DKM: Distributed *k*-Connectivity Maintenance in Wireless Sensor Networks

Piotr Szczytowski, Abdelmajid Khelil, Neeraj Suri
CS Department, TU Darmstadt, Germany
{szczytowski, khelil, suri}@informatik.tu-darmstadt.de

***Abstract*—The reliability of Wireless Sensor Networks (WSN) is detrimentally impacted by unreliable wireless communication and the finite energy of sensor nodes. An advocated approach for assuring fault tolerant WSN is providing global *k*-connectivity. This property guarantees that the failure of up to k - 1 sensor nodes does not cause network partitioning. *k*-connectivity is a well studied property of WSN including the aspects of topology control, *k*-connected dominating set construction, controlled deployment, relay nodes placement, and detection of level of *k*-connectivity. In this work, we target the repair/maintenance aspect of *k*-connectivity. Our goal is to allow the network to provide localized, sustainable maintenance, which is capable of efficiently restoring/main-taining the WSN desired *k*-connect-ivity. We present a fully distributed technique that is competitively resource efficient to state-of-the-art approaches. Unlike existing techniques, our approach also provides the necessary efficient mechanisms to avoid network partitioning and the longer routing paths caused by node failures. We present both analysis and simulations to show the effectiveness and efficiency of our solution to maintain high responsiveness.**

I. Introduction

Wireless Sensor Networks (WSN) compromise sensing, communication and computational operations. A typical WSN consists of multiple battery powered autonomous sensor devices for sensing and processing environmental attributes of interest, and sharing this data wirelessly. These characteristics allow for a wide selection of applications and high flexibility for deployment. Unfortunately, the low reliability of wireless communication and WSN nodes along with their bounded batteries lifetimes frequently leads to failures, and consequently the degradation of the overall system responsiveness and resilience. Hence the need to provide for fault-tolerance in WSN operations. A required condition for meaningful WSN operations is the maintenance of network connectivity. An established approach is to provide the so-called *k*-connectivity property. The *k*-connectivity property guarantees that the removal of up to *k - 1* sensor nodes does not lead to network partitioning (i.e., disconnection of one or more sensor nodes from the rest of the network). An oft used approach to maintain *k*-connectivity is to keep injecting new nodes whenever and where needed. The design goal is to minimize the number of added nodes without compromising the *k*-connectivity. A popular approach for providing *k*-connectivity is to stock selected sensor nodes constituting the backbone of the network with up to *k - 1* additional sensor nodes [2]. Alternatively, missing links are reconnected using *k*-stocked intermediate sensor nodes [1], [12]. This approach, although effective and theoretically sound, unfortunately does not handle the case when k co-located sensor nodes fail simultaneously, e.g., as a result of an external physical force. In order to prevent such situations, it is necessary for a *k*-connectivity algorithm to guarantee that for each sensor nodes among its k alternative paths, none of the these paths share the same sensor node location. Another important requirement for the *k*-connectivity algorithms (which is generally neglected) is to assure that the use of alternative paths, in case of failures, does not substantially degrade the performance of the network. A typical example is the failure of single sensor nodes which, while not partitioning the network, significantly extend the length of the routes resulting in higher latencies, frequent congestion and higher energy overhead. Additionally, although the network remains connected on the topology level, traffic rerouting may be necessary causing temporary communication outages until new routes are established.

Paper Contributions

In this paper, we propose a novel approach for runtime repair and preservation of the global WSN *k*-connectivity. The proposed strategy while requiring only localized information significantly reduces the resource requirements compared to all contemporary studies. Beyond maintaining the *k*-connectivity, it also prevents the network from location bound failures by elimination of single *k*-stocked links. For typical sink oriented WSN traffic, our approach prevents the network performance from degrading. The proposed technique, i.e., Distributed *k*-Connectivity Maintenance (DKM):

- Efficiently restores and preserves the *k*-connectivity.
- Eliminates potential WSN disconnections by assuring k ($k \geq 2$) disjoint paths between each two sensor nodes.
- Provides *k*-connectivity along with guaranteeing that the length of particular routes may become extended only up to k - 1 hops.
- Is fully distributed in its operations.

Additionally DKM can function in localization free context, while still providing *k*-connectivity and maintaining responsiveness. However, assuring the disjoint paths property requires localization information.

We validate DKM properties with an extensive set of evaluation studies and show DKM's effectiveness at restoring *k*-connectivity with low resource requirements.

The paper discusses related work in Section II. Following the system model in Section III, we present the maintenance/repair requirements in Section IV. Section V details the proposed DKM technique as the paper's main contribution, with DKM's evaluation in Section VI.

II. Related Work

K-connectivity is a widely studied WSN feature spanning topology control, *k*-connected dominated set construction, controlled deployment, relay node placement, repair/maintenance, and level of *k*-connectivity level detection. In the following, we briefly discuss the limitations of these approaches that we plan to address through our DKM approach.

Topology Control: Topology control strategies such as [8], [9], [15], [17], [18] are based on adjusting the radio transmission power of sensor nodes to those levels that provide *k*-connectivity within the closest neighborhood, and by extension the global *k*-connectivity property. The utility of these techniques is limited to the cases where the maximal transmission level conditions for *k*-connectivity hold.

Controlled Deployment: *k*-connectivity can also be attained by manual deployment of the entire network by deploying nodes at pre-computed locations [5], [6]. Unfortunately, manual placement of hundreds or thousands of sensor nodes is prohibitively complex and expensive.

k-*Dominated Sets Construction:* Techniques for constructing the *k*-dominated connected sets assume that the establishment of *k*-connectivity in the considered topology is possible [10], [11]. Therefore, the goal is to find *k*-connected dominated sets consisting of a minimal number of sensor nodes, from among already deployed sensor nodes. This analysis approach cannot be applied directly for restoring *k*-connectivity if the property does not appear in the topology.

k-*Connectivity Detection:* *k*-connectivity detection algorithms [13], [14] aim to determine the attainable *k*-connectivity for given topology. These techniques, while providing useful information about the state of the network, do not offer dynamic solutions for adjusting the topology to restore the desired global *k*-connectivity property.

Relay Nodes Placement: Relay nodes placement algorithms [16], [19]–[22] do not target providing actual *k*-connectivity. These strategies aim at assuring that each sensor node is connected to at least k reliable relay nodes and do not address *k*-connectivity repair strategies.

WSN Maintenance: The repair/maintenance class of strategies directly relate to our work. Current techniques follow two distinct approaches: (a) Steinerization [1], [12], and (b) minimum spanning tree (MST) [2]. In the steinerization approach, the graph edges are sorted in weighting order, where weight corresponds to the number of sensor nodes to deploy in order to connect two sensor nodes. Then, the new graph is created by adding the edges in the increasing order of weight, checking after each added edge if the graph becomes *k*-connected. For the optimization of the number of edges after attaining *k*-connectivity, each edge in reversed order is checked whether it can be removed without impacting the *k*-connectivity. This approach is computation demanding and shows limited scalability although the authors proposed a distributed approach for creating local *k*-connectivity clusters and then joining them on the same principle. Also the number of sensor nodes required for redeployment is very high as each added edge with weight $(w > 0)$ requires adding $w \cdot k + 2 \cdot (k-1)$ sensor nodes. The goal of the MST approach [2] is to decrease the number of resources needed and lower the computation requirements. The optimizations are useful for disconnected graphs. For achieving *k*-connectivity, the authors propose to find an MST of the graph and to place additional $k - 1$ sensor nodes at position of each none leaf sensor node. This approach requires less supplemental resources than [12]. Unfortunately, this approach exploits existing edges only to a low degree resulting in high demand for resources in absolute terms. Although the process of finding the MST is less computation demanding than [12], it is a complex operational procedure. Both techniques [2], [12] do not address the problem of co-located sensor nodes and do not account for maintenance objectives such as targeting shorter routes during network reconfiguration, etc.

There also exist maintenance strategies that do not focus on the establishment of the global *k*-connectivity property but try to prevent/repair the partitioning by deploying additional sensor nodes in endangered regions of the network [23], [24]. Another indirect approach to provide the maintenance is to remove topology irregularities that lead to unbalanced resource depletion throughout the network [25], [26]. Although the approaches may be effective, they do not provide quantifiable fault tolerance guarantees.

III. System Model

Conforming to contemporary WSN models, we assume a WSN consisting of *n* resource constrained sensor nodes and a sink. The sink is a powerful sensor node connected to infrastructure with unlimited energy supply. Hence, it can be considered as a sensor node virtually stacked with an infinite number of backup sensor nodes. Without loss of generality, we assume that the sink is an arbitrary sensor node that is stocked with at least *k - 1* additional backup sensor nodes. The sensor nodes have finite battery energy and usually possess limited processing and storage capabilities. The communication range *R* is limited and fixed for a given deployment. Two neighboring sensor nodes can communicate directly - establish a link - only if their Euclidean distance is smaller than *R* (it is noteworthy that this implies that we consider symmetric/undirected links). This communication dominates the imprint on the energy depletion of the sensor nodes. We model the network as a graph $G = (V, E)$, V is the set of Vertices (nodes), E is the set of Edges (existing links between nodes for the specified *R*). We do not require sensor nodes to know their positions, but available location information (e.g., as in [3]) can positively influence the efficiency of our algorithms. We consider cases where (a) all nodes are static, or (b) a mix of static and nodes of controlled mobility. We assume that sensor nodes know their hop distance to the sink, e.g., based on a shortest path tree routing. N_i refers to the set of sensor nodes that are i hops from the sink ($i \in \{0, 1, .., n\}$). N_0 is the sink (or its equivalent virtual co-located k sensor nodes). We consider sensor nodes to be aware of their 1-hop neighboring sensor nodes, including their position and hop distance to the sink. We do not consider message losses as these are typically handled at the message transport protocol level.

IV. Definitions and Requirements

We present the background concepts that represent the basis of our *k*-connectivity maintenance technique.

Definition 1: A Sensor Node L is called a *closer neighbor* of Sensor Node F if (a) L is a neighbor of F and (b) L has a smaller hop distance to the sink than F.

Lemma 1: If each sensor node has at least *k* closer neighbors then each sensor node has at least *k* independent paths to the sink.

Proof: Consider Sensor Node $F \in N_i$, i.e., placed *i*-hops from the sink. F has at least *k* neighbor nodes $\in N_{i-1}$ that are 1-hop closer to the sink than F itself. That means that F can choose from *k* possible neighbors to send the data to the sink, so at least *k* sensor nodes have to be removed to disconnect F from all of N_{i-2} nodes. For instance, each node $\in N_2$ has at least *k* paths to the sink. Therefore, by induction if F has *k* paths to the N_{i-1} then it also has *k* paths to the sink. ■

Lemma 2: If each sensor node in a WSN has *k* independent paths to the sink, then there exist at least *k* paths between each pair of sensor nodes and as the result the WSN is *k*-connected.

Proof: If each sensor node has *k* independent paths to the sink, then (as we consider undirected graph/network) reverse is also true, meaning that there exist *k* independent paths from the sink to each of the sensor nodes. If each sensor node would send data to each other sensor node over the sink then there exist at least *k* independent paths that can be used for this purpose. As we consider the sink as infinitely stocked node, or at least *k*-stocked node, then the WSN is *k*-connected. ■

Definition 2: If Sensor Node F has an equidistant neighbor H (with same hop distance to the sink) and H has a closer neighbor K that is not a neighbor of F then we call K an *indirect closer neighbor*. We call H a *semi-closer neighbor*. Each equidistant sensor node can provide at most one indirect closer neighbor and only if that indirect closer neighbor is not already provided by other equidistant sensor node. We call the set of all indirect closer neighbors of node F *ICN(F)* and the set of all closer neighbors *CN(F)*.

Definition 3: Support Nodes are any *closer* or *indirectly closer* neighbors. The set of Support Nodes of Node F is represented as $SNS(F) = ICN(F) \cup CN(F)$.

Definition 4: The set of sensor nodes that depend on routing support (towards the sink) of Sensor Node F are termed the Dependent Nodes Set of F (*DNS(F)*).

Theorem 1: If each sensor node has *k* or more support nodes ($|SNS| \geq k$), then the WSN is *k*-connected.

Proof: If a sensor node F has *j* (less than *k*) closer neighbors and at least $k - j$ semi-closer neighbors each of them providing one indirect closer neighbor, then $|SNS| \geq k$. For $|SNS| \geq k$ there has to be at least k nodes $\in N_{i-1}$ to be removed in order to disconnect F from nodes $\in N_{i-2}$, i.e., from the sink. Following the proof of Lemma 1 and concluding from proof of Lemma 2 if each sensor node $\in N_i$ has a $|SNS| \geq k$, then the WSN is *k*-connected. ■

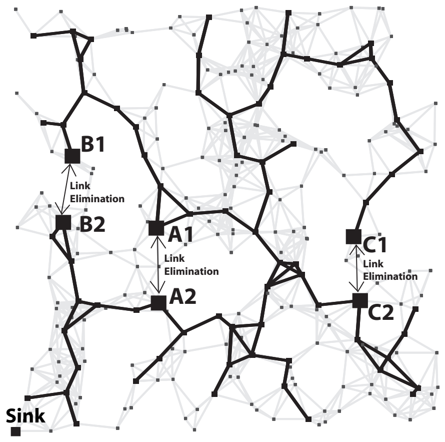

Fig. 1. Single link elimination

In order to generalize the *k*-connectivity maintenance of our solution, we define r_F as the effective redundancy factor of Node F. For classical *k*-connectivity approaches, r_F is equal to the number of sensor nodes stored at the location of F. We assume a more generic approach where r_F represents a variety of resource redundancy types (but only single type at the time) such as additional batteries, redundant hardware components on the same node, etc. Accordingly, we consider that the values of r_F are not necessarily integer but real values.

Approach Requirements: In this work, we use the proposed definitions, lemmas and Theorem 1 to design an efficient *k*-maintenance technique. Our main requirement is to achieve this goal with minimal supplemental resources measured by the additional required r_F to maintain *k*-maintenance. Our second requirement is to maintain shorter routes from sensor nodes to the sink in order to avoid degrading the WSN responsiveness. In order to better illustrate the problem, Figure 1 presents a WSN with marked Minimum Connected Dominated Set (MCDS). The *k*-connectivity in this case is provided when each of the sensor nodes belonging to MCDS is stocked with k - 1 additional sensor nodes. As can be easily observed, although *k*-connectivity is assured, a failure of even less than k nodes from among those connecting pairs (A1; A2), (B1; B2) or (C1; C2) may result in overly extended routing paths from A1, B1 or C1 to the sink.

V. *k*-Connectivity Maintenance

After an overview of our approach, we detail a technique for localized verification of the necessary conditions to provide *k*-connectivity. Subsequently, we describe how to deploy a minimal number of resources in order to restore the *k*-connectivity property. We also present an optimization mechanism to eliminate co-located links and discuss the self-sustainablity aspect of the installed *k*-connectivity.

A. A Guide through our Approach

The key idea behind our approach is based on Theorem 1. If each sensor node has at least k support nodes (i.e., $|SNS| \geq k$)

where $SNS(F)$ not necessarily assumes an integer value, as it is the sum of redundancy factors r of its members. Then each node has at least k alternative/disjoint paths to reach the sink, leading to that the WSN is *k*-connected. Accordingly, sensor nodes require only localized knowledge to verify if the condition ($|SNS| \geq k$) is met. Every sensor node, for which $|SNS| < k$ should check if all of its supporting nodes themselves fulfil this condition. If at least one member of SNS does not meet the condition, then the node waits until all of its SNS members themselves have at least k support nodes. At that point each sensor node can locally try to resolve the situation using a min-max approach. A node D sends the preliminary request for adding k - z additional resources (r_D) to all of their $SNS(D)$ members, where $z = |SNS(D)|$. The $SNS(D)$ members collect preliminary requests from all of their children and calculate the minimum. The minimum is sent as response to all the children that sent a request. Each child selects from all the responses maximal value and sends the commit request to the parent that submitted the maximum response. After each such iteration at least one, but generally most of the sensor nodes will get enough additional support nodes to meet the condition of having k alternative paths.

B. Localized Estimation of k-Connectivity

The first step in DKM is for each Sensor Node X to calculate the $|SNS(X)|$. If $|SNS(X)| \geq k$ then we call X a Resolved Node (RN) otherwise Unresolved Node (UN). *Algorithm 1* describes our distributed method for calculating $|SNS(X)|$. X checks for each of its neighbors, the routing induced hop distance to the sink. If the neighbor is placed farther from the sink than X itself, then such a sensor node is ignored. Each neighbor that is placed closer to the sink than X increases $|SNS(X)|$ by the resource redundancy factor $r_{neighor}$ of the considered neighbor's location. The neighbor identifier (ID) is also added to a temporary set called *closerNeigh* to exclude considering this node as indirect closer neighbor.

If a neighboring Sensor Node Y is equidistant to X then the neighbors of Y are further investigated. If Y possesses neighbors closer than X to the sink and not yet included in *closer-Neigh* then these neighbors are added to *closerNeigh* and they contribute to $|SNS(X)|$ by the minimum of r_Y and the sum of the marked closer Y's resource redundancy factors.

After Algorithm 1 nodes can identify if they are resolved (i.e., the resources of closer neighbors are higher than k) or not. Unresolved nodes need to execute the resolving algorithm. The message exchange cost of executing Algorithm 1 per sensor node depends on allowed message size but, in general, is limited to two messages. In order to execute the algorithm, it is enough for each sensor node to inform its neighbors about its ID, hop distance to sink and list of IDs of its neighbors along with their resources whose hop distance to the sink is smaller than its own. This process necessarily happens in two steps, each requiring sending a single message. First, each sensor node sends only its ID and hop distance. This broadcast message is received by all direct neighbors in a single transmission. Each sensor node receives the messages from its neighbors and creates the list of neighbors whose hop distance to the sink is smaller than its own. Next, each sensor node sends a second message containing the constructed list. After receiving this second message each sensor node has sufficient information to perform Algorithm 1.

Algorithm 1 Local Connectivity Resolution at each node

```
function estimateLocalConnectivity(curNode) : float
var float closerResources = 0;
var set closerNeigh = $\emptyset$;
for all sn in curNode.Neighbors do
    if sn.hop $<$ curNode.hop then
        closerNeigh $\leftarrow$ closerNeigh $\cup$ sn; //add to SNS
        closerResources += sn.resources;
    else if sn.hop $==$ curNode.hop then
        var float indirectResources = 0;
        for all snn in sn.Neighbors do
            if snn.hop $<$ curNode.hop and snn $\notin$ closerNeigh then
                closerNeigh $\leftarrow$ closerNeigh $\cup$ snn; //add to SNS
                indirectResources += min(sn.resources, snn.resources);
                if indirectResources $>=$ sn.resources then
                    break; //indirect neighbors cannot provide more resources
                    than direct ones
                end if
            end if
        end for
        closerNeigh $\leftarrow$ closerNeigh $\cup$ sn; //add to SNS
        closerResources += min(indirectResources, sn.resources);
    end if
end for
return closerNeighCount;
```

C. Resolving Nodes

A sensor node, say F, initiates the resolving process when the Local Connectivity Resolution Algorithm (Algorithm 1) returns a value lower than the desired k. This means that F needs to inform current members of SNS(F) about the required resources to itself become a resolved node. Our resolving process utilizes voting. First, the nodes whose complete SNS is resolved, such as F, send the demand for the resources as a vote request (vote representing the amount of missing resources r_F). The SNS nodes gather the votes and sum them. Simultaneously, they additionally calculate the minimal value of votes (the minimal resources required to resolve at least one node). The SNS nodes send response to their DNS (from Definition 4, $F \in DNS$). Each DNS node (e.g., F) selects from the responses (only from closer neighbors) the sender, which responded with highest accumulated votes. The highest accumulated vote indicates that this sender is the one that is shared by most of sensor nodes and therefore stocking it with additional resources will be profitable for increasing local connectivity of unresolved nodes. This sender receives a request for stocking additional resources equal to the minimal vote that it received during voting. This measure guarantees that, at each step, at least one sensor node becomes resolved to progress the process and it minimizes chances of overstocking the nodes.

We now detail the resolving algorithms (Algorithm 2 & 3) as two threads running at each sensor node (the actual implementation does not require threaded execution, we just use this abstraction for a better explanation of the algorithm). Each sensor node can be a member of a DNS or a SNS of some other sensor nodes. Algorithm 2 describes the sensor node behavior while acting as a member of a DNS and Algorithm 3 the situation when the sensor node is a member of a SNS.

Algorithm 2 Resolve Node: As member of DNS

```
1: function resolveNodeThread(curNode, k)
2: while not checkIfSNSResolved(curNode, k) do
3:     waitForNeighborNotification(); //wait until SNS is resolved, does not
       involve sending any messages
4: end while
5: while estimateLocalConnectivity(curNode) $<$ k do
6:     var vote;
7:     vote.value = k - estimateLocalConnectivity(curNode);
8:     vote.sender = curNode;
9:     for all sn in closerNeigh do
10:        sendVote(sn, vote); //ask for resources
11:    end for
12:    var maxResponse;
13:    for all response in collectResponses() do
14:        if response.sender.hop $<$ curNode.hop and
           response.value $>$ maxResponse then
15:            maxResponse = response; //choose best bidder
16:        end if
17:    end for
18:    sendSelection(maxResponse.sender);
19: end while
20: notifyNeighbors("curNode resolved");
```

Before F can start resolving its status, it checks whether all of the members of its SNS are already resolved (Algorithm 2, L. 2). Two situations are possible: (a) All members of SNS(F) are resolved as a result of already available resources, (b) some sensor nodes belonging to SNS(F) are not yet resolved and F has to wait until they resolve their status. In Situation (a), the algorithm may execute in parallel in several regions of the WSN. Not resolved nodes notify their neighbors when their status changes to resolved (Algorithm 2, L. 20), so the sensor nodes whose SNS(F) is not completely resolved may easily detect changes (Algorithm 2, L. 3). When all SNS(F) are resolved F may start sending its votes. F calculates its vote by subtracting the value of its local connectivity from the desired *k* (Algorithm 2, L. 7). The vote is sent only to closer neighbors as only those nodes will be considered for adding resources. SNS(F) members collect the votes (Algorithm 3, L. 3) and calculate their sum and value of the minimal vote (Algorithm 3, L. 6 - 7). After collecting all votes SNS(F) nodes send the response to all of the voters (including F) (Algorithm 3, L. 10 - 12). F collects the responds and selects the responder with the highest sum of votes (Algorithm 2, L. 13 - 17). Then, F sends to this responder a request for adding additional resources. SNS(F) members wait for *maxDelay* time for selections from their DNS members. Once, they receive them, they increase their r for the value of the minimal vote received in the voting step (Algorithm 3, L. 13 - 15). After SNS(F) nodes add resources, F re-evaluates its condition (Algorithm 2, Line: 5).

The message cost of each iteration (of the resolving node's algorithm) requires from DNS nodes the sending of two messages (vote and selection messages) and from SNS nodes only a single response message. The number of iterations is limited and depends on the discrepancy between the current and desired connectivity levels. In case of *k*-connectivity, at each iteration the discrepancy is reduced by at least 1. Additionally, the algorithm chooses for stocking those nodes whose stock increases the benefits for possibly many unresolved nodes. Pragmatically, *k* is generally a low number and the maintenance action is required sporadically, the number of message transmissions required for resolving nodes is small and limited.

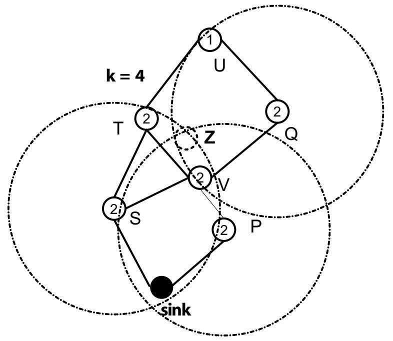

Fig. 2. Single Link Elimination

Algorithm 3 Resolve Node: As member of SNS

```
1: function voterDeamonThread(curNode, k)
2: while true do
3:     var votes = collectVotes()
4:     var response;
5:     for all vote in votes do
6:         response.value += vote.value;
7:         response.minVote = min(response.minVote, vote.val);
8:     end for
9:     response.sender = curNode;
10:    for all voter in voters do
11:        sendResponse(vote.sender, response) //actually only single message
           is sent as all recipients are direct neighbors
12:    end for//wait maxDelay for selections
13:    if collectSelections(maxDelay) != $\emptyset$ then
14:        curNode.stock += response.minVote;
15:    end if
16: end while
```

D. Avoidance of Bottleneck (Overlapping) Links

When the SNS of a sensor node (e.g., Q in Fig. 2) consists of a single node (V) then it is necessary to add an additional sensor node (Z) (extend the SNS) that is not co-located/stocked with V. If V was stocked with additional $k - 1$ sensor nodes, then although *k*-connectivity would be provided, a single failure related to location (e.g., sensor nodes burned, trampled under larger objects, stolen, etc.) could cause immediate network disconnection. For this purpose, Z should allow to forward data to the sink independent form the existing single link (over V). Z itself should become *k*-connected. In order to ensure *k*-connectivity of Z, it is enough to select a new location that is intersection of enough of SNS(V) = {S, P} such that their summed up stocks are at least equal to *k*. From the possible location, the one that is placed farthest from V but still in Q range should be selected.

E. Optimizing Required Resources

A sensor node, say X, may be over-provisioned following Algorithm 3, i.e., if r_X is larger than required by all nodes $\in DNS(X)$ for maintaining *k*-connectivity. Subsequently, we identify resource over-provisioning in order to ensure minimal cost maintenance. The calculation of over-provisioning is conducted as follows:

1) Each sensor node sends to its SNS nodes the value indicating the difference between its local connectivity and desired k (overK).
2) Each SNS node collects the overK values and calculates their minimum value ($overK_{min}$). $overK_{min}$ indicates the amount of over-provisioned resources whose removal would not cause a drop in local connectivity in any of DNS nodes of SNS below k. Each SNS node responds by sending to its corresponding DNSs a value equal to calculated $overK_{min}$ or number of its own stoked resources less one.
3) DNS nodes receiving a response from their SNS select the sensor node (SNS_B) that sent the highest $overK_{min}$. If two sensor nodes send equal $overK_{min}$ then the node identifier (ID) breaks the tie. Each DNS node sends the ID of its SNS_B back to its SNS.
4) If all received IDs are equal to the ID of receiving node than this sensor node can remove the proposed number of stocked nodes.

In case of limited resources, the sensor nodes after calculating missing resources could execute a bidding process [4], where the weight of the bid corresponds to the closeness to sink (to prevent partitioning close to sink, where it could disconnect much larger part of the network) or to the $|DNS|$ of bidding sensor node.

F. Self-Sustaining k*-Connectivity*

If the scenario of supplying the WSN with additional resources (e.g., batteries, sensor nodes, etc) is not possible but the network contains mobile elements, the mobility can be used for topology maintenance. We propose to rearrange sensor nodes in order to sustain the *k*-connectivity in face of failures. For this purpose, the WSN needs to identify sensor nodes whose repositioning does not impair the *k*-connectivity property. Each of the sensor nodes that do not belong to SNS of any other node are called leaf nodes and can safely move to the locations selected by DKM. Each removal of leaf nodes creates potentially new leaf nodes. In Fig. 2, Sensor Node U is a leave node as it does not belong to any SNS. When U gets removed, then as consequence sensor nodes T and Q become leave nodes, as they where members only of SNS(U). The leave sensor nodes can be iteratively removed and used to repair other parts of the network until remaining topology becomes *k*-connected. It is evident that as this process progresses the deployment area accordingly shrinks. Therefore, the user should balance the priorities between sensing coverage and fault tolerance. If sensing coverage is given higher priority then there should be assigned sensor nodes whose removal is not permitted. This however may impair the self-sustainability capacity of the algorithm as it may run out of resources.

G. Graph Balancing

A further scenario is where the end-to-end traffic is not bound to the sink but is also desired between sensor nodes. We

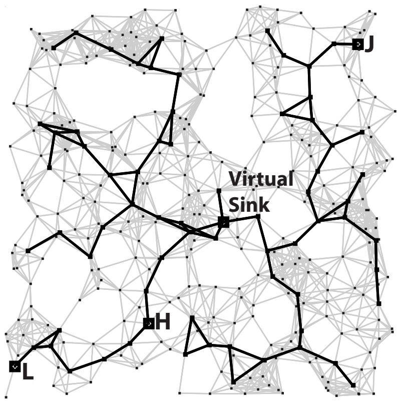

Fig. 3. Virtual Sink Selection

show that DKM also works in such scenarios. For this purpose we revert to the original concept presented in [2], with a slight modification. Instead of creating MST, we propose MCDS as this results in selecting fewer sensor nodes, which needs to be stocked but keeps the crucial property of MST for assuring *k*-connectivity. Fig. 3 shows an example of WSN with marked sub optimal MCDS using the algorithm described in [7]. We use this algorithm as it allows distributed execution, but also further techniques may equally be used to select MCDS nodes. Having established the MCDS, we refrain from stocking k - 1 sensor nodes on the location of MCDS nodes as proposed in [2]. Instead, we propose to select a virtual sink. A virtual sink is a sensor node belonging to MCDS, which assumes role of the sink under the condition of having stocked at least k sensor nodes. The position of the virtual sink should be placed possibly close to the geographic center of the deployment area, so that the average hop distance to this sink is kept minimal (in uniform networks), and so that in case of failures, when the data has to be rerouted through the virtual sink the extended routes will be kept possibly short. After finding the proper virtual sink DKM proceeds as in the real scenario.

We propose the following strategy to establish the position of the virtual sink. We allow the sensor node with the highest ID (Fig. 3 sensor node H) to create the routing spanning tree among the members of the MCDS set. As a result the members of the MCDS set establish their hop distance to H. The sensor node with the highest hop distance (Fig. 3 Sensor Node J) in uniform network is placed at the border of the deployment area. We allow now J also to start routing spanning tree construction (over MCDS) so that the sensor nodes can establish their hop distance to J. The sensor node with the highest hop distance to J we mark as L (Fig. 3). L also starts spanning tree construction among MCDS nodes. After this step each sensor node has established the hop distance to both opposing border sensor nodes J and L. In order to find the center of the MCDS, we look for the sensor node, whose value $(hop(J) - hop(L))^2$ is the lowest. This sensor node we select to function as a Virtual Sink. Although we execute the spanning tree construction algorithm three times, the number of the messages is kept low as in construction of the tree participate only the MCDS nodes. Therefore, for the whole process each MCDS node sends only three messages. It is noteworthy that as our simulations have showed, the selection of sink location at the center of the deployment area does not increase or decrease the demand for the resources. Establishing the virtual sink at a central location only serves the purpose of decreasing (only in case of failures) hop distances in case of sensor node to sensor node communication scenarios. For the construction cost of MCDS we refer the reader to [7].

VI. Performance Evaluation

A. Evaluation Settings

We consider a WSN network consisting of sensor nodes, with a communication range of $R = 3m$, deployed over the area of 30m $\times$ 30m. We gradually change the number of deployed sensor nodes from 230 (sparse scenario - low initial *k*-connectivity) to 750 (dense scenario - high initial *k*-connectivity) in order to measure the utility of DKM under different network densities and initial conditions. For each of these settings, we also vary the desired k from 2 to 10. We choose these values to cover a wide range of real deployments and future application requirements. We performed an additional study to measure the effectiveness of DKM under varying initial conditions induced by changing communication range. The number of deployed sensor nodes was kept at 350 sensor nodes and communication ranged was varied from 2.5m to 10m. Also in this case the desired *k*-connectivity was varied from 2 to 10. We executed all sets of simulation for two situations. In Situation 1, with no localization available the single link elimination step was skipped. In Situation 2, we additionally handled bottleneck overlapping links and resource over-provisioning. We use our stand-alone implementation to simulate the DKM efficiency.

We compare DKM to the solution postulated in [2] based on MST/MCDS approach and refer the readers to [2] for comparison with [1]. The general conclusion from the comparison follows that the approach in [1] requires substantially more sensor nodes to assure the *k*-connectivity. As the driving metric we apply to measure the effectiveness of DKM is the amount of resources required to be added in order to assure desired *k*-connectivity. For this purpose, we execute DKM and calculate the percentage increase in number of sensor nodes in reference to the originally deployed network. We perform the same operation for the MST/MCDS based algorithm.

B. Evaluation Results

Fig. 4 presents the demand for resources using the DKM and MST/MCSD algorithms in function of the desired *k*-connectivity. For the sparse topology (230 sensor nodes) the trends of both algorithms show linear characteristics separated by a margin starting at approximately 30% sensor nodes for k = 3 and dropping to around 10% sensor nodes for k = 10. In case of denser initial deployments (500 and 750 sensor nodes), DKM changes to an exponential trend but with small enough exponent to stay well below the linear trend of MST/MCSD based solution for the whole spectrum of tested *k*-connectivity. For higher densities, DKM requires around 50% less of initial sensor nodes to add. This is due to the fact that an increase in density does not translate to reducing the number of members of the MST/MCDS sets as compared to increasing the initial connectivity of sensor nodes. DKM exploits this fact and tries to reuse the already deployed sensor nodes in order to reduce the demand for additional resources.

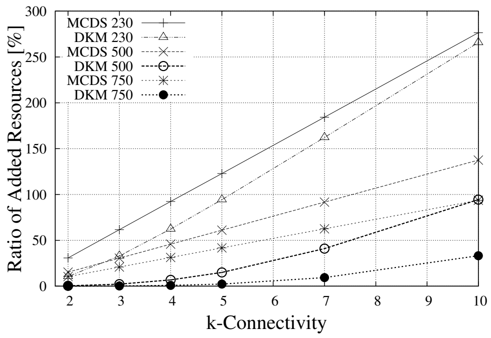

Fig. 4. Ratio of added resources in function of required *k*-connectivity

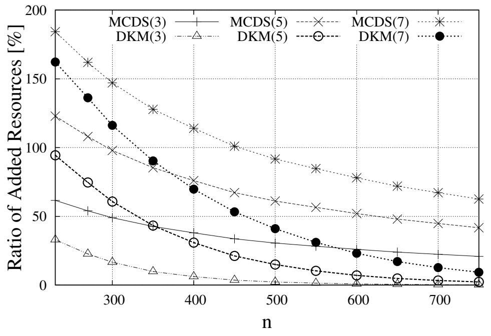

Fig. 5. Ratio of added resources in function of number of deployed nodes

Fig. 5 presents the demand for resources for fixed value of k and varying initial conditions. For each value of n DKM outperforms the reference algorithm. The discrepancy in performance for benefit of DKM rises with the rising number of initially deployed sensor nodes *n*. The initial lower discrepancy can be explained from the reasoning that in initially sparse networks the DKM operates similar to the MCDS/MST algorithm. Similarity in the discrepancy arises because in sparse networks the spanning routing tree, which DKM uses, has only few alternative parents for each sensor node. As the density of network rises with higher number of deployed sensor nodes DKM benefits from the availability of multitude of supporting nodes in the routing tree.

Another important property of WSN is the wireless communication range *R*. Fig. 6 presents the ratio of added resources in function of *R* for a fixed number of deployed sensor nodes n = 350. For low values of *R* the network is sparse DKM outperforms MCDS/MST based solutions. The performance advantage of DKM rises with *R* but only up to approximately $R = 4m$ and then slowly converges close to 0%. For high value of *R* (10m) in the considered scenario maximal hop distance is very low and that explains the outperformance of DKM, but also MCDS/MST involves only few additional nodes, as sensor nodes have a high connectivity degree.

Finally, we evaluate the established average local *k*-connectivity indicator using Algorithm 1. For each originally deployed sensor node, we calculated $|SNS|$ before and after

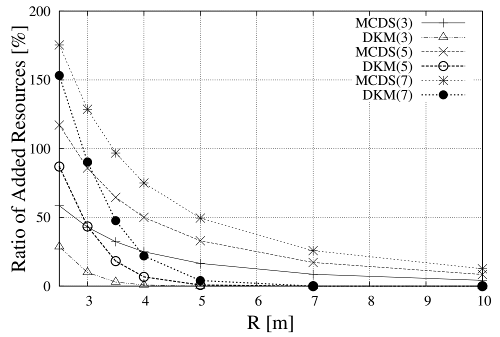

Fig. 6. Ratio of added resources in function of communication range

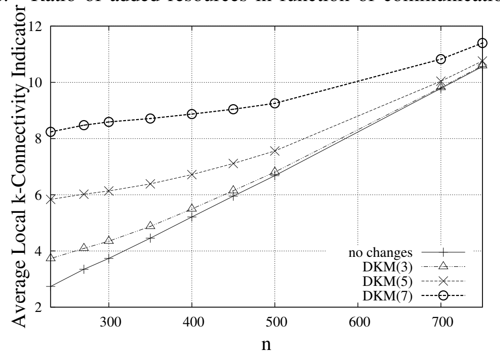

Fig. 7. Average local k-connectivity indicator

executing DKM. Afterwards, we calculate the average value for both cases. Fig. 7 shows the average local k-connectivity indicator in function of originally deployed sensor nodes for $k \in \{3, 5, 7\}$. The basic trend for the original topology of WSN (no changes) is consistently linear and shows inherent localized k-connectivity indicator for a network of a given density. If $k = 3$, the trend only initially differs from base trend and for network size n = 400 and denser it nearly mergers. For the low values of k, the inherent k-connectivity is sufficient to provide the requested fault tolerance. Therefore, there is a limited need for additional resources so that the local k-connectivity does not increase. It is different for larger $k \in \{5, 7\}$ where especially in initially sparse topologies there is a low inherent fault tolerance. It is obvious that the average local k-connectivity indicator has to increase to at least the value of k, what explains the initial discrepancy from the base trend. But as can be observed from lines (DKM(5) and DKM(7)) the achieved local k-connectivity indicator is only marginally higher than the requested k, confirming the efficiency of DKM.

VII. Conclusions and Future Work

We have presented DKM, an efficient distributed k-connectivity maintenance technique. While using only local views, DKM is capable of restoring the global k-connectivity property. The presented solution, while not optimal, offers substantially better resource utilization performance to the state of the art. The algorithm also allows the detection of redundant/spare sensor nodes that can be used for performing maintenance. Mobile sensor networks can use this capability for providing self-sustainability.

We plan to further increase the reuse of deployed resources. Currently, DKM considers only 2-hop neighborhood connectivity information for identifying closer and semi-closer neighbors. Increasing the size of immediate neighborhood information to further distances could potentially allow for the identification of more redundancies in the connectivity. Achieving these potential savings is at the cost of a higher algorithm complexity and communication overhead.

References

[1] Bredin, J., et al., "Deploying Sensor Networks with Guaranteed Capacity and Fault Tolerance", In: MobiHoc, pp. 309–319, 2005

[2] Almasaeid, H.M., Kamal, A.E., "On the Minimum k-Connectivity Repair in Wireless Sensor Networks", In: ICC, pp. 1–5, 2009

[3] He, T., et al., "Range-free localization and its impact on large scale sensor networks", In: Trans. on Embedded Computing Sys., vol. 4, pp. 877–906, 2005

[4] Wang, G., et al., Proxy-Based Sensor Deployment for Mobile Sensor Networks, In: MASS, pp. 493–502, 2004

[5] Bai, X., et al., Deploying Four-Connectivity and Full-Coverage Wireless Sensor Networks. In: INFOCOM, pp. 296–300, 2008

[6] Bai, X., et al., Complete optimal deployment patterns for full-coverage and k-connectivity (k=6) wireless sensor networks. In: MobiHoc, pp. 401–410, 2008

[7] Butenko, S., et al., "A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks", In: Recent Developments in Co-operative Control and Optimization, pp. 61–73, 2004

[8] Hajiaghayi, M., et al., "Power optimization in fault-tolerant topology control algorithms for wireless multi-hop networks", In: MobiCom, pp. 300–312, 2003

[9] Bahramgiri, M., et al., "Fault-tolerant and 3-dimensional distributed topology control algorithms in wireless multi-hop networks", In: Wirel. Netw., vol. 12, pp. 179–188, 2006

[10] Dai, F., Wu, J., "On Constructing k-Connected k-Dominating Set in Wireless Networks", In: IPDPS, pp. 81.1-, 2005

[11] Wu, Y., Li, Y., "Construction Algorithms for k-Connected m-Dominating Sets in WSNs", In: MobiHoc, pp. 83–90, 2008

[12] Pu, J., et al., "Fault-tolerant deployment with k-connectivity and partial k-connectivity in sensor networks", In: Wirel. Commun. Mob. Comput., vol. 9, pp. 909–919, 2009

[13] Atay, N., Bayazit, B., "Mobile Wireless Sensor Network Connectivity Repair with K-Redundancy", In: Springer Tracts in Advanced Robotics, pp. 35–49, 2009

[14] Jorgic, M., et al., "Localized detection of k-connectivity in wireless ad hoc, actuator and sensor networks", In: ICCCN, pp. 33–38, 2007

[15] Li, N. Hou, J., "FLSS: A Fault Tolerant Topology Control Algorithm for Wireless Networks", In: MobiCom, pp. 275–286, 2004

[16] Xiaofeng, H., et al., "Fault-tolerant Relay Node Placement in Heterogeneous Wireless Sensor Networks", In: INFOCOM, pp. 1667–1675, 2007

[17] Li, X., et al., "Fault Tolerant Deployment and Topology Control in Wireless Networks", In: MobiHoc, pp. 117–128, 2003

[18] Saha, I., et al., "Distributed Fault-Tolerant Topology Control in Static and Mobile Wireless Sensor Networks", In: COMSWARE, pp. 1–8, 2007

[19] Misra, S., et al., "Constrained Relay Node Placement in Wireless Sensor Networks to Meet Connectivity and Survivability Requirements", In: INFOCOM, pp. 281–285 , 2008

[20] Srinivas, A., et al., "Construction and Maintenance of Wireless Mobile Backbone Networks", In: IEEE/ACM Trans. Netw., vol. 17, pp. 239–252, 2009

[21] Lloyd, E., Xue, G., "Relay Node Placement in Wireless Sensor Networks", In: IEEE Trans. Comput., vol. 56, pp. 134–138, 2007

[22] Weiyi Z., et al., "Fault-Tolerant Relay Node Placement in Wireless Sensor Networks: Problems and Algorithms", In: INFOCOM, pp. 1649–1657, 2007

[23] Szczytowski, P., et al., "LEHP: Localized Energy Hole Profiling in Wireless Sensor Networks", In: ISCC, pp. 100–106, 2010

[24] Wang, G., et al.: Sensor Relocation in Mobile Sensor Networks, In: INFOCOM, pp. 2302–2312, 2005

[25] Szczytowski, P., et al., "TOM: Topology Oriented Maintenance in Sparse Wireless Sensor Networks", In: SECON, pp. 548–556, 2011

[26] Ghosh, A., Boyd, S., "Growing Well-connected Graphs", In: CDC, pp. 6605–6611, 2006