International Journal of Parallel, Emergent and Distributed Systems
Vol. , No. , , 1–35

Cluster-based service discovery for heterogeneous wireless sensor networks

R.S. MARIN-PERIANU*, J. SCHOLTEN, P.J.M. HAVINGA and P.H. HARTEL
University of Twente, The Netherlands
()

We propose an energy-efficient service discovery protocol for heterogeneous wireless sensor networks. Our solution exploits a cluster overlay, where the clusterhead nodes form a distributed service registry. A service lookup results in visiting only the clusterhead nodes. We aim for minimizing the communication costs during discovery of services and maintenance of a functional distributed service registry. To achieve these objectives we propose a clustering algorithm which makes decisions based on 1-hop neighbourhood information, avoids chain reactions and constructs a set of sparsely distributed clusterheads. We analyse how the properties of the clustering structure influence the performance of the service discovery protocol, by comparing our proposed clustering algorithm with DMAC. We evaluate the performance and the tradeoffs between the cluster-based service discovery approaches and the traditional flood-based solutions. We investigate the level of network heterogeneity where clustering is feasible for implementation in a wireless sensor network. Our analysis shows that cluster-based solutions are best suited for heterogeneous dense networks, with limited dynamics.

Keywords: service discovery, clustering, wireless sensor networks

1 Introduction

Wireless Sensor Networks (WSNs) is an emerging technology that opens a wide perspective for future applications in ubiquitous computing and ambient intelligence. Wireless sensor nodes form a dense, large scale network and are expected to function unattended. Their hardware limitations have led to the design of new protocols at the lower communication levels, such as physical, MAC and routing. As the technology evolves, the focus changes toward sensor networks providing services to clients in a wide range of applications. We envision that large-scale sensor infrastructures will eventually be integrated into homes, offices, public places and the environment, where multiple clients will configure, discover and use a variety of services.

We make the distinction between *homogeneous* and *heterogeneous* WSNs. A homogeneous WSN is composed of tiny, resource-constrained devices, using the same platform and having the same hardware capabilities. The functionality of a homogeneous WSN serves mainly the purpose of gathering the sensed data and sending it to a central location. The typical research questions focus on prolonging the lifetime of the network, by designing energy-efficient protocols which distribute the communication overhead evenly among the sensor nodes. A heterogeneous WSN employs a range of different devices, which are able to cooperate in order to achieve a global goal by combining the individual capabilities of the nodes. Small and cheap sensor nodes are deployed with high density and easily attached to people or objects moving in the environment, while the more powerful nodes are able to provide persistent data storage, intensive processing and actuation. In such a network, the objective is to distribute the workload depending on the capabilities of the nodes.

We argue that heterogeneous WSNs have the potential to provide higher quality networking and system services than the homogeneous counterparts. Compared to using one platform that imposes a set of compromises, a heterogeneous collection of devices benefits from the functionality and flexibility provided by the resource-lean nodes, in conjunction with the enhanced capabilities offered by the more endowed nodes. For example, a habitat monitoring application using such a tiered architecture collects data from numerous, inexpensive sensor nodes and processes it on a few larger, more capable and expensive devices [11]. A localization system that uses a few computationally-rich devices, which form an ad-hoc infrastructure, is capable of automatic localization and time synchronization with high precision [15]. Anycast services designed for hybrid sensor/actuator networks provide significant improvements compared to other protocols that do not take into account the capabilities of the resource-rich devices [19].

We propose a service discovery protocol for heterogeneous WSNs [26], which achieves good performance results by taking advantage of network heterogeneity. Designing a service discovery protocol for WSN environments imposes a number of challenges. Firstly, since sensor nodes are likely to be battery powered, our objective is to minimize the energy consumption. As the energy is spent mostly on operating the radio [1], minimizing the energy consumption translates into minimizing the communication cost. The problem is challenging especially in large scale, dense networks, where significant traffic is generated due to the intrinsic broadcast nature of the wireless medium. Secondly, we aim at prolonging the overall network lifetime by distributing the tasks related to service discovery according to the capabilities of the nodes. Finally, as the network is expected to be mobile, the protocol should react rapidly to the topology changes.

The traditional method for service discovery in ad hoc networks is based on flooding, which has the advantage of zero maintenance overhead. However, flooding has obvious limitations with regard to energy-efficiency and scalability, and it is mostly suitable for homogeneous networks. The problem is how to design a service discovery protocol suitable for heterogeneous wireless sensor networks, that reduces the workload of the resource-constraint devices and avoids the significant traffic induced by the traditional flood-based solutions in dense networks. We propose a solution based on clustering, where a set of nodes, selected based on their capabilities, acts as a distributed directory of service registrations for the nodes in their cluster. In this way, (1) the communication costs are reduced, since the service discovery messages are exchanged only among the directory nodes, and (2) the distribution of workload takes into account the capabilities of the nodes [29].

The main contributions of this paper are therefore:

- A lightweight clustering algorithm that builds a distributed directory of service registrations.
- An energy-efficient service discovery protocol that exploits the clustering structure.

Additionally, we address a number of more general questions, regarding the performance and the tradeoffs implied by the clustering approaches:

- *What is the impact of the chosen clustering algorithm on the performance of the service discovery protocol?* We compare our proposed clustering algorithm with DMAC [6], a state of the art counterpart. Through this comparison, we study how the properties of the clustering algorithms influence the performance of the service discovery protocol.
- *How do the cluster-based protocols compare to flooding?* Since flooding requires no maintenance effort, for low rates of service requests it may be more energy-efficient than the clustering alternatives. We investigate what is the frequency threshold for the cluster-based protocols to become more energy efficient than flooding, in terms of global energy consumption. We are also interested in the distribution of energy consumption among nodes with different capabilities. Unlike flooding, we expect that cluster-based protocols reduce the energy consumption on the resource-lean devices.
- *When is the clustering approach feasible for WSNs?* In the case of a homogeneous network composed of resource-constrained sensor nodes that cannot manage multiple service registrations, the problem is whether the clusterheads have enough resources to deal with their assigned roles. We investigate the limit in the network heterogeneity where clustering is still feasible for implementation in a WSN.

The remaining part of the paper is organized as follows: we give an overview of the related work in the fields of service discovery and clustering in Section 2. We present in detail the clustering algorithm and the service discovery protocol for heterogeneous WSNs in Sections 3 and 4. Section 5 presents the simulation environment and the settings we use in our performance evaluation. We address the aforementioned questions in detail in Sections 6, 7 and 8. Section 9 presents the conclusions.

2 Related work

We give a brief overview of distributed service discovery protocols and clustering algorithms designed for ad-hoc networks and pervasive environments.

2.1 *Service discovery protocols*

Service discovery protocols which achieve efficient service lookup in large-scale ad hoc networks exploit either flat or hierarchical overlay structures. The DHT based peer-to-peer techniques [22, 32] employ a flat structure of service registries, while the DNS-like alternatives [18] use a global hierarchy of directory nodes. These techniques generate considerable network traffic and high maintenance overhead, so they are not suitable for WSNs.

More appropriate for this environment are the protocols that use unstructured distributed storage [12, 16]. They are based on broadcast or multicast communication. The service information is obtained using either the *push model*, where service providers advertise periodically the services they offer, and/or the *pull model*, where clients flood the network with discovery messages in search for the desired service (the flooding is either limited to a number of hops or not limited). For example, Lenders et. al [24] propose a service discovery protocol inspired by electrostatic fields from physics. Nodes in the ad-hoc network determine the *potential* of a service depending on the distance to service providers. A service request packet arriving at a node is forwarded to the neighbour with the highest potential. This network-scale proactive approach is less suited for sensor networks, because service advertisements are propagated through the whole network and nodes have to maintain potentials for every advertised service, which is both energy and memory consuming.

For energy-efficiency reasons, cross-layered solutions have also been explored, where service discovery protocols piggyback on the routing messages to issue service request and get replies. Frank and Karl [14] rely on AODV [30], Varshavsky et al. [35] use DSR [20] and DSDV [31]. These routing protocols use flooding to set up paths to destinations. Flooding is a method that limits the scalability of protocols, generating substantial traffic especially for dense networks. However, depending on the topological properties, dynamics, frequency of service requests and the resources present in the network, flooding may be the only feasible method to be used by routing or service discovery protocols. In this paper, we investigate the conditions that determine when flooding achieves better results than our cluster-based solution.

Zone-based protocols and cluster-based protocols proactively maintain routing and service information inside the zone/cluster, while using a reactive search method at the network level. They are closer to our approach, as they can prevent the high-density problem from the flood-based protocols.

Helmy [17] proposes a resource discovery protocol, where each node keeps track of a number of nodes within R hops away, that defines the *zone* of the node. As part of the zone information each node maintains resource information and routes to all the nodes in its zone. Moreover, a node has knowledge of a number of contact nodes outside its zone. The search method implies forwarding the requests to the contact nodes. The disadvantage is that regardless of their capabilities, nodes need to maintain a complete topological view over a number of hops, together with the knowledge on available resources.

On the contrary, the cluster-based solutions have the advantage that the knowledge can be distributed among the members of the clusters, depending on the hierarchical level. However, the protocols proposed for ad-hoc networks build complex clustering structures that require a high maintenance effort. For example, Kozat and Tassiulas [23] build a dominating set (or backbone) to which devices register their services. Due to the high density of nodes in the backbone, lots of loops are generated when a service discovery message travels the backbone nodes. To overcome this drawback, the backbone organizes in a source-based multicast tree. However, building and maintaining two overlays for the same purpose (the dominating set and the multicast tree) is expensive for resource-constraint sensor nodes.

We design a service discovery protocol based on a simple and lightweight clustering structure, which would allow for low maintenance overhead and low discovery cost even in highly dense sensor networks.

2.2 *Clustering algorithms*

Several clustering algorithms have been proposed to support scalable MAC and routing protocols in large ad-hoc and sensor networks. The election of the clusterheads usually involves (1) the dissemination of a set of parameters of each node either to the whole network or to group of nodes and (2) the comparison of these parameters in order to choose the best nodes as clusterheads.

The WCA algorithm proposed by Chatterjee et al. [13] takes into account the node degree, transmission power, battery power and the speed of the nodes, for achieving the optimal operation of the MAC protocol. Each node calculates a combined weight from these parameters, which is disseminated in the whole network. The node with the global minimum weight is chosen as clusterhead. Other protocols base their election decisions on complete information over a number of hops. The algorithm proposed by Amis et. al [3] uses the d-hop information for clusterhead election. Each node initiates two rounds of flooding over d hops for building the cluster membership. When the election algorithm finishes, nodes are at most d hops away from the clusterhead. Another example is the algorithm proposed by Lin and Gerla [25], where the distance between two nodes part of the same cluster is maximum two hops. In order to decide on the cluster membership, each node keeps information of its “locality”, which means the nodes one-hop and two-hops away.

McDonald and Znati [27] describe an (α, t) clustering algorithm designed to support routing in large ad-hoc networks, taking the node mobility as the criteria for cluster organization. The cluster internal paths are expected to be available for a period of time t with a probability of at least α. The intra-cluster routing employs a proactive strategy, while the intercluster routing is reactive. Therefore, each node is aware of the complete intra-cluster topology information and maintains routes to the set of adjacent clusters.

Multi-layer clustering algorithms have been proposed for efficient routing in wireless ad-hoc and sensor networks [4,5]. The clusterheads from level h elect the clusterheads for the next level, $h + 1$. The data from the lower levels is aggregated and transmitted to the higher levels in the hierarchy. The multi-layer clustering algorithms hypothesize that topology changes are slow and infrequent.

We believe that the above mentioned clustering algorithms are not suitable for our purpose of supporting energy-efficient service discovery in WSNs. Firstly, electing the clusterheads based on information from nodes which are multiple hops away leads to high overhead and slow reaction to topology changes. Secondly, maintaining complete intra-cluster information regardless of the capabilities of the nodes is an expensive task for resource-lean devices. Thirdly, the complexity of the multi-layer clustering algorithms leads to a lot of effort in building and maintaining the desired structure. To sum up, we are interested in a simple clustering solution, which can react quickly to topology changes and that requires a low construction and maintenance effort.

An algorithm which meets our conditions is DMAC [6], which constructs and maintains an independent dominating set. Nodes decide based on their one-hop neighbourhood information, which assures rapid reaction to topology changes. DMAC has also the advantage that the topology can change during the cluster formation. However, DMAC suffers from the *chain reaction* phenomenon, where a single topology change in the network may trigger significant changes in dominating set. For a distributed directory composed of nodes from the dominating set, the chain reaction causes additional overhead for maintaining consistent service registries. We compare the performance of our clustering algorithm with DMAC, when using them as structural basis for our service discovery protocol.

3 Clustering algorithm

3.1 *Design considerations*

The clustering algorithm constructs an overlay network which facilitates the discovery of services in an energy-efficient fashion. We discuss from the design perspective several techniques for reducing the communication cost during (1) discovery of services and (2) maintenance of the distributed directory.

During the discovery process, messages are exchanged among the clusterhead nodes. To minimize the discovery cost, the root nodes have to be sparsely distributed on the deployment area, as a high density of root nodes (or clusters) would lead to a high communication cost during discovery. Therefore, the clustering algorithm should construct an *independent set* of clusterheads, i.e. two root nodes are not allowed to be neighbours.

In the following, we give the design considerations for minimizing the communication cost during the maintenance of the distributed directory:

- *Make decisions based on 1-hop neighbourhood information.* Clustering algorithms that require each node to have complete topology knowledge over a number of hops are expensive with regard to the maintenance cost. We aim to build a lightweight clustering structure that requires only the 1-hop neighbourhood topology information.
- *Avoid chain reactions.* A chain reaction appears when a single network topology change determine reclustering throughout the network. For a distributed directory composed of clusterhead nodes, a chain reaction leads to high overhead for maintaining consistent service registries. Therefore, an energy-efficient solution should avoid chain reactions, such that local topology changes determine only local modifications of the directory structure.
- *Distribute the knowledge depending on the capabilities of the nodes.* To minimize the maintenance effort and to relieve the resource-lean devices of energy-consuming duties, the knowledge on adjacent clusters and the intra-cluster information should be distributed depending on the capabilities of the nodes.

In theory, building an independent set of root nodes and avoiding a chain reaction comes at the expense of constructing clusters with an arbitrary height. However, in practice, we can achieve small-height clusters without imposing a maximal height limit (see Section 6.1.2).

3.2 *Network model*

We model a wireless network as an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of links that directly connect two nodes.

Two nodes u and v are neighbours if there is a direct communication channel between u and v. Each node is assigned (1) a unique hardware identifier, termed the *address* of the node, and (2) a weight, termed the *capability grade*, representing an estimate of the node's dynamics and available resources. The higher the capability grade, the more suitable is the node for the clusterhead role. We make the following assumptions:

- The capability grades are unique, as the node hardware identifier may be used to break ties.
- The lower layers (such as MAC) filter out asymmetrical links, so that we can rely on bidirectional communication.
- A node is aware of its neighbours and their capability grades.
- The one-hop communication is reliable, and can be achieved for example by using a simple stop-and-wait ARQ protocol [7].

Our clustering structure is a *forest* composed of a set of *trees* or *clusters*. The *height* of the cluster is the longest path from the root node to a leaf. We say that two trees are *adjacent* if there are two nodes, one from each tree, that are connected through a link.

Given a node v, we use the following notation:

- $c(v)$ is the capability grade of v
- $p(v)$ is the parent of v
- $r(v)$ is the root (or clusterhead) of the cluster of v
- $\Gamma(v)$ is the open neighbourhood of v, $\Gamma(v) = \{u \in V \mid (u, v) \in E\}$
- $\Gamma^+(v)$ is the closed neighbourhood of v, $\Gamma^+(v) = \Gamma(v) \cup \{v\}$
- $\Delta(v)$ is the set of children of node v, $\Delta(v) = \{u \in V \mid p(u) = v\}$
- $p_k(v)$ is the parent of order k, defined as $p_0(v) = v$, $p_{k+1}(v) = p(p_k(v))$
- $C(v)$ is the set of nodes that are part of the same cluster as v, $C(v) = \{u \in V \mid r(u) = r(v)\}$
- $T(v)$ is the set of nodes from the sub-tree rooted at v, $T(v) = \{u \in V \mid \exists k$ such that $p_k(u) = v\}$
- $R_u(v)$ is the set of adjacent clusters of node v, represented by their roots, that can be reached through node u, where $u \in \Gamma(v)$ (u is the next hop on the path to the adjacent cluster):
  - if $u \in \Delta(v)$, then $R_u(v)$ is the set of root nodes of clusters adjacent to the sub-tree rooted at v. Formally, $R_u(v) = \{r \in V \setminus \{r(v)\} \mid \exists x \in T(u), \exists y \in \Gamma(x)$ such that $r = r(y)\}$;
  - if $u \in C(v) \setminus \Delta(v)$, then $R_u(v) = \emptyset$;
  - if $u \notin C(v)$, then $R_u(v) = \{r(u)\}$.
- $S(v)$ is the set of services provided by node v
- $S_u(v)$ is the set of services registered to v by $u \in \Delta(v)$. Formally, $\forall u \in \Delta(v),\ S_u(v) = \{S(x) \mid x \in T(u)\}$.

3.3 *Construction of clusters*

The construction of clusters follows the idea of a greedy algorithm, where nodes choose a neighbour with higher capability grade as *parent*, while other nodes that do not have such a neighbour are *roots*. The message *SetRoot* is used for propagating the address of the root node to all the members of the clusters. The *Initialization* phase and the event *SetRoot* from Algorithm *1* give a formal description for the construction of clusters. Briefly, the protocol works as follows:

- Nodes that have the highest capability grades among their neighbours declare themselves clusterheads and broadcast a *SetRoot* message announcing their roles.
- The remaining nodes choose as parent the neighbour with the highest capability grade.
- When a node receives a *SetRoot* message from its parent, it learns the cluster membership and rebroadcasts the *SetRoot* message.

3.4 *Knowledge on adjacent clusters*

The root nodes learn about the adjacent clusters from the nodes placed at the cluster borders. During the propagation of the broadcast message *SetRoot* down to the leaf nodes, the message is also received by nodes from adjacent clusters. These nodes store the adjacent root identity in their $R_u(v)$ sets and report it to their parents. The information is propagated up in the tree with a message which we term *UpdateInfo*. Through this message, nodes learn the next hops for the paths leading to the clusters adjacent to their sub-trees. In particular, the root nodes learn the adjacent clusters and the next hops on the paths to reach their clusterheads. Figure 1 gives an intuitive example of learning the adjacent clusters.

The events of receiving messages *SetRoot* and *UpdateInfo* from Algorithm *1* describe how the knowledge and the paths to adjacent clusters are updated for a given node v. Duplicate *UpdateInfo* messages are discarded: a node v sends the message *UpdateInfo* to its parent if and only if the set of known root nodes changes. This means that if v is informed about a root node from one neighbour, but it knows already about this root through another neighbour, v does not propagate the information to the parent again.

3.5 *Maintenance in face of topology changes*

We analyse how the clustering structure adapts to dynamic environments. We term the events regarding topology changes *LinkAdd* and *LinkDelete*. Al-

Algorithm *1* Clustering algorithm - node v (events/actions)

```
 Initialization: // Parent is chosen
 1. $r(v) \leftarrow \perp;\ R_m(v) \leftarrow \emptyset,\ \forall m \in \Gamma(v)$
 2. choose $p(v) \in \Gamma^+(v)$ such that $c(p(v)) = max\{c(m) \mid m \in \Gamma^+(v)\}$
 3. if $p(v) = v$ then
 4.     $r(v) \leftarrow v$ // I am root
 5.     Send SetRoot $(v,\ r(v))$ to neighbours
 6. end if

 SetRoot $(u, r)$: // Receive root $r$ from neighbour $u$
 1. $R_0 = \bigcup_{m \in \Gamma(v)} R_m(v)$
 2. if $(p(v) = u) \wedge (r(v) \neq r)$ then
 3.     $r(v) \leftarrow r$
 4.     Send SetRoot$(v,\ r(v))$ to neighbours
 5.     $\forall m \in \Gamma(v),\ R_m(v) \leftarrow R_m(v) \setminus \{r(v)\}$
 6. else if $(r(v) \neq r)$ then
 7.     $R_u(v) \leftarrow \{r\}$
 8. else if $(r(v) = r)$ then
 9.     $R_u(v) \leftarrow \emptyset$
10. end if
11. if $(v \neq p(v)) \wedge (R_0 \neq \bigcup_{m \in \Gamma(v)} R_m(v))$ then
12.     Send UpdateInfo $(v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
13. end if

 UpdateInfo $(u, R)$: // Receive adjacent clusters $R$ from $u$
 1. $R_0 = \bigcup_{m \in \Gamma(v)} R_m(v)$;
 2. $R_u(v) \leftarrow R \setminus \{r(v)\}$
 3. if $(v \neq p(v))) \wedge (R_0 \neq \bigcup_{m \in \Gamma(v)} R_m(v))$ then
 4.     Send UpdateInfo$(v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
 5. end if

 LinkAdd $(u, c)$: // $u$ added to neighbourhood, with capability $c$
 1. $\Gamma(v) \leftarrow \Gamma(v) \cup \{u\}$
 2. if $c > c(p(v))$ then
 3.     if $(v \neq p(v))$ then
 4.         Send UpdateInfo $(v, \emptyset)$ to $p(v)$
 5.     end if
 6.     $p(v) \leftarrow u$ // The new neighbour becomes parent
 7.     Send UpdateInfo $(v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
 8. end if
 9. Send SetRoot $(v, r(v))$ to neighbours

 LinkDelete $(u)$: // $u$ deleted from neighbourhood
 1. $R_0 = \bigcup_{m \in \Gamma(v)} R_m(v)$
 2. $\Gamma(v) \leftarrow \Gamma(v) \setminus \{u\}$ // Remove neighbour
 3. if $u = p(v)$ then
 4.     choose $p(v) \in \Gamma^+(v)$ such that $c(p(v)) = max\{c(m) | m \in \Gamma^+(v)\}$
 5.     if $p(v) = v$ then
 6.         $r(v) \leftarrow v$
 7.         Send SetRoot $(v, r(v))$ to neighbours
 8.     else
 9.         if $r(v) \neq r(p(v))$ then
10.             $r(v) \leftarrow r(p(v))$ // Update cluster membership
11.             $\forall m \in \Gamma(v),\ R_m(v) \leftarrow R_m(v) \setminus \{r(v)\}$
12.             Send SetRoot $(v, r(v))$ to neighbours
13.         end if
14.         Send UpdateInfo $(v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
15.     end if
16. else if $(v \neq p(v)) \wedge (R_0 \neq \bigcup_{m \in \Gamma(v)} R_m(v))$ then
17.     Send UpdateInfo $(v, \bigcup_{m \in \Gamma(v)} R_m(v))$ to $p(v)$
18. end if
```

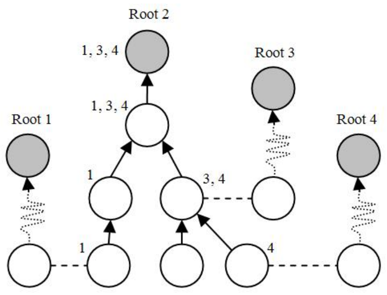

Figure 1. Nodes learn from neighbours which are the adjacent clusters and propagate the knowledge to the parents.

gorithm *1* gives a detailed description of the behaviour of node v when these events occur. In short, there are two situations where nodes adjust their cluster membership:

- A node discovers a new neighbour with a higher capability grade than its current parent. The node then selects that neighbour as its new parent.
- A node detects the failure of the link to its parent. The node then chooses as new parent the node with the highest capability grade in its neighbourhood.

Besides reclustering, topology changes may also require modifications in the knowledge on adjacent clusters. The *SetRoot* message informs nodes about the cluster membership of their neighbours, while the *UpdateInfo* message is used for transmitting the updates from children to their parents. We distinguish the following situations:

- A node v detects a new neighbour from a different cluster. Consequently, v adds the root of that cluster to its knowledge.
- A node v switches from parent p_0 to p_1. Then v (1) notifies p_0 to remove the information associated with v and (2) sends the list of adjacent clusters to p_1.
- A node v detects the failure of the link to one of its neighbours u. As a result, v erases the knowledge associated with u.
- Any change of global knowledge at node v results in transmitting the message *UpdateInfo* from v to its parent.

3.6 *A clustering alternative: DMAC*

We choose DMAC as a viable clustering alternative for our service discovery protocol. Its simplicity and good performance results [8] make it suitable for sensor environments. DMAC achieves fast convergence, as nodes decide their roles using only 1-hop neighbourhood information. The clusters are constructed based on unique weights assigned to nodes. The higher the weight, the more suitable is the node for the clusterhead role. The difference with our clustering algorithm is that DMAC imposes a maximum cluster height of one, whereas our protocol in principle may lead to an arbitrary cluster height. For the construction of clusters, DMAC uses two types of broadcast messages, *Clusterhead* and *Join*, announcing the roles of the nodes to their neighbours. The role decision of a node is dependent on the decisions of the neighbours with higher weights. Therefore, a single topology change may trigger reclustering of a whole chain of dependent nodes. This phenomenon is called a *chain reaction*. For a distributed directory composed of clusterhead nodes, the chain reaction leads to high overhead for maintaining consistent service registries. In Section 6.2 we study the impact of the cluster height and the chain reaction over the performance of the service discovery protocol, in comparison to our proposed clustering solution.

4 Service discovery protocol

During service discovery, service request messages look for a service match in the set of nodes which are part of the distributed directory. We first describe how the services are registered to the directory nodes and then we present the service discovery process.

4.1 *Service registration*

During service registration, each node sends the local service descriptions and the descriptions received from its children to the parent node. In this way, a node learns the service descriptions of the nodes placed lower in hierarchy. In particular, a root node is informed of all the service descriptions offered by the nodes in its cluster. Since the registration process requires unicast messages to be transmitted from children to parents, it can be integrated with the transfer of knowledge on adjacent clusters. Thus, the message *UpdateInfo* is used for both service registrations and transferring the knowledge on adjacent clusters. Algorithm *1* shows the integrated version of the *UpdateInfo* message, where a node updates the information on both the adjacent clusters and the known services.

In the following we describe how the distributed service registry is kept consistent when the topology changes. In the case of a parent reselection, a child node v registers the services from its sub-tree with the new parent p_1, and notifies the old parent p_0 (if it is still reachable) to purge the outdated service information. The process is transparent to the other nodes in the sub-tree rooted at v. If the overall service information at p_0 and p_1 changes due to the parent reselection, the modifications are propagated up in the hierarchy.

Algorithm *1* Service registration - node v

```
    UpdateInfo $(u, R, S)$:
    // receive adjacent clusters $R$ and services $S$ from $u$
 1. $R_0 = \bigcup_{m \in \Gamma(v)} R_m(v)$
 2. $S_0 = \bigcup_{m \in \Delta(v)} S_m(v) \cup S(v)$
 3. if $R = \emptyset$ then
 4.     $\Delta(v) \leftarrow \Delta(v) \setminus \{u\}$
 5.     $S_u(v) \leftarrow \emptyset$
 6. else
 7.     $\Delta(v) \leftarrow \Delta(v) \cup \{u\}$
 8.     $S_u(v) \leftarrow S$
 9. end if
10. $R_u(v) \leftarrow R \setminus \{r(v)\}$
11. if $(v \neq p(v)) \wedge ((R_0 \neq \bigcup_{m \in \Gamma(v)} R_m(v)) \vee (S_0 \neq \bigcup_{m \in \Delta(v)} S_m(v) \cup S(v)))$ then
12.     Send $UpdateInfo(v, \bigcup_{m \in \Gamma(v)} R_m(v), \bigcup_{m \in \Delta(v)} S_m(v) \cup S(v))$ to $p(v)$
13. end if
```

4.2 *Service discovery*

The service discovery process uses a distributed directory of service registrations. Suppose a node in the network generates a service discovery request *ServDisc*. The request is first checked against the local registrations. In the case where no match is found, the message is forwarded to the parent. This process is repeated until the *ServDisc* message reaches the root of the cluster. When a root node receives a *ServDisc* message and it does not find a match in the local registry, the message is forwarded to the roots of the adjacent clusters. The next hop on the path leading to the adjacent cluster is decided by every node that acts as forwarder of the *ServDisc* message. Each node v along the path checks its $R_u(v)$ sets and picks a neighbour that has a path to the root of the adjacent cluster. In the case where a link is deleted and v cannot forward the *ServDisc* message, it chooses another neighbour that provides a path to destination. If such a neighbour does not exist, v informs its parent that it no longer has a route to the next cluster. The same procedure is repeated until all the paths to destination are tested. If the next cluster is not reachable, the root node erases the cluster from its knowledge.

The result of a service search is typically the address of one or more service providers. This response can be returned by the first node that finds a match in its registry for the requested service. However, in certain situations it may be preferable that the service provider itself issues a reply for the service request. Examples include applications where service descriptions change frequently, or cases where the reply incorporates more information than the address of the node. In these situations, the *ServDisc* message is forwarded down the cluster until it reaches the service provider. In the case where the link to the service provider is deleted or the service description is no longer valid, the service request is sent back to the root node which forwards it to the adjacent clusters.

The service discovery reply may follow the reverse cluster-path to the client, or any other path if a routing protocol is available. For the first case, if there is a cluster partition, the path can be reconstructed using the same search strategy as for the *ServDisc* message, where this time the service is the address of the client.

Caching the service discovery messages is a technique that allows us to cope with mobility. Root nodes cache the *ServDisc* messages for a limited period of time. If a newly arrived node registers a service for which there is a match in the cache, the root node can respond to the old service request. Moreover, when a root node is notified on a new adjacent cluster, it sends the valid service request entries from its cache to the new clusterhead. As a result, the overall hit ratio is improved.

Algorithm *2* describes the protocol, where replies are generated by nodes in the distributed directory and no caching is implemented. The message *ServDisc* has four parameters: the neighbour u that sends the request, the service description s, the final destination d of the message (typically a root node) and a flag f. The flag indicates whether the message is a fresh service discovery request, or it is a failure notification of a previous attempt to reach an adjacent cluster. In the latter case, the failed route is erased from the knowledge on adjacent clusters and another message is sent using an alternate path.

Algorithm *2* Service discovery - node v

```
  ServDisc $(u, s, d, f)$:
  // receive message ServDisc from neighbour $u$, requesting service $s$, destination $d$, flag $f$
 1. if $f = TRUE$ then
 2.     if $s \in \bigcup_{m \in \Delta(v)} S_m(v) \cup S(v)$ then
 3.         Service found; generate reply
 4.     else if $p(v) = v$ then
 5.         for all $r \in \bigcup_{m \in \Gamma(v)} R_m(v)$ do
 6.             Pick $m \in \Gamma(v)$ such that $r \in R_m(v)$
 7.             Send $ServDisc(v, s, r, TRUE)$ to $m$
 8.         end for
 9.     else if $d = r(v)$ then
10.         Send $ServDisc(v, s, d, TRUE)$ to $p(v)$
11.     else if $d \in \bigcup_{m \in \Gamma(v)} R_m(v)$ then
12.         Pick $m \in \Gamma(v)$ such that $d \in R_m(v)$
13.         Send $ServDisc(v, s, d, TRUE)$ to $m$
14.     else
15.         Send $ServDisc(v, s, d, FALSE)$ to $p(v)$
16.     end if
17. else
18.     $R_u(v) \leftarrow R_u(v) \setminus \{d\}$
19.     if $d \in \bigcup_{m \in \Gamma(v)} R_m(v)$ then
20.         Pick $m \in \Gamma(v)$ such that $r \in R_m(v)$
21.         Send $ServDisc(v, s, d, TRUE)$ to $m$
22.     else if $p(v) \neq v$ then
23.         Send $ServDisc(v, s, d, FALSE)$ to $p(v)$
24.     end if
25. end if
```

5 Simulation settings

For our experiments we use the OMNeT++ [34] simulation environment. We generate a random network, by placing N nodes uniformly distributed on a square area of size $a \times a$. We consider links to be bidirectional, so nodes have the same transmission range, r. There is a link between two nodes if the distance between them is less or equal to r. We analyse the performance of our proposed algorithms under different network densities. For changing the network density, we keep the area size to a fixed value ($a = 500m$), and we vary the transmission range and/or the number of nodes .

In a heterogeneous network, the distribution of capabilities on the nodes can be assumed uniform. For example, the infrastructure of beacons used for localization in a WSN, which have the same level of capabilities, has either a uniform grid placement [9], or uses a stochastic uniform distribution [15]. Applications that require sensing and control over a large area employ resource-rich nodes capable of acting, which are uniformly distributed on the entire region [2]. Both resource-constrained and powerful nodes in a heterogeneous sensor network are uniformly distributed over the implementation area for the purpose of surveillance [28]. Following this remark, the nodes in our simulations choose their capability grades from a uniform distribution. Moreover, we assure that static nodes have higher capability grades than mobile nodes.

In the following, we use the notation C4SD (Clustering for Service Discovery) for our proposed clustering algorithm. We test the performance of both C4SD and DMAC under the same topological conditions. We extend DMAC with the algorithm for maintaining the knowledge on adjacent clusters and for updating the service registry, using the *UpdateInfo* message. We use a heartbeat broadcast message periodically sent by every node to maintain the neighbourhood information and to trigger the events *LinkAdd* and *LinkDelete*. The heartbeat is also used for the cluster setup and maintenance, replacing the *SetRoot* message for C4SD and the *Clusterhead* and *Join* messages for DMAC. The focus of our comparative simulations is the overhead induced by the *UpdateInfo* and *ServDisc* messages in dynamic environments.

For measuring the properties of the clustering structure we run static simulations, where we use the cyclic distance model for link formation, in order to avoid the border effects [8]. In this model, nodes at the border of the system area establish links via the borderline to the nodes located at the opposite side of the area. This setup approximates an area where nodes are distributed according to a Poisson point process. Similar to the DMAC performance analysis of Bettstetter [8], we provide results for three transmission ranges: $r = 0.1a$, $r = 0.2a$ and $r = 0.3a$.

For the analysis of the service discovery protocol we run dynamic experiments, where we vary both the density and the mobility of the network. For changing the network density, we take the average of the three transmission ranges considered above ($r = 0.2a$), and we vary the number of nodes. For changing the network mobility, we vary the percentage of mobile nodes in the network. Following a simplistic scenario of people walking and stopping, we use the random waypoint mobility model [21], where the mobile nodes move with a speed of 1m/s and, upon arrival at an intermediate point, pause for 30 seconds before restarting. According to the recommendation of Camp et al. [10], we discard the initial 1000 seconds of simulation time in each simulation trial and we count the number of messages for the next 1000 seconds.

6 What is the impact of the chosen clustering algorithm on the SDP?

In this section we study the impact of the chosen clustering algorithm on the performance of the service discovery protocol. For our analysis, we compare C4SD with DMAC. First, we study the properties of the algorithms in terms of cluster density and cluster height. Second, we measure the performance of the service discovery protocol running on both structures under the same topological conditions.

6.1 *Properties of the clustering algorithms*

6.1.1 ***Cluster density.*** We define the cluster density as the expected percentage of clusterheads to the total number of nodes in the network. The cluster density is an important measure for the performance of a clustering algorithm that is intended to be used as a basis for a search mechanism. A high density of clusters leads to a large number of messages exchanged in the discovery phase.

We consider the nodes distributed on an area according to a homogeneous Poisson point process with density $\rho = N/a^2$. The expected node degree, which we term the *network density*, is [8]:

$$E\{D\} = \rho\pi r^2 = N\frac{r^2\pi}{a^2} \tag{1}$$

The spatial distribution of the root nodes for both clustering algorithms belongs to the family of *hard-core point processes* [33], in which the constituent points are forbidden to lie closer together than a certain minimum distance. For our clustering algorithm, we approximate the cluster density by using the *Matérn hard-core process.* The retaining probability of nodes that become roots, i.e. the cluster density, is the following:

$$P_{C4SD} = \frac{1}{E\{D\}}(1 - e^{-E\{D\}}) \tag{2}$$

This result enables us to compute the expected number of clusters:

$$E_{C4SD} = P_{C4SD}N = \frac{a^2}{\pi r^2}(1 - e^{-\frac{N\pi r^2}{a^2}}) \tag{3}$$

The results obtained by Bettstetter [8] for the DMAC clustering algorithm indicate the following probability for a randomly chosen node to become clusterhead:

$$P_{DMAC} = \frac{1}{1 + \frac{E\{D\}}{2}} \tag{4}$$

Thus, the expected number of clusters in DMAC is:

$$E_{DMAC} = P_{DMAC}N = \frac{1}{\frac{1}{N} + \frac{\pi r^2}{2a^2}} \tag{5}$$

From Eq. 3 and 5 it can be easily shown that:

- $P_{C4SD} < P_{DMAC}$
- for r and a fixed, the function $f(N) = E_{DMAC} - E_{C4SD}$ is strictly increasing
- $\lim_{N\to\infty} E_{DMAC} = 2\lim_{N\to\infty} E_{C4SD}$

We can conclude that: (1) C4SD has a lower cluster density, (2) the difference in the number of clusters built by the two protocols increases with the network density and (3) C4SD almost halves the total number of clusters for saturated areas.

For validating the theoretical estimation of C4SD we run simulations with three transmission ranges $0.1a$, $0.2a$ and $0.3a$, and we count the number of clusters formed in each experiment. The mean of the samples are shown in Figure 2, with $5th$ and $95th$ percentile. We also plot the theoretical estimations for the three values of the transmission range r. The figure shows that the estimated values match exactly the simulation results.

In the first part of the curve, the nodes are sparsely distributed on the simulation area and form clusters with only one member. When the network becomes dense, the new nodes added either join the already existing clusters or they form their own cluster and force the root nodes in the neighbourhood to join. From a certain number of nodes in the area, adding new nodes does not change the number of clusters.

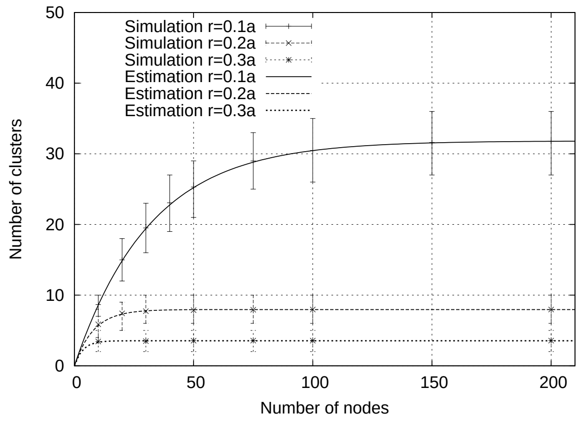

Figure 2. C4SD - average number of clusters on $a \times a$ area, with $5th$ and $95th$ percentile.

6.1.2 ***Cluster height.*** The design particularity of DMAC imposes a maximum cluster height of one, so in this section we are only interested to evaluate the cluster height for C4SD, which does not have a theoretical limit. We run simulations with three transmission ranges, and for each of them we vary the number of nodes. Figure 3 shows the results as a function of the expected node degree, with the $5th$ and $95th$ percentile values as error bars.

We can notice that for all the three transmission ranges, the points follow the same curve. We conclude that, similarly to the cluster density, the average cluster height is only a function of the expected node degree (or network density). The second conclusion is that the average cluster height is lower than 2, and at least 95% of the clusters have the hight lower or equal to three. This result indicates that we can achieve relatively small-height clusters without imposing a maximal hop diameter limit, which would increase the maintenance effort and generate the chain reaction effects.

6.2 *Service discovery performance*

We test the performance of the service discovery protocol using both DMAC and C4SD, under the same topological and mobility conditions. Due to the mentioned dissimilarities between the two protocols, we expect different behaviours when using them for discovery purposes.

6.2.1 ***Maintenance overhead.*** In the first experiment we study the impact of the network density over the maintenance overhead (number of *UpdateInfo* messages), when 50% of the nodes are moving acording to the mobility model described in Section 5.

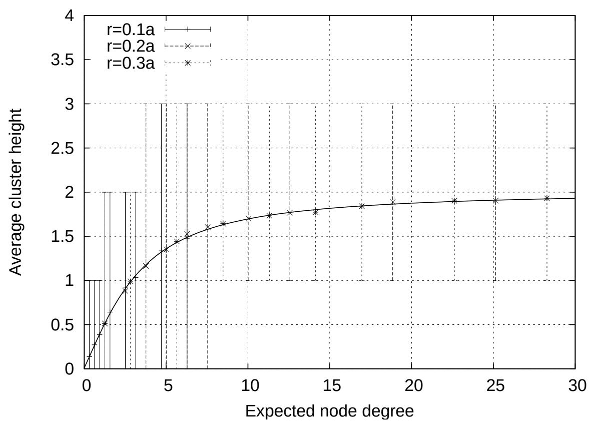

Figure 3. C4SD - average cluster height with 5th and 95th percentiles.

When a node moves from one cluster to another, the old service registration is deleted and a new registration is sent to the new clusterhead. However, the knowledge on adjacent clusters needs more overhead: when a node v moves from one cluster to another, all former neighbours of v must delete the information related to v and report the change to their parents, which can belong to different clusters. Similarly, all the new neighbours of v must add the information provided by v and send it to their parents. On the one hand, due to lower cluster density, C4SD has a lower overhead of maintaining the knowledge on adjacent clusters. On the other hand, the service registration is cheaper at DMAC due to the smaller cluster height. We are interested to examine the cumulative maintenance overhead with different network densities.

Figure 4 shows the average number of messages sent and received by a node in the network in one second of simulation time. For sparse networks, where there are few neighbouring clusters, the DMAC protocol behaves better. For dense networks, the effort for maintaining the knowledge of adjacent clusters becomes prevalent over the overhead of service registrations, and thus C4SD overtakes DMAC.

We analyse the behaviour further in terms of maintenance overhead when increasing the network mobility. Figure 5 shows the experimental results with 100 nodes and percentage of mobile nodes between 10% and 90%. We count the average number of messages per second sent and received by a node. Compared to DMAC, C4SD behaves progressively better when increasing the network mobility. The reason is that the chain reaction inherent to DMAC triggers additional maintenance overhead of the directory structure, where the service information and the knowledge on adjacent clusters have to be updated at the new clusterheads. The more dynamic the network, the more probable is this

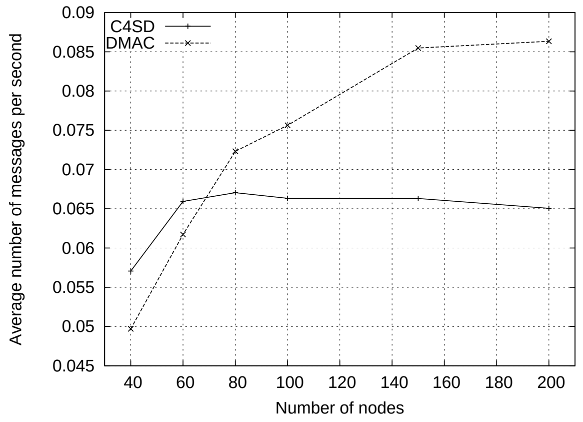

Figure 4. The average number of *UpdateInfo* messages sent and received per node in one second depending on the number of nodes.

reaction to occur.

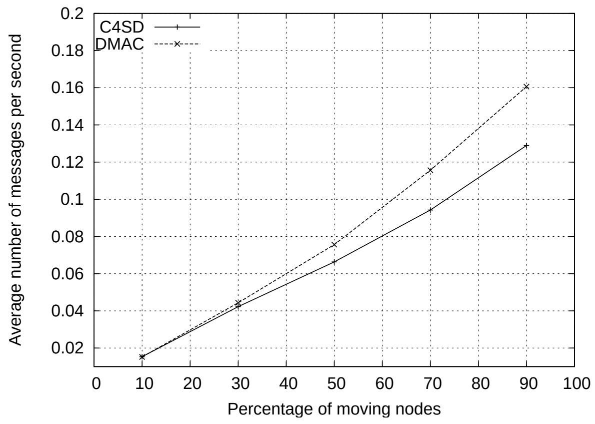

Figure 5. The average number of *UpdateInfo* messages sent and received per node in one second depending on the percentage of moving nodes.

6.2.2 ***Hit ratio.*** Since C4SD has the average cluster height higher than DMAC, the convergence of service registrations is slower. In consequence, we expect DMAC to have a better hit ratio. For a fair comparison, we assume that each node provides exactly one service and for each service there is exactly one service provider. We generate random service requests from arbitrary chosen nodes. During 1000 seconds of simulation time we issue 10 service requests, with a delay of 100 seconds. The *ServDisc* messages are forwarded to the service provider (see Section 4.2). If the service request reaches the matching service provider, we have a hit.

In our first experiments, no caching mechanism is involved. Figure 6 shows the results depending on the percentage of moving nodes. As expected, DMAC performs better than C4SD due to faster convergence. However, DMAC hit ratio drops similarly when increasing the network mobility. In our second set of experiments we implement a limited-time caching of service requests (see Section 4.2). By implementing caching we obtain a high hit ratio for both protocols, which is above 0.98 for all mobility cases that we consider (see Figure 6).

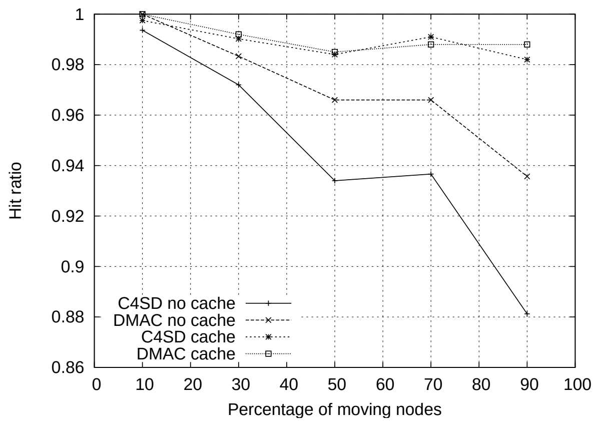

Figure 6. Ratio of successful service requests depending on the percentage of moving nodes.

6.2.3 ***The cost of service discovery.*** We are interested in the number of *ServDisc* messages exchanged during one service discovery phase. Since C4SD has a lower cluster degree, we expect that it also experiences a lower discovery cost. Figure 7 shows the average number of service discovery messages sent and received per node for a network of 100 nodes, depending on the percentage of moving nodes. We represent the cost of service discovery with and without caching. We notice that caching implies more messages spent in the service discovery phase. The discovery cost is significantly smaller for C4SD (up to 50%), due to the lower cluster density. Moreover, DMAC experiences a rapid growth in the discovery cost when caching is implemented.

Figure 8 shows the number of service discovery messages sent and received per node, for a network with 50% of moving nodes, depending on the number of nodes. For dense networks, the number of messages per node decreases when

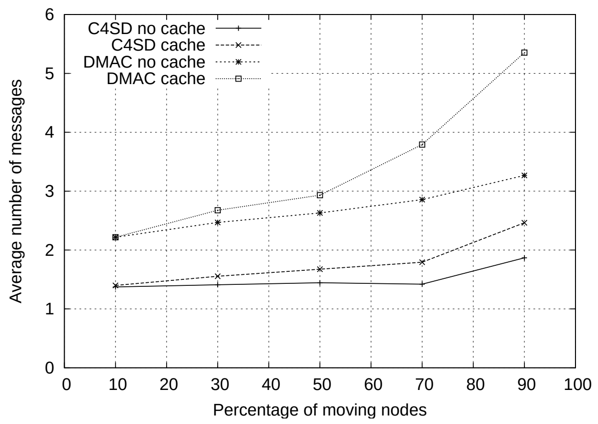

Figure 7. The average number of *ServDisc* messages sent and received per node depending on the percentage of moving nodes.

increasing the network density. This is an inherent property of our cluster-based service discovery protocol, where the discovery messages are exchanged among the root nodes. The reason is that the number of clusters converges to a finite value when the network density increases, for both C4SD and DMAC (see Section 6.1.1). The figure also shows the number of messages exchanged during a service discovery phase when using the traditional flooding method. Contrary to the cluster-based approach, the number of messages per node grows proportionally with the number of nodes in the network.

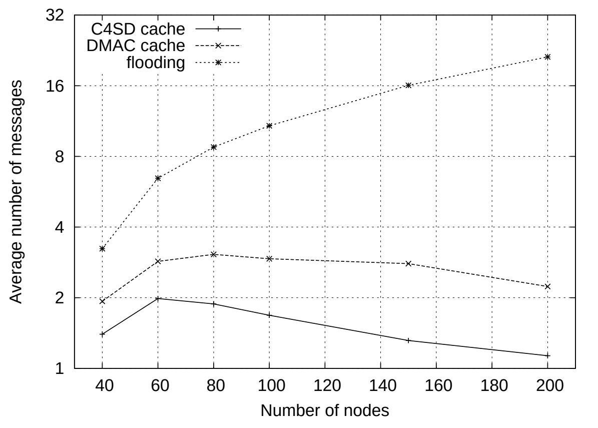

Figure 8. The average number of *ServDisc* messages sent and received per node depending on the number of nodes.

6.2.4 ***The cost of service reply.*** The service reply messages may use either a routing protocol which is implemented on the nodes, or the reverse cluster-path to the client, in case where nodes store in the local cache the service request messages, together with the previous clusterhead.

For the second case, the message travels back to the source on the reverse cluster-path. Since the clusterheads are less dynamic than ordinary nodes, this path is more stable in comparison to the reverse node-path. In case a service reply message reaches a root node that does not have in its cache the next hop to the source (for example a new root node which was not visited by the discovery message), then the root node issues a new discovery message, where the service description is the address of the client. The message travels in a similar way with any service discovery message, until it reaches the client.

In case service replies follow successfully the reverse cluster-paths to the client, we achieve a low reply cost. For example, in a network with 100 nodes, regardless of the dynamics of the nodes, which we vary between 10% and 90%, we obtain an average path length of approximately 6 hops for both protocols. However, the cost of the service reply is substantially increased in the situations where the discovery mechanism has to be used in order to reach the client. In this case, the cost of the service reply is the same as for service discovery (see Section 6.2.3). We are interested how often this situation occurs. Figure 9 shows the ratio of successful service replies which follow the reverse cluster-paths to the client. We notice that for both procols, the hit ratio decreases with the network mobility. However, C4SD has a higher hit ratio, due to the lower number of clusterheads, which are on average more stable than the clusterheads of DMAC.

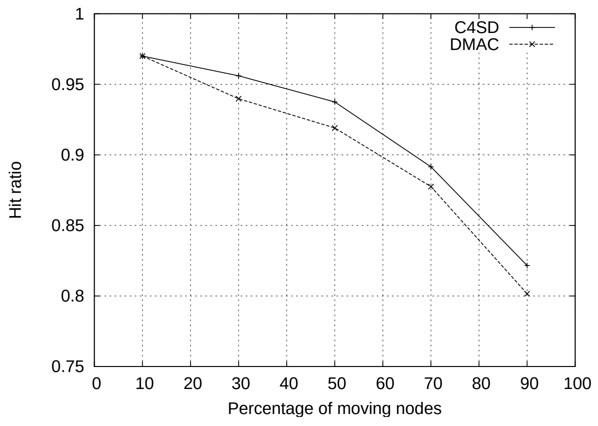

Figure 9. Ratio of successful service replies which follow the reverse cluster-paths to the client.

To conclude this section, we make the following comments:

- The lower the cluster density, the lower the maintenance, discovery and service reply costs.
- Smaller-height clusters achieve faster convergence and higher hit ratio.
- Caching of service requests can be used to improve the hit ratio.
- The chain reaction leads to higher maintenance costs.

7 How do the cluster-based protocols compare to flooding?

Compared to flooding, the cost of service discovery when using a distributed directory is lower, as the service discovery messages visit only the the nodes which are part of the distributed directory (see Figure 8). The cost of flooding grows with the network density, as each node has to receive the same message from all its neighbours. However, it has the advantage of zero-maintenance cost. Therefore, for a low frequency of service requests, flooding may be more energy-efficient than the cluster-based protocols, which spend a lot of effort in maintaining the consistency of service registries. We first investigate what is the threshold for the cluster-based protocols to become more energy efficient than flooding, depending on the mobility of nodes and the network density. Then we analyse how does the proposed service discovery protocol, unlike flooding, succeed in reducing the communication overhead of the resource-lean devices.

7.1 *Threshold estimation*

We use the following notation:

- M is the average number of maintenance messages sent and received by a node in one second.
- D_c is the average number of service discovery messages sent and received by a node during one discovery phase using a clustered structure.
- D_f is the average number of service discovery messages sent and received by a node during one discovery phase using flooding.

From the results obtained in Section 6.2, we can compute the minimum frequency of service requests F, such that the clustering alternatives are more energy-efficient than flooding:

$$F > \frac{M}{D_f - D_c} \tag{6}$$

Figure 10 shows the minimum frequency of service requests depending on the percentage of moving nodes, for a network of 100 nodes. In order for the cluster-based protocols to be more energy-efficient than flooding, the frequency of service requests has to be higher than the threshold values shown in the figure. We notice that this threshold grows as the network becomes more dynamic. As C4SD has lower maintenance and discovery overhead than DMAC, it also has a lower frequency threshold.

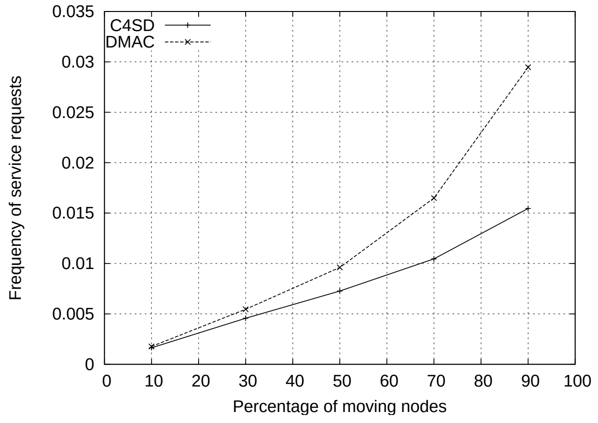

Figure 10. The minimum frequency of service requests, such that cluster-based protocols are more efficient than flooding, depending on the percentage of moving nodes.

We are interested how the threshold frequency evolves when we vary the network density. Figure 11 shows the results, where we keep constant the percentage of mobile nodes to 50%. Due to the fact that the performance of the cluster-based service discovery protocols improves when growing the network density (see Section 6.2), the frequency tradeoff for generating service requests decreases.

7.2 *Distribution of load*

We evaluate the load distribution property of the service discovery protocol based on clustering. Depending on their capabilities, nodes are exposed to different levels of energy consumption. We show that our solution achieves a low overhead for the resource-lean sensor nodes, while the more powerful nodes are entrusted heavier tasks. For measuring the load depending on the capability grades, we simulate a dynamic network of 100 nodes, with 50% of moving nodes. We sort the nodes in ascending order depending on the capability grades, and we group them in 10 classes, such that the weakest nodes belong to the first class and the most powerful nodes fit in the last

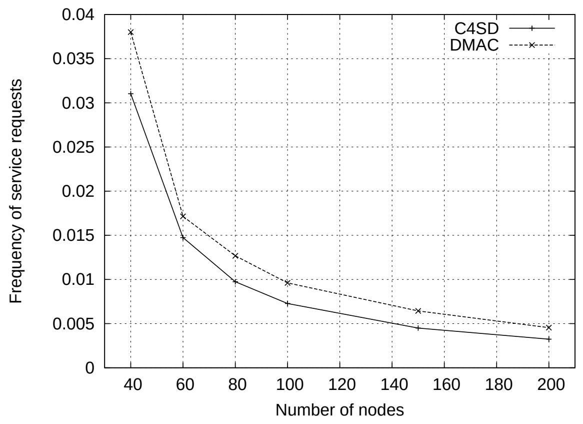

Figure 11. The minimum frequency of service requests, such that cluster-based protocols are more efficient than flooding, depending on the number of nodes.

class. The results are shown depending on the capability group each node belongs.

We are first interested how the capability grades influence the energy consumption during the maintenance of the clustering structure. In Figure 12 we present the number of maintenance messages sent and received by a node per second, depending on the capability group it belongs to. We notice that the least overhead is experienced by the two weakest groups of stationary nodes, most probably small sensors which are part of the static network. The two protocols have comparable tendencies in the division of overhead per capability groups. DMAC spends more messages on mobile and weak devices (first seven groups) and fewer messages on the most powerful and static devices (last three groups).

Next, we investigate how the capability grades influence the energy consumption during service discovery. Figure 13 shows the number of service discovery messages *ServDisc* sent and received on average by every node from a capability group during one service discovery phase, for both clustering protocols. We notice that nodes with higher capability grades spend more energy during discovery. DMAC consumes from 130% more messages for the nodes in the first group capabilities, to 55% for the last group of devices.

The following remarks conclude this section:

- The cluster-based solutions are best suited for dense networks, with limited dynamics.
- Cluster-based solutions, unlike flooding, distribute the communication load according to the capabilities of the nodes.

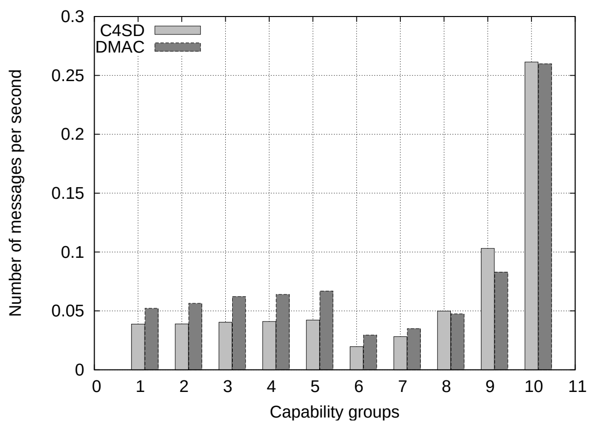

Figure 12. Average number of maintenance messages per groups of capabilities.

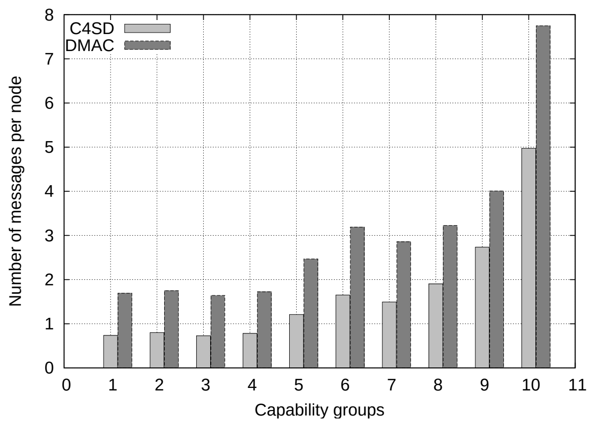

Figure 13. Average number of service discovery messages per groups of capabilities.

8 When is the clustering approach feasible for WSN?

By means of the clustering structure, the service discovery protocol delegates more duties to the nodes with high capability grades, chosen to be parents or roots, and relieves the resource-lean sensor nodes, which are assigned the roles of leaves. The problem is whether the parents and roots have enough resources to deal with their assigned roles. If the network is mostly homogeneous, it might be the case that the difference in capabilities among these nodes is not high, and the node hardware id is often used to break ties. Then, the parents and roots have similar capabilities as the leaves, which may be not sufficient to handle the service registrations and the communication overhead.

Moreover, we showed that the average cluster height for C4SD is below 2, and in 95% of the cases it is below 3. However, for certain constellations of the capability grades, it is possible that the algorithm generates big clusters, in the worst case even a single cluster for the whole network. This would lead to overwhelming tasks for the nodes on high positions in the hierarchy. But imposing a limit for the cluster height does not solve the problem, because big clusters can be constructed even in this situation. Taking for example the case of DMAC, where the maximum cluster height is one, the number of children may be higher than what the root node is able to handle.

The problem of overloaded nodes can be solved by a simple mechanism request-response, where every node asks for permission to register to the parent. If it has enough resources, the parent or root node accepts the registration of the child. Otherwise, it rejects the node. A rejected node searches for another parent to register. In the case where all the potential parents already reached their resource limit, the node becomes a root. The parent decision can take into consideration a multitude of parameters, such as the maximum cluster height, the maximum number of children, the maximum number of services, the amount of memory allocated for service registrations. Using this mechanism, the risk of overloaded nodes is eliminated. However, if all the nodes in the network have so limited resources that they can accept only a small number of children, then the number of clusters becomes significantly large, which leads to high maintenance and discovery costs. In this case, the clustered approach becomes infeasible for implementation in a WSN. We would like to address the question of clustering feasibility, depending on the resources available in the network.

8.1 *Resource distribution model*

In a heterogeneous WSN, the number of devices which are severely resource-constraint is rather high in comparison to the number of devices which are less resource-constraint [2, 11]. A distribution that captures this property is the *Pareto* distribution. If the amount of resources present in a network is Pareto distributed, than the the fraction of the population that owns a small amount of resources per node is rather high, and decreases as the amount of resources increases. We model the available resources on each node according to a Pareto distribution, which has following the probability density function:

$$f(x;k,x_m) = k\frac{x_m^k}{x^{k+1}},\ for\ x \geq x_m, \tag{7}$$

The function is valid for all $x \geq x_m$ where x_m is a positive number, representing the minimum possible value of x, and k is a positive parameter, termed the *Pareto index*. The larger the Pareto index, the smaller the proportion of powerful nodes to the total number of nodes in the network.

In our simulations, every node is assigned a value from a Pareto distribution, representing the available resources. The capability grade is computed such that: (1) the static nodes have higher capabilities than the mobile nodes and (2) within these two groups, the nodes with more resources have higher capabilities than the nodes with less resources.

Our analysis considers the number of nodes below in hierarchy as the limited resource which every node has to manage. As an observation, if we assume that each node offers one service, for each service there is exactly one service provider and the amount of memory occupied is the same for all the service descriptions, then the number of nodes below in hierarchy is proportional to the amount of memory occupied by the registered services.

We are interested how the properties of the clustering structure and the performance of the service discovery protocol are affected when we vary the parameters of the Pareto distribution. We assume that every node can afford at least one node below in hierarchy, so it can store at least one service description in addition to its own. Therefore, we take $x_m = 1$ and we vary k between 0.25 and 10. The network is composed of 100 nodes, out of which 50% are moving. For comparison purposes, we also implement a modified version of DMAC, which takes into account the available resources of the parents during the registration process.

8.2 *The properties of the clustering algorithm*

We show in Figure 14 the average number of clusters obtained for different values of k. We notice that for $k = 0.25$ and $k = 0.5$, the results are close to the basic C4SD and DMAC clustering algorithms (see Section 6.1). For higher values of k, the resource constraint becomes important and the number of clusters increases rapidly with k. For networks with a high Pareto index, almost all the nodes have the capacity of storing only one extra service description, and therefore, they can have only one node below in hierarchy. Consequently, the average number of clusters converges to a value which is close to half of the number of nodes. The clusters formed in a network with a high Pareto index have maximum two nodes. We also notice that the difference in the number of clusters between C4SD and DMAC decreases, making no longer important the clustering algorithm chosen, as the amount of available resources is now the main factor that influences the resulting structure.

Figure 15 shows the average cluster height for different values of the Pareto index. Similarly to the average number of clusters, we notice that for $k = 0.25$ and $k = 0.5$, C4SD has approximately the same average cluster height as the

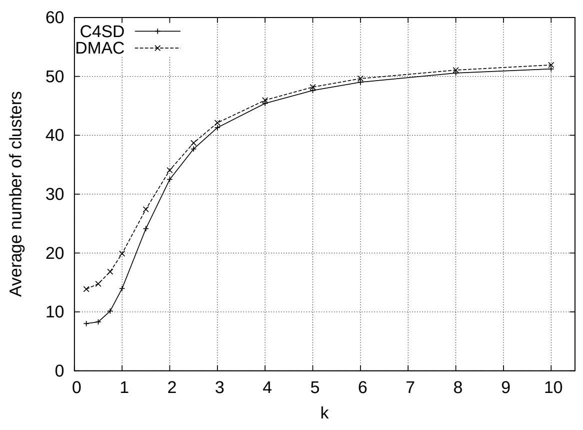

Figure 14. Average number of clusters.

one obtained in Section 6.1. DMAC does not experience significant changes in the average cluster height, since it is designed to construct clusters with a maximum height of 1. The height of C4SD decreases with k and converges to a value less than 1, very close to the average cluster height of DMAC.

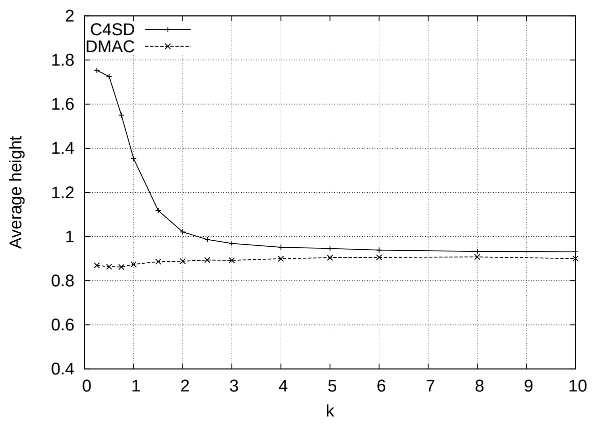

Figure 15. Average cluster height.

8.3 *Service discovery performance*

As the difference in the clustering properties of C4SD and DMAC decreases when increasing the Pareto index, we analyse in the following only the performance of the service discovery protocol when using C4SD.

In Figure 16 we show the average number of maintenance messages sent and received per node during one second of simulation time. Similar to our previous results, for $k \leq 0.5$, the number of maintenance messages is close to the results obtained in Section 6.2.1. For $0.5 < k \leq 1.5$, the number of maintenance messages rapidly increases. In the last part of the figure, the maintenance overhead slowly decreases. The reason is the following: (1) in the first part of the curve, the number of clusters and, consequently, the number of adjacent clusters considerably increase, and therefore, more messages are spent to maintain the knowledge on adjacent clusters; (2) in the second part of the curve, the nodes that were leaves and parents are now root nodes, so these nodes do not transmit any more the *UpdateInfo* messages; the knowledge on adjacent clusters is now learned by listening to the *SetRoot* broadcast messages from neighbours.

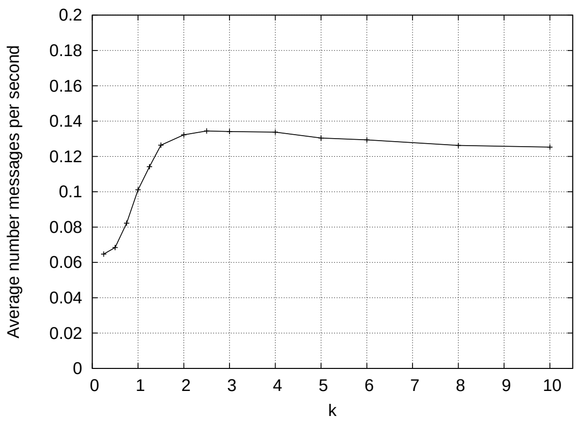

Figure 16. Average number of maintenance messages exchanged by a node per second.

The average number of service discovery messages is represented in Figure 17, together with a comparison to the average number of discovery messages when using flooding. For $k \approx 1.6$ the two plots intersect. It follows that the service discovery protocol based on clustering may be feasible only if the Pareto index is lower than 1.6. For this case, the frequency of service requests has to be considered to be able to answer the feasibility question.

We calculate the minimum frequency of service requests such that the cluster-based protocol is more efficient than flooding, following the same method presented in Section 7. Figure 18 shows the results for different values of $k \leq 1.5$. For $k = 0.25$ and $k = 0.5$, the frequency is roughly the same as in Section 7. As the Pareto index grows, the frequency threshold grows exponentially.

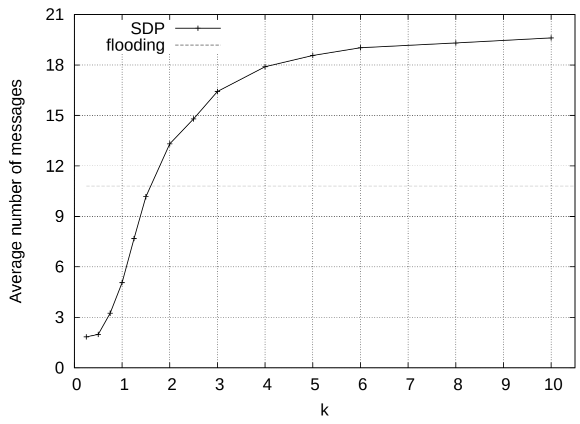

Figure 17. Average number of service discovery messages exchanged by a node during one service discovery phase.

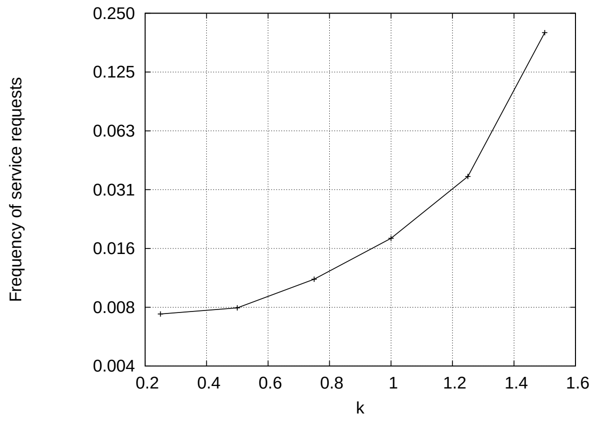

Figure 18. The minimum frequency of service requests, such that the cluster-based protocol is more efficient than flooding, depending on the Pareto index.

To conclude this section, we make the following observations:

- For a high Pareto index, the availability of resources is the main factor that influences the clustering structure.
- The performance of the service discovery protocol decreases while increasing the Pareto index, such that for rather homogeneous networks, flooding is more efficient.

9 Conclusions

This paper proposes a service discovery protocol for heterogeneous WSNs. The protocol relies on a clustering structure that offers distributed storage of service descriptions. The clusterheads act as directories for the services in their clusters. The structure ensures low construction and maintenance overhead, reacts quickly to topology changes and avoids the chain-reaction problems. A service lookup results in visiting only the directory nodes, which ensures a low discovery cost.

The evaluation of the proposed discovery solution addresses a number of questions regarding the performance and the tradeoffs implied by the clustering approaches. Firstly, we focus on the performances of the service discovery protocol depending on the underlying clustering structure, by comparing our clustering algorithm with DMAC. We show that the chain reaction of DMAC determines reclustering and re-registration of services with new clusterheads, implying higher maintenance overhead. The smaller-height clusters of DMAC leads to faster convergence and higher hit ratio. The hit ratio is improved to more than 98% for both protocols if a mechanism of limited-time caching is implemented for service discovery messages. Due to the lower cluster density, our protocol has a lower discovery cost in both implementation alternatives.

Secondly, we compare the performance of our solution with the traditional flooding approach. The cost of service discovery when using a distributed directory is much lower than the flooding alternative, as the service discovery messages visit only the the nodes which are part of the distributed directory. However, flooding has the advantage of zero-maintenance cost. Therefore, in dynamic and sparse networks, when service requests are infrequent, flooding is more energy efficient than cluster-based protocols, which spend a lot of effort in maintaining the consistency of service registries. However, flooding does not distinguish among nodes with different capabilities. We show that for the proposed clusterbased solutions, the resource-lean devices experience a low overhead, while the more powerful nodes consume more energy both during maintenance and discovery of services.

Thirdly, we investigate the limit in the network heterogeneity where clustering is still feasible for implementation in a WSN. We model the resources of the nodes according to a Pareto distribution, where the number of resource-constraint nodes is high in comparison to the more powerful devices. We notice that for a high Pareto index, the availability of resources is the main factor that influences the clustering structure. The performance of the service discovery protocol decreases while increasing the Pareto index, such that for rather homogeneous networks, flooding is more efficient.

For future work we plan to further test the feasibility of our solution by implementing the clustering and service discovery protocols on sensor nodes.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, March 2002.

[2] Ian Akyildiz and Ismail Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, October 2004.

[3] Alan D. Amis, Ravi Prakash, Dung Huynh, and Thai Vuong. Max-min d-cluster formation in wireless ad hoc networks. In *INFOCOM (1)*, pages 32–41. IEEE Computer Society Press, 2000.

[4] Seema Bandyopadhyay and Edward J. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *INFOCOM*, pages 1713–1723. IEEE Computer Society Press, 2003.

[5] Suman Banerjee and Samir Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. In *INFOCOM*, pages 1028–1037. IEEE Computer Society Press, 2001.

[6] Stefano Basagni. Distributed clustering for ad hoc networks. In *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 310–315. IEEE Computer Society, 1999.

[7] Dimitri Bertsekas and Robert Gallager. *Data Networks (2nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[8] C. Bettstetter. *Mobility Modeling, Connectivity, and Adaptive Clustering in Ad Hoc Networks.* PhD thesis, Technische Universität München, Germany, October 2003.

[9] Nirupama Bulusu, John Heidemann, and Deborah Estrin. Gps-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, October 2000.

[10] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[11] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat monitoring: application driver for wireless communications technology. *SIGCOMM Computer Communication Review*, 31(2 supplement):20–41, 2001.

[12] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, and Tim Finin. Toward Distributed Service Discovery in Pervasive Computing Environments. *IEEE Transactions on Mobile Computing*, 5(2):97–112, February 2006.

[13] M. Chatterjee, S.K. Das, and D. Turgut. WCA: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks)*, 5(2):193–204, April 2002.

[14] Christian Frank and Holger Karl. Consistency challenges of service discovery in mobile ad hoc networks. In *International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 105–114, New York, USA, 2004. ACM Press.

[15] Lewis Girod, Vladimir Bychkovskiy, Jeremy Elson, and Deborah Estrin. Locating tiny sensors in time and space: A case study. In *International Conference on Computer Design (ICCD)*, pages 214–219, Washington, DC, USA, 2002. IEEE Computer Society.

[16] S. Helal, N. Desai, V. Verma, and Choonhwa Lee. Konark - a service discovery and delivery protocol for ad-hoc networks. *Wireless Communications and Networking*, 3:2107–2113 vol.3, 2003.

[17] Ahmed Helmy. *Resource Management in Wireless Networking*, volume 16, chapter Efficient Resource Discovery in Wireless AdHoc Networks: Contacts Do Help. Springer, 2005.

[18] Todd D. Hodes, Steven E. Czerwinski, Ben Y. Zhao, Anthony D. Joseph, and Randy H. Katz. An architecture for secure wide-area service discovery. *Wireless Networks*, 8(2/3):213–230, 2002.

[19] Wen Hu, Nirupama Bulusu, and Sanjay Jha. A communication paradigm for hybrid sensor/actuator networks*. *International Journal of Wireless Information Networks*, 12(1):47–59, January 2005.

[20] D. Johnson, D. Maltz, and J. Broch. *Ad Hoc Networking*, chapter DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks, pages 139–172. Addison-Wesley, 2001.

[21] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353, pages 153–181. Kluwer Academic Publishers, 1996.

[22] Michael Klein, Birgitta Konig-Ries, and Philipp Obreiter. Lanes - a lightweight overlay for service discovery in mobile ad hoc networks. Technical Report 2003-6, University of Karlsruhe, 2003.

[23] Ulas C. Kozat and Leandros Tassiulas. Service discovery in mobile ad hoc networks: An overall perspective on architectural choices and network layer support issues. *Ad Hoc Networks*, 2(1):23–44, June 2003.

[24] Vincent Lenders, Martin May, and Bernhard Plattner. Service discovery in mobile ad hoc networks: A field theoretic approach. *Pervasive and Mobile Computing*, 1:343–370, 2005.

[25] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[26] R. S. Marin-Perianu, J. Scholten, P. J. M. Havinga, and P. H. Hartel. Energy-efficient cluster-based service discovery in wireless sensor networks. Technical Report TR-CTIT-06-43, Enschede, June 2006.

[27] A. McDonald and T. Znati. A mobility-based framework for adaptive clustering in wireless ad hoc networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 17:1466–1486, August 1999.

[28] Vivek P. Mhatre, Catherine Rosenberg, Daniel Kofman, Ravi Mazumdar, and Ness Shroff. A minimum cost heterogeneous sensor network with a lifetime constraint. *IEEE Transactions on Mobile Computing*, 4(1):4–15, 2005.

[29] R. S. Marin Perianu, J. Scholten, P. J. M. Havinga, and P. H. Hartel. Performance evaluation of a cluster-based service discovery protocol for heterogeneous wireless sensor networks. Technical Report TR-CTIT-06-61, Enschede, October 2006.

[30] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 90–100, Los Alamitos, CA, USA, 1999. IEEE Computer Society.

[31] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *SIGCOMM*, pages 234–244. ACM press, 1994.

[32] Ricky Robinson and Jadwiga Indulska. Superstring: A scalable service discovery protocol for the wide area pervasive environment. In *International Conference on Networks (ICON)*, pages 699–704. IEEE Computer Society, September 2003.

[33] Dietrich Stoyan, Wilfrid S. Kendall, and Joseph Mecke. *Stochastic Geometry and its Applications.* John Wiley and Sons, 1995.

[34] A. Varga. The OMNeT++ discrete event simulation system. In *European Simulation Multiconference (ESM)*, Prague, Czech Republic, June 2001.

[35] Alex Varshavsky, Bradley Reid, and Eyal de Lara. A cross-layer approach to service discovery and selection in manets. In *Mobile Adhoc and Sensor Systems Conference (MASS).* IEEE Computer Society, November 2005.