Performance Evaluation of a Cluster-Based Service Discovery Protocol for Heterogeneous Wireless Sensor Networks

Raluca Marin-Perianu, Hans Scholten, Paul Havinga and Pieter Hartel
University of Twente, Enschede, The Netherlands
Email: {raluca.marinperianu, hans.scholten, paul.havinga, pieter.hartel}@utwente.nl

***Abstract*— This paper evaluates the performance in terms of resource consumption of a service discovery protocol proposed for heterogeneous Wireless Sensor Networks (WSNs). The protocol is based on a clustering structure, which facilitates the construction of a distributed directory. Nodes with higher capabilities are chosen to be clusterheads. We measure the energy and memory costs depending on the capabilities and the roles of the nodes in the clustering structure, both during the maintenance of the structure and discovery of services. We show that the service discovery protocol succeeds in alleviating the resource-lean sensor nodes of heavy tasks, while delegating more consuming duties to the more powerful nodes.**

I. Introduction

Existing service discovery protocols typically consider that devices are not constrained in terms of available resources. Centralized solutions assume the existence of a powerful server capable of storing all the service descriptions in the network. Distributed approaches employ extensive caching mechanisms or complex overlay structures that facilitate scalable search. However, service discovery for Wireless Sensor Networks (WSNs) cannot afford to overlook the resource-awareness issue.

In our previous work [12] we propose a service discovery protocol for a heterogeneous WSN that builds a distributed directory on top of an clustered structure. The two main objectives are: (1) to maximize the network lifetime by minimizing the global energy consumption and (2) to reduce the memory and energy consumption of the weakest devices in the network. We showed that the prevention of global reconfiguration of the structure and the avoidance of network flooding lead to the desired goal of global energy efficiency.

The contribution of this paper is to address the second problem by investigating how the service discovery protocol succeeds in alleviating the resource-lean sensor nodes of heavy tasks, while delegating more consuming duties to the more powerful nodes in the network. The analysis focuses on memory and energy consumption, depending on the capabilities of the nodes and their roles in the clustering structure. We give a comparison with the clustering structure offered by DMAC [2], which shows the consequences of the chosen cluster algorithm on the resource consumption during both the maintenance and discovery phases.

Our targeted environment for service discovery is a heterogeneous WSN, with a variety of platforms including both static and mobile devices. Sensor nodes attached to walls, desks, chairs or other static objects are part of the static network. Body-area sensors, sensor nodes embedded in key-rings and other devices carried by people form the dynamic network. In addition, sensor nodes can be also attached to more powerful devices such as computers, that wish to be part of the wireless network.

The paper is organized as follows: Section II presents related work regarding service discovery protocols. The overlay clustering structure and the service discovery protocol are presented in Section III. Section IV evaluates the combined solution through simulations. Section V presents a summary and future work.

II. Related work

Service discovery protocols rely on either centralized [14], [16] or distributed architectures. The distributed solutions typically fall in one of the following categories:

- *Peer-to-peer caching* – Protocols designed for ad hoc networks often use peer-to-peer caching of service advertisements in the local registry [6], [5]. A service request message is first checked against the local registry and then it is broadcast or multicast in the network. Regardless of their capabilities, nodes are equally loaded.
- *Hierarchical structures* – DNS-like hierarchical structures [7], [11] enable network scaling to a large number of nodes. Parents store information about their children and queries pass up through the hierarchy until a possible match is encountered. However, global hierarchies are too heavy to maintain in a WSN environment.
- *Distributed hash table* – DHT approaches [1], [15] provide an efficient and scalable index lookup mechanism. However, this approach generates considerable network traffic and a high maintenance overhead, so it is less suitable for the WSN environment.
- *Cluster based* – This category comprises the protocols that select a set of nodes to be part of the distributed directory [9], [12]. Service discovery messages travel among these nodes and therefore, they are more loaded than the other nodes in the network.

Our aim is to analyze how the load is distributed among the nodes for a resource-aware, cluster-based service discovery protocol [12]. To the best of our knowledge, none of the

existing service discovery protocols gives an evaluation of the memory and energy consumption differentiated on the capabilities or the roles played by the nodes in the overlay structure.

III. Description of algorithms

This section briefly describes the clustering and service discovery algorithms. A more detailed and formal description is given in [12].

A. Clustering algorithm

Each node is assigned 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 the node is for the clusterhead role. These weights are assumed to be unique, as the node hardware identifier may be used to break ties.

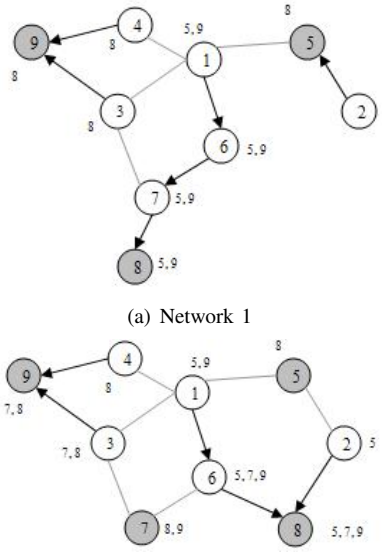
1) Cluster setup: The algorithm constructs a set of trees, based on local knowledge of neighboring nodes. The protocol works as follows:

- Nodes that have the highest capability grades among their neighbors declare themselves clusterheads and broadcast a *SetRoot* message announcing their roles.
- The remaining nodes choose as parent the neighbor 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.

The clustering protocol builds an independent set of root nodes based on 1-hop neighborhood knowledge, while avoiding the chain reaction phenomenon (see Section IV-A). In theory, this 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 [12].

Figure 1(a) shows an example network of nine nodes, where each node is assigned a capability grade. Based on these capability grades, the nodes organize into three trees (clusters). The parent-child relationship is indicated by arrows. The dashed lines connect neighboring nodes that are in different trees.

2) Knowledge of adjacent clusters: The identity of the root node is propagated in the cluster down to the leaf nodes via the broadcast message *SetRoot*. Thus, the message also reaches nodes from adjacent clusters, which store the adjacent root identity. The information is then propagated up in the tree until it reaches the root node, by using a message which we term *UpdateInfo*. Through this message, nodes learn which are the clusters adjacent to their sub-trees and the next hops on the paths leading to their clusterheads. In particular, the root nodes find out about all the adjacent clusters. Figure 1(a) shows the lists of adjacent clusters kept by each node in the network.



(a) Network 1

(b) Network 2

Fig. 1. Example of two clustered networks.

3) Maintenance in face of topology changes: Nodes adjust their cluster membership following topology changes:

- A node discovers a new neighbor with a higher capability grade than its current parent. The node then selects that neighbor 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 neighborhood.

The knowledge of adjacent clusters changes according to the changes of the clustering structure.

B. Service discovery protocol

The service discovery protocol uses the clustering structure to maintain a distributed directory of service descriptions. Nodes register their services with their parents and thus every node in the hierarchy maintains information on the services offered by the nodes in its sub-tree. The root nodes have a complete view of the services in their clusters. The service registration process is integrated with the construction and maintenance of the clustering structure, by using the same message *UpdateInfo*.

During the service discovery phase, the discovery message travels among the nodes part of the distributed directory. Suppose a node in the network generates a service discovery request *ServDisc*. The request is first checked against the local registrations. If 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 service discovery request message and it does not find any match in the local registry, the *ServDisc* 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, by using the local knowledge of adjacent clusters.

IV. Performance evaluation

We evaluate the load distribution property of the clustering algorithm and the service discovery protocol. Depending on their capabilities and the roles in the clustering structure, nodes are exposed to different levels of energy and memory consumption. We show that our solution achieves a low overhead for resource-lean sensor nodes, while the more powerful nodes are entrusted heavier tasks. Our analysis includes a comparison with the DMAC protocol [2], used as a clustering alternative for the distributed directory structure. We choose DMAC as a viable clustering choice for our service discovery protocol due to its simplicity and good performance results [3], that make it suitable for sensor environments.

In the following, we first give a brief description of DMAC. Secondly, we list some of the previous results regarding the performance of our solution. Thirdly, we introduce the general setting for both static and dynamic simulation experiments. Finally, we present the simulation results, including a performance evaluation of the service discovery protocol running on both clustering structures under the same topological conditions. In the following, we use the notation *C4SD* (Clustering for Service Discovery) for our clustering algorithm. We denote the number of nodes in the network as N.

A. DMAC Clustering Algorithm

DMAC constructs the clusters based on the same idea of assigning unique weights 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 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 neighbors. The role decision of a node is dependent on the decisions of the neighbors 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*. We showed that for a distributed directory composed of clusterhead nodes, the chain reaction leads to high overhead for maintaining consistent service registries [12].

B. Previous results

We mention some of our previous results [12] regarding the performance of the clustering algorithm and the service discovery protocol, together with a comparison with DMAC:

- For C4SD, the expected number of root nodes converges to an asymptotic value when $N \to \infty$.
- The average cluster height for C4SD is below 2, and in 95% of the cases it is below or equal to 3.
- The cluster density experienced by DMAC is higher than the cluster density of C4SD.
- C4SD has a lower maintenance and discovery cost than DMAC, and consequently lower energy consumption.

These results will be further used in the performance evaluation to justify some of the present results.

C. Simulation settings

For our experiments we use the OMNeT++ [17] simulation environment. We generate a random network, by placing N nodes uniformly distributed on a square area of size $a \times a$, where $a = 500m$. Each node chooses a capability grade from a uniform distribution. Static nodes have higher capability grades than mobile nodes. 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. In the experiments where we do not state otherwise, we take $r = 0.2a$.

We consider the expected number of neighbors (or node degree) to be a measure for the network density, calculated as [12]:

$$D(N, r) = N\frac{\pi r^2}{a^2} \tag{1}$$

For measuring the load depending on the capability grades, we simulate a network of 100 nodes, with an expected node degree $D(100, 100) = 12.56$. 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 class. The results are shown depending on the capability group each node belongs. The focus of our dynamic simulations is the overhead induced by the *UpdateInfo* and *ServDisc* messages.

In the dynamic experiments from Section IV-F, we use a simplified version of the random waypoint [8] as mobility model. We assume that the mobile nodes represent people walking, so the dynamics of the network is moderate. At the beginning, nodes are randomly placed on the simulation area, where they stay for a specified period of time. After this time expires, they choose a random destination and start moving toward that destination. Nodes are moving at $1m/s$, the approximate speed of a walking person. Upon arrival at the destination, nodes pause for 30 seconds before restarting the process. Due to the initialization problems that characterize the random waypoint mobility model [4], 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. In our simulations, 50% of the nodes are moving and they belong to the first five capability groups. The rest of the nodes are static. The focus of our dynamic simulations is the overhead induced by the *UpdateInfo* and *ServDisc* messages.

D. Role percentage

Depending on their role in the clustering structure, nodes have different responsibilities. In contrast to the leaf nodes, parents and clusterheads keep registries of services for the nodes below in hierarchy. Moreover, roots and parents also inherit the knowledge of adjacent clusters form their children. Therefore, we are interested in measuring how many leaves, parents and roots are produced by the clustering algorithm.

We represent in Figure 2 the percentage of clusterheads, parents and leaves to the total number of nodes in the network, depending on the expected node degree (see Eq. 1). We experiment with three different transmission ranges, $r = 0.1a$,

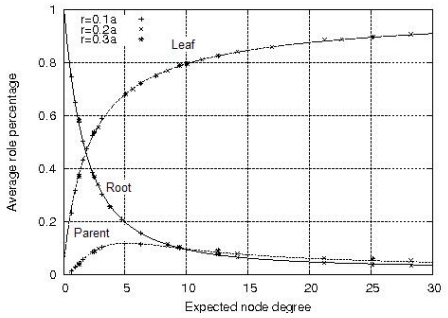


Fig. 2. Average percentage of roots, parents and leaves.

$r = 0.2a$ and $r = 0.3a$. We notice that regardless of the chosen transmission range, the points follow the same curve. Consequently, the average role percentage is only a function of the expected number of neighbors. For sparse networks, where $D(N, r) \rightarrow 0$, the number of root nodes equals the number of nodes in the network. For a network with 100 nodes, with average node degree higher than 10, the parent and root nodes represent less than 20% from the total number of nodes. For dense networks, the percentage of clusterheads and parent nodes gets close to zero, while the percentage of leaf nodes grows asymptotically to 1.

E. Memory consumption

Memory is consumed in maintaining (1) the knowledge of adjacent clusters and (2) the service registrations. We analyze each of these situations in turn.

1) Knowledge of adjacent clusters: We first investigate how many entries for adjacent clusters a table has on average. Figure 3 shows the number of entries in the table, depending on the role of the node in the clustering structure. Regardless of the assigned roles, the size of the table grows proportionally with the network density.

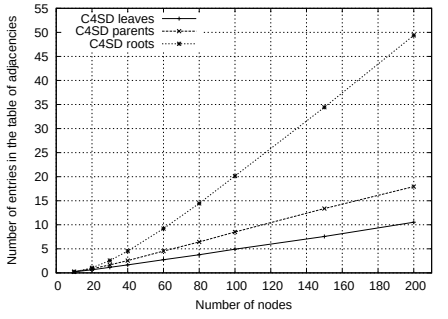


Fig. 3. Average number of entries in the table of adjacent clusters, depending on the role in the clustering structure.

We represent in Figure 4 the average number of entries depending on the capability grades, where $N = 100$. From Section IV-D we know that the parent and root nodes represent less than 20% from the total number of nodes. We notice that the first eight capability groups (mostly leaf nodes) have approximately the same number of entries in the table of adjacencies. The last capability group, comprised of mainly root nodes, is the most loaded.

Figure 4 also shows a comparison with DMAC. As the cluster density experienced by DMAC is higher than the cluster density of C4SD (see Section IV-B), the number of adjacent clusters that each node maintains is consequently higher. In our case, the first eight groups of nodes need to allocate on average between 20% and 30% more memory for DMAC than for C4SD.

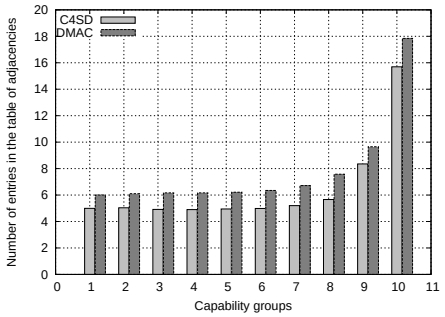


Fig. 4. Average number of entries in the table of adjacent clusters, depending on the capability group.

An entry in the table has two fields: the address of the adjacent root node and the address of the neighbor, which is the first hop on the path to the root node. The node addresses are dependent on either the platform (hardware IDs) or the network stack (MAC, routing and transport). Typical address sizes range from 2 to 6 bytes, which means that a table entry occupies from 4 to 12 bytes.

2) Service registrations: We are interested to know how large are the service registries maintained by the parent and root nodes. The answer depends on the size of service descriptions, the repartition and the level of redundancy in the service offer. Therefore, a realistic evaluation is application-oriented.

Nevertheless, we give here an example where every node offers exactly one service and for every service there is exactly one service provider. We assume that the size of the service descriptions is equal for all the services. The service registry at node v is therefore proportional to the number of nodes in the tree rooted at v.

The average number of nodes below in hierarchy is represented in Figure 5, depending on the role of the node. We notice that for sparse networks (10 nodes), the root nodes are less loaded than parent nodes, since most of the clusters incorporate only one node, the root. However, for denser networks, the root nodes are significantly more loaded compared to the parent nodes, since root nodes typically inherit the service descriptions from the parents. The average load increases linearly with the network density.

We represent in Figure 6 the average number nodes in the tree depending on the capability groups, for both C4SD and DMAC. We notice that both protocols relieve the low capa-

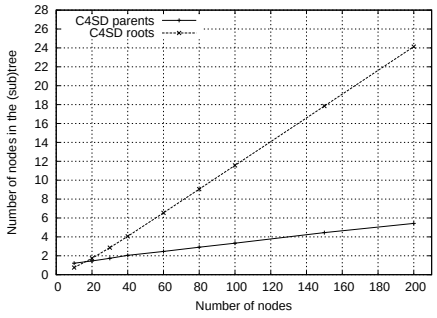


Fig. 5. Average number of nodes below in hierarchy, depending on the role of the node.

bility groups of any load regarding the service registrations. As DMAC imposes a maximum cluster height of one, service registrations are not duplicated among directory nodes. On the contrary, the services registered at the parents in C4SD are also registered at the root nodes. Therefore, the load on the C4SD nodes in the last capability group, typically root nodes, exceed DMAC with approximate 50%. C4SD is more demanding in terms of memory consumption for the nodes with high capability grades, but succeeds in releasing the low capability groups of memory and energy consumption more than DMAC, as shown in Sections IV-E.1 and IV-F.

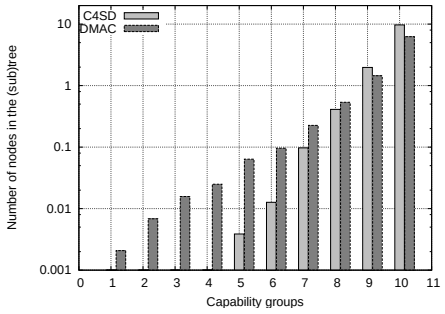


Fig. 6. Average number of nodes below in hierarchy, depending on the capability group.

We mention that a service description can vary for a simple service number of one byte length, to a more complex binary XML description, where it can occupy up to several hundred bytes [13].

F. Energy consumption

Next, we investigate how the roles and capability grades influence the energy consumption during the maintenance and discovery phases in dynamic experiments. Since the most energy is spent during communication, we assume that the energy consumption is proportional to the number of messages exchanged.

1) Maintenance overhead: We first represent the number of *UpdateInfo* messages sent and received on average by a node in 1000 simulation seconds, depending on the role in the clustering structure. We notice from Figure 7 that the number of maintenance messages exchanged by the root nodes grows proportionally with the network density. Even if after a certain network density the number of root nodes remains the same, they have to maintain the consistency of the service registries for an increased number of mobile nodes. On the contrary, for leaves, the number of messages slowly decreases after a certain network density. The reason is that for dense networks, it is likely that the information received from a new neighbor does not change the knowledge of adjacent clusters, so it is not propagated further in the tree.

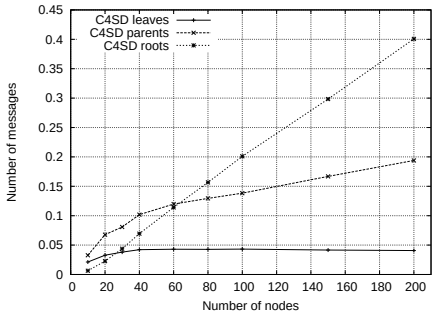


Fig. 7. Average number of maintenance messages depending on the roles in the clustering structure.

In Figure 8 we present the number of maintenance messages for both C4SD and DMAC, 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 mobile nodes have a higher maintenance overhead, due to frequent parent re-selections. The last two groups of nodes have the highest overhead.

We notice that 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).

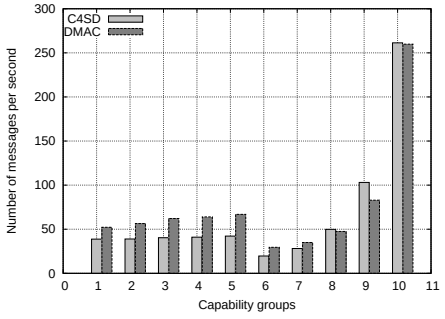


Fig. 8. Average number of maintenance messages per groups of capabilities.

2) Discovery cost: In the following, we are interested in the division of the discovery overhead. During 1000 seconds of simulation time we issue 10 random service requests, with a delay of 100 seconds. In Figure 9 we present the average

number of messages exchanged during one service discovery phase, depending on the role in the clustering structure. We notice that the load for parent and leaf nodes slowly decreases for dense networks, while the evolution of the load on root nodes indicate an asymptotically bounded behavior. The reason is that the service discovery messages travel among the clusterhead nodes, which remain constant in number after a certain network density (see Section IV-B).

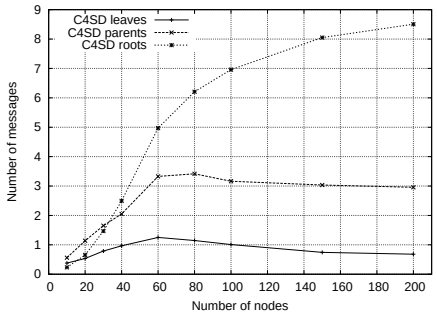


Fig. 9. Average number of service discovery messages depending on the roles in the clustering structure.

Figure 10 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.

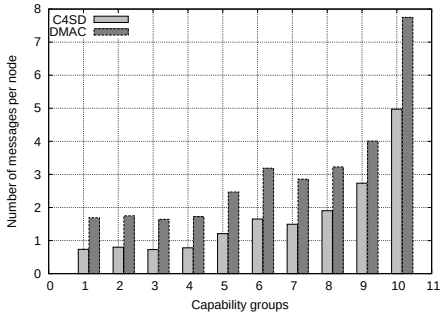


Fig. 10. Average number of service discovery messages per groups of capabilities.

The energy consumption for transmitting and receiving packets depends both on the hardware platform and the MAC protocol. Typical values for the average energy consumed by a node range between 1 and 12 mW [10].

V. Conclusions

This paper presents a performance evaluation of a service discovery protocol based on clustering, with a focus on the network heterogeneity. The protocol relies on a clustering structure, which is a solution for the scalability problem as it achieves restricted reconfiguration, distributed knowledge and it avoids network flooding. The clusters are set up depending on the available resources and dynamics of nodes. The algorithm exploits the heterogeneous nature of the network by assigning high tree levels to more powerful nodes. The simulation results show that the protocol delegates more workload to powerful nodes while relieving the weak and mobile devices. The performance comparison with DMAC shows that (1) DMAC spends more memory for maintaining the knowledge of adjacent clusters, (2) DMAC is more energy consuming, especially for weak devices, (3) the performance of C4SD comes at the cost of utilizing more memory for service registrations at powerful devices.

Our plan for the future is to avoid the possibility of overloading the root and parent nodes with service registrations. The idea is that nodes that reach their memory limit can decrease the capability grade. This dynamic adjustment of capabilities depending on the context is expected to improve the resource-awareness and to provide even better performance.

References

[1] Magdalena Balazinska, Hari Balakrishnan, and David Karger. INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Pervasive'02*, pages 195–210, August 2002.

[2] Stefano Basagni. Distributed clustering for ad hoc networks. In *ISPAN '99*, pages 310–315. IEEE Computer Society, 1999.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] K. Langendoen and G. Halkes. *Embedded Systems Handbook*, chapter Energy-Efficient Medium Access Control. CRC press, 2005.

[11] Choonhwa Lee and Sumi Helal. A multi-tier ubiquitous service discovery protocol for mobile clients. In *SPECTS'03*, pages 701–711, Montréal, Canada, 2003.

[12] 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.

[13] Raluca Marin-Perianu, Hans Scholten, and Paul Havinga. CODE: A description language for wireless collaborating objects. In *ISSNIP'05*, pages 169–174. IEEE Computer Society Press, December 2005.

[14] Sun Microsystems. Jini architecture specification version 2.0, June 2003.

[15] Ricky Robinson and Jadwiga Indulska. Superstring: A scalable service discovery protocol for the wide area pervasive environment. In *ICON'03*, pages 699–704, Sydney, September 2003.

[16] V. Sundramoorthy, J. Scholten, P. G. Jansen, and P. H. Hartel. Service discovery at home. In *ICICS/PCM'03*, pages 1929–1933. IEEE Computer Society Press, December 2003.

[17] A. Varga. The omnet++ discrete event simulation system. In *ESM'01*, Prague, Czech Republic, June 2001.