

Slime Mold Inspired Protocol for Wireless Sensor Networks*

Ke Li¹ Kyle Thomas² Louis F. Rossi³ Chien-Chung Shen¹

¹Computer and Information Sciences ²Chemical Engineering ³Mathematical Sciences
University of Delaware, U. S. A.

Abstract

The phenomenon of self-organization is pervasive in nature, where biological organisms efficiently self-organize unreliable and dynamically changing components to develop a wide diversity of functions. In addition, these biological organisms enjoy the desirable properties of robustness to the failure of individual components, adaptivity to changing conditions, and the lack of reliance on explicit central coordination. In this work, we seek inspiration from the study of the tubular network formation behavior of slime mold to design a localized protocol to connect sensors to sink(s) that balances efficiency and robustness in wireless sensor networks (WSN). Extensive simulations have been conducted to validate the effectiveness of the protocol, as well as the efficiency and robustness of the resulting network connectivity.

1. Introduction

In wireless sensor networks (WSN), information collected by a large number of sensor nodes must be transmitted to the sink node(s). Therefore, given a deployment of sink(s) and sensors, one critical issue is to connect all the sensors to the sink(s). There exist two performance metrics to quantify such connectivity: *efficiency* where data travels from a sensor to a sink in a smaller number of hops, and *robustness* where redundant paths exist between any sensor-sink pair. While centralized schemes could provide optimal solutions, they do not scale for WSN. Many localized solutions have been proposed to address these issues. Fang *et al* developed a landmark-based scheme called GLIDER [3] for efficient point-to-point routing in WSN, where global topology is discovered and partitioned into cluster-like tiles. Greedy routing is used inside tiles based on nodes' local coordinates derived from hop distances to nearby landmarks. However, GLIDER is not applicable to convergecast applications. LEACH [4] and HIT [2] are cluster-based protocols

to address energy-efficiency in data collection applications. All sensors are organized into clusters. Each elected cluster head acquires data from sensors within its own cluster either directly [4] or indirectly [2], performs data aggregation, and transmits fused signal directly to the sink, which allows most nodes to transmit in small distances and to reduce the amount of data sent in the network to save battery energy. Directed diffusion [6] is another application-aware paradigm that achieves efficiency by data aggregation at intermediate nodes. Sink first propagates *interests* (data requests) throughout the network, establishing *gradients* to "draw" events. Data sources then send event data back via empirically good and reinforced reverse paths. Although multiple paths could be taken to achieve robustness, there is no easy ways to balance efficiency and robustness as two contradictory requirements.

Through natural selection, biological systems achieve a balance between efficiency and robustness via simple self-organization without any global coordination or information. One such system is the nutrient distribution network generated by the slime mold *Physarum polycephalum*. In this paper, we mathematically model the formation of nutrient distribution network in slime mold using *Singular Potentials*, and then adapt the model to design a localized protocol to connect sensors to sink(s) in WSN that takes both efficiency and robustness into account.

Physarum polycephalum is a slime mold which, during the plasmodium phase of its life cycle, consists of a large multinuclear cell body that can span tens of centimeters in length. During the course of this phase, the organism will consume organic matter to sustain itself. Nutrients are transported over short distances through diffusion and over large distances through self-assembled channels or tubes that form a dynamic, hierarchical network throughout the cell body [12]. The network forms an essential component of the *pseudopodia*, extensions of the cell that seek new food sources. During the plasmodium phase, the slime mold will assemble and disassemble these tubes to link new food sources into its network and sustain itself by distributing nutrients through the organism. Fitness requires that resources be discovered in the environment and distributed efficiently

*This work is supported in part by National Science Foundation under grants CCF-0726556 and CNS-0347460.

throughout the organism, so *physarum polycephalum* is an excellent biological system to study when designing artificial resource distribution networks [1].

Experimental observations have shown that slime mold will connect nutrient sources into a variety of final configurations including Steiner minimal trees, minimum spanning trees and other more redundant structures [7]. Often, the tubes of a slime mold will be arranged in a geometry that balances efficiency (keeping the total tube length short) and robustness (having multiple paths in case of a tube being severed). These observations are entirely consistent with an algorithm lacking complete global information.

A variety of modeling approaches have been applied to understanding slime mold. While it can be said that any living system is complex by necessity, often specific processes are dominated by a small number of mechanisms. Considerable research has focused on observed behavioral manifestations including search for food, avoid danger, and determine the shortest paths through mazes under certain circumstances [7, 8, 9, 10]. The searching behavior of pseudopodia has also been modeled and analyzed [5]. More recently, Tero *et al* have studied a tube flow model using Poiseuille flow coupled to an adaptation term [13].

The remainder of the paper is organized as follows. We first describe a mathematical model of slime mold's tube network formation behavior using Singular Potential in the next section. The overview of the designed protocol is presented in Section 3. Section 4 explains the protocol in details. Simulation results are then presented and discussed in Section 5. And Section 6 concludes the paper.

2. Mathematical Model

We propose a simple model for connectivity establishment based on slime mold pseudopodia. Chemotactic response, a response directly relative to a local chemical gradient, is common in many biological settings. It is important to understand that there is no centralized control in *physarum polycephalum*. The assembly of the tubelike pseudopodia is a response to local environmental queues. To mimic the dominant behavior of connecting one food source to another, we model the chemical queues as a scalar field using Singular Potentials located at each food source local. A Singular Potential ψ generates a divergence-free ($\nabla \cdot \nabla\psi$) vector field everywhere except at its location. Thus, we are collapsing the food source to a point. A fundamental solution ψ satisfies

$$\nabla^2\psi = \delta(\vec{x} - \vec{x}_i) \quad (1)$$

where \vec{x}_i is the location of the food source and δ is the Dirac δ distribution. Equation (1) is linear, so we can represent multiple food sources as a sum of fundamental solutions. Solutions to this equation are well studied, and in two-dimensional domains, the solutions are logarithmic. Our

model for route discovery is that tubes will self-assemble sequentially following the gradient of ψ , referred to as a *gradient ascent*. Once a food source is added to the network, it is consumed. The consumption of nutrients can be modeled many different ways, but whatever model is used will affect ψ .

A sample of the model and a laboratory experiment are shown in Figure 1. The large scale tube formations could be reproduced using Singular Potentials corresponding to estimates of the nutrient content available at each food source. In these laboratory experiments, *physarum polycephalum* is grown in petri dishes on agar. Nutrient sources of blended oats in gelatin are placed at points of interest on the agar to study slime mold network self-assembly. We found that one CC of nutrient blend would be consumed in two days. We assume nutrients will be consumed linearly over time.

Using this information, we tested the following Physarum Polycephalum Singular Potential model.

1. Initialize the plasmodium at one or more food source.
2. The plasmodium would then stream following a gradient ascent toward a new food source. The field ψ is a superposition of solutions to (1) and so has the form

$$\psi = \sum_{n=1}^N (f_n - \bar{f}) \ln(|\vec{x} - \vec{x}_n|^2), \quad (2)$$

where N is the number of discovered food sources, f_n is the amount of nutrient remaining at the n^{th} food source and \bar{f} is the mean of f_n 's. The initial direction of the search is random, but once this initial step is taken, the path is determined by the gradient of ψ . (Food that had been discovered decays.)

3. When the plasmodium reaches a new food source, the new food source is connected to the network.
4. Return to 2.

This algorithm has a number of attractive properties for our purposes. It uses global information amongst discovered food sources via ψ , but it acquires new food sources incrementally using local interactions. The use of Singular Potentials is connected directly to electrostatic potentials and fluid streamlines. If these nodal values represented charge instead of nutrient content, ψ would present electric field lines. If the nodal values represented fluid sources and sinks with the specified mass fluxes rather than nutrient values, ψ would be the streamfunction and lines of constant ψ would correspond to material paths of the fluid flow.

Our model of biological system is not entirely isomorphic to the wireless model, so we adjust two key features of above algorithm. First, the wireless network must use discrete relay nodes whereas the biological system builds tubes

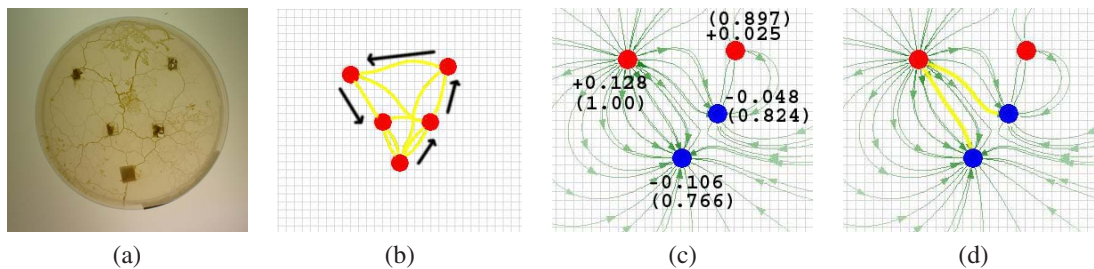


Figure 1. (a) A laboratory slime mold experiment. (b) One possible solution using the Singular Potential model. One of the incremental moves is shown in (c) and (d). In (c), the nodes, their relative nutrient values and the Singular Potential are shown. In (d), new paths are created following the minimum field lines.

in a continuous domain. Second, the wireless nodes have a finite transmission radius whereas diffusive processes in biological systems interact instantaneously over arbitrary distances (though with exponentially weak interactions). To accommodate these changes, we alter Step 2 of the algorithm as follows. To have finite transmission radius, we sum the potential only over nodes within a specified transmission range. To mimic the role of relay nodes, we perform the gradient ascent on a discrete mesh. Rather than ascending in any direction, we proceed to the relay node with position vector most closely aligned with the gradient vector.

3. Protocol Overview

We adapt the above Singular Potential model to develop a localized protocol for WSN to connect sensor nodes with data to the sink. In the biological context, the behavior of slime mold is to create an optimal tube network connecting itself to all food sources. In the context of WSN, the goal of the localized protocol is to establish optimal network connectivity among the data sink and all data sources (sensor nodes with data input). Intuitively, we map the nutrients of food in slime mold to data gathered at sensors in WSN. The biological system can respond locally to a nutrient sources because nutrient sources diffuse efficiently over short distances. The chemotactic response in slime mold can be translated to the ‘data seeking’ mechanism in the localized networking protocol. As a nutrient-seeking slime mold stretches out to reach and consume nutrients, the localized protocol works as the data sink explores forward to connect and unload data sources. At each food source that slime mold has reached, the part of the slime mold body residing there could continue extending out for new food. Similarly in the localized protocol, once a data source is connected, it could also start exploring from itself for other data sources.

The general slime mold algorithm is a centralized scheme where each food’s position is known to others in

advance. This scheme is neither practical nor scales well in WSN with a very large number of nodes. We design a localized protocol where no global information is needed. In the localized protocol, we assume each node knows its own coordinate, since position information itself is the domain variable in the Singular Potential function defined in Equation (2). In addition, neighbor information can be obtained by exchanging control messages locally within a few hops.

Furthermore, the slime mold functions as a continuous system where the organism moves in continuous paths. To mimic the continuous paths in a discrete fashion, we design the localized protocol to work in grid networks, where the continuous agar on which slime mold moves is mapped to the discrete lattice nodes that can mimic each intermediate steps for connections between non-neighbor nodes. The larger the node degree of the lattice, the better granularity obtained in simulating continuous movements. In this paper, we choose the regular grid network with node degree eight¹—the protocol can be easily adapted for other regular grid networks. Each node in the grid network can only directly communicate with its 1-hop grid neighbors.

In this paper, we assume there is only one data sink—the protocol can be extended easily to handle multiple sinks. For simplicity, in the rest of the paper, we use the term ‘sink’ and ‘food’ to refer data sink and data source, respectively.

The core of the localized protocol consists of the mechanism of food exploration by two related control packets: FORWARD REQUEST and BACKWARD REPLY. Only food or sink could be the source or destination of these exploration packets, where the source of a request is called a *requester*, and the destination of the request is called a *replier* who also is the source of its own request packet. A non-food node that receives exploration packets addressed² to itself could only become a *forwarder* by relaying packets to a next hop node based on gradient ascent of the Singular

¹Note that only non-border nodes have eight 1-hop grid neighbors: north, south, west, east, northwest, northeast, southwest, and southeast

²How to choose a next hop will be described in details in Section 4.

Potential model (for FORWARD REQUEST) or source route (for BACKWARD REPLY).

Initially, only the sink is “connected” to the network. Therefore, establishing network connections through food exploration begins with the sink, who, always as a *requester*, initiates the first FORWARD REQUEST packet to a next hop node based on *goodness*, trying to reach a food node nearby. Any food receiving a fresh (unseen before) FORWARD REQUEST addressed to itself immediately sends a BACKWARD REPLY packet to the *requester* via the reverse exploration path, both acknowledging and announcing its successful connection to the network. The connected food then starts its own forward exploration by initiating a FORWARD REQUEST packet, if this is the first time it’s connected³. Any food or sink node already present in the network can always send out a new exploration FORWARD REQUEST as long as its previous request is successfully acknowledged by a corresponding BACKWARD REPLY packet. A *requester* follows a ‘stop-and-wait’ behavior where it stops sending new requests if no reply to its pending request is received. Such behavior serves as the termination mechanism. Finally when all food nodes are connected to the network, no more request packet can be sent out — the protocol simply terminates.

4. Protocol Description

4.1. Local Data Structures

Each node i is associated with a node type ($nodeType_i$), a food value ($foodVal_i$) and a Cartesian coordinate ($xCoord_i, yCoord_i$). The value of $nodeType_i$ could be either *SLIME* (data sink, denoted as *sink*), *FOOD* (data source, *i.e.*, sensor with data input, denoted as *food*), or *OTHER* (neither of the above, denoted as *non-food*). The $foodVal_i$ specifies the amount of data in data source i , which will be reset to zero once node i is connected since data can be offloaded to the sink. Food value is always zero for the data sink, since it’s always connected like a *zero-food*. Food value for non-food nodes is undefined and ignored. Both food values and coordinates of the food or sink nodes are used in the *gradient ascent* of Singular Potential model during food exploration. In addition, the following local data structures are also maintained at each node i :

- $ngTab_i(j)$: represents the *neighbor-grid* table entry corresponding to grid neighbor j of node i , which contains the polar angle of neighbor j in the polar coordinate system with node i as the pole. Note that each node in the network can only directly communicate with its grid neighbors in its *neighbor-grid* table.

³A food node could possibly be connected multiple times, each may from a different *requester*, thus introducing redundant paths as observed in the *slime mold* tubular networks to improve robustness.

$foodId$	$foodVal$	x	y	$hopCount$
----------	-----------	-----	-----	------------

Figure 2. ADVERTISEMENT packet format

- $connTab_i$: denotes the *connection* table that includes node i ’s *parents* (uplinks to the sink) and *children* (downlinks away from the sink) nodes, as the results of the computed network connections through food exploration. Apparently, both *parents* and *children* of node i should be in $ngTab_i$
- $nfTab_i(j)$: denotes the *neighbor-food* table entry which keeps track of neighboring food node j within K hops range from i . K represents the parameter called *FOOD_RANGE*. The *neighbor-food* table entries are of the form $\langle f, d_x, d_y, d_h, t \rangle$, where f is the food value, (d_x, d_y) records the Cartesian coordinate distance of the food from node i , d_h is the food’s distance from node i in hop count, and t denotes the grid neighbor through which as the next hop to reach the food from node i . The *neighbor-food* table is the basis in determining next hops during food exploration, using approaches of either *gradient ascent* or *goodness*.
- $socTab_i(j)$: denotes the *step-one-choice* table entry representing the *goodness* of choosing grid neighbor j as the next hop during food exploration. This table is derived from and synchronized (and hence consistent) with the *neighbor-food* table. In the *step-one-choice* table, a grid neighbor j is a choice of the next hop for node i , retrieved from the field t in node i ’s *neighbor-food* table. The corresponding *goodness* heuristic, as defined in Equation (3), reflects a tradeoff between the amount of food value and the hop distance of all foods that could be reached through j (*i.e.*, use j as the next hop) — the larger food value and less hop distance through j , the better *goodness* of j . Note that the use of this heuristic supplements the method of *gradient ascent* in determining next hops.
- $seqNo_i$: initially set to 0, keeps track of the sequence number of the latest FORWARD REQUEST packet sent out when node i is either a food node or the sink node.

4.2. Neighbor Food Discovery

In order to perform *gradient ascent* of the Singular Potential model, each node needs to know the existence of its neighboring foods within K (*FOOD_RANGE*) hops from itself. This information is obtained by the propagation of an ADVERTISEMENT packet from each food or sink node, who broadcasts its own ADVERTISEMENT packet at a randomly picked time between 0 and *AD_TIMER* after its initialization. The ADVERTISEMENT packet, as shown in Figure 2,

Algorithm 1 Node i processing an ADVERTISEMENT

```
1: Input:
2:  $ad \leftarrow$  incoming ADVERTISEMENT
3:  $prev \leftarrow$  the node from which  $ad$  was received

4: Begin:
5:  $j \leftarrow ad.foodId$ 
6: if  $j \neq i$  AND  $j \notin nftab_i$  then
7:   Increase hop count in the  $ad$  packet:
      $ad.hopCount \leftarrow ad.hopCount + 1$ 
8:   Add an entry of  $j$  to neighbor-food table:
      $nftab_i(j).f \leftarrow ad.foodVal$ 
      $nftab_i(j).d_x \leftarrow ad.x - xCoord_i$ 
      $nftab_i(j).d_y \leftarrow ad.y - yCoord_i$ 
      $nftab_i(j).d_h \leftarrow ad.hopCount$ 
      $nftab_i(j).t \leftarrow prev$ 
9:   Compute a  $goodness_i(m)$ , for node  $m = nftab_i(j).t$ ,
     as the ratio of food value over food's hop distance
     from  $i$  via  $m$ :

     
$$goodness_i(m) \leftarrow \frac{nftab_i(j).f}{nftab_i(j).d_h} \quad (3)$$

10:  Increase goodness in step-one-choice table:
      $socTab_i(m) \leftarrow socTab_i(m) + goodness_i(m)$ 
11:  if  $ad.hopCount < FOOD\_RANGE$  then
12:    Rebroadcast  $ad$ 
13:  end if
14: end if
```

contains the sender's id, food value, and *Cartesian* coordinate, as well as a *hopCount* field initially set to zero to record the number of hops traveled so far by this packet.

Upon receives an ADVERTISEMENT, each node i discards the packet if it has seen an advertisement from the same originator before, since only one such packet is allowed for each food or sink. Algorithm 1 presents the pseudo code of how node i processes an ADVERTISEMENT. If this is a new advertisement, node i increments the packet's *hopCount* field by one, inserts a new entry in its *neighbor-food* table based on the information from the packet, and increases the corresponding *goodness* value in its *step-one-choice* table. The heuristic $goodness_i(m)$ of node m w.r.t node i reflects the desirability of node i choosing node m as the next hop for a FORWARD REQUEST. Defined in Equation (3), *Goodness* is the ratio of food value over food's hop distance from node i via node m . Finally, node i rebroadcasts the ADVERTISEMENT only if the *hopCount* value is less than *FOOD_RANGE*.

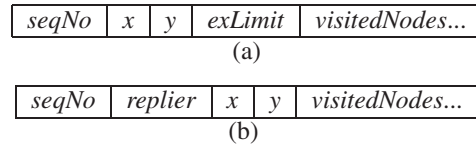


Figure 3. Food explore packet formats: (a) FORWARD REQUEST and (b) BACKWARD REPLY

4.3. Forward Explore Request

At *REQUEST_TIMER* when K -hop neighbor food information has been collected at each node, the sink becomes the first *requester* and initiates a FORWARD REQUEST packet to start network connections establishment via food exploration. Once connected, each food also becomes a *requester*, eligible to send its own request packets. A FORWARD REQUEST initiated by *requester* i , whose format is shown in Figure 3(a), contains the following fields:

- *seqNo*: the sequence number of the FORWARD REQUEST packet, which, together with the requester id i stored as the first entry in *visitedNodes*, uniquely identifies the request.
- x, y : the *Cartesian* coordinate of requester i . Note that the coordinate, together with the implicit food id (requester i) and food value (zero), make the explore packet to also serve as an ADVERTISEMENT packet for food information update.
- *exLimit*: the maximum number of hops the request is allowed to explore before reaching a food node. Initially set to *EXPLORE_LIMIT* it's decremented each time the request travels to the next hop.
- *visitedNodes*: the ordered list of nodes visited by the FORWARD REQUEST, initially set to $[i]$. The first entry of the list is always the originating requester id i , and the last entry is always the next hop id.

A *requester* node i initiating a FORWARD REQUEST invokes the procedure *SendRequest()* described in Algorithm 2 to create the request packet and find a next hop that the packet will explore. Requester i increments its local $seqNo_i$ by 1 as the request's sequence number, fills its coordinate into the packet, sets the request's *exLimit* to *EXPLORE_LIMIT* - 1, and adds itself as the first entry of *visitedNodes* in the packet. The first hop to explore is selected based on the *goodness* heuristic (defined in Equation (3)) stored in the *step-one-choice* table, which gives higher desirabilities to grid neighbors through which higher food values with shorter distances could be reached. The grid neighbor j that has the highest *goodness* value is chosen as

Algorithm 2 Procedure *SendRequest()* executed by node *i*

1: **Begin:**

2: Increase local sequence number for request:

$$seqNo_i \leftarrow seqNo_i + 1$$

3: Create a new FORWARD REQUEST packet *req*:

$$req.seqNo \leftarrow seqNo_i$$

$$req.x \leftarrow xCoord_i$$

$$req.y \leftarrow yCoord_i$$

$$req.exLimit \leftarrow EXPLORE_LIMIT - 1$$

$$req.visitedNodes[0] \leftarrow i$$

4: Determine the first jump based on *goodness*:

$$firstJump = \underset{j \in socTab_i \setminus connTab_i}{\operatorname{argmax}} socTab_i(j) \quad (4)$$

5: Append *firstJump* to *req.visitedNodes*
and send out *req*

long as *j* is neither a parent or a child in *i*'s *connection* table, as shown in Equation (4). The above exclusion encourages exploration in new directions for potentially unreachable foods. Once a next hop is decided, its id is appended to the end of *visitedNodes* and the request is broadcast.

When a node *i* receives a FORWARD REQUEST packet, it updates its local *neighbor-food* table by setting the *requester*'s food value to zero, where *requester* is retrieved from the first entry of *visitedNodes*. To be consistent with *neighbor-food* table, the corresponding *goodness* value is deducted from the *step-one-choice* table as well. Note that node *i* updates the above tables even if the FORWARD REQUEST packet might not be addressed to itself, which takes advantage of broadcast nature of wireless communications to speed up the information propagation.

After that, node *i* checks if it has seen this request before, which is identified by the tuple $\langle requester, seqNo \rangle$, as well as if its id matches the last entry in the request's *visitedNodes* field which designates the next hop for the packet. If this is an old request, or the packet is not addressed to itself, the request is discarded. Otherwise, node *i* knows that this packet is intended to itself and accepts it. Then, node *i* checks if itself is a food node. If so, node *i* realizes that it is the destination of the request packet and becomes a *replier*. The FORWARD REQUEST is then turned into a BACKWARD REPLY packet to return to the *requester*. In the packet format shown in Figure 3(b), the only difference between a reply packet and its corresponding request packet is that the *replier* field replaces the *exLimit* field, and the replier's coordinate is recorded in the packet instead of the requester's. The reply packet uses the same sequence number as the incoming request, thus identified by the same tuple as the request. The field *visitedNodes* is directly copied from the corresponding request with the

last entry removed, so that the reply packet could follow the reverse route back to the requester. The new last entry in *visitedNodes* is then recorded as a *parent* in node *i*'s *connection* table, as well as the next hop of the released reply. If this is the first time food *i* is connected to the network, besides sending back a BACKWARD REPLY packet, node *i* is also allowed to release a FORWARD REQUEST of its own exploring other unreachable food nodes, by invoking the procedure *SendRequest()* shown in Algorithm 2.

If the condition is not satisfied to convert the request to a BACKWARD REPLY packet (*i.e.*, node *i* is not a food node), node *i* is then a potential *forwarder*. It checks if the request's *exLimit* field is positive. If not, the request is discarded. Otherwise, node *i* tries to forward the request. It decrements the packet's *exLimit* by 1, and invokes the procedure *GetNextByGradient(tabu)*, shown in Algorithm 3, to compute a next hop based on *gradient ascent*, where *tabu* is a set of nodes avoided for the choice of the next hop, including all nodes inside node *i*'s *connection* table, as well as nodes that have already been visited by this request packet. The introduction of *tabu* encourages exploration of new directions for potentially unreachable foods and helps raise the efficiency of exploration. Once a valid next hop is chosen, its id is appended to the end of *visitedNodes* and the request packet is rebroadcast. Otherwise when the next hop is *INVALID*, the request packet is discarded.

Algorithm 3 Procedure *GetNextByGradient(tabu)* executed by node *i*

1: **Parameters:**2: *tabu* \leftarrow set of nodes excluded from choice of next hop3: **Begin:**4: Compute the *mean* of all food values in *neighbor-food* table:

$$\bar{f} \leftarrow \operatorname{mean}_{j \in nfTab_i} nfTab_i(j).f$$

5: Compute the *gradient* $\vec{\nabla}(\nabla_x, \nabla_y)$ of potential function defined in (2):

$$\nabla_x = \sum_j \frac{2(nfTab_i(j).f - \bar{f}) \times nfTab_i(j).d_x}{nfTab_i(j).d_x^2 + nfTab_i(j).d_y^2}$$

$$\nabla_y = \sum_j \frac{2(nfTab_i(j).f - \bar{f}) \times nfTab_i(j).d_y}{nfTab_i(j).d_x^2 + nfTab_i(j).d_y^2} \quad (5)$$

6: **if** $\vec{\nabla} \neq \vec{0}$ **then**7: *nextHop* \leftarrow the grid neighbor $k \in ngTab_i \setminus tabu$
such that $\vec{i}k$ has the closest direction as $\vec{\nabla}$ 8: **else**9: *nextHop* \leftarrow *INVALID*10: **end if**11: Return *nextHop*

As mentioned above, a forwarder node i invokes the procedure *GetNextByGradient(tabu)* to compute a next hop for a FORWARD REQUEST. Algorithm 3 shows how the calculation is done using *gradient ascent* of the Singular Potential model defined in Equation (2). The gradient $\vec{\nabla}(\nabla_x, \nabla_y)$ of the potential function at current node i is obtained by calculating the partial derivatives with respect to x-axis (∇_x) and y-axis (∇_y) respectively, as shown in Equation (5). Since $\vec{\nabla}$ in fact denotes the precise direction which promises the maximum increase of the food potential, the next hop is thus selected to be the grid neighbor k so that $i\vec{k}$ has the closest direction as $\vec{\nabla}$, as long as k is not in the *tabu* set. Note that when every food in node i 's *neighbor-food* table is *zero-food*, which means all of them are connected to the network, the above computation reaches a *zero-gradient*, where $\nabla_x = \nabla_y = 0$. In this case, no next hop is available and *INVALID* is returned, which means local food exploration activity at node i is neither possible nor needed.

4.4. Backward Explore Reply

Once a food node is reached by a fresh FORWARD REQUEST, it releases a BACKWARD REPLY via the reverse path back, acknowledging the *requester* as well as announcing to others its connection to the network. The packet format of a BACKWARD REPLY is similar to that of a FORWARD REQUEST, as compared in Figures 3(a) and 3(b).

When node i receives a BACKWARD REPLY, it refreshes local food information by setting the *replier* to *zero-food* in its *neighbor-food* table, as well as deducting the corresponding *goodness* value from its *step-one-choice* table. Note that we have mentioned similar steps of updating these tables when describing how a node processes an ADVERTISEMENT (lines 8–10 of Algorithm 1) and FORWARD REQUEST. This is because both request and reply packets can serve as *zero-food* advertisements.

Node i first checks if the reply is addressed to itself by examining the last entry in the *visitedNodes* field. If its id matches, it adds the previous hop node as a *child* into its *connection* table. Next, if node i is the *requester* that the reply is destined to, it initiates another FORWARD REQUEST packet since its previous request is successfully echoed. Otherwise, node i is a potential forwarder, and then checks if it has previously seen any request packet identified by the same tuple $\langle \text{visitedNodes}[0], \text{seqNo} \rangle$ as the reply packet. If yes, node i removes the last entry of *visitedNodes*, adds the new last entry as a *parent* into its *connection* table, and re-broadcast the BACKWARD REPLY. If not, node i discards the packet since it should have seen the request packet before the corresponding reply coming back. Note that the *connection* table should only be updated during the traveling of a BACKWARD REPLY packet, since a FORWARD REQUEST packet might fail to reach any food node at all. The

Table 1. simulation parameters

<i>FOOD_RANGE</i>	1–10 hop
<i>FOOD_PORTION</i>	0.05–0.99
<i>EXPLORE_LIMIT</i>	15 hop
<i>AD_TIMER</i>	3 sec
<i>REQUEST_TIMER</i>	6 sec

path of the FORWARD REQUEST exploration is volatile, and thus shouldn't be recorded in the *connection* table.

The design of the $\langle \text{request}, \text{reply} \rangle$ pair is essential for the health (validity and efficiency) of the protocol. Each *requester* (food or sink) has at most one 'free' reply to initiate, after which it can only start a new request at the 'cost' of an incoming reply echoing the previous request. Both request and reply packets also serve as *zero-food* advertisements to keep local food information up-to-date. Ideally, when all the food nodes have been connected to the network, no valid next hops for ongoing request packets (if any) are available (as described in Algorithm 3), thus no reply packets coming back — the protocol essentially converges and terminates.⁴ If a design did not include the *reply*, the *requesters* will have no idea when to initiate or stop further *requests*. In such case, the protocol either could not converge assuredly with a 'good' connection, or might introduce heavy control packet overhead.

5. Evaluation

To study the characteristics and evaluate the performance of the described protocol, we have conducted extensive simulation using QualNet [11]. Performance statistics were collected and illustrated under various food range and food portion, shown together with network snapshots. We've also simulated the Singular Potential model described in Section 2 using Matlab to compare with the localized WSN protocol using QualNet with realistic wireless communications.

5.1. Simulation Setup

Each network consisted of 900 nodes placed in grid topology over a terrain of size 870×870 m². Each node was positioned 30 meters away from its horizontal and vertical neighbors, equipped with a radio transceiver capable of transmitting signals up to approximately 43 meters, so that it could only communicate with 1-hop neighbors in horizontal, vertical, and diagonal directions. The underlying wireless channel had a data rate of 2 Mbps, using the two-ray path loss model without fading. IEEE 802.11 DCF was

⁴Since the physical channel in real wireless communications is lossy and error prone, the protocol might terminate before all food nodes are connected to the network. This is because either request or reply packets might be lost in the network.

used as the MAC layer protocol, and IP as the network layer on top of which our protocol was run. No application traffic was employed since the sole purpose was to establish network connections among sink and food nodes. Each network had one sink and a varying portion of food nodes with randomly generated locations and food values. The food portion P , known as the parameter *FOOD_PORTION*, varied from 0.05 to 0.99, denotes the ratio of the number of food nodes over the total number of nodes in the network. The food range K , known as the parameter *FOOD_RANGE* described in Section 4.2, varied from 1 to 10 hops, represents how many hops away a food node can be known to others. Table 1 summarizes the protocol parameters in the simulation.

The following performance metrics were collected. Each measurement was an average of 50 different randomly generated network configurations:

- **Food connection ratio:** indicates the fraction of food nodes that are connected to the network (*i.e.*, reachable to/from the sink). This metric denotes the *effectiveness* of the protocol in terms of connecting food nodes, ranging from 0 to 1. The higher the ratio, the better the *effectiveness*.
- **Number of non-food forwarders per connected food node:** indicates the ratio of the number of non-food forwarders which act as relays between the sink and food nodes, to the number of food nodes connected to the network. This metric denotes the *efficiency* of the resulting network connectivity. The smaller the number, the higher the *efficiency*. A value of zero means the perfect efficiency.
- **Average number of uplinks per connected food node:** indicates the average number of choices of established upstream links a connected food node can select to reach the sink. This metric measures the *robustness* of the resulting network connectivity. The connectivity is robust as long as this metric is greater than one. The higher the number, the better the *robustness*.

5.2. Simulation Results and Discussion

Figure 4(a) presents the snapshot of a resulting connectivity over the 900-node grid, with the food portion of 0.15 and the food range of 3 hops. About 91.2% of the food nodes (shown in black) are connected to the sink which is located at the bottom left corner. Each connected food has 1.352 uplinks to the sink and uses 1.904 non-food forwarders (shown in gray) as relays on average, representing a good balance between robustness and efficiency. One reason to the existence of some unconnected food nodes is that they are too far — more than 3 hops in this case — away

from the connected food network to be reached by means of the Singular Potential model. Increasing food range K can address the issue. In addition, due to the unreliability of wireless communication, either ADVERTISEMENTS initiated by unreached food nodes might get lost during the neighbor food discovery phase, making other nodes even within K hops to be unaware of its existence, or FORWARD REQUESTS might be lost on their way to the food. Although we could mitigate these issues via MAC layer schemes, such as random jitters, network layer schemes, such as passive acknowledgment, or combination of both, these problems cannot be easily solved completely. Moreover, increasing food range K or food portion P can lead to more control packets transmitted spatially and temporally close to each other, which in turn raise the chance of packet loss from collisions. Another anomaly resulting from the unreliable channel is the “island” of intra-connected nodes which are partitioned from the food network, an example shown at the upper right corner of Figure 4(a). The loss of BACKWARD REPLY packets prevents the establishment of the full path between a *requester* within the food network and its *replier* in an “island”. Without the idea of being in an “island”, the *replier*, however, could still send out a FORWARD REQUEST packet to reach other unconnected food node, thus growing the “island”.

Figure 4(b) shows another sample network connectivity snapshot with a smaller food range of one hop and a bigger food portion of 0.5. The food connection ratio in this case is 96%, higher than that of Figure 4(a), with average fewer upstream links of 1.193 (still robust since the value is greater than 1) and zero non-food forwarder (perfect efficiency) per connected food. The better connection ratio comes from the higher food portion, which means more food nodes in neighborhood, making connections of more food nodes easier with shorter exploration paths and resulting in less non-food forwarders as well. However, the perfect efficiency of zero non-food forwarder here is determined by the one hop food range, which means the *neighbor-food* table includes only foods among 1-hop grid neighbors. Therefore, a food node is either reached directly by a neighboring *requester* without any non-food relay, or remain unconnected.

While the snapshots visualize the network connectivity and qualitatively show the *effectiveness*, *efficiency*, and *robustness* of these connections, Figures 5 and 6 present concise and systematic depictions of the three metrics under varying food range K and food portion P quantitatively.

The first experiment varies K from 1 to 10 hops and compares the metrics among $P = 0.1, 0.3, 0.5$ in QualNet (localized WSN protocol) and $P = 0.1$ in Matlab (Singular Potential model). Figure 5(a) shows the metric of food connection ratio. All cases of P increase rapidly within the initial several hops, then level off as K continues going up. This is because a higher K includes more food nodes in the

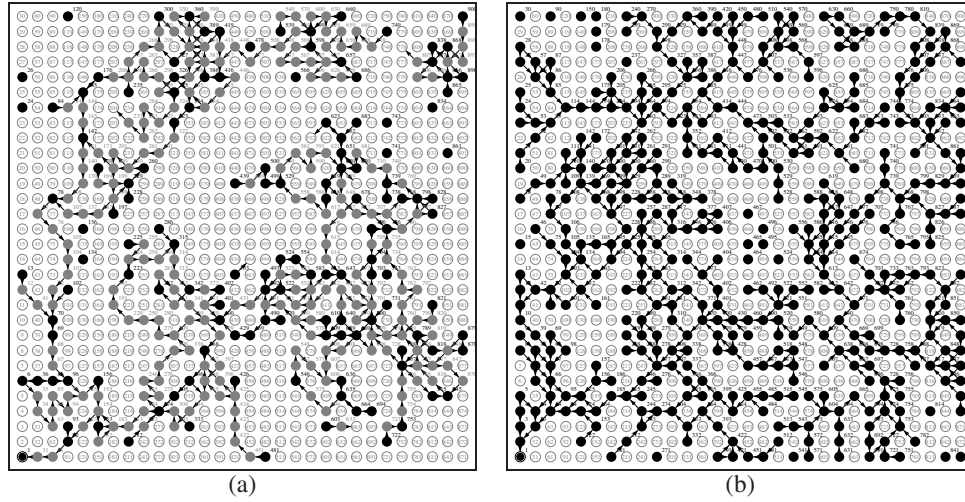


Figure 4. Sample network connections, where the sink node resides in the bottom left corner. Food nodes are shown in black and non-food forwarders in gray. (a) $K = 3$ hops, $P = 0.15$, and (b) $K = 1$ hop, $P = 0.5$

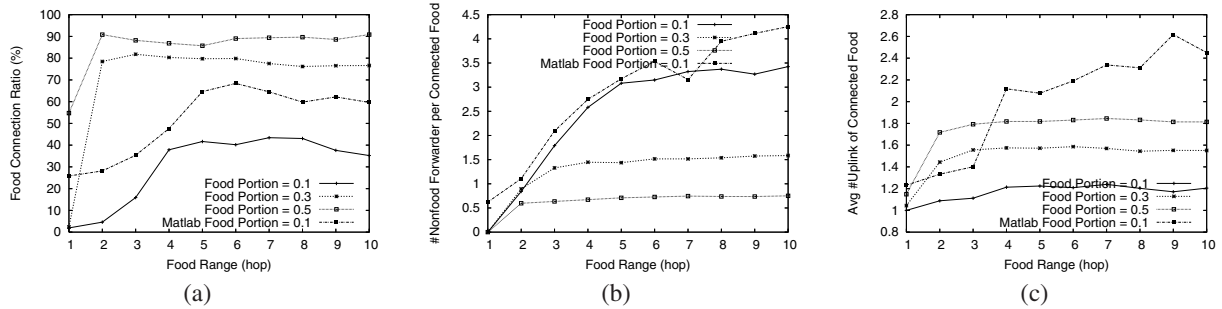


Figure 5. Comparison among $P = 0.1, 0.3, 0.5$ in QualNet and $P = 0.1$ in Matlab, with K varying from 1 to 10 hops, in terms of (a) food connection ratio, (b) number of non-food forwarders per connected food, and (c) average number of uplinks of connected food

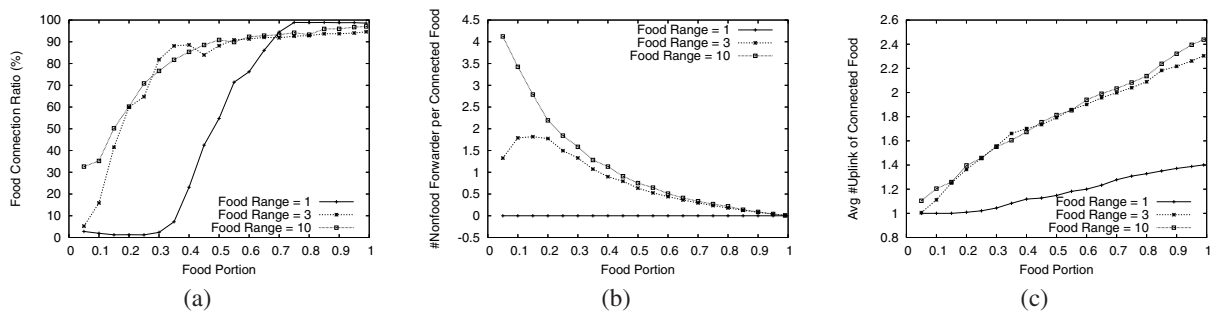


Figure 6. Comparison among $K = 1, 3,$ and 10 hops with P varying from 0.05 to 0.99 , in terms of (a) food connection ratio, (b) number of non-food forwarders per connected food, and (c) average number of uplinks of connected food

neighbor-food table, exposing more targets to reach. However, too big K also introduces much more potential packet collisions, which offsets the gain in connection ratio. Both the cases of $P = 0.3$ and 0.5 reach their peak connection ratio at $K = 2$ hops, while both $P = 0.1$ cases takes more hops, since a sparse food distribution has less potential packet collisions under the same food range. As for the two $P = 0.1$ cases, the connection ratio in the WSN protocol is comparable to that of the Singular Potential model — about 18% less, due to the lossy communications in WSN. As mentioned above in Figure 4(b), a higher food portion P enjoys a higher connection ratio, as well as a better efficiency (*i.e.*, less non-food forwarders per connected food), which is also evidenced in Figure 5(b). It can be seen in Figure 5(b) that the number of non-food forwarders starts with zero for the three WSN cases when K is 1 hop, which is true for all P as discussed in Figure 4(b) above. The metric rises linearly with K , and then remain flat at different hops, since reaching far-away foods needs more relay nodes until connection ratio can be increased no more. For the two $P = 0.1$ cases, the WSN protocol is slightly more efficient in terms of forwarder number (Figure 5(b)), and less robust w.r.t uplink number (Figure 5(c)) than the Singular Potential model, which suggests that the model trades more efficiency for robustness than the WSN protocol. Figure 5(c) shows that the metric of uplink number has a similar trend as the connection ratio, since a food node could be reached from more upstream foods when there are more potential requesters within its growing size of neighborhood until the connection ratio levels off.

In the second experiment, food portion P are varied from 0.05 to 0.99 and the same three metrics are compared among K of 1, 3, and 10 hops. As the food portion increases, connection ratio first builds up quickly and then levels off, as presented in Figure 6(a). For $K = 1$, less than 5% of foods are connected when P is less than 0.3, since each food is too far away to be known by each other in a sparse network with a short food range. Then the ratio jumps dramatically with P until peaking close to 100% at $P = 0.75$. Other two cases of $K=3$ and 10 also peak as close to 100% as P increases. This is because the parameter of food portion plays a major role in affecting the connection ratio, compared with the parameter of food range. Figure 6(b) shows that efficiency rises as the food portion increases, until reaching the perfect case of zero non-food forwarder. Note that when $K = 1$, the number of non-food forwarder is always zero no matter what P value, as depicted in Figures 4(b) and 5(b). Finally, the average number of uplinks increases linearly with P in all three cases, as seen in Figure 6(c), since a denser food distribution always results in a higher food connection ratio, which produces more upstream requesters to build up the robustness.

6. Conclusion

In this paper, we first mathematically model the tubular network formation behavior of slime mold via *Singular Potential*, and then adapt the model to design a localized protocol to connect sensors to sink(s) that balances efficiency and robustness in wireless sensor networks (WSN). The simulation results have shown that the protocol is effective in establishing network connectivities, with a balance of both efficiency and robustness.

References

- [1] E. Ben-Jacob and I. Cohen. Cooperative organization of bacterial colonies: From genotype to morphotype. In *Annual Review of Microbiology*, vol. 52, 1998.
- [2] B. J. Culpepper, L. Dung, and M. Moh. Design and analysis of hybrid indirect transmissions (hit) for data gathering in wireless micro sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(1):61–83, 2004.
- [3] Q. Fang, J. Gao, L. J. Guibas, V. Silva, and L. Zhang. Glider: Gradient landmark-based distributed routing for sensor networks. In *Proc. of the 24th Conference of the IEEE Communication Society (INFOCOM)*, pages 339–350, 2005.
- [4] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proc. of the 33rd Hawaii International Conference on System Sciences*, pages 1–10, 2000.
- [5] T. Hofer and P. K. Maini. Streaming instability of slime mold amoebae: An analytical model. *Physics Review*, E 56:2074–2080, 1997.
- [6] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.
- [7] T. Nakagaki, R. Kobayashi, Y. Nishiura, and T. Ueda. Obtaining multiple separate food sources: behavioural intelligence in the physarum plasmodium. *Proceedings of the Royal Society B-Biological Sciences*, 271(1554):2305–2310, November 2004.
- [8] T. Nakagaki, H. Yamada, and M. Hara. Smart network solutions in an amoeboid organism. *BIOPHYSICAL CHEMISTRY*, 107(1):1–5, 2004.
- [9] T. Nakagaki, H. Yamada, and A. Toth. Maze-solving by an amoeboid organism. *NATURE*, 407(6803):470–470, 2000.
- [10] T. Nakagaki, H. Yamada, and A. Toth. Path finding by tube morphogenesis in an amoeboid organism. *BIOPHYSICAL CHEMISTRY*, 92(1-2):47–52, 2001.
- [11] Scalable Network Technologies, Inc. QualNet Simulator. <http://www.scalable-networks.com>.
- [12] P. A. Stewart. The organization of movement in slime mold plasmodia. In R. D. Allen and N. Kamiya, editors, *Primitive Motile Systems in Cell Biology*. Academic Press, 1964.
- [13] A. Tero, R. Kobayashi, and T. Nakagaki. A mathematical model for adaptive transport network in path finding by true slime mold. *Journal of Theoretical Biology*, 244:553–564, 2007.