A Scalable Mutual Exclusion Algorithm for Mobile Ad Hoc Networks

Weigang Wu, Jiannong Cao, Jin Yang
Department of Computing
The Hong Kong Polytechnic University
Kowloon, Hong Kong
{cswgwu,csjcao,csyangj}@comp.polyu.edu.hk

***Abstract*—Mobile ad hoc networks (MANETs) introduce new challenges to designing algorithms for solving the distributed mutual exclusion (MUTEX) problem. In this paper we propose the first permission-based MUTEX algorithm for MANETs. In order to reduce the messages cost, the algorithm uses the so called "look-ahead" technique, which enforces MUTEX only among the hosts currently competing for the critical section (CS). We propose mechanisms to handle the "doze" mode and "disconnection" of mobile hosts. The constraint of FIFO channel, which is not feasible in MANETs, is also relaxed. Using timeout, a fault tolerance mechanism is introduced to tolerate transient link and host failures. The simulation results show that the proposed algorithm works well under various mobility levels, load levels and system scales, especially when the mobility is high or load level is low. It is also shown that the algorithm has good scalability, especially when most of the requests come from a few active hosts.**

***Keywords*— Algorithm; Distributed Computing; Mobile Ad Hoc Network; Mobile Computing; Mutual Exclusion**

I. INTRODUCTION

The characteristics of mobile computing system, in the aspects of communication, mobility and resource constraints [1][2], make the development of algorithms for solving distributed control problems much more difficult than traditional distributed systems. Mutual exclusion (MUTEX) is one of such problems, where a group of hosts intermittently require to enter the Critical Section (CS) in order to gain exclusive access to the shared resource. A solution to the MUTEX problem must satisfy the following three correctness properties:

- *Mutual Exclusion* (safety): At most one host is allowed to enter the CS at any moment;
- *Deadlock Free* (liveness): If any host is waiting for the CS, then in a finite time some host enters the CS;
- *Starvation Free* (Fairness): If a host is waiting for the CS, then in a finite time the host enters the CS.

Much work has been done on the MUTEX problem in distributed systems and many solutions have been presented [3]. These solutions can be categorized into two classes: *token-based* and *permission-based*. In token-based algorithms, a unique token is shared among the hosts. A host is allowed to enter the CS only if it possesses the token. While in a permission-based algorithm, the host that wants to enter the CS must first obtain the permissions from other hosts by exchanging messages.

During the past several years, efforts have been made to solve the MUTEX problem in mobile computing systems, including both infrastructured mobile networks, which consist of a large number of mobile hosts (MHs) and relatively fewer, but more powerful, mobile support stations (MSSs) [4][6][8] and mobile ad hoc networks (MANETs), which consist of MHs only [5][7][9][12]. In infrastructured mobile networks, the MSSs can act on behalf of the MHs. However, there is no MSS in a MANET, which makes the problem more difficult to solve. All of the proposed algorithms for MANETs are token-based algorithms. Although token-based algorithms have some desirable features, such as that the hosts only need to keep information about their neighbours and few messages are needed to pass the privilege to enter the CS, the fatal problem of token loss makes these algorithms not robust. What is worse, in the MANETs, the mobility and frequent disconnections of MHs make token loss a more serious problem and the maintenance of a tree or ring topology more difficult.

In this paper, we consider another approach—the permission-based approach. We propose the first permission-based MUTEX algorithm for MANETs. Compared with the token-based approach, permission-based algorithms have the following advantages: 1) there is no need to maintain the logical topology to pass the token, and 2) there is no need to propagate any message if no host requests to enter CS. These advantages make the permission-based approach well suitable for MANETs where all the resources, e.g. the network bandwidth and the battery power of the MHs, are limited. A problem of the permission-based approach is the large number of messages to be exchanged between the MHs. Therefore, to design a permission-based algorithm for MANETs, the main objective is to reduce the number of messages. In the proposed message-efficient algorithm, we use the so called "look-ahead" technique [4], which reduces the number of messages exchanged by keeping the hosts' competing status for CS.

Another important problem is to tolerate the link failures (e.g. signal shielded) and host failures (e.g. battery exhausted) which occur very frequently in MANETs. Furthermore, MHs may enter the "doze" mode to save power. These issues have not been adequately addressed by existing algorithms. By using timeout and retransmitting request messages, the effect of intermittent and transient link failures and host failures can be removed. By exchanging messages and adjusting the *Info_set* and the *Status_set* doze mode and disconnection can be handled.

The reminder of the paper is organized as follows. In section 2 we present a brief overview on the related work, describing algorithms for distributed MUTEX for MANETs. Section 3 first defines the system model and the assumptions, and then presents the design of the algorithm proposed in this paper. In section 4, we present the proof of the correctness of the proposed algorithm. The results of the performance

0-7803-9428-3/05/$20.00 ©2005 IEEE.

evaluation are described and discussed in section 5. Finally, section 6 concludes the paper with the description of our future work.

II. Related Works

During the past several years, algorithms for solving the MUTEX problem in MANETs have been proposed. All the algorithms make use of a token circulated along a logical ring or passed in a logical tree consisting of all the MHs.

Baldoni et al [7] presented an algorithm which aims at reducing the meaningless control messages when no host requests to access the CS. Different from general token-circulating algorithms, in the algorithm, the structure of the logical ring is computed on-the-fly, and there is a coordinator for each round. A host needs to send out a request message to the coordinator when it wants to access the CS. Thus, the token needs not to be circulated if there is no request for CS. Under low load conditions, this algorithm can greatly reduce the messages exchanged.

A token-asking algorithm is proposed in [9], which is derived from Raymond's tree-based algorithm [11] with the improvement to handle broken links caused by host mobility. This algorithm defines a structure mapped to the real topology of the network which is represented by a Direct Acyclic Graph (DAG) of token-oriented pointers, maintaining multiple paths leading to the host holding the token. Like in [11], requests are forwarded to the token holder along a path in the DAG. The token is delivered along the reverse path to the requesting host. When a host cannot find a path to reach the token holder because of host movements, it will initiate the update procedure to find a new path to the token holder. When a reverse path is broken, the token holder need to search the requesting host first. The advantage of this algorithm is that it requires hosts to keep information only about their immediate neighbors. In [10] a variation of the algorithm in [9] is presented to eliminate the overhead introduced by the process for searching the requesting host. Instead of that the token holder searches the hosts, when a host detects that there is a failure of an outgoing link, it resends its request.

Malpani et al [12] proposed a parametric token token-based algorithm with many variations. In the algorithm, a dynamic logical ring is imposed on the MHs. The successor of a host in the ring is computed on-the-fly. By applying different polices to determine the successor, the authors presented different variations. Based on the Local-Recency(LR) variation, Chen et al [13] proposed a self-stabilizing MUTEX algorithm for MANETs. The algorithm uses dynamic virtual rings formed by circulating tokens to reflect the changing topology. It requires that the topology be static while the algorithm is converging.

As mentioned before, token-based algorithms have some desirable features, such as that the hosts only need to keep information about their neighbours and few messages are needed to pass the privilege to enter the CS. However, the fatal problem of token loss makes these algorithms not robust. What is worse, in MANETs, the mobility and disconnections of MHs make token loss a more serious problem and the maintenance of a tree or ring topology more difficult.

III. Design of the Proposed Algorithm

Before we describe the proposed algorithm, we first briefly introduce the "look-ahead" technique for distributed MUTEX and discuss the issues in applying the technique to MANETs.

The "look-ahead" technique was first introduced in [4] to solve the MUTEX problem in infrastructured mobile networks. The algorithm is based on the well-known Ricart-Agrawala algorithm [14], in which, when a host wants to enter CS, it sends a request to all the other hosts to collect permissions. Requests for CS are assigned globally unique priorities using Lamport-like timestamps. When a host S_i receives a request, it sends a reply if it is not requesting or if its priority is lower than that of the incoming request; otherwise, it defers the reply. A host enters CS only after it has received a reply from every other host. The algorithm in [4] made a modification to the Ricart-Agrawala algorithm so that, instead of involving all the hosts in the system, MUTEX is enforced only among the hosts which are currently competing for CS. On each host S_i, there are two sets. The $Info\_set_i$ includes the IDs of those hosts which S_i needs to inform when it requests to enter CS, and the $Status\_set_i$ includes the IDs of the hosts which would inform S_i when they request to enter CS. If a host wants to enter CS, it just sends request to the hosts in its *Info_set*. When a host wants to disconnect from the network, it offloads the current values of its data structures to its serving MSS which would then act on be half of the host in the execution of the algorithm.

In this paper, we apply the "look-ahead" technique to design the first permission-based MUTEX algorithm for MANETs. The following issues need to be considered. First and most importantly, there is no MSS for MHs in a MANET, so some additional steps should be taken to ensure that the algorithm can continue execution after a host is disconnected. Second, the algorithm in [4] did not provide methods for some important functions. There is no method for initializing the two key data structures -- the *Info_set* and *Status_set*. Third, the assumption made about a FIFO channel becomes not feasible in MANET because the route between two MHs changes from time to time due to the movements of the MHs. Implementing a FIFO channel in MANET is very costly. Last but not the least, there is no fault tolerance mechanism for handling host failures or link failures.

We propose an efficient method to initialize the *Info_set* and *Status_set*. We also propose algorithms to handle the disconnections and the "doze" mode. When a host wants to disconnect from the network or enters the "doze" mode, it sends message to other hosts. Both the sending and receiving hosts will modify the information maintained in their two sets accordingly. When a host wakes up from the "doze" mode, it can resume the execution immediately without performing any special action. When a host reconnects to the network, it informs other hosts and resumes the execution. To relax the constraint of FIFO channel, we added a variable Q_{req}. In [4], if a host with higher priority receives the REPLY after the REQUEST from one host with a lower priority, the host with lower priority would be put into *Status_set*, and can not get a REPLY forever. With the help of Q_{req}, such request is recorded and the host with lower priority would not be put into

Status_set after the reception of the REPLY. In this way, the problem is fixed. Using timeout, a fault tolerant mechanism is developed to tolerate both link and host failures. A timeout value is set for each request message sent out. Intermittent and transient link failures and host failures are handled by resending request messages when the timeout expires.

A. System Model and Assumptions

A MANET consists of a collection of n autonomous MHs, $S = \{S_1, S_2, \ldots, S_n\}$, communicating with each other through wireless channels. Whether two hosts are connected is determined by the signal coverage range and the distance between the hosts. Each host is a router and the communication between two hosts can be multiple hops. Both link failures and host failures can occur. The topology of the MANET interconnection network can change dynamically due to mobility of hosts and failures of links and hosts.

At any moment of time, each MH is in one of three different states: *normal*, *doze*, and *disconnection*. For the disconnection mode, two different cases are considered: *voluntary disconnection* and *accidental disconnection*. An "accidental disconnection" refers to disconnection aroused by the network failure. Such disconnection occurs more frequently and unpredictably than that in wired networks because wireless communication is highly susceptible to network failure. A MH may sometimes disconnect voluntarily from the network, e.g. to save the battery power. Since the MH knows the disconnection in prior, it can execute a predefined protocol for the distributed algorithm it currently participates in.

Since a distributed system built on a MANET is an asynchronous system, there are no bounds on the processing and message passing time. To provide support for tolerating transient link and host failures, we use the timeout mechanism and message retransmission. We assume that the duration of the link failure and host failure is short and the failure is recovered within the predefined time period for retrying.

B. Data Structures

Each host S_i maintains two sets which are defined below:

Info_set$_i$: an array of the IDs of the hosts to which S_i needs to send the request messages when it wants to enter CS.

Status_set$_i$: an array of the IDs of the hosts which, upon requesting to access CS, would send the request messages to S_i.

To ensure the correctness of the algorithm, the following requirements must be satisfied:

1) $\forall S_i :: Info\_Set_i \quad Status\_Set_i = S;$
   $\forall S_i :: Info\_Set_i \quad Status\_Set_i = \phi$
2) $\forall S_i \forall S_j :: S_i \in Info\_Set_j \Rightarrow S_j \in Status\_Set_i$

Obviously, the requirement 1) guarantees that host Si knows the request status of all the other hosts and there is no redundancy information. The requirement 2) guarantees the consistency among the sets of all MHs.

In addition, each host maintains following data structures:

***ts$_{req}$*:** the timestamp for the request of S_i. It is used as the priority of the request of S_i. It is set to NULL initially. If S_i is not requesting for CS, it is also set to NULL.

***Q$_{req}$*:** the array of the IDs of the hosts which have sent requests to S_i.

***TO$_{req}$*:** the array of timers each associated with the REQUEST message sent to a different host. The timeout values of all the timers are the same.

***T$_{rec}$*:** the timestamp of the last reconnection. It is set to "0" initially.

```
--Executed by the initiator: --
//Generate the Upper Triangular Matrix
Mu
for{ i = 0 to |S|-1}
        for{j= i+1 to |S|-1}
        mij = random(0,1);
        endfor
endfor
broadcast Mu to all other hosts;
```

```
--Executed by all the hosts: -----
//Step 1: Generate the Lower Triangular Matrix Ml
for{ i = 0 to |S|-1}
for{j= 0 to i-1} mij =1-mji;  endfor
endfor
//Step 2: Initialize the Info_set and Status_set
// for host Si//
for  { j= 0 to |S|-1 and j !=i}
     if {mij==0} put Sj into Info_Seti;
     else  put Sj into Status_seti;
endfor
```

Fig. 1 Algorithm for initialization

C. The Proposed Algorithm

1) Initialization of the Info_set and Status_set

As shown in Fig. 1, we provide an algorithm to initialize the *Info_set* and *Status_set* for all the hosts. We use an $n \times n$ matrix M, where n is the number of hosts in the network, to represent the relationships among the hosts. The value of each element of M, M_{ij}, represents the relationship between the pair of hosts S_i and S_j. If $M_{ij} = 0$, the ID of S_j is in the *Info_set* of S_i. If $M_{ij} = 1$, the ID of S_j is in the *Status_set* of S_i. To ensure that the sets of all the hosts satisfy the conditions specified in section III-B, an arbitrary host, say S_0, is selected as initiator of the algorithm. The initial value of each element of M is determined by the initiator.

S_0 generates the upper triangular matrix M_u randomly and broadcasts M_u to all the other hosts. Then, all the hosts, including S_0, set value of each element of the lower triangular matrix M_l to the 2's complement of corresponding element in the upper triangle. Finally, according to the corresponding row in M, each host initializes its *Info_set* and *Status_set*. It is easy to verify that the proposed initialization algorithm can guarantee the specified requirements while the messages to be exchanged are very few.

2) Normal Execution (without Disconnection or Doze)

The pseudo code of the proposed distributed MUTEX algorithm is shown in Fig. 2. All the hosts execute the algorithm.

When a host wants to enter the CS, it first sets ts_{req} to the current time and sends the "REQUEST" messages to all the hosts in its *Info_set*. To tolerate link failure and host failure, a timeout is set in TO_{req} for each request message. The host then waits for the "REPLY" messages from all the hosts in its TO_{req}. If the *Info_set* of the host is empty, it enters the CS immediately.

When a host S_i receives a "REQUEST" message from another host S_j, it records the request in its Q_{req}. If it is not requesting for CS or its priority is lower, S_i sends a "REPLY" message to S_j and removes the record for S_j from Q_{req}. If S_j is in *Status_set$_i$*, S_i moves S_j into *Info_set$_i$* and sends a "REQUEST" message to S_j if S_i is requesting for CS but has a lower priority.

Upon the receiving of a "REPLY" message from host S_j, S_i removes timeout in TO_{req} associated with the original request

message sent to S_j. If S_i finds no request from S_j in its Q_{req}, the ID of S_j is moved into $Status\_set_i$. Then, S_i checks whether TO_{req} is empty. If yes, i.e. all its "REQUEST" messages have been replied, S_i enters CS.

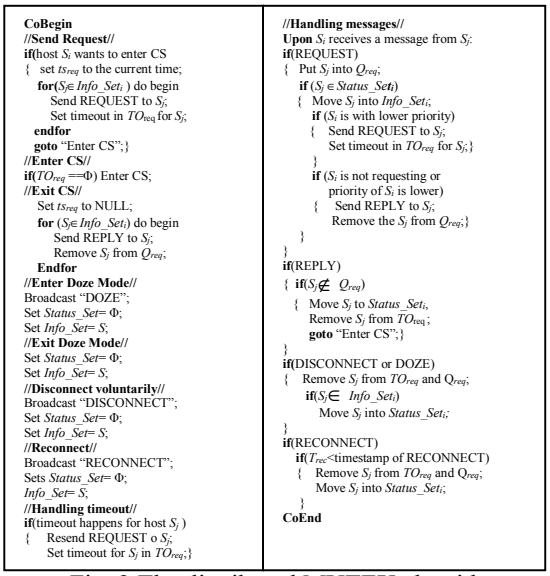

Fig. 2 The distributed MUTEX algorithm

When the timeout for a "REQUEST" message sent to a remote host expires, the requesting host sends the "REQUEST" again. When all the replies for the "REQUEST" message have been received, the requesting host enters CS.

On exiting the CS, a host sends a "REPLY" message to all the hosts in its *Info_set*.

It is worth to notice that if the data structure Q_{req} is not used to record the REQUEST of S_j, S_j may never get a REPLY from S_i. Because the channel may not be FIFO, the REQUEST from S_j may arrive at S_i before the REPLY. When the REPLY arrives, S_i will move S_j to $Status\_set_i$. So, S_i will not send a REPLY to S_j after S_i exits from CS.

3) Handling Doze Mode and Disconnection

When a host S_i wants to enter the "doze" mode, it broadcasts a "DOZE" message to all the other hosts and moves all the hosts in its *Status_set* to its *Info_set*. All the other hosts move S_i into their *Status_set*. No "REQUEST" message needs to be sent to a dozing host. This ensures that the dozing host would not be disturbed. When a dozing host wakes up, it can resume the algorithm without the need to perform any special operation.

If a host S_i wants to disconnect voluntarily, same steps would be taken except that a "DISCONNECT" message, rather than a "DOZE" message, would be broadcasted. When a host S_i reconnects to the network after a disconnection, either a voluntary one or an accidental one, the host needs to broadcast a "RECONNECT" message to inform other hosts and move all the hosts in $Status\_set_i$ into $Info\_set_i$.

When S_j receives a "RECONNECT" message from S_i, S_j compares its T_{rec} with the timestamp of the "RECONNECT" message. If T_{rec} is less (S_i reconnects late indeed), S_j removes the corresponding timeout in TO_{req} if it is waiting for a "REPLY" message from S_i and moves S_i to $Status\_set_i$. The comparison is necessary if more than one host sends out a "RECONNECT" message concurrently. For example, if both S_i and S_j send out "RECONNECT" concurrently, both S_i and S_j move the other to *Status_set* if without comparison the time of reconnection. This violates the requirements for *Status_set* and *Info_set* in section III-B.

IV. Correctness of the Proposed Algorithm

In this section we prove the correctness of the proposed algorithm by showing that the three correctness requirements for distributed MUTEX algorithms are satisfied.

Lemma 1: Based on the assumptions, the effect of transient link or host failures can be eliminated.
Argument: Without loss of generality, we assume that the link between S_i and S_j failed and some message is lost. If neither of the two hosts is waiting for reply from the other, the link failure does not affect them. Otherwise we assume that S_i is waiting for the reply of S_j. Eventually, the timeout for S_j would expire and the request is resent. Since we assume that a link failure is transient and can be recovered within the specified time period for retrying, the S_j eventually received the request after the request is resent one or more times.

Similarly, when a host e.g. S_i, failed, only the hosts waiting for the reply from S_i are affected. Since S_i can recover within the specified time period for retrying, it can eventually receive the request after it reconnects to the network.

Lemma 2: if a host S_i wants to enter CS, it eventually learns about all the hosts concurrently requesting CS.
Argument: For a host S_j in $Status\_set_i$ of host S_i, if S_j wants to enter CS, it will send a "REQUEST" message to S_i. S_i will receive this request even there are failures (Lemma 1). For a host S_k in $Info\_set_i$ of host S_i, S_i sends a "REQUEST" message to S_k. S_k will eventually receive the request (Lemma 1). If S_i receives the reply from S_k, it knows that S_k is not requesting CS. Otherwise, S_i is blocked until S_k send a reply.

Theorem 1: At most one host can be in the CS at any time (safety).
Argument: We prove the theorem by contradiction. Assume two hosts S_i and S_j are executing the CS simultaneously. From Lemma 2, each of them has learned the status of the other, which implies that they had sent reply to each other before they entered the CS. However, this is impossible because no two hosts have the same priority. This is a contradiction.

Theorem 2: The algorithm is deadlock free (liveness).
Argument: A deadlock occurs when there is a circular wait and there is no "REPLY" in transit. This means that each host in the cycle is waiting for a "REPLY" from its successor host in the cycle. Since each request has a distinct priority, there is a host, e.g. S_h who has the highest priority. We denote the successor of S_h as S_j. We claim that host S_h eventually receives one "REPLY" from host S_j. If there is no failure or disconnection, the safety property can be proved in a way similar to that in [4]. Here we only consider the cases with failures and disconnections.

Case 1: S_j runs normally. In this case, S_j would receive the request from S_h and handle it. This can be further divided into

two cases: 1) S_j has no request for CS or its request has a lower priority (because S_h has highest priority). Then S_j would send reply to S_h immediately. 2) S_j is in the CS. Q_{req} is set for S_h, and S_h should be moved to $Info\_set_j$ if it is in $Status\_set_j$ before. After S_j exits from CS, it can send a reply to S_h.

Case 2: S_j failed. If S_j failed before it sent out reply to S_h, it will send the reply after it recovers (Lemma 1). Eventually S_h can receive the reply from S_j.

In all the cases, the circular wait is broken eventually. So the proposed algorithm is deadlock free.

Theorem 3: The algorithm is starvation free (fairness).
Argument: If there is no failure or disconnection, the safety property can be proved in a way similar to that in [4]. Here we only need to consider the situations with failures and disconnections. If there are link failures or host failures, the effect would be eventually eliminated by the timeout mechanism (Lemma 1). If there are disconnected hosts, all the current requests of those hosts would be deleted after reconnection or wake. So failures and disconnections do not affect the fairness of the algorithm. So, the fairness is guaranteed.

V. PERFORMANCE EVALUATION

A. Simulation Parameters and Setup

We adopted Glomosim[15] as the platform which has been widely used for simulating algorithm on MANETs.

We set the parameters of the MANET in the simulation nearly the same values as those used in [7]. All the hosts are scattered into a rectangular territory. However, we varied the number of hosts and corresponding territory scale to evaluate the scalability of the proposed algorithm.

The requests for CS were assumed to arrive at a host according to a Poisson distribution with mean λ which represents the number of requests generated by a single host per second. Our simulation is carried out under three different load levels, i.e. high (λ=1.00E-2), middle (λ=1.00E-3) and low (λ=1.00E-4).

The simulation can be divided into three parts. First, there are only link failures in the network. By simulation, we find that the packet loss rate in the system is about 2%, so we did not simulate link failures by ourselves in this part. Second, we introduced host failures. The arrival of host failures at a host is also assumed to satisfy the Poisson distribution and the duration of host failures satisfies the exponential distribution. To simplify the simulation, we fixed the percentage of host failure to be 10%, a relative high value.

One feature of the algorithm is that the performance is better if arrival of CS requests is localized at few hosts, i.e. some hosts are more active than others, because the *Info_set* of such a host is smaller, and fewer messages would be needed for a request. This feature makes the algorithm scalable to large system. To validate this, we also carried out simulation under the condition that the hosts have different request arrival rates.

To evaluate the effect of mobility, the simulation was carried out under three different mobility settings, chosen by adjusting the pause time so that the percentage of the total simulation time the host does move is 100%, 50% and 10% respectively.

Table 1 parameters of the simulation

Number of Hosts	4, 8, 12, 16, 20
Territory Scale	313m, 443m,543m,626m, 700m
Average speed of movement	20m/sec
Mobility model	Random-waypoint
Transmission radius	200m
Routing-protocol	BELLMANFORD
Link bandwidth	2M bits/Sec

B. Simulation Results and Discussion

We adopt the mostly used metric—number of messages exchanged per CS entry to evaluate the performance of the proposed algorithm. To understand the results well, the concepts of message and hop must be distinguished first. In this paper, "message" means the applications layer message, i.e. the end-to-end message. While the "hop" means the network layer message, i.e. the point-to-point message. Because of the resource constraints of MHs, we are more concerned about the later in a mobile network. So we used two measures-- number of messages necessary per CS entry (MPCS) and number of hops necessary per CS entry (HPCS).

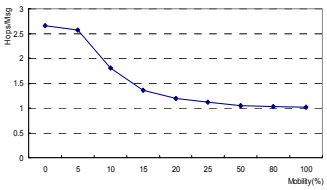

Fig. 4 No. of Hops per message

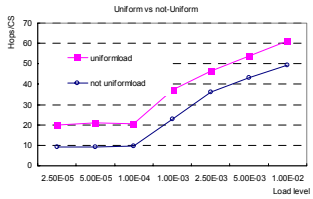

Fig. 5 Effect of host activeness

The number of hops per message is affected by the topology of the network, which had been validated in [7]. In the MANET, the topology is dynamic because of the movement of hosts. So the number of hops per message is affected by the mobility of the hosts seriously. Fig. 4 shows the average number of hops needed for each application level message with 20 hosts. From the figure we can see, with the increase of mobility, the hops decreases. In a MANET, the distance between any two hosts is changed from time to time. The higher the mobility is, the higher the probability with which the distance between any two hosts is short. So under high mobility, the number of hops is little.

The simulation results are discussed as follows. From Fig. 6 and Fig. 7, we can see the effect of the system scale, i.e. the number of hosts. The MPCS or HPCS increases nearly linearly while the system scale increases. This is very easy to understand. More hosts means larger *Info_set* and more concurrent competitors and consequently more request message need to be sent when a host wants to enter CS. The cost does not increase sharply when the system scale increases. So the algorithm is scalable to the system scale. Moreover, when the activeness of hosts is not uniform, the scalability is much better as shown in Fig. 5, where the dashed indicates the performance with uniform load level among all 20 hosts while the real line shows the HPCS when 20% of 20 hosts generate 80% of all requests.

The effect of mobility is shown in Fig. 6. It is interesting to notice that the MPCS/HPCS under low mobility is higher than that under higher mobility. This can be explained using the effect of mobility on the distance between two hosts. Just as discussed before, the higher the mobility is, the shorter the average distance between any two hosts is. So, the connections to send messages can be established more quickly and a host needs to wait for shorter time before entering CS. Short wait period means fewer concurrent competitors, and consequently fewer messages needed. Short distance also induces fewer hops per message.

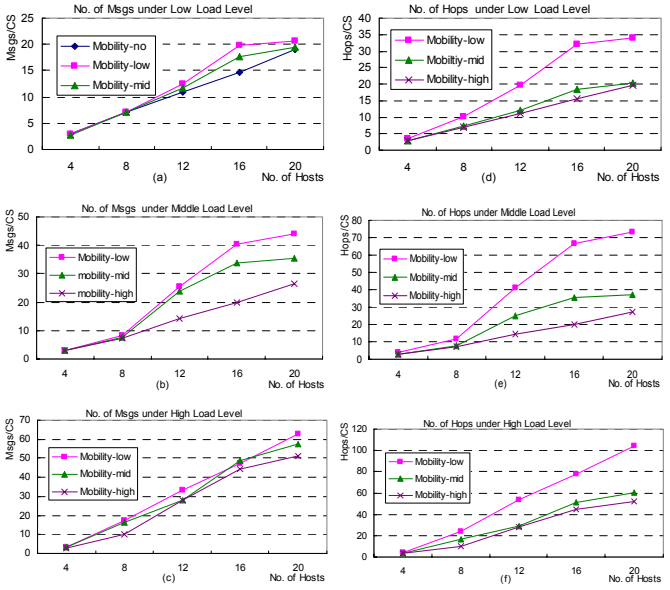

Fig. 6 MPCS/HPCS vs Number of hosts—effect of mobility

Fig. 7 shows the effect of load level and host failures. The MPCS increases with the increase of load level in principle. This can be understood easily. Higher load level means more concurrent competitors, and consequently more messages need to be exchanged. In Fig. 7 the curves named with suffix "-Fail" shows the MPCS with host failure rate 10%. Under different load levels and system scales the increase of MPCS caused by host failures fluctuate dramatically. Under some cases the messages are doubled, but for most cases about 40% more messages needed. Considering the high failure rate (10%), this is acceptable.

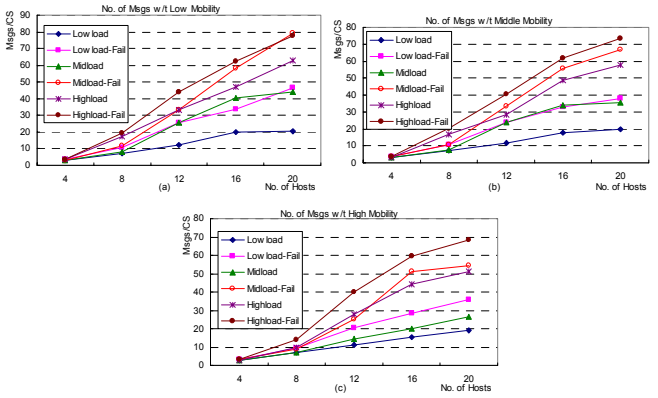

Fig. 7 MPCS vs number of hosts--effect of host failures

VI. CONCLUSIONS

In this paper, we described an efficient and reliable permission-based MUTEX algorithm for MANETs. This algorithm does not depend on any logical topology so as to eliminate the cost of maintaining logical topology. To reduce the number of message exchanged, the "look-ahead" technique is used. We designed a fault tolerance mechanism using timeout to tolerate intermittent and transient link failures and host failures which are very frequent in mobile networks. The algorithm can also handle the "doze" mode and "disconnections" of hosts. The simulation results show that the algorithm performs better under low load level and high mobility. One important feature of the algorithm is the scalability to large system scale especially when some hosts are always more active than others, the performance would be better.

ACKNOWLEDGEMENTS

This work is supported in part by the University Grant Council of Hong Kong under the CERG grant Polyu 5076/01E and China National 973 Program Grant 2002CB312002.

REFERENCES

[1] George H. Forman, and John Zahorjan, The Challenges of Mobile Computing, *Computer*, 1994.

[2] M. Satyanarayanan, Fundamental Challenges in Mobile Computing, *Proc. of PODC*, 1996.

[3] Mukesh Singhal, A Taxonomy of Distributed Mutual Exclusion, *Journal of Parallel and Distributed Computing (18),* 1993.

[4] Mukesh Singhal, and D. Manivannan, A Distributed Mutual Exclusion Algorithm for Mobile Computing Environments, *ICIIS'97*, Dec. 1997.

[5] M. Benchaïba, A. Bouabdallah, N. Badache, M. Ahmed-Nacer, Distributed Mutual Exclusion Algorithms in Mobile Ad Hoc Networks: an Overview, *ACM SIGOPS Operating Systems Review, Volume 38, Issue 1*, 2004

[6] B. R. Badrinath, A. Acharya, and T. Imielinski, Designing Distributed Algorithms for Mobile Computing Networks, *Computer Communications, Vol. 19, No. 4*, April 1996.

[7] Roberto Baldoni, Antonino Virgillito and Roberto Petrassi, A Distributed Mutual Exclusion Algorithm for Mobile Ad-Hoc Networks, *ISCC*, 2002.

[8] L. M. Patnaik, A. K. Ramakrishna, and R. Muralidharan, Distributed Algorithms for Mobile Hosts, *IEE Proc.-Comput. Digit. Tech. Vol. 144, No. 2*, Mar. 1997.

[9] J.E. Walter and S. Kini, Mutual Exclusion on Multihop, Mobile Wireless Networks, *Texas A&M Univ., College Station, TX 77843-3112, TR97-014,* Dec 9, 1997.

[10] J. Walter, J. Welch and N. Vaidya, A Mutual Exclusion Algorithm for Ad Hoc Mobile Networks, *Wireless Networks , Vol. 9, No. 6*, Nov. 2001.

[11] K. Raymond, A Tree-based Algorithm for Distributed Mutual Exclusion, *ACM Transactions on Computer Systems*, 7(1), 1989.

[12] N. Malpani, N. H. Vaidya and J. L. Welch, Distributed Token Circulation on Mobile Ad Hoc Networks, *Technical report, Intel Corporation 505 E. Huntland Dr. Suite 550, Austin TX* 78752.

[13] Y. Chen and J. Welch, Self-stabilizing Mutual Exclusion Using Tokens in Ad Hoc Networks, *Technical Report 2002-4-2, Dept. of Computer Science, Texas A&M Univ.,* April 2002.

[14] G. Ricart and A. K. Agrawala, An Optimal Algorithm for Mutual Exclusion in Computer Networks, *Communication of the ACM*, Jan. 1981

[15] UCLA Parallel Computing Laboratory, GloMoSim Manual v1.2, http://pcl.cs.ucla.edu/