International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 4, Number 2, pages 336–354, July 2014

# Employing Cooperative Communication to Recover Network Connectivity in Ad Hoc Networks

Ulisses Rodrigues Afonseca
Instituto Federal de Goiás - IFG
Campus Luziânia, 73850-000, Luziânia/Goiás, Brazil
e-mail: urafonseca@ifg.edu.br

Thiago Fernandes Neves
Department of Computer Science
University of Brasília - UnB, 70910-900, Brasília, Brazil
e-mail: tfn.thiago@cic.unb.br

and

Jacir Luiz Bordim
Department of Computer Science
University of Brasília - UnB, 70910-900, Brasília, Brazil
e-mail: bordim@unb.br

Received: February 14, 2014
Revised: May 1, 2014
Accepted: June 3, 2014
Communicated by Akihiro Fujiwara

**Abstract**

In wireless ad hoc networks, bridges and articulation nodes are critical elements that, in case of failure, render the network disconnected. Owing to their relevance, a number of works try to extend the life span of these elements. Nevertheless, in critical situations, such as the unavailability of a critical link, ways to reestablish the communication, even if for short periods of time, can be of importance in a number of urgent tasks. In this context, this work explores the concept of Cooperative Communication (CC) to monitor critical nodes and links and recover network connectivity in case of disruption. Unlike other works that perform exhaustive search to locate suitable CC-links that require global topology information, the proposed scheme identifies critical nodes and links based solely on local information. Compared to other prominent works, the proposed solution was able to reduce the computing cost to create CC-links in $\approx 67$ times in the evaluated scenarios while persisting a lower message overhead.

*Keywords:* Ad Hoc Networks, Articulation, Bridge, Connectivity Recovery, Cooperative Communication, Distributed Algorithms

## 1 Introduction

Ad hoc networks have been envisioned as an alternative solution to support urgent, critical and temporary tasks such as search and rescue, law-enforcement and the prevention of natural disasters.

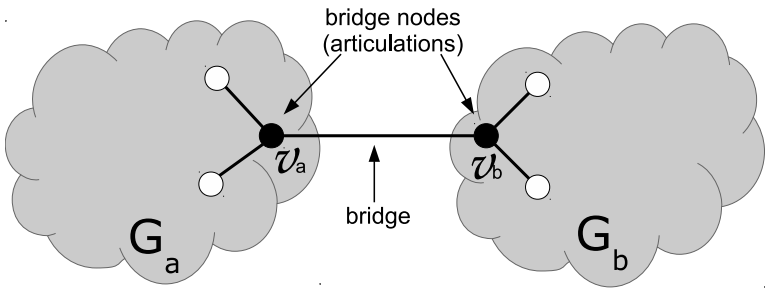



Figure 1: Graph depicting two bridge nodes and a bridge.

Since nodes cooperate in relaying packets to each other, it is paramount to maintain connectivity in such scenarios. As link and node failure are events that may occur during the course of operation, ways to prevent network partitioning and node isolation are of interest. Let $G = (V, E)$ be a planar, undirected connected graph, where $V$ is the set of nodes and $E$ is the set of edges. An edge $e \in E$ is a critical link, also referred to as a *bridge* [1], if its removal make the graph disconnected. A bridge is shared by at least an articulation node that can be defined in a similar way. A node $v \in V$ is a critical node, i.e., cut-vertex or *articulation* node, if its removal makes the graph disconnected. In what follows, an articulation node that shares a bridge is referred to a *bridge node*. Figure 1 shows a graph containing two bridge nodes.

Due to its importance in preserving network connectivity, a number of works have been devoted to locate critical edges and nodes. A naive algorithm to detect articulation and bridge nodes on a connected graph can, for each node, remove it and test if the resulting graph is still connected. Many fast algorithms, like the one proposed by Hopcroft et al. [2] and Tarjan [3], uses a depth first search to detect articulation nodes. Goyal et al. [4] proposed a centralized solution to locate articulation points in ad hoc networks. However, as centralized solutions are usually costly due to the need of gathering and maintaining topological information, localized solutions have been devised. In [5] and [6], $k$-hop neighbouring information is used to reduce computational resources to locate articulation nodes and bridges. A distributed algorithm to find articulation points was proposed by Chaudhury[7], that requires $O(|V|)$ messages and runs in $O(|V|)$ time, being optimal in communication complexity within a constant factor. In [8], Chaudhury presented a similar algorithm with same complexity to localize bridges in a wireless network. In [9], Turau presented a distributed algorithm for computing bridges, articulations, and 2-connected components of undirected graphs in $O(|E|)$ time using at most $4 \cdot |E|$ messages of length $O(\log |V|)$.

As viable solutions to locate bridges and articulation nodes have been developed, the research community focused on alternatives to extend their availability and ways to reestablish network connectivity in case of failure [7, 6, 9]. In [10], the authors employed packet aggregation techniques to reduce energy consumption of articulation nodes, thus extending their life span. The solution is based on the fact that energy consumption of articulation nodes is usually higher than other nodes, leading to premature node failure. Khelifa et al. [11] proposed the usage of dormant nodes that could be activated in case of link or node failure. When necessary, the dormant nodes would be activated to prevent network partitioning. In the same line, Goyal et al. [4] proposed the usage of limited, coordinated mobility, to recover from link and node failures. In case of network disruption, nodes would move in a coordinated way to reestablish network connectivity.

In an ad hoc network, it is not always possible to increase the number of nodes or even move nodes to the desired location to improve network connectivity. This can be the case during disaster and recovery situations where communication is vital and the time to plan, dispense or move nodes to provide network coverage may not be possible. The use of *cooperative communication*, CC for short, has been considered an attractive alternative to support the establishment of new links [12, 13]. CC is a physical layer technique that allows a single antenna device to benefit from some advantages of MIMO systems [14]. More precisely, when a source node transmits a packet using collaborative communication, a set of *helper nodes* in the vicinity that overhear the signal, simultaneously relay

independent copies of the same signal to the destination node. The destination node then combines the received signals to obtain the original packet. Thus, contrary to network layer strategies, CC technique allows nodes to improve signal quality and transmission range. Zhu et al. [15] consider the problem of selecting power efficient paths when CC-links are used. Yu et al. [12, 13] employed cooperative communication techniques to connect disjoint network components using cooperative links. As the task of selecting an optimal topology employing CC was shown to be an NP-complete problem [15], a heuristic, termed *Greedy Helper Set Selection* - GHSS, was proposed by Yu et al. [12, 13]. Neves et al. [16] exploited cooperative communications to establish power efficient links and routes to a sink node. The above schemes work under the assumption that the node's location in each component is known a priory. Neves et al. [16] assume the presence of a base station enhanced with a powerful antenna capable of reaching all the nodes of interest. That is, the base station uses the node's location to compute the best CC-links. This information is then relayed to those nodes that shall establish cooperative, directional or bidirectional links, thus improving network connectivity. Once network connectivity is ensured, the nodes use multi-hop communication to answer to the queries issued by the base station.

It should be clear that the above solutions only work in specific settings that can comply with the necessary means to gather and dispense information. Furthermore, these solutions aim to connect isolated network components before operation. This work explores a different path by proposing a proactive and localized mechanism to reestablish network connectivity in case of a bridge node failure. The proposed solution comprises of the following tasks: (*i*) identify articulation nodes and bridges; (*ii*) compute power-efficient, cooperative, backup links (i.e., CC-links); (*iii*) observe the activity of the bridge nodes; and (*iv*) prevent network partitioning by establishing CC-links in case of bridge node failure. The proposed localized algorithms to identify articulation nodes and bridges require at most $4 \cdot (|V| - 1)$ messages. Once these elements are identified, power-efficient, bi-directional CC-links, are computed using the GHSS Algorithm, proposed in [12]. This later task makes $O(\Delta(G))$ calls to the GHSS routine, per bridge node, where $\Delta(G)$ is the maximum degree of $G$. Thus, overall, the proposed solution uses $O(|V|)$ messages and makes $O(\Delta(G))$ calls to the GHSS routine. The computed CC-links are later used to reestablish communication in case of network partitioning. The proposed solutions are evaluated under the following metrics: (**M1**) computing cost to create CC-links; (**M2**) transmission power needed to establish the CC-links; and (**M3**) the percentage of network connectivity recovery. Compared to the centralized algorithm proposed in [12], the proposed solution allows to reduce the computing cost to create CC-links in $\approx 67$ times in the evaluated scenarios (**M1**), has a lower message overhead, while achieving similar results for metrics (**M2**) and (**M3**).

The remaining paper is organised as follows. Section 2 presents an overview of the closely related works while Section 3 describes the communication and network models, defines bridges and articulation nodes and formalises the main problem addressed in this work. Section 4 presents the proposed distributed solution to locate articulation points and bridges. This section also presents the mechanism to compute and establish CC-links. Simulation results are presented in Section 5 and Section 6 concludes this work.

## 2 Related Works

This section presents an overview of the closely related works. This review considered the works of two distinct groups: (*i*) those that attempt to mitigate the effects of link unavailability; and (*ii*) those interested in improving network connectivity with the aid of cooperative communication. The first group addresses the problem of recovering network connectivity and rely on proposals based on coordinated mobility, activation of dormant nodes or deployment of additional nodes. These works have gained attention once viable solutions to locate bridges and articulation nodes have been developed. Goyal et al. [4] prosed a mechanism to avoid or delay network partitioning by identifying critical links and reinforcing them. The network topology is adjusted by sending a special message to request a helper node to move to the required location that would improve network connectivity. In the same line, Khelifa et al. [11] proposed the use of dormant nodes that would be activated

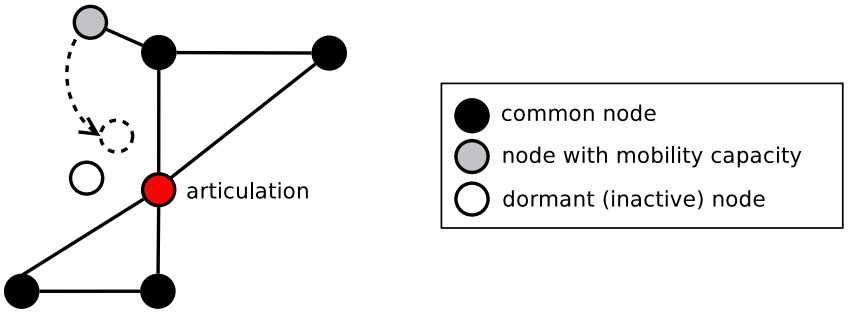



Figure 2: Solutions based on critical nodes to avoid network partitioning using coordinated mobility and dormant nodes.

in case of articulation node failure. If no such node exists, one or more nodes in the vicinity of an articulation point, having enough residual energy, would be moved to a coordinate that helps improving connectivity and prevent network partitioning. Figure 2 shows an example of the use of coordinated mobility and the activation of dormant nodes to prevent the formation of network islands. The dotted arrow-line represents the location that the mobile node is supposed to move while a dormant node waits its turn to be activated.

The use of extra, dormant, nodes that would be waiting to be activated may not be a feasible option due to the cost and the difficulty to predict the location at which such nodes would be demanded. Also, enhancing ad hoc nodes with mobility capabilities increases battery demands that could preclude the benefits of moving them. Furthermore, ways to activate the dormant nodes need to be carefully devised so that they can be awaken only when necessary as to preserve battery power as much as possible.

Works based on cooperative communication have addressed the problem of improving network connectivity, which is obtained by allowing the nodes to transpose the maximum communication radius. This task is usually carried on after deployment and before operation.

In [15, 17], the authors proposed mechanisms to explore cooperative communication to reduce energy consumption along the path to the desired destination. Another approach aim to improve network connectivity while maintaining power consumption under acceptable levels [12, 16]. Yu et al. [12] used CC as a topology control mechanism, whose purpose is to connect disjoint components through cooperative links. The authors proposed a heuristic, called $GHSS$ to select power efficient helpers nodes to minimize the overall power consumption of the nodes sharing a CC-link. Starting from an undirected, disconnected graph, the proposed algorithm, called $CoopBridges$, uses the aforementioned heuristic to create cooperative edges to connect network components. In the resulting topology, a minimum spanning tree algorithm is employed to prune costly links within network components and among them. The task of selecting power efficient helper nodes runs in $O(|V|^2)$ time. Neves et al. [16] developed a similar mechanism that interconnects components of an ad hoc network, initially with no direct connectivity, to a sink node. The solution, called $CoopSink$, uses a modified version of the heuristic proposed in [12]. $CoopSink$ uses low cost cooperative edges to interconnect the network such that the created paths would lead to the sink node. As in [12], the task of selecting power efficient helper nodes takes $O(|V|^2)$ time [16]. It should be noted that the above works use cooperative communication to increase network connectivity before operation, as a means to improve network connectivity. As mentioned earlier, these solutions work under the assumption that the cooperative links (CC-links) can be computed and established prior to the network operation.

This work proposes an alternative to maintain network connectivity by identifying and monitoring critical nodes. In case of node or link failure, cooperative links (CC-links) are used to reestablish network connectivity whenever possible. To the best of our knowledge, this is the first work to explore cooperative communication to maintaing network connectivity by identifying and monitoring critical nodes. Simulation results show that the proposed solution attains comparable results of a centralized and more costly alternatives while demanding far less network resources.

# 3 Model and Problem Definition

This section begins with the description of the cooperative communication model used throughout this paper. The network model and relevant definitions, including the problem being addressed in this work, are presented in the subsequent sections.

## 3.1 Cooperative Communication (CC) Model

Consider a wireless ad hoc network where each node $v_i$ can adjust its transmission power $P_i$ with values within the interval $[0, P_{MAX}]$. When $P_i = 0$, the node's radio is off and, when $P_i = P_{MAX}$, the node's radio operates with maximum power. In traditional cooperative communication models, a sender node $v_i$ can directly communicate with a receiver node $v_j$ only if the transmission power of $v_i$ satisfies Equation (1).

$$P_i(d_{i,j})^{-\alpha} \geq \tau \qquad (0 \leq P_i \leq P_{MAX}), \tag{1}$$

where: $\alpha$ is the path loss exponent, usually between 2 and 4, and represents the rate of signal fading with increasing distance; $d_{i,j}$ is the Euclidean distance between $v_i$ and $v_j$; and $\tau$ is the receiver sensitivity to correctly receive a packet, i.e., the threshold of the received power so that node $v_j$ can correctly decode the signal and obtain the original message.

Cooperative communication (CC) takes advantage of the physical layer design to combine partial signals to obtain complete information [14]. This way, a complete communication between nodes $v_i$ and $v_j$ can be achieved with CC if $v_i$ transmits its signal jointly with a set of *helper* nodes $H_{i,j}$ and the sum of its transmission power satisfies Equation (2). In CC, a helper node is a node that cooperatively retransmit the signal along with the transmitting node.

$$\sum_{v_k \in v_i \cup H_{i,j}} P_k(d_{k,j})^{-\alpha} \geq \tau \qquad (0 \leq P_k \leq P_{MAX}). \tag{2}$$

## 3.2 Network Model

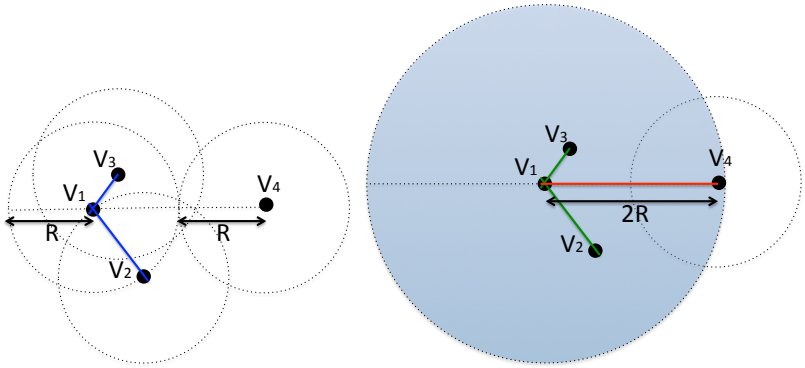
Consider a wireless ad hoc network with $n$ nodes that are capable to receive and combine partial data received in agreement with the CC model. The network topology is modelled as a planar graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is a set of wireless nodes and $E$ is the set of communication links. An edge $v_i v_j \in E$ symbolises that node $v_i$ can transmit data to $v_j$ directly and/or using CC. $N(v_i)$ is the direct neighbour set of $v_i$ within its maximum transmission range $R_{MAX}$. For every $v_k \in N(v_i)$, there is $P_i \leq P_{MAX}$ such that $P_i(d_{i,k})^{-\alpha} \geq \tau$, following Equation (1). In other words, $v_i$ can directly communicate with its neighbours in $N(v_i)$. Each node $v_i \in V$ has a unique radio, runs on battery power and has 1-hop information. Given the previous information, we define several important concepts, similar to those in [15].

**Definition 1** *(Direct link): A direct link $\overline{v_i v_j}$ is an edge in E representing that node $v_i$ can transmit data to node $v_j$ directly, that is, $P_i$ is such that $v_i$ can reach $v_j$ when $P_i \leq P_{MAX}$. A solid horizontal line over the nodes denote a direct link.*

**Definition 2** *(Helper node set): $H_{i,j}$ symbolises the set of helper nodes of $v_i$ in a cooperative communication with $v_j$. It is assumed that all helper nodes need to be direct neighbours of $v_i$, that is, $H_{i,j} \subseteq N(v_i)$, where $N(v_i)$ is the set of all direct neighbours of $v_i$.*

**Definition 3** *(CC-link): A CC-link $\widetilde{v_i v_j}$ is an edge of E that represents that node $v_i$ can transmit data to $v_j$ cooperatively by using a set of helper nodes $H_{i,j}$. A wavy horizontal line is used to denote a CC-link.*

**Definition 4** *(Helper link): A helper link is an edge from $v_i$ to one of its helper nodes in $H_{i,j}$. For example, in Figure 3a, $v_1$ can not create a direct link to $v_4$ because $v_4$ is not in its transmission range. However, as it can be seen in in Figure 3b, node $v_1$ can use nodes $v_2$ and $v_3$ as helper nodes to create a CC-link $\widetilde{v_1 v_4}$, where $H_{1,4} = \{v_2, v_3\}$, in this way, the edges $\overline{v_1 v_2}$ and $\overline{v_1 v_3}$ are considered helper edges.*




(a) (b)

Figure 3: (*a*) Graph $G = (V, E)$, where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{v_1v_3, v_1v_2, v_2v_1, v_3v_1\}$. (*b*) Creation of CC-link $\widetilde{v_1v_4}$ using nodes $H_{1,4} = \{v_2, v_3\}$ as helper nodes.

**Definition 5** *(Network topology): The union of all direct links and CC-links are* $\overline{E}$ *and* $\widetilde{E}$*, respectively. Similarly, the direct communication graph and the CC communication graph are denoted as* $\overline{G} = (V, \overline{E})$ *and* $\widetilde{G} = (V, \widetilde{E})$*, respectively. Note that* $E = \overline{E} \bigcup \widetilde{E}$*. Following the notation, if* $v_iv_j \in E$*, then:* $v_iv_j = \overline{v_iv_j}$*, if* $v_iv_j$ *is a direct link, and* $v_iv_j = \widetilde{v_iv_j}$*, if* $v_iv_j$ *is a CC-link.*

**Definition 6** *(Direct link weight): The weight of a direct link* $\overline{v_iv_j}$ *is defined as:*

$$w(\overline{v_iv_j}) = \tau d_{i,j}^{\alpha}.$$

**Definition 7** *(CC-link weight): The weight of a CC-link* $\widetilde{v_iv_j}$ *is defined as:*

$$w(\widetilde{v_iv_j}) = w_d(H_{i,j}) + (|H_{i,j}| + 1)w_{CC}(H_{i,j}),$$

*where:*

- $|H_{i,j}|$*: is the number of elements in* $H_{i,j}$*;*
- $w_d(H_{i,j}) = \left(\frac{\tau}{\max_{v_k \in H_{i,j}}(d_{i,k})^{-\alpha}}\right)$*: is the minimum transmission power of node* $v_i$ *to communicate with the most distant node in* $H_{i,j}$*;*
- $w_{CC}(H_{i,j}) = \left(\frac{\tau}{\sum_{v_k \in v_i \bigcup H_{i,j}}(d_{k,j})^{-\alpha}}\right)$*: is the minimum transmission power of node* $v_i$ *to communicate with* $v_j$*, jointly transmitting with its helpers in* $H_{i,j}$

Note that, according to Equations (1) and (2), a CC-link exists when the following relation holds:

$$\max\left(w_d(H_{i,j}), w_{CC}(H_{i,j})\right) \leq P_{MAX}.$$

In a CC from $v_i$ to $v_j$, node $v_i$ must: (*i*) send its data to its helper nodes in $H_{i,j}$; and then (*ii*) node $v_i$ and its helpers must simultaneously send the same data to $v_j$. This way, the weight of the CC-link consists in the sum of the communication costs of these two moments: $w_d(H_{i,j})$ is the cost of the first moment and $w_{CC}(H_{i,j})$ is the individual node cost to transmit a data over a CC-link. In this work, the CC model is simplified assuming that the transmission power of $v_i$ and its helper nodes are the same. Furthermore, only the power consumption in each sender node is considered. Figure 3 illustrates the concepts presented in this subsection.

According to Sedgewick and Wayne [18], an *articulation* point (a.k.a articulation node or cut-vertex) and a *bridge* can be defined as:

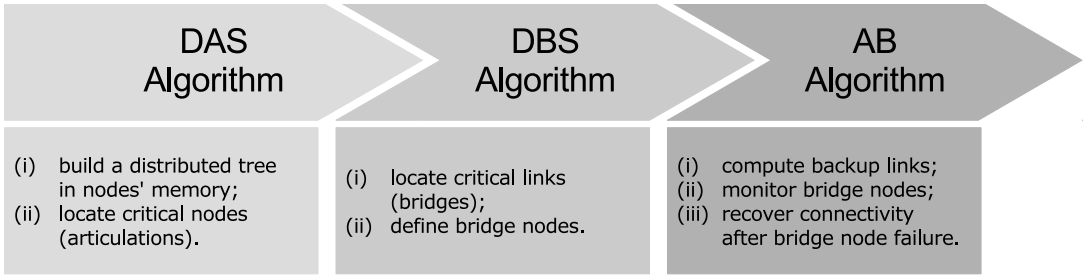



Figure 4: Flowchart for proposed solution and correponding algorithms.

**Definition 8** *(Articulation node) An articulation point is a vertex $v_a \in V(G)$ of a connected graph $G$ that if it (and its adjacent edges) were removed, the graph $G$ becomes disconnected.*

**Definition 9** *(Bridge) A bridge is an edge $v_a v_b \in E(G)$ of a connected graph $G$ that if it is removed, the graph $G$ becomes disconnected.*

**Definition 10** *(Bridge node) Bridge nodes are articulation nodes which are connected by a bridge edge. Let $v_a \in V(G)$ and $v_b \in V(G)$ be articulation nodes in $G$ such that $v_a v_b \in E(G)$ is a bridge. Then, $v_a$ and $v_b$ are bridge nodes. Note that bridge nodes are also articulation nodes but not otherwise. Figure 1 shows a connected graph $G$ depicting two bridges nodes sharing a bridge.*

### 3.3 Problem Definition

Consider the articulation nodes $v_a$ and $v_b$ such that $v_a \in V_a$ and $v_b \in V_b$, where $G_a = (V_a, E_a)$, $G_b = (V_b, E_b)$, $V_a \cap V_b = \emptyset$, $G_a \subseteq G$ and $G_b \subseteq G$. Following the Definition 10, should a bridge node fails, say $v_a \in G_a$, the nodes in the connected component $G_a$ would be unable to communicate with the nodes in the connected component $G_b$. This situation may arise in a number of scenarios in which a fast link re-establishment may be necessary to ensure network connectivity. This work focuses on enhancing network connectivity by monitoring the activity of bridge nodes. In case of a bridge node failure, CC is employed as an attempt to avoid network partitioning. As it will be shown in the subsequent sections, the proposed solution works by identifying bridge nodes and identifying potential cooperative links that are used in case of need.

## 4 Proposal Description

This section presents a distributed solution to recover network connectivity due to bridge node failure. To achieve this goal, the first task is to identify articulation points and bridges so that the bridge nodes are correctly determined. Once the bridge nodes are known, CC is employed to preventively identify backup links that shall be used in the case of bridge node failure. As can be observed in Figure 4, the proposed solution comprises three algorithms, whose details are given in the subsequent sections.

### 4.1 Identifying Articulation Nodes

The mechanism proposed to identify articulation nodes, termed *Distributed Articulation Search* (DAS), is based on the work presented by [7]. The DAS uses a distributed version of the *Depth First Search* (DFS) algorithm to detect articulation nodes. The DAS algorithm starts from a given arbitrary root node and, on completing its execution, all nodes know whether they are an articulation node or not. The details of the DAS algorithm are presented in Algorithm 1. A special message, termed $SEARCH$ message, is used to carry a three-tuple $(a, v, nt)$, where $a$ is a list of ancestor nodes, $v$ contains a list of visited nodes and $nt$ is a list of non-tree edges. The first $SEARCH$ message is

**Algorithm 1** Algorithm_DAS($root$)

```
1: # Initialization for each node v_i
2: parent_{v_i} ← 0;
3: children_{v_i} ← ancestor_{v_i} ← nontree_{v_i} ← ∅;
4: visited_{v_i} ← {i};
5: tovisit_{v_i} ← N(v_i);
6: if i = root then
7:    destination_{v_i} ← first node in tovisit_{v_i}
8:    children_{v_i} ← children_{v_i} ∪ destination_{v_i}
9:    tovisit_{v_i} ← tovisit_{v_i} − destination_{v_i}
10:   visited_{v_i} ← visited_{v_i} ∪ destination_{v_j};
11:   send message SEARCH({i}, {i}, ∅) to destination_{v_i}
12: end if
13:
14: # Node v_j receives a message SEARCH(a, v, nt) from node v_i
15: visited_{v_j} ← visited_{v_j} ∪ v;
16: tovisit_{v_j} ← tovisit_{v_j} − visited_{v_j};
17: if parent_{v_j} = 0 and j ≠ root then
18:    parent_{v_j} ← i;
19:    ancestor_{v_j} ← a;
20: end if
21: if i ∈ children_{v_j} then
22:    if j = root then
23:       if |children_{v_j}| > 1 then
24:          articulation_{v_j} ← TRUE;
25:       end if
26:    else
27:       if ∄ v_x v_y ∈ nt | x ∈ ancestor_{v_j} or y ∈ ancestor_{v_j} then
28:          articulation_{v_j} ← TRUE;
29:       end if
30:       nontree_{v_j} ← nontree_{v_j} ∪ nt;
31:    end if
32: end if
33: if tovisit_{v_j} ≠ ∅ then
34:    destination_{v_j} ← first node in tovisit_{v_j};
35:    children_{v_j} ← children_{v_j} ∪ destination_{v_j};
36:    send message SEARCH(ancestor_{v_j} ∪ {j}, visited_{v_j}, ∅) to destination_{v_j};
37: else
38:    if j ≠ root then
39:       for all node k ∈ N(v_j) − children_{v_j} − parent_{v_j} do
40:          nontree_{v_j} ← nontree_{v_j} ∪ {v_j v_k};
41:       end for
42:       send SEARCH(∅, visited_{v_j}, nontree_{v_j}) to node parent_{v_j};
43:    end if
44: end if
```

sent from the root node during the DAS initialization phase (lines 6-12). A node executing the DAS may send a $SEARCH$ message in two directions:

($i$) **Downwards (children nodes)**: when the message is sent to non visited nodes (downwards message), it creates a tree that is stored in a distributed way (line 36);

($ii$) **Upwards (parent nodes)**: when the message is sent to a visited node (upwards message), it propagates the list of non-tree edges found by the node and its descendants (line 42).

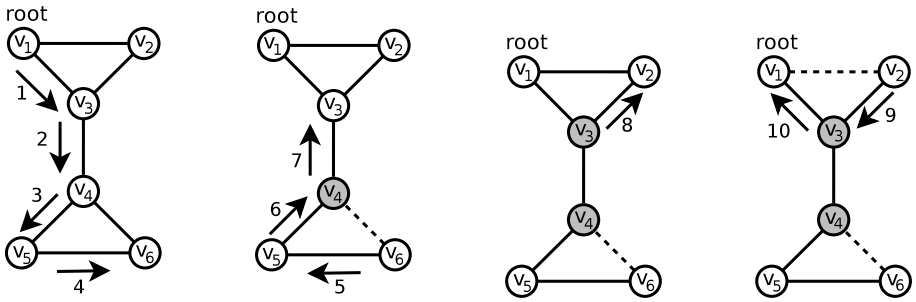



Figure 5: Example of DAS algorithm: (a) a depth first search from root node $v_1$ to leaf node $v_6$; (b) a non-tree edge ($v_6v_4$) is found and the information is propagated back to the parent node until a non visited node is found; (c) the leaf node $v_2$ is reached; (d) another non-tree edge is found and, after the propagation of this information, the algorithm finishes.

Table 1: Description of the events based on Figure 5(a)-(d) when executing the DAS algorithm.

| Figure | Message | Description |
|---|---|---|
| 5a | 1 | Node $v_1$ sends a downward message 1 to node $v_3$, chosen arbitrarily between nodes $v_2$ and $v_3$. |
| | 2 | Node $v_3$ chooses node $v_4$ to send a downward message. |
| | 3 | Node $v_4$ chooses node $v_5$ to send the downward message. |
| | 4 | Node $v_5$ has only one option to send the downward message, node $v_6$. |
| 5b | 5 | Node $v_6$ has no unvisited neighbour, then it adds the edge $(4, 6)$ to its list of non-tree edges and sends an upward message to its parent. |
| | 6 | Node $v_5$ checks that it is not an articulation, since the received non-tree list contains an ancestor, and sends an upward message to its parent. |
| | 7 | Node $v_4$ checks that it is an articulation node, since the non-tree list received has no ancestors in it, and sends an upward message to his parent. |
| 5c | 8 | Node $v_3$ checks that it is an articulation node. Since node $v_3$ still has an unvisited neighbour, $v_3$ sends a downward message to $v_2$. |
| 5d | 9 | Node $v_2$ has no unvisited neighbour, then it adds its additional link $(1, 2)$ to its non-tree list and sends an upward message to its parent $v_3$. |
| | 10 | As node $v_3$ has no unvisited neighbours, it sends an upward message to its parent, node $v_1$. Since node $v_1$ has no unvisited neighbour and it is the root, the algorithm terminates. |

When a node sends a message to an unvisited node, it arbitrarily chooses a neighbour and adds the destination to its list of children nodes (lines 34-36). The list of unvisited nodes initially contains all neighbouring nodes (line 5). This list is updated at each $SEARCH$ message received (lines 16). The following tasks are performed by the destination node upon receiving a $SEARCH$ message:

1. Define the source node as parent node (line 18);
2. Update the list of visited nodes;
3. Update the list of ancestors (line 19); and
4. Send the message $SEARCH$ to a selected unvisited neighbour.

When all of a node's children have been visited, the node returns an upward $SEARCH(\emptyset, v, nt)$ message to its parent node. This message indicates that a sub-tree has been completely processed

and provides the list non-tree edges (the *nt* list) found in that segment. These upward messages are initially sent by a leaf node. The non-tree edges are calculated removing the parent and children nodes from the list of neighbours (line 39-41). A non-leaf node propagates the list of non-tree edges after receiving this information from all its children nodes. A node $v_j$ learns that it is an articulation when it receives an upward $SEARCH$ message (lines 27-29) and verifies that none of its descendants have non-tree edges connecting to any of its ancestral. This operation can be performed by exploring the information in the *nt* list and the $ancestor_{v_j}$. Lines 23-24 deals with a special case, when the root node is an articulation. In this case, the algorithm checks whether the root has more than one child and, if true, the root is an articulation node.

The algorithm terminates when the root node receives upward $SEARCH$ messages from all of its children nodes, indicating that all nodes have been visited. Since all nodes receive exactly one downward message and sends exactly one upward message, articulation nodes can be computed using $2 \cdot |V|$ messages. Compared to the algorithm proposed in [7], the DAS algorithm uses fewer messages to locate articulation points.

Figure 5 show an example of the DAS algorithm on a graph containing six nodes. The sequence of events is detailed in Table 1. The DAS algorithm starts from $v_1$, the root node. The DAS algorithm runs until all nodes have been visited. Figure 5b-d, shows the identification of non-tree edge connecting nodes $(v_4, v_6)$ and $(v_1, v_2)$ and the identification of articulation nodes $v_4$ and $v_3$.

## 4.2 Identifying Bridges

The *Distributed Bridge Search* (DBS) algorithm is employed to locate bridge edges based on the information gathered during the execution of the DAS algorithm. The DBS algorithm correctly identify bridge edges and bridge nodes. To this end, the DBS uses a special message, called $BRIDGE$, that are sent by articulation nodes and leaf nodes to verify whether or not a common edge is a bridge or not. Recall from the DAS algorithm that only leaf nodes and articulation nodes may share

**Algorithm 2** Algorithm_DBS(*root*)

Initialization of node $v_i$
1: $bridge_{v_i} \leftarrow false$;
2: $bridgePair_{v_i} \leftarrow \emptyset$;
3: **if** $children_{v_i} = \emptyset$ **and** $nontree_{v_i} = \emptyset$ **then**
4: send message $BRIDGE(i)$ to node $parent_{v_i}$;
5: **end if**
6: **if** $articulation_{v_i}$ **and** $i \neq root$ **then**
7: **if** $\nexists\, v_x v_y \in nt \,|\, x \in ancestor_{v_j}$ **or** $y \in ancestor_{v_j}$ **then**
8: send message $BRIDGE(i)$ to node $parent_{v_i}$;
9: **end if**
10: **end if**
11: Terminate

Node $v_j$ receive a $BRIDGE(i)$ message from node $v_i$
12: **if** $i = parent_{v_j}$ **then**
13: $bridge_{v_j} \leftarrow true$;
14: $bridgePair_{v_j} \leftarrow bridgePair_{v_j} \cup \{i\}$;
15: **else**
16: **if** ($i \in children_{v_j}$ **and** $articulation_{v_i}$) **or** $i = root$ **then**
17: $bridge_{v_j} \leftarrow true$;
18: $bridgePair_{v_j} \leftarrow bridgePair_{v_j} \cup \{i\}$;
19: send message $BRIDGE(j)$ to node $v_i$;
20: **end if**
21: **end if**
22: Terminate

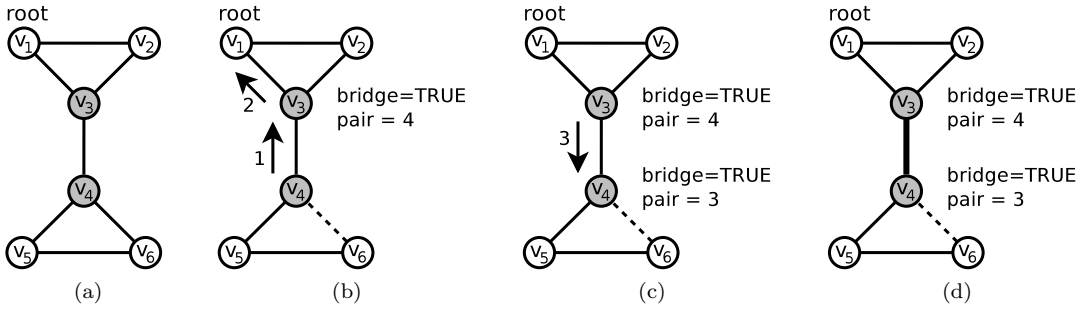



Figure 6: Example of DBS algorithm: (a) DAS input graph with the articulation nodes and leaf nodes being identified; (b) articulation nodes send *BRIDGE* messages to their parents; (c) parent node $v_3$ confirms that the edge $v_3v_4$ is a bridge; (d) output information of the DBS algorithm.

Table 2: Example of DBS based on Figure 6(a)-(d).

| Figure | Message | Description |
|---|---|---|
| 6a | | DBS input graph (output from DAS): a graph with all articulation nodes identified. Note that each node stores in memory the information gathered during the course of the DAS algorithm. The edged with arrows represents the parent node of each node while dotted edges are the non-tree edges. |
| 6b | 1 | Articulation node $v_4$ verifies that the link $v_3v_4$ is likely to be a bridge and send the message to its parent, node $v_3$. Node $v_3$ confirms that the link $v_3v_4$ is a bridge and store that information on its local memory. |
| | 2 | Articulation node $v_3$ verifies that the link $v_1v_3$ is likely to a bridge and send the message to its parent, node $v_1$. |
| 6c | 3 | Node $v_3$ sends a message back to node $v_4$ confirming that the edge $v_3v_4$ is a bridge. Node $v_4$ store the information in its local memory. |
| 6d | | Nodes $v_3$ and $v_4$ learn that they are bridge nodes (the bridge edge is drawn in a tick line). |

a bridge. All other nodes have non-tree edges connecting to their ancestors. Algorithm 2 presents the DBS details.

In the initialization step (line 3), the leaf nodes without non-tree edges send a BRIDGE message to their parents (lines 3-5). Then, each articulation node without non-tree edges to an ancestor, sends a *BRIDGE* message to its parent (lines 6-10). When the destination node receives the message, it verifies if it is also an articulation node or not. If it is not, the node finishes the execution of the algorithm. Otherwise, the node marks the edge as a bridge (line 17); adds the source node as a bridge node (line 18); and sends a message to the source indicating that it is an articulation and confirms that link is a bridge (line 19). When a node receives a confirmation message (a message from its parent), it should mark the edge as a bridge (line 12 and 14). Figure 6 shows an example for the algorithm DBS while Table 2 explains the sequence of events.

## 4.3 Bridge Nodes Monitoring and Connectivity Recovery

This section presentes the core idea behind the proposed connectivity recovery scheme. As discussed in the previous sections, the algorithms DAS and DBS ensure the identification of bridge nodes. As bridge nodes are critical nodes to ensure network connectivity, this section presents a collaborative strategy that allows, whenever possible, the nodes in the vicinity of a bridge node to take over its duties.

**Algorithm 3** Articulation-Bridges($updateInterval$)

1: **if** ($bridge_{v_i}$) **then**
2: Let $v_a$ be the articulation node running the algorithm;
3: Let $v_b$ be the articulation node in an adjacent cluster;
4: **for** every **updateInterval** seconds **do**
5: Let $N(v_a)$ be the set of direct 1-hop neighbours from $v_a$;
6: **for** $v_i \in N(v_a) - v_b$ **do**
7: Call $GHSS$ to estimate the cost of the CC-link $\widetilde{v_i v_b}$;
8: Call $GHSS$ to estimate the cost of the CC-link $\widetilde{v_b v_i}$;
9: **end for**
10: Let $\widetilde{v_{a'} v_b}$ be the best, bi-directional, CC-link from any node in $N(v_a)$ to $v_b$;
11: send message $RECOVER(v_{a'}, v_b)$ to nodes $v_{a'}$ and $v_b$;
12: **end for**
13: **end if**

14: # Actions taken by nodes receiving a $RECOVER$ message
15: Let $v_i, v_j$ the nodes receiving a $RECOVER(v_i, v_j)$ message;
16: Let $v_a$ be the articulation node that sent the $RECOVER$ message;
17: **for** every **updateInterval** seconds **do**
18: **if** $v_a$ is unavailable **then**
19: Create a CC-link $\widetilde{v_i v_j}$;
20: **end if**
21: **end for**

The proposed algorithm to monitor bridge nodes and to recover connectivity, termed *Articulation-Bridges* (AB), is shown in Algorithm 3. The following conditions are assumed:

($i$) One-hop routing information is provided by the underline routing protocol;

($ii$) Two-hop routing information can be obtained;

($ii$) Distance estimation can be obtained.

Routing protocols, such as AODV, resort to a special purpose ("hello") messages to gather one-hop neighbouring information [19]. Hence, condition ($i$) can be easily obtained when such protocols are employed. Likewise, by performing routing table exchange with its direct neighbours, two-hop routing information can be obtained. Thus, conditions ($i$) and ($ii$) can be met. The transmitting power can be explicitly informed along with the "hello" messages or along with the routing table information exchange. With the above information at hand, and given the characteristics of the communication model assumed in this work, distance estimation can be obtained. Indeed, distance estimation based on RSSI have been reported in the literature to provide reasonable accuracy [20]. The work by Mikko Kohvakka et al. [21] showed that the path loss can be determined from frames transmitted at different power levels. With the path loss and transmitted power information at hand, the distance estimation can be obtained, even if the RSSI is not known [21]. Thus, by employing the above techniques, condition ($iii$) can be satisfied. Furthermore, it should be clear that one can compensate the accuracy error in distance estimation by a slight increase in the transmitting power.

With the above information at hand, a CC-link can be computed, similarly to the one shown in Figure 3. CC-links with least transmission power are computed with the aid of the *Greedy Helper Set Selection* (GHSS) Algorithm, proposed in [12]. This algorithm takes as input the source and destination nodes, denoted as $v_a$ and $v_b$, respectively; the set of neighbouring nodes of the source node, denoted as $N(v_a)$; and the distances $d_{v_i,v_b}$, where $v_i \in N(v_a) \cup v_a$. The output of the algorithm is an estimative of the CC-link $\widetilde{v_a v_b}$ cost. As shown in lines 4-12 of the Algorithm 3, bridge nodes periodically compute the best node to take its place in case of failure. The bridge nodes use the GHSS Algorithm to calculate the most efficient bi-directional CC-link from $N(v_a)$ to $v_b$ (conversely

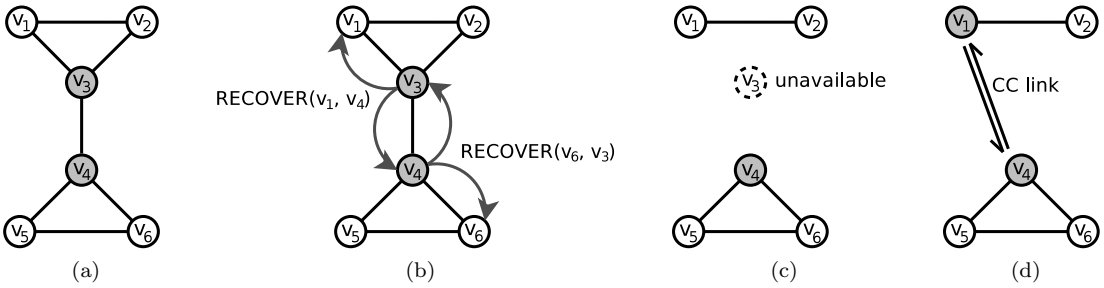



Figure 7: Establishment of CC-links in case of a bridge node failure according to the Articulation-Bridges Algorithm.

Table 3: Example of the Articulation-Bridges (Algorithm 3) based on Figure 7(a)-(d).

| Figure | Event | Description |
|---|---|---|
| 7a | Input from DBS | Nodes $v_3$ and $v_4$ are bridge nodes. $N(v_3) = \{v_1, v_2, v_4\}$, $N(v_4) = \{v_3, v_5, v_6\}$. Nodes sharing a bridge and their neighbours start Algorithm 3. |
| 7b | Update | Node $v_3$ calculates (using the GHSS routine) the CC-link cost from all its neighbours to node $v_4$ and chooses the most efficient CC-link, $v_1$ in this example. In a similar way, node $v_4$ select node $v_6$ as candidate to create the CC-link. |
| | Send msg | Node $v_3$ sends the message $RECOVER(v_1, v_4)$ addressed to $v_1$ and $v_4$ while node $v_4$ sends the message $RECOVER(v_6, v_3)$ to $v_6$ and $v_3$. |
| | Receive msg | Node $v_1$ and $v_6$, the direct neighbours of the bridge nodes, receive the message and start monitoring the state of the adjacent bridge nodes. |
| 7c | Node failure | Node $v_3$ fails. |
| | Route discovery | The nodes in $N(v_3)$ notice node that node $v_3$ failed. |
| 7d | Create CC-link | Nodes $v_1$ and $v_4$ establish a CC-link to recover network connectivity. |

from $N(v_b)$ to $v_a$). Then, the articulation node informs the selected backup node to assume its role in case of need by issuing an explicit message, $RECOVER$ message (lines 10-11). This message carries the information about the nodes that should create the cooperative link. Nodes receiving the $RECOVER$ message execute the code described in lines 15-21. In these steps, the nodes receiving the $RECOVER$ message monitor the availability of the bridge node that sent the message. If the bridge node becomes unavailable, the cooperative link is established. The created CC-link ensures that a new bridge is formed and network connectivity is ensured. As can be observed in Figure 7, the established CC-link gives rise to a new "bridge node" and the same process can be employed to maintain network connectivity. Table 3 details the actions executed during the course of the Algorithm 3 with the help of Figure 7.

The computational cost of the Articulation-Bridges Algorithm can be measured based on the number of calls to the GHSS routine. The routine is called twice, lines 7 and 8, for $|N(v_a)| \leq \Delta(G)$ times, where $\Delta(G)$ is the maximum degree of the graph $G$. Hence, the routine GHSS is called at most $O(\Delta(G))$ times per articulation node. To numerically determine the theoretical computational cost of the Articulation-Bridge Algorithm, consider a graph $G(V, E)$ with $|V|$ nodes. The "magic number", first presented in [22] and discussed in [23], indicates that a graph with degree of $\approx 6$, and consequently $\Delta(G) \approx 6$, tends to be connected. The *Handshaking Lemma* [24] determines that, in any graph, the sum of the degree of all vertices is equal to twice the number of edges. Given a planar, connected graph with node degree of 6, that is $\Delta(G) = 6$, the total number of edges is $|E| = \frac{|V| \cdot 6}{2}$. Based on these results, the number of calls to the GHSS heuristic to identify suitable nodes to

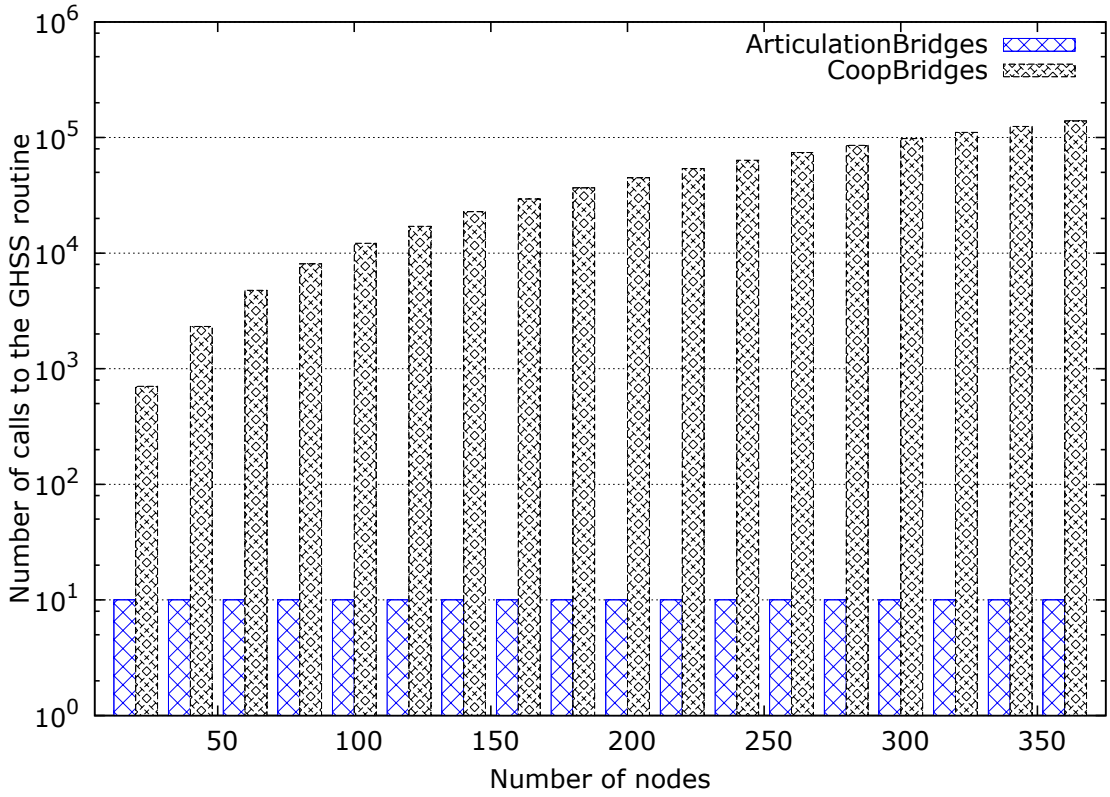



Figure 8: Number of calls to the GHSS heuristic.

establish a CC-link can be estimated. Figure 8 shows the theoretical amount of calls to the GHSS heuristic for graphs with node degree of 6 and $|V| = 20, 30, ..., 300$ nodes. The Articulation-Bridges Algorithm makes, in this case, a total of 20 to 36 calls to the heuristic on the event of a bridge node failure, independently of the number of nodes in the graph. When compared to CoopBridges, which is directly related to this work, it turns out that the proposed solution has a much lower computational cost to determine the best CC-link to recover network connectivity.

# 5 Simulation Results

To evaluate the proposed technique and to allow replication of the results, a simulator in Matlab environment [25] was developed. The simulation process consists on, from an initial topology, applying in sequence the Algorithms 1, 2 and 3, and analyse how the distributed system would handle the unavailability of bridge nodes and how effective the CC-links can be in recovering network connectivity. The evaluation scenarios are based on the following parameters (similarly to those in [12, 16, 15]): a set of nodes $n = 20, 30, ..., 60$ are randomly positioned in a $300 \times 300$m area.

Note that Eq. (1) can be easily adapted to a more suitable propagation model. Thus, for practical purposes, the radio range for the simulations is computed based on the Free Space Path Loss Model [26]. The maximum transmitting power ($P_{MAX}$) is set to 6dBm and the receiver threshold ($\tau$) is set to $-71$dBm, allowing to a maximum transmission range ($R_{MAX}$) of $\approx$ 70m on the 2.4GHz frequency band.

After the execution of the Algorithm 1 and 2, the resulting graphs with at least two network components (or clusters) united by a pair of bridge nodes are considered. This step ensures that the resulting graphs are connected and a number of bridge nodes are present (at least a pair). Next, bridge node's failure are introduced so as to verify the ability of the proposed scheme to reestablish network connectivity. It is important to note that a number of bridge nodes may fail at this step,

rendering a number of disconnected network components. We note that the proposed scheme is capable of handling multiple, distributed, bridge node failures. For this latter task, the Algorithm 3 is employed as an attempt to regain network connectivity. The results presented next are drawn from 200 simulations for each configuration set. The following metrics are used to assess the goodness of the proposed solution:

**(M1) Average cost of the CC-links:** This metric evaluates the amount of transmission power necessary to maintain network connectivity by using CC-links;

**(M2) Number of times that $GHSS$ routine is executed**: This metric evaluates the overhead introduced while creating CC-links;

**(M3) The percentage of connectivity recovery**: This metric evaluates the percentage of graphs that recovered connectivity on the event of a bridge node failure.

Given an initial network topology, as described above, bridge node failures are introduced. In this case, two different approaches are considered to recover network connectivity:

1. **Articulation-Bridges**: Using two-hop information, bridge nodes update its neighbours to assume its role in case of failure. In this case, a bi-direction CC-link is created as to recover network connectivity;

2. **CoopBridges [12]**: As mentioned in Section 2, CoopBridges uses global information to compute the best bi-direction CC-link between two different clusters to increase connectivity. This task is carried out prior to the network operation. Although CoopBridges focus is different from ours, the CC-link costs and network connectivity can be used for comparison purposes. This proposal uses a *minimum spanning tree* (MST) based algorithm to reduce the number of edges of the resulting graph to decrease power consumption in the resulting network topology.

The simulation results for metric **M1** are shown in Figure 9. The node density (number of nodes per area) is shown in the $x$-axis while the $y$-axis shows the average transmission power of the CC-links. In the figure, the bars for Articulation-Bridges and CoopBridges represent the average transmission power for the source and helpers nodes in each direction of the cooperative link. The CC model described in Section 3 shows that node density increases the chance of obtaining closer helper nodes. Thus, as expected, the transmission power necessary to reconnect network components decreases with node density in both solutions. With closer helper nodes, the values for $w_d(H_{i,j})$ decrease and the CC-link weigh $w(\widetilde{v_i v_j})$ tend to decrease as well, as can be verified in the figure. This fact holds for Articulation-Bridges as well, allowing both solutions to decrease the CC-link cost as the network density increases. Figure 9 shows that the transmission power necessary for CoopBridges and Articulation-Bridges to reconnect network components differ in at most 1dB in the simulated scenarios. Recall that the CoopBridges performs an exhaustive search to identify the best CC-link among all the nodes. Hence, it is expected that CoopBridges obtains better results in terms of CC-link transmission power, particularly at higher node density. Also, the proposed Articulation-Bridges relies on local information (two-hops) while CoopBridge needs global topological information. The cost to collect and maintain global topology information has not been considered in the above results as the focus was to assess the goodness of the CC-links computed based on local information.

Suppose that the nodes could increase the transmission power beyond the $P_{MAX}$ up to the limit necessary to reestablish network connectivity without the aid of a CC-link. Although this is an unlikely situation, as the nodes cannot rise the transmission power indefinitely due to a number of reasons, including regulatory issues, this provides a lower bound on the minimum amount of power necessary to reestablish communication. The overlapping bar `Tx(CB)` corresponds to the minimum transmission power needed to reestablish connectivity without resorting to CC-links in the resulting topology of the CoopBridge. Similarly, the `Tx(AB)` bar corresponds to the amount of power necessary to reestablish connectivity between nodes $\widetilde{v_i v_j}$ in the Articulation-Bridges. When the direct links `Tx(CB)` and `Tx(AB)` are considered, this gap gets narrower. Indeed, when a direct

links are considered, the nodes selected in both solutions present comparable results in terms of the required transmitting power to reestablish network connectivity.

The computational cost in terms of calls to the GHSS heuristic, metric **M2**, is shown in Table 4. Note that the parameter *updateInterval* has not been considered in the simulation results as to provide a fair comparison of the solutions. The column "density" corresponds to the number of nodes in the input graph. Columns "Articulation-Bridges" and "CoopBridges" correspond to the number of calls to the GHSS heuristic made by each algorithm.

Note that, in the simulations, the degree of each node is random. Thus, as the node density gets higher, nodes tend to have a larger node degree and, consequently, increasing the computational cost to reconnect the network. However, even by increasing the degree of the graph, the proposed solution presents scalable computational costs. As can be seen in the Table, the proposed solution obtained, for the evaluated cases, a reduction of up to 67 times the number of calls to the GHSS routine for density 6.67. For lower network densities, the reduction ranges from 11.7 (density 2.22) to 63.6 (density 6.67). These results corroborate to the theoretical results presented in Figure 8.

Figure 10 shows the percentage of graphs that successfully recovered connectivity after network disruption (metric **M3**). As can be was observed, both solution are comparable in terms of network connection recovery success. For node density up to 2.22, CoopBridges achieves a success rate of 99.25% while Articulation-Bridges attained a success rate of 97.75%, that is, only 1.5 percentage points below the centralized solution. Note that, at lower network densities, nodes are likely to be farther apart from each other, thus making it harder to reestablish network connectivity as there may not be enough neighbouring nodes to establish CC-links. Recall that cooperative communication can be only created when Eq. (2) is satisfied. Hence, in unfavourable circumstances, network connectivity recovery may not be possible. For node density with values between 2.78 and 3.33, both algorithms were able to recover network connectivity in approximately 99% of the cases. As the node density

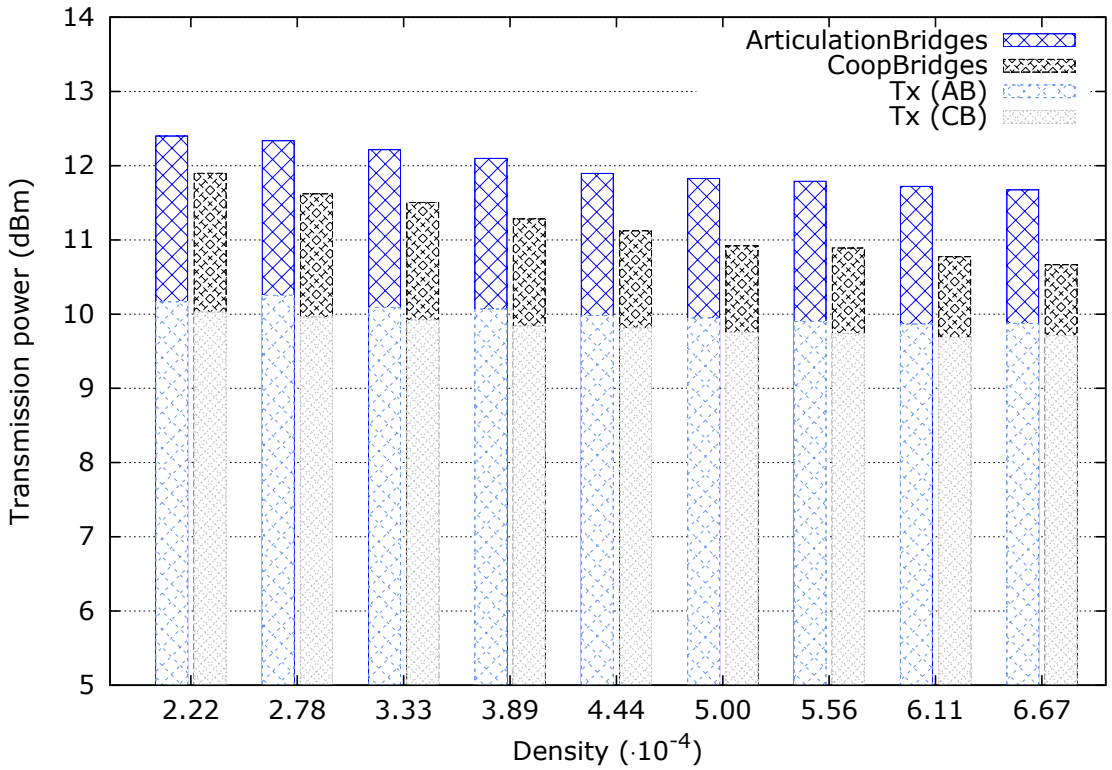



Figure 9: Average transmission power of the CC-links for both centralized and distributed solutions under different network density (metric **(M1)**).

Table 4: Number of calls to the GHSS routine (metric **(M2)**).

| Density ($\times 10^{-4}$) | Articulation-Bridges | CoopBridges |
|---|---|---|
| 2.22 | 14.78 | 174.04 |
| 2.78 | 16.38 | 276.98 |
| 3.33 | 17.18 | 408.20 |
| 3.89 | 18.86 | 562.78 |
| 4.44 | 21.98 | 742.00 |
| 5.00 | 21.36 | 950.72 |
| 5.56 | 24.96 | 1177.96 |
| 6.11 | 22.48 | 1431.12 |
| 6.67 | 25.24 | 1710.72 |

increases, the change of building suitable CC-links to reestablish network connectivity improves as well. On graphs with node density above 4.44, both algorithms have been able to recover connectivity in all evaluated cases. The results, based on the selected metrics, show that the proposed solution not only scales well but also presents comparable results in terms of network connectivity recovery.

# 6 Conclusion

Ad hoc networks have been envisioned as a viable alternative to support urgent and temporary tasks. However, ensuring network connectivity in such scenarios has been a challenge. A feasible

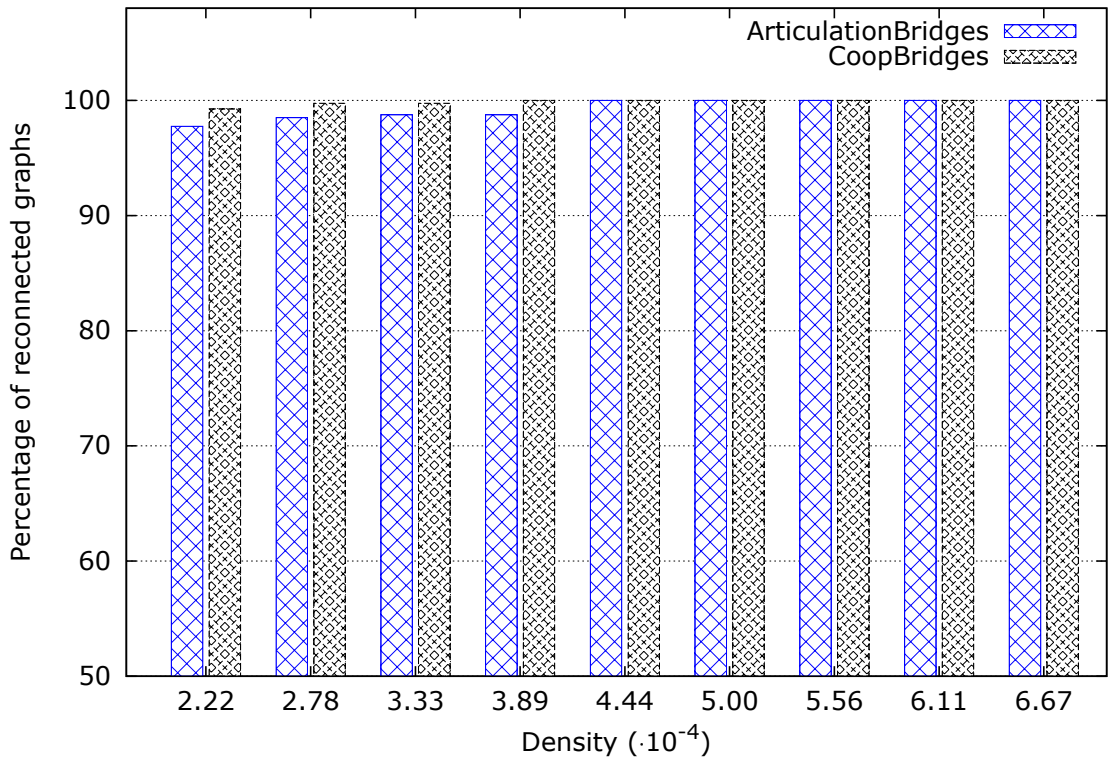



Figure 10: Average of graphs with connectivity recovered for both centralized and distributed solutions under different network density (metric **M**3).

alternative to prevent network link disruption is to identify nodes and links that may render the network disconnected. The main contribution of this work was to propose a mechanism that explored collaborative communication as an alternative to reestablish network connectivity by creating backup links. Simulation results have shown that the proposed localized scheme allowed connectivity recover in 98% of the evaluated scenarios, against an average of 99% of similar works that assume global information. Compared with the centralized approach, the proposed localized scheme was able to reduce the computational cost up to 67 times using a constant number of control messages per node.

# References


[1] Qunfeng Dong. Maximizing system lifetime in wireless sensor networks. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 13–19. IEEE, 2005.

[2] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[3] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.

[4] Devendra Goyal and James Caffery Jr. Partitioning avoidance in mobile ad hoc networks using network survivability concepts. In *Computers and Communications, 2002. Proceedings. ISCC 2002. Seventh International Symposium on*, pages 553–558. IEEE, 2002.

[5] Ivan Stojmenovic, David Simplot-Ryl, and Amiya Nayak. Toward scalable cut vertex and link detection with applications in wireless ad hoc networks. *Network, IEEE*, 25(1):44–48, 2011.

[6] Milenko Jorgic, Michaël Hauspie, David Simplot-Ryl, Ivan Stojmenovic, et al. Localized algorithms for detection of critical nodes and links for connectivity in ad hoc networks. In *Proceedings of the 3rd IFIP Mediterranean Ad Hoc Networking Workshop*, 2004.

[7] Pranay Chaudhuri. An optimal distributed algorithm for finding articulation points in a network. *Computer Communications*, 21(18):1707–1715, 1998.

[8] P Chaudhuri. An optimal distributed algorithm for computing bridge-connected components. *The Computer Journal*, 40(4), 1997.

[9] Volker Turau. Computing bridges, articulations, and 2-connected components in wireless sensor networks. In *Algorithmic Aspects of Wireless Sensor Networks*, pages 164–175. Springer, 2006.

[10] Ulisses R. Afonseca, Paulo H. Azevêdo Filho, Jacir L. Bordim, and Priscila S. Barreto. Localização e Redução do Consumo de Energia em Pontos de Articulação em Redes de Sensores Sem Fio. In *II Workshop de Sistemas Distribuídos Autonômicos*, pages 21–24, 2012.

[11] Benahmed Khelifa, Hafid Haffaf, Merabti Madjid, and David Llewellyn-Jones. Monitoring connectivity in wireless sensor networks. In *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, pages 507–512. IEEE, 2009.

[12] Jieun Yu, Heejun Roh, Wonjun Lee, Sangheon Pack, and Ding-Zhu Du. Cooperative bridges: Topology control in cooperative wireless ad hoc networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[13] Jieun Yu, Heejun Roh, and Wonjun Lee. Topology Control in Cooperative Wireless. *IEEE Journal on Selected Areas in Communications*, 30(9):1771–1779, 2012.

[14] Ahmed K Sadek, Weifeng Su, and KJ Ray Liu. Multinode cooperative communications in wireless networks. *Signal Processing, IEEE Transactions on*, 55(1):341–355, 2007.

[15] Ying Zhu, Minsu Huang, Siyuan Chen, and Yu Wang. Energy-efficient topology control in cooperative ad hoc networks. *Parallel and Distributed Systems, IEEE Transactions on*, 23(8):1480–1491, 2012.

[16] Thiago Fernandes Neves and Jacir Luiz Bordim. Topology control in cooperative ad hoc wireless networks. *Electronic Notes in Theoretical Computer Science*, 302(0):29 – 51, 2014. Proceedings of the XXXIX Latin American Computing Conference (CLEI 2013).

[17] Mihaela Cardei, Jie Wu, and Shuhui Yang. Topology control in ad hoc wireless networks using cooperative communication. *Mobile Computing, IEEE Transactions on*, 5(6):711–724, 2006.

[18] R. Sedgewick and K. Wayne. *Algorithms*. Pearson Education, fourth edition, 2011.

[19] C.E. Perkins, E.M. Royer, S.R. Das, and M.K. Marina. Performance comparison of two on-demand routing protocols for ad hoc networks. *Personal Communications, IEEE*, 8(1):16–28, Feb 2001.

[20] Kamin Whitehouse, Chris Karlof, and David Culler. A practical evaluation of radio signal strength for ranging-based localization. *SIGMOBILE Mob. Comput. Commun. Rev.*, 11(1):41–52, January 2007.

[21] M. Kohvakka, J. Suhonen, M. Hannikainen, and T.D. Hamalainen. Transmission power based path loss metering for wireless sensor networks. In *Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on*, pages 1–5, Sept 2006.

[22] Henri Koskinen. Statistical model describing connectivity in ad hoc networks. *WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 3–4, 2003.

[23] J. Ni and S.A.G. Chandler. Connectivity properties of a random radio network. *IEE Proceedings-Communications*, 141(4):289–296, 1994.

[24] J.M. Aldous and R.J. Wilson. *Graphs and Applications: An Introductory Approach*. Number v. 1 in Graphs and Applications: An Introductory Approach. Dickenson, 2000.

[25] Michael Grant, Stephen Boyd, and Yinyu Ye. Cvx: Matlab software for disciplined convex programming, 2008.

[26] Joseph C. Liberti and Theodore S. Rappaport. *Smart Antennas For Wireless Communications: IS-95 And Third Generation CDMA Applications*. Prentice Hall Communications Engineering And Emerging Technologies Series. Prentice Hall PTR, Upper Saddle River, N.J., 1999.