
Movement Control Algorithms for Realization of Fault-Tolerant Ad Hoc Robot Networks

Prithwish Basu and Jason Redi, BBN Technologies

Abstract

Autonomous and semi-autonomous mobile multirobot systems require a wireless communication network in order to communicate with each other and collaboratively accomplish a given task. A multihop communications network that is self-forming, self-healing, and self-organizing is ideally suited for such mobile robot systems that exist in unpredictable and constantly changing environments. However, since every node in a multihop (or ad hoc) network is responsible for forwarding packets to other nodes, the failure of a critical node can result in a network partition. Hence, it is ideal to have an ad hoc network configuration that can tolerate temporary failures while allowing recovery. Since movement of the robot nodes is controllable, it is possible to achieve such fault-tolerant configurations by moving a subset of robots to new locations. In this article we propose a few simple algorithms for achieving the baseline graph theoretic metric of tolerance to node failures, namely, biconnectivity. We formulate an optimization problem for the creation of a movement plan while minimizing the *total distance moved* by the robots. For one-dimensional networks, we show that the problem of achieving a biconnected network topology can be formulated as a linear program; the latter lends itself to an optimal polynomial time solution. For two-dimensional networks the problem is much harder, and we propose efficient heuristic approaches for achieving biconnectivity. We compare the performance of the proposed algorithms with each other with respect to the total distance moved metric using simulations.

Advances in electronics and mechanics have provided the basis technologies required for sophisticated robots. It is well recognized that robots have significant operational advantages over humans as they can perform tasks without requirements for rest, food, shelter, or task heterogeneity. Additionally, *teams* of robots can often perform large-scale tasks more effectively than single robots or augment humans in group operations such as search-and-rescue or military operations. Whether the robot team is performing distributed sensing and collection on Mars or providing airborne relay capabilities in disaster or military contexts, there will be a critical need for robots to initiate and maintain communications to ensure timely and efficient task completion. Indeed, the U.S. Joint Forces Command expects “autonomous networked robotics” to be typical participants on the battlefield by 2025 [1].

Traditionally, robotics researchers have proposed the use of *centralized* robot networks where all members of a team of robots communicate with a central controller (base station) over a wireless medium [2]. However, in many application scenarios, it is difficult to guarantee the presence of a wireless base station that can coordinate the flow of information between any two robot units. Moreover, the movement of robots can be severely restricted in order to keep in communication range of the base station, and this can hamper the task the robot team plans to execute. Hence, we believe that self-

forming, self-healing, and self-organizing multihop communications networks are ideal for autonomous and semi-autonomous multirobot systems.

Although numerous ad hoc network protocols, also called packet radio, mobile ad hoc networks (MANETs), or self-organizing networks, have been proposed and implemented, all of them were designed to be completely transparent to applications. The resulting extended applicability, however, comes at the cost of severe restrictions in the exchange of information between the application and the network, which makes it virtually impossible for them to anticipate each other's behavior and thus cooperate. In robotic systems such cooperation is highly desirable because robotic applications generally entail movement, which directly affects the communication network; conversely, the propagation of radio transmissions used for communication may be able to provide an additional means of sensing the environment. Such interaction is a feasible proposition since robots are unique in their integrated design in that the mission control, motion control, and networking protocols are typically all implemented within the same architecture.

In standard MANETs, the position and motion of a node are determined by its owner and cannot be controlled by other nodes; but in ad hoc networks consisting of robot nodes, these properties are controllable from other nodes in the network. In this article we focus on the specific problem of alter-

ing the positions of robots in order to achieve a desirable ad hoc network topology starting from an arbitrary initial connected configuration.¹ In particular, we focus on fault tolerance: our goal is to move a subset of robot nodes from their initial locations to a new set of locations such that the new connectivity graph is more tolerant to node failures than the initial graph. We utilize a baseline property for fault tolerance from graph theory — biconnectivity — and propose a few simple algorithms that attempt to achieve this network property in a distributed manner. We compare the algorithms against each other with respect to a *total distance traveled metric* (D_{total}) that should be minimized. We show by simulations that the block movement algorithm completely outperforms the baseline contraction algorithm with respect to this metric.

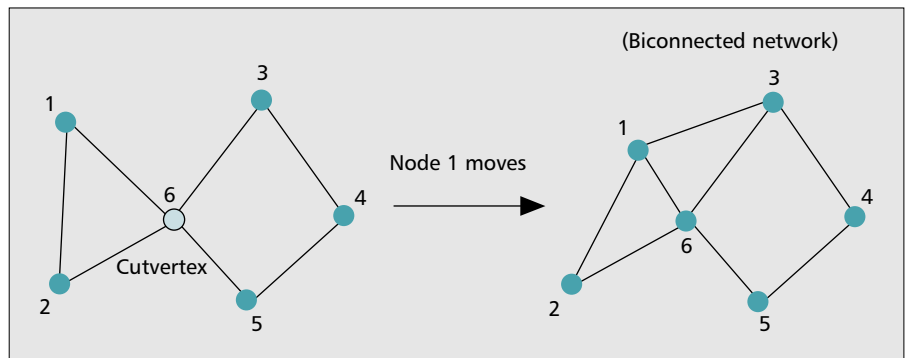
The value of maintaining fault tolerance vs. other mission goals can be balanced by some separate autonomous goal resolution algorithm within the robot. In some military contexts, the roles of the robotics may be *only* to ensure communications for other (human) participants, while in search-and-rescue operations it may be simply be a desired state to maintain in the context of the overall physical search progress.

The rest of the article is organized as follows. We present a brief introduction to fault tolerance in network design and some related work. We present a mathematical formulation of the problem and describe movement control algorithms. Performance evaluation results are presented. We conclude the article with pointers to future research directions.

Fault Tolerance in Network Design

An ad hoc network can be represented by a graph in which each vertex represents a network node, and each edge represents the fact that the endpoints are in communication range of each other. A graph is *bipartite* if its vertices can be divided into two sets so that all edges connect two vertices from different sets. A graph is *biconnected* if it remains connected after removing any of its vertices. Each such vertex is referred to as a *cutvertex* or an *articulation point*. An edge whose removal disconnects the graph is referred to as a *bridge*.² Clearly, *biconnectivity* is a desirable property for network fault tolerance. In a battle-bot scenario, if the ad hoc network formed by the robots is biconnected, even if any one robot fails or is shot down by the enemy, the network still remains connected. Hence, robots should always attempt to form a biconnected network as long as it does not interfere with their current mission. This necessitates movement for some of them (assuming no power control in the radios) in order to *create* extra links such that the resultant topology is biconnected. Figure 1 illustrates the idea. The more general property of k -connectivity is seemingly much harder to achieve, and we do not investigate it in this article.

Information about locations of nodes in k -hop neighbor-



■ Figure 1. Achieving biconnectivity by node movement.

hoods of every node (k being constant) may not be adequate for designing a deterministic algorithm that achieves biconnectivity by node movement while keeping the D_{total} metric in consideration.³ Hence, proactive link-state-based MANET routing protocols such as Optimized Link State Routing (OLSR) [3] and Hazy Sighted Link State (HSLs) [4] in which each node shares the knowledge about the rest of the network will be most easily applicable to our algorithms. For the purpose of designing and evaluating the algorithms, we imply that all nodes maintain canonical network state information and execute the same biconnectivity algorithm. This can certainly be modified to reduce the amount of redundancy without loss of our algorithm performance through the election of a single node dynamically designated to perform group decisions based on the network state.

We admit the possibility of the existence of randomized techniques that use limited neighborhood information to iteratively achieve biconnectivity with a certain probability, but we did not investigate such techniques in this work. Instead, our focus is on deterministic techniques that yield biconnectivity while attempting to minimize the D_{total} metric.

Related Work

To the best of our knowledge this is the first approach to efficient fault-tolerant network design using node movement as a primitive. Ramanathan *et al.* have proposed optimal schemes for topology control in ad hoc networks using variation of transmission power as a primitive [5]. They monotonically increase transmit power locally at every node, and attempt to satisfy properties of connectivity and biconnectivity. Our problem is very different from theirs since the movement of a node aimed at the removal of a cutvertex by creating a new edge can instead result in the creation of a cutvertex at another location in the network. Although our algorithms ensure that new cutvertices are not created during node movement, the Euclidean nature of the D_{total} metric introduces difficulties in the characterization of the optimal solution.

Li and Rus mention the use of node motion in order to relay messages between nodes in [6], but do not consider the problem of fault-tolerant network design. Winfield has proposed the use of MANETs for networking between robots [7], but he too has not considered the problem of moving robots to achieve desirable network configurations.

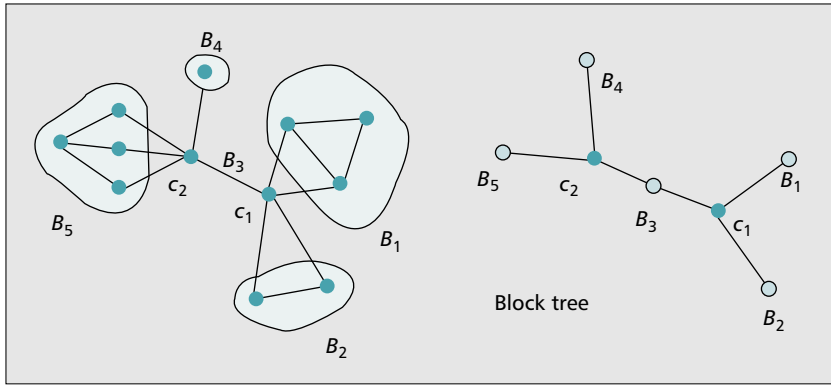
Movement Control Algorithms

In this section we present various algorithms for movement of robot nodes with the objective of making the network bicon-

¹ Since we generally advocate close cooperation between the ad hoc network subsystem in a robot and the subsystems controlling its mission and motion, dissemination of node locations is achieved without much overhead.

² Thus, the bridges in a graph are precisely those edges that do not lie on any cycle.

³ In the worst case, very little movement by a single node X may remove a cutvertex that is $O(n)$ hops away, and this step cannot be computed deterministically by X by utilizing limited neighborhood information.



■ Figure 2. a) Decomposition of a network into biconnected components; b) the corresponding block tree.

nected.

The One-Dimensional Case

We briefly analyze the 1D version of the problem where N nodes lie in a straight line and are allowed to move only in two directions along the line. Suppose the initial positions of the nodes are given by $p_i \in \mathbb{R}$, $1 \leq i \leq N$. We assume that each node's transmission range is 1.0. Our objective is to compute a *node movement schedule* such that the final configuration given by positions $x_i \in \mathbb{R}$, $1 \leq i \leq N$ is biconnected. At the same time we want to minimize the total (or average) distance moved by nodes in the network. We can formulate the above problem as follows:

$$\text{minimize } D_{total} = \sum_{i=1}^N |x_i - p_i|$$

$$x_1 \geq p_1; \quad (1)$$

$$x_N \leq p_N; \quad (2)$$

$$x_i - x_{i-1} \geq 0, \quad 2 \leq i \leq N; \quad (3)$$

$$x_i - x_{i-2} \leq 1, \quad 3 \leq i \leq N. \quad (4)$$

Constraints 1 and 2 are nonbinding constraints that just illustrate the fact that the 1D network will compress in length after a biconnected configuration is reached. The $N - 1$ linear ordering constraints in 3 restrict the search space as no node needs to move past its neighbors to achieve biconnectivity. Biconnectivity is ensured by the $N - 2$ constraints, which basically impose a condition on the nodes that every alternate pair of nodes are within transmission range of each other. It is easy to see that these constraints are necessary and sufficient for ensuring biconnectivity. Although the objective function has a nonlinear term (absolute value), the problem can be easily transformed into a linear program (LP), which can then be solved optimally in polynomial time (as a function of N). We direct the reader to [8] for details.

The Two-Dimensional Case: A Contraction Algorithm

In a higher-dimensional setting, the optimal movement problem is harder than in the 1D case due to the extra degrees of freedom for each node. The biconnectivity constraints cannot be formulated *locally* for each node in Euclidean networks as in line networks (constraint 4). Besides, the objective function (D_{total}) is nonlinear. Hence, techniques proposed earlier cannot be applied here.

Although a proof of NP-completeness of the optimal "biconnectivity by movement" problem has eluded us so far, all our attempts have indicated that the 2D case is much harder to solve than its 1D counterpart. We therefore propose practical heuristic solution approaches, and leave rigorous

analysis and the search for bounded factor approximation algorithms for future investigation.

One simple heuristic that can be implemented easily in a distributed fashion is *contraction*. Every robot node includes its location information (GPS coordinates or indoor relative location information) whenever it floods a link state update (LSU) to the rest of the network. When LSUs have arrived at node X from all other nodes, X extracts the location information from each LSU and calculates the geographic center C for the entire network using the following formula:

$$\bar{C} = \frac{1}{N} \sum_{i=1}^N \bar{p}_i,$$

where \bar{p}_i is the position vector of node i . After calculating \bar{C} , each node j independently moves toward \bar{C} by a weighted distance $(1 - \alpha) \|\bar{C} - \bar{p}_j\|$, where α is the contraction parameter. The rationale here is that as nodes move inward, more edges are added to the network and the cutvertices removed, and after a certain stage the network becomes biconnected. As a result of following the contraction algorithm, nodes near the periphery of the network move greater distances than the nodes located in the interior of the network.

The choice of parameter α is important here. Choosing a small α results in unnecessarily dense networks (albeit with high connectivity), whereas a large α results in little change in network topology. In the latter case, the contraction algorithm is applied iteratively until the network becomes biconnected. Since each node i travels on the same straight line (even over multiple iterations) joining its starting position \bar{p}_i and \bar{C} , when the network reaches a biconnected configuration with node positions \bar{x}_i , the total distance traveled is given by

$$D_{total} = \sum_{i=1}^N \|\bar{x}_i - \bar{p}_i\|.$$

The Two-Dimensional Case: A Block Movement Algorithm

We now describe a more systematic mechanism for achieving a biconnected topology that we believe will reduce D_{total} while allowing efficient execution in low order polynomial time.

Since the removal of a cutvertex breaks a connected graph G into two or more connected components, the basic rationale behind the algorithm presented in this section is to consciously remove *all* the cutvertices from the network by moving certain robot nodes appropriately to new locations. Note that the contraction algorithm presented earlier does not attempt to remove the cutvertices systematically.

In Fig. 2a the biconnected components of a graph are identified along with its cutvertices. In Fig. 2b a corresponding graph whose vertices are biconnected components (or *blocks*) and cutvertices of the original graph is depicted. Such a graph is referred to as a *block graph* [9]. A block graph has the following properties:

P1 A block can have between 0 and N nodes⁴ (both inclusive). If two cutvertices are connected by a *bridge*, the cor-

⁴ If the original graph has no cutvertices, it is already biconnected, and its block graph consists of only one node that contains all N vertices.

```

1: Given:  $G$ 
2:  $G_{orig} \leftarrow G$ ; /* Save a copy of the original graph */
3:  $BT \leftarrow \text{COMPUTE\_BICONNECTED\_COMPONENTS}(G)$ ; /* Identify Blocks and Cutvertices in  $G$  */
4: while ( $\text{NUMBER\_OF\_NODES}(BT) > 1$ ) do
5:    $\text{MARKROOTBLOCK}(BT)$ ; /* Select ROOT block with maximum number of nodes */
6:    $\text{MARKOTHERBLOCKS}(BT)$ ; /* Mark LEAF, INTERMEDIATE blocks and parents of each block */
7:    $\text{MOVE\_LEAF\_BLOCKS}(G, BT)$ ; /* Algorithm 2 for translating leaf blocks towards their parents */
8:    $BT \leftarrow \text{nil}$ ; /* Reset the Block Tree variable */
9:    $G \leftarrow \text{RECALCULATE\_EDGES}(G)$ ; /* Calculate new connectivity after translation */
10:   $BT \leftarrow \text{COMPUTE\_BICONNECTED\_COMPONENTS}(G)$ ; /* Recalculate Blocks and Cutvertices */
11: end while
12:  $G$  is biconnected now;
13:  $D_{total} \leftarrow \text{CALCULATE\_DISTANCE\_MOVED}(G_{orig}, G)$ ;

```

■ Algorithm 1. *MAKEBICONNECTED*(G).

responding block contains no nodes. Block B_3 in Fig. 2 illustrates this point.

P2 A block graph is a bipartite graph. The two classes of the bipartite graph are cutvertices and blocks. No two cutvertices can be adjacent in the block graph, nor can two blocks.

P3 A block graph is a *tree*.⁵

P4 A block tree of a graph G can be computed in linear ($O(|V| + |E|)$) time. This can be achieved during a depth first traversal/search (DFS) of G in the same pass [10].

While executing DFS on an undirected graph, we start at an arbitrarily chosen node that becomes the root. We keep traversing fresh edges and mark nodes as *visited*; on the way we keep pushing nodes into a stack data structure. This process is continued until we reach a node that is only connected to already visited nodes. At this point we keep backtracking up to a vertex that has edges connecting them to nodes hitherto not visited. With a little thought it can be seen that such a node will always be a cutvertex of the graph. Alongside the identification of the cutvertex, it is easy to pop the downstream nodes from the stack into a set that corresponds to a biconnected component or block. Since the above steps can be executed during DFS in the same pass, identification of cutvertices and blocks takes only linear time.

A Heuristic Algorithm for Translation of Blocks — After a brief introduction to the algorithm for the identification of blocks and cutvertices in a graph G , we present an algorithm, *MAKEBICONNECTED*, for computing new positions for certain nodes; it systematically collapses all blocks into a single one and thus makes G biconnected.

As described earlier, every node receives LSUs from other nodes in the network and extracts their position information from these LSUs. Additionally, neighbor information of a node is also extracted from an LSU in order to construct a view of the current network topology. Although in a perfect world knowing the locations and the transmission range of each radio is enough to construct a view of the network topology, in the real world one actually needs neighbor information from every node to construct a view. Since LSU packets contain that information anyway, no extra overhead is registered. After constructing a full view of the topology, each node independently computes the block tree BT of the topology graph G . Note that every node should have the same view of the topology; hence, internal representation of G should be consistent so that DFS produces the same block tree at every

node. This can easily be achieved by canonical insertion of nodes and edges ordered by node IDs into G at every node.

A salient property of a block is that it is a connected subgraph of G . Hence, if all nodes in a block are *translated* together using the same translation vector, distances between all pairs of nodes in that block remain intact, and hence there is no change in connectivity inside that block. On the other hand, if only *some* nodes in a block are translated, it may result in a change of connectivity within the block. Because of this fact we advocate collective translation of all nodes in a block whenever needed so that the connectivity within the block is preserved while progress is made toward increasing the connectivity of the whole network by translating the block itself. The suboptimality of our scheme stems from this fact as it may not be necessary to move all nodes in a block to achieve biconnectivity. However, this results in a faster algorithm.

Which Blocks to Move and Where? — Suppose block B_k has edges with two cutvertices c_u and c_v in BT . Let B_m and B_n be two blocks connected to c_u and c_v , respectively. Now, since we want to minimize the D_{total} metric, we should move nodes as little as possible. If we translate all nodes in B_k toward B_m , c_u may cease to be a cutvertex, but nodes in B_n may become disconnected from G as the link between B_k and c_v may be broken. Hence, we may not have made any progress toward reducing the number of cutvertices in G after this translation step. To prevent the above from happening, we only translate blocks in BT that have degree 1 toward their respective parent blocks. In order to heuristically minimize the total distance moved, we appoint the block with the maximum number of nodes as the root of BT , and identify all other blocks with degree 1 as leaf nodes.

Algorithm 1 shows the steps to make a robot network biconnected systematically. In every iteration of our algorithm we attempt to remove a number of cutvertices from BT . In the context of Fig. 1, block B_5 will be the root block in BT ; B_1 , B_2 , and B_4 will be leaf blocks. All nodes in B_4 will translate toward a suitable node in B_5 since the latter is the parent (block) of B_4 (lines 6–8 in Algorithm 2). B_1 and B_2 , on the other hand, have an empty parent block B_3 ; hence, all nodes in the respective blocks will translate toward the parent cutvertex of the parent block (i.e., c_2 , lines 9–11 in Algorithm 2).

Every block is translated toward the nearest node in the parent block, whenever applicable, by enough distance such that *exactly one* new edge appears between the current and parent blocks; this causes the cutvertex between the two blocks to vanish. Hence, in one iteration of the algorithm several cutvertices are removed. The time complexity of finding the nearest node as a target edge partner requires B^2 comparisons if B is the average number of nodes in a block. Since B

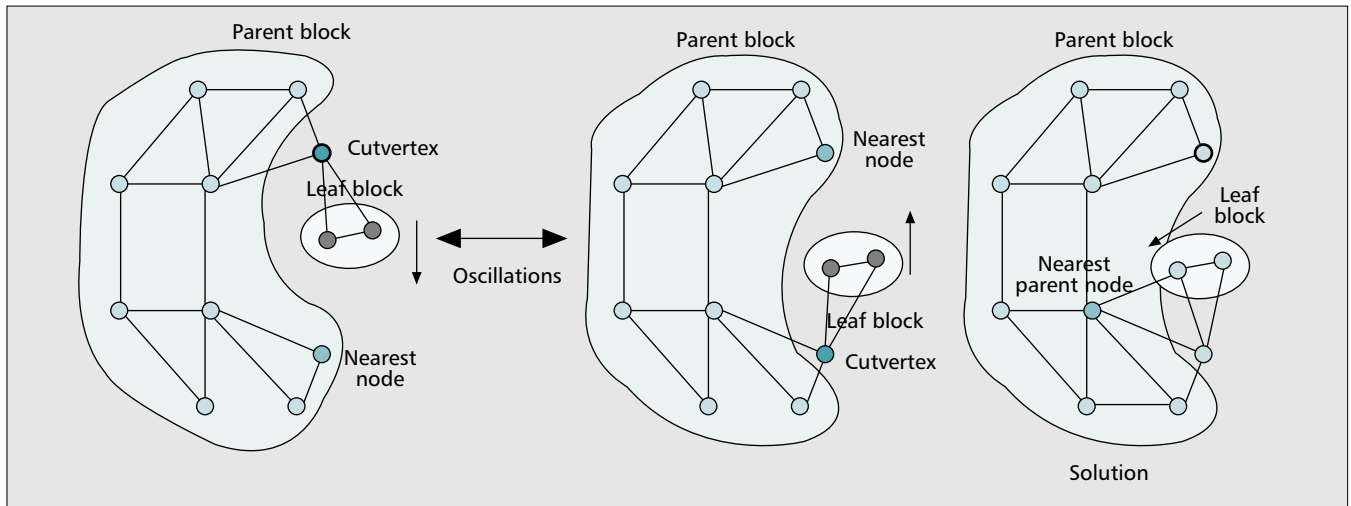
⁵ This can be proved easily. Since the block graph is bipartite, it cannot have an odd cycle. The presence of an even cycle would mean that some two blocks are connected via two different cutvertices; in that case one of the two cutvertices can be safely removed without disconnecting the graph. Thus, we arrive at a contradiction, and a block graph is a tree.

```

1: Given:  $G, BT$ 
2: for all nodes  $blk \in BT$  do
3:   if ( $blk$  is a BLOCK node and a LEAF) then
4:      $par_{cv} \leftarrow BT.parent[blk]$ ;           /* Parent cutvertex of block  $blk$  */
5:      $par_{blk} \leftarrow BT.parent[par_{cv}]$ ;     /* Parent BLOCK of block  $blk$  */
6:     if ( $NUMBER\_OF\_NODES(BT[par_{blk}]) > 0$ ) then
7:       /* Find a node in  $par_{blk}$  that has the shortest distance from any node in  $blk$  */
8:        $nearest \leftarrow FIND\_NEAREST\_NODE(G, blk, par_{blk})$ ;
9:        $TRANSLATE\_BLOCK(BT, blk, nearest)$ ;     /* Translate all nodes in  $blk$  towards  $nearest$  node */
10:    else
11:       $pcv \leftarrow BT.parent[par_{blk}]$ ;       /* If  $par_{blk}$  is empty, then select parent cutvertex of  $par_{blk}$  */
12:       $Translate\_Block(BT, blk, pcv)$ ;         /* Translate all nodes in  $blk$  towards node  $pcv$  */
13:    end if
14:  end if
15: end for

```

■ Algorithm 2. *MOVE_LEAF_BLOCKS*(G, BT).



■ Figure 3. An exception to the block movement scheme and solution.

$= O(|V|)$ in the worst case, the *Find_Nearest_Node* function has a worst case time complexity of $O(|V|^2)$.

For large networks, several iterations may be needed to remove *layers* of cutvertices before only one block remains. Also, after every iteration, as the number of blocks increases, the blocks grow in size. Hence, a small translation by a large block may contribute a significant amount to D_{total} . Thus, we always recalculate the root (the one containing maximum number of nodes), leaf, and intermediate blocks after every iteration in order to minimize the total movement.

In the worst case, we can have $O(|V|)$ iterations of the while loop in Algorithm 1 before achieving a biconnected configuration (e.g., in a line graph). However, since the number of iterations is bounded, convergence is guaranteed in almost all situations except in very special cases as depicted in Fig. 3. We solve this special case by translating the block toward the nearest node that is a direct parent of the cutvertex (as depicted in the rightmost subfigure in Fig. 3). Although doing this repeatedly also guarantees convergence, we follow this tactic only when translation toward a nearest node in the parent block *does not* remove a cutvertex. This is because the aforementioned scheme, if applied iteratively, is likely to take many more iterations before achieving biconnectivity for the whole network.

Schemes for the Actual Movement of Robots — Note that there can be two different schemes for the actual movement of the robots:

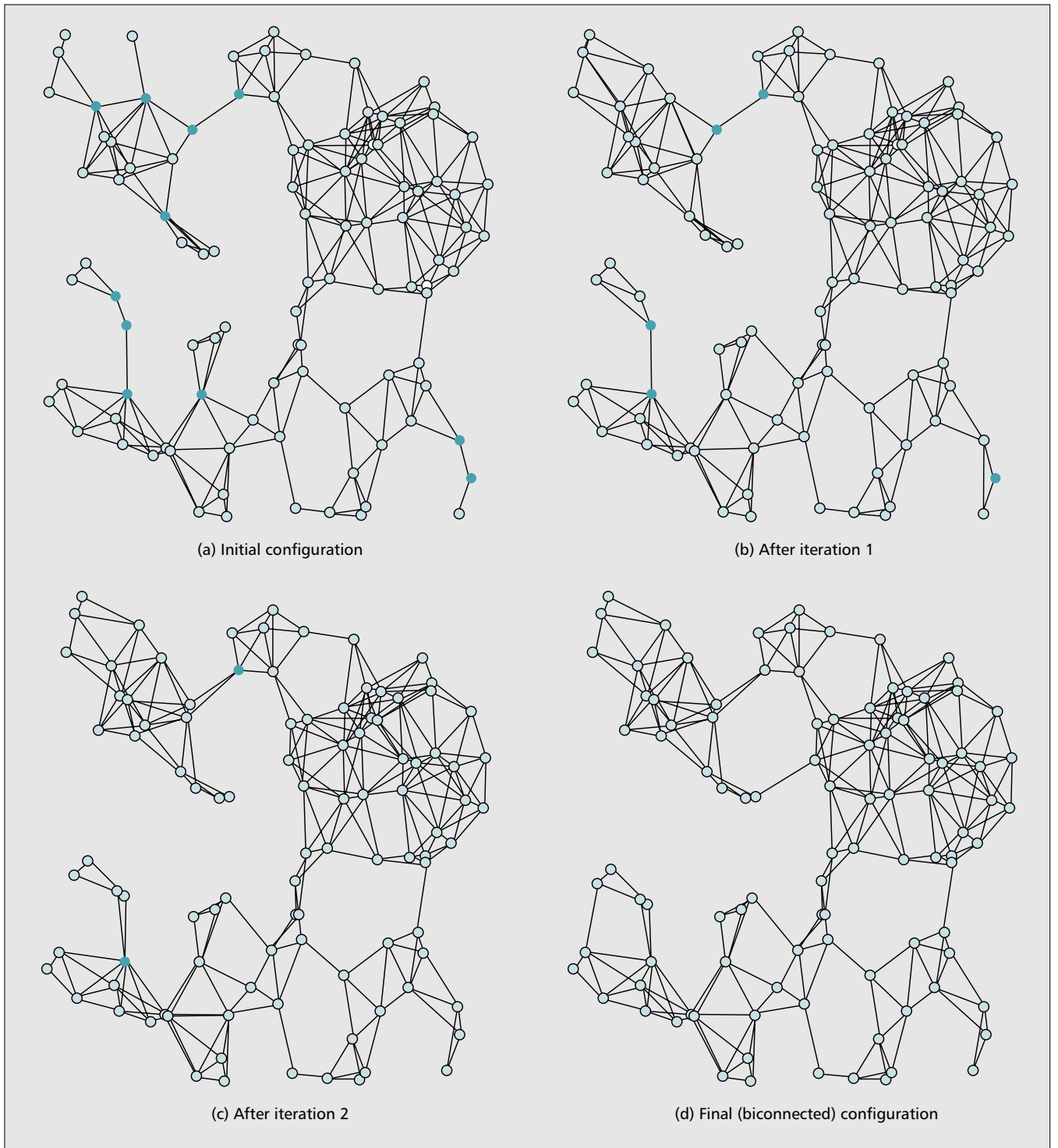
- The robots start moving as soon as a single iteration is over.
- No robot node actually starts moving until their final positions have been determined, that is, after convergence of Algorithm *MAKE_BICONNECTED*(G).

Since the convergence occurs rapidly even for large networks, we adopt the latter scheme which is better since it may result in a much lower value of D_{total} due to the vector addition of translation vectors for every node over all iterations of the algorithm.

Figure 4 depicts a complete execution of the *MAKE_BICONNECTED* algorithm on a randomly selected initial topology. The dark points represent cutvertices in the network. We can observe several steps of the algorithm in action as the graph becomes biconnected after only three iterations.

More Intelligent Block Movement Schemes — Algorithm *MAKE_BICONNECTED* attempts to translate leaf blocks only towards their parent blocks or cutvertices in order to remove cutvertices. However, it is easy to conceive of intelligent schemes in which a slight movement toward a nonparent block may cause removal of several cutvertices in a single iteration. Such schemes can reduce D_{total} as well as the number of iterations in suitable scenarios.

Another possible improvement to the *MOVE_LEAF_BLOCKS* algorithm not investigated in this article is the following. It may not be necessary to translate all nodes in a leaf block toward its parent block. For small leaf blocks it may be simple to calculate optimal translation patterns to achieve biconnec-



■ Figure 4. Execution of the block movement algorithm: a) initial configuration; b) after iteration 1; c) after iteration 2; d) final (biconnected) configuration.

tivity with the parent block, although it may be harder to do so when the leaf blocks grow in size. Both schemes outlined above are likely to be computationally intensive and are subjects for future investigation.

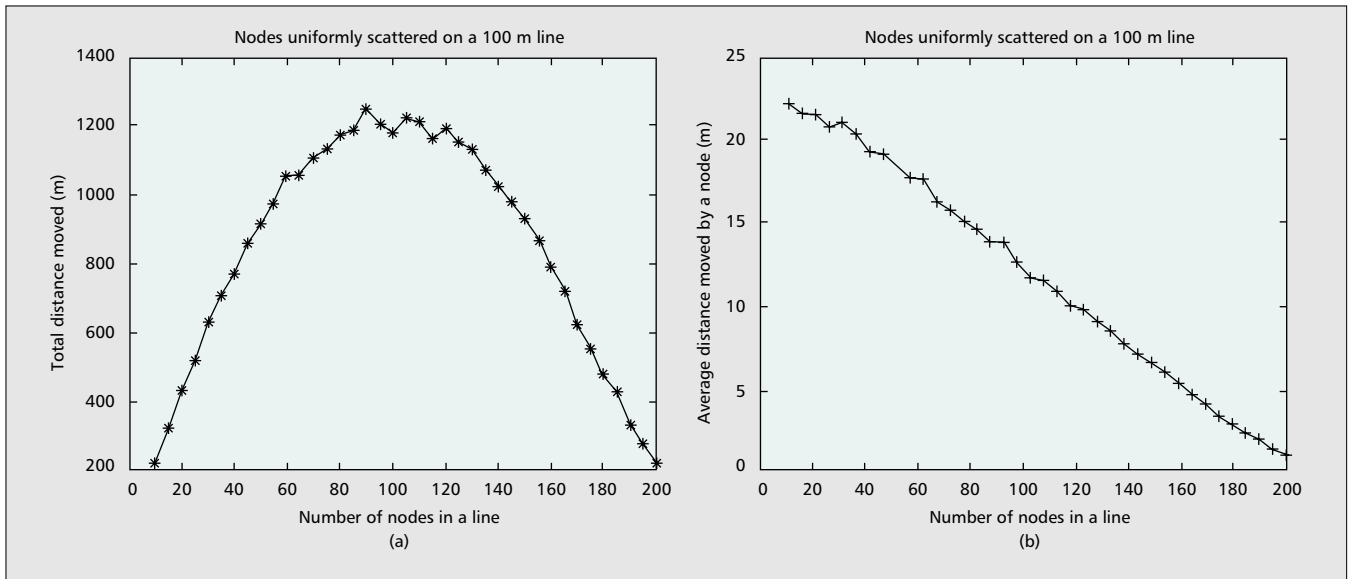
Performance Evaluation

Analytical Results for 1D Networks

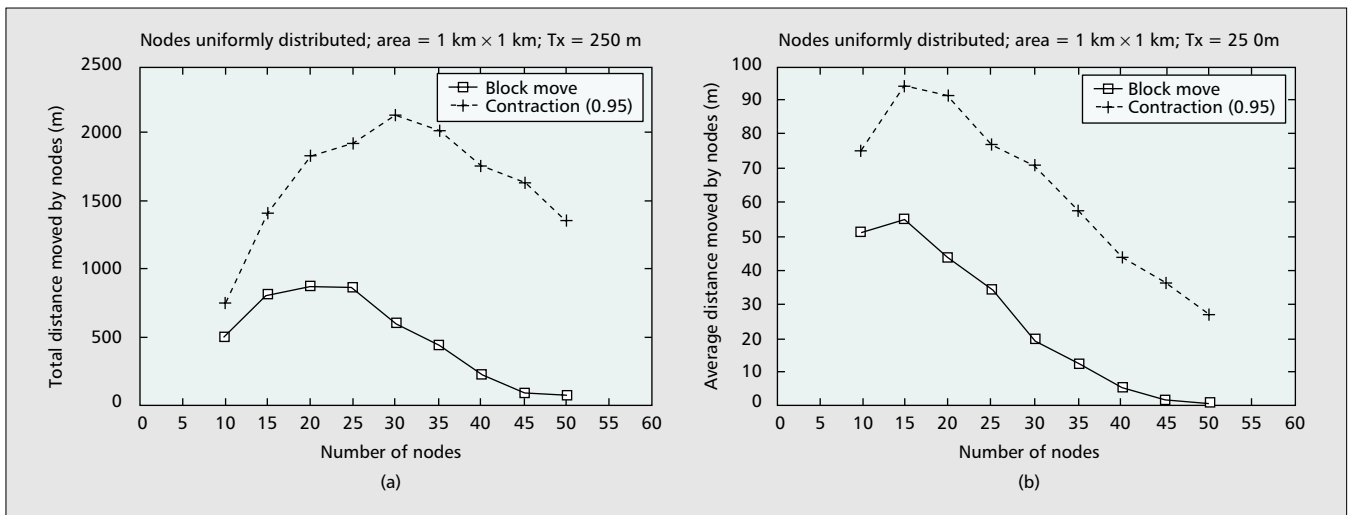
In this section we present results for the 1D version of the problem. We consider up to 200 collinear nodes, each with transmission range of 1.0 and starting positions randomly cho-

sen from $[0,100] \subset \mathbb{R}$. We formulate and solve an LP as described earlier. We measure the total and average distance moved by nodes as N is varied. The results are presented in Fig. 5. We observe that D_{total} increases and then decreases in a parabolic fashion as N is increased. This is because for low values of N , although the initial distance between nodes is large (due to uniform random distribution on $[0,100]$), there are only a small number of nodes that contribute to D_{total} .

For large $N \sim 200$ the network is very dense, and most of it is already biconnected, so D_{total} is low. The peaks are observed for values of $N \sim 100$ because the network is large and frag-



■ Figure 5. Distance moved by nodes (1D networks): a) D_{total} moved by nodes; b) D_{avg} moved by a node.



■ Figure 6. Distance moved by nodes (2D networks): a) D_{total} moved by nodes; b) D_{avg} moved by a node.

mented, and many nodes have to move significant distances to achieve biconnectivity. We observe from Fig. 5b that the average distance moved by a node decreases linearly with increase in N . This is due to the reason mentioned before.

Simulation Results for 2D Networks

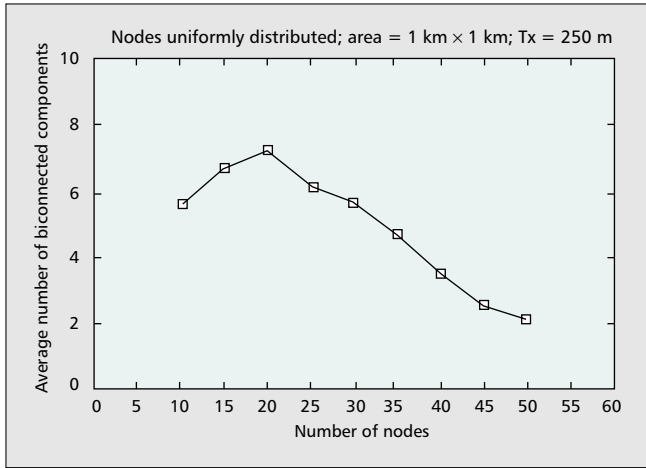
In this section, we report results of simulation of the execution of our biconnectivity algorithms on random 2D topologies. We simulated a 1 km × 1 km square area with up to 50 robots randomly distributed therein. All robots were assumed to have omnidirectional radios with transmission ranges of 250 m each. The ground is assumed to be flat and devoid of obstacles and trenches, thus allowing the robots to move anywhere they want. Harder versions of the problem that include obstacles and imperfect radio propagation are beyond the current scope of this work and are left for future investigation. The initial random configuration of robots obeys the uniform probability distribution while keeping the network connected.⁶ We simulated 100 runs for every data point with the same

⁶ We continue generation of random topologies until a connected topology is found.

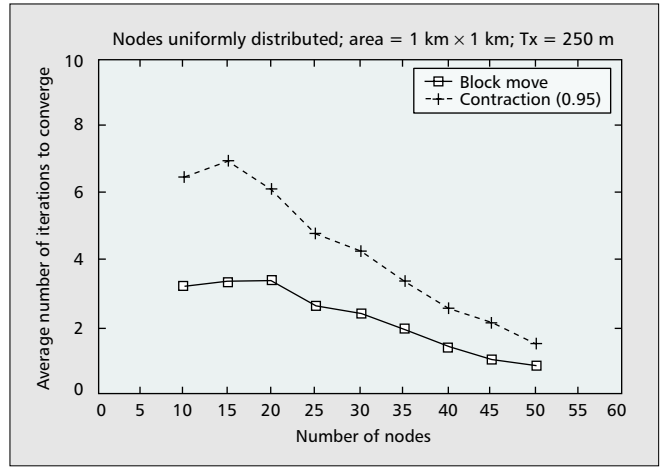
parameters.

Figure 6 compares the performance of the block movement algorithm against the baseline contraction algorithm with respect to the D_{total} metric while varying the number of nodes (and hence the density) in the network. We use $\alpha = 0.95$ in these simulations to restrict any significant extra movement for the robot network. It can be observed that the block movement algorithm completely outperforms the contraction algorithm for all values of N considered. This is because contraction is an ad hoc approach which unnecessarily moves every node towards the center.

We observe that D_{total} increases and then decreases for both algorithms as N is increased from 10 to 50. The reason behind this is similar to the one for the 1-D scenario — for low values of N , there are only a few nodes that can move and also since the topology is connected, the nodes are not very far from each other. This results in a low value of D_{total} . As N increases, more nodes have to move to make the network biconnected, and this increases D_{total} . However, as N is increased beyond a certain threshold, D_{total} begins to drop significantly. This is because higher values of N result in richer, denser topologies that do not have a large number of biconnected components. In such topologies, a slight movement



■ Figure 7. Average number of blocks in G (2D networks).



■ Figure 8. Average number of iterations (2D networks).

results in biconnectivity.

We also plot the D_{avg} metric against N in Fig. 6b. The curves look similar to their counterparts for the D_{total} metric apart from the fact that the peaks are shifted slightly to the left due to division by N . For low values of N , there are only a few blocks (Fig. 7) that are close to each other since the entire network is connected (by construction); hence, D_{avg} is low.⁷ As N is increased, the nodes are more spread out and number of blocks increases; a larger fraction of blocks have to move in order to make the network biconnected. This results in a high value of D_{avg} . For large values of N , however, the number of blocks decreases as the network is richly connected at many places, and only a small fraction of blocks needs to be moved to make the network biconnected. This results in low values of D_{avg} . In fact, for $N = 50$, a node has to move less than 5 m on average while following the block movement algorithm, whereas the contraction algorithm makes nodes move about 30 m each.

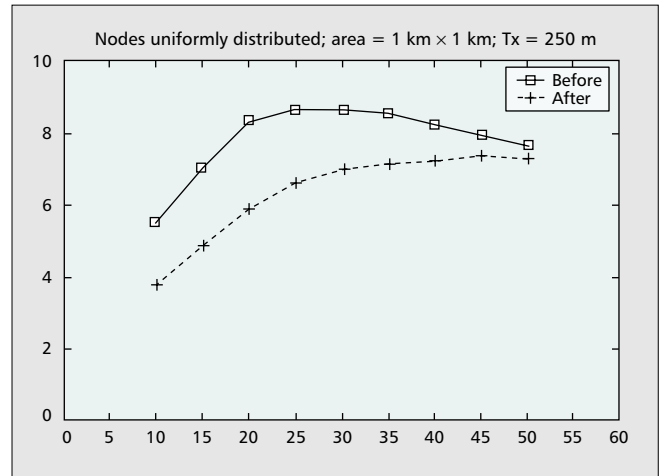
Figure 8 shows the number of iterations needed to achieve biconnectivity. We observe that the block movement scheme requires fewer iterations than the contraction scheme. To be fair to the latter, if the parameter α is lowered to the 0.7–0.8 range, fewer iterations should be required. However, in that case, there is a possibility of contracting the robot network more than necessary. Hence, we chose a high value for α .

Figure 9 illustrates the impact of the block movement algorithm on the diameter of the network, which is defined as the maximum length of a shortest path over all source-destination node pairs. As expected, the diameter shrinks for all values of N as it is a monotonic property, in the sense that it can only decrease when edges are added to the network. However, the gap between the two curves tends to shrink as N increases. This is because there is little perturbation in the network for large N (as illustrated in Fig. 6).

While this simulation study is by no means complete or representative of the real world, it definitely gives us insight into how block movement algorithms work and how they fare against simple contraction schemes. Other metrics we feel are candidates for future investigation are:

- The number of nodes moved by the algorithm
- Fairness in node movement

There can be application scenarios where these metrics are as important as the total/average distance moved metric.



■ Figure 9. The impact of block movement on network diameter: average diameter (hops) vs. N .

Simulations of Distributed Algorithms

We also simulated the proposed algorithms in the OPNET Network Simulator where all the real-world protocols for neighbor discovery, link formation, and link state routing were simulated. The distributed version of the block movement algorithm is trivial to construct from a centralized version when all nodes in the network use proactive link-state routing to share information about the entire topology. Each node then executes a copy the same algorithm and moves to a new location if the algorithm prescribes it to move there. Nodes which are kept static by the algorithm do not move themselves. We also assume that all nodes start execution of the block movement algorithm at the same time after the initial topology discovery phase (by LSU exchange) has completed and the topological views are the same at every node.

We note that the proposed algorithm cannot be used in its current form in highly mobile networks. This is primarily because location information frequently becomes stale in such situations. This could possibly be rectified by depending less on all location information but only on that of nodes in the limited (k -hop) neighborhood.⁸ Another method is to distribute node trajectory information (if known) along with

⁷ This is different from the 1D setting earlier where the initial topology is not necessarily connected. Because of this the D_{avg} curve for the 2D setting is not strictly monotonically decreasing.

⁸ This may result in greater movement by robots and sometimes may not even guarantee biconnectivity.

location updates.

We also note that a high degree of mobility may be in direct conflict with the goal to achieve biconnectivity if the mobility is mission-driven. Therefore, greater coupling is necessary between mission control, motion control, and networking subsystems in every robot before such decisions are made. This is a hard problem and is beyond the scope of this article.

Conclusions

Fault tolerance is an extremely desirable property in network design, and biconnectivity is a baseline feature of fault tolerance. Since the position and movement of nodes in an ad hoc network of robots are algorithmically controllable, greater fault tolerance can be achieved by moving nodes to locations that result in richer topologies. At the same time nodes should move as little distance as possible insofar the desired topological property is achieved. In this article we propose simple algorithms for moving nodes to new locations such that the resulting network becomes biconnected. We show that the problem can be solved in polynomial time for 1D networks by applying LP techniques. For 2D networks, we propose two efficient heuristic algorithms to achieve the goal. We show that our iterative block movement algorithm significantly outperforms the contraction heuristic in the total distance traveled metric.

Future Work

We recognize that due to the seemingly combinatorial nature of the problem space, finding an exact polynomial time algorithm for the 2D case is extremely hard, if possible at all. We are actively pursuing the search for optimal algorithms or approximation algorithms if the problem is NP-complete.

As the network grows in size, the communication overhead of global LSU updates can be prohibitively expensive. Routing protocols like HSLs [4] are used to mitigate this problem, but they can result in different nodes having different views of the network. We plan to extend our heuristics to function with such imperfect knowledge of network topology. Randomized movement algorithms are another potential area of research.

Another important dimension of the problem we plan to investigate in the future is the trade-off between total movement and *robot coverage*, which is defined as the total area covered by the team of robots. Coverage is an important metric since it has direct impact on the mission aspects of the robotic task. Contraction algorithms can result in a significant reduction in coverage and hence may not be suitable. We believe that block movement algorithms can be adapted to include coverage as a parameter in their calculations.

Acknowledgments

We thank the Defense Advanced Research Projects Agency (DARPA) for sponsoring this work. (Project ERNI was supported under contract no. DASG60-02-C-0060.) Additionally, we also thank Ram Ramanathan for sharing with us his insights into the problem domain.

References

- [1] R. Schafer, U.S. Joint Forces Command, <http://www.jfcom.mil/newslink/storyarchive/2003/pa072903.htm>
- [2] A. F. T. Winfield and O. E. Holland, "The Application of Wireless Local Area Technology to the Control of Mobile Robots," *Microprocessors and Microsystems*, series 23/10, 2000, pp. 597–607.
- [3] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol," RFC 3626 (Experimental), Oct. 2003, <http://www.ietf.org/rfc/rfc3626.txt>
- [4] C. Santivanez and R. Ramanathan, "Making Link State Routing Scale for Ad hoc Networks," *Proc. ACM MobiHoc 2001*, Long Beach, CA, Oct. 2001.
- [5] R. Ramanathan and R. Rosales-Hain, "Topology Control of Multihop Wireless Networks using Transmit Power Adjustment," *Proc. IEEE INFOCOM 2000*,

Tel Aviv, Israel, Mar. 2000, pp. 404–13.

- [6] Q. Li and D. Rus, "Sending Messages to Mobile Users in Disconnected Ad Hoc Wireless Networks," *Proc. ACM MobiCom 2000*, Boston, MA, Aug. 2000.
- [7] A. F. T. Winfield, "Distributed Sensing and Data Collection via Broken Ad Hoc Wireless Connected Networks of Mobile Robots," *Distrib. Autonomous Robotic Sys.* 4, L. E. Parker, G. Bekey, and J. Barhen, Eds., Springer, 2000, pp. 273–82.
- [8] P. Basu and J. Redi, "Movement Control Algorithms for Realization of Fault Tolerant Ad Hoc Robot Networks," BBN tech. rep. 8359, Aug. 2002.
- [9] R. Diestel, "Graph Theory," *Graduate Texts in Mathematics*, 173, Springer, 1997.
- [10] R. Sedgewick, *Algorithms*, Addison Wesley, 1984.

Biographies

PRITHWISH BASU [M] (pbasu@bbn.com) is a staff scientist in the Mobile Networking Systems Department at BBN Technologies since 2003. He has been involved in several research projects funded by ARL/DARPA/DoD during his time at BBN. He is currently serving as a key researcher for the ARL CTA program and is also a core contributor to the DARPA Mobile Network MIMO program. He holds Ph.D. (2003) and M.S. (1999) degrees in computer engineering from Boston University and a B.Tech. degree (1996) in computer science and engineering from the Indian Institute of Technology (IIT), New Delhi. His research interests include distributed application modeling for MANETs, routing and clustering in MANETs, energy-efficient protocol design for wireless sensor networks, algorithms for networking robots, and performance evaluation. He is generally interested in any algorithmic issues in MANETs. He has co-authored close to 15 conference and journal articles and two invited book chapters, and has two patents pending. He is a member of the ACM and has served on the organizing committees for MobiQuitous 2004 and ACM MobiCom 2000. He has also served as a reviewer for a number of networking conferences and journals over the past seven years.

JASON REDI [SM] (redi@bbn.com) received a B.S. in computer engineering from Lehigh University, and his M.S. and Ph.D. in computer engineering from Boston University. He is presently a division scientist in the Mobile Networking Systems Department of BBN Technologies. He has led numerous projects involved in the design and implementation of large-scale MANETs. He is currently principal investigator for a DARPA program on networking for teams of autonomous robots, as well as for the ARL Collaborative Technology Alliance on Comms and Networking, focusing on the areas of MAC protocols and energy conservation. He has also been a co-principal investigator of the DARPA/Army Future Combat System (FCS) Communications program for networking protocols utilizing directional antennas. He is the author of over 30 papers and patents in the area of mobile computing and communications, including the best classified paper at MILCOM 2002. He is Editor-in-Chief of *SIGMOBILE Mobile Computing and Communications Review* and on the editorial board of Wiley's *Wireless Communications and Mobile Computing Journal*. He has been on dozens of technical program and conference organizing committees. He is a member of ACM, Tau Beta Pi, and Sigma Xi.