# Supporting Group Communication among Interacting Agents in Wireless Sensor Networks

Jaewon Shin
Electrical Engineering Department
Stanford University
Stanford, CA 94305
Email: jwshin@cs.stanford.edu

Anthony Man–Cho So
Computer Science Department
Stanford University
Stanford, CA 94305
Email: manchoso@cs.stanford.edu

Leonidas Guibas
Computer Science Department
Stanford University
Stanford, CA 94305
Email: guibas@cs.stanford.edu

***Abstract*— Many applications of wireless ad–hoc sensor networks involve collaboration among sensor nodes to achieve a common task. Moreover, such collaboration is often dynamic in nature. For instance, in an multi–object tracking application, sensor nodes that are tracking various moving objects must share information in order to improve the tracking quality. Thus, it is important to have a protocol that maintains group connectivity in such a setting. In this paper, we study the problem of maintaining communication paths among a group of moving agents that have *interacted* with one another. As its solution, we propose a data structure called the *Distributed Collaboration Graph* (DCG), which is a communication graph obtained from the agent trajectories. DCG can be constructed in a purely distributed fashion with very little cost and can be used for group discovery and for multicasting/broadcasting among agents. We also propose a distributed protocol which maintains a *communication tree* among the agents within the DCG while the agents are moving. This allows us to maintain group connectivity and provides the infrastructure for routing among the moving agents.**

## I. Introduction

Many applications of wireless ad–hoc sensor networks (WASN) can be implemented using multiple mobile agents – multiple agents that are roaming around in a WASN while collecting and processing information about large–scale physical phenomena. Eventually, information gathered by the agents need to be reported back to a user through a gateway, and it is much more power–efficient for them to *aggregate* the information than for each agent to send its information individually.

Another, more concrete, application based on mobile agents is multi–object tracking, where each agent is instantiated at object detection and follows the object as it moves. As objects move, some of the corresponding agents could come within communication range and *interact* with one another to share some state information. In fact, for multi–object tracking, it is not guaranteed that each agent will always follow the same object [1]. Thus, agents *have to* share information when they come close to one another, so that uncertainties in the object identities could be resolved later.

The aforementioned applications in a WASN assume an efficient *group communication protocol* among roaming agents. Maintaining group membership and their connectivity while agents are roaming is a very challenging task, especially in large–scale distributed systems like sensor networks. There are two basic requirements for group communication in general: i) member discovery, and ii) a multicasting protocol. Supporting these features would typically require some global infrastructure or hierarchy. However, these are not likely to be available in distributed systems of embedded battery–powered nodes.

In this paper, we focus on designing communication protocols for the so–called *acquaintence group* (AG) [2], which is a set of agents that have *interacted* in the past. Supporting communication within an acquaintence group is critical for multi–object tracking and is also useful for many other WASN applications based on mobile agents. To this end, we propose two distributed data structures and show how to maintain them in an efficient manner when the agents are moving. Specifically, we propose:

- **Distributed Collaboration Graph** (DCG): DCG is a graph obtained from the agent trajectories and can be used for member discovery.
- **Communication Tree on DCG**: In principle, DCG itself can be used as a communication graph by a simple flooding protocol, although it is more desirable to maintain a subtree *spanning* all the agents in the DCG for efficient multicasting.

We develop light–weight distributed protocols that construct and maintain the aforementioned data structures. Moreover, we demonstrate their efficiency through analysis and simulation.

The paper is organized as follows. After briefly discussing related works in Section II, we introduce the Distributed Collaboration Graph (DCG) and discuss a protocol for constructing it in Section III. Then, we present a protocol for dynamically maintaining a communication tree within the DCG in Section IV. Finally, simulation results and conclusions are presented in Sections V and VI.

## II. Related Work

In [2], the authors introduced the notion of *collaboration group*, which is a set of nodes – or processes – that collaborate to achieve a common goal. Most of the collaboration groups mentioned in that paper can be supported by well–known geographic routing protocols like Geocasting [3] or GEAR [4], since nodes in those groups have well–defined geometric or topological relations, e.g. a group of nodes within 1–hop communication distances or a group of nodes within some region $[\Delta X, \Delta Y]$. However, this is not the case for the acquaintence group (AG), as membership is not defined by geometric/topological primitives, but by their logical relationships, e.g. a group of agents that have observed $elephant$ before.

In [5], the authors proposed a general group communication framework in a WASN. The basic idea is to maintain a *moving backbone*, which is a set of nodes used for exchanging messages among agents while they are moving. Their methods, however, require a separate member discovery mechanism for a specific group definition – or else agents could receive unwanted messages.

## III. Distributed Collaboration Graph (DCG)

### A. Acquaintence Group

We first make the following assumptions on the sensor network. All the nodes are stationary and can directly exchange messages only with the nodes within their communication range. In a multi–object tracking scenario, which our paper will focus on, nodes at the boundary of a sensor network detect objects and assign an agent to each object. In this context, an **agent** is a process running on a node. It maintains and updates the information of a single moving object, e.g. its position estimate, signature etc. An agent *follows* an object in a sensor network as the object is moving so that it can maintain the good signal–to–noise ratio and reduce communication cost – staying close to physical sources is always beneficial.

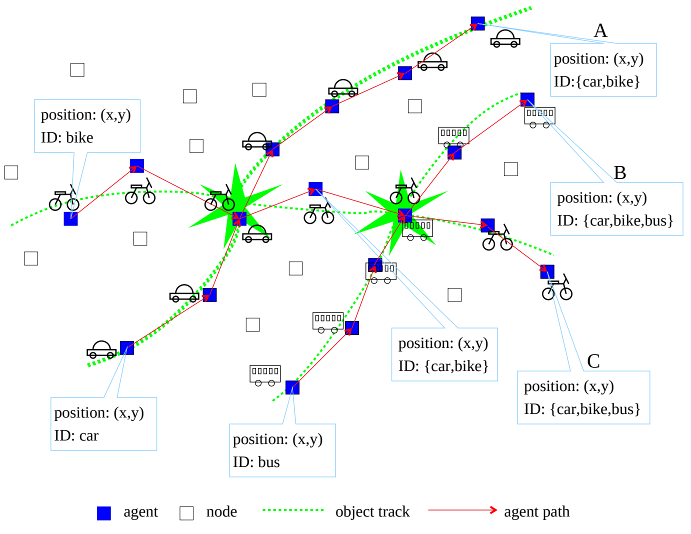


Fig. 1. A simple agent interaction for multi-object tracking application

Figure 1 illustrates a typical scenario with three agents $A$, $B$ and $C$ tracking $car$, $bike$ and $bus$, respectively. Each agent maintains two pieces of information on its target – position and ID. Near the left intersection marked by a star–shape, two objects $car$ and $bike$ are very close to each other, and the two corresponding agents can no longer distinguish the IDs of their objects[1]. Those two agents then communicate with one another to update their ID as the union of the previous IDs $\{car, bike\}$ – this object can be either $car$ or $bike$ – and this process is called *mixing*.[2] Later at the right intersection, $bike$ and $bus$ are *mixed* again and their corresponding agents update their IDs as $\{car, bike, bus\}$ – note that the agent following $bike$ has its ID $\{car, bike\}$ after the first mixing. In general, after a mixing, two agents update their IDs as the union of the two previous identities. Now, we define the notion of an acquaintence group (AG) [2].

*Definition 1:* An *acquaintence group* $AG(I)$ is a group of agents, whose ID set contains $I$.

According to the above definition, $AG(bike) = AG(car) = \{A, B, C\}$ and $AG(bus) = \{B, C\}$ in figure 1. Intuitively, $AG(bike)$ can be interpreted as a group of agents, who share information on $bike$ – no one outside this group knows anything about $bike$.

Now, let's discuss a scenario of group communication within an acquaintence group. Suppose that an agent in $AG(bike)$ observes some local evidence that indicates its object is $bike$. Then, this information needs to be shared with the rest of the members of $AG(bike)$ for consistency – this is called an *identity management* [1]. Note that we do NOT use the names of the agents for this type of communication. The naming of agents (or nodes) for the group communication is based on their data – IDs of physical objects, in this case. Thefore, the communication primitive within an $AG(bus)$ is "I want to hear from everyone who has seen $bus$".

It seems hopeless to support group communication as described above without any global infrastructure on membership management and multicasting protocol. Figure 2, however, indicates that we might be able to use agent paths for the membership discovery and group communication – groups are always formed at the intersections of agents. We will call this graph a distributed collaboration graph (DCG) and the details will be discussed in the next section.

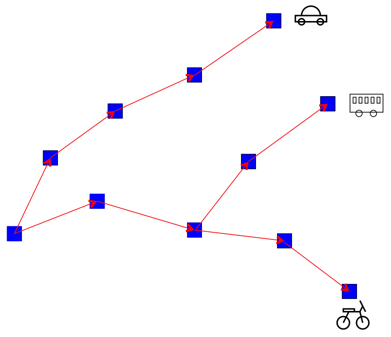


Fig. 2. Idea of using agent paths for member discovery and group communication: The agent paths after the mixing can be used as member discovery and group communication in this example.

[1]This is a data association problem, which is known to be NP–hard.

[2]For the sake of discussion, we consider only pairwise mixings throughout this paper.

## B. Construction of Distributed Collaboration Graph

The construction of DCG can be done in the following way. Each agent that belongs to more than one group[3] must leave a routing table at the node it resides before it handoffs to another node. A routing table contains information on node class, previous node(s), next node(s), group membership and current time stamp as summarized in Table I.

TABLE I
A ROUTING TABLE IN A NODE OF A COLLABORATION GRAPH

| Information stored at nodes in DCG |
|---|
| Node class: $isJunction$ |
| Previous node $\#_1$: $prevNode_1$ |
| Previous node $\#_2$: $prevNode_2$ |
| Next node $\#_1$: $nextNode_1$ |
| Next node $\#_2$: $nextNode_2$ |
| Group membership (ID Set): $G$ |
| Current time: $t$ |

There are two main classes of nodes in DCG. One is a *junction node*, which makes some routing decision between four (three) nodes – two (one) input nodes and two output nodes, and the other is a *relay node*, which just relays packets between two nodes. Junction nodes are formed at mixings; two agents involved in a mixing select a node within the intersection of the two communication ranges as a junction node. There is one special junction, which has no previous node IDs and called a *root*. All the current agent nodes are called *terminals*. Information stored at nodes can be visualized as Figure 3

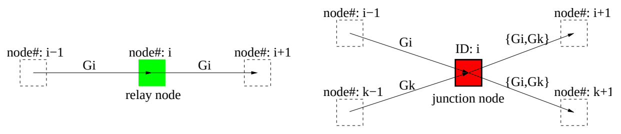


Fig. 3. Information stored at relay nodes (left) and junction nodes (right)

Let's take an example of five agents with five objects in a WASN and visualize the DCGs of each group. Figure 5

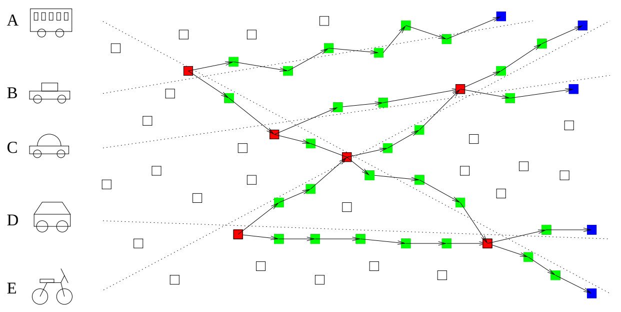


Fig. 4. Mobile agent paths

shows five collaboration graphs corresponding to five different object identities. We emphasize that these graphs are just for visualization and no single node contains global information on a DCG. For example, the left DCG in Figure 5, denoted as $DCG(bike)$, is a collection of nodes (and edges), whose routing table(s) contain $bike$ as group membership. It is easy to see why group member discovery is automatic given a DCG – the terminals of $DCG(bike)$ are precisely the members of $AG(bike)$, and vice versa.

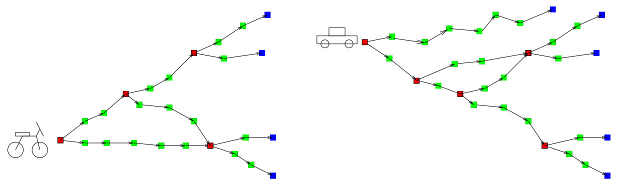


Fig. 5. Distributed collaboration graphs

*Proposition 1:* Discovery of all the members of $AG(ID)$ can be done by simply visiting all the terminals in $DCG(ID)$.

Although simple *flooding* on a DCG can be used as a group communication protocol, it is not efficient, as packets will visit all the nodes in the DCG. In the next section, we will present a protocol for maintaining a *communication tree* among the terminals (a subgraph of a DCG) in a distributed fashion.

## IV. MAINTAINING COMMUNICATION TREE ON DCG

We will now show how to maintain the DCGs and communication trees for each of the acquaintence groups as the objects move. As we shall see, a major feature of our protocol is that all the computations can be done *in a distributed and local manner*. Informally, the DCG for object $i$, which we shall denote by $DCG(i)$, keeps track of the positions as well as trajectories of those agents whose ID sets contain $i$. In other words, $DCG(i)$ can be viewed as the acquaintence group $AG(i)$ augmented with trajectory information. The communication tree for $AG(i)$ is a Steiner tree in $DCG(i)$ that spans the terminals. In particular, we will use this tree as a data structure for supporting group communication. We call an edge in the DCG *active* if it belongs to the communication tree.

Initially, there has been no interaction among the agents. Hence, the DCG for each object is empty. As the objects and their tracking agents move around in the network, we would have to update the corresponding DCGs. The updating of the DCGs is triggered by three types of events – *mixing* event, *crossing* event, and *relay* event. We now describe each of these in turn.

### A. Mixing Event

Suppose that two agents, one carrying ID set $I_1$ and the other carrying ID set $I_2$, come to close proximity of each other at the point $p$ (we shall assume that $p$ is a node in the network). We call such an event a *mixing* event. As mentioned before, we need to update the ID sets carried by the agents.

[3]This is equivalent to agents, of which objects have multiple identities.

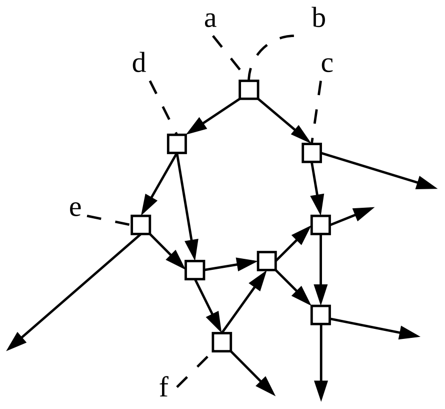


Fig. 6. A multi–target tracking scenario. In the above figure, objects are labelled $a, b, c, d, e$. The arrows indicate the trajectories of objects. Each square represents a mixing event (see Section IV-A).

Moreover, we need to update $DCG(i)$ and its communication tree for each $i \in I$ to reflect the mixing. We now describe each of the updating tasks in turn.

*1) Updating the Identity Sets:* As mentioned in Section III-A, the agents in the two outgoing edges will carry the ID set $I = I_1 \cup I_2$. Clearly, this update step can be carried out in a local manner.

*2) Updating the DCG and the Communication Tree:* For simplicity of discussion, let us fix our attention to $DCG(i)$, where $i \in I$. To update $DCG(i)$, we first declare the crossing point $p$ a *junction node* for $i$. As the agents hop to the next nodes, we declare both of the outgoing edges *active* for $i$. It is important to note that an edge may belong to many DCGs, and hence we need to specify to which group the edge is active. Now, since two agents are meeting at the common point $p$, the active edges in the updated $DCG(i)$ may contain a cycle. However, note that if $i \notin I_1 \cap I_2$, then the active edges in the updated $DCG(i)$ would not contain a cycle (see Figure 7). On the other hand, if $i \in I_1 \cap I_2$, then the active edges in $DCG(i)$ will form a cycle after the update, since both agents have information about $i$ iff they have *interacted* earlier. To remedy this situation, we first declare one of the active cycle–edges, say $e$, inactive. Then, starting from the edge $e$, we traverse around the cycle and check whether there are *dangling* active edges. An active edge is said to be *dangling* if there are no incident active edges on either one of its endpoints, *and* none of its endpoints are terminals. We would like to deactivate as many dangling edges as possible, since their deactivation should not affect the connectivity of the resulting communication tree. Indeed, we have the following proposition.

*Proposition 2:* Suppose that initially the active edges in $DCG(i)$ form a tree that spans the terminals. If a mixing event creates a cycle of active edges in $DCG(i)$, then upon applying the cycle–cancelling algorithm described above, the resulting active edges will still form a tree that spans the terminals. In particular, the subgraph induced by the active edges at the end of the algorithm is connected.

*Proof:* The first step of the algorithm declares one of the active cycle-edges inactive, and this step clearly preserves connectivity and the spanning property. Moreover, the active edges form a tree after this step. Now, observe that a dangling edge is an edge that leads to a non–terminal leaf node in this tree. Hence, its removal would neither destroy the connectivity nor the spanning property. ■

We remark that an optimal set of edges to delete from the cycle (i.e. a set that yields that maximum number of edges) while preserving connectivity can be found in $O(l^2)$ time, where $l$ is the length of the cycle. Moreover, the graph induced by the remaining active edges is a tree that spans the terminals. Thus, it follows by induction on the number of update steps that our scheme has the desired properties.

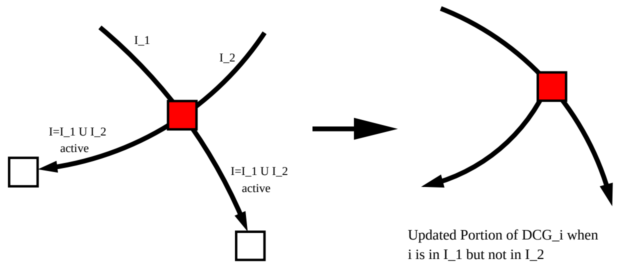


Fig. 7. Mixing Event: The updated tree does not contain a cycle.

To perform the cycle–cancelling operation, we need only information stored in the nodes along the cycle. Thus, all the above computations can be done in a distributed and local manner. To illustrate our protocol, consider the example in Figure 6. The various communication trees for object $a$, as time evolves, are shown in Figure 8.

## *B. Crossing Event*

Observe that not every intersection of the trajectories corresponds to a mixing event. It could be the case that one agent arrives at $p$ at an earlier time than the other, but that both agents carry information about object $i$. In this case, we would still need to update $DCG(i)$ in order to ensure the correctness of the data structure. We call such event a *crossing* event.

Suppose that one agent, say $x$, reach $p$ earlier. As $x$ hops to the next node, it would leave the set of identities $I_x$ at $p$. Now, when the other agent $y$ reaches $p$, it could check whether $I_x \cap I_y = \emptyset$. If $I \equiv I_x \cap I_y \neq \emptyset$, then we mark $p$ as a junction of $DCG(i)$ for each $i \in I$. Again, let us focus on one particular identity $i \in I$. (The marking of $p$ as a junction is mainly for keeping track of the entire $DCG(i)$.) After marking $p$ as a junction, we need to update the communication tree in $DCG(i)$. To do so, let $t_y$ be the terminal where agent $y$ resides. If $p$ is incident upon an active edge along the trajectory of $x$, then we update the communication tree using the mechanism described in the preceding section (see Figure 9). Otherwise, we simply add the edge $(p, t_y)$ to the

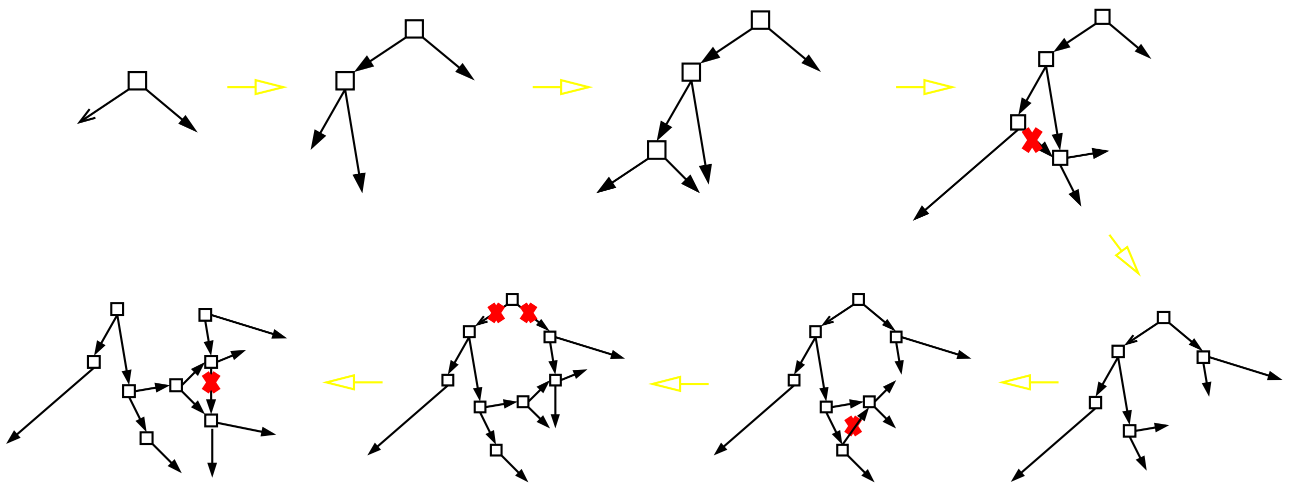


Fig. 8. The various communication trees for object $a$ as time evolves. Edges that are declared inactive are marked with crosses.

communication tree (and hence $(p, t_y)$ is an active edge). By induction on the number of update steps, we see that such a scheme maintains group connectivity, i.e. every object in the same group is connected by the communication tree. Moreover, in order to perform the update, we need only the information stored at $p$. Hence, the update step is completely local.

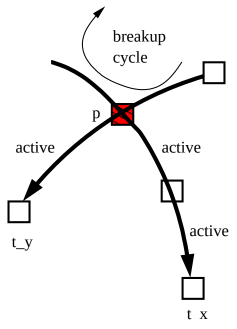


Fig. 9. Crossing Event

## C. Relay Event

As an agent follows an object, it will hop from node to node. If the agent carries an ID set $I$ with more than one element (which indicates that the agent has involved in a mixing event before), then it will leave a routing table at the node it resides (call it $u$) before hopping to another node $v$ (see Section III-B). In this case, we add the node $v$ to $DCG(i)$, where $i \in I$, and declare the edge $(u, v)$ active.

## V. Simulation

To validate the correctness and efficiency of the proposed protocols, we have performed a simulation. Approximately 1200 nodes are placed in $640 \times 640$ region with a communication range of 25 (no unit). For a given number of agents, their directions are randomly chosen, but their velocities are carefully chosen so that there are a few junctions. We mainly focus on verifying the correctness of our protocols, so control packets, link-level acknolwedgements and other low-level parameters are not simulated or ignored for simplicity. Figure 10 shows a typical scenario involving five agents.

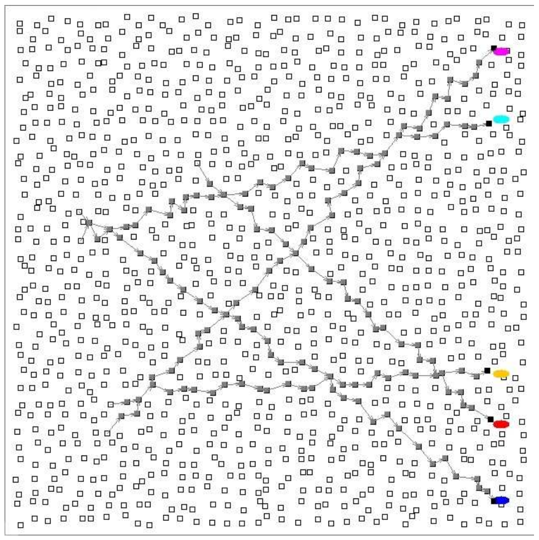


Fig. 10. A typical scenario with five agents

We compare the performances of two group communication protocols – a flooding on a whole DCG and multicasting on a communication tree. For a given set of agent trajectories, we compare the two protocols in terms of the total number of hops from a randomly selected agent to the rest of the agents. For fair comparision, we assume that a flooding scheme maintains a queue at each node so packets do not travel forever. For each number of agents, we generate 50 different random scenarios (trajectories), over which the number of hops are averaged. Figure 11 shows the comparison results; the overhead of maintaining communication trees is validated especially when there are many agents.

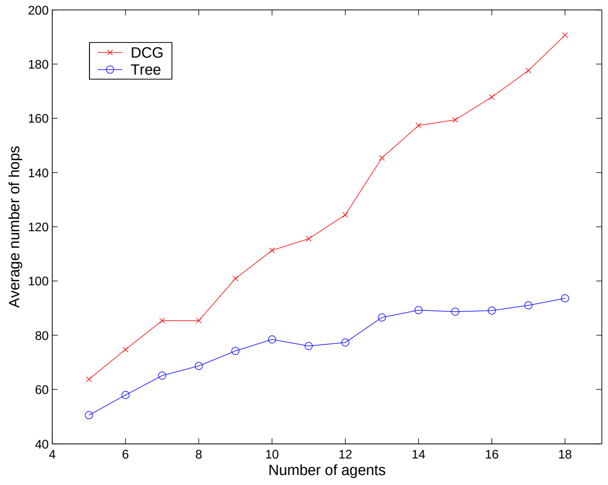
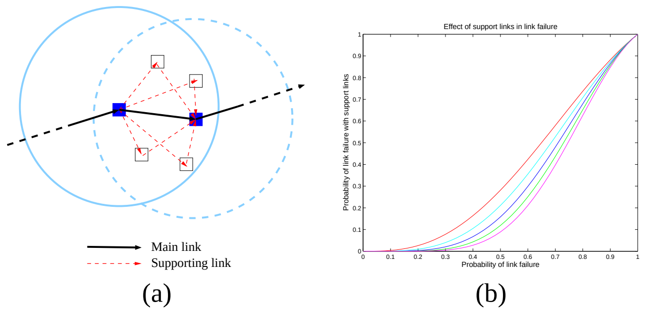


Fig. 11. Group communication performance comparison: Flooding on a whole DCG and multicasting on its spanning tree

## VI. Discussion and Conclusion

We have studied how to support a group communication among interacting agents (acquaintence group) in a WASN and proposed two distributed data structures and two corresponding protocols that support i) group membership discovery and ii) efficient multicasting in a purely distributed manner. Simulation shows the feasibility, correctness and efficiency of the proposed protocols.

There is room for improvement on both theoretical and practical aspects of the current protocols. In the current setting, if one of the nodes/links in the communication graph fails – which can happen quite often in wireless systems – then the whole group communication would be broken. Our preliminary study, however, shows that a simple *local replication* scheme can remedy the situation. Whenever an agent leaves a routing table, the agent also broadcasts the table to its one–hop neighbors, and at node/link failures, the redundant information at the neighboring nodes can be used instead.[4] Figure 12 shows that this heuristic can effectively reduce the probability of link/node failure for different node densities.



Main link
Supporting link
(a)
Effect of support links in link failure
Probability of link failure with support links
Probability of link failure
(b)

Fig. 12. A simple heuristic to deal node/linke failures

Another question we plan to address is the quality of the communication tree (in terms of hop–count or some other metric) produced by our algorithm. Observe that the communication tree is simply a Steiner tree in the DCG that spans the terminals. The problem of computing an optimal Steiner tree in a general graph is NP–hard, and there has been lots of approximation algorithms for this problem, see e.g. [6]. However, none of these algorithms are local, and they do not address the issue of moving terminals. We plan to address these issues in the future using the kinetic data structure (KDS) framework developed in [7]. In particular, it would be interesting to develop a local, distributed approximation algorithm for the Steiner tree problem under the KDS framework.

DCG can be useful for supporting a query on *identities* of objects. Suppose a user is only interested in information about a specific object, say, $bike$ in a WASN, then $DCG(bike)$ can be used as an information aggregation tree – it is not exactly a tree, but can be converted into one using the similar cycle–handling protocol described in this paper.

[4]This is a good example of exploiting *diversity* in wireless communication.

## References

[1] J. Shin, L. Guibas, and F. Zhao, "A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks," in *Proceedings of 2nd International Workshop on Information Processing in Sensor Networks (IPSN 2003)*, April, 2003, pp. 223–238.

[2] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao, "State-centric programming for sensor-actuator network systems," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 50–62, October-December 2003.

[3] J. C. Navas and T. Imielinski, "GeoCast – geographic addressing and routing," in *Mobile Computing and Networking*, 1997, pp. 66–76.

[4] Y. Yu, R. Govindan, and D. Estrin, "Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks," 2001.

[5] Q. Fang, J. Liu, L. Guibas, and F. Zhao, "RoamHBA: Maintaining group connectivity in sensor networks," in *Proceedings of 3nd International Symposium on Information Processing in Sensor Networks (IPSN 2004)*, 2004, submitted.

[6] G. Robins and A. Zelikovsky, "Improved steiner tree approximation in graphs," in *Proceedings of the 11th annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2000)*, 2000, pp. 770–779.

[7] L. J. Guibas, "Kinetic data structures - a state of the art report," in *Proc. Workshop Algorithmic Found. Robot.* A. K. Peters, Wellesley, 1998, pp. 191–209.

[8] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Mobile Computing and Networking*, 1999, pp. 263–270.

[9] B. Krishnamachari, D. Estrin, and S. Wicker, "Modelling data-centric routing in wireless sensor networks," in IEEE INFOCOM.

[10] J. Liu, J. Liu, J. Reich, P. Cheung, and F. Zhao, "Distributed group management for track initiation and maintenance in target localization applications," in *Proceedings of 2nd International Workshop on Information Processing in Sensor Networks (IPSN 2003)*, April, 2003, pp. 113–128.

[11] B. Karp and H. T. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *Proc. 6th Annual ACM Mobile Computing and Networking (MobiCom '00)*, 2000, pp. 243–254.

[12] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A geographic hash table for data-centric storage in sensornets," in *Proc. 1st ACM Workshop on Wireless Sensor Networks ands Applications*, 2002, pp. 78–87.