Ad Hoc Networks 44 (2016) 104–117

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

An adaptive, energy-aware and distributed fault-tolerant topology-control algorithm for heterogeneous wireless sensor networks

Fatih Deniz[a,*], Hakki Bagci[a], Ibrahim Korpeoglu[b], Adnan Yazıcı[a]

[a] Department of Computer Engineering, Middle East Technical University, Ankara, Turkey
[b] Department of Computer Engineering, Bilkent University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 2 December 2015
Revised 20 February 2016
Accepted 22 February 2016
Available online 3 March 2016

Keywords:
Topology control
Fault-tolerance
Energy efficiency
Prolonged network lifetime
k-connectivity
Heterogeneous wireless sensor networks

ABSTRACT

This paper introduces an adaptive, energy-aware and distributed fault-tolerant topology-control algorithm, namely the Adaptive Disjoint Path Vector (ADPV) algorithm, for heterogeneous wireless sensor networks. In this heterogeneous model, we have resource-rich supernodes as well as ordinary sensor nodes that are supposed to be connected to the supernodes. Unlike the static alternative Disjoint Path Vector (DPV) algorithm, the focus of ADPV is to secure supernode connectivity in the presence of node failures, and ADPV achieves this goal by dynamically adjusting the sensor nodes' transmission powers. The ADPV algorithm involves two phases: a single initialization phase, which occurs at the beginning, and restoration phases, which are invoked each time the network's supernode connectivity is broken. Restoration phases utilize alternative routes that are computed at the initialization phase by the help of a novel optimization based on the well-known set-packing problem. Through extensive simulations, we demonstrate that ADPV is superior in preserving supernode connectivity. In particular, ADPV achieves this goal up to a failure of 95% of the sensor nodes; while the performance of DPV is limited to 5%. In turn, by our adaptive algorithm, we obtain a two-fold increase in supernode-connected lifetimes compared to DPV algorithm.

1. Introduction

Wireless sensor networks (WSNs) are typically composed of large numbers of tiny sensor nodes that are capable of sensing, processing and transmitting data over wireless channels. Such networks can be used in numerous fields, such as battlefield surveillance [1–3], environmental monitoring [4–6] and traffic control [7–9]. Sensor nodes collaborate in a distributed, autonomous and self-organized manner to accomplish a certain task, usually in an environment with no infrastructure.

Sensor nodes in WSNs should be low-cost and should have small form-factor. This restricts sensor nodes in many ways as they have limited energy, short transmission range, relatively slow CPU and small memory. These limitations bring out many challenges unique to WSNs, such as very low power consumption. Since sensor nodes are battery powered and these batteries are usually not rechargeable, coming up with solutions that reduce energy consumption and prolong network lifetime are very important. Numerous studies address this problem [10–13] in literature. According to Li and Mohapatra [14], 90% of a sensor network's energy is still available after first node

* Corresponding author. Tel.: +90 5531866846.
E-mail addresses: fatih.deniz@tcmb.gov.tr, fatihdeniz@gmail.com (F. Deniz), hakkibagci@gmail.com (H. Bagci), korpe@cs.bilkent.edu.tr (I. Korpeoglu), yazici@ceng.metu.edu.tr (A. Yazıcı).

http://dx.doi.org/10.1016/j.adhoc.2016.02.018
1570-8705/

dies. Despite this substantial amount of remaining energy, the existence of highly-loaded and bottleneck nodes cause early network demise. There are numerous studies that address balancing energy consumption among nodes to ensure that all nodes will run out of energy at about the same time [15]. Low-energy Adaptive Clustering Hierarchy (LEACH) [16] is a well-known early study that uses dynamic transmission ranges to better balance the load and prolong network lifetime. There are also some recent studies which re-establish lost connectivity by adjusting transmission ranges. CoRAD [17] and RESP [32] can be listed as some of those studies. With the recent developments in the hardware of WSNs, dynamic transmission range assignment has become even more effective [18].

Fault-tolerance is another critical issue in WSNs. Due to the error-prone nature of wireless communication, links may fail, packets can get corrupted or congestion may occur [19,20]. There are also factors that cause long-term faults in sensor nodes, such as energy depletion, hardware failure, link breaks, malicious attacks. Multi-hop communication multiplies the chances of faulty incidents for a packet stream traveling from a source to a destination. Therefore, fault-tolerance methods, including fault-tolerant topology control, are essential for improving WSN reliability as well as network lifetime.

As stated by Liu et al. [21], most existing works on fault-tolerant topology-control aim to obtain k-vertex connectivity between any two sensor nodes, where the topology is guaranteed to remain connected until the failure of the kth sensor node.

In this study, the focus is on two-tiered heterogeneous WSNs, where the network consists of two different types of nodes: resource-rich supernodes and simple sensor nodes with limited battery power. In this network model, sensor nodes are connected to the set of supernodes via multi-hop paths. To reflect this asymmetry, [22] proposes k-vertex supernode connectivity, where each sensor is connected to at least one supernode by k vertex-disjoint paths. In such topologies, the sensor nodes remain connected to the supernodes as long as at most $k-1$ sensor nodes fail.

Most studies on fault-tolerance propose static solutions, that is, they do not adapt the topology to the changing network conditions. Bagci et al. [23] propose a static algorithm called the Disjoint Path Vector (DPV) to optimize total transmission power for a given k-vertex supernode-connected network. That study does not consider residual battery energy and disregards the unbalanced load distribution on sensor nodes. As a result, k-vertex supernode connectivity is achieved but may not be preserved for a sufficient amount of time.

In this study, we propose a novel adaptive and distributed topology-control algorithm, Adaptive Disjoint Path Vector (ADPV), which efficiently constructs a k-vertex supernode-connected network topology and adapts the topology to node failures, which in turn increases network lifetime. The contribution is two-fold. First, the residual battery power levels of individual sensor nodes are considered to prolong k-vertex supernode connectivity. Second, an adaptive solution is proposed to restore, if necessary, k-vertex supernode connectivity after a node failure.

The remainder of the paper is organized as follows: Section 2 gives some background information and discusses the related studies. In Section 3, we present our proposed adaptive topology-control solution. The results for simulation experiments are presented in Section 4. Finally, Section 5 concludes the paper.

2. Related work

In this section, we give a brief overview of some of the prominent recent work addressing fault-tolerance, connectivity restoration and heterogeneity in WSNs. We also give a brief overview of the DPV algorithm [23].

Fault-tolerance techniques can be categorized into four [24]: prevention, detection, isolation, and recovery. *Prevention* attains network connectivity and establishes redundant links/nodes when necessary. *Detection* monitors traffic and sends alerts when any indication of fault happens, such as a decrease in packet delivery rate, which would imply a packet loss, interruption, or delay. *Isolation* diagnoses and identifies the alert. As for *recovery*, after detecting and identifying the fault, the system should be able to recover in either a centralized or distributed manner. Note that due to the nature of WSNs it is essential for the recovery scheme to be a distributed method.

The replication and redundancy of components prone to failure is the most commonly used method for fault prevention and recovery [21]. For instance, if some nodes have problems and fail to sense the environment, the redundant nodes in the vicinity can still provide data. Keeping redundant links or multiple paths also provides fault-tolerance when some communication links are broken due to node failures or communication errors.

2.1. Connectivity restoration in WSNs

There are three approaches to connectivity restoration in WSNs: mobile node relocation, relay node placement and topology-control via transmission range adjustment. In the first approach, as the nodes are mobile, the main idea is to reposition the existing alive nodes to restore connectivity. One example of this method is PADRA, developed by Akkaya et al. [25]. In this approach, each node chooses one of its neighbors to be the failure handler, which will start recovery if the node dies. The restoration process only occurs if the entire network gets disconnected, in which case the closest node that can take the dead node's place is relocated to that position.

In the relay node placement approach [26–31], the objective is to place a minimum number of relay nodes in a region where sensor nodes are randomly deployed so that the resulting network topology is fault-tolerant.

These first two approaches, that is, mobile node relocation and relay node placement, may not be practical in real-world scenarios because sensor nodes are often deployed in remote and inhospitable regions with harsh environments that render manual node placement or relocation infeasible. Note that due to the dynamic nature of WSNs, node placement and/or relocation must be repeated periodically. In addition, these approaches require overall

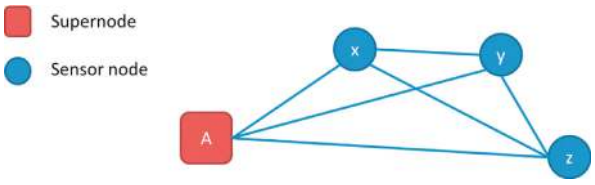

Fig. 1. Initial network.

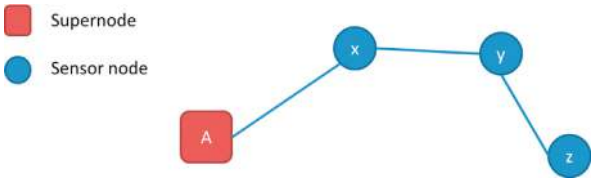

Fig. 2. Optimized network.

network information, something that is also not suitable for most real-world applications.

As a remedy to the problems discussed above, topology-control emerges as a third approach for connectivity restoration. In this approach, the topology is controlled by adjusting the sensor nodes' transmission ranges. One example of this method is RESP [32], which is an energy-aware topology-control algorithm that ensures k-edge connectivity for flat networks. The RESP algorithm assumes sensor nodes are aware of their location information via GPS or other localization techniques, and periodically updates the network topology to adapt to sensor nodes' residual battery power levels. Because of ensuring k-edge connectivity, but not k-vertex connectivity, RESP cannot keep the network connected up to $k-1$ node failures. Another recent approach, Energy-harvesting Heterogeneous WSN (EHWSN) [33], also aims to preserve k-vertex supernode-connectivity for heterogeneous WSNs. EHWSN is a centralized approach and ignores residual battery power levels, therefore not scalable and not energy-aware.

2.2. DPV algorithm

The aim of the DPV algorithm [23] is to minimize the total transmission power of a WSN while maintaining k-vertex disjoint paths from each sensor node to the set of supernodes. The DPV algorithm gets a k-vertex supernode-connected network topology as an input and generates a subnetwork consisting of the same set of sensors but fewer connections. The output of the DPV algorithm is a total transmission power optimized and k-vertex supernode-connected network topology. Consider the example topology given in Fig. 1, which consists of one supernode and three sensor nodes. When the aim is to provide one-vertex supernode connectivity, DPV removes three edges and optimizes the given network topology, as in Fig. 2. The main contribution of DPV is its efficiency in computing such network topologies. The DPV algorithm requires $O(n\Delta^2)$ message transmissions, whereas the best alternative [22] incurs $O(\Delta^5)$ messages, where n is the number of sensor nodes and Δ refers to the maximum degree of a sensor node. Note that we assume a dense network, where Δ is sufficiently large. The DPV algorithm consists of five main stages:

1. Collecting path information and calculating disjoint paths,
2. Calculating the set of required neighbors,
3. Notifying the nodes in the disjoint paths and updating the required neighbors,
4. Removing the non-required neighbors and
5. Reducing the power level to a point sufficient only to reach the farthest required neighbor.

2.3. Power consumption model

Our ADPV algorithm aims at prolonging network lifetime, and thus it should first model the amount of time until the battery powers of the sensor nodes are depleted. The ADPV algorithm uses a well-known power consumption model, proposed by Heinzelman et al. [34,35]. This approach is based on the observation that the main factor in WSN power consumption is data communication, which consists of two factors: data transmission and data reception. In this model, the power to transmit a bit to a distance of d is

$$P_t(d) = \alpha_1 + \alpha_2 \times d^n, \tag{1}$$

where α_1 and α_2 are parameters that depend on the transmitter circuitry, and n is the path loss exponent for the environment, which often has a value between 2 and 4. In our power consumption model, α_1, α_2, and n are assumed to be 50 nJ/bit, $100\,pJ/bit/m^2$ and 2, respectively.

In our model, the energy consumption for data reception is a constant value per bit. We represent this constant with β and assume it equals 50 nJ/bit.

For our experiments, we assume all sensor nodes are sensing the environment and generating traffic at a fixed rate. We also assume that data aggregation is applied and that all nodes on a path carry the same load. Therefore, total power consumption for receiving a bit and transferring it to the next hop equals:

$$P_f(d) = \beta + \alpha_1 + \alpha_2 \times d^n. \tag{2}$$

If the residual battery energy level of sensor node i is denoted as e_i, then the lifetime of node i equals:

$$l_i = e_i/((r_{ri} \times \beta) + (r_{ri} + r_{gi}) \times (\alpha_1 + \alpha_2 \times d_i^n)), \tag{3}$$

where r_{ri} is the incoming data rate to node i, r_{gi} is the data rate generated in node i and d_i is the transmission range.

3. Adaptive disjoint path vector algorithm

In this section, we present our novel adaptive and distributed algorithm, ADPV, which aims to construct and maintain a k-vertex supernode-connected topology to prolong the k-vertex supernode-connected lifetime of the network. The ADPV algorithm controls the topology by adjusting the transmission ranges of sensor nodes, and to comply with real-life situations it considers node failures. The algorithm requires only one-hop neighborhood information and constructs the network topology by a series of message exchanges.

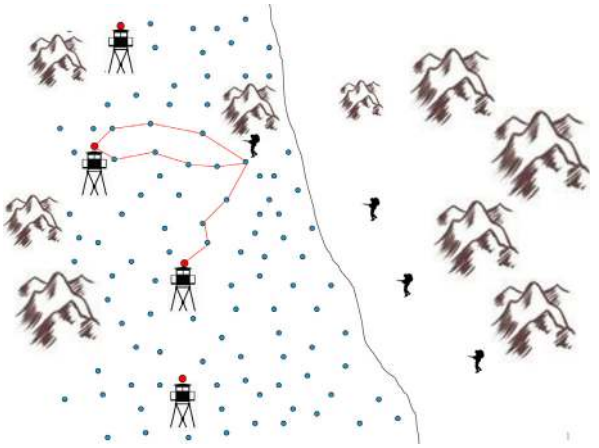

Fig. 3. Sample scenario.

The ADPV algorithm consists of two phases: initialization and restoration. It collects necessary information and builds an initial topology during the initialization phase. Whenever a node failure breaks k-vertex supernode connectivity, ADPV restores connectivity within the restoration phase. Similar to DPV, ADPV utilizes disjoint paths and within each restoration phase each sensor node decides whether or not to change its disjoint paths. At the end of each restoration phase, sensor nodes' transmission ranges are adjusted according to the intended topology. The main differences between ADPV and DPV are as follows:

- ADPV is an adaptive approach, adapting to node failures and remaining energy levels, whereas DPV is a static one.
- ADPV considers residual battery power levels of sensor nodes, therefore it is an energy-aware solution. DPV on the other hand ignores sensor nodes' remaining energy levels.
- ADPV balances energy consumption and optimizes the lifetime of disjoint paths, as opposed to DPV, which optimizes the total transmission power of sensor nodes.
- ADPV significantly prolongs both one-vertex and k-vertex supernode-connected lifetimes of the network with its solutions for restoration path selection, k-vertex supernode-connectivity verification and connectivity restoration.

3.1. Network model

Consider a mission critical border surveillance system that is integrated with a two-tiered heterogeneous wireless sensor network. In this network, there are supernodes located on each tower and regular sensor nodes that are uniformly distributed into the target area as shown in Fig. 3. In this network, sensor nodes are responsible for detecting potential intrusion activities and inform the towers by forwarding data to the supernodes located at those towers. Since it is common to lose some sensor nodes because of energy depletion, harsh environmental conditions or hostile activities of intruders, it is desired for every sensor node to have more than a certain number of independent paths to the supernodes. In the figure, we can see a soldier crossing the border, and a sensor node close-by informs some towers via three disjoint paths.

This network model is first described in [22], and also used by the DPV algorithm. In this model, the network consists of M supernodes that are deployed at known locations and N sensor nodes that are randomly distributed in the 2D plane so that $M << N$. We assume the supernodes have transmission ranges long enough to communicate with the base station or any other supernode in the network. Therefore, we do not model and are not concerned with supernode-to-supernode communication. We are only interested in sensor-to-sensor and sensor-to-supernode communication.

We represent the initial network topology with an undirected weighted graph $G = (V, E)$, where V is the set of nodes and $E = \{v_i, v_j \mid dist(v_i, v_j) < R_{max}\}$ is the set of edges; $dist(v_i, v_j)$ defines the distance between nodes v_i and v_j.

3.2. Problem definition

We first give the formal definition of k-vertex supernode connectivity.

Definition 1 (k-vertex supernode connectivity [22])**.** An heterogeneous WSN is said to be k-vertex supernode-connected if removal of any $k-1$ sensor nodes does not disconnect any sensor node from all the supernode(s), that is, each sensor node is still connected to some supernode(s).

Initially we are given a k-vertex supernode-connected network with M supernodes and N sensor nodes, where the sensor node transmission range can be adjusted up to a predefined constant R_{max}. As we model node failures, the number of active sensor nodes decreases during the network lifetime. We use N_t to denote the set of active sensor nodes at time t, where time is represented by discrete time intervals. Our problem is to determine the transmission ranges of all active sensor nodes at any time, such that the resulting topology is still k-vertex supernode-connected, so that network lifetime can be improved. Now, we formally state the problem of maximizing fault-tolerant lifetime.

Definition 2 (Fault-tolerant lifetime maximization)**.** Given a k-vertex supernode-connected WSN $G = (V, E)$ with a set $M \subset V$ of supernode vertices and a set $N_t \subset V$ of active sensor node vertices, such that $M \cap N_t = \emptyset$, find a set of edges $F \subset E$ such that $G(V, E - F)$ is k-vertex supernode-connected and $\sum_{i=1}^{|N_t|} l_i$ is maximized, where l_i is the lifetime of the minimum lifetime path among the disjoint paths of $v \in N_t$.

3.3. Residual battery power level-aware disjoint path selection

The ADPV algorithm adapts the network topology dynamically during network operation by adjusting the sensor nodes' transmission ranges according to residual energy levels. For instance, if a node has low remaining energy, it should choose closer neighbors; otherwise, it

may choose farther ones. In this way, we attain a fair distribution of total residual energy among sensor nodes.

The DPV algorithm is not an energy-aware solution, and ignores sensor nodes' residual energy levels. This design may cause early battery depletion, since a node with low residual energy may be assigned to a high transmission power range. The ADPV algorithm, on the other hand, takes residual energy levels into consideration when selecting disjoint paths. Estimating the lifetime of each sensor node on a path lies at the core of our approach. The motivation behind this method is that a chain is only as strong as its weakest link, and thus a path survives only as long as all nodes survive in the path. Therefore, the shortest node lifetime on the path determines the lifetime of the path. The ADPV algorithm chooses a set of disjoint paths such that the minimum lifetime of those paths is maximized.

We formally define the lifetime of a path as follows: Let a path P consists of nodes $n_0, n_1, \ldots, n_l$, in which n_0 is the starting sensor node and n_l is a supernode. Let b_i denote the residual energy level of sensor node n_i and d_i denote the distance between n_i and n_{i+1} for each $0 \leq i < l$. Then, the lifetime of P is defined as:

$$\text{Lifetime}(P) = \min_{0 \leq i < l} \{b_i/(\beta + \alpha_1 + \alpha_2 \times d_i^n)\}, \quad (4)$$

where β, α_1, α_2 and n are the constant parameters of power consumption, defined in Section 2.3.

3.4. Initialization phase

This section describes our proposed approach for selecting alternative routes in the initialization phase of the algorithm, where those routes are to be used to restore connectivity during restoration phases. In ADPV, each sensor node keeps alternative routes, here referred to as restoration paths, that start with that node.

The primary goal is to consume the minimum possible resources while attaining high-quality restoration paths. The resources include memory, CPU, and network. Regarding memory, for instance, if all possible paths from sensor nodes to supernodes were held, the memory requirement would be intractable. In [36], Valiant discusses the average number of paths from a node to a given set of nodes. In terms of CPU, Bagci et al. [23] show that the complexity of selecting k disjoint paths from a pool of p alternatives is $O(p^k)$. Therefore, with a higher number of restoration paths of size r, it takes longer to compute a disjoint path set of size k during each restoration phase. As for the network, which is last but not least, we aim to communicate using minimum number of messages. Each restoration path incurs communication between its nodes in order to update its lifetime. As a result, we should maintain a very restricted set of restoration paths for the sake of network performance, but at the same time, the amount of those paths should be high enough to restore connectivity whenever needed.

To overcome these restrictions and efficiently construct restoration paths, ADPV employs a well-known method, called maximum set packing (MSP) [37]. This method is the optimization version of the set packing (SP) problem and asks for the maximum number of pairwise disjoint sets among a family of sets. More formally, for a given universe $\mathcal{U}$ and a family $\mathcal{S}$ of subsets of $\mathcal{U}$, MSP is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of sets such that all sets in $\mathcal{C}$ are pairwise disjoint, and $\mathcal{C}$ uses as many sets as possible, so that the size of the packing $\|\mathcal{C}\|$ is maximum. Maximum set packing is NP-hard [38] and cannot be approximated within any constant factor [39].

Algorithm 3.1 Maximum Set Packing (MSP).

Input: S
Output: M
1: $M \leftarrow \emptyset$;
2: **while** $S \neq \emptyset$ **do**
3: $m \leftarrow$ MinIntersectingPath(S);
4: $M \leftarrow M \bigcup m$;
5: **for all** Path $p \in S$ **do**
6: **if** $p \bigcap m \neq \emptyset$ **then**
7: $S \leftarrow S - p$;
8: **end if**
9: **end for**
10: **end while**

There is a well-known greedy heuristic, shown in Algorithm 3.1, to solve the MSP problem and it runs in polynomial time. We employ this heuristic to construct restoration paths. At the beginning, we have a pool of candidate paths of a relatively large size. The heuristic performs with many iterations, where each iteration selects the most diverse path from the pool. We use the term *diverse* as being disjoint with others, that is, the one that is disjoint to the largest number of paths among others in the pool. We add the selected path into the restoration path set and remove all the paths from the pool that intersect with the selected path. The iterations continue until the pool becomes empty or the number of restoration paths reaches a predefined threshold. Since the initial sensor node and the destination supernode do not violate disjoincy, ADPV represents each path by the set of its intermediate sensor nodes.

Algorithm 3.2 Path Information Collection in ADPV.

Input: I, L, k
Output: D, R
1: $T \leftarrow \emptyset$;
2: $R \leftarrow \emptyset$;
3: **for all** received PathInfo message I **do**
4: $D \leftarrow$ MinDisSet(T, k);
5: $c \leftarrow$ Cost(D);
6: $U \leftarrow I.T \cup T$;
7: $R \leftarrow$ MaxSetPacking($R \cup U$); (Algorithm 3.1)
8: $U' \leftarrow$ MaxSetPacking(U);
9: Sort(U');
10: $T' \leftarrow \{p_i \in U' \mid i <= L\}$;
11: $D' \leftarrow$ MinDisSet(T', k);
12: $c' \leftarrow$ Cost(D');
13: **if** $c' < c$ **then**
14: $T \leftarrow T'$;
15: Transmit PathInfo(T);
16: **end if**
17: **end for**

Algorithm 3.2 shows path-information-collection and restoration-path-selection procedures. The variables

Table 1
ADPV notations.

I	Received pathInfo message
L	Maximum number of paths to be stored
k	Disjoint connectivity degree
R	Set of restoration paths
T and T'	Set of local paths
D and D'	Set of disjoint paths
c and c'	Cost of disjoint paths, which equals the minimum lifetime of the disjoint paths
U	Union of two path sets
S	Set of paths
M	Set of paths in MSP
m, p, r	Variables referencing paths
sr	Supernode ratio
n	Total number of sensors
Δ	Maximum degree of a node
r	Amount of a node's restoration paths
l	Average path length in the restoration set
n_0, n_1, .., n_i	Number of remaining sensor nodes after each restoration phase

used in the pseudo codes are defined in Table 1. In Algorithm 3.2, each sensor node maintains a local path set along with disjoint and restoration path sets. As an input, the algorithm takes a 'PathInfo' message that contains the local path set of the sender node and generates two outputs, which are the disjoint path and restoration path sets of size k and a relatively large size, respectively. Local paths are logical paths that are used for informing neighbor nodes about the paths they can use over the sender node. Therefore, local paths have a very critical role in determining disjoint and restoration path sets and need to be selected very carefully. When a sensor node receives a 'PathInfo' message containing a local path set, it first calculates the union of the sender's and receiver's local path sets. It then executes the MSP procedure on this union to eliminate paths that have too many sensor nodes in common. The procedure then determines a candidate local path set T' as the first L minimum-cost path of the remaining set. While doing so, the procedure also updates the restoration paths by executing the MSP procedure on the union of local sets and the current restoration path set.

Using the candidate local path set, the set of disjoint paths with minimum cost is calculated using Algorithm 3.3. In this algorithm, all disjoint subsets with k elements are traversed and the one with the minimum cost is selected. If the minimum-cost disjoint path set has a smaller cost than the current disjoint path set, both the disjoint and the local paths are updated and a 'PathInfo' message containing the new local path set is transmitted to the set of neighbors. This process continues until there are no more updates in the disjoint path sets.

After determining the disjoint paths, each sensor node determines its required neighbors, which include the neighbors that disjoint paths use the edges between. After determining the required neighbors, each node adjusts its transmission power to reach its farthest neighbor according to the resulting topology.

Algorithm 3.3 Finding Disjoint Paths to Supernodes (MinDisSet).

Input: T and k
Output: D
1: $D \leftarrow \emptyset$;
2: **if** $|T| > k$ **then**
3: $Q \leftarrow \{ q \subset T \mid |q| = k \}$;
4: $c \leftarrow \infty$;
5: $q_{min} \leftarrow \emptyset$;
6: **for all** $q \in Q$ **do**
7: **if** q consists of disjoint paths **then**
8: **if** Cost(q) $< c$ **then**
9: $c \leftarrow$ Cost(q);
10: $q_{min} \leftarrow q$;
11: **end if**
12: **end if**
13: **end for**
14: $D \leftarrow q_{min}$;
15: **end if**

3.5. Connectivity restoration phase

We start the connectivity restoration procedure only when k-vertex supernode connectivity is broken due to node failure. Thus, the first step after a node failure is to check whether the network is still k-vertex supernode-connected or not. As this is a costly operation [40], ADPV employs a simple distributed greedy heuristic with no false positives. That is, if ADPV postulates the network is k-vertex supernode-connected, then the network is definitely connected. However, the network can still be connected even if ADPV claims it is not. Therefore, ADPV ensures strong k-vertex supernode connectivity.

When a node failure occurs, ADPV ensures all the node's neighbors initiate a failure message to inform others about the failure. Upon receiving a failure message, a sensor node removes all paths including the failed node from its restoration set. Since frequent transmission power adjustment is difficult to realize in practice, we employ periodical transmission power control, and during each period we check whether any failed nodes exist on any of the disjoint paths. If a failed node disconnects a disjoint path, the restoration process takes place. Note that this event does not necessarily imply k-vertex supernode disconnectivity, yet because ADPV takes early action it never allows the connectivity to break. After deciding k-vertex supernode connectivity must be restored, ADPV applies a two-step process: updating the lifetimes of the restoration paths and computing minimum-cost disjoint paths from the restoration set.

In the first step, path lifetimes in the restoration set are updated via messages transmitted along the path from the source node to the destination supernode. Each node redirects a received message to the next hop in the path and returns a message that contains updated lifetime information of the sensor nodes back along that path. In the second step, minimum-cost disjoint paths are computed using the previously discussed disjoint-path-selection algorithm, Algorithm 3.3. An overview of the connectivity-checking and connectivity-restoration procedures are given in Algorithm 3.4.

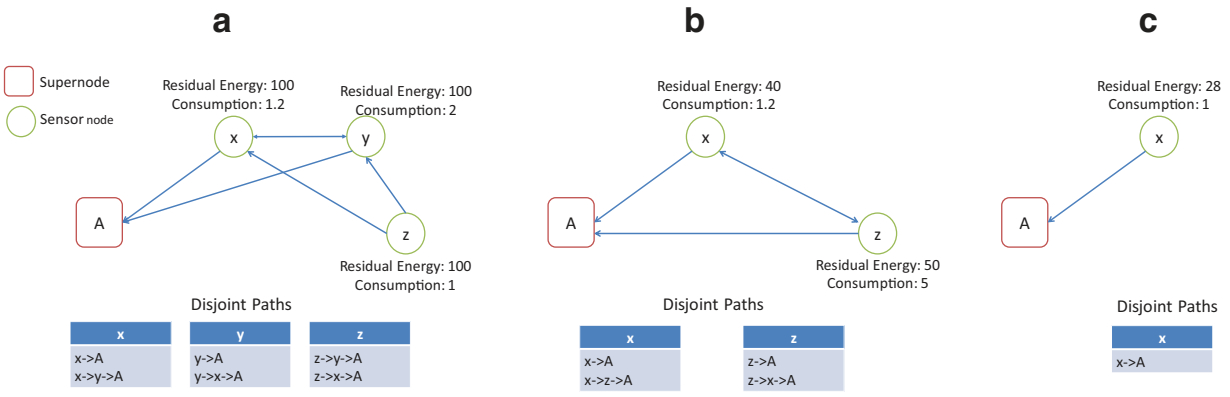

Fig. 4. Sample connectivity restoration for $k = 2$.

Algorithm 3.4 Connectivity Restoration in ADPV.

Input: k, R, D
Output: D

```
1:  FailedNodes ← ∅;
2:  for all received node failure message δ do
3:    for all Path r ∈ R do
4:      if r contains δ.FailedNode then
5:        R ← R − r;
6:      end if
7:    end for
8:    FailedNodes ← FailedNodes ∪ δ.FailedNode
9:    if certain time elapsed since last period then
10:     for all Path p ∈ D do
11:       if (p ∩ FailedNodes) ≠ ∅ then
12:         UpdateCosts(R);
13:         D ← MinDisSet(R, k);
14:         break;
15:       end if
16:     end for
17:     FailedNodes ← ∅;
18:   end if
19: end for
```

For instance, continuing from the example given in Section 2.2, for $k = 2$, ADPV optimizes the topology shown in Fig. 1, as in Fig. 4(a). In this topology, all initial energy levels are equal. Assuming the data generation rate is uniform for all nodes, the power consumption of nodes x, y and z are 1.2, 2 and 1, respectively. With this power consumption, node y dies first (100/2=50 s later), both node x and node z lose one of their disjoint paths and the network becomes one-vertex, but not two-vertex, supernode-connected. The ADPV algorithm restores connectivity, as in Fig. 4(b), by adjusting the transmission range of z, which introduces a link from node z to supernode A. Because of its increasing power consumption, node z happens to be the second dying node (50/5=10 s later) and thus node x loses two-vertex supernode connectivity once more. However, because it has no alternative routes, it adjusts its transmission power again and works as a connected node, as in Fig. 4(c), for the rest of its life (28/1=28 more seconds). For this network, the two-vertex supernode-connected lifetime is broken when node z dies. Therefore, the two-vertex supernode-connected lifetime of this network equals 60 s and the one-vertex supernode-connected lifetime equals 88 s.

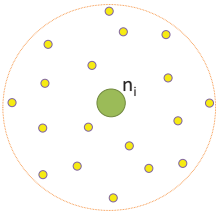

Fig. 5. One-hop neighbors of node i.

Lemma 1. *The connectivity restoration process of ADPV ensures k-vertex supernode connectivity.*

Proof. By definition, the network gets k-vertex supernode-connected if each sensor in the network is connected to at least one supernode with k-vertex disjoint paths. This translates into the disjoint path set of each sensor node of being size k, and if there exist more than k paths in the restoration set, ADPV chooses a disjoint set and ensures k-vertex supernode connectivity. □

Lemma 2. *In the restoration path set, there are at most* Δ *paths, where* Δ *is the maximum degree of a sensor node.*

Proof. We are going to prove this by contradiction. As discussed in Section 3.5, each sensor node keeps a maximum set pack of some size in their restoration sets, so that each path in the set is pairwise disjoint with the others. Let Δ denote the maximum degree of a node and assume there exists a node, say node i, that has more than Δ paths in its restoration set. Since there are more than Δ paths that are using at most Δ neighbors, according to the pigeonhole principle, there exist two restoration paths that use the same neighbor. Let Fig. 5 represent node i and its one-hop neighbors. If the neighbor that two paths have in common is a supernode, then node i will have exactly the same two paths in its restoration set, which is not possible, because the MSP procedure calculates the union of the selected paths to guarantee diversity. If that neighbor

is a sensor node, those paths will not be disjoint, which violates the MSP definition. Therefore, no neighbor, neither sensor node nor supernode, can have two paths in common, and the number of elements in the restoration set cannot exceed Δ. □

3.6. Running time analysis

We compute the lifetime of a restoration path via messages transmitted along the path from the source node to the destination supernode. Each node redirects a received message to the next hop in the path and returns a message that contains the sensors' updated lifetime information back along the same path. Therefore, for each restoration path, the number of messages equals two times the length of the path. We assume that path length is bounded by a constant, say l, following previous studies [23]. Notice that the number of restoration paths is less than or equal to Δ, where Δ is the maximum degree of a node. Then, there are at most $l \times \Delta$ messages in total, and thus, the message complexity is $O(\Delta)$ at each connectivity-restoration phase. In the worst case, for each sensor node, connectivity restoration is carried out for $O(\Delta)$ times, as the restoration path set embodies at most $l \times \Delta$ nodes. Therefore, at each sensor node, total message complexity becomes $O(\Delta^2)$ for connectivity restoration. For path information collection, ADPV has the message complexity of $O(n\Delta)$, which also equals that of DPV [23]. Therefore, the total message complexity becomes $O(\Delta^2) + O(n\Delta) = O(n\Delta)$.

The ADPV algorithm consumes computational power in the initialization phase for disjoint and restoration path construction and during the connectivity restoration phase for determining new disjoint paths from the restoration set. During the initialization phase, when sensor nodes receive a 'PathInfo' message, they calculate the union of the local path information and the received paths in the incoming message. The running time complexity of this step depends on the number of paths (p) in the local path information table. In ADPV, since the maximum number of paths that can be stored in a sensor node's path information table is set to a constant value, both calculating the union of the two path information tables and sorting the paths according to their costs take constant time.

In the initialization phase, there are two more procedures that consumes processing power: maximum set packing and disjoint-path-selection algorithms. The greedy heuristic for MSP, shown in Algorithm 3.1, is used twice: once for constructing the restoration path and again for selecting the local path information table. As discussed in the second lemma, the number of restoration paths is limited by the maximum degree of node (Δ), and the number of paths in the local path information table is a constant (l). Therefore, the MSP algorithm consists of numerous iterations, each consisting of two steps: i) selecting the minimum intersecting path and ii) removing the paths that intersect with it. In the latter step, the algorithm traverses all path pairs and determines the minimum intersecting one. The activity of removing the intersecting paths also traverses the set once more. Considering set size is represented by s, the running time complexity of the MSP algorithm equals $O(s^2 + s)$. Therefore, the MSP running time complexity in each step is $O(\Delta^2 + \Delta + l^2 + l)$, which can be reduced to $O(\Delta^2)$.

To calculate the minimum disjoint set, Algorithm 3.3 enumerates all subsets of size k and finds the set with the minimum cost. Enumerating all these subsets takes $O(p^k)$, where p represents the number of paths in the given set. Since the input of the minimum disjoint set procedure is the local path information table, which has a constant number of elements, the running time complexity of the minimum disjoint set calculation is also a constant.

Considering that the message transmission complexity of ADPV is $O(n\Delta)$ and the dominating step (MSP) is executed once for every incoming message, the running time complexity of the total initialization phase is $O(n\Delta^3)$.

For the restoration phases, as discussed above, the maximum number of restoration phases a node can execute is $O(l\Delta)$, and in each phase there are two operations: updating path costs, which only uses message transmissions, and calculating the minimum disjoint set from the restoration set. Since the maximum number of elements in the restoration set is Δ, the running time complexity for determining the minimum-cost disjoint paths from the restoration set will be $O(\Delta^k)$, and the total running time complexity of the restoration phases will be $O(\Delta^{k+1})$.

Since $n > > \Delta$, and the commonly accepted values of k are 2 and 3 [22], the ADPV running time complexity equals $O(n\Delta^3)$.

3.7. Expected number of restorations in ADPV

In this section we discuss theoretical expectations resulting from the ADPV algorithm and analyze how many times ADPV can restore k-vertex supernode connectivity for a given node. Since ADPV can restore such connectivity when there are at least k paths in the restoration set, we will determine the expected number of node failures before a node cannot restore its connectivity. Let n denote the number of sensor nodes in the network and assume the sensor node batteries deplete uniformly in any order with the same probability ρ. The parameters used in this section are given in Table 1.

Considering that the number of sensor nodes in the restoration set equals $r \times l$, the expected number of sensor nodes that die before one of these $r \times l$ sensor nodes dies equals:

$$\frac{n}{r \times l}. \tag{5}$$

For example, if there are 100 nodes in the entire network and 20 take part in the restoration set, then the expected number of node failures before one of the nodes in the restoration set fails equals five. When a node on a path dies, then that path will no longer be valid and therefore will be removed from the restoration sets available. As a result, with a node failure, the number of restoration paths will diminish by one. Therefore, when the first node on a restoration set dies, $r-1$ paths, which consist of $(r-1) \times l$ sensor nodes, will remain. At the same time, the number of remaining sensor nodes in the entire

network will equal:

$$n - \frac{n}{r \times l}. \tag{6}$$

Continuing from the previous example, $100 - 5 = 95$ sensor nodes will remain in the entire network after the first node in the restoration set dies. The remaining sensor nodes after the ith restoration path removal can be generalized as follows:

$$n_{i+1} = n_i - \frac{n_i}{(r-i) \times l}, \tag{7}$$

which also equals:

$$n_{i+1} = n_i \times \left(1 - \frac{1}{(r-i) \times l}\right) \tag{8}$$

and which can also be written as a product of:

$$n_{i+1} = n \times \prod_{j=0}^{i} \left(1 - \frac{1}{(r-j) \times l}\right). \tag{9}$$

Since ADPV can restore k-vertex supernode connectivity when there are at least k paths in the restoration set, the number of sensor nodes when k-vertex supernode connectivity of the given node cannot be restored equals n_{r-k+1} and can be calculated as:

$$n_{r-k+1} = n \times \prod_{j=0}^{r-k} \left(1 - \frac{1}{(r-j) \times l}\right). \tag{10}$$

Then by changing the parameter to $t = r - j$,

$$n_{r-k+1} = n \times \prod_{t=k}^{r} \left(1 - \frac{1}{t \times l}\right). \tag{11}$$

According to the above formula, the number of successful restorations will be proportional to the sensor node count. Also, with the increasing average path length, the number of remaining sensors increases, which in turn decreases the possibility of successful restorations. Therefore, choosing paths with smaller path lengths may be preferable.

4. Experiments and results

In this section we report our measurements regarding lifetime and other metrics for the DPV and ADPV algorithms and try to evaluate ADPV's success. For this evaluation, we implemented ADPV using an extended version of a custom simulator, which has also been used for evaluating the DPV algorithm. We added a time dimension and a battery model into the existing framework and thus provided an environment that could evaluate network lifetime.

4.1. Experimental setup

In our experiments, we assumed that sensor nodes and supernodes are uniformly and randomly deployed in an area of 600 m x 600 m and that the initial maximum transmission range R_{max} of the sensor nodes is set at 100 m. We repeated our experiments for $\{100, 150, \ldots, 500\}$ sensor node, for $k = 2, 3$ (as these are commonly accepted k values), and for a supernode ratio (sr) of 5% and 10% over the region. Finally, we assumed a packet loss rate of 10% for each message transmission. As a result, we had $9 \times 2 \times 2$ experimental instances, and on each we executed both algorithms 20 times and reported the averages. Our simulation parameters are summarized in Table 2.

Table 2
Simulation parameters.

Deployment area	600 m x 600 m
Initial transmission range of sensor nodes: R_{max}	100 m
Number of sensor nodes: N	[100 . . . 500]
Number of supernodes: M	5% and 10% of N
Degree of disjoint connectivity: k	2 and 3
Packet loss rate	10%

4.2. Results

In Fig. 6, we compare the node failure tolerance of DPV and ADPV. For each algorithm, we measure performance in terms of the fraction of dead sensor nodes when the network gets (i) supernode disconnected and (ii) k-vertex supernode disconnected. If there exists a path (single or multi-hop) between a sensor node and any one of the supernodes, then the sensor node is said to be connected. If every (alive) sensor node in the network has k disjoint paths to the set of supernodes, then the network is considered as k-vertex supernode-connected. With these measurements we determine the maximum number of node failures that can occur before supernode connectivity is broken. Here, we observe the most striking result, and at the same time, evidence of this study's motivation regarding the instability of static algorithms and effectiveness of ADPV for keeping the network supernode-connected. As seen in the figure, even before the failure of 5% of the sensor nodes, the network's supernode connectivity is broken when we employ the DPV algorithm. This result limits using DPV as a fault-tolerant alternative. On the other hand, ADPV successfully keeps the network supernode-connected up to failure of about 95% of the sensor nodes.

In the figure, we observe that when the network becomes denser, ADPV keeps it supernode-connectivity for longer. This result can be attributed to ADPV becoming more effective at finding alternative routes due to the increasing number of sensor nodes. For instance, in one extreme, when we examine the results of a 500-node network for $k = 2$ and $sr = 10\%$, as shown in Fig. 6(a), we see that the network is still supernode-connected up to a failure of 95% of the sensor nodes. In the other extreme, where the number of initially deployed sensor nodes equals 100, ADPV sustains supernode connectivity up to the failure of 20% of the sensor nodes. Looking into each of the sub-figures, when the initial number of sensor nodes is between 250 and 300, we notice that ADPV succeeds in keeping supernode connectivity even after the active sensor nodes are halved. Considering all experimental instances, on average, ADPV maintains supernode connectivity up to a failure of 52% of sensor nodes for $k = 2$, and 55% of sensor nodes for $k = 3$. Since the optimized network topologies for $k = 3$ contain more connections, it is

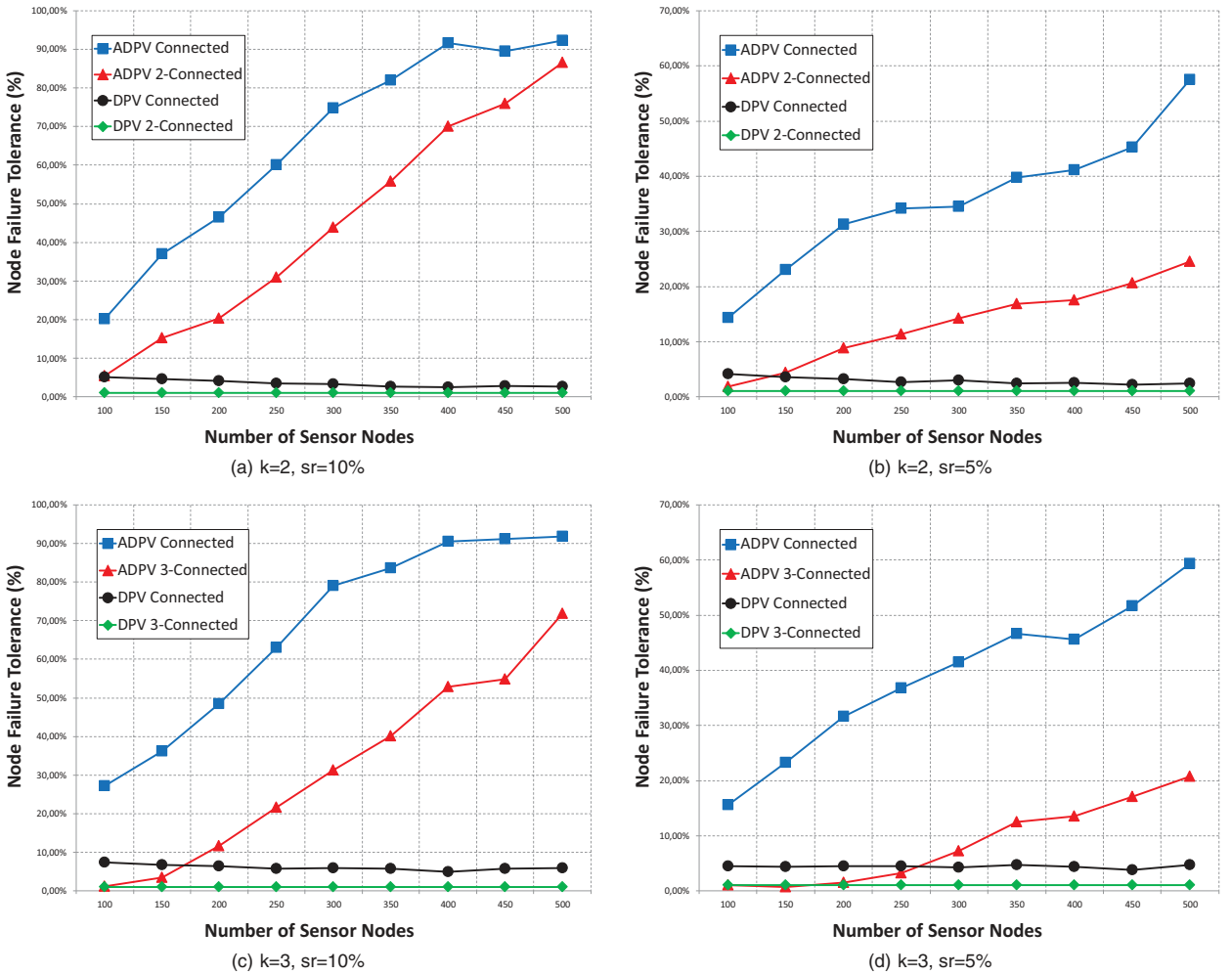

Fig. 6. Percentage of failed sensor nodes when the network becomes (i) supernode disconnected and (ii) k-vertex supernode disconnected.

expected that those networks will have a higher tolerance for node failures. On the other hand, more connections will consume more battery power, which will affect network lifetime. We therefore also examine lifetime measurements of the networks for the same set of scenarios.

Further, as we can observe in Fig. 6, network topologies generated by the DPV algorithm become k-vertex supernode disconnected after the failure of at most 1% of the sensor nodes. Even though the initial optimized topologies generated by DPV are k-vertex supernode-connected, after the failure of a few sensor nodes, the remaining topologies become at most $(k-1)$-vertex supernode-connected. The ADPV algorithm, on the other hand, maintains two-vertex supernode connectivity up to a failure of 32% of sensor nodes, and three-vertex supernode connectivity up to a failure of 21% of sensor nodes, on average among all experimental instances.

In Fig. 7, we compare the lifetime results of the same set of experimental instances with those of Fig. 6. A prominent aspect of ADPV is considering the sensor nodes' remaining energy levels; as a result, energy depletion occurs less frequently. This factor, when coupled with the adaptive nature of the algorithm, results in longer network lifetimes.

As we observe in Fig. 7, in terms of one-vertex supernode-connected lifetimes, on average, ADPV results in a two-fold increase with respect to DPV. For the same experimental instances, ADPV provides respectively 65% and 46% longer two-vertex and three-vertex supernode-connected lifetimes than DPV. As we observe in the figures, network density has almost no effect on the lifetimes of the topologies generated by DPV. On the other hand, ADPV successfully prolongs the network lifetime almost proportionally to the network density for all $k = 1, 2, 3$.

We observe in Fig. 7(c) and (d) that if the number of sensor nodes drops below a certain threshold (in our case, 150 sensor nodes in a 600 m x 600 m area when $sr = 10\%$, and 200 sensor nodes when $sr = 5\%$), it is hard to restore three-vertex supernode connectivity. This finding suggests that a minimum number of sensor nodes for every k and sr value is necessary to restore k-vertex supernode connectivity. Figs. 6 and 7, respectively compare node failure tolerance and network lifetime for different values of $sr = 5\%, 10\%$ and $k = 2, 3$. According to the results, with the

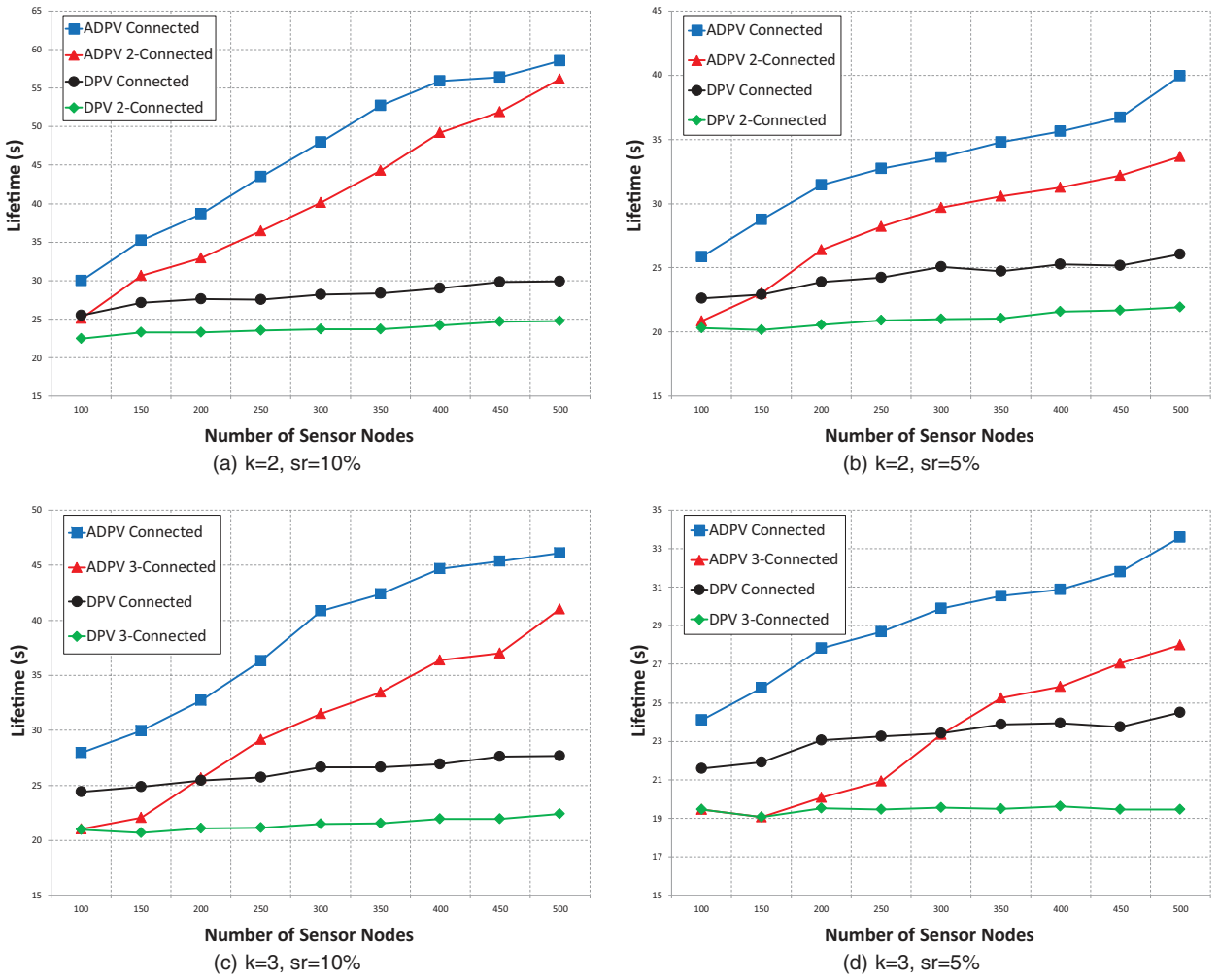

Fig. 7. Lifetime comparison of the DPV and ADPV algorithms.

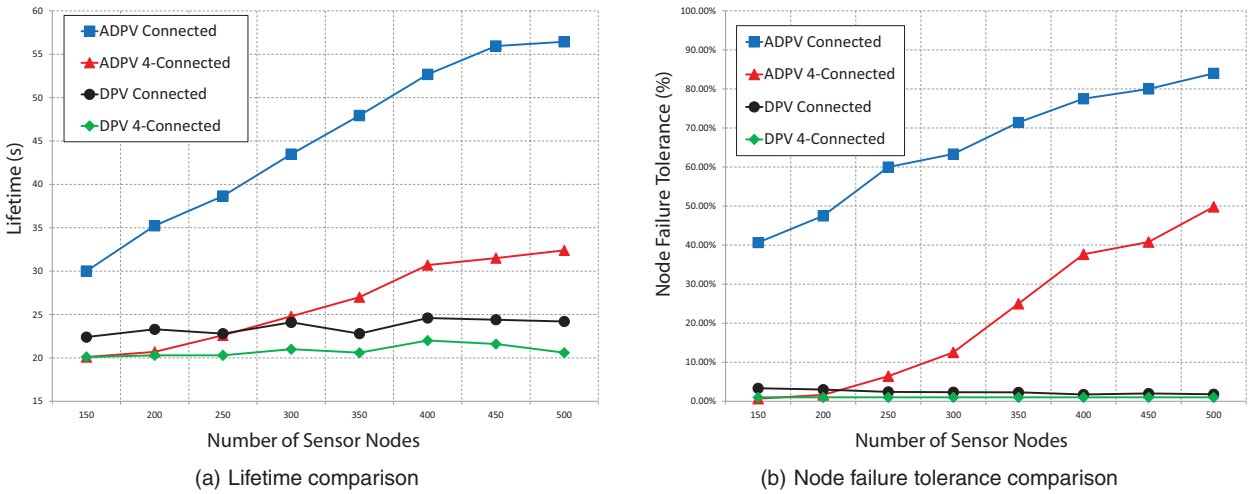

Fig. 8. Lifetime and node failure tolerance of DPV and ADPV algorithms for $k = 4$.

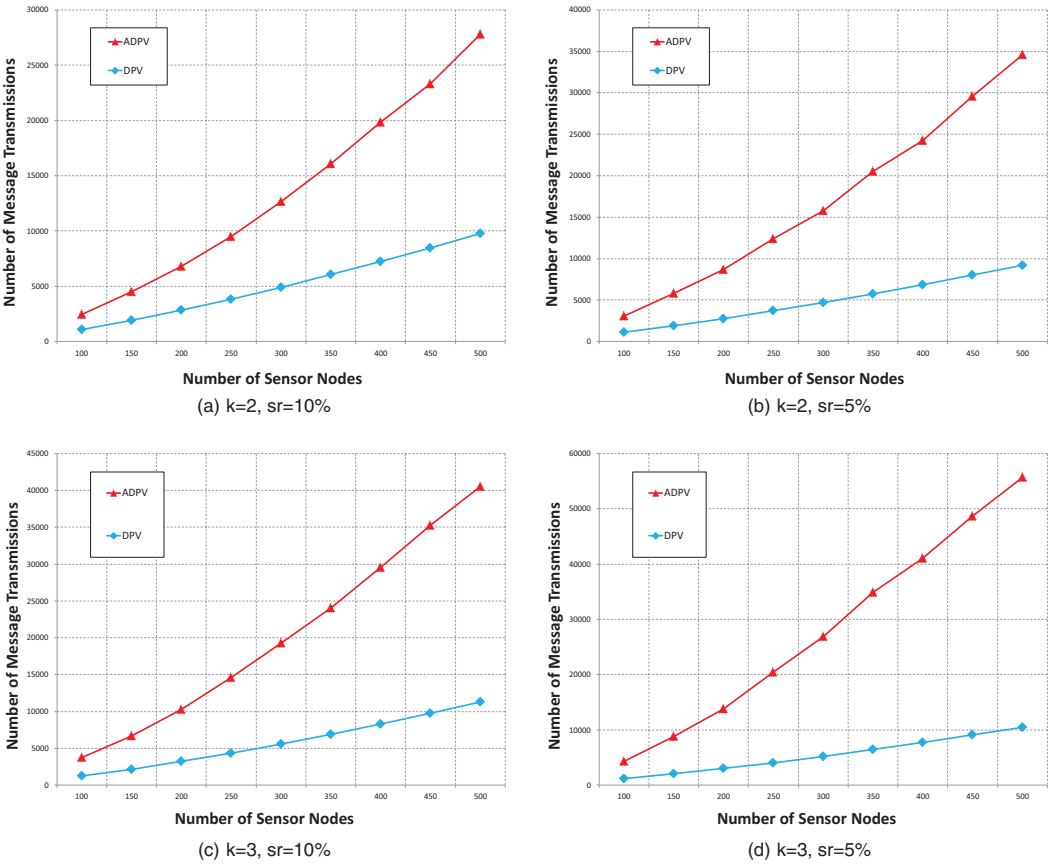

Fig. 9. Number of message transmissions in DPV and ADPV algorithms.

increasing number of supernodes, lifetime also increases, however, the relation between the increase in the number of supernodes and the increase in the lifetime is sublinear, and therefore, we expect that the increase in the lifetime becomes insubstantial as the number of supernodes exceeds a certain threshold. We also notice that, with the increasing k value, more disjoint paths are required and this makes providing alternative routes harder. In Fig. 8(a) and (b), we compare DPV and ADPV algorithms for $k = 4$ in terms of network lifetime and node failure tolerance, respectively. As seen in Fig. 8(a), ADPV successfully prolongs both one-vertex and four-vertex supernode-connected lifetimes of the network. Also, in Fig. 8(b), we see that ADPV can preserve four-vertex supernode-connectivity up to the failure of 50% of the sensor nodes on dense networks, which in turn, achieves almost a two-fold increase in the four-vertex supernode-connected lifetime.

Another important metric we measure during our analysis is the number of message transmissions. Message transmission is an important metric because we must not only consider power consumption in the resulting topologies but also consider the power required to generate those topologies, which can be viewed as a fixed cost of obtaining the final topologies. If this cost is high, then the power efficiency of the resulting topology might become meaningless. In Fig. 9, we compare the number of DPV and ADPV message transmissions. To simulate the worst-case scenario of ADPV, we set the waiting period in the restoration phase to zero, which means that every node failure that affects disjoint paths will trigger a restoration phase for that node. According to the results, for $k = 2$, ADPV makes at least 2.25 times and at most three times the message transmissions than DPV does, and for $k = 3$, ADPV makes at least three times and at most 3.5 times the message transmissions of DPV. As seen in the sub-figures,

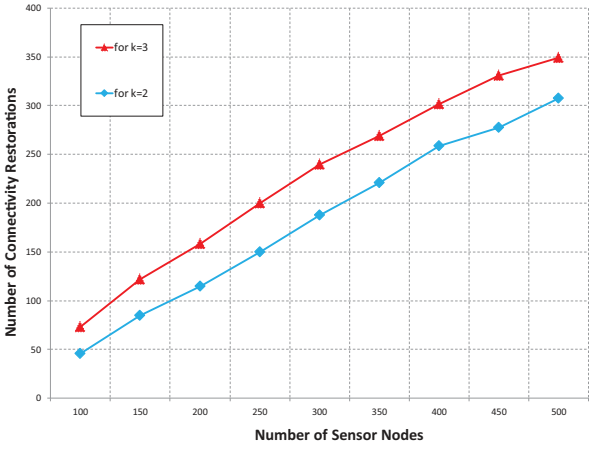

Fig. 10. Number of connectivity restorations.

the number of message transmissions of both algorithms increases almost linearly with the number of sensor nodes, and the ratio of these message counts does not significantly change. We should note that when, for instance, assuming $k = 3$, as seen in Fig. 10, ADPV restores supernode connectivity almost 350 times more than DPV does, with only 3.5 times the message transmissions. Considering the extra messages in ADPV are used for updating the residual energy levels of the remaining active sensor nodes in the disjoint paths, these messages are very crucial for better load balancing and for making maximum use of the available sensor nodes.

5. Conclusion

In this study, we present ADPV, an adaptive, energy-aware and distributed topology-control algorithm. The motivation of this algorithm is to prolong the supernode-connected lifetime of given heterogeneous WSNs. The ADPV algorithm consists of two phases: initialization and restoration. During initialization, ADPV computes alternative routes in the network. To determine routes efficiently ADPV employs a novel method based on set packing. Whenever k-vertex supernode connectivity is broken, the restoration phase is activated. To restore connectivity, ADPV utilizes those alternative routes and adjusts the sensor nodes' transmission ranges accordingly. The ADPV algorithm is a distributed algorithm in both the initialization and restoration phases. A broad set of conducted simulations agrees well with the theoretical anticipation that ADPV can significantly prolong supernode-connected lifetimes of heterogeneous WSNs. Our adaptive algorithm increases the durability of network connectivity against node failures, from 5% up to 95%. As for k-vertex connectivity, we are able to keep the network two- and three-vertex supernode-connected up to the failure of 90% and 75% of sensor nodes, respectively.

In this study, we assume that supernodes are stationary and arbitrarily deployed with no concern for sensor node positions. It would be an interesting future work to relax these assumptions. For instance, supernodes could be placed with respect to positions of already-deployed sensor nodes. In this way, one would have opportunity to find more center-like positions within sensor nodes, and in turn improve system efficiency. Similarly, instead of being stationary, supernodes can be mobile and thus could be repositioned to further increase network lifetime.

Acknowledgments

This work is supported in part by a research Grant from TUBITAK with Grant no. 114R082.

References

[1] D. Ghataoura, J. Mitchell, G. Matich, Networking and application interface technology for wireless sensor network surveillance and monitoring, Commun. Mag. IEEE 49 (10) (2011) 90–97, doi:10.1109/MCOM.2011.6035821.

[2] M. Durisic, Z. Tafa, G. Dimic, V. Milutinovic, A survey of military applications of wireless sensor networks, in: Embedded Computing (MECO), 2012 Mediterranean Conference on, 2012, pp. 196–199.

[3] Z. Sun, P. Wanga, M. Vuran, M. Al-Rodhaan, A. Al-Dhelaan, I. Akyildiz, Bordersense: border patrol through advanced wireless sensor networks, Ad Hoc Netw. 9 (3) (2011) 468–477.

[4] S. Zhuiykov, Solid-state sensors monitoring parameters of water quality for the next generation of wireless sensor networks, Sens. Actuators B: Chem. 161 (1) (2012) 1–20.

[5] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson, Wireless sensor networks for habitat monitoring, in: In Proceedings of ACM Wireless Sensor Networks and Applications(WSNA), 2002, pp. 88–97.

[6] M. Hefeeda, M. Bagheri, Wireless sensor networks for early detection of forest fires, in: Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on, 2007, pp. 1–6, doi:10.1109/MOBHOC.2007.4428702.

[7] B. Placzek, Selective data collection in vehicular networks for traffic control applications, Transp. Res. Part C: Emerg. Technol. 23 (0) (2012) 14–28.Data Management in Vehicular Networks

[8] C. Wenjie, C. Lifeng, C. Zhanglong, T. Shiliang, A realtime dynamic traffic control system based on wireless sensor network, in: Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on, 2005, pp. 258–264, doi:10.1109/ICPPW.2005.16.

[9] S. Coleri, S. Cheung, P. Varaiya, Sensor networks for monitoring traffic, in: In Allerton Conference on Communication, Control and Computing, 2004.

[10] W. Wang, V. Srinivasan, K. Chua, Using mobile relays to prolong the lifetime of wireless sensor networks, in: Proceedings of the 11th Annual International Conference on Mobile Computing and Networking, in: MobiCom '05, ACM, New York, NY, USA, 2005, pp. 270–283, doi:10.1145/1080829.1080858.

[11] S. Halder, A. Ghosal, S. Bit, A pre-determined node deployment strategy to prolong network lifetime in wireless sensor network, Comput. Commun. 34 (11) (2011) 1294–1306, doi:10.1016/j.comcom.2011.01.004.

[12] G. Anastasi, M. Conti, M. Francesco, A. Passarella, Energy conservation in wireless sensor networks: a survey, Ad Hoc Netw. 7 (3) (2009) 537–568, doi:10.1016/j.adhoc.2008.06.003.

[13] Y. Chen, Q. Zhao, On the lifetime of wireless sensor networks, Commun. Lett. IEEE 9 (11) (2005) 976–978, doi:10.1109/LCOMM.2005.11010.

[14] J. Li, P. Mohapatra, An analytical model for the energy hole problem in many-to-one sensor networks, in: Vehicular Technology Conference, 2005. VTC-2005-Fall. 2005 IEEE 62nd, vol. 4, 2005, pp. 2721–2725, doi:10.1109/VETECF.2005.1559043.

[15] J. Zhang, Y. Liu, D. Sun, B. Li, Prolonging the lifetime of wireless sensor networks by utilizing feedback control, Wirel. Netw. 20 (7) (2014) 2095–2107, doi:10.1007/s11276-014-0726-x.

[16] W. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocols for wireless microsensor networks, in: International Conference on System Sciences, 2000.

[17] P. Kar, A. Roy, S. Misra, Connectivity reestablishment in self-organizing sensor networks with dumb nodes, ACM Trans. Auton. Adapt. Syst. 10 (4) (2016) 28:1–28:30, doi:10.1145/2816820.

[18] V. Rao, P. Priyesh, S. Kar, Adaptive transmission power protocol for heterogeneous wireless sensor networks, in: Communications (NCC), 2015 Twenty First National Conference on, 2015, pp. 1–5.

[19] A. Woo, T. Tong, D. Culler, Taming the underlying challenges of reliable multihop routing in sensor networks, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, in: SenSys '03, ACM, New York, NY, USA, 2003, pp. 14–27, doi:10.1145/958491.958494.

[20] L. Paradis, Q. Han, A survey of fault management in wireless sensor networks, J. Netw. Syst. Manag. 15 (2) (2007) 171–190.

[21] H. Liu, A. Nayak, I. Stojmenovic, Fault-Tolerant Algorithms/Protocols in Wireless Sensor Networks, Guide to Wireless Sensor Networks, Springer London, London, 2009, pp. 261–291, doi:10.1007/978-1-84882-218-4_10.

[22] M. Cardei, S. Yang, J. Wu, Algorithms for fault-tolerant topology in heterogeneous wireless sensor networks, Parallel Distrib. Syst. IEEE Trans. 19 (4) (2008) 545–558, doi:10.1109/TPDS.2007.70768.

[23] H. Bagci, I. Korpeoglu, A. Yazici, A distributed fault-tolerant topology control algorithm for heterogeneous wireless sensor networks, Parallel Distrib. Syst. IEEE Trans. PP (99) (2014), doi:10.1109/TPDS.2014.2316142.1–1

[24] A. Tanenbaum, M. Steen, Distributed systems, Citeseer, 2002.

[25] K. Akkaya, F. Senel, A. Thimmapuram, S. Uludag, Distributed recovery from network partitioning in movable sensor/actor networks via controlled mobility, Comput. IEEE Trans. 59 (2) (2010) 258–271, doi:10.1109/TC.2009.120.

[26] X. Han, X. Cao, E. Lloyd, C.-C. Shen, Fault-tolerant relay node placement in heterogeneous wireless sensor networks, IEEE Trans. Mob. Comput. 9 (5) (2010) 643–656.
[27] E. Lloyd, G. Xue, Relay node placement in wireless sensor networks, IEEE Trans. Comput. 56 (1) (2007) 134–138.
[28] A. Kashyap, S. Khuller, M. Shayman, Relay Placement for Higher Order Connectivity in Wireless Sensor Networks, in: INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, 2006, pp. 1–12.
[29] J. Tang, B. Hao, A. Sen, Relay node placement in large scale wireless sensor networks, Comput. Commun. 29 (4) (2006) 490–501.
[30] W. Zhang, G. Xue, S. Misra, Fault-Tolerant Relay Node Placement in Wireless Sensor Networks: Problems and Algorithms, in: INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE, 2007, pp. 1649–1657.
[31] X. Cheng, D. Du, L. Wang, B. Xu, Relay sensor placement in wireless sensor networks, Wirel. Netw. 14 (3) (2008) 347–355.
[32] X. Wang, M. Sheng, M. Liu, D. Zhai, Y. Zhang, Resp: A k-connected residual energy-aware topology control algorithm for ad hoc networks, in: Wireless Communications and Networking Conference (WCNC), 2013 IEEE, 2013, pp. 1009–1014, doi:10.1109/WCNC.2013.6554702.
[33] Z. Yin, F. Li, M. Shen, Y. Wang, Fault-tolerant topology for energy-harvesting heterogeneous wireless sensor networks, in: Communications (ICC), 2015 IEEE International Conference on, 2015, pp. 6761–6766, doi:10.1109/ICC.2015.7249403.
[34] W. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, in: System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on, 2000, doi:10.1109/HICSS.2000.926982.
[35] W. Heinzelman, A. Chandrakasan, A. Smith, An application-specific protocol architecture for wireless microsensor networks, IEEE Trans. Wirel. Commun. 1 (2002) 660–670.
[36] L. Valiant, The complexity of enumeration and reliability problems, SIAM J. Comput. 8 (3) (1979) 410–421, doi:10.1137/0208032.
[37] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.
[38] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher, J. Bohlinger (Eds.), Complexity of Computer Computations, The IBM Research Symposia Series, Springer US, 1972, pp. 85–103.
[39] E. Hazan, S. Safra, O. Schwartz, On the complexity of approximating k-set packing, Comput. Complex. 15 (1) (2006) 20–39, doi:10.1007/s00037-006-0205-6.
[40] S. Even, An algorithm for determining whether the connectivity of a graph is at least k, SIAM J. Comput. 4 (3) (1975) 393–396, doi:10.1137/0204034.

Fatih Deniz received his BS and MS degrees from Bilkent University (2007) and Middle East Technical University (2010), respectively, both in Computer Science. He is a PhD candidate in the Department of Computer Engineering, METU. He has worked in the Central Bank of the Republic of Turkey as an IT specialist since 2012. His research interests include fault-tolerant topology-control algorithms in wireless sensor networks.

Hakki Bagci received his MS (2007) and PhD (2014) degrees from Bilkent University and Middle East Technical University, respectively, both in Computer Science. He worked as a researcher at the National Scientific and Technological Research Council of Turkey (TUBITAK) from 2004 to 2014. His research interests include distributed-algorithm design for wireless sensor networks.

Ibrahim Korpeoglu is an associate professor in the Department of Computer Engineering, Bilkent University, Ankara, Turkey. He received his MS (1996) and PhD (2000) degrees from University of Maryland at College Park, both in Computer Science. He received his BS degree (summa cum laude) in Computer Engineering from Bilkent University in 1994. Since 2002, he has been a faculty member in the Department of Computer Engineering, Bilkent University. Previously, he worked at several research and development companies in the USA including Ericsson, IBM T.J. Watson Research Center, Bell Laboratories and Bell Communications Research (Bellcore). He received Bilkent University Distinguished Teaching Award in 2006 and the IBM Faculty Award in 2009. He is a member of ACM and a senior member of IEEE.

Adnan Yazici is a full professor in and the chair of the Department of Computer Engineering at Middle East Technical University, Ankara, Turkey. He received his PhD in Computer Science from the Department of EECS at Tulane University in the US, in 1991. His current research interests include intelligent database systems, spatio-temporal databases, multimedia and video databases and wireless multimedia sensor networks. Prof. Dr. Yazici has published more than 180 international technical papers. He received the IBM Faculty Award in 2011 and the Parlar Foundation's Young Investigator Award in 2001. Prof. Dr. Yazici was a Conference Co-Chair of the 23rd IEEE International Conferences on Data Engineering in 2007, the 38th Very Large Data Bases Conference in 2012 and Program Co-Chair of the Flexible Query Answering Systems Conference in 2011. He is the director of the Multimedia Database Lab at METU. He is a member of ACM and a senior member of IEEE.