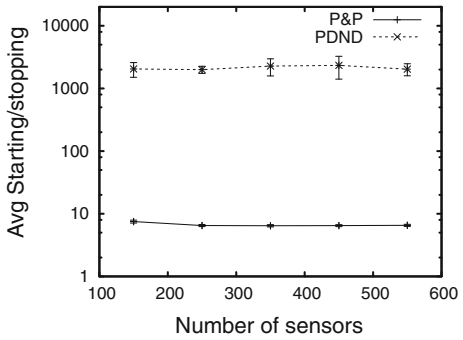

Fig. 20 Starting/braking

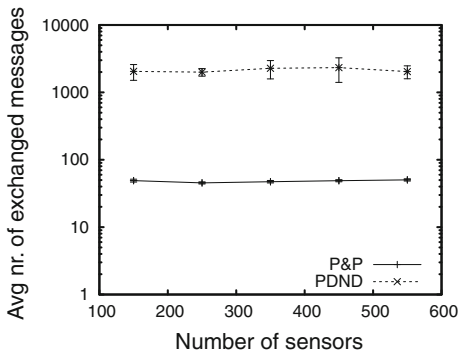

Fig. 21 Exchanged messages

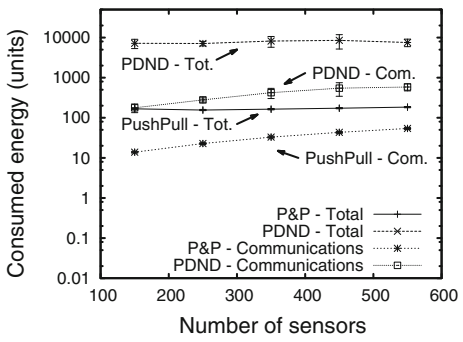

Fig. 22 Consumed energy

deployment. This is due to the fact that, in order to achieve a uniform deployment, more sensors move from their initial positions, traverse longer distances, and more coordination messages are needed. Similarly to the previous experimental set, P&P outperforms PDND. In particular, the PDND algorithm terminates in a much longer time with respect to P&P and exhibits an overall energy consumption which is two orders of magnitude higher than the one of P&P.

10 Conclusions

In this paper we introduce P&P, a communication protocol that permits a correct and efficient coordination of sensor movements in agreement with the PUSH & PULL algorithm. Unlike previous works which introduce deployment algorithms without formalizing the related protocol, we address the realistic applicability of this approach. To this end we deeply investigate the possible conflicts that may arise when asynchronous local decisions are to be coordinated, and propose protocol solutions.

Simulation results show the performance of our protocol under a range of operative settings, including conflict situations and irregularly shaped target areas. These results evidence the protocol capabilities to fulfill the algorithm requirements, in particular termination, completeness and stability of the final coverage. Furthermore, the performance comparison between the P&P protocol and the PDND algorithm, based on the virtual force approach, shows that our solution is able to achieve the final deployment with a time one order of magnitude shorter than the one needed by PDND, while consuming at least two order of magnitude less energy.

References

1. Anastasi, G., Conti, M., Falchi, A., Gregori, E., & Passarella, A. (2004). Performance measurements of mote sensor networks. In *Proceedings of the ACM international conference on modeling, analysis and simulation of wireless and mobile systems (MSWiM)*.
2. Bartolini, N., Calamoneri, T., Fusco, E.-G., Massini, A., & Silvestri, S. (2008). Snap & spread: A self-deployment algorithm for mobile sensor networks. In *Proceedings of the IEEE international conference on distributed computing in sensor systems (DCOSS)*.
3. Bartolini, N., Calamoneri, T., Fusco, E.-G., Massini, A., & Silvestri, S. (2010). Push & pull: Autonomous deployment of mobile sensors for a complete coverage. *ACM/Springer Wireless Networks, 16*(3), 607–625.
4. Brass, P. (2007). Bounds on coverage and target detection capabilities for models of networks of mobile sensors. *ACM Transactions on Sensor Networks, 3*(2), 1–19.
5. Chen, J., Li, S., & Sun, Y. (2007). Novel deployment schemes for mobile sensor networks. *Sensors, 7*(11), 2907–2919.
6. Garetto, M., Gribaudo, M., Chiasserini, C.-F., & Leonardi, E. (2007). A distributed sensor relocation scheme for environmental control. In *Proceedings of the IEEE international conference on mobile Ad-hoc and sensor systems (MASS)*.
7. Garetto, M., Gribaudo, M., Chiasserini, C.-F., & Leonardi, E. (2008). Sensor deployment and relocation: A unified scheme. *Springer Journal of Computer Science and Technology, 23*(3), 400–412.
8. Heo, N., & Varshney, P. (2005). Energy-efficient deployment of intelligent mobile sensor networks. *IEEE Transactions on Systems Man and Cybernetics, Part A, 35*(1), 78–92.
9. Howard, A., Mataric, M. J., & Sukhatme, G.-S. (2002). Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the international symposium on distributed autonomous robotics systems (DARS)*.
10. Kerr, W., Spears, D., Spears, W., & Thayer, D. (2004). Two formal fluid models for multi-agent sweeping and obstacle avoidance. In *Proceedings of the international conference on autonomous agents and multiagent systems (AAMAS)*.
11. Ma, K., Zhang, Y., & Trappe, W. (2008). Managing the mobility of a mobile sensor network using network dynamics. *IEEE Transactions on Parallel and Distributed Systems, 19*(1), 106–120.
12. Ma, M., & Yang, Y. (2007). Adaptive triangular deployment algorithm for unattended mobile sensor networks. *IEEE Transactions on Computers, 56*, 946–958.
13. Pac, M.-R., Erkmen, A.-M., & Erkmen, I. (2006). Scalable self-deployment of mobile sensor networks; a fluid dynamics approach. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)*.
14. Poduri, S., & Sukhatme, G.-S. (2004). Constrained coverage for mobile sensor networks. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (Vol. 1).
15. Tan, G., Jarvis, S.-A., & Kermarrec, A.-M. (2009). Connectivity-guaranteed and obstacle-adaptive deployment schemes for mobile sensor networks. *IEEE Transactions on Mobile Computing, 8*(6), 836–848.
16. Teng, J., Bolbrock, T., Cao, G., & La Porta, T. (2007). Sensor relocation with mobile sensors: Design, implementation, and evaluation. In *Proceedings of the IEEE international conference on mobile adhoc and sensor systems (MASS)*.
17. Wang, G., Cao, G., & La Porta, T. (2004). Proxy-based sensor deployment for mobile sensor networks. In *Proceedings of the IEEE international conference on mobile adhoc and sensor systems (MASS)*.
18. Wang, G., Cao, G., & La Porta, T. (2006). Movement-assisted sensor deployment. *IEEE Transactions on Mobile Computing, 5*(6), 640–652.
19. Wang, Y.-C., Hu, C.-C., & Tseng, Y.-C. (2008). Efficient placement and dispatch of sensors in a wireless sensor network. *IEEE Transactions on Mobile Computing, 7*(2), 262–274.
20. Wang, Y.-C., & Tseng, Y.-C. (2008). Distributed deployment schemes for mobile wireless sensor networks to ensure multilevel coverage. *IEEE Transactions on Parallel and Distributed Systems, 19*(9), 1280–1294.
21. Yoon, S., Soysal, O., Demirbas, M., & Qiao, C. (2008). Coordinated locomotion of mobile sensor networks. In *Proceedings of the IEEE international conference on sensor, mesh and ad hoc communications and networks (SECON)*.
22. Zou, Y., & Chakrabarty, K. (2003). Sensor deployment and target localization based on virtual forces. In *Proceedings of the IEEE international conference on computer communications (INFOCOM)*.
23. Opnet technologies inc. http://www.opnet.com.

Author Biographies

Novella Bartolini graduated with honors in 1997 and received her PhD incomputer engineering in 2001 from the University of Rome, Italy. She is nowassistant professor at the University of Rome. She was researcher at the Fondazione Ugo Bordoni in 1997, visiting scholar theUniversity of Texas at Dallas in 1999–2000 and research assistant at the University of Rome'Tor Vergata' in 2000–2002. She was program chair and program committee member of several internationalconferences. She has served on the editorial board of Elsevier Computer Networks and ACM/Springer-Wireless Networks. Her research interests lie in the area of wireless mobile networks and web based systems.

Annalisa Massini graduated in Mathematics in 1989 and got herPh.D. in Computer Science in 1993, at University of Rome ''La Sapienza,'' Italy.In 1996 she became assistant professor at the Department of ComputerScience, University of Rome ''La Sapienza''.Since 2001 she is associate professor at the Department of ComputerScience, University of Rome ''La Sapienza''.Her research interests include wireless networks, algorithms on interconnectionnetworks, layout of networks topologies, parallel arithmetic units.

Simone Silvestri graduated with honors in 2005 and is currently a PostDoc researcher in Computer Science at Sapienza,University of Rome, Italy. He was a visiting scholar at the-Electrical and Electronic Engineering Department, Imperial College, London.He served as program committee member of several international conferences.His research interests lie in the area of sensor networks and web based systems.