A Hierarchical Consensus Protocol for Mobile Ad Hoc Networks

Weigang Wu[1], Jiannong Cao[1], Jin Yang[1], Michel Raynal[2]
1 Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong
2 IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
{cswgwu, csjcao, csyangj}@comp.polyu.edu.hk，raynal@irisa.fr

Abstract

Mobile ad hoc networks (MANETs) raise new challenges in designing protocols for solving the consensus problem. Among the others, how to design message efficient protocols so as to save resource consumption, has been the focus of research. In this paper, we present the design of such an efficient consensus protocol. We consider the system model for MANETs with host crashes, but equipped with Chandra-Toueg's unreliable failure detectors of class ◊P. *At most f hosts can crash where f < n/2 (n is the total number of the hosts). The protocol adopts the coordinator rotation paradigm to achieve consensus. Unlike existing consensus protocols, the proposed protocol is based on a two-layer hierarchy with hosts associated with proxies. At least f+1 hosts act as proxies and each host is associated with one proxy host. The messages from and/or to the local hosts of the same proxy are merged so as to reduce the message cost. Moreover, the hierarchical approach can improve the scalability of the consensus protocol. Performance analysis shows that the proposed protocol can significantly save cost compared existing protocols.*

1. Introduction

The advent of wireless networking and portable device technologies has engendered the new paradigm of mobile computing. Wireless and mobile networks have properties fundamentally different from traditional wired networks in the aspects of communication, mobility and resource constraints, which make the development of algorithms for solving distributed control problems much more difficult. In this paper, we deal with the consensus problem in the context of mobile computing environment. Consensus is fundamental for many distributed computing applications, e.g. atomic commitment, atomic broadcast, files replication [10][9][13]. We consider mobile ad hoc networks (MANETs), where each mobile host (MH) plays the same role and directly interact with each other. Communications between MHs are peer-to-peer and multi-hops in nature. Also, the topology of a MANET is very arbitrary and can change dynamically.

Informally speaking, the consensus problem is for a set of processes/hosts to agree on a value proposed by one or more of the processes/hosts [1]. In this paper, we use "process" and "host" interchangeably. A process is said to be correct if it behaves according to an agreed specification. Otherwise, a failure is said to occur. There are three correctness properties for a consensus protocol:

i) *Termination*: Every correct process eventually decides some value;
ii) *Agreement*: All the decisions are equal;
iii) *Validity*: Any decision should be equal to the value proposed by at least one process.

If there is no failure, the consensus problem is easy to solve, but in cases of process failures, it becomes very difficult [2][3][4]. As a matter of fact, in asynchronous distributed systems, consensus is unsolvable even with only one host crash [4]. To overcome this impossibility result, Chandra and Toueg introduced unreliable failure detectors (FD) [5]. In their system model, every pair of processes is connected by a reliable communication channel, and the processes can fail by crashing. A FD can be conceived as a distributed oracle that gives (possibly incorrect) hints about which process may have crashed so far. The FDs can be classified according to their accuracy and completeness properties. The accuracy property restricts the mistakes a FD can make, while completeness represents the capacity of suspecting an actually crashed process.

Based on unreliable FDs, some consensus protocols have been proposed [5][10][11][12]. However, the characteristics of mobile computing introduce new challenges [1][2] and the existing consensus protocols for traditional distributed networks need to be adapted or even redesigned for being used in the new environments. Resource constraint, including low bandwidth, limited power supply, low process capability, is one prominent feature of wireless environments. Since fewer messages mean consuming

1066-6192/06 $20.00 © 2006 IEEE

less bandwidth, power and computation resources, one important issue in the design of consensus protocols for mobile environments is to reduce the message cost.

In this paper, we propose a message efficient consensus protocol for MANETs. To our knowledge, this is the first consensus protocol designed for MANETs. The proposed protocol is based on a versatile consensus protocol proposed by Hurfin, Mostefaoui, and Raynal [12] (HMR for short). HMR is extended to a hierarchical approach for accommodating the design requirements for MANETs. In the original HMR protocol, the coordinator of each round sends a proposal message to all the hosts and then each host sends an echo message to the decision makers and agreement keepers. Since each pair of the hosts corresponds to one proposal/echo message, the message cost is very high. To cope with this problem, we introduce a two-layer hierarchy into the consensus protocol. Some hosts in the system are selected to act as the proxies, and each MH is associated with one proxy. Only proxy hosts can be the coordinators, decision makers or agreement keepers. A coordinator sends a proposal message to each proxy host which forwards the proposal to its associated hosts. On the other hand, the echo messages from the hosts associated with the same proxy are merged into one message at the proxy host before being sent to the coordinator. In this way, the message cost can be significantly reduced. However, adding such a hierarchy is not trivial. The messages may be lost or missed due to proxy failures and/or host movements. This may make the HMR protocol invalid. To address this problem, we develop mechanisms to send redeeming messages.

The reminder of the paper is organized as follows. In section 2 we present a brief overview of the related work, describing existing consensus protocols for mobile environments. Section 3 introduces the basis of our work, i.e. the HMR protocol. Following this, in Section 4 we describe our proposed protocol, including system models, data structures and the protocol itself. Section 5 presents the proof of the correctness of the proposed protocol. The result of the performance analysis is reported in section 6. Finally, Section 7 concludes this paper and describes future works.

2. Related works

Although some efforts have been made to solve the consensus problem in mobile computing systems, all the existing protocols are designed for infrastructured networks, where each MH can communicate only with its local mobile support station (MSS).

The protocol in [6] is based on the CT protocol [5]. During the execution of the protocol, each MSS collects initial values from its local hosts and at the same time the CT protocol is executed among all the MSSs. After the MSSs achieve consensus, they propagate the decision to MHs. The main idea of the protocol is that the MSSs act on behalf of the MHs to execute the CT protocol. A handoff mechanism is used to handle the movements of MHs. Since a MH can send its initial value to more than one MSS, the handoff procedure is very simple. When a MH migrates between two MSSs, the new and the previous MSSs just need to update their MH lists and the new MSS will request the MH to send the initial value if it does not know the value yet.

A general framework for solving the consensus problem in infrastructured wireless networks was proposed in [7]. Like in [6], the MSSs act on behalf of the MHs and the consensus protocol is executed among the MSSs. However the dynamism of the set of MSSs was considered. A MSS may join and leave the consensus protocol session when some MHs move into or out of its corresponding cell. A solution is proposed by modeling the dynamism as the group membership problem in which the set of concerned MSSs is regarded as a dynamic group. The two protocols, the membership protocol and the consensus protocol, are executed concurrently while the membership protocol has a higher priority. While the group constitution is changing, the consensus protocol must be temporarily hung up until a stable configuration is reached. Since the group membership problem can also be solved by a consensus protocol [8], the authors indicated that there can be two consensus protocols involved.

All the solutions described above rely on the help of MSSs. The principle is to shift the workload from the MHs to the MSSs. In MANETs, however, there is no MSS and all the works have to be done by individual MHs. In the next section, we propose a message efficient protocol for MANETs.

3. The HMR protocol

As mentioned before, our work is based on the versatile protocol HMR [12]. There are n hosts and the maximum number of hosts that can crash is f ($f < n/2$). The protocol is executed in asynchronous rounds. Each round of the protocol is divided into two phases. HMR presents a unifying approach based on two orthogonal versatility dimensions: the class of the underlying FDs (class S or $\Diamond S$) and the message exchange pattern (from centralized pattern to fully distributed pattern) in each round. Since the protocol proposed in this paper is developed based on the HMR with FDs of class $\Diamond S$, we only describe the HMR with FDs of class $\Diamond S$.

In the first phase of round r, the coordinator host m_{cc} where $cc=coord(r)$, sends its current estimate est_{cc} to each other host with the proposal message $PROP(r,$

est_{cc}). A host m_i ($i \neq cc$) waits for the estimate value from m_{cc} unless m_{cc} is suspected. When a *PROP*(r, est_{cc}) is received, m_i updates its own estimate value est_i and timestamp ts_i, and then enters the second phase. In the second phase, the message exchange pattern is determined by two sets of hosts, *D* and *A*. *D* stands for *Decision-makers*. It is the set of hosts that need to check the decision status, i.e. whether or not they can decide in the current round. The set *A* stands for *Agreement keepers.* Since different hosts may decide in different rounds, *A* is used to ensure that once a value has been decided in a round by some host, no other value can be adopted as the decision value in later rounds.

After entering the second phase of round r, each host sends an echo message *ECHO*(r, est_i, ts_i) to all the hosts in $A \cup D$. Each host m_i in $A \cup D$ waits until it receives *ECHO*(r, est_i, ts) messages from no less than n-f hosts. If m_i is not the coordinator, it sets est_i to the value carried by the *ECHO* message with highest timestamp. Then each host m_d in sets *D* checks to see whether it can make decision in this round. If m_d receives $f+1$ ECHO messages whose timestamps are equal to m_d's current round number r, m_d decides, broadcasts the message *DECISION*(est) using a reliable broadcast mechanism, and then stops participating in the protocol.

4. The proposed protocol

4.1. System model and definitions

The consensus problem is considered in a MANET that consists of a set of n (n>1) MHs, $M = \{m_1, m_2, \ldots, m_n\}$. MHs communicate by sending and receiving messages. Every pair of MHs is connected by a reliable channel that does not create, duplicate, alter, or lose message. A MH can fail by crashing, i.e. prematurely halting, but it acts correctly until crashes. The maximum number of hosts that can crash, f, is bounded by n/2, i.e. $f < n/2$. The system is equipped with an unreliable FD of class $\Diamond P$ which has the following properties:

- *Strong Completeness:* Eventually, every crashed host is permanently suspected by every correct process.
- *Eventually Strong Accuracy*: After some time, correct hosts are not suspected by any correct host.

4.2. Data Structures and Notations

The main data structures and notations used by a MH m_i are listed below.

fl_i: the flag indicating whether m_i has made the decision or not. The initial value of the flag is *false*.

r_i: the serial number of the current round that m_i is participating in.

ph_i: the phase number of the current phase that m_i is participating in.

est_i: the current estimate of the decision value. Initially, it is set to the value proposed by m_i.

ts_i: the timestamp of est_i. The value is the number of the round in which m_i receives the est_i proposed by the coordinator host. The update of ts_i is entailed by the reception of estimate from a coordinator.

4.3. Messages Used in the Proposed Protocol

The messages used in the protocol are classified into the folloing types.

PROP(r, est_{cc}): the proposal message sent from the coordinator to all the other proxy hosts and from a proxy to its local hosts in round r. est_{cc} is the current estimate kept by the coordinator. For each round r, the coordinator tries to impose est_{cc} as the final decision by sending proposal messages.

ECHOL(r, est_i, ts_i): the echo message from m_i to its local proxy host in the round r. est_i is the estimate of m_i and ts_i is the timestamp of est_i.

ECHOG(r, v, ts_v, x, y): the echo message from one proxy host to all the other proxy hosts in the round r. *ECHOG*(r, v, ts_v, x, y) is constructed by merging the ECHOL messages from its local hosts. v is the estimate carried by the ECHOL with the highest timestamp and ts_v is the timestamp of v. x is the set of the hosts that send the ECHOL with ts_v. y is the set of the hosts that send ECHOL with other timestamps.

LEAVE(r, sn): the informing message sent to the local proxy by a host which wants to disassociate from the current local proxy. sn is the serial number to distinguish different LEAVE messages from the same host.

JOIN(r_i, sn): the message sent by a common host to its new proxy during handoff. sn is the serial number to distinguish different JOIN messages from the same host.

DECISION(est): the message broadcasted by a host that has made decision. est is the decision value.

PROPH(r, est_{cc}): same as a PROP message except that this proposal is for handoff procedure.

4.3. The Protocol

A two-layer logical hierarchical structure is imposed on the network of MHs:

Proxy layer: consists of a set *P* of MHs which act as proxy hosts to exchange messages on behalf of other hosts. Only the hosts in set *P* can be the coordinators. To guarantee the termination of protocol, at least one correct host is included. So, *P* contains at least f+1 MHs.

Host layer: consists of a set *M* of all the MHs, including those in set *P*.

P can be initialized randomly or according to some measurement, e.g. the load level and/or power level of a MH. Then each host chooses the nearest unsuspected host in P as its proxy. A host in P chooses itself all the time. The host that is associated with a proxy is called "*local host*" of the proxy host and correspondingly, the proxy is called "*local proxy*" of its local hosts.

```
--------------------------------Task 1: Consensus-------------------------------
// The code executed by each host, $m_i$
BEGIN:
(1) $r_i \leftarrow 0$; $est_i \leftarrow v_i$; $ts_i \leftarrow 0$; $fl_i \leftarrow false$;
(2) while ($fl_i \neq true$){
(3)     $r_i \leftarrow r_i+1$; $ph_i \leftarrow 1$;
        ---------- Phase 1 of round $r_i$: from $m_{cc}$ to all proxies -------------
        // let $cc$ denote $coord(r_i)$,
        //$P$ denote the set of proxies and $p$ denote the local proxy of $m_i$
(4)     if($i=cc$) send $PROP(r_i, est_i)$ to $P$;
        if($i \in P$) {
(5)         wait until ($PROP(r_i, est_{cc})$ is received or $p_{cc} \in suspected_i$);
(6)         if($PROP(r_i, est_{cc})$ message received from $p_{cc}$)
                 broadcast ($PROP(r_i, est_{cc})$ locally;
(7)         else broadcast ($PROP(r_i, \perp)$ locally;
           // $\perp$ is a value can not be accepted;
        }//endif
(8)     wait until $PROP(r_i, v)$ from $p$ is received or $p$ is suspected;
(9)     if($PROP((r_i, v)$ is received and $v \neq \perp$){$est_i \leftarrow v$; $ts_i \leftarrow r_i$;}
        -----------Phase 2 of round $r_i$: from all to $P$ -------------------------
        $ph_i \leftarrow 2$;
(10)    send message $ECHOL(r_i, est_i, ts_i)$ to $P$;
        if ($m_i \in P$) {
(11)        wait for $ECHOL(r_i, est_j, ts_j)$ from each local host $m_j$ or
                      $m_j \in suspected_i$;
(12)        merge the ECHOL messages{
                  $ts_v \leftarrow$the highest timestamp of all the ECHOL;
                  $v \leftarrow$ the estimate of the ECHOL with highest timestamp;
                  $x \leftarrow$ the set of the hosts that send ECHOL with $ts_v$;
                  $y \leftarrow$ the set of the hosts that send other ECHOL;
            }
(13)        send $ECHOG(r, v, ts_v, x, y)$ to $P$;
(14)        wait until (($\cup x \cup y$)of $ECHOG(r_i, v, ts_v,x,y)$includes
                 at least $n$-$f$ hosts ) or  (an $ECHOG(-, -, >r_i,-,-)$ received);
(15)        if($i \neq cc$) $est_i \leftarrow$the $est$  received with the highest $ts$;
(16)        if(($m_i \in P$) $\wedge$($ECHOG$ messages with ($ts= r=r_i$) represent
                  at least ($f+1$) hosts){
                  $fl_i \leftarrow true$;
(17)              $\forall j \neq i$: send $DECISION(est_i)$ to $m_j$;
             }//endif
          }//endif
      }//endwhile
---------------------------Task 2:  Reliable broadcast -------------------------
(18) upon reception of $DECISION(est)$ from host $m_k$:
       $fl_i \leftarrow true$; $\forall j \neq i, k$: send $DECISION(est)$ to $m_j$;
END
```

Fig. 1 the proposed protocol—Task 1and Task 2

The proposed protocol consists of four tasks. We first describe the Task 1 (achieving consensus) and Task 2 (reliable broadcast of the decision), which correspond to the two tasks in HMR respectively. The pseudocode of Task 1 and Task 2 is shown in Fig. 1.

Task 1 consists of two phases. At the beginning of round r, the current coordinator p_{cc} sends $PROP(r, est_{cc})$ to the hosts in set P. A proxy p waits for the proposal message from p_{cc}. If the $PROP(r, est_{cc})$ message is received, p sends the message to all its local hosts; otherwise if p suspects host p_{cc} before receiving $PROP(r, est_{cc})$, p sends a $PROP(r, \perp)$ message to its local hosts, where "$\perp$" is a value that can never be proposed or adopted. A host m_i waits until a $PROP(r, -)$ message is received from its local proxy or the local proxy is suspected. The symbol "–" in the message means any possible value. If a $PROP(r, v)$ message with $v \neq \perp$ is received, m_i updates its estimate value to v and timestamp to r. If the local proxy is suspected, m_i invokes the handoff procedure, which will be presented late. Then Phase 1 is finished.

In Phase 2, each host first sends an echo message $ECHOL(r_i, est_i, ts_i)$ to its local proxy. If the host itself is not a proxy, it enters the next round r+1. Each proxy waits for an echo message $ECHOL(r, -, -)$ from each of its local hosts if the host is not suspected. Then each proxy constructs an echo message by merging the collected $ECHOL(r, -, -)$ messages. v is the estimate value carried by the $ECHOL(r, -, -)$ message with the highest timestamp and ts_v is the timestamp. x is the set of the hosts that send the $ECHOL(r, -, -)$ messages with ts_v. y is the set of the hosts that send $ECHOL(r, -, -)$ messages with other timestamps. The proxy host then sends the echo message $ECHOG(r, v, ts_v, x, y)$ to all the other proxy hosts. Each proxy waits for the $ECHOG$ messages from other proxies until: 1) the $ECHOG(r,-,-,-,-)$ messages received can "represent" not less than (n-f) hosts, or 2) an $ECHOG(-,-,ts_v,-,-)$ with $ts_v>r$ is received. Here, the "represent" means a host is included in the set x or y of the ECHOG message. A proxy updates its estimate to the value carried by the ECHOG message with the highest timestamp, but the timestamp is not changed. Finally, a proxy host checks whether it can decide in the current round. If there are (f+1) or more hosts in the x sets of the $ECHOG(r, v, ts_v, x, y)$ messages with $ts_v=r$, it can make the decision and broadcasts the final value.

Task 2 is the simple broadcast mechanism. When a host, which has not decided, receives a DECISION message, it makes decision and forwards the DECSION message to all other hosts except the sender.

Besides the two tasks corresponding to the tasks in HMR, two additional tasks are added in the proposed protocol: handoff and handling late ECHOL messages. Fig. 2 shows the pseudocode of these two tasks.

The handoff procedure is invoked when a host m_i suspects its current proxy p or p is no longer the nearest proxy. Let q be the new proxy. First, m_i sends a leave message $LEAVE(r_i, sn)$ to p and a join message $JOIN(r_i, sn)$ to q. Upon reception of the leave message, p deletes m_i from local host list. Upon reception of the join message, q adds m_i to local host list. If a $PROP(r_q, est_{cc})$ has been received, q sends $PROPH(r_q, est_{cc})$ to m_i; otherwise q sends $PROPH(r_q, \perp)$ to m_i.

```
--------------------Task 3: Handling Late ECHOL------------------------
// The code executed by each proxy $p$;
while ($fl_i \neq true$){
(19)  wait for $ECHOL(r,v,ts)$ with ($r<r_i$);
      construct a ECHOG for the ECHOL and send it to $P$;
}//endwhile
-----------------------------------Task 4: Handoff: -------------------------
// A host $m_i$ need to change its local proxy $p$
      --------Task 4.1: Handoff code executed by a host $m_i$ ------------
(21) while($fl_i \neq true$ and ($p \in suspected_i$ or $p$ is not the nearest one)) {
(22)     $q \leftarrow$the nearest correct proxy;
(23)     send a $LEAVE(r_i, sn)$ to $p$;
(24)     send $JOIN(r_i, sn)$ to $q$;
(25)     wait until a $PROPH(r_p, v)$ is received or $q \in suspected_i$;
         if ($PROPH(r_p, v)$ is received){
             if($r_i < r_p$){
(26)             $r_i \leftarrow r_p$;
(27)             for($ts_i \leq rr < r_i$) send $ECHOL(rr, est_i, ts_i)$ to $q$;
(28)             if($v \neq \perp$){ $est_i \leftarrow v$; $ts_i \leftarrow r_i$;}
(29)             GOTO (10); //resume the normal execution;
             }else if ($r_i = r_p$){
                 if($ph_i$ =1){
(30)                 for($ts_i \leq rr < r_i$) send $ECHOL(rr, est_i, ts_i)$ to $q$;
(31)                 if($v \neq \perp$){$est_i \leftarrow v$; $ts_i \leftarrow r_i$;}
(32)                 GOTO (10); //resume normal execution;
                 }else if ($ph_i$ =2){
(33)                 for($ts_i \leq rr \leq r_i$) send $ECHOL(rr, est_i, ts_i)$ to $q$;
(34)                 $r_i \leftarrow r_i+1$; GOTO (4);//resume normal execution;}
             }else if($r_i > r_p$){
                 if($ph_i$ =1){
(35)                 for($ts_i \leq rr < r_i$) send $ECHOL(rr, est_i, ts_i)$ to $q$;
(36)                 GOTO (4); //resume normal execution; }
                 else if ($ph_i$ =2){
(37)                 for($ts_i \leq rr \leq r_i$) send $ECHOL(rr, est_i, ts_i)$ to $q$;
(38)                 $r_i \leftarrow r_i+1$; GOTO (4);//resume normal execution;}
             }//endif
         }//endif
     }//endwhile
     ----------Task 4.2: Handoff code executed by a proxy $p$---------
     while($fl_i \neq true$){
         Upon reception of $LEAVE(r_i, sn)$ from host $m_i${
(39)             delete $m_i$ from local host list;}
         Upon reception of $JOIN(r, sn)$ from host $m_i$ {
             add $m_i$ to local host list;
(40)         if($PROP(r_p, est_{cc})$ received from $p_{cc}$)
                 send $PROPH(r_p, est_{cc})$ to $m_i$;
(41)         else send $PROPH(r_p, \perp)$ to $m_i$;}
     }//endwhile;
```

Fig. 2 the proposed protocol—Task 3 and Task 4

Upon reception of the $PROPH(r_q, w)$ message from q, the behaviours of host m_i can be classified into 3 cases. 1) **($r_i < r_q$) or ($r_i = r_q$, ph_i =1):** m_i updates round number to r_q and sends $ECHOL(rr, est_i, ts_i)$ to q where $ts_i \leq rr < r_q$. If $w \neq \perp$, m_i sets its estimate to w and timestamp to r_q. m_i then resumes the normal execution by entering phase 2 of round r_q. 2) **($r_i > r_q$, ph_i =1):** m_i sends $ECHOL(rr, est_i, ts_i)$ to q where $ts_i \leq rr < r_i$ and then resumes the normal execution by continue the phase 1 of round r_i. 3) **($r_i = r_q$, ph_i =2) or ($r_i > r_q$, ph_i =2):** m_i sends $ECHOL(rr, est_i, ts_i)$ to q where $ts_i \leq rr \leq r_i$ and then resumes the normal execution by entering the next round r_i+1.

Another task added is handling the late ECHOL messages. An ECHOL message that arrives at a proxy after the proxy has sent out an ECHOG message for the corresponding round is a "late" ECHOL message. This happens when a proxy p suspects a correct local host or a host m_i joins a new proxy host with a round number greater than the ts of m_i. The hosts in set P may be blocked forever if a late ECHOL message is ignored. To avoid this, when a proxy p receives an $ECHOL(r_i, est_i, ts_i)$ with ($r_i < r_p$) or ($r_i = r_p$ but p has sent out a ECHOG for the round r_i), p constructs a redeeming ECHOG for m_i and sends it to all the proxy hosts.

5. Correctness

The validity property of the proposed protocol is obvious. The proofs for the termination property and agreement property are given in this section. In the proof, we use "*indirect suspicion*" to refer to the situation that a host itself does not suspect the current coordinator but it receives a $PROP(r, \perp)$ from its proxy.

5.1. Termination

Lemma 1. *If no host decides in the round $r' \leq r$, then all the correct hosts enter the round r+1.*

Proof. The proof is by contradiction. We assume that no host decided in a round r' with $r'<r$, where r is the smallest number of a round in which a correct host m_i is blocked forever. m_i can only be blocked in a wait statement, i.e. line 5, line 8, line 25, line 11 or line 14. We analyze the cases for these lines one by one.

Case 1: m_i is blocked at line 5. Obviously, $m_i \in P$. If $i = cc$, m_i can not be blocked (it receives the proposal message sent by itself). If $i \neq cc$, then either p_{cc} crashes or p_{cc} is correct. In the former case, m_i eventually suspects p_{cc} ; in the later case, m_i receives the proposal message from p_{cc} eventually. Therefore, m_i can not be blocked forever at line 5.

Case 2: m_i is blocked at line 8. If m_i is a proxy, it is the local proxy of itself, Since m_i can not be blocked forever at line 5 and m_i can receive the $PROP(r,-)$ message sent by itself at line 6 or 7. If m_i is not a proxy, on the other hand, there are two possible situations: the local proxy crashes or not. If the local proxy does not crash, it eventually sends out a $PROP(r, -)$ message (the proxy can not be blocked at line 5 forever) and m_i eventually receives it. If the local proxy crashes, m_i eventually suspects it and invokes a handoff procedure. So, m_i can not be blocked at line 8 forever.

Case 3: m_i is blocked at line 25. Obviously, the handoff procedure has been invoked. There are two possible cases. If the local proxy crashes, m_i eventually suspects it and invokes the handoff procedure again. Since there are no less than f+1 hosts in set P and at most f hosts can crash, m_i can eventually find a correct proxy (by the eventually strong accuracy of underlying FDs). This case turns to be the second one. In the second case, the local proxy does not crash, so it eventually sends out a *PROPH*(r,-) message to m_i (no host is blocked at line 5 forever) and m_i eventually receives it. So m_i can not be blocked at line 25 forever.

Case 4: m_i is blocked at line 11. m_i is a proxy and is waiting for the ECHOL messages from its local MHs. All its local hosts can be categorized into three classes: crashed hosts, correct hosts that have left m_i and other hosts. For the crashed hosts, m_i eventually suspects them and no longer waits for them. For the correct hosts that have left, each of them must have sent a leave message to m_i before it left (line 23). m_i eventually receives the leave message and stops waiting for the corresponding host. For the other local hosts, since they can not be blocked at line 5, line 8 or line 25, m_i eventually receives an ECHOL from each of them. So, m_i can not be blocked at line 11 forever.

Case 5: m_i is blocked at line 14. Obviously $m_i \in P$. There are two possible conditions to unblock m_i: 1) m_i receives ECHOG messages that can represent no less than n-f hosts; 2) m_i receives an ECHOG message with timestamp ts>r. We now prove that at least one of the two conditions is satisfied eventually. Since at most f hosts can crash, there are at least n-f correct hosts. Since r is the smallest round in which a correct host is blocked forever, all these n-f correct hosts eventually proceed to the round r and execute line 10 in round r eventually. Then we categorize all the correct hosts into two classes: i) the hosts with correct proxies when they execute line 10 in round r; ii) the hosts with incorrect proxies when they execute line 10 in round r. For a host m_j in class i), the local proxy of m_j eventually receives m_j's ECHOL message and includes m_j in an ECHOG message to m_i. For a host m_j in class ii), after the local proxy of m_j crashed, m_j eventually suspects the proxy and invokes the handoff procedure. As proved before, m_j eventually find a correct proxy p after one or more handoff and it eventually receives a *PROPH*(r_p,-) message from p. Then, we consider different situations according to the ts_j: a) if $ts_j \leq r$, an *ECHOL*(r,est_j, ts_j) is sent to p at line 27, 30, 33, 35 or 37; b) if ts_j>r, an *ECHOL*(-,-,>r) is sent to p at line 27, 30, 33, 35 or 37. Considering p is a correct host, it eventually includes m_j in an ECHOG to m_i. If some host belongs to b), m_i eventually receives an *ECHOG*(-,-,>r,-,-) and consequently condition 2) is satisfied; otherwise all the n-f correct host belong to i) or a), and m_i eventually receives enough *ECHOG*(r,-,-,-,-), i.e. the condition 1) is satisfied. So, m_i can not be blocked at line 14 forever. □

Lemma 2. *For any round r, if the coordinator* c_r *sends out a PROP(r, v) at time tr and less than n-f hosts suspect* c_r *directly or indirectly in Phase 1 of r, then no PROP(r', v) with r'>r can be sent out before tr.*

Proof. In proving the lemma, "v" is a value not equal to ⊥. The proof is by contradiction. We assume that at least one *PROP*(r', v) with r'>r has been sent out by the time tr. Let rm be the greatest round number of all the *PROP*(r', v) messages issued out by time tr, then rm>r and rm-1≥ r. Obviously the coordinator of round rm, c_{rm} must have finished line 14 in round rm-1. Since rm is the greatest round number in the *PROP*(r', v) messages, and the timestamp of the estimate at any host can only be changed at line 9, 28 or 31, c_{rm} can not receive a ECHOG with ts>rm-1 in round rm-1. So, at least n-f hosts sent out *ECHOL*(rm-1,-,-) in round rm-1 before time tr, which means that at least n-f hosts finished *phase* 1 of round rm-1 before time tr. Since rm-1≥ r, at least n-f hosts finished *phase* 1 round r before the *PROP(r,v)* is sent out in round r. So at least n-f hosts suspected c_r directly or indirectly in the *phase* 1 of round r, which is a contradiction to the assumption in the beginning. So the lemma holds. □

Corollary 1. *In any round r, if the coordinator of r+1,* c_{r+1} *receives an ECHOG message with ts>r, then at least n-f hosts directly or indirectly suspect* c_{r+1} *in round r+1.*

Proof. In the proof, the "v" is a value not equal to ⊥. In any round r, the timestamp ts of the estimate at any host can only be updated to r at line 9, 28 or 31, which means a *PROP*(r, v) has been sent out by the coordinator and delivered by the local proxy before the update. By the assumption, the c_{r+1} receives an ECHOG with ts>r at line 14 in round r, so some host must has sent out the *PROP*(ts, v) before c_{r+1} finishes line 14 of round r. Then we consider the status of c_{r+1} in round r+1. 1) c_{r+1} crashes before it sends out *PROP*(r+1, v) in round r+1, all the correct hosts (at least n-f hosts) suspect it eventually in round r+1. The corollary obviously holds. 2) c_{r+1} sends out *PROP*(r+1, v) in round r+1. In this case, so some host sent out a *PROP*(ts, v) with ts>r before *PROP*(r+1, v) is sent out. By Lemma 2, at least n-f hosts suspect c_{r+1} in round r+1. The corollary holds. □

Theorem 1. *If a host is correct, it eventually decides.*

Proof. If one host decides, all correct hosts eventually decide due to the reliable broadcast mechanism (line 17 and 18). So, we just prove "at least one host decides".

The proof is by contradiction. We assume that no host decides. According to the accuracy and completeness of the underlying FD, there is a time t after which all the correct hosts are never suspected by any correct host and all the crashed hosts are permanently suspected by every correct host. After time t, there is at least one correct host, say m_x, in set P, and every correct host is associated with a correct proxy. Let r be the first round coordinated by m_x and started after t. By the assumption (no process decides) and the Lemma 1, all the correct hosts eventually enter the round r. Since no new suspicion occurs after time t and at most f hosts can crash, there are at least n-f correct hosts execute the round r. By Corollary 1, m_x can not receive a ECHOG with ts>r at line 14, so m_x, eventually decide in round r, which contradicts the assumption that “there is no process decides”. So the theorem holds. □

5.2. Agreement

Lemma 3. *Let r with $r \geq 1$ be the first round in which (f+1) hosts send ECHOL(r, v, r) and r' be any round that $r' \geq r$. We have:*

1) No host decides before r;

2) If the coordinator of r' sends a proposal message, then this message carries the estimate v (i.e. the message is PROP (r', v)).

Proof. Proof for 1): We prove it by contradiction. If no host decides at line 16, no host can decide at line 18, so we only consider the decision at line 16. We assume that some host m_j decided at line 16 in some round s before r, i.e. $s < r$ and the decision value is u. To decide at line 16 in round s, m_j has to receive enough ECHOG messages carrying a timestamp equal to s and the union set of the x sets carried by the ECHOG message includes at least f+1 hosts. Since all the ECHOG messages are constructed based on ECHOL messages, at least f+1 *ECHOL*(s, u, s) must have been sent out. From the definition of r (“…first round in which…”), we have $r \leq s$, which contradicts to s<r. So the part 1) holds.

Proof for 2): In any round r, the timestamp ts of the estimate at any host can only be changed to r at line 9, 28 or 31, which means that a *PROP*(r, v) has been sent out by the coordinator of round r. So, c_r must have sent out the *PROP*(r, v) message before the f+1 *ECHOL*(r, v, r) messages are sent and at least f+1 hosts did not suspect c_r in *phase* 1 of round r. Let tp be the moment that the coordinator c_r sent out the *PROP*(r, v) message. Since n-(f+1)<n-f, by Lemma 2, all the *PROP*(r', -) messages with r'>r must be sent out after time tp. Let R be the list of the round numbers of all the *PROP*(r', -) messages with r'>r. Without loss of generality, we assume R=(r_0=r, r_1, r_2, $r_{3,\dots}r_{i,\dots}$), where the round numbers are sorted in the ascending order of the moments when they are sent out. Now, we prove that for each round r_i listed in R, the proposal value carried by *PROP*(r_i, u) is equal to v, i.e. u=v. The proof is by induction on the serial number i.

Base case: i=0 and i=1. If i=0 the lemma obviously holds. For i=1, we consider the c_{r1} at line 14 of round r. By the definition of R, r_1 is the first round that a coordinator sends out *PROP*(r',-) message with r'>r, there is no ECHOG message with ts>r before c_{r1} finishes the execution of line 14 in round r, i.e. r is the highest timestamp then. So, c_{r1} must receive ECHOG messages with round number r and representing at least n-f hosts. By the assumption in the lemma, at least f+1 hosts sent *ECHOL(r, v, r)*, so at least one *ECHOL*(r, v, r) has been merged into a ECHOG message received by c_{r1} and the ECHOG is with timestamp r and estimate value v. At the end of line 14, c_{r1} updates its estimate to the value carried by the ECHOG with the highest timestamp, i.e. v. The lemma holds.

Inductions case: Let us assume that the lemma holds for any round r_i such that $0 \leq i \leq k$, we show that the lemma holds for round r_{k+1}. By the induction assumption, for each *PROP*(r_i, w) message with $0 \leq i \leq k$ we have w=v. Now, we define two sets of hosts.

- The set G includes all the hosts that have received a *PROP*(r_i, w) or *PROPH*(r_i, w) message with $0 \leq i \leq k$. $\forall m_j \in G$: est_j =w= v and ts_j=r_i that $0 \leq i \leq k$. Since f+1 hosts send *ECHOL*(r, v, r), so $|G| \geq f$+1.
- The set B includes the hosts m_j that have not received a *PROP*(r_i, w) message with $0 \leq i \leq k$. Consequently, $\forall m_j \in B$: $ts_j < r$. So, all the timestamps of the hosts in set B are less than those of the hosts in set G.

Now we consider the behaviours of host c_{rk+1} in phase 2 of round (r_{k+1})-1. Since $c_{rk+1} \in P$, it waits for the ECHOG messages at line 14. There are two conditions to end the execution of line 14. 1) c_{rk+1} receives an *ECHOG*(-, u, tsm, -, -) with tsm >(r_{k+1})-1. Then c_{rk+1} updates its estimate to the value u at line 15. In fact the value u must come from a *ECHOL*(-, u, tsm), so the sender of this ECHOL must have received a *PROP*(tsm, u). By the definition of R and the induction assumption, we have $tsm \in \{r_0, .., r_k\}$, so u=v. 2) c_{rk+1} receives ECHOG messages with round number (r_{k+1})-1 that can represent at least n-f hosts, i.e. at least n-f *ECHOL*((r_{k+1})-1, -, -) messages are merged. Let X denote the hosts that sent these ECHOL messages. Obviously, $|X| \geq n$-f. At line 15, c_{rk+1} updates its estimate to a value u which is carried by the ECHOG with the highest timestamp tsm. Of course, this u comes from a *ECHOL*((r_{k+1})-1, u, tsm) message. Since $|X| \geq n$-f and $|G| \geq f$+1, we have $G \cap X \neq \varphi$. So, the *ECHOL*((r_{k+1})-1, u, tsm) message must be sent by a host in G. By the definition of G, we have u=v. For the both case 1) and 2), the estimate value of c_{rk+1} is updated to v in round

1066-6192/06 $20.00 © 2006 IEEE

(r_{k+1})-1 and consequently in round r_{k+1}, c_{rk+1} sends out a *PROP*(r_{k+1}, *v*). So the lemma holds for the round r_{k+1}. □

Theorem 2. *No two hosts decide differently.*

Proof. If a host decides a value at line 18, then this value must have been decided by another host at line 16. So we only consider values decided at line 16.

Let m_i be a host that decides v_i in the round r_i. As m_i decides in r_i, it received *ECHOG*(r_i, v_i, r_i, -, -) messages, which means the coordinator of round r_i sent a *PROP*(r_i, v_i). Similarly, if another host m_j decides on the value v_j in round r_j, the coordinator of round r_j must have sent *PROP*(r_j, v_j). Let *r* be the round characterized in Lemma 3 (the first round in which *f*+1 hosts send *ECHOL*(*r*, *v*, *r*)). By Lemma 3, we have $r \leq r_i$, $r \leq r_j$, so v=v_i=v_j. □

Table 1 the message cost of HMR and the proposed protocol

	k/n=5%			k/n=10%			k/n=20%			k/n=33%			k/n=40%			k/n=50%		
n	NH_{HMR}	NH_{Hier}	NH_{Hier}/NH_{HMR}	NH_{HMR}	NH_{Hier}	NH_{Hier}/NH_{HMR}	NH_{HMR}	NH_{Hier}	NH_{Hier}/NH_{HMR}	NH_{HMR}	NH_{Hier}	NH_{Hier}/NH_{HMR}	NH_{HMR}	NH_{Hier}	NH_{Hier}/NH_{HMR}	NH_{HMR}	NH_{Hier}	NH_{Hier}/NH_{HMR}
4	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	24	20	83%
8	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	88	52	59%	101	61	61%	120	76	63%
12	N/A	N/A	N/A	N/A	N/A	N/A	146	85	58%	215	101	47%	250	132	53%	301	175	58%
16	N/A	N/A	N/A	N/A	N/A	N/A	269	128	48%	405	167	41%	474	232	49%	576	320	56%
20	N/A	N/A	N/A	N/A	N/A	N/A	432	179	42%	663	251	38%	778	364	47%	951	515	54%
24	242	220	91%	374	197	53%	638	239	38%	990	354	36%	1166	530	45%	1431	763	53%
28	323	258	80%	512	237	46%	888	308	35%	1391	477	34%	1642	731	45%	2019	1066	53%
32	416	297	72%	672	280	41%	1184	385	33%	1867	620	33%	2208	968	44%	2720	1424	52%
48	911	460	51%	1555	474	30%	2842	791	28%	4557	1405	31%	5415	2293	42%	6702	3447	51%
64	1613	634	39%	2842	709	25%	5299	1357	26%	8576	2549	30%	10214	4255	42%	12672	6464	51%
96	3667	1013	28%	6701	1308	19%	12770	3000	24%	20861	5985	29%	24907	10217	41%	30976	15680	51%
128	6630	1438	22%	12365	2087	17%	23834	5361	23%	39125	11035	28%	46771	19047	41%	58240	29376	50%

6. Performance analysis

In this section we analyze the performance of the proposed protocol in comparison with the HMR protocol in respect of the message cost. Since the two protocols rely on FDs of class ◊*P* and class ◊*S* respectively, we first compare these two classes of FDs.

6.1. ◊*P* vs ◊*S*

Among all the eight classes, ◊*S* (eventually strong) is the weakest one (◊*W* is equivalent to ◊*S*) but strong enough to be used to solve the consensus problem [5][15]. Nearly all the existing protocols are based on the class ◊*S* FDs. It has been proved that class ◊*P* and class ◊*S* are equivalent in the power of solving the consensus problem [16]. However, many people have implemented the FDs of class ◊*P* [5][17]. Though ◊*P* is stronger than ◊*S*, the existing implementations of ◊*P* are not more complex than those of ◊*S*. The difference is that ◊*P* may take more time to reach a stable state. Though ◊*P* can not tolerate more failures than ◊*S*, we can still use it to design more efficient protocols.

6.2. Message Cost

First, in a MANET, the concepts of “message” and “hop” must be distinguished. In traditional distributed systems, the performance is computed in terms of the number of messages, where one “message” means one “end-to-end” message. However, one message may take one or more hops to reach the destination in the underlying network. One “hop” means one network layer message, i.e. the point-to-point message. In traditional systems, messages that cost different number of hops are regarded as messages with the same cost. However, in a MANET, where the resource constraint is serious, the number of hops can reflect the message cost more precisely. Since a MANET can be represented by a graph, the average number of hops of an end-to-end message is related to the diameter of the graph. We adopt the value log^n [14] as the average number of hops of an end-to-end message.

Since it is impossible to analyze the total number of rounds of the execution, we just consider the message cost per round. To make the HMR comparable with the proposed protocol, let $|D \cup A|$ = $|P|$ = *k*. Of course, *k* should be greater than *f*, i.e. *f*<*k*. Let NH_{HMR} and NH_{Hier} denote the number of hops per round in HMR protocol and the proposed protocol respectively. Obviously, NH_{HMR} = $(n+n*k)*log^n$. For the proposed protocol, NH_{Hier} depends on distance between the common hosts and proxies. Let *l* be the average number of hops of one message between a host and its proxy. In the first phase, the number of hops is $k*log^n+n*l$; in the second phase, the number of hops is $n*l+k^2*log^n$. Then we have NH_{Hier} = $2n*l+(k^2+k)*log^n$. We regard each proxy and its local hosts as a sub-network, we have $l = log^{(n/k)}$, where *n*/*k* is the number of hosts in a sub-network. Then, NH_{Hier}=$2n*$ $log^{(n/k)}+(k^2+k)*log^n$ = $(2n+k^2+k)*log^n$ - $2n*$ log^k The difference is:

$$\Delta = NH_{HMR} - NH_{Hier} = (n+n*k)*log^n - ((2n+k^2+k)*log^n - 2n*log^k) = (n*k-n-k^2-k)*log^n + 2n*log^k$$

Table 1 shows numerical evaluation of NH_{HMR} and NH_{Hier} under various conditions. The percentage of proxy hosts varies from 5% to 50%. To show the advantage of the proposed protocol clearly, the ratio of NH_{Hier} to NH_{HMR} is also computed. From the table we can see that the proposed protocol can save message cost significantly. The larger the system scale (the number n) is, the more the cost saved is. This is easy to understand. When the system scale is large, each proxy has many local hosts and consequently many messages are merged together. So, more cost is saved. This feature shows that the scalability of proposed protocol is very good.

The parameter k also affects the performance but the effect is more complicated. Roughly, when the k is in the medial, the advantage of the proposed protocol is great. If k is very little, one proxy needs to take care of many hosts. The number of hops of a message between proxy and local hosts becomes great and accounts for most part of the total cost. Even though some cost is saved by reducing the global messages, the total cost is not reduced much. On the other hand, if k is great, few hosts are associated with one proxy. So, few messages can be merged like the situation where the n is little.

In the discussion above, the overhead of handoff and late ECHOL messages is not considered. Obviously this is hard to analyze theoretically. However, as mentioned in many papers, most of the cases are good cases, where no or few hosts crash during the execution of consensus protocol. Such overhead should be little.

7. Conclusions

In this paper, we proposed the first consensus protocol for MANETs. The protocol is based on Chandra-Toueg's unreliable FDs of class $\Diamond P$. It is assumed that at most f hosts can crash where $f < n/2$ (n is the total number of the hosts). The coordinator rotation paradigm is adopted to achieve consensus. To reduce the message cost, we introduced a two-layer hierarchy into the protocol. At least f+1 hosts act as proxies and each host is associated with a proxy host. A coordinator only needs to send one proposal message to each proxy host and the proxy host forwards the proposal to its local hosts. On the other hand, the echo messages from the local hosts of one proxy host are merged into one message and sent to the coordinator. So, the message cost is reduced significantly. Moreover, the hierarchy can improve the scalability of the consensus protocol. All these features make the protocol suitable for MANETs. The performance analysis shows that the proposed protocol can save message cost significantly.

In future, we will carry out extensive simulations to evaluate the performance of the proposed protocol in dynamic environments. The overhead caused by handoffs and late ECHOL messages would be included. We will also extend the protocol with the help of "clustering" algorithms, so as to make the protocol adaptable to system states, e.g. load level, power level.

Acknowledgements

This research is partially supported by Hong Kong University Research Grant Council under the CERG grant PolyU 5075/02E and Hong Kong Polytechnic University under the ICRG grant A-PF77.

References

[1] G. Coulouris, J. Dollimore, and T. Kindberg, Distributed Systems: Concepts and Design (third edition), *Addison-Wesley*, 2001.

[2] N. A. Lynch, Distributed Algorithms, *Morgan Kaufmann*, 1996.

[3] M. Fischer, The Consensus Problem in Unreliable Distributed Systems, *Research Report YALE/DCS/RR-273, Yale Univ*., 1983.

[4] M. J. Fischer, N. A. Lynch, and M. S. Paterson, Impossibility of Distributed Consensus with One Faulty Process, *J. of the ACM*, Apr. 1985.

[5] T. D. Chandra and S. Toueg, Unreliable Failure Detectors for reliable distributed Systems, *journal of the ACM*, Mar. 1996.

[6] N. Badache, M. Hurfin, and R. Macedo, Solving the Consensus Problem in a Mobile Environment, *Proc. of IPCCC*, 1999.

[7] H. Seba, N. Badache, A. Bouabdallah, Solving the Consensus Problem in a Dynamic Group: an Approach Suitable for a Mobile Environment, *Proc. of ISCC*, 2002.

[8] R. Gueraoui, A. Schiper, The Generic Consensus Service, *IEEE Transactions on Software Engineering*, Jan. 2001.

[9] R. Guerraoui, M. Huifin, et al., Consensus in Asynchronous Distributed Systems: A Concise Guided Tour*, LNCS 1752*, 2000.

[10] M. Hurfin, and M. Raynal, A Simple and Fast Asynchronous Consensus Protocol Based on a Weak Failure Detector, *Distributed Computing*, Sep 1999.

[11] A. Schiper, Early Consensus in an Asynchronous System with a Weak Failure Detector, *Distributed Computing*, Oct 1997.

[12] M. Hurfin, A. Mostefaoui, and M. Raynal, A Versatile Family of Consensus Protocols Based on Chandra-Toueg's Unreliable Failure Detectors, *IEEE Trans. on Computers*, Apr 2002.

[13]R. Guerraoui, and A. Schiper, Consensus: the Big Misunderstanding, *the Sixth IEEE Workshop on Future Trends of Distributed Computing Systems*, 1997.

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

[15] T. Chandra, B. Hadzilacos, and S. Toueg, The Weakest Failure Detector for Solving Consensus, *J. of ACM*, vol.43(4), July 1996.

[16] R. Friedman, A. Mostefaoui, M. Raynal, On the Respective Power of $\Diamond P$ and $\Diamond S$ to solve One-Shot Agreement Problems, *Technical Report of IRISA*, No. 1547, July 2003.

[17]M. Larrea, A. Fernandez, and S. Arevalo, On the Implementation of Unreliable Failure Detectors in Partially Synchronous Systems, *IEEE Trans. on Computers*, vol.53(7), July 2004.