PACK: Path Coloring based k-Connectivity Detection Algorithm for Wireless Sensor Networks

Orhan Dagdeviren[a,*], Vahid Khalilpour Akram[a]

[a] *International Computer Institute, Ege University, Izmir, Turkey.*

Abstract

A k-connected wireless sensor network (WSN) can tolerate failures on k-1 arbitrary nodes without loosing the connectivity between the remaining active nodes. Hence, the k value is one of the useful benchmarks that can help to measure the network reliability. Given that the nodes in a k-connected network has at least k disjoint paths to each other, we propose the **pa**th **c**oloring based ***k***-connectivity detection algorithm (PACK) that finds the k by counting the disjoint paths between the sink and all other nodes. The proposed algorithm has two Detection and Notification phases. In the Detection phase, all nodes find their disjoint paths to the sink and in the Notification phase the minimum detected path count, which determines the global k, is sent to the sink node. We theoretically prove that the detection range of our proposed algorithm is better than the existing distributed algorithms and uses fixed length messages with $O(n\Delta log_2 n)$ bit complexity and $O(n)$ time complexity where n is the number of nodes and Δ is the maximum node degree. According to the comprehensive simulation results, the average correct detection ratio of proposed algorithm is more than 91% which is at least 2.3 times higher than the existing algorithms.

Keywords: Wireless Sensor Networks, Network Connectivity, k-Connectivity Detection, Distributed Algorithms, Asynchronous Algorithms, Fault Tolerance.

*Corresponding author

Email addresses: orhan.dagdeviren@ege.edu.tr (Orhan Dagdeviren), vahid59@gmail.com (Vahid Khalilpour Akram)

1. Introduction

Wireless sensor networks (WSNs) have a growing application field in military, industry, robotic and health care [1] which increase the importance of reliable communication. Keeping the connectivity between all active nodes is one of the most challenging tasks in WSNs. In these networks the nodes use their neighbors to send the packets to destinations. Hence, failure in some nodes can disconnect the communication path between other nodes. A network is called 1-connected if it has a critical node which its failure divides the network to disconnected parts. These critical nodes, which are also called as *cut vertices*, can be detected by distributed algorithms [2]. In a k-connected network at least k nodes must be removed from the network to completely separate the remaining active nodes. In other words, in a k-connected network we can remove k-1 arbitrary nodes without loosing the connectivity between other nodes. Finding the k value provides useful information for measuring the fault tolerance and the reliability of a given network as well as can lead to significant improvements in connectivity restoration, load balancing and reducing energy consumption.

In this paper, we propose the **pa**th ***c***oloring based ***k***-connectivity detection algorithm (PACK) to detect the k value of a given network with high accuracy and reasonable resource consumption. PACK uses a novel coloring approach in which colors are generated from the sink for each of its neighbors to count the disjoint paths from each node to the sink. PACK is a safe algorithm which means that it does not allow false positive results (higher estimations than the real k) but it may allow for false negative results (estimations with lower values than the real k). We theoretically prove that the detection range of PACK is better than those of its counterparts at the same time its complexity values are favorable. Our theoretical findings confirm with the results obtained from testbed experiments and simulations.

The rest of this paper is organized as follows. A survey on related works has been provided in Section 2. Section 3 includes the network model, motivation, objective and theoretical foundations of this paper. The descriptions of PACK

and with its example operations are given in Section 4. Section 5 includes the theoretical analysis (detection bound analysis and complexity analysis) and Section 6 gives the experimental analysis (testbed experiments and simulations) and comparisons between the algorithms. Finally, the conclusions and outcomes are presented in Section 7.

2. Related Work

The network flow algorithm can be used to find the k value of any graph. If the capacities of all edges in the graph are equal, the network flow algorithm requires at most $O(mn^{2/3})$ time complexity where n is the number of nodes and m is the number of edges. Using the network flow algorithm, it is possible to determine the k of a given graph with $O(m^2n^{1/2})$ time complexity by finding the minimum flow between all pair of nodes [3]. This approach has been improved in many researches [4, 5, 6] and the time complexity has been reduced to $O(mnk)$.

Due to the importance of connectivity in WSNs, various researches have investigated the WSNs from different aspects including the k-connected network establishment by radio power assignment [7, 8, 9, 10, 11, 12, 13, 14, 15, 16], modeling of k-connectivity probability [17, 18, 19, 20, 21, 22, 23, 24, 25], node deployment patterns for k-connectivity [26, 27, 28, 29, 30, 31, 32], k-connected topology management [33, 34, 35, 36, 37], and finally the k-connectivity detection [38, 39, 40] which is the topic of this study.

Three localized distributed k-connectivity detection algorithms for WSNs have been presented in [38]. In the first algorithm, named Local Neighbor Detection (LND), each node sets the local k value to the minimum degree of its p-hop neighbors. Using this algorithm all nodes usually find higher values than the real k because the correct k value is always smaller or at most equal to the minimum degree of nodes. In the second algorithm, named Local Subgraph Connectivity Detection (LSCD), the nodes find their p-hop local subgraph by broadcasting their neighbor list to the other nodes, and run a central algorithm on their subgraph to find a local k value. This local k value is usually lower

than the real k because the local subgraphs omit many remote paths between the nodes. The third algorithm, named Local Critical Node Detection (LCND), is similar to the LSCD and uses p-hop local subgraphs of nodes. In the LCND algorithm, the nodes try to find a cut vertex in their local subgraph. If there is a cut vertex in a local subgraph then the local k is 1. Otherwise, the nodes start removing the vertices from their subgraph one by one, until they find a cut vertex. The number of removed vertices will be the local k value. For the same reason of LSCD, the LCND algorithm usually underestimates k in most of the nodes.

Two safe algorithms for k-connectivity testing problem have been proposed in [39, 40]. These algorithms accept a value as candidate k and return true if the network is at least k-connected. To detect a close value to the real k using the NLT or CWSE algorithms, we should run the algorithm with the incremental values for k. This iterative operation can be very resource consuming as shown in this paper. Distributed K-connectivity Maintenance (DKM) algorithm [34] can detect a local value for k in each node based on the number of its closer neighbors to the sink. However, using the hop distance between nodes and the sink may produce wrong values for k. A distributed algorithm for finding the k value in synchronous networks has been presented in [41]. This algorithm has $O(log\, n)$ approximation ratio and estimates the k in $O(D + \sqrt{n})$ rounds where D is the diameter of the network. The authors have proposed an approach for distributed graph decomposing into multiple spanning trees which decomposes a k-connected graph into vertex disjoint weighted dominating trees where the total weight of each tree is $\Omega(k/log\, n)$. The decomposition algorithm does not need any initial k value. Hence, one of the outcomes of the proposed algorithm is a $O(log\, n)$ approximation for k. Since the algorithm is synchronous, we need an extra synchronizing mechanism to implement it on WSNs which can significantly increase the message overhead, time and energy consumption of detection process. The theoretical analysis and simulation results given in this paper show that our proposed algorithm provides more accurate estimation for k than all existing distributed algorithms while consuming reasonable amount

of resources.

3. Problem Formulation

This section includes the network model, motivation, objective and theoretical foundations for the proposed algorithm.

3.1. Network Model

We assume that each node has a unique id and there is one special node, called as the sink node, which acts as a bridge between the users and other nodes in the network. We further assume that the nodes are stationary during the execution of the algorithm. In mobile or dynamic WSNs, the proposed algorithm should be executed periodically or after adding/removing nodes to find the updated k value. The nodes are randomly distributed in the environment without any coordinate information. Each node in a WSN can communicate with other nodes by sending or receiving radio messages. So, we can assume that there is a communication channel between the nodes which are in radio ranges of each other. Due to different radio ranges of nodes, the communication channels can be unidirectional. Unidirectional communication channels are not generally preferred for reliable data transfer since an acknowledgement based protocol can not be implemented. For example, in Fig. 1 the radio range of node 1 is greater than the radio range of node 0, hence node 0 receives the sent messages by node 1 but node 1 does not. The presented network in Fig. 1 is 1-connected because removing node 2 disconnects node 0 from other nodes. Therefore, the unidirectional link between node 0 and 1 has no effect on k value. In the networks with unidirectional links we can remove (ignore) these links and still find the same k value because removing unidirectional links do not change the k value. To distinguish the unidirectional links, all nodes find their neighbor list (by broadcasting a *Hello* message) and broadcast this list to their neighbors. If node u does not find its id in the received neighbor list from node v, it marks the link to v as unidirectional and ignores all upcoming messages from v.

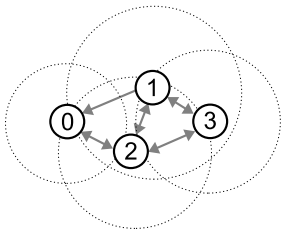

Figure 1: Network Model with Unidirectional Links.

After removing unidirectional links, we can model a WSN as undirected graph G=(V,E) where V is the set of nodes and E is the set of edges.

3.2. Motivation and Objective

The k value determines the number of node failures that a WSN can tolerate without losing its connectivity to the remaining active nodes. Therefore, the k value is one of a good metric for estimating the fault tolerance status and connection reliability of a network. When we identify k=1 then we find that the network has critical node(s) and possible routing bottleneck(s). If $G = (V, E)$ is a k-connected graph, then there are at least k vertex disjoint paths between each $(u, v) \in V$ [42]. Therefore, finding the k value of a network determines the minimum number of vertex disjoint paths between each pair of nodes which is one of the critical information in multi-path routing protocols [43]. Also finding the minimum number of disjoint paths between the nodes can help to distribute routing load more evenly by using different paths for packets flow. Using different paths for packets transmissions may increase the network lifetime.

In a k-connected network, each node has at least k neighbors, hence networks with higher k values are generally denser than those with lower k values [39]. Thus, finding the k value provides an estimation for network density which affects various aspects of WSNs including topology control, sensor coverage and clustering. In a k-connected network, each node shares the commu-

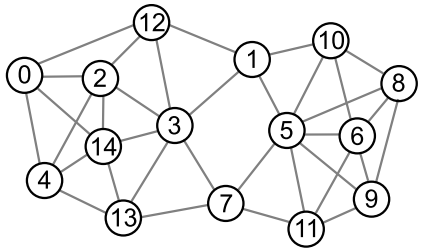

Figure 2: Sample 2-connected WSN.

nication medium with at least k other nodes hence finding the k value can be a significant information for selecting or configuring the medium access control (MAC) protocol. Moreover, detecting the k value is a required phase in most k-connectivity restoration algorithms [33, 44]. After finding the current k, the restoration process can continue (if necessary) with adding new nodes, increasing the radio range of some nodes or moving mobile nodes in the network [28, 33, 44]. Consequently, an efficient k-connectivity estimation algorithm not only reveals various significant information about the network, but also can improve fault tolerance, reliability, connectivity restoration, load balancing and reducing energy consumption in WSNs.

Fig. 2 shows a sample 2-connected network with 15 nodes. If we apply LSCD or LCND algorithm on this network, almost all nodes find $k = 1$ because most of the nodes have at least one critical node in their 2-hop local subgraph. In the NLT or CWSE algorithms almost all nodes reject 2 for k value because the network is not dense enough and the 2-hop local subgraph of many nodes (such as nodes 1, 3, 5 and 7) are 1-connected. In the LND algorithm all nodes find $k \geq 4$, which is far from 2, because the minimum degree of the network is 4. If we suppose that node 0 is the sink, almost all nodes will have at least 3 closer neighbors to the sink, hence they find $k \geq 3$ in DKM algorithm. On the other hand, to find the exact k value using a central algorithm, we should gather the entire topology information in a single node which requires a lot of message passing. In Section 6, we show that using a central algorithm to find k, consumes much more energy than the distributed algorithms. In this manner,

our objective is estimating the k value of a given network with an efficient distributed asynchronous algorithm in terms of detection accuracy and resource usage (time, message, space and computation).

3.3. Theoretical Foundations

The following definitions and notations are used in the rest of the paper.

Definition 1. *$\Gamma(v)$ is the set of neighbors of v. $d(v)$ is $|\Gamma(v)|$ which is referred as the node degree in graph theory. $\delta = min\{d(v_i) : v_i \in V\}$ is the minimum node degree and $\Delta = max\{d(v_i) : v_i \in V\}$ is the maximum node degree in a graph.*

δ is an upper bound for k in G because there is at least one node v in graph G that has exactly δ neighbors and removing all δ neighbors of v disconnects v from other nodes. We give a formal definition of the disjoint path in following.

Definition 2. *A set of paths $P(u,v)= \{p_0(u,v), p_1(u,v),..., p_t(u,v)\}$ between u and v are disjoint if $\forall p_i \in P(u,v)$ and $\forall p_j \in P(u,v)/p_i$ we have $p_i(u,v) \cap p_j(u,v) = \{u,v\}$.*

For example, in the presented 2-connected graph in Fig. 3a the disjoint path set between nodes 0 and 8 is $P(0,8) = \{(0,4,7,8),(0,3,5,8)\}$. The next definition is related with a graph model which includes the set of vertices, edges and layers.

Definition 3. *The graph $G(v)=(V,E,L)$ is obtained from a graph $G=(V,E)$ where $L=\{L(i)$: $0 \leq i \leq D\}$, $L(0) = \{v\}$, $L(1) = \Gamma(v)$ and for $i \geq 1$ the nodes in $L(i)$ is one hop closer to the v than the nodes in $L(i+1)$. We use the term $h_{G(v)}$ to refer to the layer count in $G(v)$.*

Fig. 3b shows an example graph $G(0)$ of the sample graph G given in Fig. 3a. In this graph, $L(2) = \{5,6,7\}$ and $h_{G(0)}$ is 3. Definition 4 provides a classification of a node's neighbors in $G(v)$.

Definition 4. *$\Gamma_x(u)$, $x \in \{-1,0,1\}$, is the set of nodes that are neighbors of $u \in L(i)$ and belongs to $L(i+x)$ in $G(v)$.*

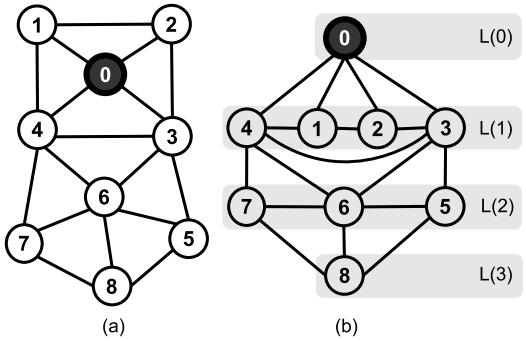

Figure 3: a) Graph G. b) Graph $G(0)$ of G.

For example in Fig. 3b, $\Gamma_1(4) = \{6, 7\}$ and node 6 has two disjoint paths to the sink which are $p_0(6,0)=(6,4,0)$ and $p_1(6,0)=(6,3,0)$. Another classification of the nodes is given in Definition 5.

Definition 5. *$\Lambda(i)$ is the set of nodes $u \in L(i)$ which satisfies $\Gamma_1(u) \neq \emptyset$.*

In Fig. 3b, $\Lambda(1) = \{3, 4\}$ and $\Lambda(2) = \{5, 6, 7\}$. $\Lambda(i)$ is the set of nodes that connects $L(i)$ to $L(i+1)$. We can identify a constraint on k using the definition of $\Lambda(i)$ as following.

Lemma 1. *For any G=(V,E) and $v \in V$, $\lambda = min\{|\Lambda(i)|: 1 < i < h_{G(v)}\}$ is an upper bound for k in G.*

Proof. If we remove the nodes $|u \in \Lambda(i)|$ from $L(i)$ then all nodes in $L(i+1)$ are disconnected from the graph because the remaining nodes in $L(i)/\Lambda(i)$ have no edges with $L(i+1)$. Hence, $\lambda = min(|\Lambda(i)|)$ is an upper bound for k in any $G(v)$ and graph G. □

We can consider the layer with the lowest node count as a bottleneck in $G(v)$ which its removal breaks the connectivity of layers in the graph. Finding the disjoint path count between each pair of nodes in a WSN is a practical way to find the value of k in that network. If each node in a network has k disjoint paths to the sink, then we have at least k paths between any pair of nodes and the network is k-connected [34]. Hence, the value of k is the minimum disjoint path count between each node and the sink. In other words:

$$\forall v \in V : \quad k = min(|P(v, sink)|)$$

If we model the network as a graph G and establish $G(sink)$, we can determine a bounded range of possible values for k as in Theorem 1.

Theorem 1. *For any $G(v)$:*

$$\gamma \leq k \leq min(\lambda, \delta)$$

where $\gamma = min\{|\Gamma_{-1}(u)|: u \in V/(L(0) \cup L(1))\}$.

Proof. According to Lemma 1 for any node $u \in V$ in $G(v)$ we have:

$$k \leq \lambda \quad \wedge \quad k \leq \delta \quad \rightarrow \quad k \leq min(\lambda, \delta)$$

which proves the upper bound limit. We prove the lowest bound by induction.

Base case: Each node in $L(1)$ is directly connected to v. For i=2, each node $u \in L(2)$ has at least γ neighbors in $L(1)$ and consequently γ disjoint paths to v. The possible edges between nodes in $L(2)$ may increase k, so we have:

$$\forall u \in L(2) : \gamma \leq k$$

Inductive step: We assume that each node $u \in L(1) \cup L(2) ... \cup L(i)$ has at least γ disjoint paths to v and prove that every node $u \in L(i+1)$ also has at least γ disjoint paths to v.

As a contradictory assumption let $t \in L(i+1)$ be a node that has at least γ neighbors in $L(i)$ but has less than γ disjoint paths to v. So, there must be a set $S \subseteq V/\{v, t\}$ with size $|S| < \gamma$ which its removal disconnects t from v. We have:

$$|\Gamma_{-1}(t)| \geq \gamma \quad \wedge \quad |S| < \gamma \quad \rightarrow \quad |\Gamma_{-1}(t)| > |S|$$

Let $w \in \Gamma_{-1}(t)/S$ be a node in $L(i)$. According to the inductive assumption w has γ disjoint paths to the v hence removing all nodes in S from G does not disconnect w, and consequently t, from v which is a contradiction. □

4. Proposed Approach

Our proposed asynchronous distributed algorithm aims to achieve accurate and precise detections of k values between the bounds given in Theorem 1 by counting the disjoint paths from each node to the sink and finding the minimum of them in a resource efficient manner. PACK has Detection and Notification Phases where the $Forward/Backward$ messages are broadcasted to find $|P(v_i, sink)|$ for each $v_i \in V$ in the Detection Phase and $Notify/Reply$ messages are sent to find k value in the Notification Phase. To detect $|P(v_i, sink)|$, we count the number of nodes in $L(1)$ that are reachable from v_i through disjoint paths.

4.1. PACK Algorithm

The steps of PACK are given in Alg. 1 and the algorithm works as follows. The sink node selects a color c_i and unicasts *Forward(c_i, sink)* to each of its neighbors. The first parameter c_i is a distinct color that the sink assigns to its neighbor i where this color can be defined as i. The second parameter is the id of the node which the sender received color from it. At the start of the algorithm the sink selects itself as the source of all colors. When the unicast sending finishes, the sink broadcasts all sent colors to its neighbors.

Each node maintains a set C for keeping the pairs of accepted colors and their senders. Each node accepts an incoming color c from the sender w as long as it has not accepted color c before and has not accepted any other color from w. In this way, each accepted color in node u indicates a disjoint path to the sink from u.

Upon receiving a $Forward(c, p)$ in node u, it adds the sender to S_c which is the set of senders of color c. Then u compares p with its own id and if it finds p equals to its own id, it realizes that the sender has accepted its previous broadcasted color and adds the sender to both B_c and R. Otherwise u inserts (c, w) to C if none of the pairs in C has same c or w values. B_c is the set of nodes that node u expects to receive $Backward(c)$ messages from them. R is the set of

nodes that have accepted a color from u and will not accept another color from node u. An accepted color c from w is rebroadcasted by u in a $Forward(c, w)$ message. In this way, the $Forward$ messages are flooded in the network.

Algorithm 1 PACK.

Initially
$C = \emptyset$ // *accepted* $(color, sender)$ *pairs*
$R = \emptyset$ // *receivers that accept a color*
$S_c = \emptyset$ // *senders of color* c
$B_c = \emptyset$ // *expected backward senders for color* c

Detection Phase
$\forall i \in \Gamma(sink)$: the sink **unicasts** $Forward(c_i, sink)$ to i.
$\forall i \in \Gamma(sink)$: the sink **broadcasts** $Forward(c_i, sink)$.
upon receiving $Forward(c, p)$ in u from w:
  u **adds** w to S_c.
  if $p = u$ **then** u **adds** w to both B_c and R.
  else
    if $\nexists (c', s) \in C_u$ **s.t** $c = c'$ **or** $s = w$ **then**
      u **adds** (c, w) to C and **broadcasts** $Forward(c, w)$.
    if $\exists (c', v) \in C$ **s.t** $B_{c'} = \emptyset$ **and** $\Gamma(u) = S_{c'} \cup R$ **then** u **unicasts** $Backward(c)$ to v.
upon receiving $Backward(c)$ in u from w:
  u **removes** w from B_c.
  if u is the sink **then**
    if $\forall i \in \Gamma(u) : B_i = \emptyset$ **and** $\Gamma(u) = R$ **then** u **broadcasts** $Notify(u)$.
  else if $B_c = \emptyset$ **and** $\Gamma(u) = S_c \cup R_u$ **then**
    u **unicasts** $Backward(c)$ to v **where** $(c, v) \in C$.

Notification Phase
upon receiving $Notify(v)$ in u from w:
  if this is the first $Notify$ in u **then**
    u **sets** $prnt \leftarrow w$, $Ch \leftarrow \emptyset$ and **broadcasts** $Notify(w)$.
  else if $v = u$ **then** u **adds** w to Ch.
when u **receives** $Notify(v)$ from all $i \in \Gamma(u)$:
  if $Ch = \emptyset$ **then** u **unicasts** $Reply(|C|)$ to $prnt$.
  else
    u **waits** to **receive** $Reply(c)$ from all $i \in Ch$.
    u **sets** $\forall i \in Ch : k_u \leftarrow min(|C|, c)$.
    if u is the sink **then** u **returns** k_u as k.
    else u **unicasts** $Reply(k_u)$ to $prnt$.

When node u broadcasts a $Forward(c, w)$ message, it can divide its neighbors into 3 groups; the first group is the neighbors that receive the color c from other nodes rather than u and broadcast $Forward(c, v)$ messages where $v \neq u$. Node u adds these neighbors to S_c which is the sender set of color c. The second

group is the neighbors that accept c and rebroadcast $Forward(c, u)$ messages. Node u adds these nodes to R and B_c to remember that:

1. It will not receive a $Forward$ message from any $v \in R$ if it broadcasts a new $Forward$ message in the future.
2. It should wait for $Backward(c)$ messages from all nodes $v \in B_c$ before sending a $Backward(c)$ message.

Finally, the third group is the neighbors that have already accepted another color from node u where R set of node u already includes the ids of these nodes. Therefore, when node u broadcasts a $Forward(c, w)$ it knows that it will not receive a response from any node $v \in S_c \cup R$. It is possible that node u receives a $Forward(c, t)$ with $t \neq u$, after broadcasting a $Forward(c, w)$ which means that its neighbor has accepted c from another node before receiving the $Forward$ from node u. In this case, that neighbor is added only to S_c. Therefore, the termination condition for color c in node u is $S_c \cup R = \Gamma(u)$. When node u finishes the broadcasting of color c, it sends a $Backward(c)$ message to node w (the node that u accepted c from it). If none of the neighbors of u accept c from u, its B_c remains empty and it can send a $Backward(c)$ message upon the termination condition $S_c \cup R = \Gamma(u)$ satisfied. The Detection Phase is terminated when the sink receives $Backward$ messages from all neighbors. In this case, the sink starts the Notification Phase by broadcasting a $Notify(sink)$ message. When the node u receives its first $Notify$ message from node w, it selects the w as its parent and broadcasts a $Notify(w)$ message. In this way each node finds and adds its children to set Ch.

If node u receives $Notify$ messages from all of neighbors and its $Ch = \emptyset$ then it sends a *Replay* message to its parent which includes the accepted color count by u. Otherwise node u waits until it receives the *Reply* messages from its all children. After that node u sends the minimum of received values in *Reply* messages and its accepted color count to its parent. In this way, the sink receives the minimum accepted color count k_u which shows the minimum disjoint path between the sink and all other nodes.

4.2. A Descriptive Example

In this section, we apply the PACK on a sample network and show the exchanged messages in each step. Fig. 4 shows the exchanged messages in *Detection* phase of PACK on a sample network. In Fig. 4a, the sink starts the algorithm by sending 4 different *green*($\mathbf{g}$), *yellow*($\mathbf{y}$), *red*($\mathbf{r}$) and *blue*($\mathbf{b}$) colors to its neighbors. After receiving the green color, node 1 broadcasts this color which is accepted by nodes 2 and 4. Similarly, the broadcasted blue color by node 3 is accepted by nodes {2,4,5,6} and so on. After receiving the broadcasted colors from node 2 and 4 the S sets in node 1 will be $S_1 = \{b\{0\}, g\{0\}, y\{0,2\}, r\{0,4\}\}$ which means that node 1 has received green and blue from 0, yellow from 0 and 2 and red from 4. The R and B sets of node 1 still remain empty but the C set will be $C_1 = \{(g,0),(r,4),(y,2)\}$ which are the accepted colors and their senders in node 1. For the other nodes in Fig. 4a, the R and B sets remain empty but $S_2 = \{r\{0\}, y\{0\}, b\{0,3\}, g\{0,1\}\}$, $C_2 = \{(y,0),(g,1),(b,3)\}$, $S_3 = \{g\{0\}, r\{0,4\}, b\{0\}, y\{0,2\}\}$, $C_3 = \{(r,4),(y,2),(b,0)\}$, $C_4 = \{(r,0),(g,1),(b,3)\}$ and $S_4 = \{y\{0\}, r\{0\}, b\{0,3\}, g\{0,1\}\}$. The nodes rebroadcast their accepted colors, hence node 1 broadcasts yellow and red colors (Fig. 4b) which are not accepted by its neighbors because both nodes 2 and 4 have already accepted green color from 1. In Fig. 4b, we assume that node 1 broadcasts yellow before red (inner circle before outer circle) and the same order holds for the other nodes. The broadcasted blue and green colors by node 2 are not accepted by its neighbors, the broadcasted blue and red colors by node 6 are accepted by nodes 7 and 5 respectively, the broadcasted blue color by node 5 is accepted by node 8 and so on.

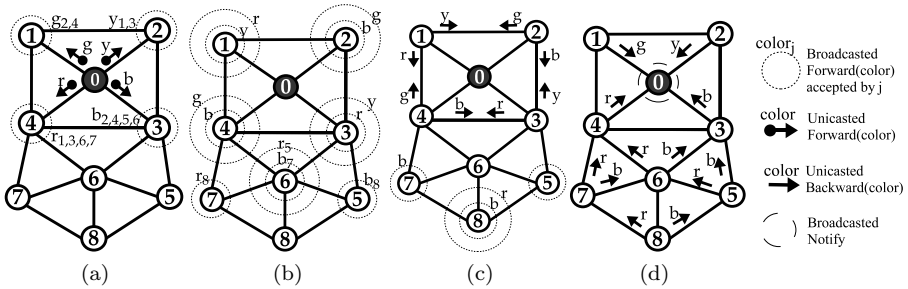

Figure 4: Exchanged Messages in Detection Phase of PACK.

Fig. 4c shows the subsequent *Forward* and *Backward* messages. After receiving the broadcasted colors from 2 and 4, node 1 updates its local sets as $S_1 = \{b\{0,4,2\}, g\{0,4,2\}, y\{0,2\}, r\{0,4\}\}$, $C_1 = \{(g,0),(r,4),(y,2)\}$, $R_1 = \{2,4\}$ and $B_1 = \{g\{4,2\}\}$. For the accepted $(r,4)$ in node 1, we have $B_{1(r)} = \emptyset$ (the B_r set of node 1) and $S_{1(r)} \cup R_1 = \{0,4\} \cup \{2,4\} = \Gamma(1)$, these show that the neighbors of 1 either have accepted another color from 1 or have received red from another node. Therefore, node 1 sends $Backward(r)$ to 4 because it is sure that all of its neighbors has rejected its broadcasted red color. Likewise, node 1 sends $Backward(y)$ to 2, node 2 sends $Backward(b)$ to 3 and $Backward(g)$ to 1 and so on. After receiving $Backward(b)$ from nodes 2 and 3, node 3 removes them from its expected backward set $B_3 = \{b\{5,6\}\}$, which means that it will wait to receive $Backward(b)$ from nodes 5 and 6 before sending $Backward$ to 0. Similarly, the expected backward set of other nodes in Fig. 4c will be $B_4 = \{r\{6,7\}\}$, $B_5 = \{b\{8\}\}$, $B_6 = \{r\{5\}, b\{7\}\}$, $B_7 = \{r\{8\}\}$ and $B_8 = \emptyset$.

Fig. 4d shows the flow of other *Backward* messages in the network. Upon receiving the blue color from 8, node 5 sends $Backward(r)$ to 6 because it finds $B_{5(r)=\emptyset}$ and $S_{5(r)} \cup R = \{3,6\} \cup \{8\} = \Gamma(5)$. After receiving the blue and red colors from node 5 and 7, the local sets of node 8 will be $S_8 = \{b\{5,6,7\}, r\{5,6,7\}\}$, $C_1 = \{(r,7),(b,5)\}$, $R_8 = \emptyset$ and $B_8 = \emptyset$. Since B_8 is empty and $S_{8(b)} \cup R = S_{8(r)} \cup R = \Gamma(8)$, node 8 sends $Backward(b)$ to 5 and $Backward(r)$ to 7. When node 5 receives $Backward(b)$ from 8, it immediately sends $Backward(b)$ to 3 because $B_{5(b)}$ becomes empty and $S_{5(b)} \cup R$ covers $\Gamma(5)$. Node 7 sends

Backward(b) to 6 after receiving blue from node 8 which keeps $B_{7(b)} = \emptyset$ and makes $S_{7(b)} = \{4, 6, 8\} = \Gamma(7)$. Also it sends *Backward*(r) to 4 after receiving red from node 8 which makes $S_{7(r)} = \{4, 6, 8\} = \Gamma(7)$ and $B_{7(r)} = \{\}$. When node 6 receives *Backward*(r) and *Backward*(b) from nodes 7 and 5, its B sets become empty and it sends *Backward*(r) to 4 and *Backward*(b) to 3. In this way, nodes 3 and 4 receive their pending *Backward* messages and send a *Backward* to 0 and node 0 starts *Notification* phase by broadcasting a $Notify(0)$ message.

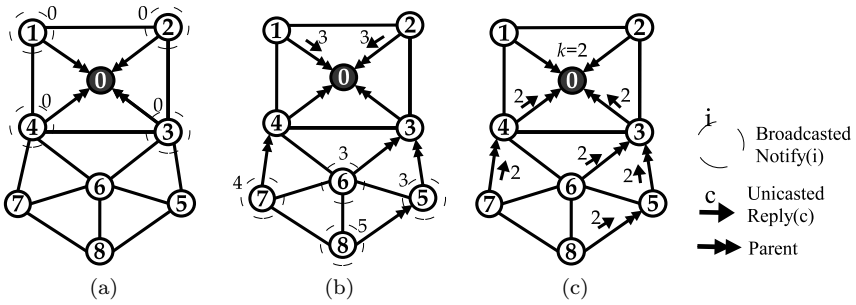

Figure 5: Exchanged Messages in Notification Phase of PACK.

Fig. 5a shows that after receiving $Notify(0)$ from 0, nodes 1, 2, 3 and 4 select 0 as their parent and rebroadcast $Notify(0)$ message. In Fig. 5b, the broadcasted $Notify$ messages by nodes 3 and 4 are received by nodes 1 and 2 and they send *Reply*(3) to 0 (because their Ch list are empty and they have received $Notify$ from all neighbors), where 3 is the cardinality of their C set. At the same time by receiving the $Notify$ message, node 5 selects 3 as its parent and broadcasts $Notify(3)$ and node 7 selects 4 as its parent and broadcasts $Notify(4)$. In Fig.5b, we assumed that node 6 receives $Notify$ from 3 before node 4, hence it selects node 3 as its parent. Therefore, the children lists of node 3 and 4 will be $Ch_3 = \{6, 5\}$ and $Ch_4 = \{7\}$. Similarly, we assume that node 8 receives the broadcasted $Notify$ from node 5 before other nodes and selects 5 as its parent.

Fig. 5c shows the flow of subsequent *Replay* messages. After receiving

$Notify$ from all neighbors, node 8 sends $Replay(2)$ to 5, node 6 sends $Reply(2)$ to 3 and node 7 sends $Reply(2)$ to 4 because their Ch set remains empty. Upon receiving $Reply(2)$ from 8, node 5 removes 8 from its Ch list and sends $Reply(2)$ to 3 where $min(|C_5|, 2)$ is 2. When node 3 receives $Reply(2)$ from 5 and 6, it finds $min(|C_3|, 2, 2) = min(3, 2, 2)$ and sends $Reply(2)$ to 0. Similarly, node 4 sends $Reply(2)$ to 0 and node 0 finds $k = min(3, 3, 2, 2)$.

4.3. An Underestimation Case

PACK may find a lower value than real k in some situations. In this section we show that finding a lower value than real k by PACK, requires various conditions. Fig. 6 shows a sample network with k=2 where node 0 is the sink and PACK potentiality may find $k = 1$ because the closer neighbor count of nodes 5 and 7 to 0 is 1 ($\Gamma_{-1}(5) = \Gamma_{-1}(7) = 1$). In Fig. 6a, to start the PACK algorithm, the sink sends different colors to its neighbors. Assume that nodes 1, 2 and 4 receive and broadcast their colors before receiving the blue color to node 3. The broadcasted blue color by node 4 will be accepted by nodes 1, 3 and 7. Therefore, node 7 will not accept another color from 4. Node 3 may receive yellow from 2, blue from 0 and red from 4 with different orders. Consequently, node 3 may broadcast any of these colors before other. If node 3 broadcasts blue before other colors (Fig. 6b), node 5 accepts blue from 3 and rebroadcasts it which will be accepted by nodes 6, 8 and then 7. Therefore, these nodes accept 2 colors (blue from 3 branch and red from 4 branch) and the algorithm finds correct k=2. Similarly, PACK finds the correct k if node 3 broadcasts yellow before other colors.

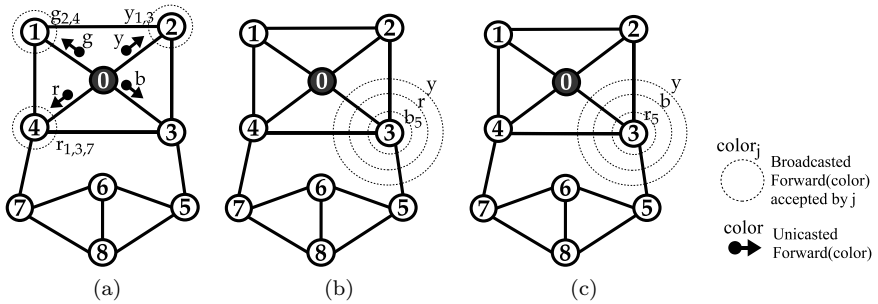

Figure 6: Necessary Conditions for Underestimation of k in PACK.

However, node 5 would accept red from 3 and ignore all upcoming colors from it if node 3 broadcasts red before other colors and if node 5 has still not received red from 6 or 8 (Fig. 6c). In this way both nodes 5 and 7 accept red and ignore future colors. So, the nodes 5, 6, 7, and 8 will receive only 1 color which leads to estimating $k = 1$. Therefore, to underestimate the k in this network, the minimum of closer neighbor count to 0 must be less than 2, node 4 should broadcast its first color before 3, node 3 should receive and broadcast red before blue and yellow and node 5 should receive red from 3 before 6 and 8. In the next section, we will prove that the detected k value by PACK is bounded by γ and k respectively.

5. Theoretical Analysis

We provide the detection bound analysis and the complexity analysis in this section.

5.1. Detection Bound Analysis

We analyze the detection bounds of PACK and validate its correctness and safety properties in this section. We start our analysis by proving the detected lower bound by PACK with the following lemma.

Lemma 2. *The minimum detected value for k in PACK is γ.*

Proof. We use induction to prove this lemma.

Base case: In PACK each node $v \in L(1)$ broadcasts a unique color. Each node $u \in L(2)$ directly receives all broadcasted colors from its neighbor in $L(1)$ and accepts all of them. Hence, each node $u \in L(2)$ accepts at least $|\Gamma_{-1}(u)|$ colors which satisfies the base case.

Inductive step: As the inductive step we assume that each node $u \in L(i)$ accepts at least $\gamma_i = min\{|\Gamma_{-1}(u)|: u \in L(2) \cup L(3)... \cup L(i))\}$ colors from the nodes in upper layers. We should prove that each node $v \in L(i+1)$ accepts at least γ_{i+1} colors.

According to PACK each node rebroadcasts the accepted colors, hence at least γ_i colors are passed from $L(i)$ to $L(i+1)$. If we assume $\rho = min\{|\Gamma_{-1}(u)|: u \in L(i+1))\}$, there is at least one node $u \in L(i+1)$ that can accept at most ρ colors from $L(i)$. Therefore, the minimum accepted color count by any node $u \in L(i+1)$ is $min(\gamma_i, \rho)$ which is equal to γ_{i+1}. □

Lemma 3. *Each accepted color in node v represents a disjoint path between v and the sink.*

Proof. We prove by induction.

Base case: Each node $u \in L(1)$ that receives a $Forward(c, sink)$ message from the sink as a unicast message, finds a disjoint path $p_c(u, sink) = \{u, sink\}$ to the sink and broadcasts the color c to its neighbors. Therefore, each node v that receives a broadcasted $Forward(c, sink)$ message from node u finds a path $p_c = \{v, u, sink\}$ to the sink. This path is disjoint because node v does not use u in future paths (i.e it does not accept a new color from node u) and also node v ignores all broadcasted $Forward(c, t)$ messages from the other nodes as all of them begin from u.

Inductive step: If we assume that node w which broadcasts $Forward(c, t)$ has a disjoint path $p_c(w, sink)$ to the sink, then node v that receives the color c for the first time can create a path as $p_c(v, sink) = \{v\} \cup p_c(w, sink)$ and keep it disjoint by ignoring all messages from w and all messages with color c in the future. □

Theorem 2. *In any network $\gamma \leq k_{pack} \leq k$ where k_{pack} is the detected connectivity value by PACK.*

Proof. According to Lemma 2 the minimum detected value by PACK is γ. According to Lemma 3 each accepted color in node v indicates a disjoint path between the sink and v. So, the maximum accepted color count in each node is k. Therefore, the detection range for k is $\gamma \leq k_{pack} \leq k$. □

Theorem 3. *The range for the k values detected by the LND algorithm (k_{LND}) is $\delta \leq k_{LND} \leq \Delta$.*

Proof. In LND, each node selects the minimum node degree of its 2-hop neighbors as k. Therefore, in LND the minimum and maximum detected values in each node are δ and Δ respectively. □

Theorem 4. *The range for the k values detected by the LSCD and LCND algorithms (k_{LSCD},k_{LCND}) is $1 \leq k_{LSCD}, k_{LCND} \leq \Delta$.*

Proof. The maximum value that LSCD and LCND may detect for k is Δ because each node in its local subgraph has at most Δ neighbors. The minimum detected value in these algorithms are 1 because it is possible that we have a node with degree 1 in the established p-hop subgraph. □

Theorem 5. *The range for the k values detected by the NLT and CWSE algorithms (k_{NLT},k_{CWSE}) are $1 \leq k_{NLT}, k_{CWSE} \leq k$.*

Proof. Both NLT and CWSE are safe hence the detected values by these algorithms are equal or smaller than the real k. The minimum detected value by these algorithms is 1 because in both algorithms having a cut vertex in p-hop local subgraph of a single node is enough for rejecting 2 and detecting k as 1. □

Theorem 6. *The range for the k values detected by the DKM algorithm (k_{DKM}) is $\gamma \leq k_{DKM} \leq \Delta$.*

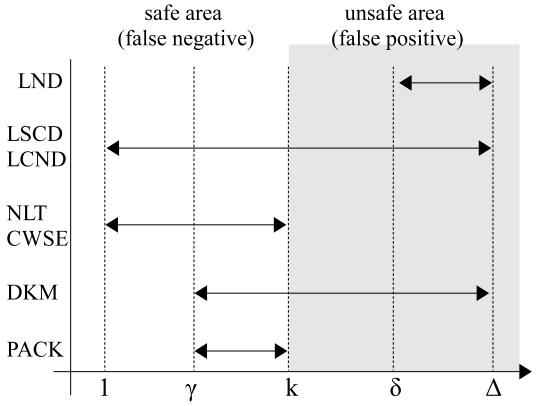

Figure 7: Detection range for k in various algorithms ($1 \leq \gamma \leq k \leq \delta \leq \Delta$).

Proof. The minimum detected value for k in DKM is the minimum size of any node v's set of support nodes. In other words, it is the number of node v's one hop neighbors which are closer to the sink where this set equals to $\Gamma_{-1}(v)$. Therefore, the minimum detected value by DKM is γ according to Theorem 1. The upper bound of detected values for k in DKM is Δ because each node may add its all one hop neighbors to the set of support nodes. □

Fig. 7 shows the detection ranges for k in various algorithms. Among these algorithms PACK has the best detection range theoretically. To support this finding, we provide testbed experiments and simulations in the following sections.

5.2. Complexity Analysis

In this section, we provide bit, time, space and computational complexities of the proposed algorithm. At the end of this section, we compare the proposed algorithm with the related work in terms of complexities.

Theorem 7. *The bit complexity of PACK is $O(n\Delta log_2 n)$ in the worst case and $\Omega(nlog_2(n)k)$ in the best case.*

Proof. In the Detection Phase of PACK each node receives at most $d(sink)$ new colors. In the worst case, all nodes accept and rebroadcast all $d(sink)$ colors

which leads to $nd(sink)$ messages. Considering $d(sink) \leq \Delta$ the maximum number of broadcasted $Forward$ and $Backward$ messages will be $O(n\Delta)$. Each *Forward* or *Backward* message carries a color $c < \Delta$ and an $id < n$, so the length of messages will be at most $log_2 n + log_2 \Delta \in O(log_2 n)$ bits. Thus, the bit complexity of the Detection phase is $O(n\Delta log_2 n)$.

In the Notification phase, all nodes send their detected k values to the sink and to do this, each node sends exactly one $Notify$ message and one *Reply* message. The maximum length of *Notify* and *Reply* messages are $O(log_2 n)$ and $O(log_2 \Delta)$ respectively. Consequently, the bit complexity of reporting the estimated k values to the sink is $O(nlog_2 n)$. Therefore, the bit complexity of PACK is $O(n\Delta log_2 n + nlog_2 n) \in O(n\Delta log_2 n)$. In the best case, all nodes broadcast only k (instead of Δ) colors, which reduces the bit complexity to $\Omega(nlog_2(n)k)$ bits. □

Theorem 8. *The time complexity of PACK is $O(n)$ in the worst case and $\Omega(D)$ in the best case.*

Proof. Broadcasting a single data to the entire network takes $\Omega(D)$ time in the best case. Flooding of all colors are concurrent hence we can assume that there is only one $Forward$ message flooded in the network which is replied by $Backward$ messages. Beside the $Forward/Backward$ messages, PACK floods $Notify/Replay$ messages in the network. Therefore, in PACK we have two broadcast/convergecast phases in the entire operation. The time complexity of the PACK is dominated by the convergecast operation which is $O(D)$ at the best case and $O(n)$ at the worst case. □

Theorem 9. *The space complexity of PACK is $O(n\Delta log_2 n)$ bits.*

Proof. PACK keeps the received colors and *id*s of their senders in C, R, S, B and Ch sets to detect the new colors and senders. Each *id* or color requires $log_2 n$ bits in a memory of a node. The maximum color count in the algorithm is limited by the sink's neighbor count which is bounded by Δ. Therefore, the

	Bit	Time	Space	Comp.
LND	$O(nlog_2\Delta)$	$O(1)$	$O(nlog_2\Delta)$	$O(n\Delta)$
PACK	$O(n\Delta log_2 n)$	$O(n)$	$O(n\Delta log_2 n)$	$O(n\Delta^2)$
DKM	$O(n\Delta^2 log_2 n)$	$O(n)$	$O(n\Delta^2 log_2 n)$	$O(n\Delta^3)$
LCND	$O(n\Delta^3 log_2 n)$	$O(\Delta)$	$O(n\Delta^3 log_2 n)$	$O(n\Delta^6)$
LSCD	$O(n\Delta^3 log_2 n)$	$O(\Delta)$	$O(n\Delta^3 log_2 n)$	$O(n\Delta^6 k)$
CWSE	$O(n\Delta^3 log_2(n)k)$	$O(\Delta k)$	$O(n\Delta^3 log_2 n)$	$O(n\Delta^6)$
NLT	$O(n\Delta^3 log_2(n)k)$	$O(\Delta k)$	$O(n\Delta^3 log_2 n)$	$O(n\Delta^6 k)$
CENTRAL	$O(n^2\Delta log_2 n)$	$O(n)$	$O(n^2)$	$O(n^3 k)$

Table 1: Analytical Comparison of k-Connectivity Detection Algorithms.

maximum size of all sets in each node is Δ and the total space complexity in the entire network is $O(n\Delta log_2 n)$ bits. □

Theorem 10. *The computational complexity of PACK is* $O(n\Delta^2)$.

Proof. Upon receiving a *Forward* message each node compares the received color and its sender with the previous received colors and senders to decide whether to discard or accept the color. The maximum color and neighbor counts in each node are Δ and each node may search the accepted color set at most Δ times. Hence, the total computational complexity in the entire network is $O(n\Delta^2)$. □

Table 1 shows an analytical comparison of k-connectivity detection algorithms. LND has the best complexity values among other algorithms. However, it usually overestimates k and its detection range is very higher than PACK algorithm. PACK has the second best complexity values in terms of bit, space and computational complexities.

In CENTRAL algorithm we gather the entire topology information in the sink node. To do this, the sink broadcasts a *Hello* message to its neighbors. Each node that receives the first *Hello* message selects the sender as its parent and rebroadcasts the *Hello* message. In this way, each node learns its neighbors and constructs a path to the sink node. The neighbor list of each node is sent along the constructed path to the sink. After getting the neighbor list of all nodes, the sink creates a graph of the network topology, which leads to O(n^2) space complexity, and runs a central k-connectivity detection algorithm [6] to find the

k. In the worst case, if the network has a linear topology, $n(n+1)/2$ neighbor list messages are sent in the network in $O(n)$ time where the maximum size of each message is $O(\Delta log_2 n)$. After k value is found by the sink node, k value is reported to each node in $O(D)$ time with $O(n)$ messages which are $O(log_2 k)$ bits. Therefore, the total bit, time, space and computational complexities are $O(n^2 \Delta log_2 n)$, $O(n)$, $O(n^2)$ and $O(n^3 k)$ respectively.

6. Experimental Analysis

The results of experimental analysis on a real testbed and a simulation environment have been presented in this section.

6.1. Testbed Experiments

To evaluate the performance of the proposed algorithm on real networks, we implemented the PACK, CWSE, DKM, LND, LSCD and CENTRAL algorithms on TinyOS [45] using 20 Crossbow IRIS [46] motes with a 128 kB programmable flash memory, 8 kB RAM and 2.4 GHz IEEE 802.15.4 compliant transceiver. We established 10 random topologies with 20 nodes and different k values in range of 1 to 5 and run each algorithm on all topologies. To establish the intended topologies in laboratory environment we reduced the transmission power of IRIS motes, which restricts their radio range, and also increased the minimum received signal strength indicator (RSSI) threshold of receiver nodes to only receive the packets with high signal strength. By adjusting these two parameters (transmission power and RSSI) we established the desired WSNs with arbitrary k value in laboratory environment.

In setup phase, before executing the algorithms, we have used the explained method in Section 3.1 to avoid the unidirectional links. Since all algorithms assume bidirectional links and the message passing overhead of setup phase for all algorithms are same we did not count this overhead in the performance measurements. To avoid the packet collisions, we used a carrier sense multiple access/collision avoidance MAC protocol with extra random waiting time (between 100 ms and 1000 ms) before sending the packets. Using up to 1000 ms

	MSE	Sent	Received	FP	FN	Correct	Time(s)
CENTRAL	0	2032	17354	0	0	100	10.4
PACK	0.061	771	5483	0	10	90	8.2
LCND	0.414	1998	17060	5	38	57	9.8
LSCD	0.648	1998	17060	4	42	54	9.9
NLT	1.09	3313	28207	0	60	40	11.3
CWSE	1.537	3289	28035	0	70	30	11
DKM	1.947	1117	8441	59	13	28	7.6
LND	2.738	210	1393	77	0	23	2.1

Table 2: Results of Testbed Experiments.

random delays slightly increases the wallclock time, but in our small networks (20 nodes), almost eliminates the interference possibility.

Table 2 shows the average mean square errors (MSE), sent bytes (Sent), received bytes (Received), false positive detections percentage (FP), false negative detections percentage (FN), correct detection percentage (Correct) and wallclock times (Time) of algorithms. The correct detection percentage has been calculated as $correct = 100 - (FP + FN)$ and the table has been sorted according to the MSE values.

The LND has the lowest time and sent/received bytes, but the MSE value of this algorithm is higher than all other algorithms and only 23% of the nodes find the correct k using this algorithm. After LND, the next algorithm with the lowest sent/received bytes is PACK whereas its wallclock time is the third best among the other algorithms. Furthermore, it has detected the closest k value to the real k among non-exact k connectivity detection algorithms (i.e. excluding CENTRAL) with 0.061 MSE and 90% correct detection ratio. CWSE, NLT, LSCD and LCND consume more resources than PACK while their produced correct ratios vary between 30% and 57%. Even though, CENTRAL always finds the correct k, its sent and received bytes are higher than all other algorithms. To evaluate the algorithms on larger graphs, we provide simulations in the next section.

6.2. Simulations

We simulated PACK and all other algorithms in TOSSIM simulator [47]. TOSSIM is a built-in TinyOS simulator which can simulate the source code of the real motes on large networks. TOSSIM allows to define the propagation model and network topology as a directed weighted graph where the vertices are nodes and weight of each edge is the received signal strength between the sender and receiver which must be higher than a threshold value to allow the message passing. Most of the existing researches on k-connectivity usually accept the networks with $k \geq 1$ and $k \leq 6$ [12, 13, 19, 27, 28, 29, 31, 38], hence we have considered the range [1..6] for k value in our simulations.

We defined 5 general classes of geometric random bidirectional graphs with different node counts starting from 50 up to 250 nodes (stepping 50 nodes at each class). For each class, we created random networks with 6 different k values starting from 1 to 6 (stepping 1) and 5 different transmission ranges starting from 10 m to 30 m (steeping 5 m) which leads to 150 different topology types. For each topology type with specific node count, transmission range and k value, we created 10 random instances which leads to 1500 random topologies in total. Considering that the generated graphs are bidirectional we did not apply the setup phase in the simulation environment.

To create a geometric random bidirectional topology with specific node count n, transmission range r and desired k value, we assigned an *id* to each node (starting from 0 to $n-1$) and randomly distributed them in a 1000 m $\times$ 1000 m area. The (x, y) coordinates of each node is uniformly selected between 0 and 1000. After that, we created a bidirectional link between the nodes that their distance is less than r. Then, we found the k value of generated topology with a central k-connectivity detection algorithm [6] and repeated the generation process until we have 10 instances of each topology type.

To eliminate the packet interferences, we used a 3-hop coloring based time division multiple access (TDMA) protocol for the MAC layer. The color of each node determines its slot number in the current frame of TDMA protocol. The nodes with the same slot number can broadcast their packets at the same time

Simulator	TinyOS-TOSSIM
Target mote	IRIS
Topology count	1500
Node count	From 50 up to 250, stepping 50
Transmission range	From 10 m up to 30 m, stepping 5 m
MAC protocol	3-hop coloring TDMA
Slot length of TDMA	100 ms
Frame length of TDMA	100 ms × (color count)
k	From 1 up to 6, stepping 1
Node distribution	Random distribution
Network model	Custom topology
Area	$1000 \times 1000\ m^2$

Table 3: Key parameters of simulation environment.

without collision possibility because there will be at least 2 hops between the nodes with the same color. This will lead to shorter frame length in TDMA protocol and reduce the wall clock time of algorithms in the simulation. Table 3 summarizes the properties of our simulation environment.

Calculating the consumed energy by the nodes is not supported in TOSSIM, hence we calculated the consumed energy in each node from the sent bytes (S) and received bytes (R). The transmission data rate of IRIS motes is 250 kbps (i.e 31.25 kBps) and they consume about 16 mA current in receive mode and 17 mA in sent mode with $TX = 3\ dBm$ [46]. The input voltage of these motes is 3300 mV. Considering the general formula $E = I * V$ we have:

$$E \approx ((S * 17 + R * 16)/31.25) * 3.3$$

For the proposed PACK algorithm we aggregated the time and energy consumptions of both *Detection* and *Notification* phases in all simulation results. Fig. 8 shows the wallclock time of PACK for various k values. Fig. 8 shows that the wallclock time of PACK is a linear function of node count and the k value determines the slope of line. On average, PACK takes about 10 s in the networks with 50 nodes and 68 s in the networks with 250 nodes. Fig. 9 compares the wallclock times of algorithms over various node counts. The wallclock time of

each algorithm in this figure is the average of wallclock time for all k and r values. The wallclock time of LND is very lower than the other algorithms. DKM, LSCD and LCND have shorter wallcock times than PACK because all of them find the k value based on local information, but PACK finds the global k value. The NLT and CWSE have longer wallclock times than PACK because they are called with incremental k values. Also the CENTRAL (CENT) algorithm takes more time than PACK because in this algorithm the neighbor list of each node must be transmitted to the sink node.

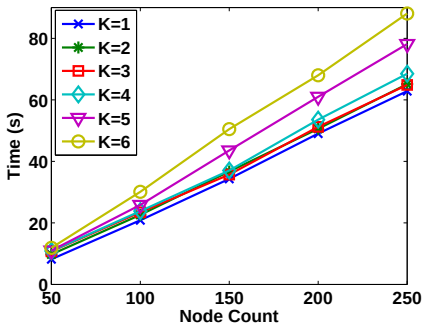

Figure 8: Wallclock times of PACK against node count.

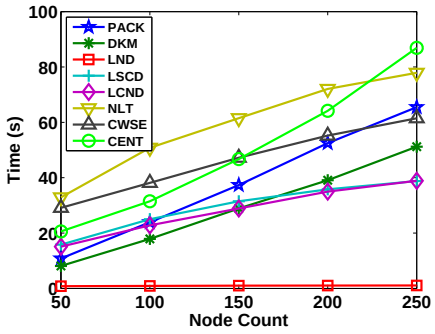

Figure 9: Comparison of wallclock times of algorithms.

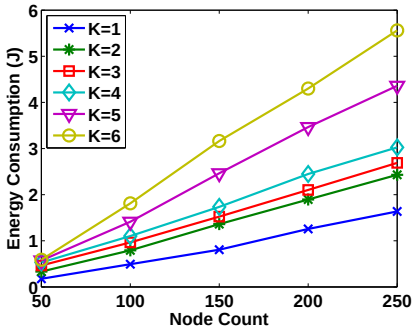

Figure 10: Energy consumption of PACK against node count.

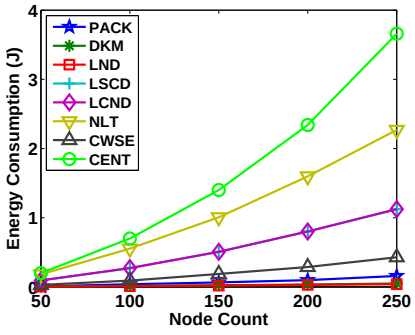

Figure 11: Comparison of energy consumption of algorithms.

Fig. 10 shows the energy consumption of PACK for different k values over various node counts. The energy consumption of PACK is almost a linear function of node count and the k value determines the slope of line. In the 6-connected networks with 250 nodes, PACK consumes about 5.5 J energy which is about 3.5 times higher than the 1-connected networks with the same node count. Fig. 11 compares the consumed energy in all algorithms. CENTRAL algorithm has the highest sent and received bytes among the other algorithms hence it consumes more energy than the others. NLT has the highest energy consumption after CENTRAL because it may run the entire algorithm many times. The energy consumption of LCND and LSCD algorithms are equal because they sent equal amount of bytes. CWSE usually rejects the proposed k value earlier than NLT, hence its energy consumption is lower than NLT. LND, DKM and PACK have significantly lower energy consumptions than the other algorithms.

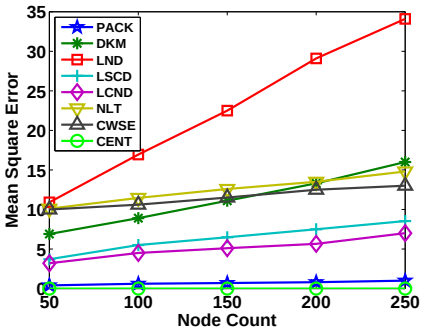

Figure 12: Comparison of mean square errors of detected k by algorithms.

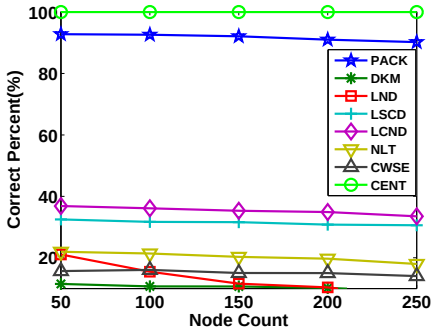

Figure 13: Comparison of correct detection ratios of algorithms.

The average mean square errors of detected k value by various algorithms have been presented in Fig. 12. The mean square error of CENTRAL is always 0 because it always finds the exact k value. LND has the highest mean square errors among the other algorithms because it usually finds very higher values than the correct k. The mean square errors of DKM, NLT and CWSE differ from 6 to 16 and the mean square errors of LSCD and LCND are less than 10.

The mean square errors of PACK are always less than 1 in all networks which means that PACK usually finds the correct or a very close value to the real k. Fig. 13 compares the correct detection ratios of k in various algorithms. The correct detection percentage of CENTRAL is 100% in all topologies because it always finds the correct answer. After CENTRAL, PACK has the next highest correct detection ratio which is more than 91% for all topologies. The correct detection ratio of other algorithms are less than 39% which are at least 2.3 times lower than PACK.

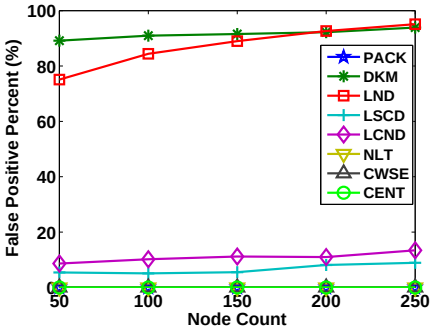

Figure 14: Comparison of false positive detection ratios of algorithms.

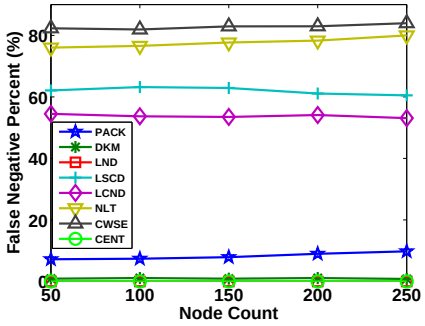

Figure 15: Comparison of false negative detection ratios of algorithms.

Fig. 14 shows the false positive detection ratios of k. The false positive detection ratios of PACK, NLT, CWSE and CENTRAL are 0 because they are safe algorithms. In all topologies, the false positive detection ratios of LCND and LSCD are less than 10% while the false positive detection ratios of LND and DKM are more than 70%. Fig. 14 shows that LND and DKM overestimate k most of the time because in both algorithms the nodes detect k based only on their neighbor count.

The false negative detection ratios of the algorithms are given in Fig. 15. The false negative detection ratios of LND are always 0 because the minimum of node degrees is the upper bound for k. Also, the false negative detection ratio of CENTRAL algorithm is always 0. The false negative detection ratios of DKM are less than 1% in most topologies but DKM suffers from the false

positive detections as shown in Fig. 14. The false negative detection ratios of PACK are less than 9% in all simulation setups. The false negative detection ratios of LCND and LSCD vary between 55% and 65%. The false negative detection ratios of NLT and CWSE are more than 77%. These results show that our theoretical analysis generally conform with the experimental analysis that PACK outperforms its counterparts in terms of detection accuracy at the same time consuming reasonable amount of resources.

7. Conclusions

In this paper, we proposed an algorithm for k-connectivity detection in WSNs. Our proposed algorithm called PACK detects k in two phases where paths are colored in the Detection Phase and k value is found in the Notification Phase. We show through theoretical and experimental analysis that PACK is a distributed, asynchronous and safe algorithm and at the same time it detects k values within a better range than the other non-exact detection algorithms by consuming time and energy wisely. We obtained from testbed experiments and simulations that the correct detection ratio of PACK is more than 91% which at least 2.3 times better than the other non-exact detection algorithms. The MSE of PACK is less than 1 in all topologies and after LND, which has very high MSE, it is the second best algorithm among k-connectivity detection algorithms in terms of energy consumption. These results show that PACK is a significant contribution to the resource-efficient k-connectivity detection in WSNs.

Acknowledgment

This work was supported by the TUBITAK (Scientific and Technical Research Council of Turkey) [Project number 113E470]. Authors would like to thank the anonymous reviewers for their valuable suggestions to improve the quality of the paper.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE communications magazine 40 (8) (2002) 102–114.

[2] O. Dagdeviren, V. K. Akram, An energy-efficient distributed cut vertex detection algorithm for wireless sensor networks, Comput. J. 57 (12) (2014) 1852–1869.

[3] S. Even, R. E. Tarjan, Network flow and testing graph connectivity., SIAM J. Comput. 4 (4) (1975) 507–518.

[4] S. Even, An algorithm for determining whether the connectivity of a graph is at least k, SIAM J. Comput. 4 (3) (1975) 393–396.

[5] Z. Galil, Finding the vertex connectivity of graphs, SIAM J. Comput. 9 (1) (1980) 197–199.

[6] M. R. Henzinger, S. Rao, H. N. Gabow, Computing vertex connectivity: New bounds from old techniques, J. Algorithms 34 (2) (2000) 222–250.

[7] N. Li, J. C. Hou, Flss: A fault-tolerant topology control algorithm for wireless networks, in: Proc. of the MobiCom '04, ACM, 2004, pp. 275–286.

[8] Z. Nutov, Approximating minimum-power k-connectivity., Ad Hoc and Sensor Wireless Networks 9 (1-2) (2010) 129–137.

[9] B. Gupta, A. Gupta, On the k-connectivity of ad-hoc wireless networks, in: Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, SOSE '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 546–550.

[10] D. Tian, N. D. Georganas, Connectivity maintenance and coverage preservation in wireless sensor networks, Ad Hoc Networks 3 (6) (2005) 744–761.

[11] M. Segal, H. Shpungin, On construction of minimum energy k-fault resistant topologies, Ad Hoc Networks 7 (2) (2009) 363–373.

[12] F. Deniz, H. Bagci, I. Korpeoglu, A. Yazıcı, An adaptive, energy-aware and distributed fault-tolerant topology-control algorithm for heterogeneous wireless sensor networks, Ad Hoc Networks 44 (2016) 104–117.

[13] H. Bagci, I. Korpeoglu, A. Yazıcı, A distributed fault-tolerant topology control algorithm for heterogeneous wireless sensor networks, IEEE Transactions on Parallel and Distributed Systems 26 (4) (2015) 914–923.

[14] P.-J. Wan, C.-W. Yi, Asymptotic critical transmission radius and critical neighbor number for k-connectivity in wireless ad hoc networks, in: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing, ACM, 2004, pp. 1–8.

[15] X. Jia, D. Kim, S. Makki, P.-J. Wan, C.-W. Yi, Power assignment for k-connectivity in wireless ad hoc networks, Journal of Combinatorial Optimization 9 (2) (2005) 213–222.

[16] H. Zhang, J. Hou, On the critical total power for asymptotic k-connectivity in wireless networks, in: Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies., Vol. 1, IEEE, 2005, pp. 466–476.

[17] J. Zhao, O. Yağan, V. Gligor, On the strengths of connectivity and robustness in general random intersection graphs, in: 53rd IEEE Conference on Decision and Control, IEEE, 2014, pp. 3661–3668.

[18] J. Zhao, O. Yağan, V. Gligor, Exact analysis of k-connectivity in secure sensor networks with unreliable links, in: Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2015 13th International Symposium on, IEEE, 2015, pp. 191–198.

[19] J. Zhao, Minimum node degree and k-connectivity in wireless networks with unreliable links, in: 2014 IEEE International Symposium on Information Theory, IEEE, 2014, pp. 246–250.

[20] F. Yavuz, J. Zhao, O. Yağan, V. Gligor, On secure and reliable communications in wireless sensor networks: Towards k-connectivity under a random pairwise key predistribution scheme, in: 2014 IEEE International Symposium on Information Theory, IEEE, 2014, pp. 2381–2385.

[21] O. Georgiou, C. P. Dettmann, J. P. Coon, k-connectivity for confined random networks, Europhys. Lett.

[22] S. G. Natarajan Meghanathan, On the probability of k-connectivity in wireless ad hoc networks under different mobility models, Int. Journal on App. of Graph Theory in Wireless ad hoc Ntws. and Sensor Ntws. 2.

[23] K. Chan, F. Fekri, et al., On connectivity properties of large-scale sensor networks, in: Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on, IEEE, 2004, pp. 498–507.

[24] Q. Ling, Z. Tian, Minimum node degree and k-connectivity of a wireless multihop network in bounded area, in: IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference, IEEE, 2007, pp. 1296–1301.

[25] J. Zhao, O. Yağan, V. Gligor, On k-connectivity and minimum vertex degree in random s-intersection graphs, in: Proceedings of the Meeting on Analytic Algorithmics and Combinatorics, Society for Industrial and Applied Mathematics, 2015, pp. 1–15.

[26] S. K. Gupta, P. Kuila, P. K. Jana, Genetic algorithm for k-connected relay node placement in wireless sensor networks, in: Proceedings of the Second International Conference on Computer and Communication Technologies, Springer, 2016, pp. 721–729.

[27] J. L. Bredin, E. D. Demaine, M. T. Hajiaghayi, D. Rus, Deploying sensor networks with guaranteed fault tolerance, IEEE/ACM Transactions on Networking (TON) 18 (1) (2010) 216–228.

[28] H. M. Almasaeid, A. E. Kamal, On the minimum k-connectivity repair in wireless sensor networks, in: Proc. of the IEEE ICC'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 195–199.

[29] Z. Yun, X. Bai, D. Xuan, T. H. Lai, W. Jia, Optimal deployment patterns for full coverage and k-connectivity ($k \leq 6$) wireless sensor networks, IEEE/ACM Transactions on Networking (TON) 18 (3) (2010) 934–947.

[30] M. Younis, K. Akkaya, Strategies and techniques for node placement in wireless sensor networks:a survey, Ad Hoc Networks 6 (4) (2008) 621–655.

[31] X. Bai, D. Xuan, Z. Yun, T. H. Lai, W. Jia, Complete optimal deployment patterns for full-coverage and k-connectivity ($k \leq 6$) wireless sensor networks, in: Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc '08, ACM, New York, NY, USA, 2008, pp. 401–410.

[32] J. Barrera, H. Cancela, E. Moreno, Topological optimization of reliable networks under dependent failures, Operations Research Letters 43 (2) (2015) 132–136.

[33] S. Wang, X. Mao, S.-J. Tang, X. Li, J. Zhao, G. Dai, On movement-assisted connectivity restoration in wireless sensor and actor networks, IEEE Trans. Parallel Distrib. Syst. 22 (4) (2011) 687–694.

[34] P. Szczytowski, A. Khelil, N. Suri, Dkm: Distributed k-connectivity maintenance in wireless sensor networks., in: WONS, IEEE, 2012, pp. 83–90.

[35] N. Atay, O. B. Bayazit, Mobile wireless sensor network connectivity repair with k-redundancy., in: WAFR, Vol. 57 of Springer Tracts in Advanced Robotics, Springer, 2008, pp. 35–49.

[36] M. Younis, I. F. Senturk, K. Akkaya, S. Lee, F. Senel, Topology management techniques for tolerating node failures in wireless sensor networks: A survey, Computer Networks 58 (2014) 254–283.

[37] L. Zhang, X. Wang, W. Dou, Design and analysis of a k-connected topology control algorithm for ad hoc networks, in: International Symposium on Parallel and Distributed Processing and Applications, Springer, 2004, pp. 178–187.

[38] M. Jorgic, N. Goel, K. Kalaichelvan, A. Nayak, I. Stojmenovic, Localized detection of k-connectivity in wireless ad hoc, actuator and sensor networks, in: ICCCN, IEEE, 2007, pp. 33–38.

[39] A. Cornejo, N. Lynch, Fault-tolerance through k-connectivity, in: Workshop on Network Science and Systems Issues in Multi-Robot Autonomy: ICRA 2010, Vol. 2, 2010.

[40] A. Cornejo, N. Lynch, Reliably detecting connectivity using local graph traits, in: Proceedings of the 14th International Conference on Principles of Distributed Systems, OPODIS'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 87–102.

[41] K. Censor-Hillel, M. Ghaffari, F. Kuhn, Distributed connectivity decomposition, in: Proceedings of the 2014 ACM symposium on Principles of distributed computing, ACM, 2014, pp. 156–165.

[42] G. Dirac, Short proof of menger's graph theorem, Mathematika 13 (01) (1966) 42–44.

[43] O. Yilmaz, O. Dagdeviren, K. Erciyes, Localization-free and energy-efficient hole bypassing techniques for fault-tolerant sensor networks, J. Netw. Comput. Appl. 40 (2014) 164–178.

[44] N. Atay, B. Bayazit, Mobile wireless sensor network connectivity repair with k-redundancy, in: Algorithmic Foundation of Robotics VIII, Springer, 2009, pp. 35–49.

[45] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler, Tinyos: An operating system for sensor networks, in: in Ambient Intelligence, Springer Verlag, 2004.

[46] M. Inc, Iris datasheet.
URL http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf

[47] P. Levis, N. Lee, M. Welsh, D. Culler, Tossim: Accurate and scalable simulation of entire tinyos applications, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys '03, ACM, New York, NY, USA, 2003, pp. 126–137.