PAGER: A Distributed Algorithm for the Dead-end Problem of Location-based Routing in Sensor Networks

Le Zou, Mi Lu, and Zixiang Xiong
Department of Electrical Engineering, Texas A&M University
{lezou,mlu,zx}@ee.tamu.edu

Abstract

The dead-end problem is an important issue of location-based routing in sensor networks, which occurs when a message falls into a local minimum using greedy forwarding. Current methods for this problem are insufficient either in eliminating traffic/path memorization or finding satisfied short paths. In this paper, we propose a novel algorithm, named Partial-partition Avoiding GEographic Routing (PAGER), to solve the problem. The basic idea of PAGER is to divide a sensor network graph into functional sub-graphs, and provide each sensor node with message forwarding directions based on these sub-graphs. That results in loop-free short paths without memorization of traffic/paths in sensor nodes.

We implement our algorithm in a protocol and evaluate it in sensor networks with different parameters. Results show that PAGER generates considerably shorter paths, higher delivery ratio and lower energy consumption than the Greedy Perimeter Stateless Routing protocol. At the same time, PAGER achieves better performance in handling large-scale networks than the Ad-hoc On-demand Distance Vector protocol.

1. Introduction

Sensor networks consist of a large number of tiny sensors with limited computational capability and memory. Data collected by sensors are routed to the base station in a multi-hop manner [1]. Recently, a number of routing protocols have been proposed for sensor networks, such as non-location-based routing [2-6] and location-based routing [7-12].

Among these proposed protocols, location-based solutions [7-12] have received more attention because of their inherent scalable and power-efficient properties. The dead-end problem is an important issue of location-based routing in sensor networks. This problem arises when greedy forwarding fails at a sensor node that has no closer neighbor with respect to the distance to the base station.

To handle this situation, both the memorization-based [8] and stateless [9,12] recovery algorithms have been proposed. Memorization-based recovery methods require nodes to memorize past traffic or paths, which consume storage spaces of resource-constrained sensors and is not scalable. Stateless recovery methods do not require a node to memorize past traffic or paths. The Greedy-Face-Greedy (GFG)/Greedy Perimeter Stateless Routing (GPSR) algorithm [9,12] is currently the most widely accepted stateless recovery algorithm in ad-hoc/sensor networks in existing literature. GFG/GPSR elegantly handles the dead-end problem by routing a message along faces. However, this algorithm generates long paths with loops compared to shortest-path, which increases traffic burden, energy consumption and risk of losing of data packets in large-scale sensor networks. Further, GFG/GPSR require the planarization of underlying graphs, which increases computation complexity.

In this paper, we propose a distributed algorithm, named Partial-partition Avoiding GEographic Routing (PAGER), which divides a sensor network graph into functional sub-graphs and provides forwarding directions for each sensor node based on these sub-graphs. PAGER does not require a node to memorize past traffic or paths; it constructs loop-free paths with lengths close to shortest-path. Furthermore, PAGER does not need the planarization of the underlying graphs. We implement PAGER in a protocol and evaluate its performance in different parameterized network topologies. Experimental results show that PAGER constructs considerably shorter paths and has higher delivery ratio than GPSR in large-scale sensor networks. Additionally, PAGER is more scalable than Ad-hoc On-demand Distance Vector protocol (AODV) [13]. Finally, PAGER is shown to be energy efficient.

The rest of the paper is organized as follows. The dead-end problem of location-based routing in sensor networks is introduced in section 2. The proposed PAGER algorithm for solving this problem is described in section 3. In section 4, implementation of PAGER detailed and experimental results presented. In the end, we give conclusions in section 5.

2. Dead-end Problem in Sensor Networks

We consider the following situation: a base station is adjacent to an unobstructed two-dimensional plane that consists of a number of randomly deployed sensor nodes. These sensor nodes are modeled by a unit graph. All nodes within communication range of a
node $x$ are considered as neighbors of $x$ and have bidirectional links with node $x$, as shown in Fig. 1. The base station has a communication range $R$, which is long enough to cover all sensor nodes. Information collected at a sensor node is forwarded to the base station in a multi-hop manner. As Fig. 1 shows, in greedy forwarding [4], node $x$ forwards a message to neighbor $y$ because node $y$ has closer distance to the base station. This greedy forwarding approach normally can lead a message to the base station in dense networks, as the message originated from node $x$ shown in Fig. 1. However, there are exceptions. As shown in Fig. 1, after node $z$ receives a message from node $y$, it finds that there is no closer neighbor to the base station to forward the message even though there is a path. Thus, the dead-end problem arises. In Fig. 1, node $z$ is called a concave node.

![Fig. 1. The network model and dead-end problem.](image)

The dead-end problem of location-based routing in sensor networks is associated with concave nodes. In the next section, we propose PAGER to solve the dead-end problem.

### 3. Algorithm Description

PAGER uses two phases to solve the dead-end problem in sensor networks. The first phase is called the shadow-spread phase, which divides a connected graph into sub-graphs originated from concave nodes. The second phase is called the cost-spread phase, which establishes paths on a given graph based on the sub-graphs divided in the first phase.

#### 3.1. The Shadow-spread Phase

The shadow-spread phase tries to locate the "dangerous areas" close to concave nodes. Let us consider the following process of shadow-spread algorithm on graphs shown in Fig. 2(a)-Fig. 2(c). In Fig. 2(a), node $A$ is a concave node while nodes $B$ and node $C$ are not. If we disconnect node $A$ from the graph (shown in Fig. 2(b)), we can see that nodes $B$ and $C$ become concave nodes. We go one step further by disconnecting nodes $B$ and $C$ from the graph. As Fig. 2(c) shows, by disconnecting the sub-graph consisting of nodes $A$, $B$ and $C$ from the original graph, the resulting graph contains no concave node. Therefore, messages originated from the resulting graph in Fig. 2(c) will be forwarded to destination using greedy forwarding without encountering the dead-end problem.

In the example shown in Fig. 2, we denote nodes $A$, $B$ and $C$ as shadow nodes, the rest of the nodes on the graph become bright nodes. Specifically, the bright nodes adjacent to shadow nodes are called border nodes, which on the graph shown in Fig. 2(a) are nodes $D$ and $E$.

Differentiating the nodes on the graph shown in Fig. 2 by statuses (shadow/bright), we divide the original graph (Fig. 2(a)) into two sub-graphs. We call the sub-graph that contains shadow nodes (A, B and C) shadow area. Similarly, the sub-graph that contains bright nodes is called bright area. Further, as shown in Fig. 2, the void area on the graph that is encompassed by the concave node $A$ is called partial partition, which actually partially partitions the sensor network in Fig. 2 and creates concave node $A$.

In order to divide the original graph into shadow areas and bright areas as shown in Fig. 2(c), nodes should be able to exchange information of their status (shadow/bright) along with their location information. This information exchange is realized by periodically broadcasting beacon messages that contain two fields: status and location. Every node on a graph should be able to obtain the location information of the base station. Based on our network model, sensor nodes obtain the base station's location from its broadcast channel.

The process shown in Fig. 2(a)-Fig. 2(c) is an example of the shadow-spread phase, which is given in detail in Fig. 3. The auxiliary function DecideShadow shown in Fig. 4 is used to determine whether a node is a shadow node.

There is an important property of shadow/bright areas with a given connected graph. We give the property in proposition 1 (proof is omitted due to space limitations).

**Proposition 1:** In a connected graph $G$, after shadow areas and bright areas are divided, a message originated or being forwarded in shadow nodes will find a path ending with a concave node using greedy forwarding [4]; a message originated or being forwarded in a bright node will find a path ending with the destination (base station) using greedy forwarding [4] if we disconnect shadow areas from $G$. 

510
Fig. 2. An example process of the shadow-spread ((a)-(c)) and cost-spread ((d)-(h)) phases. (a) The graph contains concave node A. (b) After disconnecting node A from graph, new concave nodes B and C appear. (c) After excluding the sub-graph consisted by A, B and C, resulted graph contains no concave nodes. (d) Before phase 2, shadow nodes (A, B, C) are determined. (e) Node A increases its cost to 22. (f) Node B and C increase their costs to 23 and 25 respectively. (g) Node A increases cost again to 28. (h) All shadow nodes (A, B and C) are satisfied with their costs; cost gradients are established from high-cost-to-low-cost.

Shadow-Spread (node x):
status(x) = bright;
while active do
  if receive beacon(status, location) from a neighbor y then
    Refresh the neighbor set N(x) by updating the status and location information of y;
    if DecideShadow(x) then
      status(x) = shadow;
    else
      status(x) = bright;
    end if
  end if
  if beacon timeout occurs then
    Copy status and location information of node x to Beacon(status, location) and broadcast;
  end if
end if

Fig. 3. The Shadow-spread phase running at node x.

Bool DecideShadow(node x):
shadow = true;
for each node y ∈ N(x) do
  if ((status(y) = bright)
    && (distance(y) < distance(x)) then
    return false;
  end if
end if
return shadow;

Fig. 4. Auxiliary function DecideShadow.

The proposition shows that a shadow node will always send a forwarding message to a "shadow" neighbor using greedy forwarding. For example, node B on the graph shown in Fig. 2(a) will always forward a message to node A. Therefore a message being forwarded in a shadow node will eventually be trapped in a concave node using greedy forwarding. On the other hand, messages from bright nodes will be safely forwarded to the base station without encountering dead-ends if we disconnect shadow nodes from a graph. For example, node E will forward a message to the base station via node F rather than C if node C is disconnected from the graph; otherwise, all messages from E will be forwarded to node C and lost in concave node A.

The convergence time (in term of beacon interval B) of the shadow-spread phase is related to the size of a shadow area (number of shadow nodes) given a connected graph. Here we give the convergence time of shadow-spread phase in proposition 2 (proof is omitted due to space limitations).

Proposition 2: Given a connected graph G, the convergence time of shadow-spread phase is linear with respect to the maximum size of shadow areas of G.

3.2. Cost-spread Phase

Based on the sub-graphs resulted from the first phase, the second phase begins. To illustrate the process of the second phase, we show an example in Fig. 2(d)-(f). Initially each node on the graph has a variable equal to its Euclidean distance to the base station as Fig. 2(d) shows. We mark the variables on each node. In the first step, every shadow node tries to avoid being surrounded by neighbors with larger variables. As Fig. 2(e) shows, node A finds that all its neighbors have larger variables. In response to this situation, it increases its variable to 22 to be larger than the maximum value of the variables in neighbors by Δ (Δ is set to 3 in this example). In the second step, nodes B and C find that all their neighbors have larger variable (since node A increased its variable). In response to this situation, they increase their variable to 23 and 25, respectively, as Fig. 2(f) shows. This process ends in the third step when node A increases it variable from 22 to 28, as shown in Fig. 2(g). Now, every shadow node (A, B and C) is
satisfied because they can at least find a neighbor with a smaller variable. We denote the variable maintained in each node as the cost. If we give each node a direction to a neighbor with the lowest cost, as Fig. 2(h) shows, we establish gradients across the whole graph. We call these directions cost gradients. As shown in Fig. 2(h), paths from sensor nodes to the base station are established following these gradients.

To make the process shown in Fig. 2(d)-Fig. 2(h) possible, cost information should be exchanged between nodes. Like status information used in the first phase, this is realized by adding the cost information to the periodically broadcasted beacon messages.

Based on the above process, the cost-spread phase is detailed in Fig. 5.

```plaintext
Cost-Spread (node x)
cost(x) = distance(x);
while active do
  if receive beacon(cost) from a neighbor y then
    Refresh the neighbor set N(x) by updating the cost of y;
    if cost(x) ≤ minimum cost of neighbor set N(x) then
      cost(x) = maximum cost of neighbors + Δ;
    end if
  end if
  if beacon timeout occurs then
    Copy cost of node x to Beacon(cost) and send out;
  end if
end while
```

Fig. 5. The Cost-spread Phase.

After cost gradients are established, we have the following observations from the example graph shown in Fig. 2(h):

1) A message originated from the bright area will be eventually forwarded to the base station following cost gradients without entering the shadow area. 2) A message originated from a shadow area will be leading out of a shadow area to bright area and eventually reaches the base station via bright nodes following the cost gradients. 3) A message originated from any node on the graph will not experience the same node twice following the cost gradients.

Based on the above observations, we have the following proposition (proof is again omitted due to space limitations):

**Proposition 3:** Following the cost gradients established on a given connected graph G, all paths originated from sensor nodes will be ended with the destination (base station) and are loop-free.

The convergence time (in term of beacons broadcast interval $B$) of the cost-spread phase depends on the size (number of nodes) of the shadow areas on a given graph. After analysis, we give the following proposition without proof.

**Proposition 4:** Given a connected graph $G$, the convergence time (in term of beacons broadcast interval $B$) of cost-spread algorithm is linear with respect to the maximum size (number of nodes) of shadow areas on $G$.

By combining the shadow-spread and cost-spread phases, PAGER tries to avoid messages from entering "dangerous areas" close to dead-ends, which are actually caused by partial-partition defined previously. This is why we call our scheme PAGER (Partial-partition Avoiding GEOgraphic Routing).

Given a sensor network, the passive mode of sensors decreases the average degree of the original network topology. This may cause more dead-ends. But as long as the resulting network with active sensors is connected, PAGER will always converge to a stable state, and hence guarantee each active sensor a path to the base station.

4. Performance Evaluation

To test the performance of our proposed PAGER algorithm, we implement PAGER into a protocol called PAGER-S for topologies with limited mobility, where "S" denotes for topologies with limited mobility. Similar to SPEED [10], PAGER-S reduces the amount of beacon messages by prolonging the beacon broadcast interval. To shorten the convergence time, all nodes in PAGER-S send out their beacon messages containing status and cost information after their status and costs change. We reduce the number of collisions occurring at the MAC layer by randomizing the sending time of these beacon messages.

We use the wireless extensions of ns-2 to simulate PAGER-S. The simulation parameters are listed in Table 1. The base station is located at (500,1000). We simulate 7 CBR flows originated from 7 randomly chosen nodes across the whole networks. Each CBR flow is sent at 256 bps and uses 32-byte packets. Random topologies (15 for each degree, number of nodes) are created. All simulation results are based on the average values of these topologies.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>100 - 500</th>
<th>Average Degree</th>
<th>6 - 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload</td>
<td>256 bps \times 7</td>
<td>Region</td>
<td>1000 m \times 1000 m</td>
</tr>
<tr>
<td>Simulation Length</td>
<td>100s</td>
<td>Routing Protocol</td>
<td>GPSR, PAGER-S, AODV</td>
</tr>
<tr>
<td>MAC Layer</td>
<td>802.11</td>
<td>Propagation Model</td>
<td>Two-Ray</td>
</tr>
</tbody>
</table>

We first simulate the distribution of shadow/bright areas in 500-node networks. Then the convergence time of PAGER-S is obtained in sparse networks (average degree =6) with different number of nodes (100–500 nodes). Finally, by comparing with GPSR [12], AODV [13] and shortest path, we evaluate the following metrics of PAGER-S: 1) packets delivery
ratio, 2) path length, 3) control overhead, 4) end-to-end delay time, and 5) energy consumption.

We compare PAGER-S with AODV and GPSR in large-scale networks (500 nodes). The reason we choose AODV rather than DSR, which is chosen in [10,12], is because we find that AODV performs better in large networks. GPSR is chosen because it is currently the most widely accepted stateless recovery method of location-based routing.

Distribution of Shadow Nodes/Areas
The performance of PAGER depends on the distribution of shadow/bright areas. We randomly create sparse network topologies (100-500 nodes, average degree=7). The average distribution of shadow nodes/areas is shown in Fig. 6. From the results we can see that there are 15%-25% nodes in these networks are shadow nodes, which will cause at least this percentage of packet miss rate using greedy forwarding without solving the dead-end problem. From Fig. 6 we can see that the maximum size of shadow areas is small (26 nodes) even in large-scale networks (500 nodes). Since the convergence time of the PAGER algorithm depends on the maximum size of shadow areas, as we see later, PAGER-S quickly converges to a stable state.

Convergence Time of PAGER-S
PAGER needs time to stabilize the cost gradients. The average convergence time of PAGER in sparse networks (degree=7) is shown in Table 2. The average convergence time of PAGER-S is 4.0-5.7 seconds in 500-node networks. In 100-node networks (100 nodes), the average convergence time of PAGER-S reduces to 0.5-3.1 seconds.

Packet Delivery Ratio
The packet delivery ratio of PAGER-S is higher than that of AODV and GPSR in 500-node networks. As Fig. 8 shows, PAGER-S delivers about 5% more data packets to the base station than AODV even with a light traffic load (7 CBR flows from 7 nodes). This is because AODV has to flood control packets throughout the networks to find paths to the base station. We observe a low packet delivery rate (<82%) of GPSR in sparse networks (degree=6-8) as shown in Fig. 8. This is due to the long path length of GPSR that increases a data packet’s risk of being dropped. Different from AODV and GPSR, PAGER-S delivers more data packets by establishing the cost gradients across the whole networks before data transmissions. Further, PAGER-S maintains low control overhead by utilizing geographic information as we see later.

Path Length
We simulate 500-node topologies to compare the path length of PAGER-S, AODV, GPSR and shortest path. Simulation results are shown in Fig. 9. PAGER-S has a path length close to shortest path and is shorter than that of GPSR and AODV. In sparse networks (average degree=6), the path length of GPSR is about 6 hops longer than that of PAGER-S. PAGER-S also constructs shorter paths than AODV in all these topologies as shown in Fig. 9.

Control Overhead
To test the control overhead of PAGER-S, we compare PAGER-S with AODV and GPSR in sensor networks with 500 nodes. The results are shown in Fig. 10. In all cases, the number of control packets generated in AODV is over 4000 higher than that of PAGER-S. Compared to GPSR, PAGER-S produces a larger amount of control packets. This is because nodes...
in PAGER-S send out extra beacon messages when they detect status/cost changes.

End-to-end Delay

Since PAGER-S constructs paths with shorter length than AODV and GPSR, it is not surprising to see the data packet's end-to-end delay in PAGER-S is smaller than that in AODV and GPSR. As shown in Fig. 11, PAGER-S has delay time lower than that of AODV and GPSR in all cases.

Energy Consumption

With low routing overhead and short path length, again it is not surprising to see the energy efficiency of the PAGER algorithm. Here we give the energy consumptions of the routing protocols in Fig. 12. GPSR has the highest energy consumption because of its long detours when dead-ends are met, which happens frequently in sparse networks.

5. Conclusion

In this paper, we have presented a distributed routing algorithm called PAGER to solve the dead-end problem of location-based routing in sensor networks. We showed that PAGER guarantees loop-free delivery. We implemented PAGER in a protocol called PAGER-S for sensor networks with limited mobility. The performance of PAGER-S is compared with AODV, GPSR and shortest path. Experimental results confirmed the advantages of PAGER.

References