Design and Performance Evaluation of Efficient Consensus Protocols for Mobile Ad Hoc Networks

Weigang Wu, Jiannong Cao, *Senior Member*, *IEEE*, Jin Yang, and Michel Raynal

Abstract—Designing protocols for solving the consensus problem faces new challenges in mobile computing environments. Among others, how we can achieve message efficiency for saving resource consumption has been the focus of research. In this paper, we present the HC protocol, a message efficient consensus protocol for MANETs. We consider the widely used system model where the hosts fail by crashes and the system is equipped with Chandra-Toueg's unreliable failure detectors. Unlike existing consensus protocols, the HC protocol uses a two-layer hierarchy based on clusters to achieve message efficiency. The messages from and to the hosts in the same cluster are merged so as to reduce the message cost. However, adding such a hierarchy is not trivial. Due to host movements and failures, the hierarchy changes from time to time and this may cause message loss. In designing HC, we also propose methods to handle such message losses. Extensive simulations have been carried out to evaluate and compare the performance of the HC protocol and similar protocols in a MANET environment. Simulation results show that, in most cases, our protocol can significantly reduce both the message cost and time cost. With increases in the system scale or the percentage of faulty hosts, the advantage of our protocol becomes more obvious.

Index Terms—Consensus, mobile ad hoc network, mobile computing, distributed algorithm, failure detector, fault tolerance.

1 Introduction

Mobile wireless networks have properties fundamentally different from traditional wired networks in the aspects of communication, mobility, and resource constraints, which make the design of distributed algorithms much more difficult than in traditional distributed systems. Resource constraint, for example, low bandwidth, limited power supply, or low process capability, is one of the prominent features of mobile environments [2], [12]. Fewer messages consume less bandwidth, power, and computation resources, so reducing message cost is a very important issue in the design of distributed algorithms for mobile wireless environments.

In this paper, we design efficient protocols to solve the consensus problem in the context of a mobile computing environment. Consensus is a fundamental problem for many distributed computing applications, for example, atomic commitment, atomic broadcast, and file replication [14], [15], [16]. Broadly speaking, the consensus problem involves getting a set of processes to agree on a value proposed by one or more of the processes [7], [21]. In a distributed system, especially a mobile network, processes are prone to failure. A process is said to be correct if it behaves according to an agreed specification in a run of a consensus protocol; otherwise, a failure occurs and the process is said to be faulty. More precisely, a correct solution to a consensus problem should have three correctness properties:

1. *Termination*. Every correct process eventually decides upon some value.
2. *Agreement*. All of the decision values are equal.
3. *Validity*. Any decision value should have been proposed by at least one process.

Unfortunately, it has been proven that, in asynchronous distributed systems, consensus cannot be solved deterministically, even with only one process crash [11]. To overcome this impossibility result, Chandra and Toueg introduced unreliable failure detectors (FDs) [5]. An FD gives (possibly incorrect) hints about which process may have crashed so far. It is constituted by modules local to each process and periodically consulted by the corresponding process. FDs can be classified according to their accuracy and completeness properties. The accuracy property restricts the mistakes that an FD can make, whereas the completeness property represents the capacity of suspecting an actually crashed process. Among all eight classes of FD, $\diamond S$ is the weakest but strong enough to solve the consensus problem [4], [5]. $\diamond S$ is defined using the following properties:

- *Strong completeness*. Eventually, each crashed process is permanently suspected by each correct process.
- *Eventually weak accuracy*. There is a time after which some correct process is not suspected by any correct process.

- *W. Wu, J. Cao, and J. Yang are with the Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong. E-mail: {cswgwu, csjcao, csyang}@comp.polyu.edu.hk.*
- *M. Raynal is with IRISA, Campus de Beaulieu, Université de Rennes 1, Avenue du Général Leclerc, 35042 Rennes Cedex, France. E-mail: raynal@irisa.fr.*

Manuscript received 17 Apr. 2006; revised 9 Nov. 2006; accepted 12 Feb. 2007; published online 10 Apr. 2007.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0149-0406.
Digital Object Identifier no. 10.1109/TC.2007.1053.

Many consensus protocols based on FDs have been proposed for distributed systems [5], [17], [18], [27]. However, the characteristics of mobile networks introduce additional challenges in the design of consensus protocols for the new environments.

There are two major types of mobile networks: *infrastructured networks* and *mobile ad hoc networks* (MANETs). The infrastructured network consists of two distinct sets of entities: a large number of mobile hosts[1] (MHs) and relatively fewer but more powerful mobile support stations (MSSs). Each MH can only communicate with its local MSS. Some protocols [1], [28] have been proposed to solve the consensus problem in infrastructured mobile networks. The principle of designing these protocols is to shift the operations of achieving consensus from MH-based to MSS-based.

In a MANET, however, there is no MSS and each MH plays the same role. Communications between MHs are peer-to-peer and multihop in nature. Also, the topology of a MANET is very arbitrary and can change dynamically. The principle used in designing consensus protocols for infrastructured networks is not applicable in MANETs, where all the work must be done by the MHs themselves. Although some probabilistic protocols for MANETs have been proposed [6], [30], to our knowledge, no work has been reported on deterministically achieving consensus in MANETs. Of course, protocols for traditional networks can be used in MANETs, but they are not efficient in terms of the message cost, especially for large-scale MANETs [23], [32].

In this paper, we propose a message-efficient consensus protocol for MANETs, which is named the "Hierarchical Cluster-based" (HC) protocol. The HC protocol is based on a variant of the versatile protocol proposed by Hurfin et al. (HMR) [17], extending it to a hierarchical approach for accommodating the design requirements of MANETs. A two-layer hierarchy is imposed on the system by clustering MHs into clusters. Clustering has been widely used in MANETs to achieve message efficiency, stability, and scalability. In the HC protocol, some hosts are selected to act as clusterheads and each MH is associated with one clusterhead. All messages to or from MHs are forwarded by clusterheads. The messages with the same type are merged by a clusterhead before they are forwarded to other hosts. Similarly, when some messages of the same type need to be sent to some hosts in the same cluster, these messages are also first merged and then sent to the destination clusterhead, which will unmerge these messages and deliver them. This way, the message cost can be significantly reduced.

However, adding such a hierarchy is not trivial. First, the messages are not simply forwarded by the clusterhead: A cluster member needs to synchronize with its clusterhead in the message exchange step. Due to the mobility and clusterhead failure, an MH may need to switch between clusterheads that are executing different steps. Therefore, the switching procedure should be delicately handled in order to maintain the synchronization between an MH and its clusterhead. Second, nearly all consensus protocols, including the CT protocol [5], HMR protocol [17], and BHM protocol [1], require that no message can be lost. However, the change of the hierarchy in a MANET may cause message losses, even if the communication channel is reliable. To distinguish such message losses caused by the dynamics of the hierarchy from those caused by lossy channels, we use two different terms: "switching message loss" and "transmitting message loss." To cope with switching message losses, some redeeming messages should be sent. What and when "redeeming messages" should be sent depends on the execution state of the MH and the clusterhead. We develop efficient mechanisms for handling redeeming messages.

Besides the design of the HC protocol, another contribution of the paper is the performance evaluation of consensus protocols in a MANET environment. We have conducted extensive simulations to quantitatively evaluate the performance of different consensus protocols, including our proposed one. Through the evaluations, we obtained insights into the effects of the main parameters on the performance and the features of different protocols in the MANET environment. This should be the first paper that quantitatively and directly evaluates the performance of consensus protocols by simulations. Such a simulation study is valuable to the application of these protocols in a real network.

The rest of the paper is organized as follows: In Section 2, we briefly review existing FD-based consensus protocols, including a variant of the HMR protocol, which is the basis of our work. Following this, we describe our proposed protocol in Section 3, including the system model, the data structures, and the protocol itself. Section 4 proves the correctness of the HC protocol. The performance evaluation by simulations is reported in Section 5. We describe the extension of our protocol to the context of lossy channels in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

Several $\diamondsuit S$-based consensus protocols have been proposed for traditional fixed networks [5], [18], [27]. They all have the same assumption that a majority of the processes are correct, which has been proved to be a necessary condition for achieving consensus in an asynchronous system [5]. All of the protocols are based on the rotating coordinator paradigm and executed in asynchronous rounds. A round is usually divided into several phases. Each process has an estimate of the final decision value. During each round, a predetermined coordinator process attempts to impose its own current estimate on others by sending its estimate to all of the processes. On the reception of the proposal from the coordinator, a process updates its own estimate and sends the echo message to some or all of the processes. Based on the echo messages received during a round, a process can update its estimate and determine if it can make a decision. These protocols mainly differ in the message exchange pattern in a round. The protocol presented in [5] adopts a centralized message exchange pattern, whereas the protocols in [18], [27] use the fully distributed pattern. The latter two protocols differ in the way in which they cope with failures and the mistakes made by the underlying FD. More precisely, the protocol in [18] "trusts" the FD, whereas the protocol in [27] does not.

1. In this paper, the terms "process" and "host" are used interchangeably.

The HMR [17] protocol presents a unifying approach of achieving consensus with two orthogonal versatility dimensions: the class of the underlying FD (class S or $\diamondsuit S$) and the message exchange pattern (from a centralized pattern to a fully distributed pattern) in each round. Similar to other FD-based protocols, HMR uses the rotating coordinator paradigm and executes in asynchronous rounds. In the first phase of a round, the coordinator attempts to impose its own current estimate on others by sending its estimate to all the processes. However, in HMR, in the second phase of a round r, the exchange pattern of echo messages is determined by two sets: D and A. D (*decision_makers*) is the set of hosts that needs to check the decision status, that is, whether they can decide in the current round. A (*agreement_keepers*) is used to ensure that, once a value has been decided upon in some round, no other value can be decided on in later rounds. In the second phase, each host sends an echo message to all of the hosts in $A \cup D$.

Compared with other existing consensus protocols, HMR is simple but versatile. There are only two types of messages involved and the message processing in each phase is very simple. These features make HMR suitable for MANETs, where the system resources, including battery power, memory space, and so forth, are very scarce. Moreover, due to versatility, HMR can give rise to a large and well-identified family of FD-based protocols [17]. However, HMR is not efficient in terms of message cost. In each round, the coordinator needs to send the same proposal to every MH and each MH needs to send the same echo to every *decision_maker/agreement_keeper*. Therefore, we propose the two-layer hierarchy to improve the message efficiency.

All of the above protocols rely on reliable communication channels between hosts, that is, the channels that do not create or duplicate messages and will eventually deliver every message unless the receiver crashes. The protocols reported in [8], [22] can tolerate transmitting message losses. To do so, each host needs to periodically resend the latest message that it has sent and, if a message sent from a higher round is received, a host skips the current round.

Researchers have proposed some consensus protocols for mobile environments. Based on the CT protocol [5], Badache et al. [1] proposed the BHM protocol for infrastructured mobile networks. The basic idea of BHM is to let the MSSs achieve consensus for the MHs. MSSs collect initial values from their local hosts and execute the CT protocol to make the decision on some value collected. After the MSSs achieve consensus, they propagate the decision value to the MHs. A simple handoff mechanism is used to handle the movements of MHs.

To some extent, the hierarchy in our protocol is similar to that in the BHM protocol. The clusterheads (similar to MSSs in BHM) act as privileged hosts and do more work than ordinary MHs. However, there is great deal of difference between BHM and HC. In BHM, the MHs only provide initial values and do not participate in the protocol for achieving consensus, but, in our HC protocol, all of the hosts need to execute the rounds of message exchange and processing. In terms of the function of privileged hosts, the HMR protocol (or the CT protocol) and BHM protocol are two extremes and our HC protocol is in between. More importantly, our protocol is designed for MANETs, where no infrastructure of MSSs exists. Although the clusterheads play some privileged roles, they are also MHs interconnected in an ad hoc manner.

The protocol in [28] extends the BHM protocol by considering the dynamics of the set of MSSs. Using a group membership protocol, the MSSs of the cells without any MH are deleted from the set of MSSs executing the consensus protocol. Since the group membership problem can also be solved using a consensus protocol [16], there can be two consensus protocols involved, which are executed concurrently. The solutions in [1] and [28] 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 of the work has to be done by the MHs themselves.

Chockler et al. [6] developed a partition-based consensus protocol for MANETs. The network is divided into nonoverlapping grids, each of which is a single-hop subnetwork. Single-hop consensus is first achieved within each grid and, then, each host gossips its grid consensus value. A host can decide after it has received a value from every grid. Another consensus protocol for MANETs is reported in [30]. Several fault-tolerant broadcast algorithms for MANETs are designed and applied to a randomized consensus protocol [10], which can only probabilistically guarantee the termination property by using a random number generator.

Both of the protocols in [6] and [30] are probabilistic with respect to their approaches of achieving a global consensus in MANETs. In this paper, we consider the message-efficient protocol that can deterministically guarantee the termination property.

3 The HC Protocol

3.1 System Model

The consensus problem is considered in a MANET that consists of a set of n $(n > 1)$ MHs, with $M = \{m_1, m_2, \ldots, m_n\}$. All MHs are distributed into clusters. Some of the MHs are selected as clusterheads and each is in charge of one cluster. The number of clusterheads is denoted by k. An MH can only fail by crashing, that is, prematurely halting, but it acts correctly until it possibly crashes. An MH that crashes in a run is a faulty host; otherwise, it is correct. The maximum number of faulty MHs in a run, denoted by f, is bounded by k and $n/2$, that is, $f < minimum(k, n/2)$.

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 a message. It is important to notice that the assumption on reliable channels can be reduced to one on lossy channels, which is more feasible for MANETs, but requires a much more complicated design. This is discussed in Section 6. For simplicity, we assume reliable channels in the description of our HC protocol.

The system is equipped with an unreliable FD of class $\diamondsuit P$, which is defined using the following properties:

- *Strong completeness*. Eventually, each crashed process is permanently suspected by each correct process.

- *Eventual strong accuracy*. There is a time after which every correct process is not suspected by any correct process.

As mentioned before, among all eight classes of FDs proposed by Chandra and Toueg, $\diamondsuit S$ is the weakest, but it is strong enough to solve the consensus problem [4], [5]. $\diamondsuit P$ has a stronger accuracy property than $\diamondsuit S$, but it has been proven that $\diamondsuit P$ and $\diamondsuit S$ are equivalent in the power of solving the consensus problem [13]. Although $\diamondsuit P$ is stronger than $\diamondsuit S$, existing implementations of $\diamondsuit P$ [5], [20] are not more complex than those of $\diamondsuit S$. Of course, $\diamondsuit P$ may take a longer time to reach a stable state.

Our protocol uses $\diamondsuit P$ instead of $\diamondsuit S$ because the eventually strong accuracy property is necessary to guarantee the termination. There are two necessary conditions to guarantee the termination of our HC protocol. First, there is at least one correct host to act as a clusterhead. This can be satisfied by including more than f MHs in the set of clusterheads. Second, after some time, all of the correct clusterheads must be no longer suspected by any correct MH. The reason is twofold: 1) It can guarantee that at least one correct and unsuspected clusterhead can act as the coordinator and 2) it can prevent an MH from endless clusterhead switches due to false suspicions. However, an FD of class $\diamondsuit S$ can only guarantee that at least one correct host (may not be a clusterhead host) is never suspected after some time. Therefore, $\diamondsuit P$ is used to satisfy the second condition (see the proof in Section 4 for more details).

3.2 Data Structures and Message Types

When executing the protocol, each host m_i needs to maintain necessary information about its state. Such information is stored in the following variables:

- fl_i: The flag indicating whether m_i has made the decision.
- r_i: The sequence number of the current round in which m_i is participating.
- ph_i: The phase number of the current phase in which m_i is participating.
- 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 sequence number of the round in which m_i receives est_i from the coordinator. The update of ts_i is entailed by the reception of estimate from a coordinator.

The message types involved in the proposed protocol are listed as follows:

- $PROP(r, est_{cc})$: The proposal message sent from the coordinator to the clusterheads or from a clusterhead to the hosts in its cluster. est_{cc} is the current estimate kept by the coordinator. In each round, the coordinator tries to impose est_{cc} on other hosts by sending proposal messages.
- $ECHOL(r, est_i, ts_i)$: The echo message from m_i to its clusterhead in round r.
- $ECHOG(r, v, ts_v, x, y)$: The echo message from a clusterhead to other clusterheads in round r. It is constructed by merging the $ECHOL$ messages from the hosts in the same cluster. v is the estimate carried by the $ECHOL$ message with the highest timestamp and ts_v is the timestamp of v. x is the set of MHs that sends the $ECHOL$ message with ts_v, whereas y is the set of MHs that sends other $ECHOL$ messages.
- $DECISION(est)$: The message sent by an MH to broadcast the decision value est.
- $JOIN(r_i, sn)$: The message sent by an MH to the clusterhead of a new cluster that the MH wants to join. sn is a sequence number to distinguish $JOIN$ messages from the same host.
- $LEAVE(r, sn)$: The message sent by an MH to its clusterhead to inform the clusterhead that the MH wants to disassociate itself from the current cluster. sn is the sequence number to distinguish different $LEAVE$ messages from the same host.
- $PROPH(r, est_{cc})$: This is same as a $PROP$ message, except that this is for an MH that a new cluster joins.

3.3 Operations of the Protocol

A two-layer hierarchy is imposed on the network of MHs. The *Clusterhead layer* consists of a predefined set H of MHs which act as clusterheads to merge/unmerge and forward messages for MHs. The *Host layer* consists of a set M of all MHs, including those in set H.

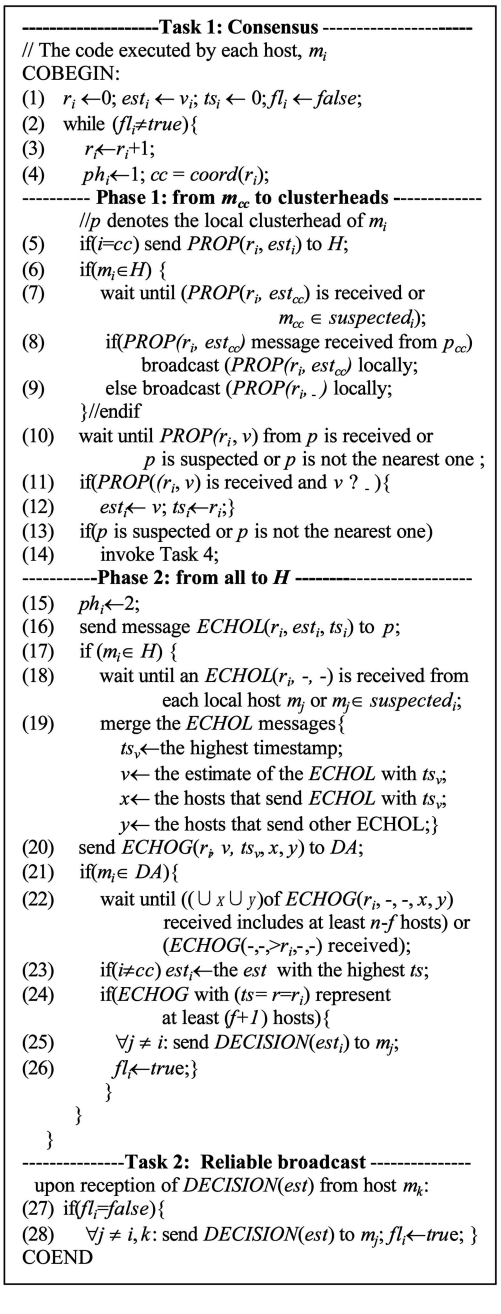
Only the hosts in set H can act as coordinators or *decision_makers/agreement_keepers*. To guarantee the termination of the protocol, at least one correct host should be included in H, that is, $|H| = k \geq f + 1$. Each host associates itself with the nearest[2] unsuspected clusterhead in H. The distance between two hosts is defined as the path length in hops. Such distance information can be obtained through the underlying routing protocol, which is in charge of the establishment and maintenance of the path between any two hosts.[3] Obviously, a clusterhead host always chooses itself as its clusterhead. The hosts that choose the same clusterhead constitute a cluster. A host associated with a clusterhead is called a "*local host*" of the clusterhead and, correspondingly, the clusterhead is called the "*local clusterhead*" of its local hosts.

To balance the workload and energy consumption, MHs can take turns serving as the clusterhead for different runs of the protocol. Since the only requirement for forming the set H is that at least one clusterhead is correct, it does not matter that a crashed host appears in H. Although H remains unchanged for each run of the protocol, it can be periodically reformed, for example, by using a predetermined function so that an ordinary MH can switch to being a clusterhead and vice versa.

The proposed protocol consists of four tasks. Like most existing consensus protocols, Task 1 is the main body of the protocol for making decisions and Task 2 is a simple broadcast algorithm for propagating the value decided upon. Other two additional tasks need to be performed in our protocol. Task 3 is used to handle late $ECHOL$ messages arriving at a clusterhead and Task 4 is used for a host to switch its cluster. The pseudocode of Tasks 1 and 2

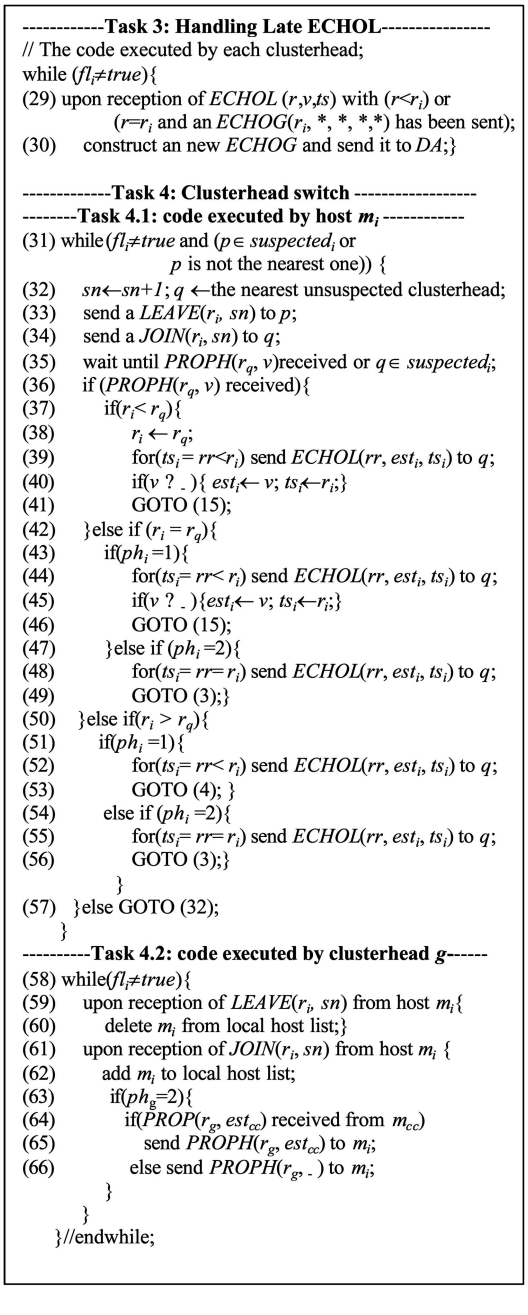
2. A threshold of the difference between the distances to the old and new clusterheads can be set.

3. For geographical routing protocols, the "distance" can be defined as the geographical distance between two hosts.



```
----------Task 1: Consensus ----------
// The code executed by each host, m_i
COBEGIN:
(1)  r_i ←0; est_i ← v_i; ts_i ← 0; fl_i ←false;
(2)  while (fl_i≠true){
(3)    r_i←r_i+1;
(4)    ph_i←1; cc = coord(r_i);
---------- Phase 1: from m_cc to clusterheads ----------
       //p denotes the local clusterhead of m_i
(5)    if(i=cc) send PROP(r_i, est_i) to H;
(6)    if(m_i∈H) {
(7)      wait until (PROP(r_i, est_cc) is received or
                                  m_cc ∈ suspected_i);
(8)      if(PROP(r_i, est_cc) message received from p_cc)
             broadcast (PROP(r_i, est_cc) locally;
(9)      else broadcast (PROP(r_i, _) locally;
       }//endif
(10)   wait until PROP(r_i, v) from p is received or
             p is suspected or p is not the nearest one ;
(11)   if(PROP((r_i, v) is received and v ? _ ){
(12)     est_i← v; ts_i←r_i;}
(13)   if(p is suspected or p is not the nearest one)
(14)     invoke Task 4;
----------Phase 2: from all to H ----------
(15)   ph_i←2;
(16)   send message ECHOL(r_i, est_i, ts_i) to p;
(17)   if (m_i∈ H) {
(18)     wait until an ECHOL(r_i, -, -) is received from
             each local host m_j or m_j∈ suspected_i;
(19)     merge the ECHOL messages{
           ts_v←the highest timestamp;
           v← the estimate of the ECHOL with ts_v;
           x← the hosts that send ECHOL with ts_v;
           y← the hosts that send other ECHOL;}
(20)   send ECHOG(r_i, v, ts_v, x, y) to DA;
(21)   if(m_i∈ DA){
(22)     wait until ((∪ x ∪ y)of ECHOG(r_i, -, -, x, y)
             received includes at least n-f hosts) or
             (ECHOG(-,-,->r_i,-,-) received);
(23)     if(i≠cc) est_i←the est with the highest ts;
(24)     if(ECHOG with (ts= r=r_i) represent
             at least (f+1) hosts){
(25)       ∀j ≠ i: send DECISION(est_i) to m_j;
(26)       fl_i←true;}
         }
       }
   }
----------Task 2: Reliable broadcast ----------
  upon reception of DECISION(est) from host m_k:
(27) if(fl_i=false){
(28)   ∀j ≠ i, k: send DECISION(est) to m_j; fl_i←true; }
COEND
```

Fig. 1. The proposed protocol: Task 1 and Task 2.



```
----------Task 3: Handling Late ECHOL----------
// The code executed by each clusterhead;
while (fl_i≠true){
(29) upon reception of ECHOL (r,v,ts) with (r<r_i) or
          (r=r_i and an ECHOG(r_i, *, *, *,*) has been sent);
(30)   construct an new ECHOG and send it to DA;}

----------Task 4: Clusterhead switch ----------
--------Task 4.1: code executed by host m_i ----------
(31) while(fl_i≠true and (p∈ suspected_i or
                p is not the nearest one)) {
(32)   sn←sn+1; q ←the nearest unsuspected clusterhead;
(33)   send a LEAVE(r_i, sn) to p;
(34)   send a JOIN(r_i, sn) to q;
(35)   wait until PROPH(r_q, v)received or q∈ suspected_i;
(36)   if (PROPH(r_q, v) received){
(37)     if(r_i< r_q){
(38)       r_i ← r_q;
(39)       for(ts_i= rr<r_i) send ECHOL(rr, est_i, ts_i) to q;
(40)       if(v ? _ ){ est_i← v; ts_i←r_i;}
(41)       GOTO (15);
(42)   }else if (r_i = r_q){
(43)     if(ph_i =1){
(44)       for(ts_i= rr< r_i) send ECHOL(rr, est_i, ts_i) to q;
(45)       if(v ? _ ){est_i← v; ts_i←r_i;}
(46)       GOTO (15);
(47)     }else if (ph_i =2){
(48)       for(ts_i= rr= r_i) send ECHOL(rr, est_i, ts_i) to q;
(49)       GOTO (3);}
(50)   }else if(r_i > r_q){
(51)     if(ph_i =1){
(52)       for(ts_i= rr< r_i) send ECHOL(rr, est_i, ts_i) to q;
(53)       GOTO (4); }
(54)     else if (ph_i =2){
(55)       for(ts_i= rr= r_i) send ECHOL(rr, est_i, ts_i) to q;
(56)       GOTO (3);}
         }
(57)   }else GOTO (32);
     }
----------Task 4.2: code executed by clusterhead g------
(58) while(fl_i≠true){
(59)   upon reception of LEAVE(r_i, sn) from host m_i{
(60)     delete m_i from local host list;}
(61)   upon reception of JOIN(r_i, sn) from host m_i {
(62)     add m_i to local host list;
(63)     if(ph_g=2){
(64)       if(PROP(r_g, est_cc) received from m_cc)
(65)         send PROPH(r_g, est_cc) to m_i;
(66)       else send PROPH(r_g, _ ) to m_i;
         }
       }
   }//endwhile;
```

Fig. 2. The proposed protocol: Task 3 and Task 4.

is shown in Fig. 1, and the pseudocode of Tasks 3 and 4 is shown in Fig. 2.

Task 1. This task consists of two phases. At the beginning of a round r, the current coordinator m_{cc} sends $PROP(r, est_{cc})$ to the hosts in set H. Upon receiving the $PROP(r, est_{cc})$ message from m_{cc}, a clusterhead forwards the $PROP$ message to all of its local hosts. If a clusterhead suspects m_{cc} before receiving $PROP(r, est_{cc})$, then it 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 clusterhead or its local clusterhead is suspected or its local clusterhead is no longer the nearest one. The symbol "$-$" in a message means any possible value. If a $PROP(r, v)$ message with $v \neq \perp$ is received, then m_i updates its estimated value to v and timestamp to r. If its local clusterhead is suspected or it is no longer the nearest one, then m_i invokes Task 4, which is the "switch" procedure, to associate with another clusterhead, which will be presented later. Then, Phase 1 is finished.

In Phase 2, the message exchange pattern is determined by the set of *decision_makers* and *agreement_keepers*. As in HMR, *decision_makers* are the hosts that need to check the decision predication to know if they can decide during the current round and *agreement_keepers* are the hosts that should keep the updated estimate of the final decision. Differently from HMR, we use a single set DA to store both the *decision_makers* and the *agreement_keepers*. The roles of a *decision_maker* and an *agreement_keeper* are the same in terms of message exchange. Combining the sets D and A can help

increase the probability of making decisions in a round without causing any additional overhead. Therefore, in HC, each host in DA simultaneously plays two roles: *decision_maker* and *agreement_keeper*. DA is defined by the function *dec_agr*(*r*), which must satisfy three constraints:

1. *dec_agr*(*r*) is deterministic. Hence, during the same round r, all hosts have the same DA.
2. *dec_agr*(*r*) returns only clusterheads, that is, $DA \subseteq H$.
3. *dec_agr*(*r*) returns at least the coordinator of rounds r and $r+1$.

Phase 2 is started by sending $ECHOL$ messages. Each host first sends an echo message $ECHOL(r_i, est_i, ts_i)$ to its local clusterhead. If the host itself is not a clusterhead, then it enters the next round $r+1$. Each clusterhead waits for an echo message $ECHOL(r, -, -)$ from each unsuspected local host. Then, each clusterhead constructs an echo message $ECHOG(r, v, ts_v, x, y)$ by merging the $ECHOL(r, -, -)$ messages collected. v is the estimate value carried by the $ECHOL(r, -, -)$ message with the highest timestamp and ts_v is that timestamp. x is the set of the hosts that sends the $ECHOL(r, -, -)$ messages with ts_v, whereas y is the set of the hosts that sends $ECHOL(r, -, -)$ messages with other timestamps. The clusterhead then sends the $ECHOG(r, v, ts_v, x, y)$ message to the hosts in set DA. Each clusterhead in DA waits for $ECHOG$ messages until 1) the $ECHOG(r, -, -, -, -)$ messages received represent no less than $n-f$ hosts or 2) an $ECHOG(-, -, ts_v, -, -)$ with $ts_v > r$ is received. Here, "represent a host" means that the host is included in the set x or y of the $ECHOG$ message. A clusterhead in DA updates its estimate to the value carried by the $ECHOG$ message with the highest timestamp, but keeps the timestamp unchanged. Finally, each clusterhead in DA checks whether it can decide in the current round. If there are $f+1$ or more hosts in x sets of the $ECHOG(r, v, ts_v, x, y)$ messages with $ts_v = r$, then it decides upon the value v and broadcasts the final value.

Task 2. Task 2 simply broadcasts the decision value. When a host that has not decided receives a *DECISION* message, it forwards the *DECISION* message to all of the other hosts except the sender and decides upon the value carried by the *DECISION* message.

Task 3. Task 3 handles the late $ECHOL$ messages. An $ECHOL$ message is "late" if it arrives at a clusterhead that has sent out an $ECHOG$ message for the corresponding round. This happens when a clusterhead p suspects a correct local host or a host m_i joins a new cluster whose clusterhead is in a round greater than ts_i. If a late $ECHOL$ message is ignored, then a clusterhead in set H may be blocked forever. To avoid this, when a clusterhead 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 an $ECHOG$ for the round r_i, p constructs a redeeming $ECHOG$ message for m_i and sends it to all hosts in H.

Task 4. This task is for an MH to switch its clusterhead. It is invoked when a host m_i suspects its current clusterhead p or p is no longer the nearest clusterhead. m_i chooses a new clusterhead q, which is the nearest among the unsuspected clusterheads, and then sends a message $LEAVE(r_i, sn)$ to p and a message $JOIN(r_i, sn)$ to q. Upon reception of the $LEAVE$ message, p deletes m_i from its local host list. Upon reception of the $JOIN$ message, q adds m_i to its local host list. Then, if q is in Phase 2, it sends m_i the same proposal message $PROPH(r_q, est_{cc})$ or $PROPH(r_q, \perp)$, as sent in Phase 1. Upon reception of the $PROPH(r_q, w)$ message from q, the behavior of m_i can be classified into three cases.

Case 1: $(r_i < r_q)$ or $(r_i = r_q, ph_i = 1)$. m_i updates its round number to r_q and sends $ECHOL(rr, est_i, ts_i)$ messages to q, where $ts_i \leq rr < r_q$. If $w \neq \perp$, then 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.

Case 2: $(r_i > r_q, ph_i = 1)$. m_i sends $ECHOL(rr, est_i, ts_i)$ messages to q, where $ts_i \leq rr < r_i$, and then resumes the normal execution by continuing Phase 1 of round r_i.

Case 3: $(r_i = r_q, ph_i = 2)$ or $(r_i > r_q, ph_i = 2)$. m_i sends $ECHOL(rr, est_i, ts_i)$ messages to q, where $ts_i \leq rr \leq r_i$, and then resumes the normal execution by entering the next round $r_i + 1$.

4 Correctness of the Protocol

Since the validity property of the HC protocol is obvious, in this section, we only present proofs for the termination property and agreement property. The term "*indirect suspicion*" used in the proof refers to the scenario that an MH itself does not suspect the current coordinator, but it receives a $PROP(r, \perp)$ from its local clusterhead.

4.1 Termination

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

Proof. If some correct host blocks forever before round $r+1$, then there must be a smallest round, say $rs (rs < r+1)$, during which some correct host is blocked forever. Therefore, we only need to prove that "no correct host can be blocked forever in round rs." The proof is by contradiction.

Assume that some correct host m_i is blocked forever in round rs. Then, m_i must be blocked in a wait statement at line 7, 10, 35, 18, or 22 of round rs. Let us analyze these cases one by one.

Case 1: m_i is blocked at line 7. Obviously, m_i is a clusterhead. If the coordinator m_{cc} is a correct host, then m_i eventually receives the proposal message from m_{cc}. If m_{cc} is a faulty host, then m_i eventually suspects m_{cc} after m_{cc} crashes. Therefore, m_i cannot be blocked forever at line 7.

Case 2: m_i is blocked at line 10. If m_i is a clusterhead, then it is the local clusterhead of itself. Since m_i cannot be blocked forever at line 7, it eventually receives the $PROP(r, -)$ message sent by itself at line 8 or 9. If m_i is not a clusterhead, then there are two possible situations. Let p be the local clusterhead of m_i. If p is a correct host and stays the nearest to m_i, then it eventually sends out a $PROP(r, -)$ message and m_i eventually receives it. Otherwise, m_i eventually invokes the clusterhead switch procedure. Therefore, m_i cannot be blocked forever at line 10.

Case 3: m_i is blocked at line 35. Obviously, the clusterhead switch procedure has been invoked. If the new clusterhead selected is a faulty host, then m_i

eventually suspects it after it crashes and invokes the clusterhead switch procedure again. Since at least one clusterhead is correct ($k \geq f+1$), m_i eventually finds a correct clusterhead (by the accuracy of $\Diamond P$). Then, the new clusterhead eventually sends a $PROPH(r,-)$ message to m_i (no host is blocked at line 7 forever) and m_i eventually receives the message. Therefore, m_i cannot be blocked forever at line 35. It is important to notice that, if an FD of $\Diamond S$ is used here, then some correct clusterheads in set H may always be suspected by correct MHs from time to time. Consequently, a correct MH may be entangled in endless switches between two or more correct clusterheads falsely suspected from time to time, which may prevent the protocol from termination.

Case 4: m_i is blocked at line 18. Obviously, m_i is a clusterhead waiting for $ECHOL$ messages from its local MHs. All of the hosts in the local host list of m_i can be categorized into three classes: 1) faulty hosts, 2) correct hosts that have left m_i (but m_i has not received their $LEAVE$ messages), and 3) the other hosts. For hosts in class 1, m_i eventually suspects them after they crash. For hosts in class 2, each of them must have sent a $LEAVE$ message to m_i before it leaves m_i (line 33). m_i eventually receives the $LEAVE$ messages and deletes them from the local host list. For hosts in class 3, m_i eventually receives an $ECHOL$ from each of them because they cannot be blocked at line 7, 10, or 35. Therefore, m_i cannot be blocked forever at line 18.

Case 5: m_i is blocked at line 22. Obviously, m_i is a *decision_maker/agreement_keeper*. There are two possible conditions to unblock m_i: 1) m_i receives $ECHOG$ messages that can represent no less than $n-f$ hosts and 2) m_i receives an $ECHOG$ message with timestamp $ts > rs$. We now prove that at least one of these conditions is eventually satisfied. By assumption, rs is the smallest round in which a correct host is blocked forever, so at least $n-f$ correct hosts eventually proceed to the round rs and execute line 16. Then, we categorize all of the correct hosts into two classes:

1. the hosts with correct clusterheads when they execute line 16 and
2. the hosts with faulty clusterheads when they execute line 16.

For a host m_j in class 1, the local clusterhead 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 2, after its clusterhead crashes, m_j eventually invokes the cluster switch procedure, finds a correct clusterhead host q, and, after one or more cluster switches, receives a $PROPH(r_q,-)$ message from q. Then, we consider different situations according to ts_j:

2.a. If $ts_j \leq rs$, then an $ECHOL(r, est_j, ts_j)$ is sent to q at line 39, 44, 48, 52, or 55.

2.b. If $ts_j > rs$, then an $ECHOL(-,-,>rs)$ is sent to q at line 39, 44, 48, 52, or 55.

Considering that q is a correct host, it eventually includes m_j in an $ECHOG$ to m_i. Let us examine the ECHOG messages received by m_i in round rs. If some host belongs to class 2.b, m_i eventually receives an $ECHOG(-,-,>rs,-,-)$ and, consequently, condition 2 is satisfied; otherwise, all $n-f$ correct hosts belong to class 1 or 2.a and m_i eventually receives enough $ECHOG(r,-,-,-,-)$, that is, condition 1 is satisfied. Therefore, m_i cannot be blocked forever at line 22. □

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 round r, then no $PROP(r',v)$ with $r' > r$ can be sent out before tr.*

Proof. The proof is by contradiction. Assume that at least one $PROP(r',v)$ message with $r' > r$ has been sent out by the time tr. Let rm be the greatest round number of all of the $PROP(r',v)$ messages that have been sent out by time tr. Then, $rm > r$ and $rm - 1 \geq r$. Obviously, the coordinator of round rm, the host c_{rm}, must have finished line 22 of round $rm-1$ because it has sent out $PROP(rm,v)$ before tr. Since the timestamp of the estimate at any host can only be changed at line 12, 40, or 45 and rm is the greatest round number in $PROP(r',v)$ messages by time tr, c_{rm} must not have received an $ECHOG$ with $ts > rm-1$ in round $rm-1$. Therefore, c_{rm} must have received $ECHOG$ messages representing at least $n-f$ hosts at line 22 of round $rm-1$. This means that at least $n-f$ hosts sent out $ECHOL(rm-1,-,-)$ messages in round $rm-1$ before time tr, so at least $n-f$ hosts finished *Phase* 1 of round $rm-1$ before time tr. Since $rm-1 \geq r$, at least $n-f$ hosts finished *Phase* 1 of round r before c_r sent out $PROP(r,v)$ in round r. Thus, at least $n-f$ hosts suspected c_r directly or indirectly in *Phase* 1 of round r, which contradicts the assumption in the lemma. □

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

Proof. By Lemma 2, the corollary obviously holds. □

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

Proof. If one host decides, then all correct hosts eventually decide due to the reliable broadcast mechanism (lines 25 and 28). Therefore, we only prove that at least one host decides. The proof is by contradiction.

Assume that no host decides. According to the accuracy and completeness of $\Diamond P$, there is a time t after which all correct hosts are never suspected by any correct MH and all faulty hosts are permanently suspected by every correct MH after they crash. Since there is at least one correct host m_x in H after time t, every correct host eventually associates itself with a correct clusterhead and does not suspect it any more. Let r be the first round coordinated by m_x and started after t. By the assumption ("no host decides") and Lemma 1, eventually, all correct hosts enter round r and m_x decides in round r, which contradicts the assumption. The theorem holds. □

4.2 Agreement

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

1. *No host decides before r.*
2. *If the coordinator of r' sends a PROP message, this message carries the estimate value v.*

Proof. Proof for Part 1. The proof is by contradiction. If no host decides at line 26, then no host can decide at line 28. We therefore only consider the decision at line 26. Assume that some host m_j decided at line 26 in some round s before r, that is, $s < r$, and the decision value is u. m_j must have received at least one $ECHOG$ message carrying a timestamp equivalent to s and the union set of the x sets in those $ECHOG$ messages includes at least $f+1$ hosts. Since all $ECHOG$ messages are constructed based on $ECHOL$ messages, at least $f+1$ $ECHOL(s,u,s)$ have been sent out. From the definition of r ("... first round in which ..."), we have $r \leq s$, which contradicts the assumption $s < r$. Part 1 holds.

Proof for Part 2. In any round r, the timestamp ts of the estimate at any host can only be changed to r at line 12, 40, or 45. By the assumption in the lemma, a $PROP(r,v)$ has been sent out by c_r, which is the coordinator of round r, and at least $f+1$ hosts have received the $PROP(r,v)$ in *Phase* 1 of round r. Let tp be the moment when c_r sent out the $PROP(r,v)$ message. Since $n-(f+1) < n-f$, by Lemma 2, all $PROP(r',-)$ messages with $r' > r$ must be sent out after time tp. Let R be the list of the round numbers of all $PROP(r',-)$ messages with $r' > r$. Without loss of generality, we assume that $R = (r_0 = r, r_1, r_2, r_3, \ldots r_i, \ldots)$, where the round numbers are sorted in the ascending order of the moments when the corresponding $PROP$ messages are sent out.

We now prove that, for each round r_i in R, the proposal value carried by $PROP(r_i,u)$ is equal to v, that is, $u = v$. The proof is by induction on the sequence number i in R.

Base case: $i = 0$. According to the HC protocol, a host sends an $ECHOL(r,v,r)$ only if it has received a $PROP(r,v)$ or $PROPH(r,v)$. Therefore, the local clusterhead of this host must have received a $PROP(r,v)$. The lemma holds.

Induction hypothesis: $i > 0$. 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}. We define two sets of hosts:

- The set G includes all of the hosts that have received a $PROP(r_i,w)$ or $PROPH(r_i,w)$ message with $0 \leq i \leq k$. By the induction hypothesis, $\forall m_j \in G : est_j = w = v$, and $ts_j = r_i$. Since at least $f+1$ hosts have sent $ECHOL(r,v,r)$, we have $|G| \geq f+1$.
- The set B includes the hosts that have not received a $PROP(r_i,w)$ message with $0 \leq i \leq k$. Obviously, $\forall m_j \in B : ts_j < r$. Therefore, all timestamps of the hosts in set B are less than those of the hosts in set G.

Now, let us consider the behavior of host c_{rk+1} in *Phase* 2 of the round $(r_{k+1})-1$. By the definition of DA, we have $c_{rk+1} \in DA$ during round $(r_{k+1})-1$, so c_{rk+1} waits for the $ECHOG$ messages at line 22 of round $(r_{k+1})-1$. There are two conditions to end the waiting at line 22:

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 23. In fact, the value u must come from an $ECHOL(-,u,tsm)$, so the sender of this $ECHOL$ must have received a $PROP(tsm,u)$ or $PROPH(tsm,u)$. This means that the local clusterhead of the sender of this $ECHOL$ message must have received a $PROP(r,u)$. By the definition of R and the induction hypothesis, $tsm \in \{r_0, \ldots, r_k\}$, so $u = v$.
2. c_{rk+1} receives $ECHOG((rk+1)-1,-,-,-,-)$ messages that can represent at least $n-f$ hosts, which means that at least $n-f$ message $ECHOL((r_{k+1})-1,-,-)$ are merged. Let X denote the set of the hosts that sent these $ECHOL$ messages. Obviously, $|X| \geq n-f$. At line 23, c_{rk+1} updates its estimate to the value u carried by the echo message $ECHOL((rk+1)-1,u,tsm)$, where tsm is the highest time stamp. Since $|G| \geq f+1$, we have $G \cap X \neq \emptyset$. Therefore, the message $ECHOL((r_{k+1})-1,u,tsm)$ must be sent by a host in G. By the definition of G, $u = v$.

Then, for both cases 1 and 2, the estimate value of c_{rk+1} is updated to v in round $(r_{k+1})-1$ and, consequently, c_{rk+1} sends out a $PROP(r_{k+1},v)$ in round r_{k+1}. The lemma holds. □

Theorem 2. *No two hosts decide upon different values.*

Proof. If a host decides upon a value at line 28, then this value must have been decided upon by another host at line 26. Therefore, we only consider values decided upon at line 26.

Let m_i be a host that decides upon a value v_i in round r_i. Since m_i decides in round r_i, it has received at least one $ECHOG(r_i,v_i,r_i,-,-)$ message. Therefore, the coordinator of round r_i had sent out a $PROP(r_i,v_i)$. Similarly, if another host m_j decided upon another value v_j in round r_j, then the coordinator of round r_j must have sent out $PROP(r_j,v_j)$. 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, $r \leq r_i$ and $r \leq r_j$, so $v = v_i = v_j$. The theorem holds. □

5 Performance Evaluation

We have carried out extensive simulations to evaluate and compare the performance of the proposed HC protocol, the HMR protocol, and the BHM protocol in a MANET environment. In this section, we report the simulation results.

5.1 Performance Metrics

The performance of a consensus protocol involves two aspects: time cost and message cost. In the literature, the "number of communication steps/rounds" is usually used to analytically evaluate the performance of a consensus protocol [18], [27]. This metric is closely related to both time cost and message cost because, roughly speaking, more rounds or communication steps mean longer time and more

TABLE 1
Settings of the Simulations

No. of the Hosts	10 to 100
Territory (m)	200 to 630
f/*n*	10% to 50%
Mean life of crashed hosts	30 ms
Stabilization interval	600 ms
Transmission radius	100 m
Mean link delay	5 ms
Max link delay (after stabilization interval)	100 ms
fderr, error rate of FD	10% to 50%
Interval of Heartbeat messages	10 ms
Routing Protocol/Policy	Least hops
Threshold of clusterhead switch	2 hops
Moving Speed	10 ~30 m/s
Mobility model	Random Waypoint
Mobility Level	50%

messages. However, this metric cannot precisely/directly reflect either of the costs. Different rounds of an execution may overlap and the duration of one round in different protocols or scenarios may be different. Similarly, the number of messages exchanged per round also significantly affects the message cost.

Furthermore, in a MANET, the concepts of "message" and "hop" must be distinguished. In traditional distributed systems, the message cost is computed in terms of the number of "end-to-end" messages. One message may take one or more hops to reach the destination. One "hop" means one network layer message, that is, a point-to-point message. In traditional systems, the costs of messages transmitted in different numbers of hops are regarded as the same. In a MANET, however, the resource is seriously constrained, so the cost must be measured more precisely. In this paper, we use four metrics:

1. *Number of rounds* (*NR*): The average number of rounds executed by the hosts to achieve the global decision.
2. *Execution time* (*ET*): The actual time required to achieve the global decision.
3. *Number of messages* (*NM*): The total number of messages exchanged to achieve the global decision. For the HC protocol, the additional messages for cluster switch, including $LEAVE$, $JOIN$, late $ECHOL$ (in Task 4.1), and $PROPH$ messages, are also included.
4. *Number of hops* (*NH*): The total number of hops of the messages exchanged to achieve the global decision.

5.2 Simulation Setup

The simulation system consists of three modules: the network, the FD, and the consensus protocol. The main parameters of the simulations are shown in Table 1.

All of the hosts are randomly scattered in a rectangular territory. To evaluate the scalability of the protocols, we varied the number of hosts, that is, the system scale, and, accordingly, the territory scale in proportion so that the performances under different numbers of hosts are comparable. We also varied f by changing the value of f/n from 10 percent to 50 percent. The lifetime of a faulty host satisfies the exponential distribution. The MHs move according to the well-known random waypoint mobility model [3]. The mobility level, defined as the percentage of the time during which a host moves over the total lifetime of the host, is fixed to 50 percent.

For message routing, we implemented a simple protocol based on the "least hops" policy, which is adopted in many classical routing protocols in MANETs such as AODV [25], DSDV [24], and DSR [19]. A routing table is maintained at each host proactively. The message delay is also assumed to satisfy the exponential distribution.

The FD is simulated using a heartbeat mechanism as in nearly all implementations of unreliable FDs [20]. The FD module randomly makes mistakes, with an average error rate *fderr*. To guarantee the properties of $\Diamond P$, the network is set to be partially synchronous [5]: the bounds on the message delay and processing speed are unknown and hold only after an unknown stabilization interval. After the stabilization interval, the FD makes no mistake.

The consensus protocols are implemented as separate modules that are attached to MHs. For the HMR protocol, a single set DA of *decision_makers* and *agreement_keepers* is used, as in our HC protocol. Due to the dependence on MSSs, BHM cannot be directly implemented in a MANET. We simulated a variant of BHM by selecting $2f + 1$ MHs as privileged hosts which execute the HMR protocol. The rest of the hosts only passively wait for the decision value (because each privileged host has its own initial value, it does not need to collect initial values from others). To get stable results, each execution was repeated 100 times and the average values are reported.

5.3 Simulations Results

Since HMR is the basis of the other two protocols, we first study the HMR protocol. The performance of our HC protocol and the comparison of the three protocols are then presented. If there is no explicit indication, then *fderr* is set to 10 percent in the simulations.

5.3.1 Performance of HMR

The performance of HMR against the system scale is shown in Figs. 3, 4, 5, and 6. Since the size of DA has a significant effect on HMR's performance, we use three representative values: 2 (the current and next coordinator), n, and $n/2$. The curves with different sizes of DA are labeled "SmallSetDA," "FullSetDA," and "MiddleSetDA," respectively.

The results show that *NR*, *ET*, *NM*, and *NH* all increase with the increase of n. This is because a larger system scale results in more messages and, consequently, more time in a round. Then, more failures may occur in a round and more time and messages are needed to achieve consensus.

Now, let us see the effect of the size of DA. When $|DA| = n$ ($|DA| = 2$), HMR needs the smallest (largest) NR and, when $|DA| = n/2$, *NR* is in the middle. A smaller DA

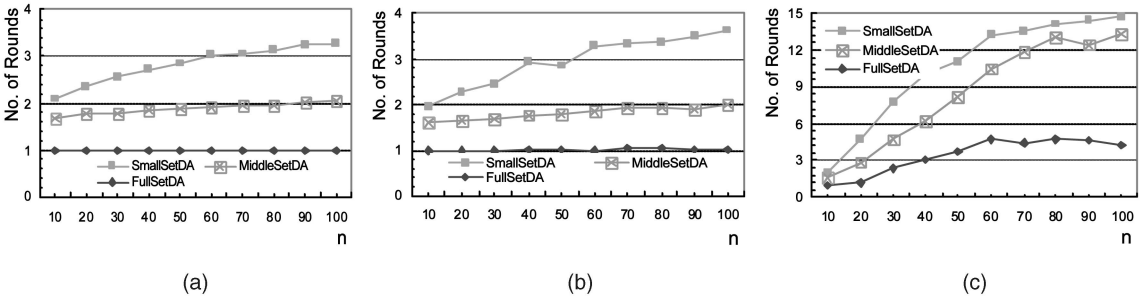


Fig. 3. The *NR* of HMR.

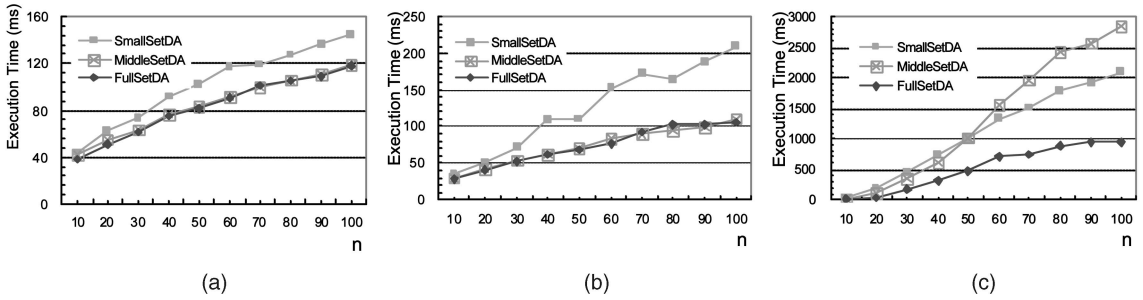


Fig. 4. The *ET* of HMR.

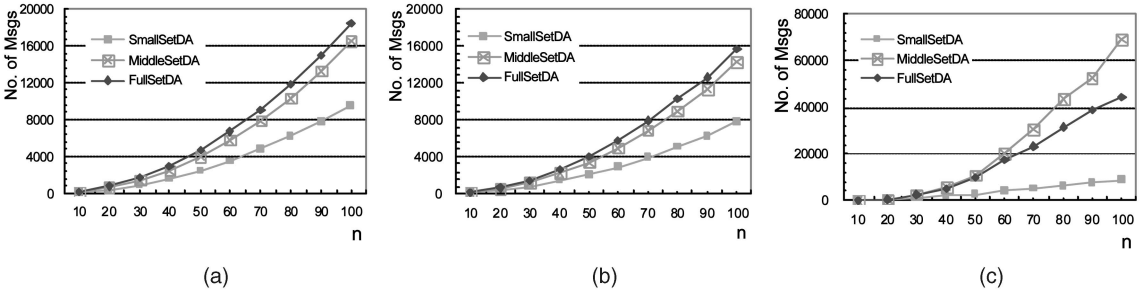


Fig. 5. The *NM* of HMR.

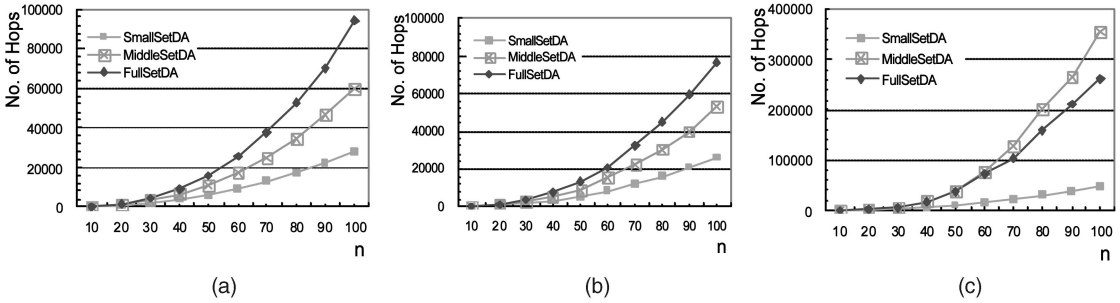


Fig. 6. The *NH* of HMR.

results in a smaller probability of making a decision in a round and, consequently, more rounds are needed to achieve consensus. The effect of $|DA|$ on *ET* and *NM/NH* is more complex. When $|DA| = n$ ($|DA| = 2$), HMR needs the shortest (longest) time but the largest (smallest) NM/NH. As discussed in Section 5.1, *ET* or *NM/NH* is the accumulation of two values: the NR and the time/message cost per round. A smaller DA results in fewer messages and less time per round but more rounds. Since the value of *NR* is much smaller than the NM per round ($O(10)$ versus $O(n^2)$), the NM per round dominates the change of *NM/NH* when $|DA|$ changes, as shown in Figs. 5 and 6. However, Fig. 4 shows that the change of *ET* with different sizes of DA is dominated by *NR*, which indicates that the change of the time cost per round with different sizes of DA is very small.

In general, with the change of $|DA|$, there is a trade-off between the message cost and the time cost. A smaller DA results in smaller message cost but larger time cost. However, the effect on *ET* is not so significant as that on *NM/NH*, so we fix $|DA| = 2$ in the rest of the simulations.

Fig. 7 shows the effect of f/n clearly. When f/n increases, more rounds are needed. This is because a large f/n means a large probability that the round is coordinated

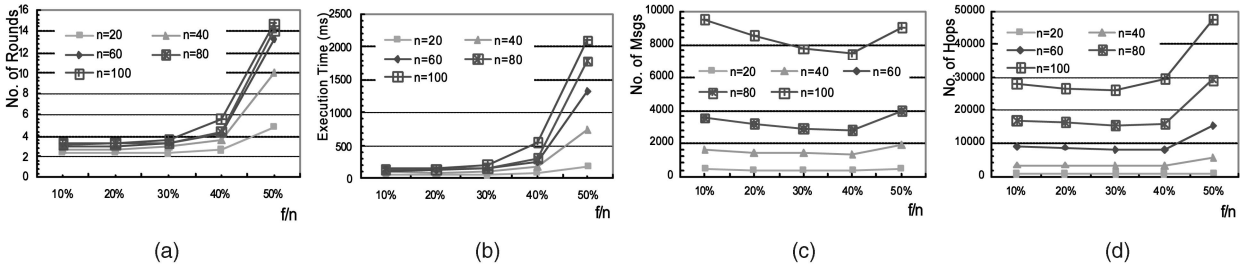


Fig. 7. Performance of HMR versus f/n, with $|DA| = 2$.

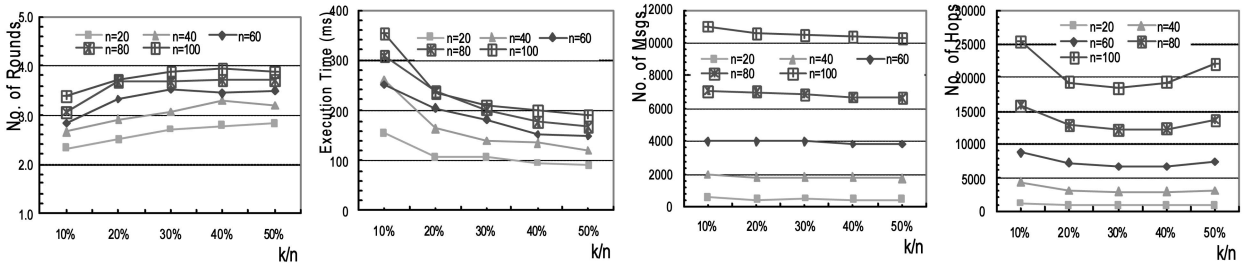


Fig. 8. Performance of HC versus k/n, $f/n = 10\%$.

by a crashed host, which is prone to fail to make a decision. Consequently, more rounds are executed to achieve consensus. The sharp increase when f/n increases from 40 percent to 50 percent may be caused by the increase in false suspicions of the coordinator. When f/n changes, *ET* changes similarly to *NR*, but *NM/NH* is affected differently. With the f/n increasing from 10 percent to 50 percent, *NM/NH* first decreases and then increases again. On one hand, more faulty hosts cause more rounds. On the other hand, more faulty hosts result in fewer hosts actually participating in the execution. As a joint result, the fewest messages/hops are needed when f/n reaches about 40 percent.

5.3.2 Performance of HC

Besides the system scale, which affects the performance of HC similarly as it does in HMR, the size of DA and the size of H (that is, the parameter k) are other parameters that significantly affect the performance of the HC protocol. Based on the simulation results of HMR, we fixed $|DA|$ to 2 in the simulation of HC.

The performance of HC against k/n is plotted in Fig. 8. Due to the constraint of $f < k$, k/n cannot be varied to a large extent under a large f/n, so, in Fig. 8, f/n is fixed to 10 percent.[4] Fig. 8 shows that, with the increase of k/n, the *NR* increases slowly, whereas the *ET* decreases. This can be explained by the operation at line 22 in the HC protocol. Upon the reception of an $ECHOG$ message with a higher timestamp, waiting at line 22 may be ended earlier, which will shorten the average waiting time of a host at line 22, but may miss a potential decision. A larger k/n means more $ECHOG$ messages exchanged in a round and, due to the asynchrony of the network, more hosts end the wait at line 22 earlier. Consequently, more rounds but shorter time are needed to achieve consensus.

NM decreases very slowly when k/n increases. The effect of k/n on *NM* is twofold. The increase of k/n causes the increase of global messages (that is, messages between the hosts in DA and the hosts in H), but reduces the number of additional messages for cluster switch, as shown in Fig. 10a (see later discussion). As a cumulative result, the *NM* changes very little when k/n increases.

As with *NM*, *NH* also decreases very slowly with the increase of k/n, but there is thorough in the middle (especially when the system scale is large). *NH* is affected by *NM* and the number of hops per message. When k/n becomes large, the average distance between a host and its clusterhead in hops is reduced and, consequently, the average NH per message is reduced. As a cumulative result of *NM* and the number of hops per message, *NH* becomes the smallest when k/n is about 30 percent.

Fig. 9 shows the performance of HC against f/n with $k/n = 50$ percent. Comparing Figs. 7 and 9, we can find that the effect of f/n on HC is nearly the same as that on HMR, so Fig. 9 can be explained similarly as Fig. 7.

Now, let us examine the overhead of maintaining the two-layer hierarchy, that is, the message cost for the cluster switch. Fig. 10 shows the percentage of additional messages for the cluster switch, including $LEAVE$, $JOIN$, late $ECHOL$ (in Task 4.1), and *PROPH* messages, under various conditions. The overhead of the cluster switch increases as f/n (Fig. 10b) or *fderr* (Figs. 10c and 10d) increase, but decreases if k/n (Fig. 10a) increases.

When f/n increases, more MHs need to switch their clusters due to crashes of faulty clusterheads and, therefore, higher overhead of clustering is caused. The effect of *fderr* on the overhead of the cluster switch comes from the effect of false suspicions. The more the mistakes made by the FD,

4. In fact, due to the constraint of $f < k$, f is set to $(n^*10\%) - 1$, but, for convenience, we still use "10 percent" to refer to the value of f/n. To guarantee fairness of comparisons, the f of HMR and BHM is set in the same way.

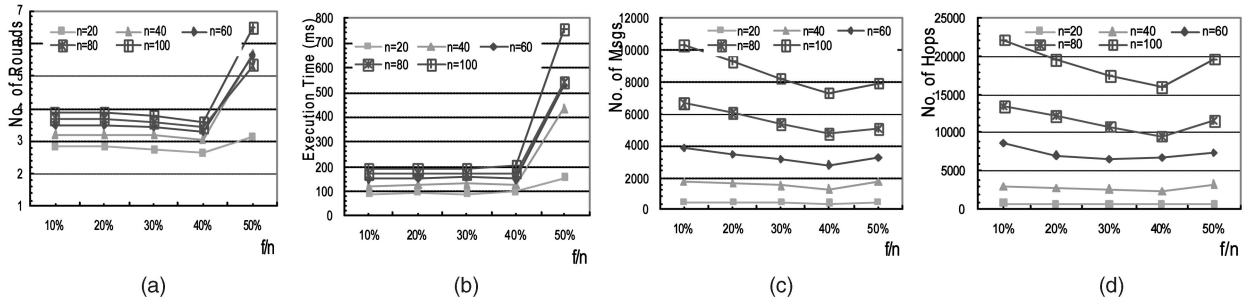


Fig. 9. Performance of HC versus f/n, $k/n = 50\%$.

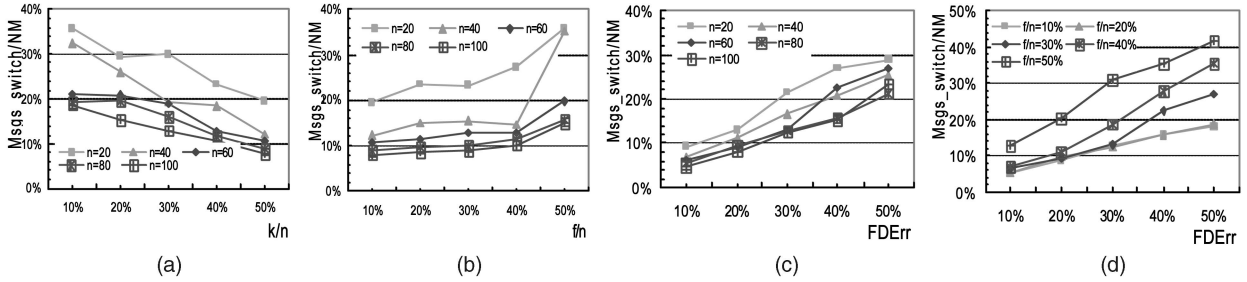


Fig. 10. Overhead of clustering.

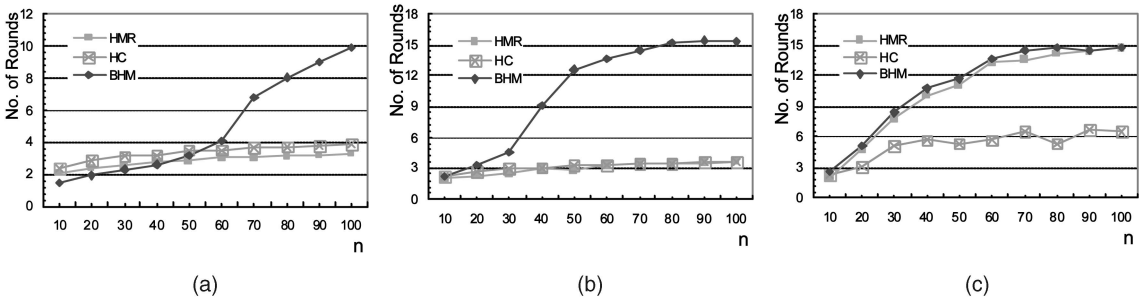


Fig. 11. Performance comparison: *NR*.

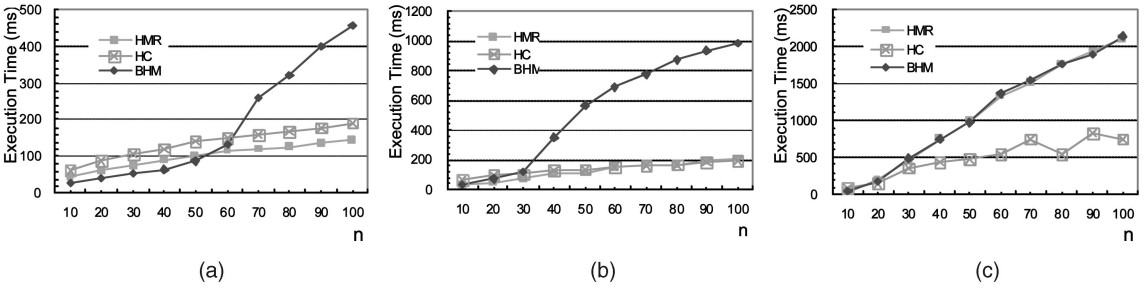


Fig. 12. Performance comparison: *ET*.

the higher the probability that an MH falsely suspects its current clusterhead and switches to another one. Consequently, the percentage of messages for the cluster switch increases. The effect of k/n can be explained as follows: A clusterhead host always selects itself as its local clusterhead. The more the hosts act as clusterheads, the fewer the hosts need to switch their clusters and, therefore, the fewer the messages caused by the cluster switch.

In general, the overhead caused by the cluster switch is not high. In most cases, it is less than 20 percent of the total NM, especially when the system scale is large and the FD performs well. More importantly, even with the overhead of the cluster switch, our proposed protocol can still achieve improvement in performance due to the two-layer hierarchy, as seen from the discussion in the following section.

5.3.3 Performance of BHM and Comparisons

Figs. 11, 12, 13, 14, 15, 16, 17, and 18 show the performance of BHM and the comparisons of all three protocols. Similarly to the other two protocols, BHM costs more messages and longer time with the increase of n and f/n. Now, let us compare the three protocols. Without loss of generality, the k/n of HC is fixed to 50 percent in the comparisons.

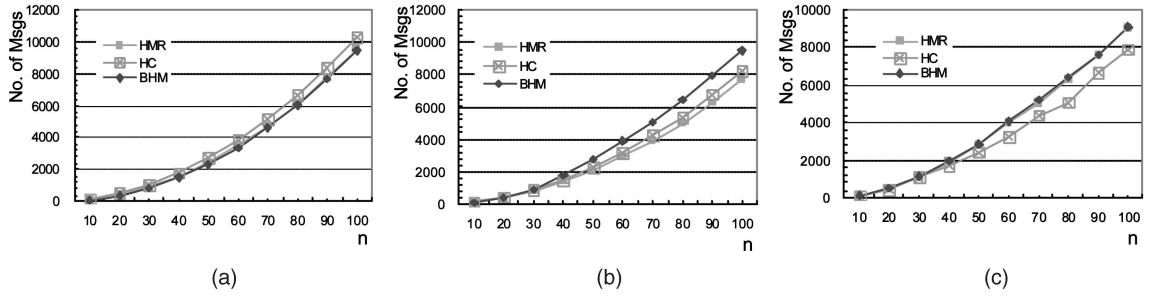


Fig. 13. Performance comparison: *NM* versus n, $fderr = 10\%$.

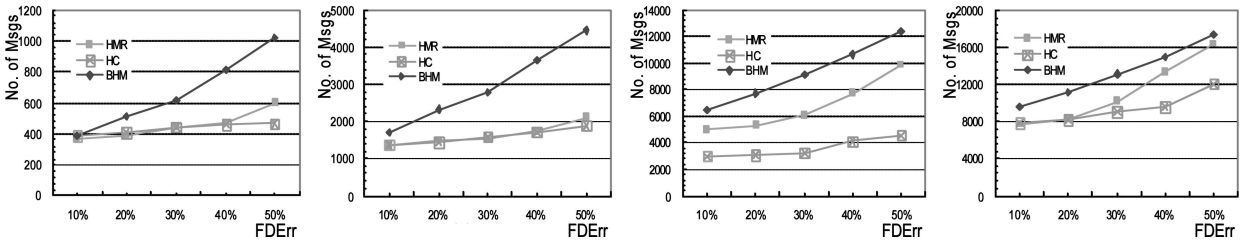


Fig. 14. Performance comparison: *NM* versus *fderr*, $f/n = 30\%$.

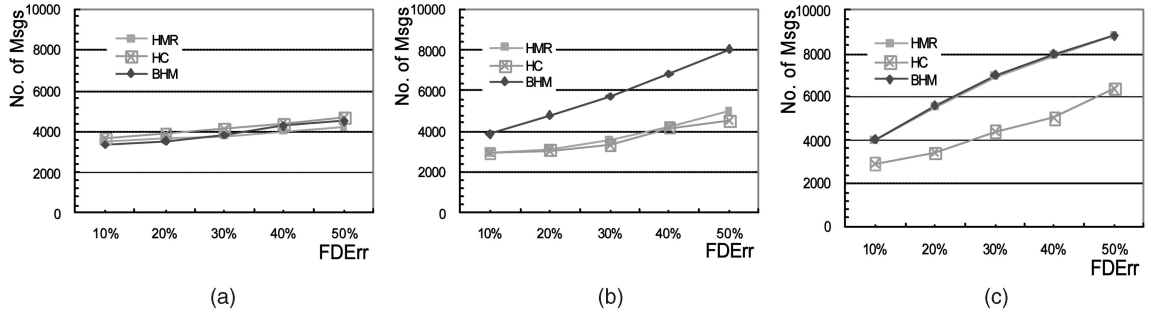


Fig. 15. Performance comparison: *NM* versus *fderr*, $n = 60$.

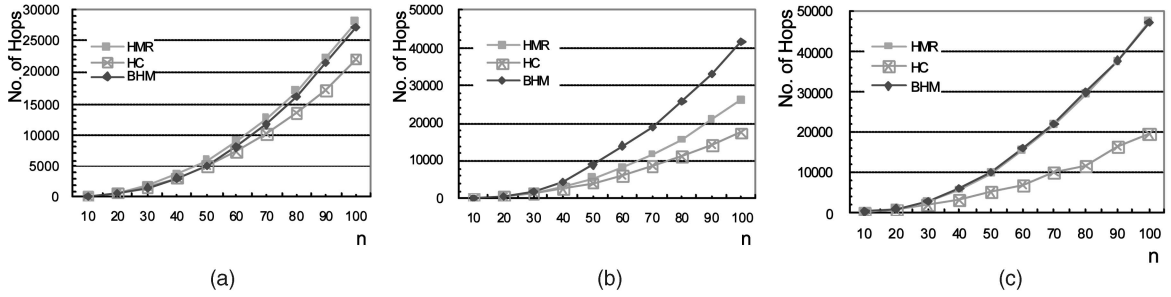


Fig. 16. Performance comparison: *NH* versus n, $fderr = 10\%$.

Comparisons in NR and ET. Figs. 11 and 12 show the *NR* and *ET* of all three protocols under different values of n and f/n. When n and f/n are small, BHM can achieve consensus with the fewest rounds and shortest time. Under a small f/n, the BHM protocol involves many fewer hosts in the procedure of achieving consensus, that is, it can be viewed as an HMR protocol running in a much smaller system. Therefore, BHM can achieve consensus with fewer rounds and shorter time than HC and HMR do. However, with the f/n increasing from 10 percent to 50 percent, the number of hosts actually participating in the rounds of message exchange in BHM gradually turns out to be the same as in HMR and, consequently, the performance of BHM becomes the same as that of HMR.

When the system scale becomes large, the *NR* and *ET* of BHM increase sharply and become the largest among the three protocols. As discussed above, BHM can be viewed as an HMR protocol running in a system of $2f + 1$ hosts. The actual percentage of faulty hosts in such a "smaller" system may vary between 0 and $f/(2f + 1)$, though the average percentage equals f/n. The larger the system scale is, the higher the actual percentage of faulty hosts can vary. Since a large percentage of faulty hosts results in the sharp increase of *NR* in HMR (as shown in Fig. 7), the *NR* and *ET* of BHM

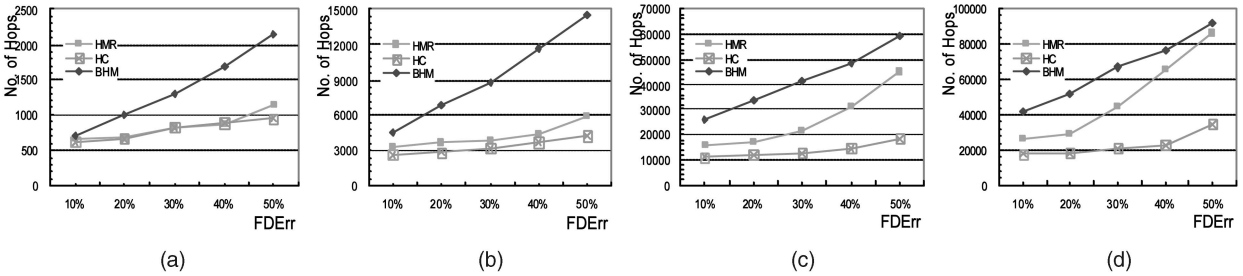


Fig. 17. Performance comparison: *NH* versus *fderr*, $f/n = 30\%$.

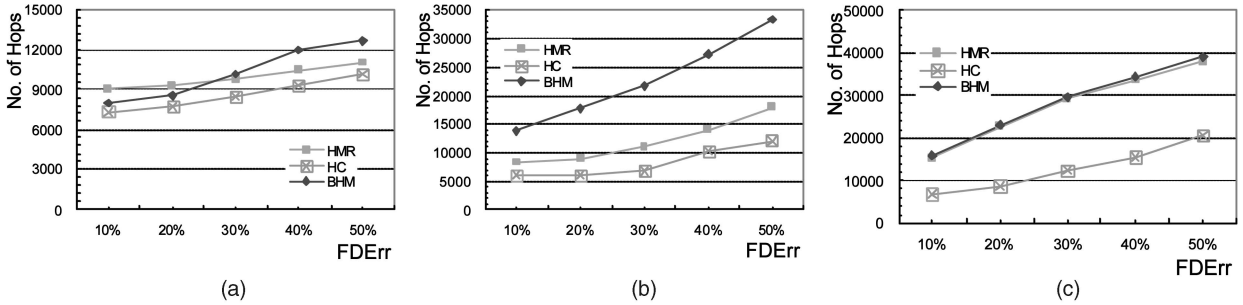


Fig. 18. Performance comparison: *NH* versus *fderr*, $n = 60$.

under a large system scale become the largest among the three protocols.

The difference in *NR* and *ET* between HMR and HC is also affected by the value of f/n and n. Due to the message forwarding at clusterheads, HC needs two more communication steps than HMR. Therefore, each round of HC lasts longer than that of HMR and more failures may occur during one round. Consequently, HMR achieves consensus earlier and faster. However, when f/n is large, HC performs better in terms of *NR* and *ET*. This is also caused by the message forwarding mechanism of the two-layer hierarchy. In the HMR protocol, when an ordinary host (that is, a host that does not belong to DA) suspects the coordinator, it proceeds to the next round after sending echo messages. In the HC protocol, however, a host outside H has to wait for its clusterhead to forward the proposal message, so the proceeding of HC in the first phase is determined by only clusterheads. Therefore, the postponement of making decisions due to false suspicions made by ordinary hosts is avoided and, consequently, fewer rounds are needed to make decisions. With the increase of f/n, the probability of such a false suspicion increases. Therefore, the difference between HC and HMR is reversed under a large f/n.

With the increase of n, the advantage of HC in *NR* and *ET* also increases. This can also be explained based on the discussion above. The proceeding of the first phase is determined only by clusterheads, so the effect of the system scale on *NR* in the HC protocol stems from the change of the number of clusterheads, that is, $|H|$. Since $|H| = n/2$, $|H|$ changes more slowly than n changes and, consequently, the *NR* and *ET* of HC increase more slowly than those of HMR.

Comparisons in NM and NH. Figs. 13, 14, and 15 show the results in *NM* under various conditions. Figs. 16, 17, and 18 show the results in *NH* corresponding to Figs. 13, 14, and 15.

The HC protocol performs badly only when very few hosts can crash and the system scale is very small. With the increase of n and f/n, HC performs much better. When $f/n = 50\%$ and $n = 100$, HC achieves consensus with only less than half of the hops cost by BHM or HMR.

As discussed before, both *NM* and *NH* are determined by *NR* and the message cost per round. Comparing Fig. 13 with Fig. 11, we can see that the relationship among the three protocols in *NM* is nearly the same as in *NR*. Therefore, *NR* dominates the difference in *NM*.

The difference in *NH* between BHM and HMR is also determined by *NR*. However, the difference between HC and the other two is not dominated by only *NR*. When the system scale is not very small, HC can achieve consensus with the fewest hops, even if its *NR* is not the smallest. Such an advantage comes from the two-layer hierarchy, which reduces the message cost per round in hops by merging messages. The larger the system is, the more messages are merged and, consequently, the greater the cost savings. This indicates that our objective of reducing the message cost by using the two-layer hierarchy is fulfilled.

Besides n and f/n, we also varied the error rate of FD *fderr*. The results show that all three protocols cost longer time and more messages when more mistakes are made by the FD. This, in fact, comes from the effect of false suspicions. The more mistakes are made by the FD, the more false suspicions may occur. As discussed earlier, more suspicions may cause chances for postponing the decision making in HMR and BHM and, consequently, longer time and more messages are needed. Although HC can alleviate such postponement with the help of the two-layer hierarchy, the overhead of the cluster switch is affected by false suspicions, as discussed in Section 5.3.2. Therefore, the cost of all three protocols increases with the increase of *fderr*. However, it is important to notice that the change of *fderr* almost does not affect the advantage of our HC protocol compared with the other two protocols.

6 Tolerance of Transmitting Message Loss

As in most consensus protocols, our HC protocol assumes reliable communication channels among hosts. However, due to the characteristics of wireless communications, transmitting message losses occur more frequently in MANETs than in wired networks. To cope with transmitting message losses, one direction is to design reliable communication protocols [9], [29], [31]. However, how reliable end-to-end channels can be provided is still a challenging topic in MANETs. Here, we take an alternative approach by enhancing the HC protocol to handle transmitting message losses in the protocol operation. We divide the channel failures into two types—*permanent* failures and *transient* failures—and design solutions for handling the two types of channel failures, respectively.

6.1 Handling Permanent Channel Failures as Host Failures

If a channel fails by crashing, that is, permanently losing all of the messages transmitted through it, then a permanent failure occurs. To circumvent permanent channel failures, a possible solution is to treat the transmitting message losses as host crashes. If a channel between a pair of hosts loses messages, then the sender, instead of the channel, is said to be faulty. This way, the system only has host failures and all channels can be thought of as reliable. Although the correctness of the HC protocol will not be affected, the resilience, that is, the capability of tolerating faults, of the protocol is degraded. There can be at most t $(t < minimum(k, n/2))$ hosts that can crash or be viewed as crashed due to channel failures.

6.2 Reducing Reliable Channels to Fair-Lossy Channels

A channel with a transient failure only loses messages for some finite time and then recovers to be a correct channel. Such a failure may recur for the same channel. More precisely, such a channel is defined as a fair-lossy channel [22], [26].

If a host m_i sends an infinite NM to host m_j, then the channel attempts to deliver an infinite NM to m_j.

Following the approach in [8], [22], our HC protocol can be extended for use in a system with fair-lossy channels. To tolerate transmitting message losses caused by channels with transient failures, three rules are added:

1. When a clusterhead p enters a new round r that is not coordinated by p, it sends a NEW(r) message to all other clusterheads.
2. Each host periodically resends the latest message that it has sent. For example, during the wait at line 10 of round r_i, a clusterhead periodically resends the $PROP(r_i, ^*)$ message to its local hosts, that is, it periodically repeats the execution of lines 8 and 9.
3. When a host m_i, either an ordinary host or a clusterhead, receives a message with a higher round number r with $r > r_i$, m_i aborts its current round and enters the round r.

With these rules, a correct host blocked due to a transmitting message loss can eventually be unblocked and the termination property can be guaranteed. Since the validity and agreement are not affected by any message loss, the enhanced HC protocol still satisfies the correctness requirements of a consensus protocol. Due to space limitation, we do not present the proof here.

7 Conclusion

In this paper, we proposed a message-efficient and scalable consensus protocol for achieving consensus in MANETs, based on Chandra-Toueg's unreliable FDs of class $\Diamond P$. A cluster-based two-layer hierarchy is imposed on the system to reduce the message cost. A number k of all of the n MHs are selected to act as clusterheads. Each MH is associated with a clusterhead and all of the hosts associated with the same clusterhead constitute a cluster. The MHs can fail by crashing and the number of faulty hosts is bounded by f, where $f < minimum(k, n/2)$. With the hierarchy, a coordinator sends proposal messages only to clusterheads and a clusterhead unmerges and forwards the proposal to its local hosts. Similarly, echo messages from hosts in the same cluster are merged into one message before they are sent to *decision_makers* and *agreement_keepers*. Therefore, the message cost is significantly reduced.

To evaluate the performance of the proposed protocol and similar ones, extensive simulations have been conducted under various conditions. Through simulations, we obtained insights into the effects of the main parameters on the performance and the features of different protocols in a MANET environment. The results show that the performance of all of the consensus protocols is significantly affected by the system scale and the percentage of faulty hosts. Compared to existing protocols, the proposed protocol can significantly reduce both message cost and time cost, especially when the system scale or the percentage of faulty hosts is large.

Acknowledgments

The authors would like to thank the anonymous referees for valuable insights that helped to improve this paper. This research is partially supported by the Hong Kong University Research Grant Council under the CERG grant PolyU 5183/04E, the France/Hong Kong Joint Research Scheme F-HK16/05T, and the China National "973" Program Grant 2002CB312002.

References

[1] N. Badache, M. Hurfin, and R. Macedo, "Solving the Consensus Problem in a Mobile Environment," *Proc. IEEE Int'l Performance, Computing, and Comm. Conf. (IPCCC '99)*, 1999.
[2] B. Badrinath, A. Acharya, and T. Imielinski, "Designing Distributed Algorithms for Mobile Computing Networks," *Computer Comm.*, vol. 19, no. 4, Apr. 1996.
[3] T. Camp, J. Boleng, and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research," *Wireless Comm. and Mobile Computing*, vol. 2, no. 5, 2002.
[4] T. Chandra, V. Hadzilacos, and S. Toueg, "The Weakest Failure Detector for Solving Consensus," *J. ACM*, vol. 43, no. 4, July 1996.
[5] T. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *J. ACM*, vol. 43, no. 2, Mar. 1996.
[6] G. Chockler, M. Demirbas, S. Gilbert, C.C. Newport, and T. Nolte, "Consensus and Collision Detectors in Wireless Ad Hoc Networks," *Proc. ACM Symp. Principles of Distributed Computing (PODC '05)*, July 2005.

[7] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design,* third ed. Addison-Wesley, 2001.

[8] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi, "Failure Detectors in Omission Failure Environments," Technical Report TR96-1608, Dept. of Computer Science, Cornell Univ., Sept. 1996.

[9] H. Elaarag, "Improving TCP Performance over Mobile Networks," *ACM Computing Surveys,* vol. 34, no. 3, Sept. 2002.

[10] P. Ezhilchelvan, A. Mostefaoui, and M. Raynal, "Randomized Multivalued Consensus," *Proc. Fourth IEEE Int'l Symp. Object-Oriented Real-Time Computing (ISORC '01),* May 2001.

[11] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM,* vol. 32, no. 2, Apr. 1985.

[12] G. Forman and J. Zahorjan, "The Challenges of Mobile Computing," *Computer,* vol. 27, no. 4, Apr. 1994.

[13] R. Friedman, A. Mostefaoui, and M. Raynal, "On the Respective Power of $\Diamond P$ and $\Diamond S$ to Solve One-Shot Agreement Problems," Technical Report 1547, IRISA, July 2003.

[14] R. Guerraoui, M. Hurfin, A. Mostefaoui, R. Oliveira, M. Raynal, and A. Schiper, "Consensus in Asynchronous Distributed Systems: A Concise Guided Tour," *Lecture Notes in Computer Science,* vol. 1752, 2000.

[15] R. Guerraoui and A. Schiper, "Consensus: The Big Misunderstanding," *Proc. Sixth IEEE Workshop Future Trends of Distributed Computing Systems (FTDCS '97),* 1997.

[16] R. Guerraoui and A. Schiper, "The Generic Consensus Service," *IEEE Trans. Software Eng.,* vol. 27, no. 1, Jan. 2001.

[17] M. Hurfin, A. Mostefaoui, and M. Raynal, "A Versatile Family of Consensus Protocols Based on Chandra-Toueg's Unreliable Failure Detectors," *IEEE Trans. Computers,* vol. 51, no. 4, Apr. 2002.

[18] M. Hurfin and M. Raynal, "A Simple and Fast Asynchronous Consensus Protocol Based on a Weak Failure Detector," *Distributed Computing,* vol. 12, no. 4, Sept. 1999.

[19] D. Johnson and D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Computing,* chapter 5, Kluwer Academic, 1996.

[20] M. Larrea, A. Fernandez, and S. Arevalo, "On the Implementation of Unreliable Failure Detectors in Partially Synchronous Systems," *IEEE Trans. Computers,* vol. 53, no. 7, July 2004.

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

[22] R. Oliveira, "Solving Consensus: from Fair-Lossy Channels to Crash-Recovery of Processes," PhD dissertation 2139, Swiss Federal Inst. of Technology (EPFL), 2000.

[23] T. Park and K.G. Shin, "Optimal Tradeoffs for Location-Based Routing in Large-Scale Ad Hoc Networks," *IEEE/ACM Trans. Networking,* vol. 13, no. 2, Apr. 2005.

[24] C. Perkins and P. Bhangwat, "Highly Dynamic Destination-Sequenced Distance-Vector (DSDV) Routing for Mobile Computers," *Proc. ACM SIGCOMM,* Aug. 1994.

[25] C. Perkins and E. Royer, "Ad-Hoc On-Demand Distance Vector Routing," *Proc. Second IEEE Workshop Mobile Computing Systems and Applications (WMCSA '99),* Feb. 1999.

[26] M. Raynal, "A Short Introduction to Failure Detectors for Asynchronous Distributed Systems," *ACM SIGACT News, Distributed Computing Column,* vol. 36, pp. 53-70, Mar. 2005.

[27] A. Schiper, "Early Consensus in an Asynchronous System with a Weak Failure Detector," *Distributed Computing,* vol. 10, no. 3, Oct. 1997.

[28] H. Seba, N. Badache, and A. Bouabdallah, "Solving the Consensus Problem in a Dynamic Group: An Approach Suitable for a Mobile Environment," *Proc. Seventh Int'l Symp. Computers and Comm. (ISCC '02),* 2002.

[29] K. Sundaresan, V. Anantharaman, H. Hsieh, and R. Sivakumar, "ATP: A Reliable Transport Protocol for Ad Hoc Networks," *Proc. ACM MobiHoc '03,* June 2003.

[30] E. Vollset and P.D. Ezhilchelvan, "Design and Performance-Study of Crash-Tolerant Protocols for Broadcasting and Reaching Consensus in MANETs," *Proc. 24th IEEE Symp. Reliable Distributed Systems (SRDS '05),* Oct. 2005.

[31] X. Yu, "Improving TCP Performance over Mobile Ad Hoc Networks by Exploiting Cross-Layer Information Awareness," *Proc. MobiCom 2004,* Sept. 2004.

[32] Q. Zhao and L. Tong, "Energy Efficiency of Large-Scale Wireless Networks: Proactive versus Reactive Networking," *IEEE J. Selected Areas in Comm.,* vol. 23, no. 5, May 2005.

Weigang Wu received the BSc degree in materials science in 1998 and the MSc degree in computer science in 2003, both from Xi'an Jiaotong University, China, and the PhD degree in computer science from Hong Kong Polytechnic University, Hong Kong, in 2007. He is currently a postdoctoral fellow in the Department of Computing at Hong Kong Polytechnic University. His research interests include parallel and distributed computing, networking, mobile and wireless computing, and fault tolerance. His recent research has focused on distributed coordination algorithms in mobile environments and mobility management in wireless mesh networks. He has published a number of papers in conferences and journals and served as a reviewer for many international journals and conferences.

Jiannong Cao received the BSc degree in computer science from Nanjing University, Nanjing, China, in 1982 and the MSc and PhD degrees in computer science from Washington State University, Pullman, in 1986 and 1990, respectively. He is currently a professor in the Department of Computing at Hong Kong Polytechnic University, Hung Hom, Hong Kong. He is also the director of the Internet and Mobile Computing Laboratory of this department. He is also a member of the IEEE Technical Committee on Distributed Processing, IEEE Technical Committee on Parallel Processing, and IEEE Technical Committee on Fault-Tolerant Computing. He has served as a member of the editorial boards of several international journals, as a reviewer of international journals/conference proceedings, and also as an organizing/program committee member of many international conferences. His research interests include parallel and distributed computing, networking, mobile and wireless computing, fault tolerance, and distributed software architecture. His recent research has focused on mobile and pervasive computing systems, developing testbeds, protocols, middleware, and applications. He has published more than 200 technical papers in the above areas. He is a senior member of the China Computer Federation, a senior member of the IEEE, and a member of the IEEE Computer Society, the IEEE Communication Society, and the ACM.

Jin Yang received the MSc degree from Xia'men University, China, and the PhD degree in computer science from Hong Kong Polytechnic University in 2006. He has worked at Lucent Technologies Optical Networks Research Center for three years. His current research interests include fault-tolerant distributed computing, mobile computing, failure detectors, and agreement protocols. Besides academic research, he also has a close relation with the industrial community. Currently, he is focusing on implementing distributed/mobile computing in Linux kernel to support network elements in mesh networks.

Michel Raynal received the "doctorat d'Etat" degree in computer science from the University of Rennes, France, in 1981. He is currently a professor of computer science at IRISA (CNRS-INRIA-University joint computing research laboratory located in Rennes), where he founded a research group on distributed algorithms in 1983. His research interests include distributed algorithms, distributed computing systems, and dependability. He has published more than 99 papers in journals and more than 210 papers in conferences. He has also written seven books devoted to parallelism, distributed algorithms, and systems (MIT Press and Wiley). His main interest lies in the fundamental principles that underlie the design and the construction of distributed computing systems. He belongs to the editorial boards of several international journals. He has served on the program committees of more than 85 international conferences and chaired the program committees of more than 15 international conferences. He received the IEEE ICDCS Best Paper Award three times in a row: in 1999, 2000, and 2001.