Providing Reliable and Fault Tolerant Broadcast Delivery in Mobile ad-hoc Networks

Elena Pagani, Gian Paolo Rossi

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
via Comelico, 39. I-20135 Milano, Italy
E-mail: {pagae,rossi}@dsi.unimi.it

Abstract Mobile ad-hoc networks are making a new class of mobile applications feasible. They benefit of the fast deployment and reconfiguration of the networks, are mainly characterized by the need to support many-to-many interaction schema within groups of cooperating mobile hosts and are likely to use replication of data objects to achieve performances and high data availability. This strong group orientation requires specialized solutions that combine adaptation to the fully mobile environment and provide the adequate level of fault tolerance. In this paper, we present the reliable broadcast protocol that has been on purposely designed for mobile ad-hoc networks. The reliable broadcast service ensures that *all* the hosts in the network *deliver* the same set of messages to the upper layer. It represents the building block to obtain higher broadcast and multicast services with stronger guarantees and is an efficient and reliable alternative to flooding. The protocol is constructed on top of the wireless MAC protocol, which in turn sits over the clustering protocol. It provides an exactly once message delivery semantics and tolerates communication failures and host mobility. Temporary disconnections and network partitions are also tolerated under the assumption that they are eventually repaired, as specified by a liveness property. The termination of the protocol is proved and complexity and performance analysis are also provided.

Keywords: mobile ad-hoc networks, clustered architecture, reliable broadcast, fault-tolerance.

1. Introduction

We consider the problem of providing reliable broadcast delivery of messages in fully mobile networks. The term fully mobile indicates a network in which the host mobility arbitrarily changes the network connectivity and where an infrastructure of base stations is not present. The host freedom to move about without any infrastructural constraints is enabling the construction of 'mobile information systems' whose use is mainly associated to specialized applications such as, vehicle traffic, fleet and events management, emergency disaster relief, rescue and battlefield operations. They generally involve a small set of mobile hosts that are supposed to cooperate tightenly. For this reason, we consider the term ad-hoc mobile networks well suited to our purposes. Most of such mobile systems have in common the applications' need to support many-to-many and one-to-many interaction schema amongst co-operating hosts and are likely to use replication of data objects to achieve performances and high data availability, despite moving or staying in a congested zone. The need to support such group operations is bringing together the requirements of (i) a set of transport services that address the group paradigm and provide Quality of Service-based broadcast/multicast and the adequate level of fault tolerance, and, (ii), a broadcast/multicast routing platform to efficiently route packets to the entire population of the network nodes or to a subset of them.

In literature, a few proposals have been recently presented that describe (unreliable) multicast solutions for ad-hoc networks based on specialized routing infrastructures [4,13]. Very few are the proposals that address the problem of reliable delivery to a group in the presence of mobile hosts (e.g. [6] and [1,2]). They mainly consider mobility in single-hop networks where the presence of fixed stations can be exploited. In this paper, we present a novel solution to the reliable broadcast problem that is intended for use in fully mobile networks. The reliable broadcast service ensures that *all* the mobile hosts in the network *deliver* the same set of messages to the upper layer. This represents a strong delivery guarantee, whose main issue is to ensure reliability and fault tolerance despite mobility that affects the possibility of maintaining a routing infrastructure based on some knowledge of (the whole or a part of the) network topology.

We consider mobile systems where the rate of topology change is difficult to predict and can vary in a wide range. In this context, our protocol is a first attempt to show that the reliable broadcast service can be achieved as a primitive, low level service in a wireless network protocol stack and that a mechanism more reliable and more efficient than flooding can be obtained without assumption about the host mobility. The importance of having a low level broadcast service with quite strong delivery guarantees may have different motivations: 1) it drastically reduces the complexity to obtain higher multicast services with stronger guarantees (e.g. causal and total ordering guarantees), independently of whether they are obtained as services of the protocol stack or provided at the application layer; 2) it can benefit of the services of the MAC protocol to achieve performances, to control the protocol message growth and to allow the implementation of an integrated protocol stack for ad-hoc mobile networks; 3) it can provide an alternative to flooding and ease the construction of other basic services (e.g. we are designing a link state-based routing protocol for small ad-hoc networks which uses a version of the reliable broadcast).

The protocol is composed of two phases: the scattering phase, in which the message is diffused to all receiver members, and the gathering phase, in which acknowledgements are collected from the receivers. The adopted routing infrastructure, the forwarding tree, or FT, is constructed during the scattering phase and does not require any route maintenance mechanism. Its life time is at most the time required to broadcast a message and to gather acknowledgements. We have adopted, for multi-hop mobile networks, the same packet recovery policy that in [1,2] is used for single-hop mobile networks with fixed stations. We show that in presence of mobility, the use of acknowledgements allow to quickly intervene with focused recovery actions in the zone where the failure occurred, before it affects a larger set of network resources. The use of a distributed recovery mechanism together with the FT actually reduces in most cases the amount of the generated messages. However, when the rate of change becomes too high and the FT vanishes, the protocol switches to flooding (if an underlying unicast routing protocol is not available), but, since we expect that the mobility rate is not uniformly spread over all the hosts, the protocol reacts by adapting the transmission according to the situations encountered in the different network zones.

The rest of the paper is organized as follows. In section 2 we describe the adopted system and failure model; in section 3, we give an informal definition of the reliable broadcast problem and introduce some properties that are used to ensure the protocol termination. In section 4, we present the protocol under different mobility conditions: static hosts, mobile hosts, mobile clusterheads. In section 5 we informally provide the correctness proof of the protocol. Section 6 gives some preliminary numerical analysis. In section 7, we describe how the protocol can be adapted to work in single-hop networks, and we compare it with an existing broadcast protocol for these environments.

2. The system and failure model

The system we consider is composed of N mobile hosts, or hosts for short, with unique identifiers $p_1, \ldots, p_N$, that communicate via a packet radio network. We assume that all the mobile hosts have similar computing and storage resources. The topology of the multihop packet radio network can be described as a graph $G = (N, L)$, where N is the set of hosts and L the set of logical links. We indicate as *one-hop neighbours* two hosts that can receive from one another.

All communications are broadcast, that is, when a host p_i sends a message m, m is heard by all the p_i's one-hop neighbours. In this work, with the term *unicast* message, unless differently specified, we mean a one-hop message addressed to only one of the one-hop neighbours of the sender host; all the sender's neighbours hear the message, but only the destination delivers it while the other hosts discard it.

To reach the destinations k hops away, we assume that a *clustering* protocol, exploiting the underlying neighbourhood information[1], is executed to construct

[1] The neighbourhood information is obtained by having each mobile host periodically issuing an "`I-am-alive`" signal heard by

a clustered architecture covering the entire population of system hosts. Most of existing clustered architectures are based on the notion of *clusterhead* and can be used for our purposes; the clusterhead coordinates the transmissions within the cluster and represents the infrastructure to route inter-cluster packets. By following, for instance, the approaches described in [8,11], the clustered architecture of figure 1 can be obtained. Each node is a one-hop neighbour of the clusterhead and at most two hops far from the other members of the cluster. Two clusterheads cannot be neighbours, while they

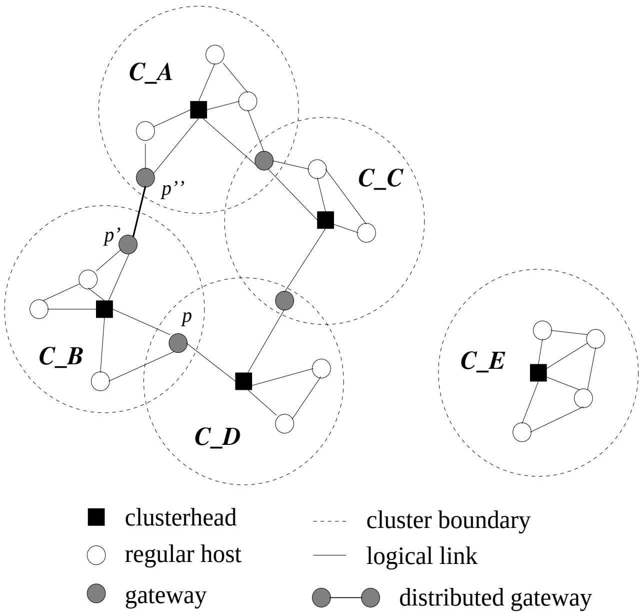

Figure 1. Example of a clustered multi-hop network.

can:

- be connected through a *gateway*, that is a host able to hear from different clusterheads (such as the clusters C_B and C_D through the host p in figure 1);
- be connected through a *distributed gateway*, that is a couple of hosts belonging to different clusters but that are able to communicate with each other (such as the clusters C_A and C_B through the hosts p' and p'' in figure 1);
- be disconnected (such as the clusters C_C and C_E in figure 1). In this case, the network is partitioned;
- be *indirectly* connected via another cluster, such as for instance C_A and C_D in figure 1.

Each gateway chooses one of the clusterheads it hears from as its *primary* clusterhead. In this work, we indicate with ch_A the clusterhead of a cluster C_A, while the gateway among two clusterheads ch_A and ch_B is indicated with gw_{AB}.

The clustering protocol is re-executed in the event of topology changes. It provides a framework for code separation and re-use among clusters. We assume that the reliable broadcast protocol is layered on top of the MAC protocol that is on top the clustering protocol; the reliable broadcast protocol is independent of both of them. No specific routing information are required (e.g. such as those provided by the protocols described in [7,4]), because the protocol mostly uses broadcast and one-hop unicast messages. If some unicast routing protocol were available, it could be placed below our protocol to avoid flooding that is activated during the gathering phase under high mobility conditions.

We consider a system where hosts do not suffer from permanent failures; all types of failures are temporary, although it is not specified how long the faulty condition will last. Similarly, both host disconnections and network partitions, such as the case of cluster C_E in figure 1, are tolerated under the assumption that they are sooner or later repaired. These assumptions are suitable to model mobility and communication failures over wireless links and are based on a precise design choice addressed to reduce the complexity at this level of the protocol architecture to simplify the construction of a membership protocol. Through the membership protocol, a set of hosts obtains a common agreed view of the group membership which accounts for membership changes. Changes occur when hosts voluntarily join or leave the group or when they crash; by our failure assumption, the membership management at this level only processes the application requests to join and leave the group. This makes the problem feasible[2]. If required, the identification of crashed hosts can be performed elsewhere to satisfy specific application requirements.

We consider a many-to-many interaction scheme where the probability of generating a message is uniformly distributed among the hosts.

its neighbours.

[2] In [9] the impossibility of reaching the agreement on the group membership in the presence of the crash of one host is proved.

3. Definitions and assumptions

The system described in section 2 does not guarantee the message delivery (i.e. the indication of the message reception to the upper layer) to all destinations when communication failures or topology changes occur. This service can be provided by the reliable broadcast protocol. The mobility and the adopted failure model lead to a slightly different problem definition with respect to the one usually adopted in literature [12]:

Definition 3.1 (Reliable Broadcast).Given a network of N hosts in which each host, at any time, can generate a broadcast message, messages are delivered to all receivers so that the following properties are guaranteed:

Validity and Agreement: if a host broadcasts a message m then all the hosts in the network receive m (at-least-once delivery).

Integrity: for each message m, all the hosts delivers m at most once, and only if m has been broadcast by a host in the network.

Termination: if a host p broadcasts a message m, then the Integrity, Validity and Agreement properties are satisfied for m within a finite time.

According to the given definition, the message delivery is guaranteed to *all* the hosts in the network, and the reliability requirement is extended to *all* the generated messages. By the definition, an exactly-once delivery service semantics is provided. To satisfy the general requirement of resource saving [14], an additional property is considered with respect to the diffused messages:

Definition 3.2 (Eventual subsidence).For each broadcast message m, eventually no more messages are generated regarding m and m can be deleted from the hosts memory.

This property guarantees that eventually all the processing, storage and bandwidth resources are released.

Faulty conditions and host mobility delay the protocol termination. To prevent the protocol execution to run indefinitely, the following properties must be satisfied:

Property 3.1 (Liveness).Each host p is free to arbitrarily move and to be temporarily disconnected. However, a given time exists, in which p is connected to a clusterhead ch and it remains connected the time needed to complete the exchange of status information with ch. Furthermore, if p, or ch, knows a message that has not been received by the other, then p eventually remains connected to ch the time required to successfully exchange the message before the next topology change[3].

Property 3.2 (Clusterhead stability).A host eventually acts as clusterhead the time needed to guarantee that the exchange of status information with the other clusterheads is successfully completed and, if there are partially diffused messages, at least one of them is successfully exchanged amongst the clusterheads.

The above properties do not specify time constraints to mobility or give some upper bound to the disconnection time of a given host. Time constraints are not required to guarantee the eventual termination of the protocol; they are likely to be specified by the application requirements and specific conditions can be handled directly at that level. In fact, as long as the liveness property is not satisfied, a host destination of a message m could either (i) never receive m, or (ii) eventually receive m, but could never succeed in acknowledging m. If the clusterhead stability property is not guaranteed, then the messages could be never successfully diffused throughout different clusters. In both cases, the protocol cannot terminate.

4. The Reliable Broadcast protocol

We initially introduce the service primitives and the adopted routing infrastructure. We explain how the protocol works in the case of static topology with transient link failures; then, we discuss the mechanisms used to cope with host mobility and changing clusterheads. Throughout this work, we assume that messages are uniquely identified by the pair $\langle$*sender ID, sender sequence number*$\rangle$. A message is *stable* when it has been received by all its destinations. When a message is stable, bandwidth and host memory are no longer used for it, and the protocol then can terminate.

In Appendix A the protocol is described by means of finite-state automata for each type of involved host: clusterhead, gateway or regular host.

[3] With "successfully exchanged" we mean that the sender (either p or ch) receives the ack.

4.1. The service

The reliable broadcast service can be accessed through the primitives `RBCAST.Rq`(msg, $dest\_ID$) and `DELIVER.Ind`(msg). The primitives are non-blocking and the upper user entities can proceed in parallel with the service provider. When the `RBCAST.Rq` is called, the calling entity remains blocked only the time needed to know that the message has been correctly received by its (local) clusterhead. Once the control is returned, the calling entity knows that the message will be delivered to all the hosts in $dest\_ID$. The value of the $dest\_ID$ field is potentially the identifier of a host, in the case of a unicast message, of a group of hosts in the case of a multicast message, or *all* the hosts in the case of a broadcast message. In the current implementation the case of broadcast is the only considered. The `DELIVER.Ind`(msg) primitive notifies the reception of a message to the upper layer entity. Upon receiving the Indication, the called destination entity knows that all the other members in $dest\_ID$ will receive the same message.

4.2. Outline of the protocol

The reliable broadcast protocol we are describing, is characterized by two phases: the *scattering* phase, in which a message is diffused from the source to all the destinations, and the *gathering* phase, in which the source collects all the acknowledgement messages, or acks for short, from the destinations. Acknowledgements represent a high cost to pay here and they are unavoidable. In fact, negative acknowledgement schemes could be in principle adopted, under the assumption of either a continuous message generation or some synchronization amongst stations (negative-ack are used in Totem [3], where a logical ring is used to synchronize the exchange of status information among group members). It is not adopted in practice because these assumptions are difficult to apply in presence of mobility and because it makes difficult to decide when a message is stable and, as the consequence, when a message can be deleted from internal queues. Another possible approach, that we call optimistic, eliminates ack messages by charging the receivers of the responsibility to periodically exchange status information with all the members of the group. If one receiver identifies that it is missing a message, it broadcasts the request and the message owner replies by broadcasting the copy. The use of broadcast is motivated by the need to reach more receivers with one broadcast. When the error rate is low and when some knowledge about the network topology is available, such as in wired networks, the optimistic approach can guarantee the reliable multicast/broadcast service and good performances. For instance, the Scalable Reliable Multicast (SRM) protocol [10] adopts a similar mechanism to provide packet recovery at the application layer. In the mobile environment we are considering, the failure rate is higher and having a number of receivers that cyclically broadcast update requests will quickly congest the network. Here, what matters is detecting the failure the faster as possible to initiate the packet recovery, whose impact should be local to the area where the failure occurred without the originator being involved, and to be capable of using the routing infrastructure before it vanishes, as the consequence of mobility. Acknowledgements satisfy these requirements; they are not costless, but we have observed that the message growth can be controlled. To this purpose, our protocol distributes the packet recovery capability among the clusterheads and initiates the recovery by exploiting a source based routing infrastructure, which is called *forwarding tree*, or FT, as described in the next sections.

In the following, the protocol is described in the following order: first, the behaviours of the forwarding tree are briefly introduced; secondly, its use during the scattering and gathering phases and the way a message is stabilized are described by considering the case of a single message diffusion and static hosts. Finally, we describe how the protocol copes with hosts and clusterheads mobility.

4.2.1. The Forwarding Tree

The forwarding tree is a source-centered routing tree which is deployed on demand during the scattering phase and does not require route maintenance despite mobility. Its life time is at most the time needed to complete the scatter and gather of a single message and is constructed by connecting adjacent clusterheads through common gateways. FT follows the fastest (not necessarily the shortest) path that, at the scattering time, connects a given source to its destinations.

The main purpose of FT during the scattering phase is to provide an efficient loop free delivery route. It is much more useful, during the gathering phase, to route the acknowledgements back to the message source by starting from leaf clusterheads. However, the FT can be broken as the consequence of clusterhead mobility and failures. In this event, the protocol does not undertake any recovery action addressed to maintain or to reconstruct the tree. FT simply gets lost and, when a clusterhead has no ways to route ack messages, the protocol resorts to flooding. In this case, however, if a unicast routing protocol were available, costly flooding packets could be replaced by unicast packets routed over the paths provided in accordance to the routing information.

The FT has a node in each network clusterhead; for this reason it is mainly suitable to broadcast or, at most, to multicast over a *dense* group, i.e. a group with at least a member in nearly each cluster. When *sparse* groups are considered, FT becomes inadequate and more specialized tree infrastructures should be adopted [7,4,5].

4.2.2. Scattering and gathering phases

In figure 2(a), the protocol is sketched with respect to the network topology shown in figure 1. We consider a host p_s in cluster C_A that originates a message m having destinations in clusters C_A, C_B, C_C, C_D. p_s originates m by sending a unicast message to its local clusterhead ch_A[4]. ch_A is called the *originator clusterhead* of m and is the root of the FT under construction. ch_A adds to m the clusterhead header, broadcasts the message within cluster C_A and waits for acknowledgements to get aware of the *local*, i.e. within the cluster, stabilization of the message. Upon receiving m, each destination host p_d in C_A replies with an acknowledgement. A gateway that receives m from ch_A forwards m to all the adjacent clusterheads to guarantee message propagation. When the gateway gw_{AB} forwards m to ch_B, it registers in m that m is coming from ch_A. When ch_B receives the first copy of m (other copies could be forwarded through other gateways), it chooses ch_A as its own predecessor and publishes its choice when it broadcasts m in the cluster. The publishing of these information avoids the loop establishment due to the multi-path scattering and helps in discarding duplicates. gw_{AB} sends its acknowledgement to ch_A only after receiving the broadcast of m from ch_B; this way, gw_{AB} can report that gw_{AB} and

[4] If p_s is a gateway, its local clusterhead is its primary clusterhead.

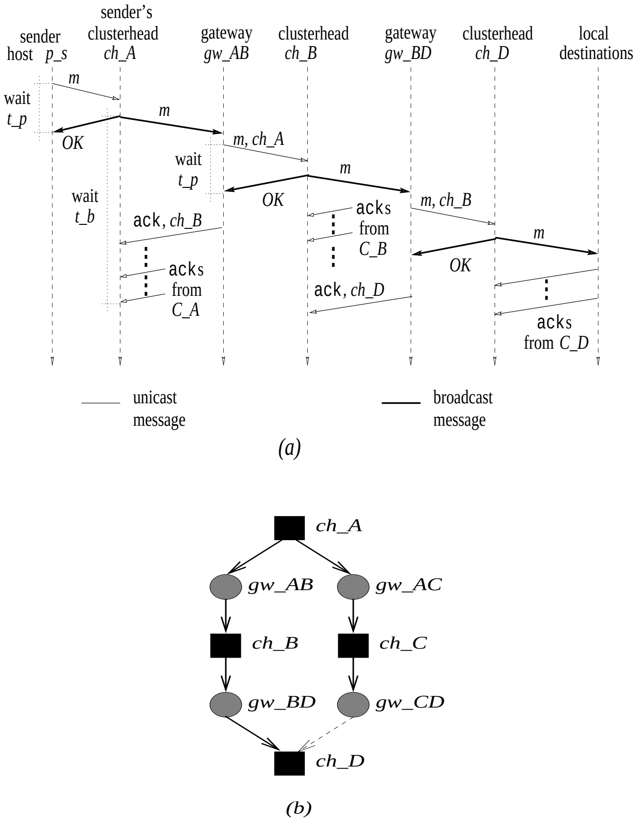

Figure 2. (a) The broadcast protocol in the case of static topology. (b) The corresponding FT.

ch_B have correctly received m and that one of the successors of ch_A on the FT for m is ch_B. The process is iterated, for instance, between ch_B and ch_D. Some clusterhead may experiment conflicting situations despite the fact that they are used to publish their choices. In figure 2(b), ch_D might receive, nearly at the same time, m from both gw_{BD} and gw_{CD}; in this case, it randomly selects one of the conflicting paths. If ch_D chooses ch_B, when the message is broadcast in cluster C_D both gw_{BD} and gw_{CD} reply respectively to ch_B and ch_C. As a result of this message exchange, ch_B knows to have the successor ch_D for message m and ch_C knows to be a leaf node in the FT for message m.

At the end of the m's scattering phase, each clusterhead knows the identities of the one-hop-predecessor (upstream) and of the one-hop-successor (downstream) clusterheads, and it knows whether it is a leaf of the FT or not. The up- and down-stream nodes (if any) on the FT are the only information each clusterhead maintain about FT. Gateways also maintain local information about the identifiers of the processed messages. The information are used to cut loops, to discard duplicates and to force the scattering of a message to proceed forward by locking the arrival branch. By recording the FT structure only at the clusterheads it is possible to make the protocol independent of the gateway mobility and to ease the loop avoidance.

From figure 2(a), it is clear that when ch_A diffuses m, p_s knows that ch_A received m. At that time, the calling entity in p_s can proceed in parallel with the service provider which performs the actions to diffuse and to stabilize the message over the entire population of hosts. Similarly, the diffusion of m in the other clusters informs the upstream gateways that they know m.

The gathering phase is initiated by the FT leaf nodes when they receive a message m. When a leaf clusterhead, e.g. ch_D, has collected all its local acks it packs them into a `cluster-ack` message and forwards it to its predecessor, i.e. clusterhead ch_B. Under favourable conditions (i.e. static topology and low error rate), we can have only one `cluster-ack` message per each branch of the FT; the message stops at an intermediate clusterhead, one hop upstream, loads local ack information and proceeds to the next stop until it reaches the FT root.

Transient link failures can cause the loss of the downstream `cluster-ack` message. To avoid a non-leaf clusterhead, e.g. ch_B, to be blocked indefinitely, a simple waiting policy is adopted; ch_B delays for the time t_{ack} the transmission of its `cluster-ack` message, waiting to receive the ch_D's `cluster-ack`. t_{ack} is greater or equal to two round trip delays over one hop. In the event of time out, ch_B proceeds autonomously to send its `cluster-ack`, thus creating a second `cluster-ack` in the same FT branch. In the event the downstream `cluster-ack` arrives too early, i.e. before the local acks have been collected, it could be useful to implement a *hold-back* queue where early messages can be delayed. In our implementation of the protocol, the *hold-back* queue is emptied by a deamon whose activation rate is t_{ack}, to avoid message starvation as the consequence of mobility.

4.2.3. Message stability

In this section we describe how the FT is used to guarantee the reliable message stabilization.

By following the protocol steps described in the previous section and always considering the FT in figure 2, when ch_A receives the ack from all the m's destinations, it considers m as stable. Message stability must be achieved despite link failures and host mobility.

The protocol adopts a packet retransmission policy to cope with transient link failures that cause packets (data and `cluster-ack`) loss; a timeout t_b is used to decide retransmissions, whose value is roughly two round trip delays over one hop. All the clusterheads of the FT have the responsibility to enforce packet recovery in the portion of the network that is under their control, i.e. the cluster and the up- and down-stream hosts of the FT. This approach has the advantage to distribute the recovery load, thus avoiding to start retransmissions at the originator clusterhead. The benefits in terms of bandwidth and latency time are paid with an extra cost in terms of memory occupation at each host, where a copy of the message must be maintained as long as it becomes aware of the message stability. To

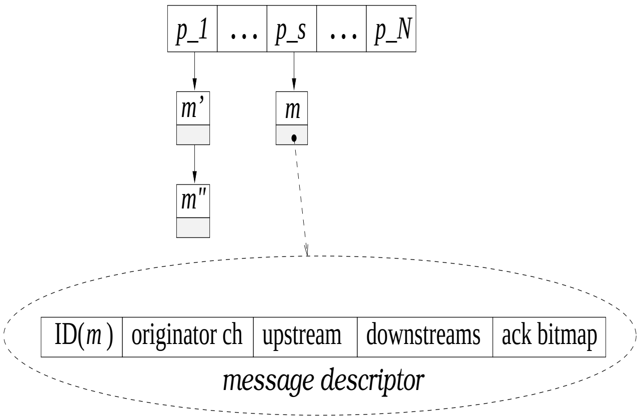

Figure 3. Layout of the $unstable_{ch}$ structure and of the message descriptor.

this purpose, each host maintains two data structures: *unstable*, (figure 3), where the copy of the received messages per source host are enqueued in the order given by their sequence number, and *stable*, (figure 4), which is an array of N items, whose value of the j-th entry is the maximum sequence number of messages that p_j generated such that all the p_j's messages having up to that sequence number are stable. In figure 4, the p_j's messages with sequence numbers in the range 0-5 are stable. Furthermore, each clusterhead maintains a message descriptor for each message in *unstable* (figure 3). To understand the use of these data structures, let us consider the host ch_B in the branch ch_A-ch_D of the FT in figure 2. ch_B updates $unstable_B$ and creates the message descriptor during the scattering phase. If failures occur, ch_B keeps up retransmitting m within the cluster until it receives all the ack messages from its neighbours (gateways are included, of course)[5]. If duplicates are generated as the consequence of retransmission, the *unstable* data structure allows to detect and discard them. Upon receiving the ack packets, the field `ack-bitmap` in the message descriptor is updated to record that a given host received m. The information contained in the `ack-bitmap` are used to complete the `cluster-ack` message according to the procedure given in the previous section. For this reason, the `cluster-ack` packet must be acked too, and retransmitted if not; in this latter case, the FT connectivity is likely to be broken (see, section 4.3).

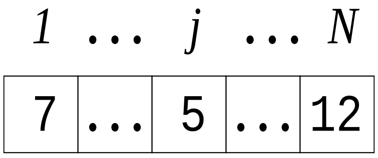

Figure 4. Layout of the $stable_p$ array.

At a given time, ch_A, the originator of m, will receive the ack information from all the receivers of m, i.e. its `ack-bitmap` will be complete. At that time, ch_A knows that m is stable and can remove m from its *unstable* and add the sequence number of m in *stable*. To get all the other hosts acquainted with the achieved stability of m, and to allow local data structure updates, the stability information should be reliably diffused. To this purpose, the *stable* data structure is the pay-load of the messages during the scattering phase; if a clusterhead has no messages to send and *stable* updates to propagate, it generates an empty message, i.e. without data field, and initiates a new scattering phase for it. As the consequence of the fact that messages are reliably diffused during the scattering phase, eventually all the clusterheads of the FT receive the message from ch_A containing the *stable* updates. By considering the case of ch_B, it merges $stable_A$ and $stable_B$, keeps the maximum and $unstable_B$ is updated accordingly. Hosts purge their *unstable* data structures according to the *stable* array they receive from their clusterhead.

[5] The local ack collection may be optimized by specializing the approach according to the underlying wireless MAC protocol.

4.3. Topology changes

The data structures introduced in the previous section are the key element to guarantee reliable message diffusion and distributed recovery load in presence of host mobility too. We start by considering the case in which the movements do not cause changes to the FT infrastructure.

4.3.1. Hosts mobility

Host mobility leads a host p to move from a cluster C_A to a cluster C_B. Two events are relevant to our discussion: (i) p is the sender of a message m that is still unstable while p moves, and (ii) p is destination of a message m' and ch_A did not receive the ack from p for m'. While in the former case p does not know if ch_A has received m, in the latter, ch_A is unaware of whether or not p received m' before moving. Both cases are solved by exchanging status information between p and ch_B. As soon as p enters the new cluster, p and ch_B exchange the respective *stable* and *unstable* (in practice, the message identifiers only) information and proceed by exchanging the missing message copies (if any) and by updating the ack information. If ch_B collects new ack information about messages it has already acknowledged, it can send a separate `cluster-ack` to the originator. As the consequence of the above status exchange, in the infrequent event in which p moves to ch_B faster than the message it sent in ch_A, two originator clusterheads for the same message may compete to diffuse the message and to collect acks. This conflicting situation generates duplicates of the message, that are normally acknowledged but discarded, and requires the identification of a single responsible for collecting the acks. According to the adopted policy, the originator clusterhead, whose cluster currently contains the sender host p, becomes the root of FT and the collector of the `cluster-ack` packets.

4.3.2. Clusterhead changes

Topology changes may cause the election of a new clusterhead. When a clusterhead ceases to cover its role, the information it maintains about the message stability are lost and must be reconstructed by the new elected clusterhead. Let us consider a cluster C_A and let us indicate with old_ch_A and new_ch_A respectively the former and the new clusterhead of C_A. Three problems may arise: (i) old_ch_A could have some message m, whose diffusion either has not yet been initiated or (*ii*) it has been only partially performed; or (*iii*) old_ch_A could know the stability of some message and it has not yet diffused that information. Point (*i*) is solved by forcing the sender to re-start the broadcast of m, because all the current clusterheads do not have m. In the other two cases, new_ch_A must reconstruct the status information held by old_ch_A.

When new_ch_A appears, all its neighbours send to it their status information because they have a new clusterhead (as described in section 4.3.1). This way, new_ch_A obtains the updated versions of the local data structures. Similarly, if old_ch_A is in a different cluster, it sends its own status to its current clusterhead, say ch_B; additionally, it informs ch_B that it has been a clusterhead.

Once new_ch_A knows the local status, it reliably broadcasts these information to all the other clusterheads. Each clusterhead replies with its own status, possibly merged with the one of old clusterheads that appeared in its cluster. Status information are composed of the *stable* array and of the identifiers of the unstable messages, together with the received acks. This way, the information concerning the stable messages known by old_ch_A are recovered. This procedure is very similar to the one described to send data messages and to collect acknowledgements. It is not blocking with respect to the generation of new messages, whose diffusion is started as soon as the new clusterhead is elected.

Let us consider a partially diffused message m. Two relevant cases can occur: either only new_ch_A has m, or more clusterheads have m. In the former case, new_ch_A re-starts the m diffusion as the new originator clusterhead. In the latter case, each clusterhead ch_B having m marks it as "suspended", and possibly stops sending `cluster-ack`s for m. If the m sender, say p, reconnects to a clusterhead not having m, then p restarts the m diffusion. If, on the contrary, p reconnects to a clusterhead ch_C having m as "suspended", ch_C re-starts the m diffusion as soon as it notices the p reconnection. We break the symmetry among possible multiple originator clusterheads of the same message as discussed in section 4.3.1.

Finally, a message m could exist that is held by all the clusterheads, but whose stability is not known. If m is locally stable in each cluster when the topology change occurs, then new_ch_A detects its stability. Otherwise, m must be re-diffused by some clusterhead before its stability being noticed.

4.3.3. Ack management

Hosts mobility can destroy the FT that ack messages should follow backward. Let us consider the case of ch_D that sends the `cluster-ack` message upstream to ch_B, via the gateway gw_{AB}. If ch_D does not receive an ack within t_b from ch_B, it is unable to know if the `cluster-ack` has been lost because of a link failure, or because the FT has been destroyed in that point. Provided that these two cases are indistinguishable, the protocol follows an alternative route to ensure that the `cluster-ack` is eventually received by the originator. If an underlying unicast routing protocol were available, the `cluster-ack` can be forwarded by addressing a (*multi-hop*) unicast packet to the originator (the information is maintained in the message descriptor). If routing information are not available, i.e. the protocol is layered over the wireless MAC protocol, there is no alternatives to flooding, and a `flood-ack` message is sent to the originator clusterhead with the same information as the `cluster-ack`. This is the only situation in which the reliable broadcast protocol may benefit of the underlying routing protocol, whose availability becomes important under high mobility conditions in which the clustering protocol cannot guarantee clusterhead stability.

Upon receiving a `flood-ack`, a clusterhead first tries to forward it via the FT. If it is still available, the flooding is interrupted, otherwise it is continued. This approach allows the protocol to adapt the transmission according to the situations encountered in different network zones. However, flooding generates duplicates of the ack information that must be dropped by using the information in *unstable*.

5. Protocol correctness

In this section we prove the protocol correctness and termination with respect to the problem specification. Proofs are based on the state-machines reported in Appendix A.

We initially notice that the hypothesis that hosts never crash implies that all their data structures have never to be recovered from failures. As the consequence:

Proposition 5.1. Neither the messages recorded in the message queues and in the *unstable* structures, nor the information recorded in the *stable* structures are ever lost.

This result underlies all the following proofs.

Lemma 5.2. If a host p sends a message m to a one-hop neighbour host q, then eventually q receives m.

Proof If the movements destroy the two hosts neighbourhood, then the hypothesis does not apply. Otherwise, the claim follows from the Liveness Property. □

This lemma holds in spite of transient failures.

To prove that the described protocol correctly solves the Reliable Broadcast problem, we firstly consider a static topology.

Lemma 5.3. In the case of static topology with transient failures, a message m diffused by a clusterhead ch_A (i) is eventually received by both all the hosts in C_A and all the clusterheads connected to ch_A via gateways; and (ii) is delivered by ch_A.

Proof The claim follows from the protocol description and the Liveness Property, by repeatedly applying Lemma 5.2 to the transmission of m from (i) ch_A to each one of the hosts in C_A, and (ii) each gateway in C_A to each other clusterhead it hears from. □

Lemma 5.4. Let p be a host that receives a message m; then p delivers m exactly once.

Proof The claim follows by the protocol and the hypothesis that hosts never crash: when p receives m for the first time, it delivers m and records m in its *unstable* queue. From now on m is recorded either in $unstable_p$ or in $stable_p$. Hence, p detects and discards possible duplicates of m and does not anymore deliver m. □

Theorem 5.5. The protocol solves the Reliable Broadcast Problem in the case of static topology without partitions.

Proof Let p be a host that generates a message m; let ch_A be the originator clusterhead of m. We prove the claim with respect to m. The Integrity property follows from Lemma 5.4.

Thanks to the Liveness property and the absence of partitions, by repeatedly applying Lemma 5.3 to all clusterheads we prove that all the hosts receive m, and they deliver m thanks to Lemma 5.4. Hence, the Validity and Agreement properties follow.

By the Liveness Property, each transmission of m over a logical link is successfully completed within a finite time. Hence, the diffusion of m to all the hosts in the system is completed within a finite time and the Termination property follows. □

We now prove that the problem is correctly solved also in the event of hosts mobility that does not affect the clusterhead infrastructure.

Lemma 5.6. Let p be a host that generates a message m. In the case of host mobility that does not affect the clusterhead infrastructure, eventually a clusterhead, let us say ch_A, starts the diffusion of m.

Proof Thanks to the Liveness Property, p eventually enters a cluster, let us say C_A, and it stays connected the time needed to allow that the exchange of status information is completed and a message is successfully exchanged between p and ch_A. By the status exchange, p gets aware of the fact that the current clusterhead has not m. Eventually, m is the message successfully sent from p to its current clusterhead, after the status exchange. According to the protocol and the hypothesis that clusterheads do not change, ch_A eventually starts the diffusion of m. □

Lemma 5.7. Let p be a host destination of a message m. In the case of host mobility that does not affect the clusterhead infrastructure and without partitions, eventually p delivers m.

Proof The absence of partitions and the Liveness Property guarantee, by repeatedly applying Lemma 5.2, that each clusterhead eventually receives m. By applying the same arguments of Lemma 5.6, we prove that p eventually enters a cluster, let us say C_A, and it stays connected the time needed to allow that the exchange of status information is completed, ch_A gets aware of the fact that p has not yet received m, and m is successfully sent from ch_A to p. From Lemma 5.4, p delivers m. □

Theorem 5.8. The protocol solves the Reliable Broadcast problem in the case of host movements that do not affect the clusterhead infrastructure and without partitions.

Proof Let p be a host that generates a message m. By Lemma 5.6, p eventually sends m to its current clusterhead, let us say ch_A, that starts the broadcast of m. By the same arguments of Theorem 5.5, we prove that all the clusterheads and the hosts that do not move eventually receive and deliver m exactly once. Let q be a host that moves while m is diffused. From Lemma 5.7, q eventually delivers m exactly once. Hence, all the hosts deliver m and the Integrity, Validity and Agreement properties are satisfied.
The Liveness Property holds within a finite time; as a consequence:

1. within a finite time p sends m to ch_A, that starts the broadcast of m (by Lemma 5.6);
2. within a finite time m is reliably broadcast to all the hosts that do not move (by Th.5.5);
3. within a finite time m is received by all the hosts that move (by Lemma 5.7).

Hence, within a finite time m is reliably broadcast to all destinations, thus satisfying the Termination property. □

To prove the main result, we now consider the effects of the changes in the clusterhead infrastructure. Let us recall that we assumed that network partitions are eventually repaired; as the consequence, eventually a path exists to communicate with a host.

Lemma 5.9. Let p be a host that generates a message m. In the case the p's clusterhead old_ch_A is replaced by new_ch_A before m becomes stable, m is eventually diffused to all its destinations.

Proof The following relevant cases can occur after the topology change:

1. no clusterhead has m; then, by the Liveness Property, p eventually re-starts the broadcast of m from its current clusterhead. Theorem 5.8 then applies;
2. only new_ch_A has m. By the protocol, new_ch_A eventually re-starts the m diffusion. Theorem 5.8 then applies;
3. either only an old clusterhead or more clusterheads know m. By the Liveness Property eventually p reconnects to a clusterhead, let us say ch_B, and it stays connected enough to allow that the exchange of status information are completed and a message is successfully exchanged between p and ch_B. If ch_B has not m, then the same arguments as in Lemma 5.6 apply. Otherwise, by the protocol ch_B re-starts the m broadcast. In both cases, once the broadcast of m has been re-started, the same arguments as in Theorem 5.8 apply.

□

Lemma 5.10. Let p be a host that generates a message m, with originator clusterhead ch_A. If one destination clusterhead old_ch_B is replaced by new_ch_B, m is eventually diffused to all its destinations.

Proof By the status information exchange, ch_A[6] gets aware of the fact that new_ch_B has not m. By the Liveness property, by repeatedly applying Lemma 5.2 to the transmission of m from an old clusterhead to new_ch_B, within a finite time m is successfully sent to new_ch_B, after the status exchange. By the protocol, eventually new_ch_B broadcasts m in its cluster. Theorem 5.8 then applies. □

Theorem 5.11. The protocol solves the Reliable Broadcast problem in the considered environment.

Proof We already proved the claim in the cases of static topology and of host movements that do not affect the clusterhead infrastructure. In the case of changes to the clusterhead infrastructure, the Liveness Property and Clusterhead Stability Property guarantee that the protocol eventually makes progress also when the clusterheads dynamically change. In the worst case, all the hosts become cyclically clusterhead. By repeatedly applying the Lemmata 5.7, 5.9 and 5.10, we prove that eventually each host receives m either when it is a clusterhead or when it is in a cluster whose clusterhead knows m. Hence, eventually, all the hosts receive m and deliver it by Lemma 5.4, thus satisfying the Validity and Agreement properties. The Integrity property is verified thanks to Lemma 5.4. The Termination property follows from the fact that both the Liveness Property

[6] And possibly some other old clusterheads.

and the Clusterhead Stability Property hold within a finite time. □

Theorem 5.12. The protocol satisfies the Eventual Subsidence property in the described environment.

Proof Let m be a message broadcast by a host p, with originator clusterhead ch_A. By Theorem 5.11, eventually all the hosts receive and deliver m. Thanks to the Liveness Property, each host eventually succeeds in sending an ack in reply to the m reception; hence, eventually all the clusterheads learn the local stability of m in their clusters. If the clusterhead infrastructure does not change, in the worst case all the clusterheads generate `flood-ack` messages addressed to ch_A. Thanks to the Liveness Property, by repeatedly applying Lemma 5.2 to the transmission of these acks, eventually ch_A learns the global stability of m. If the clusterhead infrastructure changes, thanks to the Clusterhead Stability Property eventually the status information exchange amongst clusterheads is completed and the new clusterheads learn the global stability of m. The same arguments apply to all the messages generated by p before m; hence, eventually the m identifier enters the *stable* of some clusterhead. By applying Theorem 5.11 to the broadcast of either the first message with which *stable* is diffused, or *stable* itself, we prove that eventually all the hosts know the stability of m and of all the messages that precede m. By the protocol, eventually all the hosts delete m from their *unstable* queues and do not send further messages about m. □

6. Protocol analysis

In this section, we give a complexity analysis of the Reliable Broadcast protocol. We identify the 'communication complexity' MSG, defined as the amount of messages that the protocol exchanges to broadcast one message from a given originator, the 'time complexity' TL, defined in terms of latency time, i.e. the elapsed time between the generation of a message and its stabilization at the originator node, and the 'memory complexity' UNS, defined as the length of the *unstable* data structure.

In our complexity analysis, we assume that one message requires a single network packet and that the internal processing time is zero. We indicate with t_r the time required to transmit one message within a cluster. Within a time $2t_r$ a clusterhead can receive the acknowledgements from its neighbours and within $4t_r$ it can become aware that the successor clusterheads received a message. Hence, $t_b \simeq 4t_r$; in the case of a WaveLAN wireless network with 2 Mbps of bandwidth the timer may roughly lasts 20 msec. If D is the distance in hops between the originator clusterhead and the farest leaf clusterhead in the FT, then TL, when the topology is stable, is: $TL = O((4+2D)t_r)$; in fact, the message traverses twice the network diameter and 4 hops are required to allow the communication between the source node and its clusterhead and between the last clusterhead and its neighbours. Further $D+1$ hops, each requiring a time $O(t_r)$, are needed because the message stability is known by the nodes different from the originator. In addition, the message that reports the stability information to the originator node must wait $1/2f_m$ in average, where f_m is the hosts' message generation rate, before transmission, thus leading to a time $TL = O((3D+5)t_r + 1/2f_m)$ required to stabilize a message all over the system. By assuming n the group cardinality, we can obtain: $MSG = O(n)$, and $UNS = O(f_m \cdot TL)$. For instance, let us suppose that the wireless network is a WaveLAN having a bandwidth of 2 Mbps. If $n = 20$, $D = 4$, $f_m = 50$ msgs/sec. and the maximum size of *data* is 1024 bytes, then the approximate value of TL is around 50 msec., while the amount of generated messages is 40.

For what concerns the size of the message descriptor, it mainly depends on the number of the downstreams, i.e., of the adjacent clusters. We estimate that, since two clusterheads cannot ever be connected, clusters are not too dense and the number of downstreams is thus much lower than n. The `ack-bitmap` size is at most n, while 4 bytes are enough to store the message identifier and the identifiers of the originator and the upstream clusterheads.

This preliminary result indicates that as long as the topology changes do not affect clusterhead and gateways, the complexity of the algorithm is $O(n)$. As the consequence of flooding, it can degenerate to $O(n^2)$ otherwise. In particular, the movement of a single host p between two clusters, so that it does not affect the clusterhead infrastructure, imposes an additional delay of $O((D+4)t_r) + t_{move}$ before a message becomes stable in the system, where t_{move} is the elapsed time between the generation of m by p and the connection of p to the new clusterhead. The computational complexity of the protocol increases in this case of at most $O(D+4)$ unicast messages.

The election of a new clusterhead imposes an additional delay of $O((5D+7)t_r + 1/f_m) + t_{elect}$ to the time required to stabilize a message, with t_{elect} the time spent for the execution of the clustering protocol. This upper bound also includes the cost of re-starting the broadcast of an unstable message. The communication complexity also increases of $O(n+2D+2)$.

We are evaluating the performance of the reliable broadcast protocol through simulations [16]. Preliminary results show that, with a system of 50 hosts that move along random trajectories over a square area of 150×150 mt. and cluster radius of 60 mt., a message is reliably diffused to all the hosts in 0.675 sec. on average, while consistency about its global stabilization is reached after 1.48 sec. on average. We are using a TDMA MAC protocol; hence, nodes compete for the bandwidth and collisions within a cluster may occur. Collisions negatively affect the gathering phase and the time needed to know the message stabilization.

7. Porting to single-hop networks

The reliable broadcast protocol has been designed with the aim of reducing as much as possible the amount of information that are maintained at each mobile host to describe the system global state. All the nodes, except the clusterheads, maintain only local information. This approach is making the porting of the protocol to a single-hop network environment [1,2] feasible. In this case, in fact, the clusterhead functions can be mapped onto the base station and each cluster becomes an abstractions of the cells. No gateways are required because the cells interconnection is provided by the wired network. FT is reduced to a static infrastructure that supports the routing among MSSs over wired links.

If we assume that base stations never fail and that the processing time is negligible with respect to the communication time, the complexity evaluation of the indexes introduced in section 6 can be obtained. In Table 1 RBCAST is compared with the protocol described in [1], which also can provide a reliable broadcast service.

We indicate with B and b the broadcast messages that flow respectively through the wireless and the wired network, and with U (u) the unicast messages. TL is computed as the sum of the number of hops in the wired network (h) and the number of hops in the wireless network (H). Let m be the number of MSSs and n the number of hosts in the system. In the case of static topology RBCAST requires:

Table 1
Complexity comparison.

	STATIC TOPOLOGY	
	MSG	TL
RBCAST	$O(b + mB + (n+1)U + mu)$	$O(3H + 2h)$
[1]	$O(b + mB + (n+1)U + mu + b)$	$O(3H + 3h)$
	LINK FAILURES	
	MSG	TL
RBCAST	$O(b + m\rho B + (n+1)\rho U + mu)$	$O(3\rho H + 2h)$
[1]	$O(b + m\rho B + (n+1)\rho U + mu + b)$	$O(3\rho H + 3h)$
	HOST MOBILITY	
	MSG	TL
RBCAST	$< \infty$ /* $O(2U)$ $\forall$ move */	$< \infty$ /* $O(2H)$ */
[1]	$< \infty$ /* $O(2U + 2u)$ $\forall$ move */	$< \infty$ /* $O(2H + 2h)$ */

- 1 broadcast message from the originator MSS to the other MSSs;
- 1 broadcast message in each cluster to the local mobile hosts;
- n unicast messages for the acks of the MHs to their respective clusterheads;
- 1 unicast message from each MSS to the originator MSS, containing the locally gathered acks.

The protocol described in [1] requires one more broadcast message, that is generated by the originator MSS to inform the other MSSs that a message is stable and can be deleted from the buffers. From MSG, TL can be easily derived.

In the case wireless links are affected by failures, no upper bound could exist on both MSG and TL. Results in Table 1 are obtained by considering the expected number of retries ρ.

Finally, in the case of hosts mobility both protocols only guarantee the eventual termination, by exploiting the *liveness* property (named *mobility assumption* in [1]). Whenever a RBCAST mobile host moves to a new cell it exchanges two status messages with the new MSS. The protocol described in [1] does not require message exchange while a host enters a cell, but, as a counterpart, its *handoff* procedure requires the exchange of two messages between the new and the old MSS.

Let f_m be the hosts' message generation rate; then, the message that reports the stability information generated by a originator clusterhead must wait $1/2f_m$ in average, before transmission. Then, the elapsed time between the message generation and its stabilization is $TL + 1/2f_m + H + h$. From the previous performance indexes, the *memory complexity* UNS can be also derived, which is defined as the length of the *unstable* data structure: $UNS = O(f_m \cdot TL)$.

8. Concluding remarks and future work

Advances in wireless communications are making fully mobile networks feasible. This is fuelling the demand of applications that can benefit of the fast deployment and reconfiguration of these networks and that highlight the need for a set of fault tolerant group-oriented services. This paper presents a multicast architecture devised for mobile ad-hoc networks and shows that the reliable broadcast service can be provided despite failures and mobility. We believe that this type of service is the fundamental block that eases the design of a class of fault tolerant distributed applications and enables the design of higher value-added group services such as total ordering and agreement.

While designing this protocol, one of our first apprehensions was about the possibility of obtaining a terminating protocol without resorting to system or failure constraints that could affect the use of the protocol in practice. For this reason, the development of the protocol in a simulation environment followed its correctness and termination proofs.

We are currently running simulations. The first results, of which the paper includes some preliminary report, although quite encouraging when clusterheads move slowly, have shown that packet dropping at clusterhead nodes is the primary cause of message loss; retransmission of lost messages increases the traffic load, increases the network latency and can lead to congest the system. Simulations provided the numerical evidence of the fact that a flow control policy has to be devised. This is mainly due to the fact that the primitive is non-blocking and the user entities are free to proceed to generate messages, while the stabilization of a message can take a long time because of mobility and failures. The uncertainties due to the topology changes are forcing the design of a flow control policy with a totally distributed control. We are currently designing a clusterhead centered control mechanism; a clusterhead can decide in autonomy its lock-unlock policy without information exchange between adjacent clusterheads intended to maintain coordinated actions. The resulting protocol is practically costless and totally avoids the event of orphan locks due to host mobility. However, it still has a few drawbacks that deserve further analysis and tuning. First, the mechanism provides a slow reaction to changing traffic conditions, so that it is not suitable to adapt to traffic bursts. Second, it is difficult to identify the signs of congestion by only observing local data information. Third, it could show some fairness problems because the neighbours of a slow clusterhead are likely to be often locked. While the first two limitations can be overwhelmed, the third is probably unavoidable under disomogeneous load conditions.

We are also working in different directions. We are designing a link state routing protocol that uses a modified version of the reliable broadcast in alternative to flooding. A multicast routing protocol has been designed. It is based on a flexible shared routing infrastructure that, similarly to the described protocol, is constructed on demand and does not require any route maintenance policy.

References

[1] A. Acharya and B. R. Badrinath, A Framework for Delivering Multicast Messages in Networks with Mobile Hosts, ACM/Baltzer Journal of Wireless Networks, Special Issue on Routing in Mobile Communication Networks 1 (1996), pp. 199–219.

[2] A. Acharya and B. R. Badrinath, Delivering Multicast Messages in Networks with Mobile Hosts, in: *Proc. 13th Intl. Conf. on Distributed Computing Systems*, 1993, pp. 292.

[3] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal and P. Ciarfella, The Totem Single-Ring Ordering and Membership Protocol, ACM Trans. on Computer Systems 13 (1995), pp. 311–342.

[4] C.-C. Chiang and M. Gerla, Routing and Multicast in Multihop, Mobile Wireless Networks, in: *Proc. ICUPC'97*, 1997.

[5] C.-C. Chiang, M. Gerla and L. Zhang, Shared Tree Wireless Network Multicast, in: *Proc. 6th IEEE Intl. Conf. on Computer Communications and Networks*, 1997.

[6] K. Cho and K. P. Birman, A Group Communication Approach for Mobile Computing, in: *Proc. Mobile Computing Workshop*,Santa Cruz, CA, 1994.

[7] M. S. Corson and A. Ephremides, A Distributed Routing Algorithm for Mobile Wireless Networks, ACM/Baltzer Journal of Wireless Networks 1 (1995), pp. 61–81.

[8] A. Ephremides, J. E. Wieselthier and D. J. Baker, A Design Concept for Reliable Mobile Radio Networks with Frequency Hopping Signaling, Proceedings of the IEEE 75 (1987), pp. 56–73.

[9] M. Fischer, N. Lynch and M. S. Paterson, Impossibility of Distributed Consensus with One Faulty Process, Journal of the ACM 32 (1985), pp. 374–382.

[10] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu and L. Zhang, A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, in: *Proc. SIGCOMM'95*, ACM Computer Communication Review 25 (1995), pp. 342–356.

[11] M. Gerla and J. Tzu-Chieh Tsai, Multicluster, Mobile, Multimedia Radio Network, ACM/Baltzer Journal of Wireless Networks 1 (1995), pp. 255–265.

[12] V. Hadzilacos and S. Toueg, Fault-Tolerant Broadcasts and Related Problems, in: *Distributed Systems*, ed. S. Mullender, Addison-Wesley, 1993, pp. 97–145.

[13] T. G. Harrison, C. L. Williamson, W. L. Mackrell and R. B. Bunt, Mobile Multicast (MoM) Protocol: Multicast Support for Mobile Hosts, in: *Proc. MobiCom'97*, 1997, pp. 151–160.

[14] T. Imielinski and B. R. Badrinath, Mobile Wireless Computing: Challenges in Data Management, Communications of the ACM 37 (1994), pp. 18.

[15] M. Parsa and J. J. Garcia-Luna-Aceves, A Protocol for Scalable Loop-Free Multicast Routing, IEEE Journal on Selected Areas in Communications 15 (1997), pp. 316–331.

[16] J. Short, R. Bagrodia and L. Kleinrock, Mobile Wireless Network System Simulation, in: *Proc. ACM Mobile Communication Networking Conference*, 1995.

Appendix

A. Protocol state-machines

In the following tables we describe the protocol by means of the finite-state machines for a clusterhead, a gateway and a regular host. In figures 5, 6 and 7 the three tables are introduced by means of simplified automata. The state-machines represent the actions executed by the hosts for a single message m; the concurrent management of other messages is not considered, although supported by the protocol. This has two consequences. Firstly, the procedure for the management of `cluster-ack` messages is activated as soon as a message is received. In general, it is autonomously executed, independently of the message arrival. Secondly, the information about the global stabilization of m are supposed to be always distributed with a separate broadcast of the *stable* array, while in general it is piggybacked on a successive message.

Similarly, the gateway state-machine has been simplified by hypothesizing that a gateway only connects two clusterheads, ch and ch'. In the gateway state-machine, we describe the forwarding of either broadcast or `cluster-ack` messages between adjacent clusters.

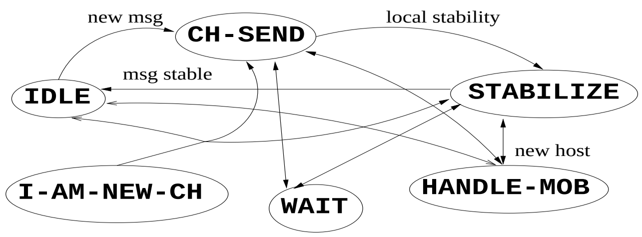

Figure 5. Clusterheads' automaton.

In figure 5, a clusterhead switches from the IDLE state to the CH-SEND state when either it receives a message m to diffuse from one of its local hosts or it generates its own broadcast message. In the CH-SEND state, a clusterhead re-transmits m until it has gathered all the local acks for m. When the clusterhead knows the local stabilization of m, it switches to the STABILIZE state, waiting to know the global message stabilization. In the STABILIZE state, the clusterhead sends its `cluster-ack` or `flood-ack` messages for m. When it obtains a *stable* structure reporting the global stabilization of m, it returns to the IDLE state. The HANDLE-MOB state describes the actions taken to manage the presence of a new local host in the cluster, namely the status exchange. A regular host that is elected clusterhead switches from the regular host's automaton, (figure 7), to the I-AM-NEW-CH state of the clusterhead's automaton. In the I-AM-NEW-CH state, the new clusterhead collects the status information from all its local hosts and from the other clusterheads. A clusterhead ch switches to the WAIT state when the originator clusterhead of m disappears, and ch does not know to whom `cluster-ack` and `flood-ack` messages for m should be addressed. In this state, ch marks m as suspended and

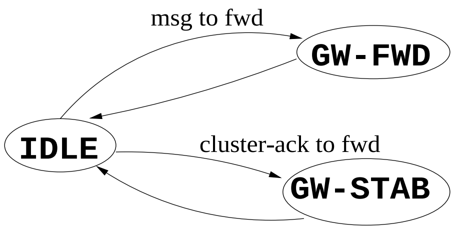

Figure 6. Gateways' automaton.

waits to know who is the host taking over as the new originator clusterhead of m.

In figure 6, a gateway gw switches from the IDLE state to the GW-FWD state when it receives a broadcast message m to forward to all the other clusterheads it hears from. In this state, gw waits to receive the broadcast of m from each other adjacent clusterhead. After that, gw sends its ack to the clusterhead from which it firstly received m and returns to the IDLE state. Similarly, the state transition from IDLE to GW-STAB occurs when gw has to forward a `cluster-ack`; gw returns to the IDLE state after having forwarded the ack relative to the `cluster-ack`. At the same time, a gateway host can also execute the regular host's automaton.

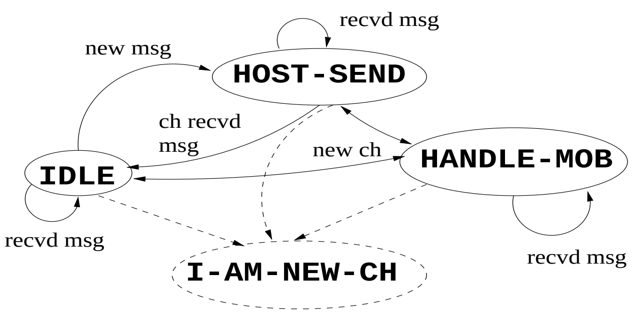

Figure 7. Regular hosts' automaton.

In figure 7, a regular host p switches from the IDLE state to the HOST-SEND state when it has a message m to send to its local clusterhead for diffusion. In this state, p retransmits m to its clusterhead until it receives the broadcast of m from the clusterhead; then, p returns to the IDLE state. A regular host switches to the HANDLE-MOB state when it has a new clusterhead; in this state, the interaction with the new clusterhead is performed to exchange the status information.