Available online at www.sciencedirect.com
ScienceDirect
Ad Hoc Networks 5 (2007) 299–312
Ad Hoc Networks
www.elsevier.com/locate/adhoc

An efficient reliable broadcasting protocol for wireless mobile ad hoc networks

Chih-Shun Hsu [a,*], Yu-Chee Tseng [b], Jang-Ping Sheu [c]

[a] Department of Computer Science and Information Engineering, Nanya Institute of Technology, 414, Sec. 3, ChungShang East Rd., Chung-Li 320, Taiwan
[b] Department of Computer Science and Information Engineering, National Chiao Tung University, Hsin-Chu 300, Taiwan
[c] Department of Computer Science and Information Engineering, National Central University, Chung-Li 320, Taiwan

Received 3 March 2005; received in revised form 1 November 2005; accepted 28 November 2005
Available online 28 December 2005

Abstract

The *mobile ad hoc network* (*MANET*) has recently been recognized as an attractive network architecture for wireless communication. Reliable broadcast is an important operation in MANET (e.g., giving orders, searching routes, and notifying important signals). However, using a naive flooding to achieve reliable broadcasting may be very costly, causing a lot of contention, collision, and congestion, to which we refer as the *broadcast storm* problem. This paper proposes an efficient reliable broadcasting protocol by taking care of the potential broadcast storm problem that could occur in the medium-access level. Existing protocols are either unreliable, or reliable but based on a too costly approach. Our protocol differs from existing protocols by adopting a low-cost broadcast, which does not guarantee reliability, as a basic operation. The reliability is ensured by additional acknowledgement and handshaking. Simulation results do justify the efficiency of the proposed protocol.

Keywords: Broadcast storm; Mobile ad hoc network (MANET); Mobile computing; Reliable broadcast; Wireless communication

1. Introduction

One wireless network architecture that has attracted a lot of attention recently is the *mobile ad hoc network* (*MANET*). A MANET consists of mobile hosts only (without base stations). Under such architecture, mobile hosts may have to communicate with others in a multi-hop manner, and thus each mobile host has to serve as a router. The challenge is that every mobile host can roam freely at any instant. Since collecting a global network topology is prohibitive, it is difficult to optimize the communication cost. MANET can be deployed quickly, and thus has applications in such as battlefields or disaster areas.

Broadcasting is a fundamental operation in all kinds of networks. In MANET, since the network topology is so dynamic, broadcasting could be used more frequently (in events such as giving orders, searching routes, or notifying important signals). However, due to MANET's dynamic feature and

* Corresponding author. Tel.: +886 3 4361070.
E-mail address: chison.hsu@msa.hinet.net (C.-S. Hsu).

1570-8705/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2005.11.006

radio's broadcasting nature, designing a broadcast protocol for MANET should be cautious to prevent unnecessary deficiency.

In the literature, broadcasting has been addressed intensively from different aspects [2,6,9,12,15–25]. A TDMA-based broadcast protocol is proposed in [2] to assign mobile hosts to *transmission sets* according to their IDs, where transmission sets are used to indicate on which time slots to send data. The protocol is mobility-transparent and can complete in a deterministic number of time slots, but the maximum degree of the network must be known in advance. An *intra-team* broadcast protocol is proposed in [9] that uses a minimum spanning tree or a minimum node cover set as the backbone for broadcasting, so a global topology has to be collected. But unfortunately the collision problem is not considered in this protocol. It is shown in [12] that using a naive flooding to broadcast may cause a *broadcast storm* in a MANET; several schemes, namely counter-based, distance-based, and location-based protocols, are proposed to alleviate this problem. To further alleviate the broadcast storm problem, several adaptive schemes are proposed in [23] to achieve higher reliability and better efficiency. To improve the location-based protocol proposed in [12], a location-aided broadcast protocol is proposed in [18,19], which allows each host to construct an optimal local cover set to relay the broadcast packet. The concept of the connected dominating set is used in [17,25] to reduce the number of redundant rebroadcasts; however, the potential collision problem is not considered in these works. Ref. [15] also discusses how to reduce redundant rebroadcasts. To solve QoS routing, Ref. [24] discusses how to construct an energy-efficient broadcast/multicast tree in a non-mobile environment (assuming that transmission powers are adjustable).

Several proposed systems exist to increase the reliability of MAC-layer broadcast. A TDMA-based protocol is proposed in [6]. Each host has a unique time slot. To broadcast, a RTB (request to broadcast) and a CTB (clear to broadcast) packets are defined to reduce the hidden-terminal problem and thus improve broadcast reliability. It is shown in [22] that a host can elect a neighbor host as its *collision detector* to provide feedback to the sender and thus increase broadcasting reliability. A simple RTS/CTS-like protocol is proposed in [20] to increase the reliability of a MAC-layer broadcast. Highly reliable MAC-layer broadcast protocols are proposed in [16,21]. These protocols are highly reliable, but may have a lot of redundant rebroadcasts and still cannot guarantee 100% delivery.

Most of the above referenced protocols [6,9,12,15,17,16,19,20,22–25] do not guarantee a 100% delivery of the broadcast message to all hosts. The protocol proposed in [2] can achieve 100% delivery in deterministic time slots, but is TDMA-based which is difficult to implement in multi-hop ad hoc networks. In this paper, we adopt an IEEE 802.11-based medium access model [13]. Reliable broadcasting is essential when a message/signal is urgent; otherwise, unexpected disaster may occur. Two reliable broadcasting protocols known to us are [1,14]. Alagar and Venkatesan [1] proposes a protocol based on *flooding*. Accurate 1-hop neighbor information is the first guard to increase reliability, and a *handshake* procedure serves as the second guard to guarantee 100% delivery when collisions occur. The protocol is simple but suffers from serious broadcast storm effect due to many rebroadcasts and acknowledgements. Pagani and Rossi [14] proposes a protocol based on the *clustering* structure [5,8,11]. Through distributing the broadcast message, a *forwarding tree (FT)* consisting of cluster heads is constructed. Reliability is guaranteed by many unicasts among cluster heads and collection of acknowledgements from cluster members. The protocol pays cost in maintaining the cluster structure in a regular manner, even when broadcasting is not requested, which is sometimes costly. It also critically relies on the accuracy of the FT and cluster structure to ensure efficiency. This is not easy in a MANET, especially when the host mobility is high. More detailed reviews of these two protocols are in Section 2.

In this paper, we propose a new reliable broadcasting protocol, which only keeps a loose tree relation among hosts. To alleviate the problems in [1,14], a host does not need to know which hosts are its children and, instead, a child only needs to keep track of some possible parents. Our protocol is characterized by ensuring reliability by adopting a low-cost unreliable broadcasting protocol [12] as a basis. Our protocol works in three phases: scattering, gathering, and purging. Reliability is enforced only at necessary points, i.e., in the gathering phase. This is what makes our protocol more efficient than existing ones. Simulation results do justify the effectiveness of our approach.

The rest of our paper is organized as follows. Preliminaries are given in Section 2. In Section 3, we describe our reliable broadcast protocol. Simulation

results are presented in Section 4. Section 5 concludes this paper.

2. Preliminaries

2.1. Problem statement

A MANET consists of a set of mobile hosts each equipped with a wireless transceiver. No base stations are supported in such an environment. Due to the limitation of transmission distance and radio shadows, mobile hosts may not be able to communicate with each other directly. In this case, communication should go on in a *multi-hop* manner. Hence each mobile host in a MANET should serve as a router.

In this paper, we study the *reliable broadcast problem* in a multi-hop MANET, where guaranteed delivery of a message to every host is required. We assume that the MANET may become disconnected temporarily, but will resume connectivity eventually. All members of the MANET are closed and known in advance and are numbered $1, 2, \ldots, n$. Hosts in the network share a single common channel based on an IEEE 802.11-like protocol [13]. Due to the dynamic feature of MANET, collecting a global topology of the MANET is prohibitive. Without prior knowledge of the network topology, the goal is to achieve reliable broadcasting by efficiently using the scarce wireless bandwidth.

Fig. 1 shows the location of the proposed reliable broadcast in the protocol stack. We assume that the underlying MAC protocol follows the IEEE 802.11 standard [13]. Our protocol will be developed on top of the unicast and 1-hop broadcast mechanisms defined in IEEE 802.11. The details of the MAC protocol are described in the following section.

2.2. IEEE 802.11's MAC protocol

The IEEE 802.11's medium access (*MAC*) protocol used in *MANET*s is the distributed coordination function (*DCF*) which is based on the *Carrier Sense Multiple Access with Collision Avoidance* (*CSMA/CA*) mechanism. When a mobile host wants to transmit frames, it first detects the status of the medium. If the medium is busy, the host will defer until the medium is idle for a period of time equal to *DIFS* (*DCF* interframe space). After this *DIFS* idle time, the host will generate a random backoff period, where $backoff\ time = Random() \times ST$. *Random*( ) is a random function, which is uniformly distributed between the interval $[0, CW]$ and *ST* is the length of a backoff time slot. The initial value of the *CW* is $CW_{\min}$. When a host wants to send data, it first sense the medium. If the medium is idle for a period of time equal to *DIFS*, the backoff procedure will decrease the backoff time, otherwise, it will stop decreasing the backoff time. When the backoff timer expires, the host will transmit the frame. After the sender transmits the frame, if it is a broadcast, the receivers do nothing. Otherwise, if it is a unicast, the receiver will wait for a period of time equals to *SIFS* (short interframe space, $SIFS < DIFS$) and then reply an *Ack* to the sender. If the sender does not receive an *Ack* from the receiver, the sender will double the size of its contention window and repeat the *DCF* procedure again.

2.3. Broadcast storms caused by flooding

A naive approach to achieve broadcast is by *flooding*. However, as shown in [12], flooding is very costly. For example, in Fig. 2(a) and (b), it shows that two transmissions are sufficient to resolve broadcasting in the two MANETs, but flooding will cost 4 and 7 messages, respectively. Through analyses, [12] shows that flooding in MANET may cause a lot of redundancy, contention, and collision. We

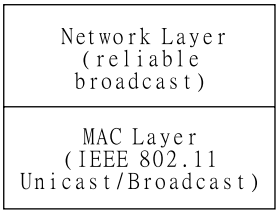

Fig. 1. Location of the proposed reliable broadcast in the protocol stack.

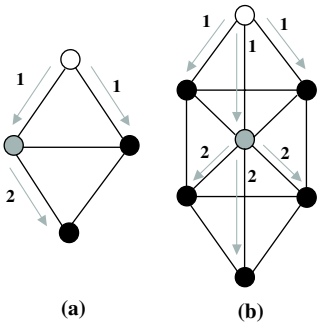

Fig. 2. Two optimal broadcasting schedules in MANETS. White nodes are sources, and gray nodes relaying hosts.

briefly summarize the disadvantages of flooding as follows:

- *Redundant rebroadcasts:* This happens when a mobile host tries to rebroadcast a broadcast packet to its neighbors, but all its neighbors have already received the packet.
- *Contention:* When a host's broadcast is heard by several of its neighbors, these neighbors' rebroadcasts will contend for the medium at around the same time. Since these hosts must be physically close, the contention might be heavy, especially when the environment is crowded.
- *Collision:* Because there is no RTS/CTS dialogue and no collision detection mechanism provided, collisions are more likely to occur. This may even cause some mobile hosts not receiving the broadcast packet.
- *Congestion:* With flooding, broadcasting may even be completed with longer latency.

2.4. Reviews of broadcasting protocols

2.4.1. The AV protocol

Alagar and Venkatesan [1] proposes a reliable broadcast protocol based on flooding (termed as *AV protocol* below). The source host should transmit the message through broadcast to all its 1-hop neighbors. Each receiver should return an acknowledgement to the sender, and has responsibility to broadcast the message again to all its 1-hop neighbors. If the sender does not receive an acknowledgement from any neighbor after a certain time interval, it rebroadcasts the message.

Apparently, reasons such as collision or temporary network disconnection may result in some hosts missing the broadcast message. The AV protocol uses a *handshake* procedure when two hosts meet each other to exchange their histories. On finding a missing broadcast message, a host can ask the other to supply it.

Potential drawbacks of the AV protocol include the broadcast storm effect as discussed earlier. The transmission of acknowledgements may further aggravate the storm effect. Requiring acknowledgements in fact demands each host maintain an up-to-date neighbor list (which is sometimes inhibitive due to host mobility). Any inaccuracy in the neighbor list may result in repeated resending of the broadcast message until timeout. In addition, one hidden problem is that a host has to acknowledge every neighbor who has sent the broadcast message to it, which causes a lot of redundancy. This has not taken missing acknowledgements into account; an acknowledgement which experiences collision will cause the sender to rebroadcast again.

One thing not addressed in the AV protocol is when and how a host can clear its history records. Since no collection of the stability of broadcasts is conducted, the history records may grow infinitely. However, we believe that this can be solved by some traditional global-state protocols in distributed computing theory [4,10].

2.4.2. The PR protocol

Pagani and Rossi [14] proposes a protocol based on a clustering structure (termed as *PR protocol* below). It is assumed that clusters are maintained in a regular manner by any of the earlier protocols [5,8]. To broadcast, a scattering phase is taken first, where the source host unicasts the message reliably to its cluster head, which will in turn unicast the message reliably along gateways to its neighboring cluster heads. Each cluster head then sends the message, through broadcast, to its cluster members. During the scattering phase, a *forwarding tree (FT)* is constructed. The cluster head of the source host is the root of the FT. Every host in the FT keeps its parent and children. Then a gathering phase is taken, where acknowledgements are collected by each cluster head and forwarded along the FT from leaves to the source. If the cluster head cannot gather all acknowledgements from its cluster members, it retransmits the message (until timeout). In case that a child cannot forward an acknowledgement to its *parent* successfully, it will send the acknowledgements to the root according to a unicast routing protocol. If the unicast routing protocol is not available, it broadcasts a *flood-ack* message to its neighboring cluster heads. Upon receiving the *flood-ack*, the neighboring cluster head will forward the acknowledgement along the FT, if possible; otherwise, it will continue broadcasting the *flood-ack* message.

When the root gathers all the acknowledgements, it will consider the broadcast message to be stable. Then another broadcast will be issued from the source to announce this fact. This can be achieved by another scattering similar to the earlier step.

There are several drawbacks associated with the PR protocol. First, the cluster structure might be maintained even when no broadcast is desired. Maintaining clusters could be costly when the network has mobility [7]. Second, requiring each cluster

head maintain an up-to-date cluster member list is not easy, and sometimes prohibitive due to host mobility. Any inaccuracy in the member list may result in repeated resending of the broadcast message by the cluster head (until timeout). A new host may be discovered by a cluster head after it has forwarded its acknowledgements, which may further complicate the problem. Third, it is very critical if the FT becomes broken during the gathering phase, because many *flood-ack*'s might be triggered. Fourth, cluster heads and gateways may become a traffic bottleneck (with a lot of contentions and collisions).

3. Our reliable broadcasting protocol

Our protocol tries to reduce the traffic overheads while ensuring reliability of the broadcast. The protocol works in three phases. In the first *scattering phase*, the source host tries to send the broadcast message to as many hosts in the MANET as possible, with efficiency in mind. A *handshake* procedure is used to keep those hosts who miss the broadcast message informed. Then a *gathering phase* is used to collect acknowledgements from all hosts. After the message is stabilized, a *purging phase* is initiated to inform all hosts to tear down data structures related to this broadcast. Note that both the scattering and purging phases will be based on an *unreliable* broadcast to reduce the traffic overheads. The reliability only counts on the second phase. This is what makes our approach different from, and more efficient than, existing protocols.

Each broadcast message M, when first initiated, is assigned a unique *id* (one simple format of *id* is to use a local sequence number plus the source host identity). Associated with each M, the following data structures are kept by each host x:

- $C(id)$: a counter to keep track of the number of times M is heard.
- $Par(id)$: an ordered list of hosts serving as x's parents in the upstream gathering tree (initially, $Par(id)$ is an empty list).
- $T_{\text{ack}}(id)$: the acknowledgement timer for M.

The following data structures are not for per broadcast, but are used by all broadcasts:

- C_{th}: the counter threshold.
- T_{hand}: the handshake timer.
- *hand_req*: handshake request bit.
- H: the broadcast history, which contains the following fields:
  - *stable_id*$[1\ldots n]$: an array of *id*s such that all broadcast messages of host i up to identity *stable_id*$[i]$ are stabilized. A broadcast message is called *stable* if the source host has gathered all the acknowledgements from all the hosts in the network; otherwise, it is *unstable*.
  - *unstable_msg*: a list of messages known by x but are unstable.
  - *unstable_id*: a list of *id*s corresponding to the unstable messages in *unstable_msg*.
  - *unstable_ack*: a list of acknowledgement vectors corresponding to the unstable messages in *unstable_msg*, where an *acknowledgement vector* is an n-bit stream such that the ith bit indicates whether host i has acknowledged the receipt of the broadcast message or not.
  - *pending_ack*: a list of acknowledgement packets that are unable to deliver through the upstream gathering tree collected by x (initially, it is an empty list).

The packets used in our protocol are listed below:

- $B(M, id)$: broadcast packet, with content M and identity *id*.
- $ACK(id, v)$: acknowledgement packet, for message *id* with acknowledgement vector v.
- $NACK(H.unstable\_id)$: negative acknowledgement.
- $HAND(H.stable\_id[1\ldots n], H.unstable\_id, H.pending\_ack)$: handshake packet.
- $PURGE(id)$: to notify all hosts that message *id* has stabilized.

3.1. Scattering phase

In the scattering phase, we use a *counter-based* scheme revised from [12]. The scheme tries to send the broadcast message to as many hosts as possible with the minimum efforts, but is unreliable. It works as follows. When a host receives a broadcast message $B(M, id)$ for the first time, it will set a counter $C(id) = 1$. Then $B(M, id)$ will be scheduled for rebroadcasting. However, before $B(M, id)$ is sent on the air, whenever the same message is heard again, $C(id)$ will be increased by 1. To avoid blind flooding (and thus the broadcast storm effect), before M is really transmitted, if $C(id)$ reaches the threshold C_{th}, $B(M, id)$ will be withdrawn from the message queue. The rationale is that most of

the neighbors of the host may have already received the broadcast message. In [12], it is shown surprisingly that in many situations such a simple scheme can deliver the broadcast message to more hosts, while saving much traffic as opposed to flooding.

In our protocol, we will adopt this counter-based scheme, in hope of propagating $B(M, id)$ to more hosts in the first round. In addition, a "weak" tree will be established through the message propagation. Specifically, hosts from which $B(M, id)$ is received will be regarded as the receiving host's parents, and appended to the list $Par(id)$. By doing so, multiple reverse paths pointing to the source host will be constructed. Later on, the receiving host will try to send acknowledgements to one of the hosts in $Par(id)$.

The protocol is formally shown by event-driven rules below for a host x. Note that in the following description, whenever we say "broadcast a packet", this means a *1-hop* broadcast and is unreliable, which should not be confused with the reliable broadcast problem discussed in this paper.

S1-1. On desiring to send a broadcast message $M \Rightarrow$
Obtain a unique id for M.
Append M to $H.unstable\_msg$.
Append id to $H.unstable\_id$.
Append an n-bit vector to $H.unstable\_ack$ with the xth bit, the only bit, equal to 1.
Broadcast $B(M, id)$.

S1-2. On receiving from a host y a broadcast message $B(M, id)$ for the first time $\Rightarrow$
$C(id) := 1$.
Append M to $H.unstable\_msg$.
Append id to $H.unstable\_id$.
Append an n-bit vector to $H.unstable\_ack$ with the x and yth bits, the only bits, equal to 1.
Append y to $Par(id)$.
Reset the timer $T_{ack}(id)$.
Put $B(M, id)$ in transmission queue for delivery.

S1-3. On receiving from another host y' a duplicate, but undelivered, $B(M, id) \Rightarrow$
$C(id) := C(id) + 1$.
if $(C(id) \geqslant C_{th})$ **then** remove $B(M, id)$ from the transmission queue.
Update the proper vector in $H.unstable\_ack$ by setting the y'th bit to 1.
Append y' to $Par(id)$.
Reset the timer $T_{ack}(id)$.

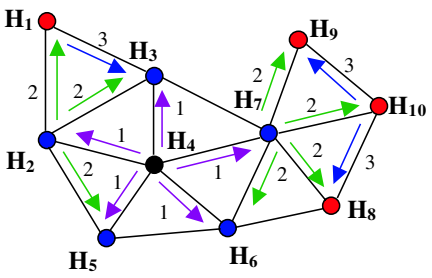

Fig. 3. A scattering example.

For example, consider the network in Fig. 3 with H_4 being the source host. Suppose the $C_{th} = 2$. After receiving H_4's broadcast message, H_2, H_3, H_5, H_6, and H_7 will add H_4 into their parent lists. Suppose H_2 and H_7 are the first hosts rebroadcasting this message. Then H_1, H_3, and H_5 will add H_2 into their parent lists, while H_6, H_8, H_9, and H_{10} will add H_7 to their parent lists. Finally, if H_{10} rebroadcasts first, H_8 and H_9 will add it into their parent lists. Now, all hosts' counters have reached 2, so there will be no further rebroadcasting. The final upstream gathering tree is as shown in Fig. 4.

Since the above counter-based scheme does not guarantee 100% delivery, we adopt a *handshake* mechanism for hosts to exchange broadcasting history. As defined earlier, each host keeps a broadcast history H. The history will be broadcast periodically controlled by the handshake timer T_{hand}. However, note that handshaking is only necessary when there are unstable broadcasts pending or the *hand_req* bit is set (to be explained later); otherwise, doing so is wasteful.

S2-1. On timer T_{hand} expiring $\Rightarrow$
if $(H.unstable\_id \neq \emptyset) \vee (hand\_req = TRUE)$ **then**

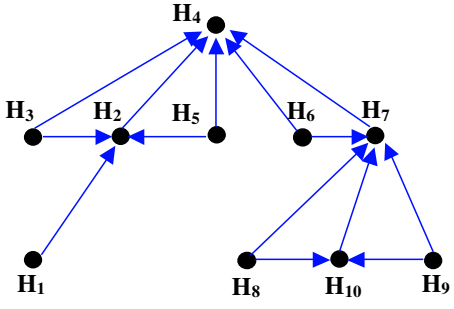

Fig. 4. The corresponding gathering tree.

Broadcast $HAND(H.stable\_id, H.unstable\_id, H.pending\_ack)$.[1]
$hand\_req := FALSE$.
end if.
Reset the timer T_{hand}.

The three fields in the handshake packet will help the receiver to discover which message has become stable, which message is missing, and which acknowledgement is undeliverable through the upstream gathering tree. The following **for**-loop performs two tasks: (i) the first **if**-statement is to tear down the related records in the history for each newly stabilized broadcast message, and (ii) the second **if**-statement is to set the handshake request bit to enforce the host to send a HAND packet in S2-1 (the reason will become clear in the purging phase). In the third **if**-statement, we adopt a *receiver-initiated* rule to acquire missing broadcasting messages. An NACK packet will be sent explicitly for this purpose. Finally, the pending acknowledgements are joined together (the purpose will be discussed later).

S2-2. On receiving $HAND(H'.stable\_id, H'.unstable\_id, H'.pending\_ack)$ from a host $y \Rightarrow$
for $i := 1$ **to** n **do**
if $(H'.stable\_id[i] > H.stable\_id[i])$ **then**
Remove the newly stabilized messages from H (from *unstable_msg*, *unstable_id*, and *unstable_ack*)
set $H.stable\_id[i] := H'.stable\_id[i]$.
elseif $(H'.stable\_id[i] < H.stable\_id[i])$ **then**
$hand\_req := TRUE$.
end if.
end for.
if $(H'.unstable\_id - H.unstable\_id \neq \emptyset)$ **then**
Send a $NACK(H.unstable\_id)$ to y through unicast.
end if.
$H.pending\_ack := H.pending\_ack \cup H'.pending\_ack$.

The NACK packet will trigger the receiving host to send the missing broadcasts to the requesting host, as shown below.

[1] Note again that this means a 1-hop broadcast.

S2-3. On receiving a $NACK(H'.unstable\_id)$ from host $y \Rightarrow$
For each id in $H.unstable\_id - H'.unstable\_id$, send $B(M, id)$ to y through unicast, where M is the corresponding broadcast message in $H.unstable\_msg$.

Note that the message is now sent through unicast and thus is more reliable than broadcast. However, note that this may still experience failure due to high host mobility. In any case, we will count on HAND and NACK packets to ensure reliability. Also, note that the above $B(M, id)$ will trigger the previous rules (such as S1-2 and S1-3). The treatment is similar.

3.2. Gathering phase

To ensure reliability, a host which has heard the broadcast message should return an ACK packet to the source. In our protocol, ACKs will be propagated following the pointers in $Par(id)$. To increase delivery rate, unicast is always used when sending acknowledgements. The first host in the list $Par(id)$ will be tried first. If after a number of trials the ACK still experiences failure, the next host in $Par(id)$ will be tried. If unfortunately no host in $Par(id)$ is available to accept the acknowledgement (which may be due to many reasons, such as mobility, congestion, and collision), this packet will be appended to $H.pending\_ack$. Then this becomes the handshake procedure's responsibility to deliver the ACK.

Another problem yet to be addressed is when a host should return its acknowledgement. One naive approach is to send immediately after the broadcast message is received. However, serious contention may occur among hosts who receive the broadcast message at around the same time. Also, one apparent goal is to combine multiple acknowledgements together and return them by one packet to reduce traffic. In the following, we propose an approach based on a timeout mechanism. Recall the timer $T_{\text{ack}}(id)$, which will be set in rule S1-2 and S1-3. Whenever it expires, the host should send its acknowledgement.

S3-1. On timer $T_{\text{ack}}(id)$ expiring $\Rightarrow$
Send $ACK(id, v)$ to the first host in $Par(id)$ through unicast, where v is the acknowledgement vector for message id from the $H.unstable\_ack$.

S3-2. On failing to deliver $ACK(id, v) \Rightarrow$
Delete the first element of $Par(id)$.
if $(Par(id) \neq \emptyset)$ **then**
Send $ACK(id, v)$ to the first host in $Par(id)$ through unicast.
else
Append $ACK(id, v)$ to $H.pending\_ack$.
end if

For example, consider the example in Fig. 5. Suppose the link from H_9 to H_7 is broken. Then H_9's ACK will be sent to H_{10}, which will forward the ACK to H_7. If unfortunately the link from H_7 to H_4 is also broken, this ACK will be appended to H_7's *pending_ack*. Now this becomes the handshaking packets' responsibility to forward the acknowledgement to the source H_4 (one possibility is to go through the link from H_7 to H_6 and then to H_4).

Even after sending its acknowledgement, the host may still receive ACKs from its downstream hosts. In this case, the host will include these acknowledgements, if they are not yet known to the host, into its broadcast history. Then timer $T_{ack}(id)$ will be set for forwarding these acknowledgements. In the following rule, $\bar{u}$ is the complement of u, $\bigcirc$ denotes bitwise logic-AND, and $\odot$ denotes bitwise logic-OR.

S3-3. On receiving $ACK(id, v) \Rightarrow$
Let u be the acknowledgement vector in $H.unstable\_ack$ corresponding to message id.
if $(\bar{u} \circ v \neq 0)$ **then**
$u := u \odot v$.
Reset $T_{ack}(id)$.
end if.

3.3. Purging phase

When the source host finds that a message (M, id) initiated by itself has stabilized, a PURGE packet should be broadcast to inform all hosts to tear down the related data structures. In our protocol, this can be found by checking the corresponding acknowledgement vector in $H.unstable\_ack$. We say that (M, id) can be *purged* if itself is stable and all other messages broadcast by the same host with smaller identities have also stabilized.

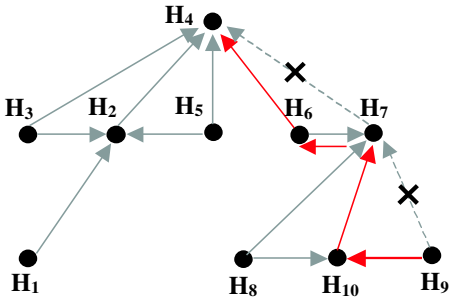

Fig. 5. A gathering example.

S4-1. On finding that a message (M, id) can be purged $\Rightarrow$
Remove the related entries in $H.unstable\_msg$, $H.unstable\_id$, $H.unstable\_ack$, and $H.pending\_ack$.
$H.stable\_id[x] := id$.
Broadcast $PURGE(id)$.

The PURGE packet is also broadcast through the unreliable counter-based scheme. So the following rules are similar to S1-2 and S1-3.

S4-2. On receiving $PURGE(id)$ for the first time $\Rightarrow$
Remove the related entries in $H.unstable\_msg$, $H.unstable\_id$, $H.unstable\_ack$, and $H.pending\_ack$.
$H.stable\_id[x] := id$.
$C(id) := 1$.
Put $PURGE(id)$ in transmission queue for delivery (through broadcast).

S4-3. On receiving a duplicate, but undelivered, $PURGE(id) \Rightarrow$
$C(id) := C(id) + 1$.
if $(C(id) \geqslant C_{th})$ **then** remove $PURGE(id)$ from the transmission queue.

Since PURGE is sent through an unreliable broadcast, some host may miss this message. In this case, this host (which owns unstable message) will periodically broadcast HAND packets. This will trigger other hosts (which already received the PURGE packet) to set their *hand_req* bits in S2-2; this will in turn trigger them to perform S2-1 to broadcast HAND packets. Thus, as long as the network remains connected, eventually every host will realize which broadcasts have been stabilized and tear down the related data structures.

3.4. Optional rules

In our protocol, ACK packets are sent through unicast. However, due to radio's broadcasting

nature, they can still be overheard by other irrelevant hosts (using a promiscuous mode). Thus, a host can use an overheard ACK to identify which broadcast message it has missed. This may help the host to request for missing message(s) at an earlier time (as opposed to using handshaking). As shown below, this is implemented by sending a NACK, which will trigger S2-3 to send the missing message(s).

S5-1. On overhearing $ACK(id, v)$ from host y such that id is not in $H.unstable\_id \Rightarrow$
Send a NACK($H.unstable\_id$) to y through unicast.

3.5. Memory requirement and computation cost

Assume that there are n hosts in the networks, the broadcast rate is r broadcasts/s, and the maximum time for a broadcast to become stable is s seconds. Each host will require $O(n)$ memory space to maintain the necessary data structure for each unstable broadcast and each host will require at most $O(srn)$ memory space to maintain the necessary data structure for all the unstable broadcasts. Assume that a host has at most k neighbors. It will take at most $O(kn)$ computations for a host to merge the gathered acknowledgements in each reliable broadcast.

4. Simulation results

To compare our protocol to other protocols, we have developed a simulator using C. The MAC part follows the IEEE 802.11 standard [13]. In our simulations, the following parameters are fixed: transmission radius = 250 m, area size = 1000 m × 1000 m, transmission rate = 2M bits/s, beacon interval = 1 s, pause time = 30 s, contention window $MIN\_CW = 31$ and $MAX\_CW = 1023$, retry limit = 5, $T_{\text{hand}} = 1$ s, and $T_{\text{ack}} = MAX\_CW \times ST$.

We adopt the random waypoint mobility model [3]: each node randomly generates a destination in the simulated region, and then move to the destination at the chosen velocity. Upon arriving at the destination, the host pauses for a certain period of time and then repeats the same procedures until the simulation is over.

Five parameters are tunable in our simulations:

- Counterthreshold = 2–5.
- Moving speed of hosts = 0–20 m/s.
- Broadcast rate = Broadcasts are generated by a Poisson distribution with rate between 1 and 4 broadcasts/s (in the whole network).
- Number of hosts = 50–250.
- Broadcast packet size = 256–1024 bytes.

We mainly compare to the aforementioned AV and PR protocols. In our simulation, the protocol proposed in [8] is adopted to maintain PR's clustering structure. Each simulation lasts for 100 s and the simulation results are derived from the average of 100 simulations. Three performance metrics are used:

- Average data traffic: the average amount of data (in bytes) transmitted per host for each reliable broadcast request.
- Average control traffic: the average amount of control packets (in bytes) transmitted per host for each reliable broadcast request (including the beacon frame, broadcast ACK, ACK for unicast, NACK, and handshake packets).
- Average latency: the interval from the time the source host initiating the broadcast to the time all the hosts have received the broadcast packet.

In the following sections, we make observations from several aspects.

4.1. Impact of counterthreshold

To observe the effect of counterthreshold (C_{th}), we vary C_{th} between 2 and 5. Fig. 6 shows the impact of counterthreshold. As C_{th} increases, only a few hosts would receive C_{th} or more duplicate broadcast packets, hence, most of the hosts would rebroadcast the broadcast packet, and thus

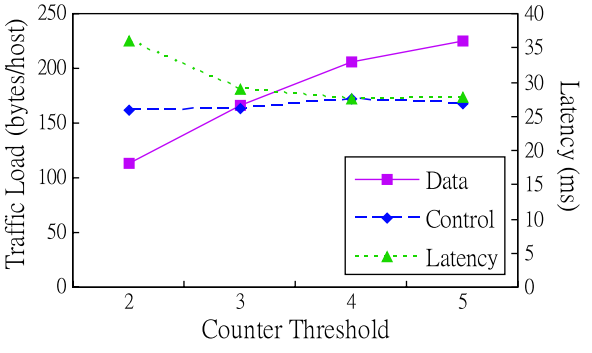

Fig. 6. Impact of counterthreshold (number of hosts = 100, mobility = 5 m/s; rate = 1 broadcast/s; packet size = 256 bytes).

increases the data traffic. On the other hand, all the hosts still need to transmit the ACK packet through the gathering tree no matter what C_{th} is set, C_{th} has little effect to the control traffic. As for the latency, when C_{th} increases, since most of the hosts need to rebroadcast the broadcast packet, we can always achieve 100% delivery ratio during the scattering phase, and thus reduces the latency. However, when C_{th} is small, a few hosts may not be able to receive the broadcast packet during the scattering phase, they may retrieve the broadcast packet from their neighbors during the gathering phase, and thus increases the latency. Overall, when C_{th} is set as 2, the traffic load is the lowest; when C_{th} is set as 3, the traffic load is lower than that of the higher C_{th} and the latency is close to that of the higher C_{th}. Therefore, we will set C_{th} as 2 and 3 in the following simulations. For the ease of presenting the simulation results, our protocol is denoted as Ours(C_{th}).

4.2. Impact of host mobility

In this section, we vary the moving speed between 0 and 20 m/s to observe the impact of mobility. The broadcast rate is fixed to 1 broadcast/s with packet size of 256 bytes.

As Figs. 7–9 show, all the three protocols' traffics and latency increase as the moving speed increases. Our protocol incurs lower traffic load and latency than AV and PR do in all cases. AV is sensitive to mobility because it is based on each host's neighbor list. As mobility increases, hosts' neighbor list becomes inaccurate, which causes unnecessary rebroadcast, and thus increases traffic load and latency. PR's control traffic and data traffic increase as mobility increases due to higher cluster maintenance cost and retransmission cost as cluster membership changes. As to the broadcast latency of PR, it takes longer time to complete as mobility increases, because the cluster head and the new cluster members will not realize each other until their beacon frames are received by each other. Mobility has a little impact to our protocol because it is a partially counterbased, partially tree-based approach. The handshake mechanism and the existence of secondary parents help acknowledgement packets to be transmitted to source hosts but cause some extra traffics and latency. *Ours*(2) incurs lower traffic and higher latency than *Ours*(3) does because of the reason mentioned in Section 4.1.

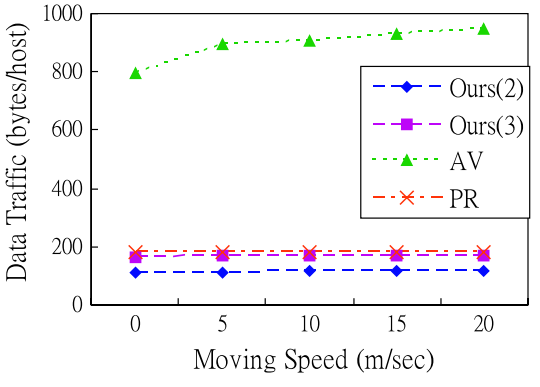

Fig. 7. Average data traffic vs. mobility (number of hosts = 100; rate = 1 broadcast/s; packet size = 256 bytes).

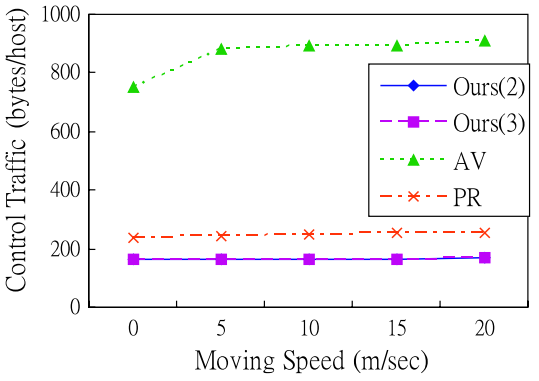

Fig. 8. Average control traffic vs. mobility (number of hosts = 100; rate = 1 broadcast/s; packet size = 256 bytes).

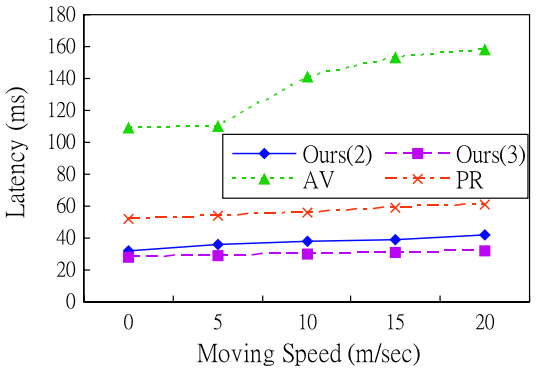

Fig. 9. Average latency vs. mobility (number of hosts = 100; rate = 1 broadcast/s; packet size = 256 bytes).

4.3. Impact of host density

In this simulation, we tune the number of hosts to see its impact. Since the area size is fixed, tuning the number of hosts in fact reflects the host density.

As shown in Figs. 10–12, our protocol is the best in all density ranges. The data traffic of PR and our protocol decreases, as the density increases. In a high density network, since higher ratio of hosts will receive C_{th} or more duplicate broadcast packets, lower ratio of hosts will rebroadcast the packet and thus decreases the data traffic. Similarly, since PR protocol is a cluster-based protocol, lower ratio of hosts will become cluster heads and rebroadcast the packet and thus decreases the data traffic. In PR and our protocols, since every host needs to forward its ACK packet to the source host, higher density of host will incur more collisions and retransmissions, and thus increases the control traffic and latency. As for the AV protocol, the traffic load increases when host density increases. This is even more significant for latency. This is because there is more contentions and collisions when the network is dense (recall the broadcast storm problem reviewed earlier).

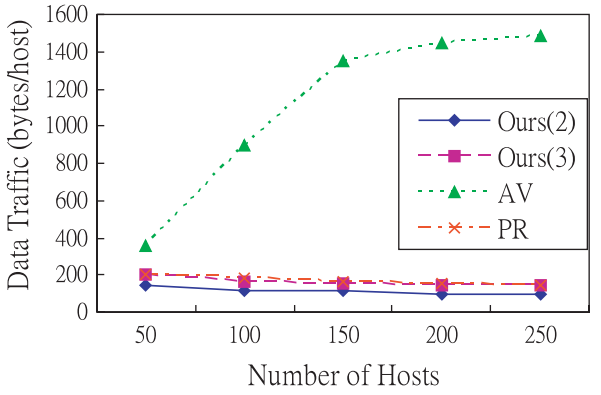

Fig. 10. Average data traffic vs. host density. (mobility = 5 m/s; rate = 1 broadcast/s; packet size = 256 bytes).

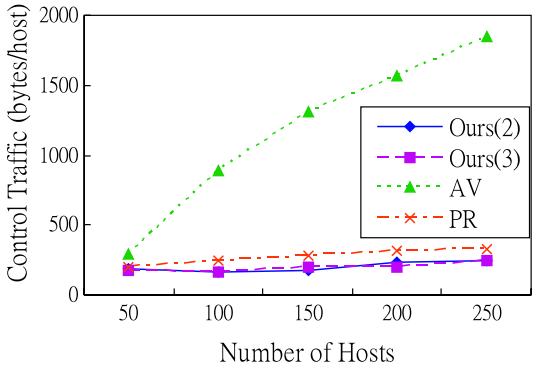

Fig. 11. Average control traffic vs. host density (mobility = 5 m/s; rate = 1 broadcast/s; packet size = 256 bytes).

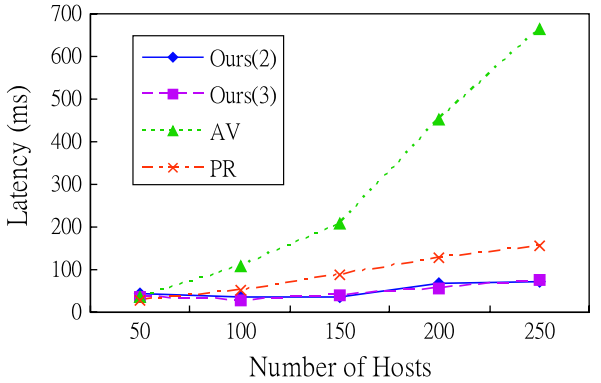

Fig. 12. Average latency vs. host density (mobility = 5 m/s; rate = 1 broadcast/s; packet size = 256 bytes).

4.4. Impact of broadcast rate

In this experiment, we vary the broadcast rate. Figs. 13 and 14 show that AV, PR and our protocols will have higher traffic as rate increases due to

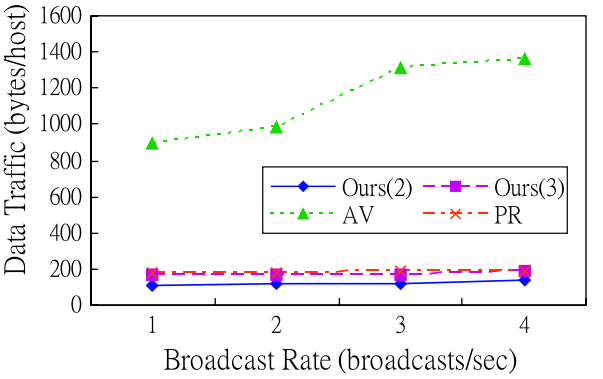

Fig. 13. Average data traffic vs. broadcast rate (number of hosts = 100; mobility = 5 m/s; packet size = 256 bytes).

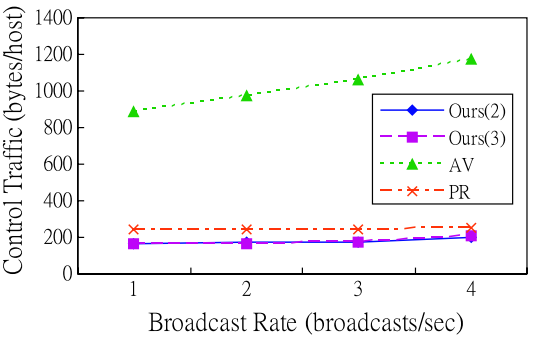

Fig. 14. Average control traffic vs. broadcast rate (number of hosts = 100; mobility = 5 m/s; packet size = 256 bytes).

higher chances of collisions, which cause retransmission. Fig. 15 compares the average latency. Our protocol still outperforms others at high traffic load.

4.5. Impact of packet size

In wireless communication, larger packets suffer from high chances of collision. This is particularly true for broadcast since no RTS/CTS dialogue is used. Figs. 16–18 demonstrate this effect on these three protocols. As the packet size increases, collisions, retransmissions, and transmission time for each broadcast packet also increases, and thus incurs more traffic load and longer latency.

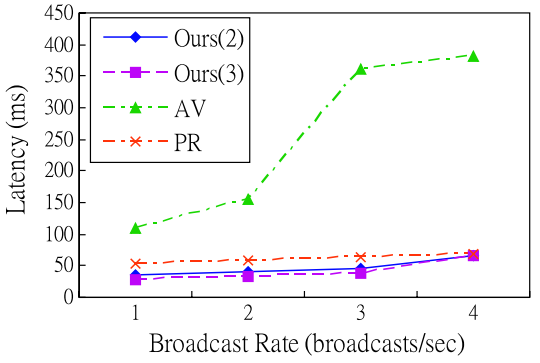

Fig. 15. Average latency vs. broadcast rate (number of hosts = 100; mobility = 5 m/s; packet size = 256 bytes).

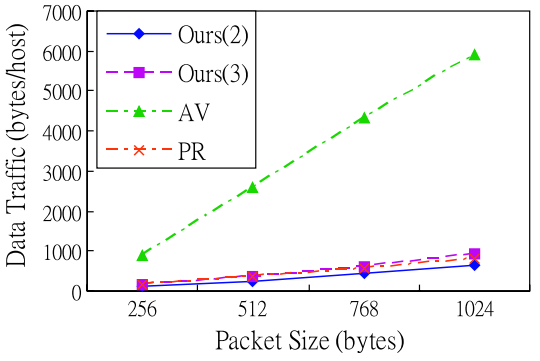

Fig. 16. Average data traffic vs. packet size (number of hosts = 100; mobility = 5 m/s; rate = 1 broadcast/s).

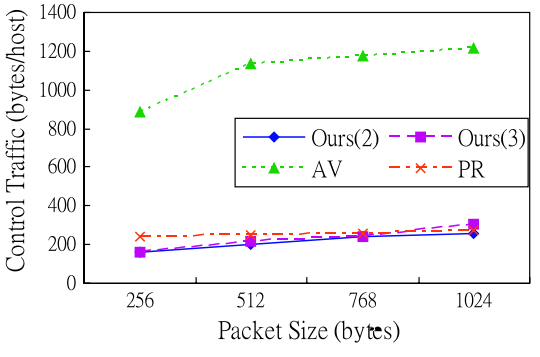

Fig. 17. Average control traffic vs. packet size. (number of hosts = 100; mobility = 5 m/s; rate = 1 broadcast/s).

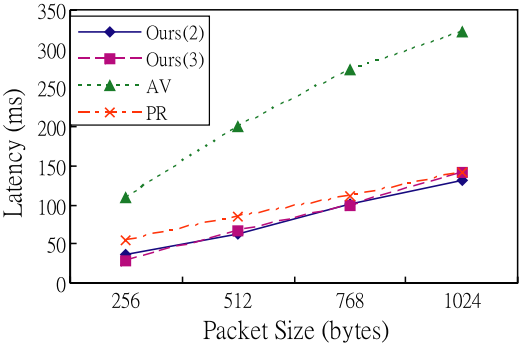

Fig. 18. Average latency vs. packet size (number of hosts = 100; mobility = 5 m/s; rate = 1 broadcast/s).

5. Conclusions

We have proposed an efficient reliable broadcast protocol. This protocol takes a mixture of counter-based and tree-based approach. The counterbased mechanism relieves the broadcast storm effect. Our tree is in fact a very loose tree—in order to return acknowledgement, a host tries to maintain multiple parents as reverse paths to the source host. The tree structure is not actually maintained, and a handshake mechanism is used to resolve this problem when the tree is broken. We also avoid using the brute-force approach that is adopted by existing schemes that a host will repeatedly rebroadcast the same broadcast packet if it does not receive acknowledgements of the packet from all of its neighbors. Through simulations, we show the advantage of our protocol in situations with various parameters.

References

[1] S. Alagar, S. Venkatesan, Reliable broadcast in mobile wireless network, in: Proceedings of the Military Communications Conference, vol. 1, 1995, pp. 236–240.
[2] S. Basagni, D. Bruschi, I. Chlamtac, A mobility-transparent deterministic broadcast mechanism for ad hoc networks, IEEE Transactions on Networking 7 (1999) 799–807.
[3] J. Broch, D.A. Maltz, D.B. Johnson, Y.C. Hu, J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, in: Proceedings of the International Conference on Mobile Computing and Networking, 1998, pp. 85–97.

[4] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Transactions on Computer Systems 1 (1985) 63–75.
[5] C.C. Chiang, M. Gerla, Routing and multicast in multihop mobile wireless networks, in: Proceedings of the IEEE International Conference on Universal Personal Communications, vol. 2, 1997, pp. 546–551.
[6] I. Chlamtac, A.D. Myers, V.R. Syrotiuk, G. Zaruba, An adaptive medium access control (mac) protocol for reliable broadcast in wireless networks, in: Proceedings of the IEEE International Conference on Communications, vol. 3, 2000, pp. 1692–1696.
[7] M. Gerla, T.J. Kwon, G. Pei, On demand routing in large ad hoc wireless networks with passive clustering, in: Proceedings of the IEEE Wireless Communications and Networking Conference, vol. 1, 2000, pp. 100–105.
[8] M. Gerla, J.T.C. Tsai, Multicluster, mobile, multimedia radio network, Wireless Networks 1 (1995) 255–265.
[9] I. Koutsopoulos, D. Connors, A. Savvides, S.K. Dao, Intra-team multi-hop broadcasting (itmb): a mac layer protocol for efficient control signaling in wireless ad-hoc networks, in: Proceedings of the IEEE International Conference on Communications, vol. 3, 2000, pp. 1723–1727.
[10] H.F. Li, T. Radhakrishnan, K. Venkatesh, Global state detection in non-fifo networks, in: Proceedings of the 7th International Conference on Distributed Computing Systems, 1987, pp. 364–370.
[11] C.R. Lin, M. Gerla, Adaptive clustering for mobile wireless networks, IEEE Journal of Selected Areas in Communications 15 (1997) 1265–1275.
[12] S.Y. Ni, Y.C. Tseng, Y.S. Chen, J.P. Sheu, The broadcast storm problem in a mobile ad hoc network, in: Proceedings of the International Conference on Mobile Computing and Networking, 1999, pp. 151–162.
[13] L.M.S.C. of the IEEE Computer Society. Ieee std 802.11-1997, wireless lan medium access control (mac) and physical layer (phy) specifications. IEEE, 1997.
[14] E. Pagani, G.P. Rossi, Providing reliable and fault tolerant broadcast delivery in mobile ad-hoc networks, Mobile Networks and Applications 4 (1999) 175–192.
[15] W. Peng, X.C. Lu, On the reduction of broadcast redundancy in mobile ad hoc networks, in: Proceedings of the First Annual Workshop on Mobile and Ad Hoc Networking and Computing, 2000, pp. 129–130.
[16] S.T. Sheu, Y.J. Tsai, J.H. Chen, A highly reliable broadcast scheme for ieee 802.11 multi-hop ad hoc networks, International Conference on Communications (2002) 610–615.
[17] I. Stojmenovic, M. Seddigh, J. Zunic, Internal nodes based broadcasting algorithms in wireless networks, in: Hawaii International Conference on System Sciences, 2001.
[18] M.T. Sun, T.H. Lai, Computing optimal cover set for broadcast in ad hoc network, International Conference on Communications (2002) 3291–3295.
[19] M.T. Sun, T.H. Lai, Location aided broadcast in wireless ad hoc network systems, International Conference on Communications (2002) 597–602.
[20] K. Tang, M. Gerla, Mac layer broadcast support in 802.11 wireless networks, in: Proceedings of the Military Communications Conference, 2000, pp. 544–548.
[21] K. Tang, M. Gerla, Mac reliable broadcast in ad hoc networks, Military Communications Conference (2001) 1008–1013.
[22] J. Tourrilhes, Robust broadcast: improving the reliability of broadcast transmissions on csma/ca, IEEE International Symposium on Personal, Indoor and Mobile Radio Communications 3 (1998) 1111–1115.
[23] Y.C. Tseng, S.Y. Ni, E.Y. Shih, Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc network, International Conference on Distributed Computing Systems (2001) 481–488.
[24] J.E. Wieselthier, G.D. Nguyen, A. Ephremides, On the construction of energy-efficient broadcast and multicast trees in wireless networks, in: Proceedings of INFOCOM, 2000, pp. 585–594.
[25] J. Wu, H. Li, On calculating connected dominating set for efficient routing in ad hoc wireless networks, in: Proceedings of DIAL-M, 1999, pp. 7–14.

Chih-Shun Hsu received his B.S. degree in computer education from National Taiwan Normal University, Taiwan, in 1990, and the M.S. degree in computer science from National Taiwan University, Taiwan, in 1992. He obtained his Ph.D. degree in Computer Science from National Central University, Taiwan, in 2004. He joined the faculty of the Department of Information Management, Nanya Institute of Technology, Taiwan, as an instructor in 1996, and has become an associate professor since August 2004. He has become the chairman of the Department of Computer Science and Information Engineering, Nanya Institute of Technology, since August 2005. His current research interests include wireless communications and mobile computing.

Yu-Chee Tseng received his Ph.D. in Computer and Information Science from the Ohio State University in January of 1994. He is currently a Professor and the Chairman at the Department of Computer Science, National Chiao-Tung University. He served as a Program Chair in the *Wireless Networks and Mobile Computing Workshop*, 2000 and 2001, as a Vice Program Chair in the *Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2004, as a Vice Program Chair in the *IEEE Int'l Conf. on Mobile Ad-hoc and Sensor Systems (MASS)*, 2004, as an Associate Editor for *The Computer Journal*, as a Guest Editor for *ACM Wireless Networks* special issue on "Advances in Mobile and Wireless Systems", as a Guest Editor for *IEEE Transactions on Computers* special on "Wireless Internet", as a Guest Editor for *Journal of Internet Technology* special issue on "Wireless Internet: Applications and Systems", as a Guest Editor for *Wireless Communications and Mobile Computing* special issue on "Research in Ad Hoc Networking, Smart Sensing, and Pervasive Computing", as an Editor for *Journal of Information Science and Engineering*, as a Guest Editor for *Telecommunication Systems* special issue on "Wireless Sensor Networks", and as a Guest Editor for *Journal of Information Science and Engineering* special issue on "Mobile Computing".

He received the Outstanding Research Award, by National Science Council, ROC, in both 2001–2002 and 2003–2005, the Best Paper Award, by Int'l Conf. on Parallel Processing, in 2003, the Elite I.T. Award in 2004, and the Distinguished Alumnus Award, by the Ohio State University, in 2005. His research interests include mobile computing, wireless communication, network security, and parallel and distributed computing. He is a member of ACM and a Senior Member of IEEE.

Jang-Ping Sheu received the B.S. degree in computer science from Tamkang University, Taiwan, Republic of China, in 1981, and the M.S. and Ph.D. degrees in computer science from National Tsing Hua University, Taiwan, Republic of China, in 1983 and 1987, respectively.

He joined the faculty of the Department of Electrical Engineering, National Central University, Taiwan, Republic of China, as an Associate Professor in 1987. He is currently a Professor of the Department of Computer Science and Information Engineering and Director of Computer Center, National Central University. He was a Chair of Department of Computer Science and Information Engineering, National Central University from 1997 to 1999. He was a visiting professor at the Department of Electrical and Computer Engineering, University of California, Irvine from July 1999 to April 2000. His current research interests include wireless communications, mobile computing and parallel processing. He was an associate editor of Journal of the Chinese Institute of Electrical Engineering, from 1996 to 2000. He was an associate editor of Journal of Information Science and Engineering from 1996 to 2002. He was an associate editor of Journal of the Chinese Institute of Engineers from 1998 to 2004. He is an associate editor of the IEEE Transactions on Parallel and Distributed Systems and International Journal of Ad Hoc and Ubiquitous Computing. He was a Program Chair of IEEE ICPADS'2002.

He received the Distinguished Research Awards of the National Science Council of the Republic of China in 1993–1994, 1995–1996, and 1997–1998. He was the Specially Granted Researchers, National Science Council, from 1999 to 2005. He received the Distinguished Engineering Professor Award of the Chinese Institute of Engineers in 2003. He received the Distinguished Professor award of the National Central University in 2005. He is a senior member of the IEEE, a member of the ACM, and Phi Tau Phi Society.