A Dead End Avoidance Method for Geographic Forwarding in MANETs

Jian-Ming Chang  
Dept. of Computer Science & Information Engineering, National Dong Hwa University  
Hualien, Taiwan  
a0128866@niu.edu.tw

Han-Chieh Chao  
Dept. of Electronic Engineering and Institute of Computer Science & Information Engineering, National Ilan University, I-Lan, Taiwan  
hcc@niu.edu.tw

Jiann-Liang Chen  
Dept. of Electrical Engineering, Nation Taiwan University of science and Technology  
Taipei, Taiwan  
lchen1215@gmail.com

ABSTRACT
Geographic Forwarding is part of geographic routing that each node only needs to know the location of its neighbor and the destination. This method can reduce the cost that table-driven routing needs to maintain the whole path even if the path are not in use, and save time when searching the path compared with the reactive routing. When we using geographic forwarding, it usually encountered the local maximum that cannot forward the packet directly to the destination; this fundamental problem is also called “Dead End”. We use the algorithm called “A star” that usually used in role playing game or strategic game to detour the terrain that cannot pass through directly. Nodes must vote their agent in a specific area to be a decision-maker to find a reference route for source node. When the route is decided, then we will use geographic forwarding according to this reference path to the destination to avoid the dead end.

Categories and Subject Descriptors
C.2.1 [Network Architecture and Design]: Wireless communication.

General Terms
Algorithm, Performance, Design, Experimentation.

Keywords
Geographic forwarding, Dead end, A* algorithm, MANET.

1. INTRODUCTION
Ad hoc network can categorized into topology based networks and position based networks. In topology based networks, each node must know the link state between its neighbors, and record the route in the routing table. Or send route request when some node want to communicate with the other node. Although these kinds of networks can easy to find a path to destination, very large routing table and flooding will be paid in such networks. In position based networks, each node must have its own GPS and UID to identify the location and identity. The only thing that node needs to know is its one-hop neighbors’ location information. Every node exchanges the geographic information with its neighbors using hello message. Once the source node gets the destination information from location server, and then it can use geographic forwarding to send data packets. When performing geographic forwarding in the uneven or sparse networks, there is a fundamental problem called “Dead End”. And this situation might cause packet loss and heavy overhead in large networks. Our proposal is to find an efficient path for source node to refer to, and then send data packets using geographic forwarding. We use “A* algorithm” that can handle the unexplored area. Nodes in the square will vote their own agent to be a path finder to perform A* algorithm. After that, the source node can send packet without encounter dead end.

Figure. 1 [1] shows the dead end situation in MANETs. Node S wants to communicate with D, but it cannot directly send the packets to node D. It must forward packets through its neighbor. While looking for the next relay node, node will follow the rule “the position who closer to the destination in my transmission range” to send data packets. But path (S—A—B—C) will lead data packets into a dead end. And we must find the other path to detour the dead end.

Figure 1. Dead end situation

GPSR [2] use geographic forwarding with greedy mode and perimeter mode for recovery the dead end situation. When
packets encounter the dead end, it will perform perimeter routing to help the packets around the dead end. Due to the uncertain path, if GPSR encounter dead end and switch to perimeter routing that use right hand rule, it might be encounter the other dead end. And this would cause heavy overhead or even packet loss. When we apply A* algorithm to the geographic forwarding, it not only deal with dead end situation, but also find a stable way to the destination. The rest of the paper is organized as follows: in section II, we give some background about geographic forwarding and the A* algorithm. Section III is the proposed method to avoid the dead end situation. Section IV and V are simulation results and conclusions, respectively.

2. BACKGROUND

2.1 Geographic Routing

Geographic routing [3] consists two parts: location service [4] and geographic forwarding. Before using geographic forwarding, we must perform the location service to get the location information of the destination. In the location service, each node registers with its location server periodically. When a node wants to communicate with the other node, it will perform location service to query the location server and get the location information of the destination. In geographic forwarding, there are two assumptions that each node has their own ID and a positioning device (ex. GPS). And each node will store the information about its one hop neighbors.

There are different methods to find a relay node [5], as shown in Figure. 2.

![Figure 2. Different relay nodes using geographic forwarding](image)

- A. Nearest with Forwarding Progress (NFP)
- B. Most Forwarding progress within Radius (MFR)
- C. Compass Routing
- E. Greedy forwarding
- S. Source node
- D. Destination node

2.2 A* algorithm

A* algorithm [6] [7] is usually used in role playing game or strategic game to find the path to the destination. The characters are always finding a way to their destination without stick in some terrain like lakes or mountains. When using A* algorithm, two main list must be maintained: the OPEN_list and the CLOSED_list. The OPEN_list stores squares that not yet explore or prepare to explore; The CLOSED_list stores squares that have been explored. Source square will choose the next square depend on the function \( F(n) = G(n) + H(n) \) that with the minimal cost \( F \). \( F \) is the total moving cost, \( G \) is the moving cost from source square to current square, and \( H \) is a heuristic estimation [8] from current square to destination square. Because of the unexplored area, we will ignore the dead end effect when calculate \( H \) cost. Figure. 3 shows the example of the A* algorithm. At beginning, source node (16) will calculate the minimal movement cost to the destination node. Let the cost of each horizontal or vertical square move is 10, and the cost of diagonal is 14. Then we can get the \( G \) cost. Since the \( H \) cost is a heuristic estimation, here we simply use Manhattan distance [10] that calculates the total number of squares from current square to reach destination square moving horizontally and vertically. And the cost of moving one square is 10. Each time that the current square determines its next square, the next square will leave a pointer point to its previous square.

![Figure 3. Shows the example of the A* algorithm.](image)

The simply steps are shown below:
1. Add the source square to the OPEN_list.
2. Repeat the following steps:
   a. Search the OPEN_list to find the square with the minimal \( F \), and we call this square the “current square”.
   b. Remove this square ID from the OPEN_list and add it to the CLOSE_list.
   c. For each of the 8 neighbors closer to this current square
      - If it cannot pass or already on the CLOSED_list, leave it.
      - If it is not on the OPEN_list, add it. Make the current square a parent square and record the cost of \( F \), \( G \) and \( H \).
      - If it already on the OPEN_list, check the cost of \( G \) that if path through it can be better, modify the pointer, the cost of \( G \) and \( F \). The less \( G \) cost, the better the path is.
   d. Get stop when:
      - Destination square is added to the CLOSED_list, go back in accordance with the pointer.
      - Before add the destination square, the OPEN_list is already empty. In other words, there is no path.
3. Save the path according to the pointer that point to its parent square.
Once we found a reference path from A* algorithm, we can use geographic forwarding to send data packets without dead end. In the following section, we’ll describe how to combine these two mechanisms and perform a great routing protocol.

3. APPROACH
To avoid the dead end situation when using GPSR and routing on the stable path that wouldn’t change mode too often, we try to find a reference path for source node before using geographic forwarding. We first divide the network environment into several square areas according to node’s transmission range. Every node uses the following function to calculate the square ID:

\[ SID = \left( \left\lfloor \frac{X}{UL} \right\rfloor * UL, \left\lfloor \frac{Y}{UL} \right\rfloor * UL \right) \]  

(1)

Nodes use square ID to distinguish the location between neighbors. Each square area will have an agent on behalf of it voted by nodes in that square. This agent will periodically announce the existence and the location information to its neighbor nodes in the square. Although the size of the square area affects the performance, we must choose a suitable square size. The larger the size is, the fewer the number of agents we need. Bad square selection could still lead to dead end situation. However, if we choose the size of square area is too small, this means that there would be too many agents. The size of square area depends on node’s transmission range. Assume that the node’s transmission range is \( r \), in order to cover the eight square areas of neighbor, the minimal length of side must be \( \frac{r}{\sqrt{2}} \), as shown in the Figure. 4.

\[ \left( \left\lfloor \frac{X}{UL} \right\rfloor * UL + \frac{UL}{10} \right) < X < \left( \left\lfloor \frac{X}{UL} \right\rfloor * UL - \frac{UL}{10} \right) \]  

(2)

\[ \left( \left\lfloor \frac{Y}{UL} \right\rfloor * UL + \frac{UL}{10} \right) < Y < \left( \left\lfloor \frac{Y}{UL} \right\rfloor * UL - \frac{UL}{10} \right) \]  

(3)

Node uses the following function to determine whether it within the threshold or not:

After setting the size of square area, we set the threshold to prevent the agent too close the square boundary. Figure. 5 shows the chosen agent must considered within the threshold. The node within the threshold might remain more time staying in the square than outside ones. But if there isn’t the other node with the same square ID, that means only one node in the square, this node must be the agent.

When source agent get that information, it will perform the A* algorithm to find the next square with the minimal cost \( F \) according the function \( F(n) = G(n) + H(n) \). Here we modified the original A* algorithms functions and add a parameter \( \beta \) that keep the square more effective. If there are more nodes in the same square area, which means this square will have more choice to help the data packets to forward to, and we will arise the...