Distributed Fault-Tolerant Backup-Placement in Overloaded Wireless Sensor Networks*

Gal Oren[1,2], Leonid Barenboim[3], and Harel Levin[2,3]

[1] Department of Computer Science, Ben-Gurion University of the Negev, P.O.B. 653, Be'er Sheva, Israel
[2] Department of Physics, Nuclear Research Center-Negev, P.O.B. 9001, Be'er-Sheva, Israel
[3] Department of Mathematics and Computer Science, The Open University of Israel, P.O.B. 808, Ra'anana, Israel
orenw@post.bgu.ac.il, leonidb@openu.ac.il, harellevin@gmail.com

Abstract. Wireless Sensor Networks (WSNs) frequently have distinguished amount of data loss, causing data integrity issues. Sensor nodes are inherently a cheap piece of hardware - due to the common need to use many of them over a large area - and usually contain a small amount of RAM and flash memory, which are insufficient in case of high degree of data sampling. An overloaded sensor can harm the data integrity, or even completely reject incoming messages. The problem gets even worse when data should be received from many nodes, as missing data becomes a more common phenomenon as deployed WSNs grow in scale. In cases of an overflow, our Distributed Adaptive Clustering algorithm (D-ACR) reconfigures the network, by adaptively and hierarchically re-clustering parts of it, based on the rate of incoming data packages in order to minimize the energy-consumption, and prevent premature death of nodes. However, the re-clustering cannot prevent data loss caused by the nature of the sensors. We suggest to address this problem by an efficient distributed backup-placement algorithm named DBP-ACR, performed on the D-ACR refined clusters. The DBP-ACR algorithm re-directs packages from overloaded sensors to more efficient placements outside of the overloaded areas in the WSN cluster, thus increasing the fault-tolerance of the network and reducing the data loss.

Keywords: Wireless Sensor Networks · Distributed Backup-Placement · Data Loss · Networks Connectivity.

1 Introduction

1.1 Fault-Tolerance and Data Loss in WSNs

Considering their independent and environmentally-varied work-fashion, one of the most important factors in WSN applications is fault-tolerance. Due to the fact that the possibilities of an absent sensor node, damaged communication

*This work was supported by the Lynn and William Frankel Center for Computer Science, the Open University of Israel's Research Fund, and ISF grant 724/15.

link or missing data are unavoidable in wireless sensor networks, fault-tolerance becomes a key-issue. Among the causes of these constant failures are environmental factors, battery exhaustion, damaged communications links, data collision, wear-out of memory and storage units and overloaded sensors [1]. WSN can be in use for a variety of purposes, nevertheless its fault-tolerance needs to depend mostly on the application type. Wireless video sensor networks, for example, tends to rely on accurate and precise massive amount of sensed data, thus demanding WSNs to support high degree of data sampling [2], which consequently means high quality of recording, processing and transmitting of the data from the captured environment, which sometimes might be unstable. The data storage capacity on the sensors is crucial because whereas some applications [3] require instantaneous transmission to another node or directly to the base station, others demand intervallic or interrupted transmissions. Thus, if the amount of data is large - as a derivative of the data precision needed by the application [4] - WSN nodes are required to store those amounts of data in a rapid and effective fashion till the transmission stage [5].

However, since those requirements are mostly depend on the hardware and the wireless settings, WSNs frequently have distinguished amount of data loss, causing data integrity issues [6]. Sensor nodes are inherently a cheap piece of hardware, due to the common need to use many of them over a large area, sometimes in a non-retrievable environment - a restriction that does not allow a usage of a pricey tampering or overflow resistant hardware, and a damaged or overflowed sensor can harm the data integrity, or even completely reject incoming messages [6]. The problem gets even worse when there is a need for high-rate sampling or when data should be received from many nodes since missing data becomes a more common phenomenon as deployed WSNs grow in scale [7]. Therefore, high-rate sampling WSNs applications require fault-tolerant data storage, even though this requirement is not realistic.

1.2 Data Storage in WSNs

WSNs usually consist of sensor devices that are able to sense or receive data, process it fully or partially, and finally transmit it to another node - either a cluster-head or a base-station. At the receiving-sensing state, in most cases, the data is placed in the memory in a serial fashion, processed minimally or not at all, then transmitted in a First-In-First-Out mode, and then deleted from the sensor device memory [8]. This technique has obvious disadvantages from the energy-efficiency point of view. As a common bypass, a well-known approach to reduce energy-consumption is to place as many data as possible in the RAM, and only when reaching to the full capacity to send it in a batch fashion forward. Another option is to process it minimally and send less data than originally received (e.g. minimum function), as transmitting the data is a very costly operation. Moreover, the relatively recent introduction of flash memory into sensor devices also expanded their capability to store data locally [9]. However, one of the critical problems with this storage is the dramatic disproportionate write-erase granularity. Specifically, erasing one block (64 to 128 pages) while writing

only one page (512B to 8KB). This means that continuous high-rate data acquisition applications cannot rely on it, since the device will reach its memory limit eventually. Also, because the RAM acts as a buffer towards the flash memory, the pages amount is limited by the RAM capacity, which is usually about an order of magnitude smaller than the flash capacity. Another known problem of the flash memory is its sensitivity to constant write-erase cycles which characterizes high-rate applications, causing a premature fatigue of the hardware, thus to data loss [8].

In order to overcome those flash technology limitations new and enhanced storage technologies were introduced, including the NVSRAM, FRAM and MRAM - which managed to push write-erase granularity performances to the limit [10]. Nonetheless, physical limitation derived by the sensor device shape and budget are not enabling an embedding of more than a 1-megabyte storage per unit. Therefore, for continuous high-rate data acquisition applications, this is not a realistic solution. In the following sections we will devise a solution relying on the reconfigurable nature of the whole network, which eventually would be able to balance the load in an overloaded network properly between the different nodes.

2 The Backup-Placement Problem in WSNs

The backup-placement problem in general graphs was introduced by [11]. This problem turned out to be very challenging in general networks. An approximate solution with a polynomial number of rounds was presented in [11].

In the current work, however, we focus on WSNs, and we are interested in significantly better than polynomial (and even than linear) solution. To illustrate the backup-placement problem in WSNs consider the following scenario: several nodes (sensors) in a WSN have packages (sensed or received data) whose backups (or the package itself) need to find a placement elsewhere in the network, due to overload on the area of the network which those nodes belongs to, in order to improve fault-tolerance and data integrity. Because of the lack of capacity to store or process the data on the node itself or in neighbor nodes - which also might be burdened by an overflow of data - it is mandatory, when the local memory is full, to find a backup-placement to the data outside of the overloaded area. The backup-placement problem is defined as follows. (1) How to place the data only once in a safe and stable node in order to assure with a high degree of certainty the data integrity and minimization of the WSN load, and (2) how to do so without creating an additional data overflow on other areas of the network.

In terms of graph theory, the problem in its simplest form is defined for a network graph $G = (V, E)$ as follows. Each vertex in V must select a neighbor, such that the maximum number of vertices that made the same selection is minimized [11].

In order to demonstrate this problem, we can examine two test cases. Let $G = (V, E)$ be a graph representing a network. If, for example, G is a cycle graph, the optimal solution to the backup-placement problem would be if each node selects its succeeding neighbor to be its backup node, and by that the

maximal backup burden for each node would be of only one unit (Fig. 1, right). However, if, for example, G is a star graph, the optimal solution will force all the nodes to choose the center node (beside the center node itself) to be its backup node, and by that the maximal backup burden would be of $|V|-1$ (Fig. 1, left) - a very problematic solution due to the overload on a single node, which, as previously explained, we specifically wish to avoid. While this is unavoidable in a star graph, it becomes possible in wireless-network topologies.

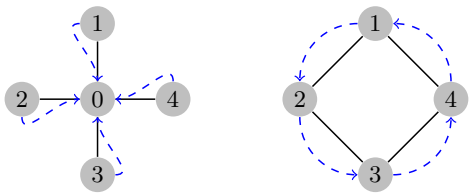

Fig. 1. Optimal backup placement in star graph (left) and cycle graph (right)

Next, we examine a WSN which can be represented as a unit disk graph (UDG) on which a spanning tree has been computed, in order to demonstrate the considerations that must be taken while designing a distributed backup-placement algorithm. We choose UDG because it tends to represent the immanent behavior of wireless communications [8].

Let $G=(V,E)$ be a spanning tree of a UDG representing a sensor network. Let v_0 be the root of G. Now, even if the number of nodes inside v_0's transmission area is large, most of them are interconnected, and form at most 6 different cliques, as the UDG definition forces. It means that v_0 children form at most 6 cliques. Assuming that nodes IDs are consecutively numbered, in each clique each node selects a node with ID greater by 1 than its own to be its backup node, assuming the existence of such a node. If such a node is absent, but some neighbors have greater IDs, then the selection is the neighbor with the closest ID to the backed-up node ID, out of these neighbors. Consequently, only one node in each clique - the one with the highest ID - will be left without a backup node. In total there will be 6 nodes without backup-placement within their clique. Each such node v will set its parent node $\pi(v)$ as its backup node.

Algorithm 1 The 1-hop Back-Placement Algorithm

1: **procedure** 1-HOP-BP(NODE v, GRAPH G)
2: **Find** a node u in the list of sibling nodes connected with v in G such that $ID(u)=ID(v)+k$, where k is the smallest positive integer for which a node u exists in the list
3: **if** found **then**
4: **Select** u to be v backup node
5: **else**
6: **Select** $\pi(v)$ to be v backup node

Using this algorithm, the maximum extra-load per node is $12=O(1)$. This is because each node in V is selected by at most 6 of its children, and by at most 6 additional sibling nodes. Indeed, two siblings of the same clique cannot select

the same sibling neighbor. This is because in this case the latter sibling ID is greater than both IDs of siblings that selected it. But then, one of the selecting siblings does not select an ID that is closest to its own, which contradicts step 3 of the algorithm.

Therefore, the load is optimal up to a small constant factor. The computation is performed within a constant radius of each node and requires $O(1)$ rounds, and so the distributed running time complexity is $O(1)$. Furthermore, instead of representing the original graph as a spanning tree, it is possible to decompose the graph into a set of trees in a way that all those trees will form a spanning forest (See, e.g., [12]). The time complexity of this action is $O(1)$, which consequently turns the total distributed backup-placement complexity of this algorithm to be $O(1)$. However, the 1-hop backup-placement algorithm is not practical for real-world WSNs. One of the reasons is that the distribution of the load on the network is almost never focused on one node, but on an area, which several nodes belong to. Therefore, a backup-placement of one node in this area to its neighbor - which also belongs to this overloaded area - is counterproductive [13]. In this state, all the nodes in the area are overloaded, but nevertheless try to place their incoming packages to other nodes which are also overloaded, and by that create an even greater pressure on this area of the network, subsequently causing a data loss and energy-waste.

Hence, it is clear that the solution to the backup problem with overloaded areas is by transferring the packages outside of the area dynamically, to a non-overloaded area. In order to do so we need to address three main difficulties: (1) How is the distributed algorithm supposed to detect which nodes under which area are not overloaded? (2) How is the distributed algorithm supposed to choose the backup nodes and transfer the packages to it in an optimal fashion? (3) How during this selection process are the parent nodes in the tree will not be overloaded?

It is a necessity to address those difficulties, mainly because (1) the detection of the overloaded areas is an expensive task, (2) the selection of the backup nodes outside of the overloaded areas and the packages transfer to it should be based on an optimal routing scheme as the communication and energy consumption are the top priorities , and (3) the structure of the tree does not insure that many nodes will not transfer their backup packages through small amount of parent nodes, which will consequently cause to their premature death due to energy consumption, and thus to the reduce of connectivity of the whole network [14]. In order to tackle those problems, we first use our distributed Adaptive Clustering Refinement algorithm (or D-ACR) [15] in order to re-cluster the network in a way which will reduce the burden on specific nodes of the WSN as possible, and afterwards we will take advantage of the hierarchical fashion of the D-ACR refinement algorithm tree in order to find a placement for data from overloaded nodes using the DBP-ACR algorithm. These two algorithms are complementary as we will present in the next sections.

3 The DBP-ACR Algorithm

3.1 Problem Formulation

In order to formulate the problem, we first need to define the setting of the network. We assume a WSN with an array of n nodes - including the base station (BS), the cluster-heads (CH) and the non-cluster-heads (NCH) - in a confine $Latitude \times Longtitude(m^2)$ quadrate, while the position of each node in this area is represented by the coordinates (x_i, y_i). We assume a communication model in which the transmission energy is calculated using the d^2 power-loss model. The optimization problem is to find a backup-placement for sensed data packages that could not be stored on their sensing node due to data stream overload and physical constraints, thus creating a data loss. By doing so it is possible to keep data integrity while consuming the minimum energy possible from the WSN as a whole, and also keep a load-balanced WSN structure that will not burden heavily on some parts of the network - resulting premature death of nodes and damage the WSN lifetime and connectivity at once. It is worthwhile mentioning that we take for granted all the general presumptions regarding the network structure as presented in [15].

3.2 D-ACR algorithm

Unlike other WSN clustering algorithms, that do not re-cluster the network after deployment (except in nodes join/leave), our hypothesis is that it is advisable, in terms of prolonging the network lifetime, to adaptively re-cluster specific regions that are triggered significantly more than other regions in the network. By doing so, it is possible to minimize or even prevent the premature death of CHs, which are heavily burdened with sensing and transmitting actions - much more than other parts of the WSN. In order to do so we introduced the centralized and distributed Adaptive Clustering Refinement (ACR) algorithms [15].

In general (Fig. 2), the D-ACR load-balancing-based algorithm tries to locate the load burdened areas. The amount of the revealed areas establishes the regions amount that will need to re-cluster themselves. Because the load factor determines the refinement action, the D-ACR algorithm can recursively identify the load burdened areas and re-cluster them in a hierarchical tree assembly in which the deeper the nodes are in the tree depth, the more burden they are.

Algorithm 2 The Distributed Adaptive Clustering Refinement Algorithm

```
1: procedure D-ACR(Node v, Threshold t)
2:     if v is a leaf and energy_use(v) ≥ 3t then
3:         Refine(v)
4:         Add new CHs as children of v
5:     if all children of v are leafs and average_energy(children(v)) < (t/5) then
6:         Coarsen(v)
7:         Remove the children of v from the tree and mark v as a leaf
```

The D-ACR algorithm is executed by all CHs in parallel. Initially, each CH is provided with a threshold value t of a plausible energy use. If these values

are unknown in the beginning of the execution, they can be computed using a single execution of the centralized ACR algorithm (C-ACR). This combination of an initial global execution with numerous local executions following it, is still more efficient than performing several executions of the C-ACR. In the initial configuration, all these values t are the same, and represent a balanced environment. The algorithm can start from any cluster-hierarchy tree, where the simplest configuration is a single CH, which is the root (equivalently, a single leaf). Once an energy use of a leaf v reaches $3t$, we perform a local refinement in the cluster of v [15]. This results in adding new CHs to the tree as leafs that become the children of v. These new leafs correspond to the newly formed clusters. This refinement results in a better energy use in each such newly formed cluster, specifically, bounded by $\frac{3}{10} \cdot 3t < t$ instead of $3t$ [15]. In other words, we balance clusters of excess energy-use by decomposing them into smaller clusters that require less energy.

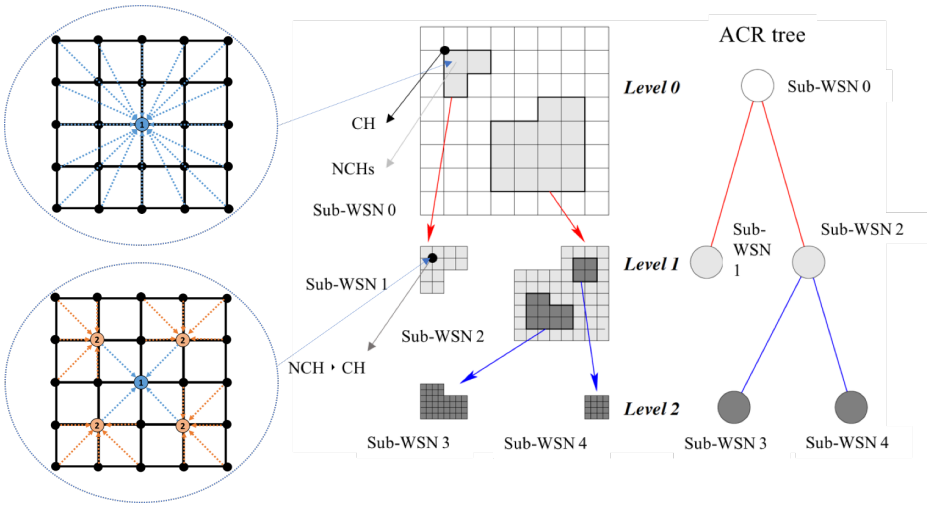

Fig. 2. An ACR WSN-Graph Tree formation example with 2 levels of refinement (right). A finer resolution WSN-Graph is applied each time a sub-WSN is created (middle); A magnification of the square which represent a 5×5 WSN grid with one CH to 24 NCHs before refinement (upper left), and the same grid after refinement with 4 NCHs turned into CHs (lower left).

Once the average energy consumption in the children of a CH node u whose all children are leafs becomes less than $(t/5)$, a coarsening operation is performed (this operation is the opposite of refinement). Specifically, the clusters represented by u and its children are merged into a single cluster. Then u becomes its CH, and former children of u become NCHs. Consequently, the energy use of the newly-formed larger cluster grows, but the tree-distance between the root to some leafs decreases. This completes the description of the algorithm. The algorithm provided below is executed periodically by each CHs.

3.3 DBP-ACR Algorithm and Analysis

In order to present the DBP-ACR algorithm we will first explain its synergy to the D-ACR algorithm. In Fig. 3 there is a WSN with 4 CHs which each has several NCH. We assume that at certain time the load on the leftmost CH sub-network increased dramatically, and that the reconfiguration of the network is based on D-ACR algorithm. Therefore, the network will immediately start re-cluster itself in order to create a load-balanced network.

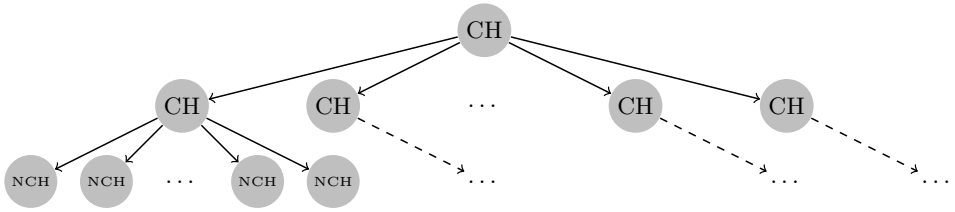

Fig. 3. A WSN with 4 CHs which each has several NCH.

In [15] it was proven that the maximum depth of a refined tree is bounded by $O(\log(L/t))$, and the maximum tree size is bounded by $2^{O(\log(L/t))}$, where L is the maximum load, t is the threshold value of a plausible energy use, and $k = 2$ (i.e. each refined CH creates $k^2 = 4$ child-CHs). At each D-ACR refinement, the new CHs know their refinement level in the tree by getting the refinement level from their parent. This requires only one additional message per new CH, each message of size $O(\log\log(L/t))$ bits to represent the depth in the D-ACR tree. The total number of CH nodes in the generated D-ACR tree is bounded by $O(n^2)$, where n is the number of CHs in the input D-ACR tree [15]. Additionally, each package which is needed to be transferred to a backup node will need an additional bit to reference that this package is not a sensed data package, but rather a package that routes to its final destination. We next provide the psudocode of the algorithm and then explain it.

The DBP-ACR algorithm is executed by all nodes in parallel once a package has received. Initially, each node checks its memory vacant capacity (we suppose that the sensor provides this datum). If there is no vacancy in the memory system and the package has arrived from a local sensing area (line 3) the package will be marked as a package which searches a backup-placement in other part of the WSN. This step is essential because of the need of the rest of the nodes in the WSN to figure out if the package has arrived from their local sensing area or either it's in a route towards a backup-placement destination, meaning that the node will not need (or should not need) to perform any processing on the package, or rather handle it as a sensing data of its children. Then, the package will be sent to the parent of the current node in the D-ACR tree.

Afterwards, the package will arrive to the CH nodes which have been created during the D-ACR refinement stage. Those nodes most likely will not be over-loaded due to the D-ACR re-clustering which creates a load-balancing at the WSN refined zones. At those CHs the package is already marked as a backup-placement package which searches a placement in other part of the WSN (line 6).

Algorithm 3 The WSN Distributed Backup-Placement Algorithm

```
1: procedure DBP-ACR(Node v, Package P)
2:     Initially, v.RR_index ← 0
3:     if v.memory reach full capacity ∧ P.bp = False then
4:         P.bp ← True
5:         Send P → v.father
6:     else if P.bp = True then
7:         if v.ACR_depth ≠ NULL then
8:             Send P → v.father
9:         else
10:            P.bp ← False
11:            if v.RR_index = P.send_ad then
12:                v.RR_index ← (v.RR_index + 1) mod len(v.children)
13:            Place P → v.children[v.RR_index]
14:            v.RR_index ← (v.RR_index + 1) mod len(v.children)
```

The algorithm will transfer the package directly all the way through the D-ACR tree newly formed CHs (the depth of the D-ACR tree) until reaching the root node from which the D-ACR refinement process has started (lines 7, 8). The sibling nodes of this root node are most likely have not been overloaded, and not been refined by the D-ACR algorithm (otherwise, most likely they would have been part of the current D-ACR tree). Those nodes, and their sub-trees nodes, are suitable to store or process the package. When the package reached this point (line 9) it stops marking itself as a package searching for backup-placement (line 10) and the root node send it to its final destination by a counter index of its sons (line 13), which distribute the incoming packages evenly between its other children (except the one from which the package has arrived which is stored in *P.send_ad* lines 11 and 12) in a round-robin fashion controlled by *v.RR_index* (line 14) in order to balance the load on those nodes. It is important to notice that although there is no massive load on those backup nodes at the beginning, as time goes by more and more packages will try to find a backup-placement outside of their D-ACR tree, it means that those other nodes will initiate eventually a D-ACR in order to ease the burden of those packages on the CHs, and eventually start the DBP-ACR algorithm too in order to find placement for the excessive packages arrival. This means that the two distributed algorithms, the DBP-ACR and the D-ACR work in a strong collaboration in order to create a load-balanced WSN, even though the two are completely independent in their work fashion.

An exemplification of the algorithm action is presented in Fig. 4. As previously explained, the algorithm is based on the refinement re-clustering of the D-ACR algorithm, and the tree in Fig. 4 is based on such refinement of the tree in Fig. 3 (stage 1). Afterwards, one of the leaf nodes of the tree (i.e. NCH) reach the top of its memory capacity and forced to mark the incoming package as a package which need to find a backup-placement. The package is then transferred to the parent node - which is a newly formed CH by D-ACR actions. Then, the package is directly traversed through the n levels of refinement CHs (stage 2) until it reaches the root node - i.e. the father of the node from which the D-ACR

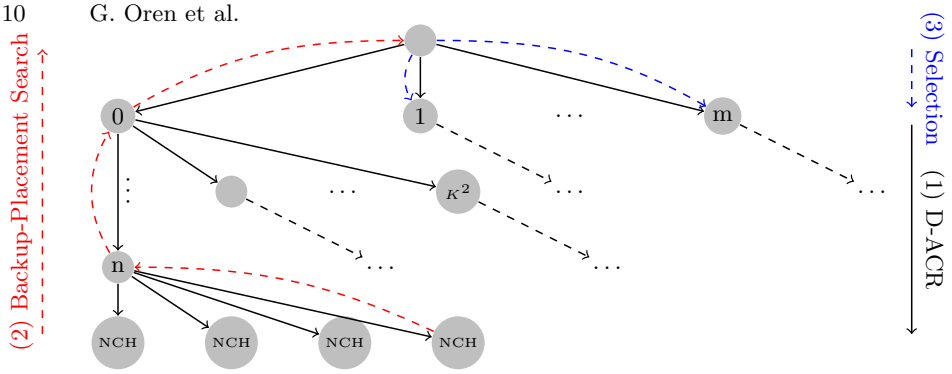

Fig. 4. The DBP-ACR algorithm stages: (1) The D-ACR refinement (2) The backup-placement search (3) The selection of a node as a backup-placement node outside of the overloaded area.

initiated at the beginning (node 0). The package then finds his placement among the other descendants of the root node, which are not burdened. Those action are performed locally at each node through the whole process.

The running time of the algorithm is presented in Theorem 1, and the memory consumption of the algorithm in Theorem 2. The energy consumption of the algorithm package transmission is provided by Theorem 3.

Theorem 1. *The running time of the algorithm is bounded by* $O(\log(L/t))$

Proof. Suppose we have maximum load L. The load may be divided to sub-clusters. Every D-ACR refinement step will decrease the load, allowing no more then $L/3$ load. The D-ACR refinement will repeat this division q times, until the load will be less than $3t$. Thus, we obtain the following formula describing the number of rounds until the necessary refinement is achieved: $L(1/3)^q \geq 3t \Rightarrow q \leq O(\log(L/t))$, while q is the average ratio of graph sizes between two levels of graphs. The DBP-ACR algorithm can initiate at any NCHs under the D-ACR refinement tree, which only adds 1 hop between this node and the immediate parent CH. Also, when the packet reaches the top of the D-ACR tree it finds a backup node in a round-robin fashion at one of the m child nodes of the root node, which means a complexity of $O(1)$. Therefore, the total time complexity of the DBP-ACR algorithm is $1 + O(\log(L/t)) + 1 = O(\log(L/t))$.

Theorem 2. *The extra-memory consumption of the algorithm is bounded by* $O(\log\log(L/t))$

Proof. As previously explained, at each D-ACR refinement, the new CHs knows it refinement level in the tree by getting the refinement level from their parents. Because we proved that the maximum depth of the D-ACR tree is bounded by $O(\log(L/t))$, it means that the number of bits we will need in order to represent this information is $O(\log\log(L/t))$. The additional bit for *P.bp* and the additional locally allocated argument at the root CH $v.RR\_index$ will result that the extra memory consumption of the algorithm is bounded by $O(\log\log(L/t) + 1 + \log(m)) = O(\log\log(L/t))$, as the m child nodes of the root node are must be small [15].

Theorem 3. *The energy consumption of the algorithm package transmission is bounded by* $O(\log(L/t)E_{elec} + \epsilon_{fs}(\log(L/t)d_{ref})^2)$

Proof. Based on the D-ACR refinement, we can assume the maximum distance between a NCH at the lowest refinement level and its CH in a D-ACR refinement tree to be the distance between any CH in this tree and its parent, which is also a CH. This equality is kept all the way through the distance between the CH at the first refinement level and the root CH node, from which the D-ACR refinement started. We denote the maximum distance between any two nodes in a consecutive hierarchy level (i.e. level i and level $i+1$) which transfer package for backup-placement purpose to be d_{ref}. Thus, the energy consumption needed to transmit a bit from one node to its parent in the D-ACR tree would be:

$$E_{ACR\_Node} = E_{elec} + \epsilon_{fs}{d_{ref}}^2 \tag{1}$$

As previously proven (Theorem 1), the maximum depth of a D-ACR tree is bounded by $O(\log(L/t))$. Therefore, the maximum amount of energy that would be spent on this transmission will be:

$$E_{ACR\_BP} = \log(L/t)E_{ACR\_Node} = \log(L/t)(E_{elec} + \epsilon_{fs}{d_{ref}}^2) \tag{2}$$

As shown in the DBP-ACR algorithm, the last step, of actually placing the package in a backup-placement node, take place outside of the D-ACR tree - meaning that the maximum distance between the D-ACR root node and its parent, and the distance between the D-ACR root node parent and one of its children - where the package will be placed - is at most $\log(L/t)d_{ref}$ each, as this multiplication equals the maximum distance between a node and its child in the original non-refined network (the original maximum distance at is $d = \log(L/t)d_{ref}$). Therefore, the maximum total energy consumption needed in order to find a backup-placement node and place the package in it is:

$$E_{BP\_TOT} = E_{ACR\_BP} + 2E_t = \tag{3}$$

$$\log(L/t)(E_{elec} + \epsilon_{fs}{d_{ref}}^2) + 2(E_{elec} + \epsilon_{fs}(\log(L/t)d_{ref})^2)$$

Therefore, the energy consumption of the algorithm package transmission is bounded by $O(\log(L/t)E_{elec} + \epsilon_{fs}(\log(L/t)d_{ref})^2)$.

4 Conclusions

In this paper we introduced the Distributed Fault-Tolerant Backup-Placement in Overloaded Wireless Sensor Networks algorithm, named the DBP-ACR. We first surveyed the main fault-tolerance issues that WSNs facing, specifically the data integrity and loss phenomenon and the sensor nodes storage advantages and disadvantages. Afterwards, we defined the Backup-Placement Problem in WSNs and defined the considerations needed in order to actually manage to perform an optimal backup-placement in WSNs. Finally, we showed the complementary fashion of the D-ACR algorithm and the DBP-ACR algorithm, and proved its energy, memory and running time complexities to be close to optimal.

References

1. Kakamanshadi, Gholamreza, et al. A survey on fault tolerance techniques in wireless sensor networks. Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on. IEEE, 2015.
2. Seema, Adolph, et al. Towards efficient wireless video sensor networks: A survey of existing node architectures and proposal for a Flexi-WVSNP design. IEEE Communications Surveys & Tutorials 13.3 (2011): 462-486.
3. Mottola, Luca, et al. Programming wireless sensor networks: Fundamental concepts and state of the art. ACM Computing Surveys (CSUR) 43.3 (2011): 19.
4. Oren, Gal, et al. Adaptive Distributed Hierarchical Sensing algorithm for reduction of wireless sensor network cluster-heads energy consumption. Proceedings of the 13th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 980-986, IEEE 2017.
5. M. Balazinska, et al. Data Management in the Worldwide Sensor Web. IEEE Pervasive Computing, vol. 6(2), pp. 30-40, April-June 2007.
6. Kong, Linghe, et al. Data loss and reconstruction in sensor networks. INFOCOM, 2013 Proceedings. IEEE, 2013.
7. Luo, Chong, et al. Compressive data gathering for large-scale wireless sensor networks. Proceedings of the 15th annual international conference on Mobile computing and networking. ACM, 2009.
8. Wagner, Dorothea, et al. Algorithms for sensor and ad hoc networks: advanced lectures. Springer-Verlag, 2007.
9. Vieira, Marcos Augusto M., et al. Survey on wireless sensor network devices. Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference. Vol. 1. IEEE, 2003.
10. Nuns, Thierry, et al. Evaluation of recent technologies of non-volatile RAM. Radiation and Its Effects on Components and Systems, 2007. RADECS 2007. 9th European Conference on. IEEE, 2007.
11. Halldrsson, Magns M., et al. Distributed backup placement in networks. Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures. ACM, 2015.
12. Barenboim, Leonid, et al. Distributed graph coloring: Fundamentals and recent developments. Synthesis Lectures on Distributed Computing Theory 4.1 (2013): 1-171.
13. Wajgi, Dipak, et al. Load balancing based approach to improve lifetime of wireless sensor network. International Journal of Wireless & Mobile Networks 4.4 (2012): 155.
14. Kakiuchi, Hirofumi. Dynamic load balancing in sensor networks. Stanford, 2004.
15. Oren, Gal, et al. Load-Balancing Adaptive Clustering Refinement Algorithm for Wireless Sensor Network Clusters. International Conference on Wired/Wireless Internet Communication. Springer, Cham, 2017.