Workshop on Algorithms and Computational Molecular Biology, ICS 2002

A Wave Algorithm for Mobile Ad Hoc Networks

Sheng-Hsiung Chen*and Ting-Lu Huang

Dept. of Computer Science
and Information Engineering
National Chiao Tung University
1001 Ta-Hsueh Road
Hsin-Chu, Taiwan 30050
Republic of China
phone: 886-3-5731648
fax: 886-3-5724176
e-mail: {chenss, tlhuang}@csie.nctu.edu.tw

Abstract

In designing distributed algorithms, several general problems appear frequently as subtasks, including broadcasting, leader election, mutual exclusion, computing a global function of which each process holds part of the input. Each of these tasks requires a basic scheme, called wave algorithms, that ensure the participation of all processes. This paper presents a wave algorithm for mobile ad hoc networks in which links may fail or reform. Without assuming that the network topology is always connected, the algorithm only requires the network to be *consistently* connected, a very mild condition imposed on the network ensuring that each message flooded by a process will be received by all processes eventually. The algorithm guarantees that a decide event eventually occurs in each process, and each decide event is causally preceded by an event of each other process.

Keywords: wave algorithm, ad hoc network, mobile computing, distributed algorithm

*Contact Author: Sheng-Hsiung Chen

1 Introduction

Within the last few years there has been a surge of interest in mobile ad hoc networks (MANET) [1]. A MANET is defined as a collection of mobile platforms or nodes where each node is free to move about arbitrarily [2]. A pair of nodes communicates by sending messages either over a direct wireless link, or over a sequence of wireless links including one or more intermediate nodes. A pair of nodes can communicate directly only if they lie within one another's transmission radius. A link forms between a pair of nodes when nodes move into one another's transmission radius; in contrast, a link fails when nodes move out of one another's transmission radius.

Due to link failures and link formations, designing distributed algorithms for MANET is a challenging task. Much previous work focus on routing protocols such as [5, 6, 7, 8]. Several distributed algorithms are modified for MANET, including mutual exclusion algorithms [9], leader election algorithms [10], etc.

This paper studies wave algorithms for MANET. A wave algorithm ensures the participation of all processes and has be known as a useful building block in distributed systems. For example, it can be used for some fundamental tasks, e.g. broadcasting [13] and computing some global functions of which each process holds part of the input [14]. In addition, wave algorithms can be used in more complicated problems such as leader election, termination detection, and mutual exclusion [4].

In MANET, all applications mentioned above for distributed systems can be constructed as usual if a wave algorithm designed for the ad hoc network is available as a building block. A wave algorithm for MANET hides the dynamic nature of the network topology. Thus, we can design algorithms based on a wave algorithm without dealing with the dynamic nature of MANET.

In every wave process, there is a special type of internal event called a *decide* event. A wave algorithm exchanges messages and then the algorithm makes at least one decision, which depends causally on some event in each process. A process is an *initiator* if it starts the execution of its local algorithm spontaneously; in contrast, a *non-initiator* becomes involved in the algorithm only when a message of the algorithm arrives and triggers the execution of the process algorithm.

A wave algorithm is called *centralized* (e.g., [12, 13]) if there must be exactly one initiator in each execution, and *decentralized* (e.g., [3, 14]) if the algorithm can

be started spontaneously by an arbitrary subset of the processes. A decentralized algorithm is more general. However, more messages are needed.

If there are two nodes that always can't communicate directly or indirectly, no wave algorithm exists. Thus, some restriction must be made on nodes' mobility to solve this problem. Obviously, if we assume that the network is always connected, the problem is avoided. In this paper, a weaker restriction on nodes' mobility is made. We restrict the nodes' mobility such that the network is *consistently* connected. That is, a message flooded by a process will be received by all processes eventually. All links along the paths from the sender to all processes may not exist simultaneously, but the message can be stored and forwarded through these paths. Under this restriction, we present a decentralized wave algorithm. Each process in our algorithm will have a decide event eventually.

The rest of the paper is organized as follows: Section 2 presents the system model and the formal definition of wave algorithms. In section 3, we presents a wave algorithm. A proof of correctness is given in section 4. Finally, section 5 presents conclusions.

2 System Model and Definition of Wave Algorithms

In this section, we describe the system model and the correctness conditions of wave algorithms. The system model is adapted from the one in [9].

2.1 System Model

We consider an asynchronous distributed system consisting of a finite set P of independent mobile nodes, communicating by message passing over a wireless network. A link between two nodes indicates they are within one another's transmission radius. Assumptions on the mobile nodes and network are:

1. the nodes have unique node identities,
2. node failures do not occur,
3. communication links are bidirectional,

4. neighbor-awareness, that is, a link-level protocol ensures that each node is aware of the set of nodes with which it can currently directly communicate by providing indications of link formations and failures,

5. the network is connected initially,

6. the network is *consistently* connected.

If there exists two nodes that always can't communicate between one another directly or indirectly due to nodes' mobility, no wave algorithm exists. A link between two nodes must persist a sufficient period of time so that a message can be transmitted through it correctly. Thus, assuming that the network is always connected and each link persists a sufficient period of time ensures that each pair of nodes always has a chance to communicate. However, the weaker restriction we made, the more mobility of nodes is allowed. We desire to restrict on nodes' mobility weaker than always connected such that each pair of nodes can communicate with one another. A new definition of the connectivity requirement, consistently connecting, is presented. For this we need few definitions.

First, we define a set $\mathcal{L}$ of links with existing time to describe the network topology. A quadruple (p, q, t_1, t_2) is in $\mathcal{L}$ if a link between node p and q exists from time t_1 to t_2, and $t_1 < t_2$. Since the links are bidirectional, if (p, q, t_1, t_2) is in $\mathcal{L}$, so is (q, p, t_1, t_2). A link between p and q must persist a sufficient period of time, denoted by Δt_{pq}, such that a message sent by p will be correctly received by q. Therefore, a *consistent link* between two nodes at time t is defined to mean that the link persist a sufficient period of time such that one can correctly receive a message sent at time t by the other.

Definition 1 *There is a consistent link (p, q, t) between node p and q at time t if there exists a quadruple (p, q, t_1, t_2) in $\mathcal{L}$ such that $t \geq t_1$ and $t + \Delta t_{pq} \leq t_2$.*

Then, we define an access function $\gamma : P \times T \rightarrow 2^{P \times T} \setminus \emptyset$. T is the set of non-negative real numbers representing time. A pair (q, t') in $\gamma(p, t)$ represents that there exists a finite sequence of consistent links from p at time t to q at time t'. This sequence of consistent links contains at least one consistent link, except $q = p$ and $t' = t$. t' may be arbitrarily large but finite. That is, $\gamma(p, t)$ contains all that are

eventually reachable through a sequence of consistent links, hence can be defined by the usual reflexive and transitive closure of the consistent link relation.

Definition 2 *$\gamma(p,t)$ is recursively defined as follows.*

1. $(p,t) \in \gamma(p,t)$.
2. $(q,t') \in \gamma(p,t)$ *if* $\exists (r,t'') \in \gamma(p,t)$ *and a consistent link* $(r,q,t''') : t''' \geq t''$ *and* $t' = t''' + \Delta t_{rq}$.
3. *No other pairs are in* $\gamma(p,t)$.

Based on $\gamma(p,t)$, the reachability set $\mathcal{R}(p,t)$ is defined as a set of processes that are reachable from p after time t.

Definition 3 *$\mathcal{R}(p,t)$ is defined as follows.*

$$\mathcal{R}(p,t) = \{q \in P \mid \exists\, t' : (q,t') \in \gamma(p,t)\}.$$

If each process is reachable from p after time t, we say that the network is consistently connected with respect to node p at time t.

Definition 4 *A network is consistently connected with respect to node p at time t if $\mathcal{R}(p,t) = P$.*

Finally, a consistently connected network is defined as follows.

Definition 5 *A network is consistently connected if for all (p,t) pairs, the network is consistently connected with respect to p at time t:*

$$\forall p \in P : \forall t : \mathcal{R}(p,t) = P.$$

A consistently connected networks may be partitioned, but some links are supposed to dynamically reform so that every node at any time eventually has sequences of consistent links to all other nodes. An example is given below.

Example. The network contains three processes, p_1, p_2, and p_3, and the changes of the topology is $\mathcal{L} = \{(p_1,p_2,0,5),\ (p_2,p_1,0,5),\ (p_2,p_3,0,5),\ (p_3,p_2,0,5),\ (p_1,p_2,10,\infty),\ (p_2,p_1,10,\infty),\ (p_2,p_3,15,\infty),\ (p_3,p_2,15,\infty)\}$. That is, links (p_1,p_2) and (p_2,p_3) exist from time 0 to 5. Then, these links fail. Until time 10, link (p_1,p_2)

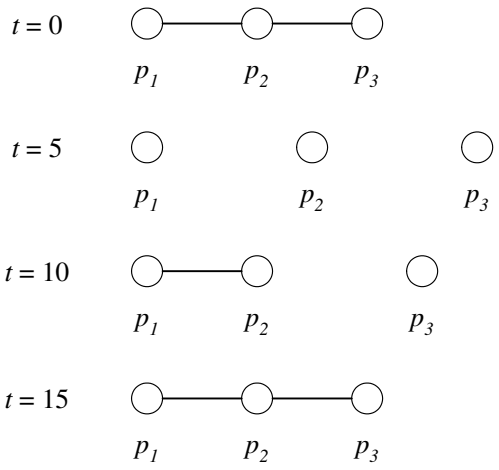

Figure 1: The changes of the network topology.

reforms and persists thereafter. Until time 15, link (p_2, p_3) reforms and persists thereafter. Fig. 1 shows the changes of the network. Assume that the Δt for each pair of nodes equals to 5.

Although the network is partitioned at time 5, the network is still consistently connected with respect to each process at time 5. For process p_1, p_2 is reachable from p_1 by a consistent link $(p_1, p_2, 10)$, and p_3 is reachable from p_1 by a sequence of consistent links $(p_1, p_2, 10), (p_2, p_3, 15)$. Thus, $\mathcal{R}(p_1, 5) = \{p_1, p_2, p_3\}$. Similarly, $\mathcal{R}(p_2, 5) = \{p_1, p_2, p_3\}$ and $\mathcal{R}(p_3, 5) = \{p_1, p_2, p_3\}$. □

Previous wave algorithms (e.g. [3, 12, 13, 14]) work correctly only if the network is always connected and each link is permanent, a special case of consistently connected network. In this paper, we design a wave algorithm working correctly in consistently connected networks.

Next, we assume that each node has a wave process, modelled as a state machine, with a set of states, some of which are initial states, and a transition function. The transitions are associated with named *events*. The events are classified as either *internal*, *input*, or *output*. The inputs and outputs are used for communication with the environment, while the internal actions are visible only to the process itself. The internal events at node p_i including the ones below.

- Init_{p_i}: if node p_i is an initiator, this event is enabled spontaneously to start the wave process.

- Decide_{p_i}: the decide event of node p_i.

Input events are as follows.

- $\text{Recv}_{p_i}(p_j, m)$: node p_i receives message m from node p_j.
- $\text{LinkUp}_{p_i}(p_j)$: node p_i receives notification that the link between p_i and p_j is now up.
- $\text{LinkDown}_{p_i}(p_j)$: node p_i receives notification that the link between p_i and p_j is now down.

The transition function takes as input the current state of the process and the input or internal event, and produces as a (possibly empty) set of output events and a new state for the process. Output event is:

- $\text{Send}_{p_i}(p_j, m)$: node p_i sends message m to node p_j.

Each state contains a local variable that holds the node's identity and a local variable that holds the current neighbors of the node. The neighbor set variable of each process must be properly updated by the state transition function in response to a LinkUp or LinkDown event.

A *configuration* is a set consisting of each process' state, describing the instantaneous state of the whole system. In an *initial* configuration, each state is an initial state and the neighbor variables compose a connected undirected graph.

An *execution* is an infinite sequence of the form C_0, int_1, in_1, out_1, C_1, int_2, in_2, out_2, C_2, ..., where the C_k's are configurations, the int_k's are internal events, the in_k's are input events, and the out_k's are sets of output events. The subscripts are positive real numbers, representing the time at which that events occur. At most one event by each process can occur at a given time. An execution must satisfy the following additional conditions.

- C_0 is an initial configuration.
- If in_k occurs at node p_i, then out_k and p_i's state in C_k are correct according to p_i's transition function operating on in_k and p_i's state in C_{k-1}.

- $\text{LinkUp}_{p_i}(p_j)$ occurs at time t if and only if $\text{LinkUp}_{p_j}(p_i)$ occurs at time t. Furthermore, $\text{LinkUp}_{p_i}(p_j)$ only occurs if p_j is currently not a neighbor of p_i. The analogous condition holds for LinkDown.

- If $\text{Recv}_{p_j}(p_i, m)$ occurs at some time t, then there is a corresponding $\text{Send}_{p_i}(p_j, m)$ occurs at some previous time t', and the link connecting p_i and p_j is continuous up between t' and t. That is, if p_j received a message from p_i at time t, the message must be sent by p_i before time t. However, we do not assume that every message sent by processes will always be received. A message may be lost in our model.

2.2 Definition of Wave Algorithms

In order to define wave algorithm, we first formally define the happened-before relation on events [11].

Let E be an execution. The relation $\prec$, called the *causal order*, on the events of the execution is the smallest relation that satisfies

1. If e and f are different events of the same process and e occurs before f, then $e \prec f$.

2. If s is a Send event and r the corresponding Recv event, then $s \prec r$.

3. $\prec$ is transitive.

A wave algorithm is an algorithm that satisfies the following conditions.

1. **Decision**. Each execution contains at least one decide event.

2. **Dependence**. In each execution each decide event is causally preceded by an event in each process.

3 Wave Algorithm for MANET

Finn's algorithm [3] is a wave algorithm that can be used in arbitrary networks but doesn't tolerate link failures and link reformations. We adapt Finn's algorithm so that it works in MANET.

First, we extract the basic idea of Finn's algorithm from [4]. In this algorithm, each process p maintains two sets of process identities, Inc_p and $NInc_p$. Informally speaking, Inc_p is the set of processes q such that an event in q precedes the most recent event in p, and $NInc_p$ is the set of processes q such that for all neighbors r of q an event in r precedes the most recent event in p. This relation is maintained as follows. Initially $Inc_p = \{p\}$ and $NInc_p = \emptyset$. Process p sends messages, including Inc_p and $NInc_p$, each time one of the sets has increased. When p has received a message from all neighbors, p is inserted into $NInc_p$. When the two sets become equal, p decides. The informal meaning of the two sets implies that for each process q such that an event in q precedes Decide_p, for each neighbor r of q also an event in r precedes Decide_p, which implies the dependence of the algorithm.

However, the network may be partitioned because each node has mobility in MANET. In this case, it is possible that a process at some partition decides before its decide event is causally preceded by an event in each process. In order to solve this problem, each process maintains an additional set of process identities, $Leave_p$, consisting of process q such that no event in q precedes the most recent event in p and q may be partitioned from p. Process decides only if $Inc_p = NInc_p$ and $Leave_p = \emptyset$. Thus, the problem is avoided.

3.1 Data Structures

- $active_p$: indicates whether process p is active. A process is active if it started spontaneously or received a message from its neighbor. Initially, $active_p =$ false.

- Inc_p: the set of processes q such that an event in q precedes the most recent event in p. Initially, $Inc_p = \{p\}$.

- $NInc_p$: the set of processes q such that for all neighbors r of q an event in r precedes the most recent event in p. Initially, $NInc_p = \emptyset$.

- $Leave_p$: the set of processes q such that no event in q precedes the most event in p and q is no longer p's neighbor now. Initially, $Leave_p = \emptyset$.

- $Neigh_p$: the set of all processes in direct wireless contact with process p.

```
var   active_p   : boolean          init false;
      Inc_p      : set of processes init {p};
      NInc_p     : set of processes init ∅;
      Leave_p    : set of processes init ∅;
      Neigh_p    : set of processes init {p's neighbors};

      Init_p:
1:    begin
2:       active_p := true;
3:       forall r ∈ Neigh_p do Send_p(r, ⟨sets,Inc_p, NInc_p, Leave_p⟩);
4:    end

      Recv_p(q, ⟨sets,Inc_q, NInc_q, Leave_q⟩):
5:    begin
6:       if active_p = false then active_p := true;
7:       Inc_p := Inc_p ∪ Inc_q;
8:       NInc_p := NInc_p ∪ NInc_q;
9:       Leave_p := Leave_p ∪ Leave_q;
10:      Leave_p := Leave_p \ Inc_p;
11:      if Neigh_p ⊆ Inc_p then
12:         NInc_p := NInc_p ∪ {p};
13:      if Inc_p, NInc_p, or Leave_p has changed then
14:         forall r ∈ Neigh_p do Send_p(r, ⟨sets,Inc_p, NInc_p, Leave_p⟩);
15:      if Inc_p = NInc_p ∧ Leave_p = ∅ then
16:         Decide_p;
17:   end
```

Figure 2: Pseudocode triggered by Init_p and Recv_p events.

3.2 The Algorithm

The wave algorithm is event-driven. Actions triggered by an event are assumed to be executed atomically.

The pseudocode triggered by Init_p and Recv_p is shown in Fig. 2.

$Init_p$ event. If process p is an initiator, Init_p occurs spontaneously. When it occurs, process p will set $active_p$ as true and then send Inc_p, $NInc_p$, and $Leave_p$ to all neighbors.

$Recv_p$ event. When process p receives a message from q, p will set $active_p$ as true if $active_p$ is false. Then, Inc_q, $NInc_q$ and $Leave_q$ are inserted into p's versions of these sets. Note that each process both in $Leave_p$ and Inc_p is removed from $Leave_p$. If $Neigh_p$ is a subset of Inc_p, p is inserted into $NInc_p$. If Inc_p, $NInc_p$ or $Leave_p$ has changed, process p sends a message, including Inc_p, $NInc_p$ and $Leave_p$, to all neighbors. Finally, if $Inc_p = NInc_p$ and $Leave_p = \emptyset$, Decide_p is enabled.

```
     LinkDown_p(q):
18:  begin
19:      Neigh_p := Neigh_p \ {q};
20:      if q ∉ Inc_p then
21:      begin
22:          Leave_p := Leave_p ∪ {q};
23:          if Neigh_p ⊆ Inc_p ∧ active_p then
24:          begin
25:              NInc_p := NInc_p ∪ {p};
26:              forall r ∈ Neigh_p do Send_p(r, ⟨sets,Inc_p, NInc_p, Leave_p⟩);
27:          end
28:      end
29:  end

     LinkUp_p(q):
30:  begin
31:      Neigh_p := Neigh_p ∪ {q};
32:      Leave_p := Leave_p \ {q};
33:      if active_p then
34:          Send_p(q, ⟨sets,Inc_p, NInc_p, Leave_p⟩);
35:  end
```

Figure 3: Pseudocode triggered by LinkDown_p and LinkUp_p events.

The pseudocode triggered by LinkDown_p and LinkUp_p is shown in Fig. 3.

$LinkDown_p$ event. When process p senses the failure of a link to a neighboring process q, it removes q from $Neigh_p$. If p hasn't received any message from q, q is inserted into $Leave_p$. In addition, if $Neigh_p$ is a subset of Inc_p and $active_p$ is true, p is inserted into $NInc_p$ and sends Inc_p, $NInc_p$ and $Leave_p$ to all neighbors.

$LinkUp_p$ event. When process p detects a new link to process q, q is inserted into $Neigh_p$ and removed from $Leave_p$. Then, if process p is active, p sends Inc_p, $NInc_p$ and $Leave_p$ to q.

4 Correctness Proof

4.1 Liveness Property: Decision

Lemma 1 *For each active process p, p is eventually in Inc_q of each process q.*

Proof. Since $p \in Inc_p$ initially and the network is consistently connected, p will be propagated to all processes after p becomes active. □

Theorem 1 *If E is an execution in which there is at least one initiator, then E contains at least one decide event.*

Proof. Since there is at least one initiator and the network is consistently connected, every process will receive at least one message and then become active. By lemma 1, $Inc_p = P$ eventually for each process p.

Thus, $Neigh_p \subseteq Inc_p$ eventually and p will be added into $NInc_p$, according to the algorithm. Using the assumption that the network is consistently connected again, p will in $NInc_q$ of each process q. As a result, $NInc_p = P$ eventually for each process p.

Finally, because $Leave_p := Leave_p \setminus Inc_p$ and $Inc_p = P$, $Leave_p = \emptyset$. $Inc_p = NInc_p$ and $Leave_p = \emptyset$ will hold eventually, and therefore p will decide. □

4.2 Safety Property: Dependence

For each process p, $Neigh_{I_p}$ denotes the neighbors of process p in the initial state.

Lemma 2 *For each process p, $Neigh_{I_p} \subseteq Neigh_p \cup Leave_p \cup Inc_p$ is an invariant.*

Proof. Initially, $Neigh_p = Neigh_{I_p}$, so the proposition holds. Then, the actions triggered by events will be considered in turn to show that they preserve the proposition.

Init$_p$: Observe that the actions triggered by Init$_p$ do not change $Neigh_p$, $Leave_p$ and Inc_p, so the proposition is preserved.

Recv$_p$: Line 7, 8, 9 and 10 do not decrease the set of $Neigh_p \cup Leave_p \cup Inc_p$, so the proposition is preserved.

LinkDown$_p$: Line 19 removes q from $Neigh_p$ but q is added to $Leave_p$ if $q \notin Inc_p$ by line 22. Thus, $Neigh_{I_p}$ is still a subset of $Neigh_p \cup Leave_p \cup Inc_p$. The proposition is preserved.

LinkUp$_p$: Line 32 removes q from $Leave_p$ but line 31 adds q into $Neigh_p$. Thus, the proposition is preserved.

□

Lemma 3 *For each process p, if $Inc_p = NInc_p$ and $Leave_p = \emptyset$, then $Inc_p = P$.*

Proof. Since $p \in Inc_p$ and $Inc_p = NInc_p$, $p \in NInc_p$. Thus, $Neigh_p \subseteq Inc_p$. Because $Neigh_{I_p} \subseteq Neigh_p \cup Leave_p \cup Inc_p$ (by lemma 2), $Neigh_p \subseteq Inc_p$ and $Leave_p = \emptyset$, we can get $Neigh_{I_p} \subseteq Inc_p$.

Then, it will be shown that $Neigh_{I_r} \subseteq Inc_p$ for each process $r \in Neigh_{I_p}$. Since $r \in Inc_p$ and $Inc_p = NInc_p$, $r \in NInc_p$. Consequently, $r \in NInc_r$. This implies that $Neigh_r \subseteq Inc_r$. $Neigh_{I_r} \subseteq Neigh_r \cup Leave_r \cup Inc_r$ can be rewritten as $Neigh_{I_r} \subseteq Leave_r \cup Inc_r$. Because $Inc_r \cup Leave_r \subseteq Inc_p \cup Leave_p$ and $Leave_p = \emptyset$, $Neigh_{I_r} \subseteq (Inc_p \cup Leave_p) = Inc_p$.

By the same reasoning and the assumption that the network is connected initially, $Neigh_{I_r} \subseteq Inc_p$ for each process $r \in P$, and therefore $Inc_p = P$. □

Theorem 2 *$Decide_p$ in process p is preceded by an event in each process.*

Proof. Decide_p in process p is enabled only if $Inc_p = NIncp_p$ and $Leave_p = \emptyset$. By lemma 3, $Inc_p = P$. For each process $q \neq p$ in Inc_p, since q is only in Inc_q initially, there exists a message chain from q to p. Thus, there is an event in q causally preceded Decide_p. □

5 Conclusion

We presented a wave algorithm for MANET, based on Finn's algorithm. The message complexity can be improved if the wave algorithm applies to a network of broadcast radios. In our algorithm, whenever Inc_p, $NInc_p$ or $Leave_p$ of process p has changed, p will send a message to all neighbors. This can be done by broadcasting only one message in networks of broadcast radios. Each process within transmission radius of p will receive the same message.

The liveness property of our algorithm is achieved by assuming that the network is consistently connected. A consistently connected network may be partitioned, but there always exists a sequence of consistent links between every pair of nodes. We conjecture that this assumption is necessary for wave algorithms.

References

[1] C.E. Perkins, *Ad Hoc Networking*, Addison-Wesley, 2001.

[2] J. Macker and M.S. Corson, "Mobile Ad Hoc Networking and the IFTF," *ACM Mobile Computing and Communication Review* 2(1), pp.9-14, Jan. 1998.

[3] S.G. Finn, "Resynch Procedures and Fail-safe Network Protocol," *IEEE Trans. Commun.*, vol. COM-27, no.6, pp.840-845, June 1979.

[4] G. Tel, *Introduction to Distributed Algorithms,* Cambridge University Press, 2000.

[5] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *Wireless Networks* 1(1), pp.61-81, 1997.

[6] E. Gafni and D. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Transactions on Communications* C-29(1), pp.11-18, 1981.

[7] Y.B. Ko and V.H. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks," *Proc. of 4th ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, pp. 66-75, 1998.

[8] P. Krishna, N.H. Vaidya, M. Chatterjee and D.K. Pradhan, "A cluster-based approach for routing in dynamic networks," *Proc. of ACM SIGCOMM Computer Communication Review*, pp. 372-378, 1997.

[9] J.E. Walter, J.L. Welch and N.H. Vaidya, "A Mutual Exclusion Algorithm for Ad Hoc Mobile Networks," *Wireless Networks* 7(1), pp. 585-600, Nov. 2001.

[10] N. Malpani, J.L. Welch and N. Vaidya, "Leader Election Algorithms for Mobile Ad Hoc Networks," *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, Aug. 2000.

[11] L. Lamport, "Time, Clock, and the Ordering of Events in a Distributed System," *Commun. ACM*, vol. 21, no. 7, pp.558-565, July 1978.

[12] E.J.-H. Chang, "Echo Algorithms: Depth Parallel Operations on General Graphs," *IEEE Trans. Sofew. Eng.*, vol. SE-8, no. 4, pp. 391-401, July 1982.

[13] A. Segall, "Distributed network ptotocols," *IEEE Trans. Inf. Theory*, vol. IT-29, pp.23-35, 1983.

[14] G. Tel, *Topics in Distributed Algorithms,* vol. 1 of *Cambridge Int. Series on Parallel Computation*, Cambridge University Press, 1991.