An Algorithm for Distributing and Retrieving Information in Sensor Networks*

Hugo Miranda[1], Simone Leggio[2], Luís Rodrigues[1], and Kimmo Raatikainen[2]

[1] University of Lisbon
Portugal
{hmiranda,ler}@di.fc.ul.pt
[2] University of Helsinki
Finland
{simone.leggio,Kimmo.Raatikainen}@cs.helsinki.fi

Abstract. Replication of data items among different nodes of a wireless infrastructureless network may be an efficient technique to increase data availability and improve access latency. This paper proposes a novel algorithm to distribute data items among nodes in these networks. The goal of the algorithm is to deploy the replicas of the data items in a way such that they are sufficiently distant from each other to prevent excessive redundancy but, simultaneously, they remain close enough to each participant, such that data retrieval can be performed using a small number of messages. In most scenarios, our approach allows any node to retrieve a data item from a nearby node. The paper describes the algorithm and provides its performance evaluation for several different network configurations.
Keywords: Data replication, Ad hoc networks, Sensor Networks

1 Introduction

Information management in wireless infrastructureless (ad-hoc) networks (with or without node movement) is not a straightforward task. The inherently distributed nature of the environment, and the dynamic characteristics of both network topology and medium connectivity, are a challenge for the efficient handling of data. There are several policies that can be followed when dealing with information management in these networks. In particular, one must first choose whether to follow a centralised or decentralised approach.

Usually, ad-hoc networks are formed by peer nodes, with limited capabilities, so it is unpractical to elect a single node to act as a repository for the data needed by the other nodes; such node would require and consume significantly more resources than other nodes. Moreover, this approach introduces a single point of failure; this is unacceptable given that node failures are an integral part

* This work was partially supported by the MiNEMA programme of the European Science Foundation and by FCT project P-SON, POSC/EIA/60941/2004 through FCT and FEDER.

of ad-hoc networks (failures may happen due to voluntary departures, crashes, or simply due to medium impairments). A decentralised approach is, therefore, strongly favoured.

In a decentralised approach, the data items are spread among all the nodes of the network. The data dissemination algorithm should balance the need to provide data replication (to cope with failures) with the need to avoid excessive data redundancy (as nodes may have limited storage capability). Furthermore, data items should be distributed as evenly as possible among all the nodes that form the network, avoiding clustering of information in sub-areas; an even dissemination of data items should leverage lower access latency to any item from any node in the network, i.e, whenever a data item is requested by a node S, the distance to the node that provides the reply should be approximately the same, regardless of the location of S. Naturally, the actual distance depends on multiple parameters, such as the number of nodes in the network, the amount of memory made available at each node, and the number of data items. Finally, since in wireless networks both bandwidth and battery power are precious resources, the algorithm should also minimise the amount of signalling data.

In this paper, we address the problem of data dissemination for wireless infrastructureless networks where nodes do not move, are unaware of their geographical location and have limited resources. This scenario is similar to one of those typically associated with wireless sensor networks. The paper introduces an algorithm for efficiently spreading and retrieving data items. The implementation of these operations is orchestrated such that a limited number of messages is required for retrieving any data item from the network, independently of the addition or removal of nodes.

Our algorithm can be used to implement a shared white board to support indirect communication among nodes of the sensor network. Any node would publish or read information from the shared white board. Readers would not be required to know, a priori, the publisher of the information and would be able to find the desired data in their vicinity without being required to maintain a copy of every item.

The rest of the paper is divided as follows. Section 2 describes the algorithm in detail, illustrating its working principles and parameters. Section 3 evaluates and comments the algorithm, by presenting the results of extensive simulations. Section 4 presents related work, and finally, Section 5 summarises the main issues raised in this paper and presents pointers for future work.

2 A Dissemination and Query Algorithm

In this paper, we assume that nodes have limited resources and are only able to store a limited number of data items, representing a fraction of all the data items advertised. Nodes always try to keep this space filled, occupying all free space before beginning to overwrite other entries. The entries to be overwritten are randomly selected. The system does not require the space at all nodes to be of the same size. Each data item is composed of a key and a value with

application dependent semantics. An expiration time and a version number field are available to update or remove data items. The goal of the algorithm is to provide an adequate distribution of data items so that each node is able to find a significant proportion of the total items in its memory or in one of its closest neighbours.

We assume that the dissemination of new or updated data item is triggered by one of the participants, who is said to be the owner of the item. To ensure that at least one copy of each item exists, nodes do not replace the items they own with items advertised by other nodes. It is assumed that owned items are stored in a separate region of the memory space of the devices so that the space available for storing third-party items is kept constant.

The algorithm provides two distinct operations: data dissemination and data retrieval. These operations are implemented using three types of messages. In the dissemination process, nodes cooperate to provide an adequate distribution of the replicas of new or updated versions of data items. *Dissemination messages* are broadcast following an algorithm to be described later. The retrieval process is triggered by applications requesting the value associated with a key. The algorithm first verifies if the value is stored locally and if it is not, it broadcasts *query messages*. Nodes listening to the request and storing the corresponding value send a *reply message* to the source of the query.

Messages share a common header that describes the type of message (dissemination, query or reply) and a *time to live* (TTL) field, decremented by each node that forwards the message. Like in many other algorithms for ad hoc networks, messages are propagated using a store-and-forward approach: a node receives a message, decides if the message must be forwarded or not and, before forwarding the message, it may update one or more fields in the message. Query messages accumulate the path to be used by a reply in a field, here identified as *route stack*.

2.1 Broadcast Algorithm

We use the Pampa [1] message broadcast algorithm to propagate dissemination and query messages. The advantage of Pampa is that it requires a considerably smaller number of nodes to forward messages than a conventional flooding operation. In Pampa, nodes more distant to the previous forwarder broadcast the message earlier. Furthermore, Pampa prevents nodes whose expected additional coverage would be smaller (locally evaluated by counting the number of retransmissions listened) from retransmitting. This is achieved by having each node to hold the retransmission of a message by a period of time proportional to the reception power of the first retransmission listened.

Due to the store-and-forward nature of the algorithm, Pampa is not used as a black-box. Instead, every time Pampa decides to forward or drop a message, it first issues a callback: the data dissemination algorithm may then update the contents of the message, or overrule Pampa's decision on the fate of the message. In particular the algorithm may opt to force the transmission of a message that would be otherwise dropped by Pampa.

The query and dissemination algorithms could be used in conjunction with other broadcast mechanisms (e.g. [2, 3]). The reason to use Pampa is that it offers useful information about the relative position in space of the nodes in the neighbourhood of a given sender. Namely, knowledge of the following Pampa properties are required to understand the dissemination algorithm:

- When a message is broadcast, the first node that becomes ready to retransmit the message is the node that is more distant from the source.
- If a node notices that n neighbours have already forwarded a message, it suggests to drop the message (i.e., it decides not to retransmit the message). The value of n is a configuration parameter. We use Pampa with n set to 2. Elsewhere [1], we have shown that $n = 2$ provides delivery ratios similar to higher values on networks where nodes do not move and with densities ranging from 625m^2/node to 40000m^2/node. The advantage of a smaller value is that it requires a lower number of messages per dissemination operation.

2.2 Dissemination Process

Dissemination of data items is triggered by the source node with the broadcast of a *dissemination* message using Pampa. The dissemination algorithm is presented in Fig. 1.

Function REGISTER (Fig. 1, l. 1-6), shows that after the reception of a call to advertise some data, the node prepares a registration message and uses the Pampa broadcast algorithm to disseminate it. In dissemination messages, the *time from storage* (TFS) field indicates the distance (in number of hops) from the sender to the closest node that is known to have stored the items. Therefore, the source node sets the TFS field to zero to indicate that the records are stored locally.

Based on the Pampa dissemination strategy, each node receiving a message places it on hold for a period of time proportional to the distance to the source (l. 7-12). The holding period suggested by Pampa may be biased to leverage the storage of the replicas among neighbouring nodes (l. 11). During the hold period (l. 13-14), the node calculates *mintfs*, given by the lowest value of the TFS from the original message and of all retransmissions listened. At the end of the hold period, *mintfs* will indicate the distance in hops to the closest node(s) that stored a copy of the item.

When the hold period expires (l. 15-24), the node decides on the fate of the message and to store or not the corresponding data item. The decision takes as input the following parameters: i) the output of the Pampa's algorithm, that takes into account the distance from the source and the number of retransmissions listened; and ii) the *mintfs* computed during the hold period.

The algorithm tries to keep a replica of each item at most *DbC* (*Distance Between Copies*) hops away of every node in the system. *DbC* is a configuration parameter that should be set according to the network conditions.[3] The data

[3] Section 3 shows how different values of DbC affect the performance of the algorithm.

```
1: procedure REGISTER(data)                                   ▷ At the sender
2:     CACHE.ADDLOCAL(data)
3:     tfs← 0; ttl←GETNETWORKDIAMETER
4:     mid←CREATEMSGID
5:     msg←(REG,mid,data,tfs)
6:     PAMPA.BROADCAST(msg,ttl)

7: function REGRECEIVED(mid,data,tfs, ttl,delay)              ▷ At all other nodes
8:     data_mid ←data; ttl_mid ←ttl
9:     mintfs_mid ←tfs
10:    if tfs=DbC then                                        ▷ Bias Pampa delay
11:        delay← (1+CACHE.OCCUPRATIO)×delay
12:    return delay

13: procedure DUPRECEIVED(mid,tfs)
14:    mintfs_mid ← min{mintfs_mid,tfs}

15: procedure TIMEREXPIRED(mid,forward)▷ Decision to forward and store the data
16:    if mintfs_mid=DbC then
17:        CACHE.ADD(data_mid)
18:        forward←true
19:        tfs← 0
20:    else
21:        tfs←mintfs_mid + 1
22:    if forward ∧ ttl_mid > 0 then
23:        msg←(REG,mid,data_mid,tfs)
24:        PAMPA.BROADCAST(msg,ttl_mid − 1)
```

Fig. 1. Data dissemination algorithm

item is stored if, at the end of the hold period, the value of *mintfs* remains equal to *DbC*. Note that if some other node in the vicinity decides to store the data item, it will retransmit the message with TFS set to zero and, therefore, *mintfs* will be reset accordingly. The message is forwarded if the data item was stored or if the output of Pampa's algorithm suggests its dissemination. When a message is forwarded, the TFS will be set accordingly to the decision to store the data item: zero if the item was stored or *mintfs+1* otherwise.

A node will store the data item if it is the first node in its own vicinity, to decide to forward a message received with TFS=*DbC*. Depending on the deployment of the nodes and of the location of the sources of the dissemination messages, some nodes may have their probability of being required to store items increased. We compensate those effects by making Pampa's waiting time not only dependent on the distance to the source but also on the available storage space of the node when the probability of the node being required to store the data item is high (that is, when the TFS of the first received message is equal to *DbC*). In our algorithm, a node that has few memory will volunteer later to

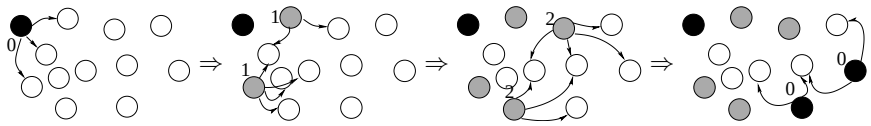

Fig. 2. Progress in dissemination of an item

potentially store (and retransmit) an item than a node that has more memory (l. 11). This is achieved by multiplying Pampa's delay by a factor b proportional to the occupancy ratio of the node's memory (in our experiments we have set b to vary in the range $[1, 2]$).

Fig. 2 exemplifies a dissemination when DbC is set to 2. Nodes that forwarded the message are represented in gray and nodes that stored and forwarded the item in black. Three copies of the item were stored. The first at the source node and the remaining because the nodes had a *mintfs* of two. For clarity, only a subset of the message receptions are represented. The numbers close to each node represent the value of the TFS field advertised in the message.

2.3 Retrieval Process

To retrieve some value from the network, a node begins by looking for the key in its local storage (Fig. 3, l. 4-11). If the value is not found, the node starts an expanding ring search for the value, again using Pampa to reduce the number of nodes forwarding the query. The Time-To-Live (TTL) of the first round is defined using an adaptive function shown in lines 15 to 19 of Fig. 3. Assuming that the dissemination algorithm achieved an adequate distribution of the items, the distance (in number of hops) from the source of the query to any item should be approximately the same and will depend mostly of the number of neighbours of the node and their storage space. The adaptive function weights past experiences of the node. For the q^{th} query performed by the node, the TTL of the first round is given by the recursive function $qTTL(q) = k \times qTTL(q-1) + (1-k) \times dR(q-1)$ where $0 < k < 1$ is a constant to leverage the relevance of the most recent result, $dR(q-1)$ is the distance (in hops) at which the reply to the previous query was found and $qTTL(q-1)$ is the TTL of the previous query. In our experiments, we adopted $qTTL(1) = 1$ and $k = 0.5$.

A node receiving a query message (l. 20-27), and that does not find the value locally, will use Pampa and the value of the TTL field to decide if the message should be retransmitted. Before retransmitting, the node appends the address of the previous hop to the *route stack* field, in a route construction process similar to the route discovery algorithm in some source routing protocols for MANETs (e.g. [4]). If a node receiving a query message finds the requested key, it does not enter the holding period of Pampa. Instead, it sends a point to point reply to the source of the query. The reply message follows the path constructed in the *routeStack* field of the query.

```
1: procedure INIT                                   ▷ Set default values and constants
2:     qTTL← 1
3:     K← 0.5

4: procedure QUERY(key)                             ▷ At the source of the query
5:     if CACHE.HASVAL(key) then
6:         value←CACHE.GETVAL(key)
7:         DELIVER(key,value)
8:     else
9:         routeStack←{}; mid←CREATEMSGID; key_mid ←key
10:        msg_mid ←(QRY,mid,key)
11:        PAMPA.BROADCAST(msg_mid,routeStack,qTTL)

12: procedure NOREPLYTIMEOUT(mid)
13:    ttl←GETNETWORKDIAMETER; routeStack←{}
14:    PAMPA.BROADCAST(msg_mid,routeStack,ttl)

15: procedure REPLYRECEIVED(mid,value,ttl)
16:    qTTL← qTTL × k + (1 − k) × ttl               ▷ Update default value for ttl
17:    if ttl>1 then
18:        CACHE.ADD(key_mid,value)
19:    DELIVER(key_mid,value)

20: procedure QRYRECEIVED(mid, key, routeStack, ttl)   ▷ At other nodes
21:    if CACHE.HASVAL(key) then
22:        value←CACHE.GETVAL(key)
23:        msg←(RPLY,mid,value,ttl)
24:        SEND(routeStack,msg)
25:    else if ttl> 0 then
26:        routeStack←routeStack ⨁ localAddr
27:        PAMPA.FWD(msg,routeStack)          ▷ Message passed for Pampa decision
```

Fig. 3. Data retrieval algorithm

A reply found far away from the source of the query signals an ill distribution of the item (l. 15-19). Therefore, the node that issued the query stores the item locally if the reply was received from a node located more than one hop away. It should be noted that the algorithm is orthogonal to the underlying routing protocol. The reply message is unreliably sent to the network without verifying if the destination is still in range. No provision is taken to limit the number of replies sent to the node. Therefore, there is a reasonable probability that at least one of the routes constructed during the query propagation remains valid until the reply is delivered. This assumption is similar to that of many reactive routing protocols for MANETs [4, 5] for route discovery.

2.4 Robustness Considerations

A premise of ad-hoc networks is the permanent addition and (possibly temporary) disconnection of nodes. Because it does not rely on a membership protocol, the only impact of node removal in the system is a reduction of the number of replicas of some of the items and the corresponding performance decrease. The addition of a node to the network also does not imply any message exchange. The node will begin to fill its storage space as soon as dissemination messages start to be listened by the node.

3 Simulation Results

We have implemented a prototype of our algorithm in the *ns-2* network simulator v. 2.28. The simulated network is composed of 100 nodes uniformly disposed over a region with 1500mx500m. Each node stores at most 10 data items advertised by other nodes. The simulated network is an IEEE 802.11 at 2Mb/s. Network interfaces have a range of 250m using the Free Space propagation model. No routing protocol was used.

Runs are executed for 900s of simulated time. The traffic on each run consists of 100 dissemination and 400 queries. Each node disseminates one data item with 300 bytes in a time instant selected uniformly between 0 and 400s. All measurements have been taken in number of messages and number of data items stored at the nodes. Therefore, the size of the data item is only relevant for estimating the traffic generated at the network. Simulations do not consider expiration of items since they could artificially improve the performance of the protocol by freeing additional resources on the nodes.

Queries start at 200s and are uniformly distributed until the 890s of simulated time. The nodes performing the queries and the queried items are selected using an uniform distribution. The simulation ensures that only advertised records can be queried so that the evaluation of the protocol does not become obfuscated by bogus queries.

No warm-up period is defined. All values presented below average 100 independent runs, combining different deployment and traffic files. Most of the evaluation uses two metrics. The "average distance of the replies" measures the distance (in number of hops) from the querying node to the source of the first reply received. The distance of a reply is 0 if the value is stored in the querying node. The "average number of transmissions per query" measures the total number of query and reply messages (initial transmissions and forwarding) performed by all nodes and divides it by the number of queries.

3.1 Sensitivity to Different Network Configurations

The performance of the algorithm is affected by the number of nodes in the neighbourhood of each node, the storage size at every node and the number of items advertised in the network. The effect of the variation of each of these parameters was evaluated individually, keeping the remaining according to the baseline configuration described in the beginning of the section.

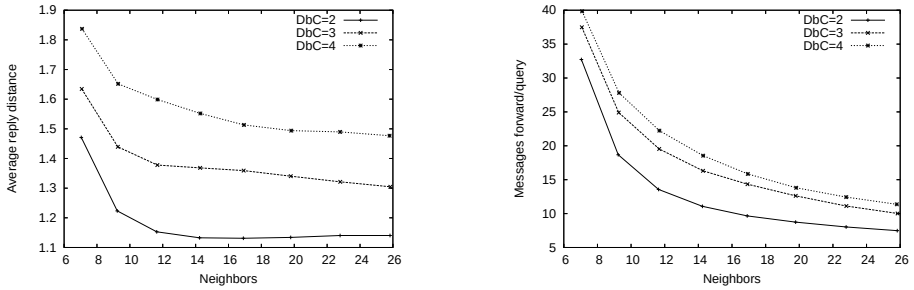

Fig. 4. Variation of network density

Network Density. To evaluate the behaviour of the algorithm in different densities of nodes, we have varied the number of neighbours by configuring the nodes with different transmission powers while keeping the size of the simulated space constant. The transmission power was set such that transmission ranges varied between 150 and 325 meters. The number of neighbours was estimated by averaging the number of nodes that received every broadcast message on each simulation with the same transmission range. The average reply distance and the ratio of messages per query are presented in Fig. 4.

An interesting aspect of the figure is the small gains in the distance of the replies as soon as the network density is sufficient to guarantee message propagation. We identify this as a positive feature of the algorithm: it confirms that the algorithm does not consume additional resources of the devices beyond those that are associated to the predefined value of the *DbC* configuration parameter. The algorithm preserves this storage space for other items.

Although the distance of the replies tends to stabilise with the grow of the network density, the number of messages continues to decay. We attribute most of this behaviour to the capabilities of the Pampa dissemination algorithm to adapt to different network densities, preventing redundant retransmissions from nodes which would not provide a significant contribution to the message propagation.

Cache Size. To evaluate the behaviour of the algorithm for nodes with constrained resources, we varied the number of items that can be kept by each node between 2 and 16. The number of items disseminated was kept constant and consistent with the baseline configuration. Fig. 5 shows that the average reply distances for *DbC* 2 and 3 approach as the resources available in the neighbourhood of each device decrease, suggesting that *DbC* play an important role in the performance of the algorithm, which is the target of further analysis below. Like before, results tend to stabilise when the storage space made available surpasses the requirements of the predefined DbC value.

Number of Items. The results for the different simulations varying the number of items advertised are presented in Fig. 6. The figure confirms that the selection of an adequate value for *DbC* affects the performance of the algorithm. In the case

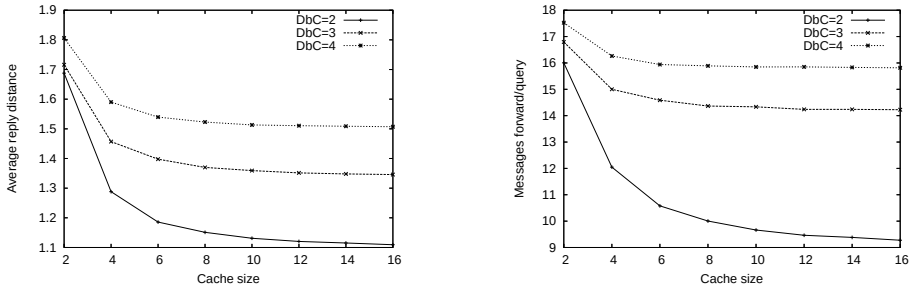

Fig. 5. Variation of cache size

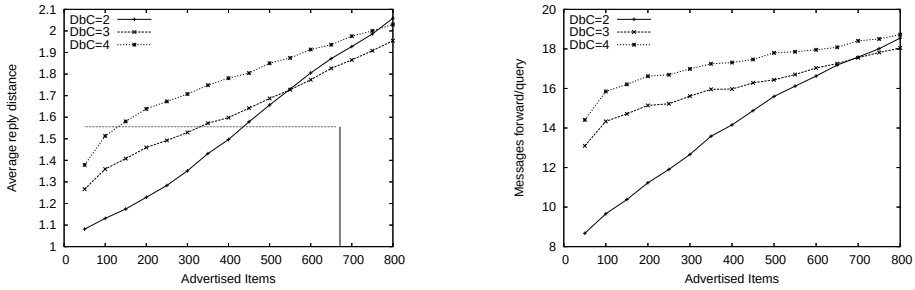

Fig. 6. Variation of the number of items

presented in the figure, the performance of *DbC* 2 degrades gracefully but becomes worst than *DbC* 3 when more than 550 items are advertised. The number of messages follows a pattern similar to the increase in the average distance of the replies, meaning that the adaptive function that defines the TTL of the first query message is adequately preventing the nodes from flooding the network to find a copy of the item, even in demanding conditions.

Analytical Evaluation. We analytically compare the variation of the number of items with a theoretical distribution of the items following our model. Assuming that there is sufficient storage space available, when DbC=n, all nodes should be able to find a copy of any data item at a distance not higher than $\frac{n+1}{2}r$ (where r is the transmission range of the devices) or $\lceil \frac{n+1}{2} \rceil$ hops. The rationale for DbC=2 is presented in Fig. 7 and can be similarly derived for other values. The figure shows nodes that forwarded the item in gray and nodes that forwarded and stored the item in black.

From the tests where the transmission range was varied, we know that the average number of nodes within a circle of 250m radius (the transmission range used in the baseline configuration) is 17.85. Therefore, it is possible to conclude that if the radius increases to 1.5 the original value, the number of nodes should become $1.5^2 \times 17.85 \approx 40.163$. Of these, 44,4% of the nodes require 1 hop to reach the centre node while the remaining 55.6% are 2 hops away. The average

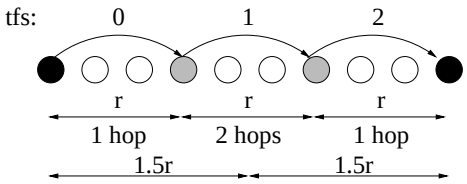

Fig. 7. Storage of a data item

distance of the replies should therefore be, in this configuration, 1.556. This value is plotted in a horizontal line in the left graphic of Fig. 6.

The storage capacity of the nodes in the 1.5 times the radius of the transmission range is given by $n \times \left(SS + \frac{i}{N}\right)$ where n and N are respectively the number of nodes in the region and in the simulation (recall that data items are uniformly advertised by all nodes and that nodes store the items advertised in a separate region of the cache), i is the number of items advertised and SS is the Storage Space made available at each node. Relevant for our study is to find the maximum number of items that can be stored in our baseline configuration with DbC=2, which corresponds to the solution of the equation $i = 40.163 \times \left(10 + \frac{i}{100}\right) \Rightarrow i \approx 671.21$. We define this value as the saturation point of our algorithm for the baseline configuration and when DbC=2.

The saturation point is relevant to define the theoretical limit of a configuration from the number of nodes, their cache sizes and the number of advertised items. The figure shows that the simulation results cross the expected distance of the replies at approximately 2/3 of the saturation point. Our interpretation of this result is that there is some undesirable redundancy of the data items stored that prevents other items from occupying their position. We attribute this behaviour to different factors:

- A fundamental assumption of the model above is that at every forwarding, there exists one node located precisely at distance r from the previous source. Although Pampa favours the forwarding of the messages by the nodes more distant to the source of the previous transmission, it is only possible to assume that the node will be located at some distance lower or equal to r. These smaller distances result in additional hops travelled by the dissemination and query messages and reduces the effective area that should be accounted in the estimation of the saturation point;
- The particularities that occur at the borders of the simulated space, where it is possible that some nodes remain an higher number of hops away from the data (in Fig. 7 consider for example that the rightmost node does not exist);
- Concurrency issues and anomalies in the uniform distribution of the nodes. Notice for example that the dissemination presented in Fig. 2 places two copies of an item close to each other because there is no link between the nodes.

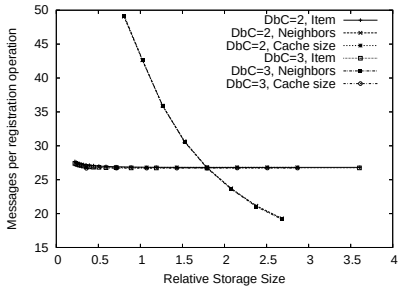

Fig. 8. Messages used per registration operation

It should be noted that even in demanding conditions, the algorithm is capable of providing an acceptable performance requiring a small number of messages and providing the replies within a reasonable number of hops.

3.2 Traffic for Dissemination

Figure 8 shows the number of messages per dissemination for different values of *DbC* and network configurations. The figure shows that the number of messages is independent of variations in the storage space or in the number of items advertised but, as expected, decreases with the network density. This confirms that Pampa adapts the number of transmitting nodes to the node density, reducing it for higher densities. The x axis of the graphic harmonises the different variations to the baseline configuration in a metric named Relative Storage Size (RSS) which counts the proportion of advertised items that can be stored in a node and in its 1-hop neighbourhood. RSS is given by $(\mathrm{nneigh}+1)\frac{\mathrm{SS}}{\mathrm{Items}}$ where *nneigh* is the average number of neighbours, *SS* the storage space at every node and *Items* the total number of advertised items.

We emphasise that the number of messages presented in Fig. 8 are those required by Pampa to flood the entire network and that this would be the number of messages required by queries if no early dissemination of the items was performed. A comparison with the number of messages presented in Figs. 4 to 6 shows that the additional number of messages spent on item dissemination is rapidly attenuated after a small number of queries. Furthermore, our algorithm provides the additional advantage of replicating the data items, making the network more robust against failures and departures of nodes.

4 Related Work

Distributing replicas of each data item is an effective way to achieve data availability and reduce latency in MANETs. However, the limited resources of the devices must be considered in the estimation of the number of replicas of each

data item to be stored. Most of the research in distributed storage follows some deterministic algorithm for deciding the location of the replicas.

In some cases, the goal is to store replicas of the data close to the nodes that are more likely to access it in the future. Estimation of future requirements is based for example on the history of previous accesses or by user preferences. The frequency of the accesses to the data items is used in [6] to define three deterministic algorithms for the allocation of replicas. Neighbourhood awareness is taken into account as well, by eliminating replica duplication among neighbouring nodes or group of nodes. The results show that neighbour awareness improves the accessibility of data items, at the expenses on more traffic in the network to maintain neighbourhood information. More recently [7], the same author investigated particular scenarios where exists a correlation between different data items and how the inference of these correlations could benefit caching policies.

In autonomous gossiping [8] the data items themselves try to identify other hosts which may be interested in the item, based on the data item's own profile and host's profile, advertised during the registration phase. This approach is in contrast to the traditional push model were data items are injected in the networks by the possessor nodes. Profiles are maintained in a distributed self-organising way, and updated using gossiping techniques. When data items arrive at a node, the autonomous gossiping algorithm is applied to decide what the data item will do, in an autonomous and self-organising fashion. Data items in a host decide whether continue to reside, migrate or replicate to another host.

The replication model presented in [9, 10] assumes the presence of a single data source in the ad-hoc network. In [9], queries are unicast, addressed to the data source, and if a node in the path has either the data cached locally or the path to a node that holds the queried item, a reply is immediately returned to the querying node; otherwise, the request is forwarded to the data source. In [10] the querying scheme is broadcast; a four-way handshake is implemented to prevent nodes from receiving more than one reply to a given issued query. We deem that such a scheme is useful when the target data item is large in size. If the advertised items are small, it may be faster to send the queried item directly. The performance of both algorithms depends of the location of the querying node and of the previous queries for the same item.

In Data-centric models for sensor networks, data is aggregated according to its type in well-known nodes in the network. A popular approach [11–13] defines a hash function that maps the type of each data item on a geographical coordinate, creating a Geographical Hash Table (GHT). All these approaches have in common the requirement that the devices are aware of their geographical location. Like in our algorithm, some of the proposals geographically distribute the replicas by sensors at different regions of the network in an effort to reduce the overhead of queries [14, 12, 15]. The majority of the algorithms benefits from the geographical information made available to the nodes to partition the sensor network area, keeping a replica at each partition.

A study on the trade-off between the energy spent in the replication of data items and on queries in sensor networks was presented in [16]. However, the

study can not be applied to our model because the authors assume a random deployment of the replicas and an expanding-ring search query. This is in contrast with our work where the goal is to evenly distribute as much replicas as necessary to cover the entire network. As simulations have shown, in most of the cases, our algorithm is able to prevent the energy demanding expanding ring search operation by resolving the query at the first round.

Non-uniform information granularity [17] proposes to trade off information accuracy by the energy spent in the dissemination by decreasing the accuracy of the information as it becomes more distant from the source. The gains are obtained by decreasing the number of messages forwarded with the distance to the source. In contrast with our proposal, authors assume that sensors are powerful enough to store all the events advertised.

Instead of using knowledge gathered from past experiences or user preferences, our algorithm relies on the location of the nodes to place a replica close to each node. However, our algorithm relies on Pampa to increase the distance between copies, instead of requiring nodes to be aware of their location. The uniform distribution of the algorithm makes it adequate for networks composed of resource constrained devices, in particular, without a location device and with limited memory. Access patterns more favourable to the algorithm are those where it is not possible to infer future accesses to the data from past experiences or when the queries to a data item are uniformly distributed over the network.

5 Conclusions

This paper has presented an algorithm for retrieving and distributing information in ad-hoc networks. The algorithm is fully distributed as it does not assume the presence of only a few data sources in the ad-hoc network; each node, instead, can advertise own data items. The main goal of the algorithm is ensuring an even geographical distribution of the disseminated data items, so that requests for a given data item are satisfied by some nodes close to the source of the query.

This goal is obtained by combining a mixture of different techniques. Broadcast messages are used to leverage the algorithm from the presence of an underlying routing protocol. This approach enables using neighbourhood information without requiring nodes to run an expensive membership protocol. Simulation results show that the algorithm achieves a fair dissemination of items throughout the network. Furthermore, simulation results have also shown that the initial cost of data item dissemination is rapidly absorbed by the substantial reduction in the number of messages required for retrieving a value from the network.

There are different interesting applications for the algorithm. It can be used to implement a shared white board in resource constrained devices like those in sensor networks. A possible subject of future work can be to tailor the algorithm to specific applications. An interesting application would be a naming service, such that nodes advertise a combination of user name and contact address in the ad-hoc network. Nodes could query for a given user name (the key used in queries) and retrieve the contact address as value for the requested key.

References

1. Miranda, H., Leggio, S., Rodrigues, L., Raatikainen, K.: A power-aware broadcasting algorithm. In: Proc. of The 17th An. IEEE Int'l Symp. on Personal, Indoor and Mobile Radio Communications (PIMRC'06). (2006)
2. Haas, Z., Halpern, J., Li, L.: Gossip-based ad hoc routing. In: Proc. 21st Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2002). Volume 3. (2002) 1707–1716
3. Tseng, Y.C., Ni, S.Y., Chen, Y.S., Sheu, J.P.: The broadcast storm problem in a mobile ad hoc network. Wireless Networks **8**(2/3) (2002) 153–167
4. Johnson, D.B., Maltz, D.A.: Dynamic Source Routing in Ad Hoc Wireless Networks. In: Mobile Computing. Kluwer Academic Publishers (1996) 153–181
5. Perkins, C.E., Royer, E.M.: Ad-hoc on-demand distance vector routing. In: Proc. of the 2nd W. on Mobile Computing Systems and Applications. (1999) 90–100
6. Hara, T.: Effective replica allocation in ad hoc networks for improving data accessibility. In: Proc. of the 20th Ann. Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2001). Volume 3. (2001) 1568–1576
7. Hara, T., Murakami, N., Nishio, S.: Replica allocation for correlated data items in ad hoc sensor networks. SIGMOD Rec. **33**(1) (2004) 38–43
8. Datta, A., Quarteroni, S., Aberer, K.: Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in mobile ad-hoc networks. In Bouzeghoub, M., Goble, C., Kashyap, V., et al., eds.: Proc. of the Int'l Conf. on Semantics of a Networked World, IC-SNW'04. Volume 3226 / 2004 of Lecture Notes in Computer Science. (2004)
9. Yin, L., Cao, G.: Supporting cooperative caching in ad hoc networks. IEEE Transactions on Mobile Computing **5**(1) (2006) 77–89
10. Lim, S., Lee, W., Cao, G., Das, C.: A novel caching scheme for improving internet-based mobile ad hoc networks performance. Elsevier Journal on Ad-Hoc Networks **4**(2) (2006) 225–239
11. Ratnasamy, S., Karp, B., Shenker, S., Estrin, D., Govindan, R., Yin, L., Yu, F.: Data-centric storage in sensornets with GHT, a geographic hash table. Mobile networks and applications **8**(4) (2003) 427–442
12. Ghose, A., Grossklags, J., Chuang, J.: Resilient data-centric storage in wireless sensor networks. IEEE Distributed Systems Online (2003)
13. Sarkar, R., Zhu, X., Gao, J.: Double rulings for information brokerage in sensor networks. In: MobiCom '06: Proc. of the 12th Ann. Int'l Conf. on Mobile computing and networking. (2006) 286–297
14. Li, J., Jannotti, J., Couto, D.S.J.D., Karger, D.R., Morris, R.: A scalable location service for geographic ad hoc routing. In: Proc. of the 6th Ann. Int'l Conf. on Mobile computing and networking (MobiCom '00), ACM Press (2000) 120–130
15. Liu, D., Stojmenovic, I., Jia, X.: A scalable quorum based location service in ad hoc and sensor networks. In: Proc. of the 3rd IEEE Int'l Conf. on Mobile Ad-hoc and Sensor Systems. (2006)
16. Krishnamachari, B., Ahn, J.: Optimizing data replication for expanding ring-based queries in wireless sensor networks. In: Proc. of the 4th Int'l Symp. on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt'06). (2006) 1–10
17. Tilak, S., Murphy, A., Heinzelman, W.: Non-uniform information dissemination for sensor networks. In: Proc. of the 11th IEEE Int'l Conf. on Network Protocols (ICNP'03). (2003) 295–304