Outdoor 2.5D-Terrain IoT Deployment Algorithms: Repairing Partitions While Mitigating Coverage Loss

Kerry Veenstra
Baskin School of Engineering
University of California
Santa Cruz, United States
veenstra@ucsc.edu

Katia Obraczka
Baskin School of Engineering
University of California
Santa Cruz, United States
katia@soe.ucsc.edu

***Abstract*—We present and evaluate new hybrid recovery algorithms for outdoor IoT networks. Rather than using the 2D disk model, the algorithms use 2.5D terrain data to compute intersections of commsheds that identify potential positions for recovery nodes, and they compute unions of viewsheds to prioritize potential recovery positions. The algorithms are evaluated using actual terrain data. Results show that on average, TARCI-GP achieves post-recovery coverage that approaches that of a similar algorithm that incorporates an exhaustive step, while using two orders of magnitude less CPU time during pre-failure planning. Also, TARCI-GP achieves better post-recovery coverage compared to a proxy for 2D recovery algorithms. And finally, since the TARCI variants consider coverage when choosing recovery positions, we discovered that on average, TARCI variants that consider more distant recovery positions during pre-failure planning tend to achieve better post-recovery coverage.**

***Index Terms*—Outdoor IoT, 2.5D Terrain, Network Recovery**

I. Introduction

Outdoor IoT networks have a range of applications including surveillance and environmental monitoring. IoT nodes are equipped with sensors that sense environmental data and, through the wireless networks that interconnects them, transmit sensed data to data collection sites known as *data sinks*.

Sensors that equip outdoor IoT nodes are of two main kinds. *Contact* sensors measure environmental conditions at a node's location, for instance temperature and humidity. *Non-contact* sensors measure environmental conditions at a distant location and require a clear line of sight, or *CLOS*. Examples of non-contact sensors are ultrasonic microphones, cameras, and infrared imagers. This distinction is important because the CLOS requirement of non-contact sensors depends on the deployment area.

Our work considers outdoor IoT networks of non-contact sensors deployed over terrain. (Sometimes we will refer to these nodes as cameras.) This combination of camera nodes and terrain is important because the landforms of the terrain itself affect a camera's view and network connectivity, and thereby the effectiveness of a node's position.

In order to guarantee that sensor nodes are able to deliver sensed data to the data sink, outdoor IoT networks must be kept *connected*, meaning that all nodes of the network should be able to communicate with the data sink, either directly or indirectly *through* neighboring nodes. Although network connectivity can be ensured during initial deployment, subsequent node failures can partition the network, blocking one or more otherwise functional nodes from communication with the sink.

Recovery algorithms compensate for node failures, typically through node mobility. The primary goal of a recovery algorithm is to restore connectivity, but possible additional goals include mitigating loss of coverage and controlling the energy expenditure of repositioning nodes. Most existing solutions assume two-dimensional (2D) models of the environment, in which nodes are restricted to a plane. Some research uses 3D models, in which nodes are free to move unconstrained through the air or underwater. For our work, however, the deployment area is the surface of outdoor terrain, which we call *2.5D*.

Recovery from node failures has been researched for deployments whose nodes are limited to 2D and deployments whose nodes can freely travel in 3D, with researchers proposing numerous mechanisms. (See the surveys by Younis et al. [1] [2].) But outdoor 2.5D IoT networks are unique because the nodes are restricted to the surface of terrain, meaning that an outdoor IoT deployment is poorly modeled as either a 2D plane or a 3D space. Hence, current algorithms for 2D and 3D recovery are a poor match for outdoor IoT, and so we explore recovery algorithms that consider the effects of outdoor terrain directly. We discuss in more detail the effects that 2.5D terrain has on communication and sensing in Section II.

This paper introduces the first suite of recovery algorithms for outdoor IoT networks deployed over 2.5D terrain that repair network partitions while mitigating coverage losses. One of the proposed algorithms yields results that are comparable to an exhaustive approach but uses two orders of magnitude less CPU time during pre-failure planning. We also show that results of our algorithm, which is designed for 2.5D terrain, are superior when compared to results that would be possible by algorithms that are designed assuming 2D deployments.

TABLE I
DEFINITIONS

cut vertex	A node whose failure will partition a network, thereby preventing communication between at least one node and the sink.
critical node	A node of the network that is a cut vertex, and with some heuristic recovery algorithms, a node that is suspected to be a cut vertex.
CLOS	Clear Line Of Sight. Describes a direct path between two terrain points that has no obstructions.
visibility	The ability of a sensor to see a target without obstruction. Requires CLOS. In practice, visibility is restricted by a maximum range. The range limit can be due to, for example, attenuation (a microphone hearing sound) or resolution limits (a camera viewing a distant object).
viewshed	The positions of terrain that are visible to a sensor or a set of sensors.
coverage	The number of targets in a viewshed (point coverage) or the area of a viewshed (area coverage).
commshed	Communications viewshed. The positions of terrain at which one could communicate with the node or nodes that define the commshed.

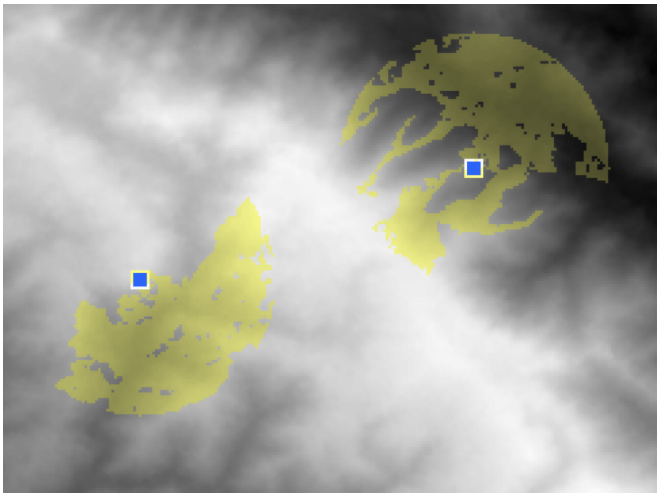


Fig. 1. Range-limited viewshed of a two-node network on 2.5D terrain. Lighter areas represent regions with higher elevation, and darker areas represent regions with lower elevation. Blue squares represent the positions of nodes. Yellow areas indicate the locations of targets that have CLOS (clear line of sight) visibility to at least one of the nodes.

The rest of this paper is organized as follows: Section II defines and states our problem, and we discuss related work in Section III. In Section IV, we present four variants of our 2.5D recovery algorithm, TARCI. Then in Section V, we describe how we evaluated the algorithms, and in Section VI we present our results. Section VII concludes the paper with some directions for future work.

II. Problem Statement

We explore the problem of repairing an outdoor IoT network following the failure of a single node. An important distinction of our recovery work is in the definition of the deployment region and the characteristics that the deployment region has on communication and visibility. Table I summarizes terminology definitions that will be used throughout the paper.

We define the deployment region as *2.5D terrain*: As previously pointed out, terrain landforms affect the sensing and communication range of outdoor IoT nodes and thus 2D terrain models are not adequate. We do not consider the region 3D because the third dimension $z = h(x, y)$, which is the height of the terrain's surface, depends on the other coordinates (x, y) rather than being an independent dimension. So instead, we call outdoor IoT terrain 2.5D [3]. This means that the nodes rest on the upper surface of the hills and valleys of terrain, and the terrain itself can block both communication and visibility.

Identifying the deployment region as 2.5D terrain is key because it leads us to understand that we cannot use a simple 2D disk model to predict coverage and connectivity. Instead, we must use a Clear Line of Sight (CLOS) visibility model. However, a pure CLOS model is inadequate for both coverage and communication.

Regarding coverage, practical cameras (and other types of non-contact sensors) have limited resolution, in the case of cameras, optical resolution. This means that, for a given sensor and environmental conditions (e.g., camera/lens pair, lighting, atmospheric conditions, and target size and contrast) there is a maximum event-detection (or *sensing*) range, R_S. Consequently, we use a *range-limited* CLOS model for coverage, meaning that a target must be line-of-sight visible to the node *and* it must be no farther away than R_S. (Note that while our analysis uses arbitrary values for R_S, one can adjust the values to model a chosen resolution.)

For measuring coverage, the range-limited CLOS model defines the concept of a *viewshed* [4], which is the region of terrain that is visible to a node. Fig. 1 shows an example of the viewsheds of a two-node deployment.

Given a node's viewshed, the viewshed's area measures the area coverage or *fitness* of the node. However, we find it better to use *point coverage*, which measures fitness as the number of targets covered, for three reasons. First, it is easier to write algorithms to process point coverage. And second, raster terrain data easily maps to a regular grid of point targets. (Note that when all targets represent equal area, which happens when targets are arranged in a rectangular grid, point coverage and area coverage are equal except for a constant multiple of the area of a target.)

The third advantage of using point coverage is that viewsheds are sets, and then the viewsheds of a network's nodes can be joined using the *union* set operation to construct the viewshed of the network: that is, the *union* set operation constructs the set of targets that are visible to at least one of the network's nodes. Then the number of visible targets is the coverage of the network.

We can then use the *change* in the coverage to evaluate the fitness of *repositioning* a node. That is, given a candidate new position for a node, the fitness of repositioning the node is the difference between the coverage of the original network viewshed and the coverage of the network viewshed that would result from repositioning the node. Now, if presented with

more than one potential node repositioning that will repair a network, we can mitigate coverage loss by ranking the candidate new positions by fitness and choosing the best.

The concept of a viewshed can be extended to communications as a *communications viewshed* or a *commshed* [5]. We observe that a CLOS model can be appropriate for communications since microwaves cannot pass through soil. Then, as with viewsheds, we limit communication range with a value R_C for the maximum communication range. (Note that while our work uses an arbitrary value for R_C, one can adjust the value to any radio propagation range.)

One important difference between viewsheds and commsheds is that except under specific circumstances, CLOS is only an approximate radio propagation model. That is, we acknowledge that there are more realistic radio propagation models, such as those that incorporate Fresnel zones [6]. We use the CLOS model for two reasons. First, the CLOS model lets us use an efficient dynamic programming visibility algorithm, and adding Fresnel Zone clearance computations will likely increase algorithm runtime significantly. Second, since this analysis compares various algorithms' performance when combining the same commshed shapes, we believe that adjusting the commshed shapes with Fresnel Zone clearance computations for all algorithms will have little effect on relative results. That said, we plan to confirm our beliefs in future work that explores the effect of different radio propagation models on our algorithms.

While our algorithms perform recovery operations through node mobility, we should also point out that they use a hybrid approach: nodes are stationary during normal operation to conserve energy, but they can become mobile during recovery operations when it is necessary for one or more nodes to reposition. Then the distance that a node travels during repositioning measures the cost of repositioning.

We design, implement and evaluate our algorithms' performance using the following system model.

- **Network Model:** It is assumed that initial deployment creates a connected network. The algorithms rely on lower-level protocols to ensure reliable message delivery.
- **Communications Model:** TARCI uses six message types which are summarized in Table II. The way that these messages are used by TARCI algorithms is described in more detail in Section IV.
- **Failure Model:** We assume single-node failures, i.e., that only a single node fails at a time. We briefly explore how we plan to relax this assumption as part of our discussion of future work (Section VII).

Summarizing our problem statement: we explore algorithms that use node mobility to restore connectivity in outdoor IoT networks over 2.5D terrain following node failure, while simultaneously mitigating coverage loss. Our algorithms compute, combine, and compare viewsheds and commsheds as part of selecting and ranking recovery actions.

TABLE II
MESSAGE TYPES

Message Type	Purpose	Payload
NEIGHBORS-R	Sink requests neighbor lists.	
NEIGHBORS	Node sends its neighbor list to the sink.	Neighbor List
ASSIGNMENT	Sink sends a recovery assignment to a recovery node.	Assigned Node
ACTIVATE	Sink instructs nodes to start recovery tasks.	
KEEP-ALIVE	Nodes detect the failure of an immediate neighbor.	
FAILED-NODE	A node that detects a failed neighbor broadcasts a FAILED-NODE message that identifies the neighbor.	Failed Node

III. RELATED WORK

Recovery from node failures has been well researched for deployments over two-dimensional (2D) planes, with researchers proposing numerous mechanisms. The surveys in [1] [2] classify and describe existing 2D recovery approaches. In the remainder of this section, we reference their taxonomy of fault tolerance to discuss related work and its applicability to 2.5D terrain.

A. Proactive Methods

According to [1] [2], one class of recovery approach is based on proactive mechanisms. For example, avoiding faults through *proactive* over-provisioning. One can do this by deploying k-connected wireless networks where $k > 1$, meaning that every pair of nodes is connected by k node-disjoint paths. Such a strategy ensures that a network can withstand up to $k - 1$ node failures without becoming partitioned.

An example of this approach using $k = 2$ is Basu & Redi [7]. In their work, a bi-connected network is created from a connected network by repeatedly identifying a connected subset of nodes and then moving the subset as a block toward the rest of the network. Each block of nodes moves until the appearance of a redundant path eliminates a cut vertex.

We note some shortcomings to this approach. First, while provisioned approaches in general prevent consequential loss of connectivity following a node failure, such solutions usually bear the cost of requiring redundant nodes. Second, if a provisioned algorithm does not add redundant nodes, then the area of coverage will be reduced by the necessity of moving nodes closer together to create redundant paths. Third, proactive approaches incur overhead even when no faults happen.

We also note a drawback from using block moves over 2.5D terrain. Any strategy for 2D can assume that block moves that maintain intra-block node-to-node distances also will maintain intra-block node-to-node communication. However, with 2.5D terrain, two nodes of a moving block may lose contact if one node drops into a valley or if both nodes move to opposite sides of a hill. That is, generally, any algorithm that assumes

that a block move will maintain intra-block communication is inappropriate for 2.5D terrain.

B. Reactive Methods

Reactive methods are an alternative to proactive approaches where recovery mechanisms are activated only when a node failure that causes the network to partition is detected.

One example of a reactive approach is RIM (Recovery by Inward Motion) [8]. The RIM algorithm works as follows: after detecting a node failure, the node's 1-hop neighbors move inward until they form a connected subnetwork. Then, if the inward motion of the failed node's 1-hop neighbors causes disconnection with any of *their* neighbors, there will be a cascading motion of nodes, with each node moving toward its inner neighbor until it reconnects.

The problem with RIM when used with 2.5D terrain is that it assumes that nodes that are closer than R_C can communicate. However, as we mentioned earlier, on 2.5D terrain, any repositioning that ignores hills and other obstructions can cause loss of communication, regardless of inter-node distance. Consequently, RIM is not appropriate for 2.5D terrain deployments.

DARA [9], an algorithm that is similar to RIM, avoids the shortcomings of deployment over 2.5D terrain. DARA will direct a node to move only to a position that was just vacated by a neighboring node. Consequently, a DARA-recovered network never attempts to establish new links between node positions. That is, DARA's cascading node motion will change which nodes occupy which positions, but the *set of positions* themselves will remain unchanged, and hence connectivity between positions will not be altered. (Aside from any connections to the leaf-node position that gets abandoned as part of DARA's network recovery effort.) This "reuse" of node positions by DARA means that it will work with 2.5D terrain. Although we must note that the very position restrictions that let DARA ensure node connectivity also prevent the algorithm from considering new connected positions that would mitigate coverage loss or even improve coverage.

C. Hybrid Methods

As just discussed, it's possible that a recovery mechanism can be purely reactive, with nodes formulating a strategy only after the need for network recovery is detected, but alternative *hybrid* approaches also are possible, with nodes' responses also incorporating pre-failure communication.

Some examples of hybrid approaches are PADRA [10], PCR [11], and NORAS [12]. During the pre-failure phase of these approaches, the critical nodes of the network self identify using a distributed algorithm. Then each critical node uses a heuristic mechanism to designate a recovery node for itself (also called a *backup*) that will activate should the critical node fail.

After the pre-failure phase is complete, the network is primed to respond to the failure of a critical node. For these algorithms the response is similar: following the failure of a critical node, the corresponding recovery node moves to take the position of the failed node, possibly initiating cascading node movement in order to maintain network connectivity. Critical nodes of PADRA and PCR potentially select a recovery node that is not an immediate neighbor, and so to balance energy usage, cascading node movement is initiated along the path between the failed node and its backup.

These approaches have a few shortcomings. First, while the distributed algorithms that are used always successfully identify cut vertices as critical nodes, they sometimes mistakenly designate a node that is *not* a cut vertex as critical. Although such a "mistake" will not harm connectivity since failure of a mistakenly identified node merely will activate the recovery mechanism on a still-connected network, such unnecessary activation will cause unnecessary energy usage.

Second, considering 2.5D terrain, while the examples that we have cited are similar to DARA in that they direct a node to move only to a position that was just vacated by a neighboring node (and so node-to-node connectivity assumptions will be maintained with 2.5D terrain), we also note that, as with DARA, these position restrictions prevent the algorithm from considering new connected positions that would mitigate coverage loss or would improve coverage.

IV. Terrain Aware Recovery with Commshed Intersections

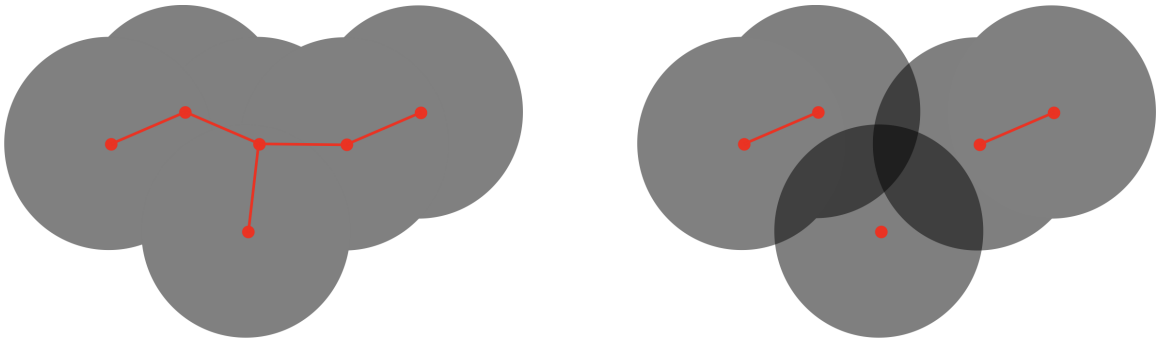
We propose a suite of hybrid repair algorithms that addresses the requirements of Outdoor IoT deployments over 2.5D terrain. We call our algorithms TARCI for Terrain Aware Recovery with Commshed Intersections.

TARCI leverages the notions of commsheds and viewsheds as they quantitatively capture our goals of connectivity restoration and coverage-loss mitigation, and based on available 2.5D terrain models, TARCI uses this data to compute the commsheds and viewsheds that guide its pre-failure planning decisions.

In other words, the key idea behind TARCI is to use commsheds and viewsheds to compute the set of recovery positions for a partitioned network by finding the intersection of the partitions' commsheds (Fig 2). Then, given a set of potential recovery positions provided by the commshed intersection, one can compute the viewsheds of each of the potential recovery positions to prioritize the positions based on the amount of additional coverage that each potential recovery position provides.

During algorithm development we observed that in some circumstances the commshed intersection is quite large, meaning that we should try to limit the number of potential recovery positions (and corresponding viewsheds) that are evaluated during the pre-failure planning phase. So to understand the necessity and benefits of such limits, we developed and evaluated four variants of TARCI that use different approaches to prioritize potential positions for each candidate recovery node:

- TARCI-E exhaustively considers all possible new positions for a recovery node. It expends the most energy during pre-failure planning of all of the algorithm variants,



(a) Commshed of a network. The gray region indicates positions that are within distance R_C of at least one node of the network. A new node that is placed anywhere in the commshed can communicate with the network.

(b) Commsheds of a partitioned network. The darkest gray region indicates the intersection of all of the partitions' commsheds. A new node that is placed anywhere in the intersection can communicate with all three partitions and will repair the network.

Fig. 2. Conceptual example of using commshed intersections to identify candidate positions for a recovery node. For clarity this example uses the 2D disk model, but one could extend the example to 2.5D by replacing the disk model with a terrain-based range-limited CLOS model, similar in appearance to the viewsheds shown in Fig 1.

but it will find the optimal recovery solution. Hence, we use TARCI-E as baseline in our evaluations, i.e., worse case for energy consumption and best case for recovery optimality.

- TARCI-H uses the height of each possible new position as a proxy of the position's potential communication effectiveness, and so it evaluates the highest H positions of the commshed intersection. It expends much less energy during pre-failure planning than TARCI-E. When choosing a value for control parameter H, there are trade-offs between the quality of the recovery solution and the pre-failure energy usage, which we discuss in Section VI.
- TARCI-GP is similar to TARCI-H, but it uses the Grid Partition [13] mechanism to distribute the considered positions across the entire commshed intersection, rather than considering only the highest positions, which will tend to cluster together. Grid Partition places a grid of approximately H squares over the commshed intersection and then evaluating the highest position of each grid square. For a given value of H, TARCI-GP expends about the same amount of energy during pre-failure planning as TARCI-H. And as with TARCI-H, the chosen value for control parameter H leads to trade-offs between the quality of the recovery solution and the pre-failure energy usage (see Section VI).
- TARCI-D minimizes energy usage during both pre-failure planning and recovery. During pre-failure planning, it immediately chooses the recovery position within the commshed intersection that is closest to the candidate recovery node's current position. Then during recovery, the node can reach the commshed intersection by traveling the shortest distance.

The pre-failure planning phase of a hybrid recovery algorithm typically corresponds to a relatively small amount of time compared to the network's total deployment time. We also note that the sinks usually are larger nodes equipped with more computation capabilities and either are equipped with larger battery packs/solar panels or are connected to continuous energy sources. Hence, we made the design choice of using a centralized algorithm instead of a distributed one during pre-failure planning to: (1) identify critical nodes and (2) choose appropriate failure responses. As our experimental results demonstrate (see SectionVI), this results in a relatively small energy usage increase due to additional communication.

Algorithm 1 Pre-Failure Planning: Network Initialization by the Sink. Input: network graph N.

```
1: Compute each node's commshed.
2: V ← cut vertices of N
3: for every cut vertex v ∈ V do
4:     for every node r ∈ N where r ≠ v do
5:         Evaluate the effectiveness of moving node r to recover from the failure of cut vertex v. {See Algorithm 2.}
6:     end for
7:     Choose the node r with the best recovery response and assign it as the recovery node for cut vertex v.
8: end for
```

A. Pre-Failure Planning

Pre-failure planning computations comprise the bulk of the recovery algorithm's work and are performed while the network is whole. The overall goal of pre-failure planning is for TARCI to select a recovery response for every cut vertex of the network and then send appropriate instructions to corresponding recovery node(s).

Pre-failure planning begins when the sink broadcasts a NEIGHBORS-R request message All messages exchanged are summarized in Table II. In response, each node sends a list of its neighbors to the sink in a NEIGHBORS message. Using this information, the sink identifies the network's cut vertices.

Algorithm 2 Pre-Failure Planning: Evaluating the effectiveness the moving node r to recover from the failure of cut vertex v. Inputs: network graph N, node r, and cut vertex v.

1: $N' \leftarrow N - r - v$
2: Split N' into its connected components $P_1 \dots P_n$.
3: $C \leftarrow \bigcap_{i=1}^{n}$ `commshed(`P_i`)`
{Intersection C contains all of the potential new positions for node r that will reconnect the network.}
4: **if** $C = \varnothing$ **then**
5: Return failure.
6: **else**
7: Select all or a subset of the positions $c_i \in C$. {See text.}
8: For every selected c_i compute `fitness(`c_i`)`.
9: Return the best `fitness(`c_i`)` and the corresponding c_i.
10: **end if**

Algorithm 3 Node Failure-Response Task #1.

1: **if** this node detects that a neighbor has failed **then**
2: Broadcast a FAILED-NODE message that identifies the failed neighbor.
3: **if** this node is the recovery node for the failed neighbor **then**
4: Take the assigned recovery action.
5: **end if**
6: **end if**

Algorithm 4 Node Failure-Response Task #2.

1: **if** this node is the assigned recovery node for a received FAILED-NODE message **then**
2: Take the assigned recovery action.
3: **end if**

Then, for every cut vertex identified, the sink evaluates every other node as a potential recovery node and then sends an ASSIGNMENT message to the best node seen as the recovery node for the cut vertex (see Algorithm 1). Using commshed intersections to identify new recovery node positions and then ranking those positions using viewsheds makes up the core of TARCI's pre-failure planning as illustrated in Algorithm 2.

Below we describe both Algorithms 1 and 2 in more detail.

Algorithm 1 – Network Initialization by the Sink: Input is the network graph N. The algorithm relies on knowledge of each node's commshed, and so the commsheds are computed in **Step 1**. **Step 2** identifies the cut vertices of the network. We note that since the relative execution time of this step is relatively small, one can use the simple approach of removing each vertex one-by-one and then checking for a disconnected network with depth-first search. However, if one desires better asymptotic performance, Problem 22-2 of [14] presents an alternative. **Steps 3–8** form a loop that formulates a recovery response for the failure of each cut vertex v. Within the loop, **Steps 4–6** evaluate the capability of every other node r to respond for the failure of cut vertex v. Then **Step 7** compares node capabilities and chooses the best node r to act as the recovery node for cut vertex v.

Algorithm 2 – Recovery Node Effectiveness: This subroutine accepts as input a network graph N, the position of a failed cut vertex v, and the position of a recovery node r. **Step 1** removes cut vertex v and node r from the network N. This network modification is needed to evaluate the failure of cut vertex v along with node r moving in response. **Step 2** computes the combined effect of these changes: a set of network partitions $P_1 \dots P_n$. Next, **Step 3** determines which potential new positions for node r could repair the network. This set of potential new positions, C, is the intersection of all of the partitions' commsheds. **Steps 4–10** determine the result of the algorithm. If C is the empty set, then the algorithm indicates that there is no new position for node r that will let the network recover from the loss of cut vertex v. (One might think that moving node r to the position of failed cut vertex v always would repair the network, but r might be a cut vertex itself, and so finding a repair position for r is not certain.) If set C is not empty, then it contains one or more potential new positions for node r, and in **Step 7** all or some of these positions are evaluated for fitness. The set of positions that are evaluated depends on the variant of TARCI that is running (TARCI-E, TARCI-H, TARCI-GP, and TARCI-D), as mentioned earlier. **Step 8** determines the fitness of all of the selected positions, and then **Step 9** compares the fitness results to return the best position.

Once all recovery assignments have been made, the sink broadcasts an ACTIVATE message, and the nodes start their Failure-Response Tasks.

B. Failure Response

After pre-failure planning, all nodes receive an ACTIVATE message and run a pair of failure-response tasks. One task (Algorithm 3) uses KEEP-ALIVE messages to detect the failure of an immediate neighbor. Upon detecting such a failure, it broadcasts a FAILED-NODE message to its partition that identifies the failed neighbor. Then, if it is the failed neighbor's designated recovery node, it performs its assigned recovery action.

The other task (Algorithm 4) is run by nodes that have been designated as a recovery node. This task checks all FAILED-NODE messages to determine whether the node needs to activate its recovery response for a failed node.

V. Evaluation

A. Experimental Evaluation Infrastructure

To evaluate and compare the different variants of TARCI, we modeled and simulated its algorithms using a C++ version of TAFFI (Terrain-Aware Framework for IoT) [15]. The TAFFI/C++ experimental platform better matches TARCI's centralized algorithms since the alternative of using TAFFI's Cooja-Contiki version would have added the complication of

Algorithm 5 Connected Greedy Adding. Input: network graph N and position of the sink s.

1: $C \leftarrow \text{commshed}(s)$
2: **for** every node $x_i \in N$ **do**
3: Place node x_i at the position in C that best improves coverage.
4: $C \leftarrow C \cup \text{commshed}(x_i)$
5: **end for**

running a centralized algorithm as a Java-based model in a distributed Cooja-Contiki simulation environment.

Using the TAFFI/C++ framework, our analysis tasks were facilitated through the following features:

- A C++ library of line-of-sight algorithms that aid the creation of command-line simulation models for 2.5D terrain. Using the library improves code quality and programming efficiency.
- A benchmark suite of actual terrains that help with reproducibility and algorithm comparisons.
- Command-line job management using GNU Make [16], preventing our inadvertently duplicating simulation runs and also helping us run simultaneous simulations on multiple servers.
- Data collection from thousands of simulation logs using GNU Awk [17] scripts.
- MATLAB [18] scripts that analyze data and generate plots as Encapsulated PostScript files, which are used with LaTeX [19] in the creation of technical papers.

Simulations were run on a compute server with two Intel® Xeon® Processor E5-2670 CPUs running at 2.60 GHz, providing the server with a total of 32 threads. As one of our performance metrics, we measured processor time using the C++ `clock()` function.

B. Benchmark Networks

Each of the TARCI algorithm variants was evaluated for its post-recovery coverage and estimated energy usage as it repaired 475 different cut vertices in 200 benchmark networks.

Each of the 200 benchmark networks was created using a three-step process: (1) choose a benchmark terrain subtile from a terrain dataset, (2) place a sink in one of ten predefined positions on the subtile and deploy ten nodes into a connected network, and (3) if necessary, decrease the communications range, breaking links until the network has one or more cut vertices.

For our source of 2.5D benchmark terrain subtiles, we leveraged our prior work on two terrain classifiers that help us verify that a set of Shuttle Radar Topography Mission (SRTM) [20] terrain subtiles represent a wide range of landform variation [13] [15].

The ten sink positions are chosen from locations on a regular triangular grid. Using a grid ensures that the possible sink positions will be distributed uniformly over the subtile and will help create a variety of experimental setups.

Then, after placing the sink, we run *Connected Greedy Adding*, or CGA (see below for more details on CGA) to compute an initial network deployment of ten nodes. We chose to use ten-node networks because from earlier work with this source of terrain data, we found ten nodes to be ideal with the 240-by-180 unit terrain subtiles and our chosen value of R_C. Choosing a larger number of nodes with the same subtile would require us to decrease the value of R_C, which would reduce the resolution of the commsheds.

Connected Greedy Adding is a modification of the set-cover heuristic Greedy Adding [21] that, in addition to trying to optimize coverage, also includes steps to ensure that a network is connected (see Algorithm 5). The salient additions are **Step 1** and **Step 4**, which construct and maintain the commshed C. This commshed is the set of all positions of the terrain that can communicate with the sink or with any already placed node of the network. The algorithm ensures network connectivity by construction because **Step 3** places nodes only in positions that are part of the current commshed.

The result of the algorithm is initial positions for all nodes x_i, thereby defining a network that has good coverage and that also is connected. We repeat this first step using 200 different pairs of subtiles and sink positions to generate 200 benchmark networks.

Although the benchmark networks that are generated by this first step have good coverage and also are connected, some of the networks also are inadvertently bi-connected, meaning that they lack cut vertices. Since a benchmark network for testing a recovery algorithm must have at least one cut vertex, we perform a third step on any bi-connected networks to generate one or more cut vertices. The third step is to reduce the communications range R_C for the benchmark network until at least one cut vertex appears. After this process is complete, all of the 200 benchmark networks have a specific value for R_C and one or more cut vertices.

C. Experiments

We evaluated all of the algorithms using three performance metrics: (1) CPU time used during pre-failure planning (to compare relative pre-failure-planning energy usage), (2) change in coverage following network recovery after the loss of a cut vertex, and (3) distance traveled during network recovery after the loss of a cut vertex (to compare relative recovery energy usage).

Also, to allow comparing the performance of TARCI variants against 2D algorithms, we devised a proxy algorithm that will perform no worse than the 2D algorithms. Since 2D recovery algorithms often ignore coverage, we expect that they will perform no better than a proxy algorithm that merely moves the network's remaining nine nodes into the original positions of nodes x_1 through x_9. The idea is that nodes x_1 through x_9 are in the positions chosen by Connected Greedy Adding for a nine-node network, and so they should provide better coverage than the positions chosen by any 2D recovery algorithm that ignores coverage. We call this algorithm *9-Node CGA* and use it as a proxy for 2D recovery algorithms like

TABLE III
CONTROL PARAMETERS

Parameter Purpose	Name	Value
Number of Nodes	m	10
Initial Communications Range	R_C	130
Event-Detection Range	R_S	50
Number of Candidate Positions	H	$\{10, 20, 30, 40, 50, 100\}$

TABLE IV
PRE-FAILURE PLANNING CPU TIME

Algorithm	H	Pre-Failure Planning Minutes CPU Time	Post-Recovery Change in Coverage
TARCI-E	—	65	-4.7%
TARCI-GP	100	2.4	-5.0%
TARCI-GP	30	0.74	-5.2%
TARCI-GP	10	0.30	-6.0%
TARCI-H	100	2.7	-5.9%
TARCI-H	30	0.77	-6.6%
TARCI-H	10	0.37	-7.6%
9-Node CGA	—	0.0	-6.5%
TARCI-D	—	0.064	-11.2%

DARA [9]. Since 9-Node CGA simply restores the remaining nodes to the original positions of nodes x_1 through x_9, it expends essentially zero energy during pre-failure planning.

We ran algorithms TARCI-E, TARCI-H, TARCI-GP, and TARCI-D over all 475 cut verticies of the benchmark networks. Table III shows the values of control parameters used during the experiments. We chose values of m, R_C, and R_S that worked well in other research experiments with this size of terrain subtile.

Since TARCI-H and TARCI-GP evaluate H positions, in order to determine the sensitivity of these algorithms to the control parameter H, we repeated the tests using the six values of H listed as *Number of Candidate Positions* in Table Table III.

VI. RESULTS

CPU time for pre-failure planning: Table IV compares the average pre-failure-planning CPU time with the average post-recovery change in coverage. As expected, the exhaustive TARCI-E algorithm uses the most CPU time, but TARCI-GP with $H = 30$ shows nearly as good post-recovery coverage as TARCI-E while using about two orders of magnitude less CPU time for pre-failure planning! TARCI-H is similar, but its performance is slightly worse than TARCI-GP. Finally, we note that looking at the post-recovery change in coverage, TARCI-H and TARCI-GP best 9-Node CGA, our proxy for the 2D recovery algorithms like DARA [9].

Change in coverage: Another view of post-recovery change in coverage is presented in Fig. 3 where we can see that nearly all of the TARCI algorithms (except for TARCI-D and TARCI-H with $H = 10$) have superior recovery responses.

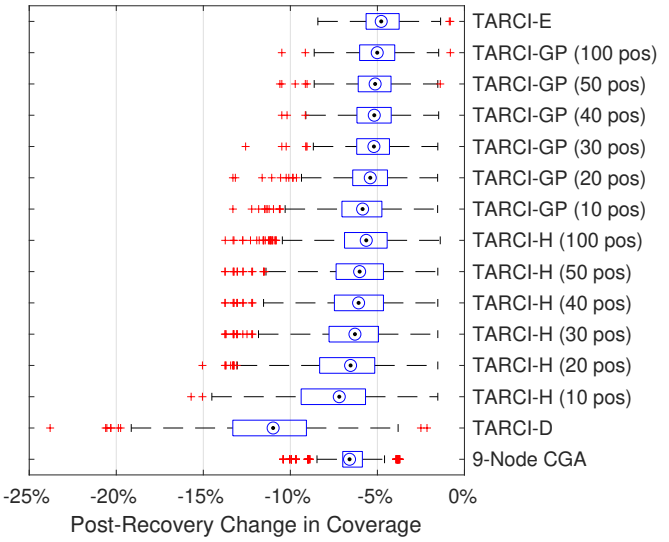


Fig. 3. Change in coverage boxplot.

We are not surprised by this result because—similar to 2D algorithms DARA, PADRA, PCR, and NORAS—9-Node CGA never will direct a node into a previously unoccupied position, while the TARCI algorithms, without this restriction, can find positions with superior coverage. As noted in the previous section, the 2D algorithms that ignore coverage are expected to perform no better than 9-Node CGA.

Distance traveled during recovery: Fig 4 estimates energy usage during failure response from the distance that a backup node travels. Comparing TARCI-GP and TARCI-H, we see that TARCI-GP reduces coverage less than TARCI-H, but it also expends more recovery energy. The positive correlation between the goodness of the recovery response and the distance traveled can be explained by noting that TARCI-GP searches more widely, and so it is able to find a superior, more distant position. The datapoint for TARCI-D is consistent with this explanation, since TARCI-D always chooses the shortest possible recovery distance, and consequently it ignores any positions that have better coverage. Comparing datapoints of all of the TARCI variants, the positive correlation mentioned above between the average distance traveled and the average change in post-recovery coverage is clear.

Finally, considering the sensitivity of TARCI-GP and TARCI-H to the value of H, Fig. 5 shows that TARCI-GP's average effectiveness is nearly as good as that of TARCI-E when TARCI-GP considers 30 positions. The graph also shows that TARCI-H must consider at least 40 positions in order to better the results of 9-Node CGA.

VII. CONCLUSION

In this paper, we introduced TARCI, a suite of recovery algorithms for outdoor IoT networks deployed over 2.5D terrain. TARCI's main contribution is that it repairs network partitions while mitigating coverage loss. It leverages the concepts of commsheds and viewsheds to restore connectivity and reduce

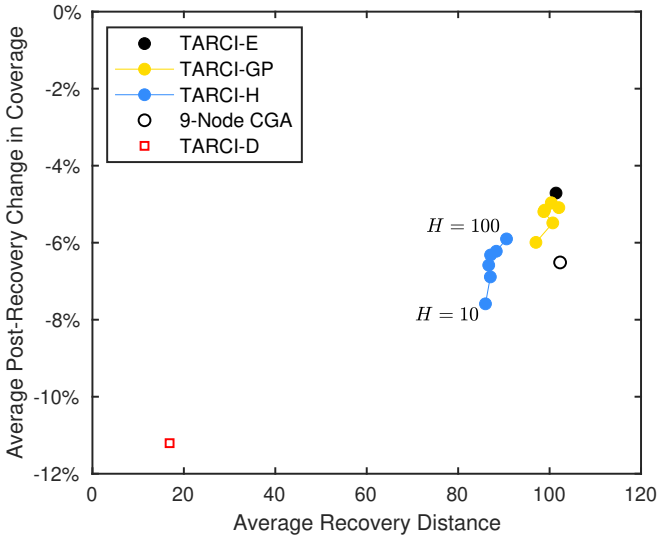


Fig. 4. Change in coverage vs distance.

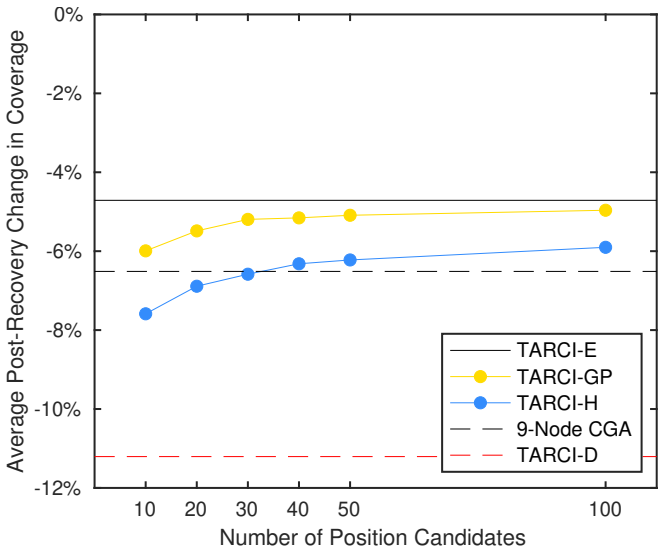


Fig. 5. Change in coverage vs num positions.

the effect of node failures on coverage. Our experimental results obtained on a wide variety of scenarios indicate that one of TARCI's variants, TARCI-GP achieves nearly ideal average results compared to the TARCI-E exhaustive search variant, but with two orders of magnitude less CPU time. We also discovered that TARCI-GP performs better than a proxy for 2D recovery algorithms. Finally, we discovered that considering more distant recovery positions during pre-failure planning can lead to better post-recovery coverage.

Some directions of future work we have identified are as follows. We plan to compare the effect of different radio-propagation models on TARCI's results, such as including the effect of Fresnel Zones. Another area we plan to explore is to evaluate the benefit of increasing node communication power as a recovery mechanism, either instead of or in addition to TARCI-GP's hybrid algorithm. We will also explore mechanisms to reduce the probability of node failures, e.g., by instructing nodes to report their battery's state of charge to the sink, thereby helping to avoid simultaneous node failures by initiating recovery of some nodes early.

References

[1] M. Younis, S. Lee, I. Senturk, and K. Akkaya, "Topology management techniques for tolerating node failure," in *The Art of Wireless Sensor Networks, Volume 1: Fundamentals*, H. M. Ammari, Ed. Berlin: Springer, 2014, pp. 273–311.

[2] M. Younis, I. Sentruk, K. Akkaya, S. Lee, and F. Senel, "Topology management techniques for tolerating node failures in wireless sensor networks: a survey," *Computer Networks*, vol. 58, pp. 254–283, 2014.

[3] Wikipedia contributors, "2.5D (machining) — Wikipedia, the free encyclopedia," 2019. [Online]. Available: http://en.wikipedia.org/wiki/2.5D_(machining)

[4] K. Chang, *Introduction to geographic information systems*, 6th ed. McGraw-Hill, 2012.

[5] H. M. Dodd, "The validity of using a geographic information system's viewshed function as a predictor for the reception of line-of- sight radio waves," Master's thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, August 2001.

[6] Wikipedia contributors, "Fresnel zone — Wikipedia, the free encyclopedia," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Fresnel_zone

[7] P. Basu and J. Redi, "Movement control algorithms for realization of fault-tolerant ad hoc robot networks," *IEEE Network*, vol. 18, no. 4, pp. 36–44, 2004.

[8] M. Younis, S. Lee, and A. Abbasi, "A localized algorithm for restoring internode connectivity in networks of moveable sensors," *IEEE Trans. Computers*, vol. 59, pp. 1669–1682, 12 2010.

[9] A. A. Abbasi, K. Akkaya, and M. Younis, "A distributed connectivity restoration algorithm in wireless sensor and actor networks," in *32nd IEEE Conference on Local Computer Networks (LCN 2007)*. New York: IEEE, 2007, pp. 496–503.

[10] K. Akkaya, A. Thimmapuram, F. Senel, and S. Uludag, "Distributed recovery of actor failures in wireless sensor and actor networks," *2008 IEEE Wireless Communications and Networking Conference*, pp. 2480–2485, 2008.

[11] M. Imran, M. F. Younis, A. B. M. Said, and H. Hasbullah, "Partitioning detection and connectivity restoration algorithm for wireless sensor and actor networks," *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pp. 200–207, 2010.

[12] K. Vaidya and M. F. Younis, "Efficient failure recovery in wireless sensor networks through active, spare designation," *Proc. of the 1st Int'l Workshop on Interconnections of Wireless Sensor Networks (IWSN'10)*, 2010.

[13] K. Veenstra and K. Obraczka, "Grid partition: an efficient greedy approach for outdoor camera iot deployments in 2.5D terrain," in *29th International Conference on Computer Communications and Networks (ICCCN)*. New York: IEEE, 2020, pp. 1–9.

[14] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge: MIT Press, 2009.

[15] K. Veenstra and K. Obraczka, "TAFFI: A distributed deployment framework for iot networks over 2.5D terrain," 2022, unpublished.

[16] R. M. Stallman, R. McGrath, and P. D. Smith, "The GNU Make Manual," 2020. [Online]. Available: https://www.gnu.org/software/make/manual

[17] A. D. Robbins, "Gawk: Effective AWK Programming," 2020. [Online]. Available: https://www.gnu.org/software/gawk/manual

[18] The MathWorks, Inc., "MATLAB," 2021. [Online]. Available: https://www.mathworks.com/products/matlab.html

[19] L. Lamport, *LATEX: A Document Preparation System*. Boston, Massachusetts: Addison-Wesley Pub. Co, 1986.

[20] T. G. Farr, P. A. Rosen, E. Caro, R. Crippen, R. Duren, S. Hensley, M. Kobrick, M. Paller, E. Rodriguez, L. Roth, D. Seal, S. Shaffer, J. Shimada, J. Umland, M. Werner, M. Oskin, D. Burbank, and D. Alsdorf, "The shuttle radar topography mission," *Reviews of Geophysics*, vol. 45, no. 2, pp. 1–33, 2007.

[21] R. Church and C. ReVelle, "The maximal covering location problem," *Papers of the Regional Science Association*, vol. 32, pp. 101–118, 12 1974.