Using Failure Models for Controlling Data Avaialability in Wireless Sensor Networks

Mirko Montanari, Riccardo Crepaldi, Indranil Gupta, Robin H. Kravets
{mmontan2,rcrepal2,indy,rhk}@illinois.edu

Abstract— **This paper presents Pirrus, a replica management system that addresses the problem of providing data availability on a wireless sensor network. Pirrus uses probabilistic failure models (e.g., derived from environmental conditions and estimation of available energy) to adaptively create and maintain a number of replicas of the data. The replica management is formulated as an energy optimization problem, then solved with a greedy heuristic that uses only information gathered from neighbors. Intuitively, Pirrus trades off the energy saved by limiting the number of replicas when the network health is good to extend the lifetime when more replicas are needed. Our simulation results show how the solution provided by Pirrus achieves good performance with a sustainable computational cost. Compared to the performance of a fixed number of replicas, Pirrus extends the network lifetime of 25%.**

I. Introduction

Traditionally, replica management systems have been designed to operate over relatively reliable machines, where the probability of failure of one machine is low enough to allow a small fixed number of replicas (e.g., two or three) to provide very high data availability. In such systems, the probability of failure of each machine or device is assumed to be constant in time. For example, the reliability of hard drives is provided by the manufacturer as a Mean Time Between Failures (MTBF) and is considered to be constant in time. However, this is only an approximation of the real failure behavior.

In sensor networks, these assumptions hold even less, since nodes can fail for many different reasons, such as depletion of the available battery energy, hardware defects in the node and harsh environmental conditions in which the node operates. Moreover, this failure rate is not constant: at the beginning of the network life-cycle, batteries are charged and nodes have just been tested, so the failure probability of a node is low. As the network continues, nodes consume energy and are more likely to run out of energy. Additionally, the presence of particularly harsh environmental conditions, such as very high or very low temperatures, can affect the failure rate of nodes.

To cope with this risk, one possible solution is to periodically send data to a base station with more reliable hardware, which can safely store data avoiding loss. However, several factors might make it impossible or too expensive to reach a base station: the presence of intermittent network connectivity or the high energy cost of continuous communication favors a solution where data is locally stored on the sensor nodes.

The solution to store data on the nodes has many advantages: there is no need of constant communication and it is resilient to network partitioning. Later, when data is requested, the user can collect it from the node using queries or data mules. However, the possibility of node failure puts data at risk, even when data location and retrieval are reliable operations. This is because when a node loses its ability to communicate, either for a fault or battery depletion, the data it stores is lost. Therefore, replicating data on multiple nodes represents a method for reducing the possibility of data losses. In this context, the problem of replica placement is critical.

Data replication has been proposed to prevent such data loss in distributed environments. Traditional replication approaches use a fixed number of replicas defined *a-priori* during network deployment [1]. Given the dynamic failure potential of the nodes in the network, these static solutions are either too costly or cannot provide enough availability during the network lifetime. If the number of replicas selected is too small, when the nodes become more prone to failure over time, data availability decreases. However, if there are too many replicas in the system, energy consumption increases unnecessarily. This is particularly undesirable toward the beginning of the network lifetime when most nodes are very reliable.

To cope with the variability in the network, replica management should be adaptive and adjust the number and location of replicas to provide the same degree of data availability. However, designing a system that dynamically adapts to the network conditions is not trivial. An approach that bases its decisions on complete knowledge of the network does not scale. Moreover, disseminating information about node health to the entire network is not energy-efficient. The challenge is to find a replica placement strategy that uses local information to compute globally energy-efficient solution.

The contribution of our research is threefold. First, we formalize the problem of providing energy-efficient replica placement for data availability as an energy optimization problem. Given that the optimal solution requires global knowledge of network state, we next define a heuristic approach that only requires local information. Finally, we adapt the heuristic approach to design a novel distributed protocol, called Pirrus, which adopts a greedy approach and provides near-optimal results under varying conditions.

The key idea in Pirrus is that each node only uses local information about their n-hop neighbors to solve the problem of space availability. This local approach scales to large networks without suffering from excessive overhead or computational complexity. Since the environment has a strong influence on node reliability, Pirrus is designed to be flexible and allows users to define appropriate probabilistic failure models. In comparison to static approaches that do not consider the reliability of the nodes in the network, Pirrus adapts to changing network conditions. When data availability is a priority, results show that Pirrus is able to extend network lifetime compared to other static approaches.

The rest of the paper presents the replica placement problem and describes our approach to solve it. For ease of presentation, the system model and assumptions will be discussed as the paper develops. Section II presents an overview of the problem of adaptive data replication. Section III formalizes the replica placement problem and describes the Pirrus protocol and its replica placement strategy. Finally, section V presents an evaluation of Pirrus in comparison to other current approaches.

II. Motivation and related work

Distributed databases are well-studied in wireless sensor networks. Systems such as COUGAR [2] and TinyDB [3] provide a database-like view of a sensor network enabling data storage on nodes and retrieval of only the data that the user or the application is interested in. However, wireless nodes are unreliable devices. Factors such as hardware defects, energy depletion and considerations about the environment where the network is deployed affect their reliability. For example, if the network is deployed on a volcano, a drastic temperature increase could be a signal that some violent event is about to happen. In this case, nodes closer to the top of the volcano will be more prone to failure. In a distributed database, such a failure will cause an irreversible loss of data.

In these environments, the goal is to provide reliable storage over an unreliable network. The reliability level of the storage is a parameter of the problem and it is quantified as a the probability p_{req} at least one copy of the data surviving. The user can tune this data availability requirement to the needs of the application: high availability can be requested for important data, while lower availability can be requested for non-critical information. The high unreliability of motes is likely to make the probability of successfully accessing a node and its data lower than the requested p_{req} over time.

Replication can be used to provide data availability. In the context of sensor networks, the biggest challenge is the placement and management of replicas. To achieve the required availability, the system needs to provide a sufficient number of replicas K such that the probability of finding at least one of them when needed is greater than p_{req}. In this work, we focus on replica placement and management strategies for data availability and we do not consider data access: in several data collection deployments, the frequency at which data is collected by the user is much lower than the rate at which data is gathered from the environment.

Several reasons could cause data to be unavailable. Two of these reasons are the failure of the node where the data is stored and the impossibility of reaching it because of a lack of network connectivity. For the purposes of this paper, we focus on the problem of node failure: we assume that data is not available if the node on which it is stored stops operating. This assumption is motivated by the fact that, even in networks with unreliable connectivity, data can be collected using data mules that, by moving physically close to the node, can directly communicate with it. To focus on the formulation of the problem, we also assume that communication is always successful. This assumption can be relaxed delegating the communication reliability the MAC layer.

The factors that affect nodes' reliability can be captured by a probabilistic model of failure to estimate the reliability of a node. Failure models based on temperature, humidity and other working conditions have been defined for industrial equipment [4]. Even if, to the extent of our knowledge, none of these models have been defined for sensor nodes, similar reliability engineering techniques could be applied in the sensor network context. Probabilistic reliability models could be obtained from theoretical considerations about the system or from experimental data.

Using these models, a probability of failure $p_j(t)$ can be defined for each node. These failure probabilities are assumed to be independent (i.e., the variations in the failure probability of a node do not affect the failure probability of the rest of the network). This probability, that can change over time, is abbreviated as p_j in the rest of this work.

When the availability requirements are integrated with the failure probability specified by the failure model, replica placement needs to satisfy the following condition:

$$\prod_{j}^{K} p_j < 1 - p_{req}, \tag{1}$$

If the assumption of independence is not true the approach is generalizable, but the equation 1 has to be adjusted properly.

Creating, maintaining and deleting replicas have a communication cost, which depends on their location (i.e., the distance between the replica and the original data source). A replication strategy, to be efficient, needs to consider these costs when choosing the replica locations.

The next section introduces related work in the area of data replication in distributed systems and sensor networks.

Related Work

The general problem of data replication has been studied intensively. In traditional distributed systems, there are two approaches to replica management: reactive replication and proactive replication. Systems that use reactive replication, such [5], react to node failures and replicate data to provide the required availability level. On the other hand, proactive systems [6] create more replicas than strictly required to provide robustness to node failures. The dynamic nature of sensor networks and its scarcity of resources (e.g., storage space and energy) make the use of a reactive approach that limits the number of replicas and creates them in response to changes in the network an efficient approach.

An interesting system for the management of replicas in traditional P2P is Total Recall [7]. In this system, the user is able to assign a probabilistic availability level to each file in the distributed storage and the system automatically manages replicas according to the expected availability of each host. The context of wireless sensor networks is different from the traditional P2P environment: protocols and solutions for sensor networks need to explicitly optimize energy consumption and integrating energy optimization in Total Recall is complex.

Replica placement strategies have been previously proposed in sensor networks. However, goals of these strategies are providing load balancing, energy savings and efficient data access. None of the previous work in this area selects a replica placement to provide a constant level of data availability.

A method for automatically caching frequently accessed data in intermediate nodes during routing was proposed by Yin and Cao [8]. The proposed approach allows nodes in the path to the final destination to answer queries using cached information, if it is recent. Tang et al. [9] propose a distributed algorithm for the optimal allocation of replicas in an adhoc network. This work analyzes replica placement as an optimization problem: each node needs to access some data contained in other nodes. Data closer to the requesting nodes is less expensive to obtain. Thus, the problem becomes finding the optimal placement of replicas to minimize the access cost and satisfy the constraints on the space available in each node. Further work [10], [11], [12] proposed the use of heuristics to allocate replicas close to the hosts that frequently access data. The goal of these replica systems is not to increase data availability, but to improve the access time to the data. For this reason, they do not offer the user the ability to provide quantitative levels of data availability.

The use of data replicas has also been proposed in the context of data-centric storage [13]. In this work, data is replicated at multiple locations to provide load balancing and increase network lifetime. Experiments show that this method provides graceful performance degradation in the presence of faults, but no quantitative availability levels are provided.

The concept of Quality of Service is introduced in [1]. This work enhances the Geographic Hash Table storage [14] by adding the concept of explicit QoS guarantees for each class of data. The QoS level is provided by the programmer as the number of required replicas. This approach requires to set a fixed number of replicas at the beginning of the network lifetime and does not adapt to the changing conditions of the network.

III. Replica Placement

The problem of replica placement has many angles that makes its formulation a challenge. The goal of replica placement is to find a suitable set of nodes on which data replicas can be placed, such that the availability requirements are met, while minimizing the energy spent by the system. Communication is one of the most expensive operations in WSNs, hence the optimization goal can be expressed as the minimization of the number of messages used to manage replicas.

This section introduces our modeling of the problem as the minimization of the number of messages exchanged for replica management. The formulation addresses the cost of creating, maintaining and removing replicas. In general, each node could have different costs for replica management. For the purpose of simplicity, the formulation considers only three different costs: creating, maintaining and removing a replica. However, our approach is able to handle different costs for every node.

In our system model, each node stores a set of environmental samples taken during a predefined period T. After an initial set up time, the size of the collected data does not change: new data replaces the oldest samples contained in the set. This model represents a system where environmental samples are collected continuously, but only the most recent ones need to be kept for analysis.

To obtain a model of the replica placement problem in this context, it is necessary to analyze the trade-off involved in replica creation and maintenance: each replica increases data availability, but at the same time consumes storage memory and requires periodic communication to be kept up-to-date. For example, consider a situation where a certain level of availability can be obtained by either replicating data on a few unreliable nodes located near the data source node or by replicating them on a reliable node further away from the source, requiring multi-hop communication to reach the node.

To capture the complexity of this trade-off, we present a model of the general replica placement problem. The problem is first modeled assuming a network where no replicas are present before placement. Then, the formulation is refined to take into account a network where previous replica placement has been performed. Both formulations assume complete and omniscient knowledge of the network, which we will later relax in Section IV.

Replica placement can be modeled as a binary optimization problem. The number of messages exchanged for replica maintenance is the value to minimize, subject to constraints over data availability and storage space available on the nodes. The binary decision variables represent the presence or absence of a data replica on a particular node. Formally, if we consider a system with N nodes, the binary variables can be represented in a matrix X of size $N \times N$ where each variable x_{ij} is equal to 1 if j stores a replica for node i:

In the system model that we consider, each node stores a data set of size w_i, where i refers to the node that generated and maintains that replica. Each node j has a limited amount of memory in which it can store replicas. It reserves a part of its storage space for its own data, and provides W_j bytes to store replicas for the system. Formalizing this concept, it is necessary to ensure that the sum of the space occupied by the replicas on a node j is smaller than the available space, W_j:

$$\sum_{i=1}^{N} x_{ij} \cdot w_i < W_j. \tag{2}$$

The communication cost of creating a new replica can be expressed as the number of messages that are needed to send the data to the new replica location. For a single-hop network, this number is expressed as the constant k_a. When more than a single hop is needed to reach the new location, the total number of messages is proportional to the number of hops, indicated with $d(i, j)$, between the source node i and the destination node j. When all possible sources and destinations are considered, the total number of messages is expressed as the sum of all messages sent by every node to create their replicas, as follows:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} k_a \cdot x_{ij} \cdot d(i, j). \tag{3}$$

According to this formula, for each replica created in the system, a number of messages k_a is sent from the source node and are forwarded over $d(i, j)$ hops to reach the node where the replica is stored.

The choice of the nodes on which replicas are placed needs to take into account storage space constraints and availability requirements. The problem can now be expressed as an optimization problem: the function to minimize is the number of messages exchanged in the network, subject to the constraints of availability of each data and storage space on each node (shown in the same order in the equation). The problem can be expressed as following:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} k_a \cdot x_{ij} \cdot d(i,j), \tag{4}$$

$$\begin{array}{ll}
\prod_{j|x_{ij}=1} p_j < 1 - p_{req}^{(i)} & \forall i \\
\sum_{i=1}^{N} x_{ij} \cdot w_i < W_j & \forall j \\
x_{ii} = 1 & \forall i
\end{array}$$

The solution to this problem defines the optimal configuration for replica placement.

The first constraint of this problem is non-linear. However we can transform it into a linear problem by considering the logarithm on each of the two terms. The logarithm of the products can be changed into a sum of logarithms. Formally, the operation can be expressed as following:

$$\log \prod_{j=1}^{N} p_j^{x_{ij}} < \log\left(1 - p_{req}^{(j)}\right), \tag{5}$$

this allows us to rewrite the first constraint of (4) as follows:

$$\sum_{j=1}^{N} \left(x_{ij} \cdot \log p_j\right) < \log\left(1 - p_{req}^{(j)}\right). \tag{6}$$

This optimization problem can compute the optimal placement of replicas starting from a system configuration where no replicas are present. During normal operation of the system, replicas already present may not be sufficient to satisfy the requirements: changes in failure probability of the nodes over time can make the provided availability to be less than the required one. The optimization algorithm needs to evaluate the options to keep the current set of replicas, create new ones or remove them. It is then possible to define a new optimization problem, taking into account not only the creation cost, but also the maintenance and deletion cost, since this operation also requires the exchange of messages between the peers. Let $\hat{X}$ be the matrix representing the current replica placement, whose elements $\hat{x}_{ij}$ are defined as for the matrix X. Let D and A be two matrices of size $N \times N$. The elements of D, d_{ij}, are binary variables representing the deletion of replica i from node j, while the elements of A, a_{ij}, represents the addition of a replica.

We also define two additional constants: k_m, the cost of maintaining an existing replica, and k_d, the cost of sending an invalidation command for an existing replica. These two costs are defined as the number of messages required to perform the replica management operations in each period T.

The optimization problem can now be reformulated to take into account the creation and the deletion of replicas: k_m messages are sent for every replica that was present in the previous solution and it is not eliminated in this period, while k_d messages are sent to eliminate a replica. When creating a new replica, k_a messages still need to be sent. All of these costs are proportional to the number of hops between the source node i and the destination node j.

The new formulation of the problem is:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} \left(k_m \cdot (\hat{x}_{ij} - e_{ij}) + k_a \cdot a_{ij} + k_d \cdot e_{ij}\right) \cdot d(i,j) \tag{7}$$

$$\begin{array}{ll}
\sum_{j=1}^{N} \left((\hat{x}_{ij} - e_{ij} + a_{ij}) \cdot \log p_j\right) < \log\left(1 - p_{req}^{(i)}\right) & \forall i \\
\sum_{i=1}^{N} (\hat{x}_{ij} - e_{ij} + a_{ij}) \cdot w_i < W_j & \forall j \\
a_{ij} \leq 1 - \hat{x}_{ij} & \forall i,j \\
e_{ij} \leq \hat{x}_{ij} & \forall i,j \\
a_{ij} + d_{ij} \leq 1 & \forall i,j \\
a_{ii} = 0, e_{ii} = 0, x_{ii} = 1 & \forall i.
\end{array}$$

As in (4), the function to optimize (line 1) is the number of messages sent in the system. The constraints on availability and storage space (line 2 and 3) are still present: the changes reflect only the addition of the variables representing replica creation and deletion. With these new variables, the correct formulation of the new optimization problem requires the definition of additional constraints between $a_{ij}, \hat{x}_{ij}$ and e_{ij} to restrict the feasible solutions of the problem only to valid solutions for our model. First (line 4), a replica can be added only if it was not present in the previous solution (i.e., it is not possible to add twice a replicas on the same node). Similarly, it is not possible to remove replicas that were not present before on the node (line 5). Another constraint has been added to avoid solutions that contain a replica that is both added and removed within the same optimization (line 6). Finally, it is not possible to add or eliminate replicas on the data source node.

This global formulation of the problem provides a global optimal for replica placement. The problem is a binary optimization problem with a number of variables that grows as the square of the number of nodes, making the problem intractable. Moreover, an implementation in a centralized solution requires to gather information about the states of every node in the network: this is a very expensive operation in sensor networks.

To go toward the definition of a distributed solution, this global problem can be partially rewritten as a set of individual minimization problems, one for each node i. In this local formulation, each node optimizes the number of messages that it sends to create and maintain its own replicas. Binary variables similar to the ones used in the global problem are defined: $\hat{x}_j$ represents the presence of a replica of the data on node j, while e_j and a_j represents the elimination and addition of a replica on the node.

These optimization problems are not completely independent: every time that a replica is allocated on a node j, the space available to store other replicas is decreased. This fact is expressed by considering a set of variables $\hat{x}_j^{(k)}, d_j^{(k)}, a_j^{(k)}$ that represent the allocation and removal of replicas on node j by the other optimization problems.

The problem for a node i can now be expressed as follows:

$$\min \sum_{j=1}^{N} (k_m \cdot (\hat{x}_j - e_j) + k_a \cdot a_j + k_d \cdot e_j) \cdot d(i,j) \tag{8}$$

$$\begin{array}{ll}
\sum_{j=1}^{N} (\hat{x}_j - e_j + a_j) \log p_j < \log(1 - p_{req,i}) & \\
(\hat{x}_j - e_j + a_j) w_j < W_j - \sum_{k=1}^{N} (\hat{x}_j^{(k)} - d_j^{(k)} + a_j^{(k)}) w_k & \forall j \\
a_j \leq 1 - \hat{x}_j & \forall j \\
e_j \leq \hat{x}_j & \forall j \\
a_j + e_j \leq 1 & \forall j \\
a_i = 0, e_i = 0, x_i = 1 & .
\end{array}$$

In this formulation, the optimization function (line 1) is simplified to take into account only messages sent by a specific node. The availability constraint (line 2) maintains its previous structure. The storage space (line 3) constraints needs to consider less than W_j as space available for replica allocation: it needs to take into account the solutions to the other sub-problems (represented by $\hat{x}_j^k$, $d(k)_j$, $a(k)_j$) that allocated replicas on j. The rest of the constraints have the same purpose as the respective constraints in 7.

Optimal placement of replicas requires knowledge of the current state of the system. The available capacity for replicas on each destination node, and hence the optimal replica placement for each of the single sub-problems, depends on the solution of all other sub-problems in the network. Additionally, each node needs to know the failure probability of all other nodes in the system. The distributed nature of a sensor network and its dynamic conditions makes the problem of maintaining an up-to-date view of the entire system at each of the nodes too expensive in terms of communication, and so not scalable.

A local approximated solution to this problem can be defined using only local information. The rest of the section defines this approximated solution.

Local replica placement

To obtain a distributed and scalable solution, it is necessary to limit the amount of information that each node requires to compute replica placement. Our first observation is that the cost of creating and maintaining replicas is proportional to their distance from the source node. If the failure probability is geographically uniform (i.e., there are no areas of the network that are less failure-prone than others), a set of replicas placed on local nodes is less expensive than a set of replicas created far from the data source. When the failure probability is not uniform, a set of replicas placed on nodes far from the data source could represent a better solution to the problem. However, disseminating up-to-date information about failure probability of distant nodes results in significant overhead in sensor networks. Investigating a solution where only information collected from neighbor nodes is used to compute a solution represents an interesting approach to limit the network overhead and, at the same time, providing enough information to compute a feasible replica placement.

Another problem of optimal replica placement, formalized in the previous section, is its computational complexity. Even if the optimal solution can be computed solving the separate sub-problems, the interaction between them does not change the overall complexity: to obtain a scalable solution, it is necessary to limit the interaction between nodes. An efficient replica placement strategy needs to locally and independently compute a new placement without the need of complex interaction with other nodes. An efficient heuristic replica placement approach can consider the available space in each node j to be only the space not previously allocated to other replicas. This solution will not provide an optimal replica placement, however it greatly reduces the communication between nodes.

The formulation of this heuristic is similar to the formulation of the optimal problem (8). However, the constraint over the available space on each node considers only $W_{rem,j}$, the space not assigned to other replicas on node j. The constraint can be rewritten as follows:

$$(\hat{x}_j + a_j - d_j) w_j < W_{rem,j} \quad \forall j. \tag{9}$$

The computation of this heuristic placement requires solving a binary optimization problem. However, running a binary optimization algorithm in a wireless node is too computationally expensive. For this reason, the next section introduces a greedy solution implementable on a sensor node. The next section introduces an approach for finding an approximate solution to the problem based only on local information.

IV. Pirrus

Pirrus is a distributed approach for replica placement. In this approach, replicas are placed according to the variable probability of failure of nodes. The system continuously monitors the state of the replicas and reacts to failures or to decreasing availability. This approach requires a small cost for replica maintenance when the network is reliable, while providing availability even in adverse conditions. Pirrus is designed to be a framework where the user can embed a model of failure formulated for the specific deployment. The use of a more precise model results in a more effective protocol behavior.

The next section describes the distributed strategy used by Pirrus for replica placement. Section IV-B describes the protocol used to implement it in a WSN.

A. Greedy replica placement

The new storage space constraint defined in the local replica placement problem (Eq 9) specifies a set of nodes where enough space for the placement of a new replica is available. Considering only the nodes that satisfy this constraint, the replica placement problem becomes similar to a knapsack problem [15]: minimization of the number of messages subject to the constraint of availability requirements (representing the capacity of the knapsack problem).

For this reason, it is possible to use a well-known greedy strategy to provide a feasible solution to the replica placement problem. When expressed according to our problem formulation, the greedy strategy involves ordering the nodes with enough storage space according to the ratio between the gain that placing a replica on node j would provide, and its cost. The gain is expressed as the coefficient in the availability constraint, $\log p_j$, while the cost is expressed as the number of hops $d(i,j)$.

$$\frac{\log p_j}{d(i,j)}. \tag{10}$$

In conclusion, Pirrus is defined by the following set of actions:

1) Collect $< W_{rem,j},\, p_j >$ about nodes within h hops
2) Sort in descending order a list L of nodes with enough storage space according to the above greedy strategy.
3) Select the first q nodes from L such that:
   $\prod_{j=1}^{q} p_{L(q)} < 1 - p_{req} \forall i$
   where $p_{L(q)}$ is the probability of failure associated with the q-th element of the list.
4) Send requests for the allocation of replicas to the selected nodes
5) After receiving the allocation requests, allocate the replica on node j. If multiple requests are received, accept only the first one received.
6) If the allocation request has been dropped, choose a different node and start again from step 1.

B. Protocol definition

For the purpose of the protocol, each node is characterized by two parameters: its current *probability of failure* and its *available storage space*. To publish this information to the rest of the network, each node periodically broadcasts heartbeat messages to its neighbors. These messages are forwarded after increasing their hop-count field. The forwarding stops after a maximum number of hops, d_{max} is reached. Each heartbeat is characterized by a unique sequence number to avoid multiple retransmissions.

When nodes receive a heartbeat from node A they update their view of the network. Those nodes who are allocating replicas on A also update their data availability consequently. When the failure probability of data generated by a node falls below the required level of availability, the node applies the Pirrus strategy using the information received.

For each selected node, the source node A starts sending the data to replicate to it. A sends an INIT message to the selected node B that can reply with a positive acknowledgment or can refuse the replica if its available storage is too low. This could happen if different peers select the same node as a candidate for a replica or if the information about the available storage is not up-to-date. In this case, A skips node B and establishes a connection with the next selected node. The data transfer is then performed using a Selective Repeat ARQ technique. If node A does not receive a response from a node it contacted after a specified timeout, that peer is declared unresponsive and the next selected node is contacted.

The protocol uses unicast communication with replica hosts to send periodic updates. The updates replace the oldest data stored in the node, so the replica always has a constant size.

Nodes from which no heartbeat are received for a certain time are marked as lost and removed from the set of current replicas. The choice of this timeout needs to be based on the trade-off between bandwidth used by heartbeats and the reactiveness of the protocol to failures.

V. Evaluation

The main goal of our evaluation is to demonstrate the efficiency of the greedy approach of Pirrus in solving the replica placement problem. The first part of this evaluation compares the local replica placement defined in section III with Pirrus. The metric we are considering in this case is the communication cost, expressed as the number of bytes sent in each solution, while varying the number of nodes in the network and their average probability of failure. Pirrus has a lower computational cost with respect to the local replica placement. We investigate the difference in terms of energy cost between the two approaches. Pirrus performs close to the local replica placement. This makes it a good candidate for an implementation on WSNs, where a protocol has to deal with not only reliability but also low computational power and energy constraints. These considerations are supported by the results presented in the next section.

Second, we analyze the performance of Pirrus, focusing on its ability to extend the lifetime of the network, compared to other non-adaptive solutions implementable on WSNs. We consider the network lifetime to be expired when at least one node is not able to satisfy the availability requests. For this analysis we focused on the evolution of the number of replicas managed at each moment and the relative energy cost. The results highlight how the adaptive behavior of Pirrus can save energy in the initial phase of network lifetime, and use this energy later when the network becomes less reliable.

We performed simulations both in MATLAB and *ns-2* to evaluate the different aspects of our strategy.MATLAB has been chosen for its ability in providing algorithms for the optimal solution of binary optimization problems and for the simplicity of the implementation of the prototype. *Ns-2* allowed us to obtain a accurate characterization of Pirrus in a realistic environment.

A. Comparison of local replica placement and Pirrus

In this first set of simulations, we compare the performance of the greedy strategy used by Pirrus with the more complex heuristic approach of the local replica placement. For this purpose, we have used MATLAB to simulate networks of different sizes and different failure probabilities. The optimal solution to the general replica placement problem presented in III would provide a interesting additional comparison, but the problem becomes intractable when the network is composed of more than 10 nodes: the number of decision variables grows with the square of the number of sensors.

The simulations are performed using two network topologies: a grid topology and a random topology. We consider the transmission of an update over one hop as a cost unit. In our scenario a replica is storing up to 80 updates. Creating a new replica requires the transmission of the whole dataset. The cost of this operation is 80 cost units times the number of hops that separate the replica and the original data. Additionally, we assume that each node is able to store up to 12 replicas. The results obtained with this set of simulations are generalizable to other parameter values. The plotted data are the average of 20 executions of each simulation.

The performance of Pirrus and of the local replica placement are similar for both grid and random topologies, as shown in Figure 1a. This fact supports our choice of using the greedy strategy in Pirrus. Also, the total cost of the solution grows linearly with the number of nodes, providing evidence of the scalability of our Pirrus. The figure shows the cost of the solution obtained by each of the algorithms when varying the size of the network. The data availability requirement has been fixed to 0.9 and the average node failure probability is 0.5. For each network size, a new topology is generated and a new placement that satisfies the constraints is computed using the proposed algorithms. The lines start from a network size of 9 nodes. Below this number no solution can be provided with the chosen parameters due to the small number of nodes.

The cost of creating and maintaining replicas depends on the failure rate of the network. More replicas are required in a network where the average failure rate is higher. Pirrus and the local replica placement solutions performs similarly in networks with different average failure rates. As the failure probability increases, the number of replicas to create, and hence the cost of creating them, grows linearly. A solution is considered impossible when a node cannot find a suitable placement for its replicas. The average failure probability over which replica placement becomes impossible depends on the size of the neighborhood that it is used to compute the allocation, (i.e., max distance parameter, see Figure 2b). Figure 1b compares the cost for Pirrus and the local placement strategy to allocate replicas in a network of 100 nodes as the average failure probability of the nodes changes. The solutions are computed using information collected from the nodes within 3 hops.

With a larger hop-count Pirrus could solve the availability problem with an higher average failure rate, having a larger number of neighbors to choose from. However keeping up-to-date information about many neighbors results in a high energy cost that eventually will shorten the network lifetime. We studied the performance of the Pirrus strategy with different values for the hop-count, the results are shown in Figure 2b. A value of 4 will extend the resolvability of the problem when the average failure probability is higher than 70%, but this process is too expensive. We chose a value of 3 hops as a good tradeoff between resolvability and energy cost.

A second parameter that can affect the total cost of replica placement is the value of the availability that the system has to provide. The request of a data availability of 99% doubles the number of messages that nodes running Pirrus have to exchange when compared with a request of only 90%. This choice depends mainly on the application. Figure 2a shows the cost of providing different availability requirements on network of different sizes. The average failure rate has been set to 0.5 and the topology is a grid.

B. Protocol simulations

We implemented Pirrus as described in section IV-B in *ns-2*. We compared its greedy solution with other strategies that can be used to provide data availability: sending data to a base station connected to a reliable storage or providing and maintaining a fixed number of replicas for the whole network lifetime.

The network used in the simulations is composed by 250 nodes placed on a regular grid of size 5200 meters. This topology is comparable with the grid topology used in the MATLAB simulations. The channel access is performed using a simple CSMA scheme, obtained from the 802.11 MAC implementation. The RTS/CTS mechanism is disabled, the data rate is 400 Kbits/s and the maximum packet size is 500 bytes.The network uses the AODV reactive routing protocol. A more sophisticated MAC scheme like S-MAC [16] could improve the energy efficiency of the communications. This goes beyond the scope of this paper. S-MAC or any other efficient MAC protocol can be applied, thus extending the battery life. The adaptive behavior of Pirrus will still provide longer network lifetime compared to the non-adaptive solutions.

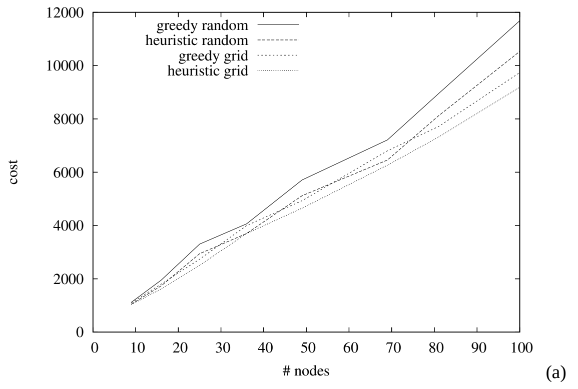
The nodes consume 0.0035 mW while in idle state, 0.06 mW in while in transmission and 0.03 mW in reception. The amount of energy stored in the battery in the beginning is a random variable uniformly distributed between 1875 J and 2500 J. The simulation collected data for a week of simulated time. We selected these parameters to have about a week of estimated battery lifetime (no sleep schedule policy was implemented). Each replica in the system has a size of 80 Kbytes, storing enough samples for three days of information. Each replica update has a size of 1 Kbyte. The maximum packet size is 500 bytes. An update is sent hourly, while heartbeats are broadcast every 300 seconds.

We implemented a very simple failure model to run our simulations. The node availability is proportional to the remaining energy of the battery, capturing the unreliability of the estimation of this value and of the energy consumption of the application running on the nodes. To address all aspects that could affect the nodes' availability, a more complex model has to be defined. This is not the goal of this paper. However a new model can be embedded in Pirrus to enhance its ability to provide data availability.

In the first set of simulations, we compared the data availability provided by Pirrus with the data availability provided by a system that creates a fixed number of replicas. The replica placement strategy that we compare to is presented in [1], and creates a fixed number of replicas on the nodes located nearby the nodes that originate the data to replicate. Other systems [13], [8], [9] replicate data on nodes that are further away from the data source, incurring more communication costs. The goal of these systems is to provide quick access to data and not necessarily to increase its availability. This fundamental difference makes any comparison between Pirrus and this latter approaches not interesting.

In these simulations, the fixed replica system selects the replicas among neighbors within a distance of three hops, starting from the closest ones that have available storage. As in Pirrus, this system uses heartbeat messages to verify that nodes on which replicas are allocated are still alive.

Our simulations compare the average data availability provided by the various approaches (see Figure 3). Additionally, we also analyze the minimum value for the availability in the network at each time tick (Figure 4). The results highlight



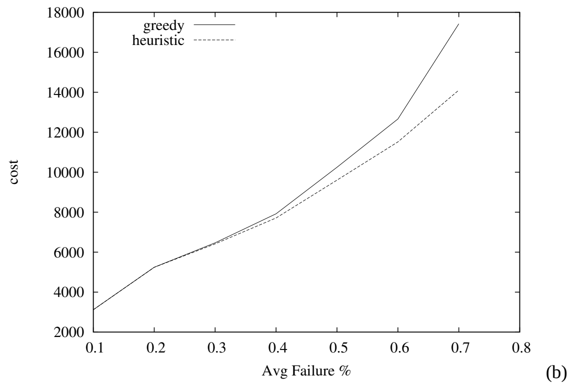
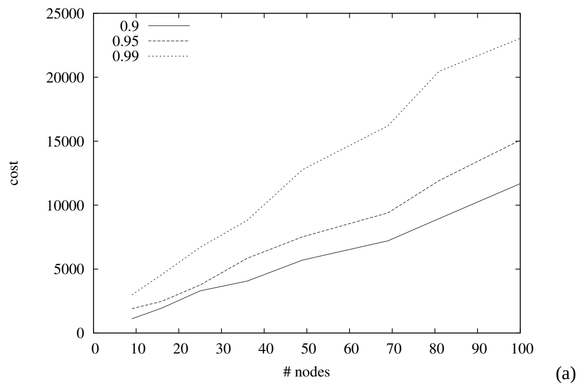


Fig. 1. The cost of the local replica placement and Pirrus are compared while varying the network size or the average failure probability of nodes.



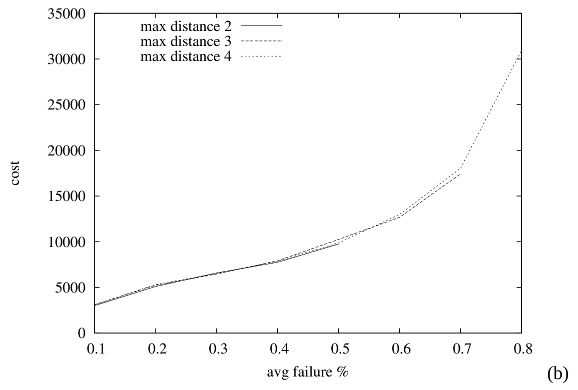


Fig. 2. Cost of the solutions provided by Pirrus while varying the network size for different values of the requested availability (a), and for different values of the neighborhood size while varying the average failure probability (b).

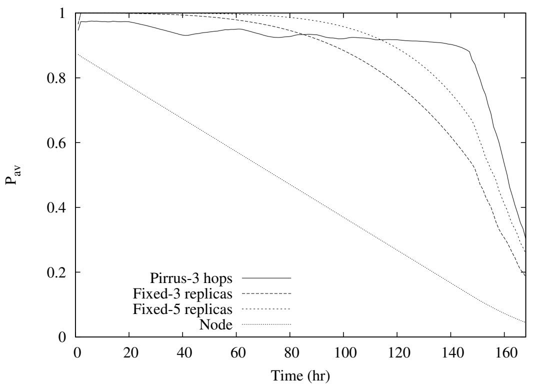


Fig. 3. Average data availability over time

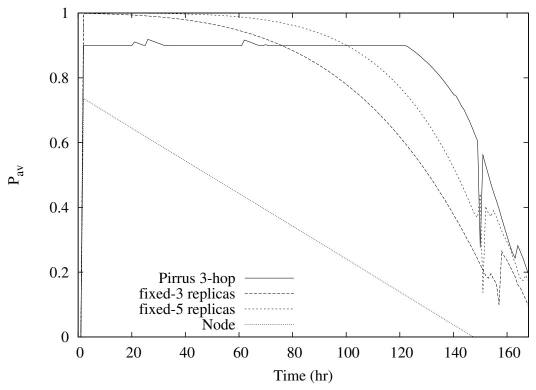


Fig. 4. Minimum value of data availability for each hour.

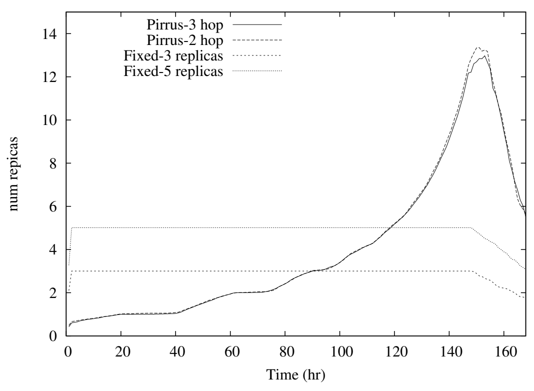


Fig. 5. Average number of replicas over time.

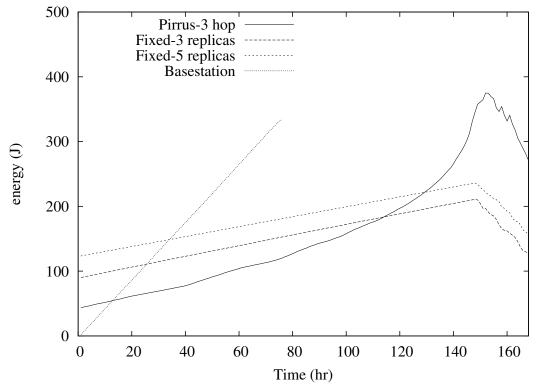


Fig. 6. Average energy spent by each node in transmitting or receiving data.

how, for both the metrics, Pirrus outperforms the non-adaptive solutions. In particular, our definition of lifetime considers the network dead when data availability drops below a threshold (fixed to 90% in these simulations) for at least one dataset. Looking at the graph presenting the minimum value of availability, it is clear that Pirrus extends the network lifetime almost 50% compared to the 3-replica solution, and more than 20% with respect to the 5-replica solution.

To investigate the source of this improvement, we investigate how the number of replicas changes with the evolving conditions of the network (Figure 5). At the beginning of the network lifetime, nodes availability is high, most nodes can satisfy the availability requests without creating replicas. Only a few nodes require some replicas and this results in an average number of replicas that is less than one. However, from the very first moment, the other approaches create a fixed number of replicas that they maintain as long as the energy left in the batteries is enough to provide connectivity. This results availability which is higher than requested, for both the average and the minimum values (see again Figure 3 or Figure 4). As time progresses, the average node availability starts decreasing and so does data availability. Pirrus reacts by increasing the number of replicas, keeping the minimum data availability constant.

We identify a break-even point when the number of replicas of Pirrus exceeds the number of replicas managed by the other approaches (80 hours for 3 replicas and 100 hours for 5 replicas). In companion, the latter start failing to provide the requested availability while Pirrus is still achieving its goal. The system fails after 120 hours. At this point, the average node failure probability (Figure 3) exceeds 0.7, which is the point where Pirrus cannot find solutions within a 3-hops neighborhood (Figure 2b).

To explain how Pirrus can extend the network lifetime, it is necessary to look at energy consumption. In our analysis, we only consider the energy spent in transmitting or receiving data (see Figure 6). Creating and maintaining a fixed number of replicas costs energy from the beginning. The energy spent grows linearly with time. Pirrus instead creates replicas only when needed, requiring less communication and reducing the energy spent. This savings can be used later when more replicas are required to provide availability.

We introduce in this last analysis a third solution in which nodes do not use replication but instead send all updates to a basestation that is assumed to have reliable storage. The energy spent grows very fast and when the batteries of the nodes close to the basestation are empty, the network becomes disconnected and fails.

We intentionally ignored the energy spent in idle state, since as stated at the beginning of this section, a sleep schedule, paired with an appropriate MAC protocol will be able to save energy thus extending the battery life. Intuitively, this should extend network lifetime for all the replication strategies in a proportional way, preserving the validity of the comparison.

VI. Conclusions

This paper presented Pirrus, a data replication system that address the problem of providing data availability over an unreliable network. We analyzed the optimal replica placement problem and defined a simple sub-optimal greedy strategy implementable on wireless nodes. Results show that the adaptive behavior of Pirrus is able to achieve the desired level of data availability with a low energy cost when the nodes are more reliable. The energy saved is used to extend the network lifetime even when the probability of failure for each node is very high.

The approach presented in this paper offers many directions for further research. The failure estimation method that we used in our simulations takes into account only a single cause of failure: the available energy. More complex estimation techniques can be used to take into account environmental conditions or information about the failure of neighbor nodes. Additionally a combination of Pirrus with a system designed to provide efficient data access would be an interesting evolution.

References

[1] M. Albano, S. Chessa, F. Nidito, and S. Pelagatti, “Q-NiGHT: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks,” *Mobile Data Management, 2007 International Conference on*, pp. 166–173, 2007.

[2] J. Gehrke and S. Madden, “Query processing in sensor networks,” *IEEE Pervasive Computing*, vol. 3, no. 1, pp. 46–55, 2004.

[3] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, “TinyDB: an acquisitional query processing system for sensor networks,” *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 122–173, 2005.

[4] C. Ebeling, *An introduction to reliability and maintainability engineering*. McGraw Hill, 1997.

[5] F. Dabrek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area cooperative storage with CFS,” *Proceedings of the ACM Symposium on Operating System Principles (SOSP)*, pp. 202–215, 2001.

[6] E. Sit, A. Haeberlen, F. Dabek, B. Chun, H. Weatherspoon, R. Morris, M. Kaashoek, and J. Kubiatowicz, “Proactive replication for data durability,” *5rd International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.

[7] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker, “Total recall: system support for automated availability management,” *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation-Volume 1 table of contents*, pp. 25–25, 2004.

[8] L. Yin and G. Cao, “Supporting cooperative caching in ad hoc networks,” *Mobile Computing, IEEE Transactions on*, vol. 5, no. 1, pp. 77–89, Jan. 2006.

[9] B. Tang, H. Gupta, and S. R. Das, “Benefit-based data caching in ad hoc networks,” *Mobile Computing, IEEE Transactions on*, vol. 7, no. 3, pp. 289–304, March 2008.

[10] T. Hara, “Effective replica allocation in ad hoc networks for improving data accessibility,” *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1568–1576 vol.3, 2001.

[11] T. Hara, N. Murakami, and S. Nishio, “Replica allocation for correlated data items in ad hoc sensor networks,” *ACM SIGMOD Record*, vol. 33, no. 1, pp. 38–43, 2004.

[12] M. Shinohara, T. Hara, and S. Nishio, “Data Replication Considering Power Consumption in Ad Hoc Networks,” *Mobile Data Management, 2007 International Conference on*, pp. 118–125, 2007.

[13] A. Ghose, J. Grossklags, and J. Chuang, “Resilient Data-Centric Storage in Wireless Ad-Hoc Sensor Networks,” *Mobile Data Management: 4th International Conference, Mdm 2003, Melbourne, Australia, January 21-24, 2003: Proceedings*, 2003.

[14] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, “Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table,” *Mobile Networks and Applications*, vol. 8, no. 4, pp. 427–442, 2003.

[15] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*, 1990.

[16] W. Ye, J. Heidemann, and D. Estrin, “An energy-efficient MAC protocol for wireless sensor networks,” *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2002.