Reliable Broadcast in Mobile Multihop Packet Networks

Elena Pagani Gian Paolo Rossi
Dipartimento di Informatica
Università degli Studi di Milano
via Comelico, 39. I-20135 Milano, Italy
{pagae,rossi}@dsi.unimi.it

Abstract

We describe a reliable broadcast protocol for mobile multi-hop packet radio networks. The reliable broadcast service ensures that *all* the hosts in a group *deliver* the same set of messages to the upper layer. It represents the building block to construct fault tolerant distributed applications, whose demand is growing in a mobile environment too. The protocol is intended for use in networks where the host mobility arbitrarily changes the network connectivity and where the rate of change is not so fast as to impose flooding as the only option. In this context, our protocol is a first attempt to show that a mechanism more flexible than spanning tree and more efficient than flooding is possible and a service such as the reliable broadcast can be obtained.

The protocol is constructed on top of a multi-hop, multi-cluster architecture. It provides an exactly once message delivery semantics and tolerates communication failures and host mobility. Temporaneous disconnections and network partitions are also tolerated under the assumption that they are eventually repaired.

Keywords: mobile hosts, multi-hop radio networks, clustered architecture, reliable broadcast, fault-tolerance.

1 Introduction

We consider the reliable broadcast problem in multihop mobile packet radio networks. The reliable broadcast service ensures that *all* the hosts in a group *deliver* the same set of messages to the upper layer. This service is the building block to construct value-added multicast services, such as agreement and total ordering, or it can be utilized to support the applications that involve groups of cooperating hosts, use replication to achieve fault tolerance or share common resources. Most of these applications include the hosts mobility as a primary requirement. As a consequence, a few proposals have been recently presented that address this problem in the frame of network configurations with fixed base stations connected through a wired infrastructure [1, 2, 8, 12]. The resulting protocols benefit of the fixed part of the network.

In this paper, we describe a reliable broadcast protocol that is intended for use in networks where the host mobility arbitrarily changes the network connectivity and where an infrastructure of base stations is not present. This is the case of different applications, such as vehicle traffic and fleets management or decision support in emergency relief and battlefield operations; the groups are generally of small size and hosts are requested to interact according to a many-to-many scheme. Without a global knowledge of the network topology, the design of the broadcast and of fault tolerance is affected by the rate of topological change. In the presence of high mobility, the topology changes too fast to allow the construction of some logical routing structure and there is no alternative to flooding. In the presence of very slow mobility, the well known spanning tree algorithms, such as those described in [4], can be utilized. We consider mobile systems where the rate of topology change is difficult to predict and can range in between the two extreme bounds. In this context, our protocol is a first attempt to show that a mechanism more flexible than spanning tree and more efficient than flooding is possible and a service such as the reliable broadcast can be obtained. When the rate of change becomes too high, the protocol switches to flooding, but, since we expect that the mobility rate is not uniformily spread over all the hosts, the protocol reacts by adapting the transmission according to the situations encountered in the different network zones.

To obtain these results, the protocol assumes the existence of an underlying multi-cluster multi-hop packet network. Following the clustering approach, the hosts are grouped into clusters and a clustering algorithm, such as described in [10, 11], is supposed to identify the set of interconnected clusters that covers the whole

*This work has been in part supported by the Italian Government under contract MURST, 1996.

population of hosts.

Unlike other network hosts that are stateless, clusterheads maintain information necessary to ensure the message stabilization, to provide recovery when failures occur and to maintain a local view of the clusterhead infrastructure. This indicates that clusterheads are likely to become a potential hot spot of packet flow and a cluster bottleneck. To avoid this event, we are designing a simple and effective flow control mechanism to locally control the message traffic to prevent congestion and excessive packet dropping.

The protocol provides an exactly once message delivery semantics and tolerates host mobility and communication failures; it also tolerates temporaneous disconnections and network partitions under the assumption that they are eventually repaired and messages reach their group destinations. When the clusterhead functions can be located at a well known site, such as in the case of fixed base stations, the protocol can be easily adapted to the new environment.

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 outline the assumptions about the underlying routing algorithm. In section 4, we present the protocol under different mobility conditions: static hosts, mobile hosts, mobile clusterheads. In section 5 we sketch the correctness proof of the protocol; section 6 gives some preliminary numerical analysis.

2 The System Model

The system we consider is composed of n mobile hosts, 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 nodes and L the set of logical links. We indicate as one-hop *neighbours* two hosts that are connected by a logical link. If two hosts are *neighbours*, then they can receive from one another. A *link level protocol* is responsible of providing the following properties:

• at each time each host knows the identities of its neighbours;

• a channel access protocol is executed to solve contentions over logical links;

• if a message is sent over a correct link, then it is correctly received by the neighbour at the other end of that link.

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. To reach the destinations k hops away, we assume that a *clustering* protocol, exploiting the underlying neighbourhood information, is executed to construct 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 [10, 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 from the other members of the cluster. Two clusterheads cannot be neighbours, while

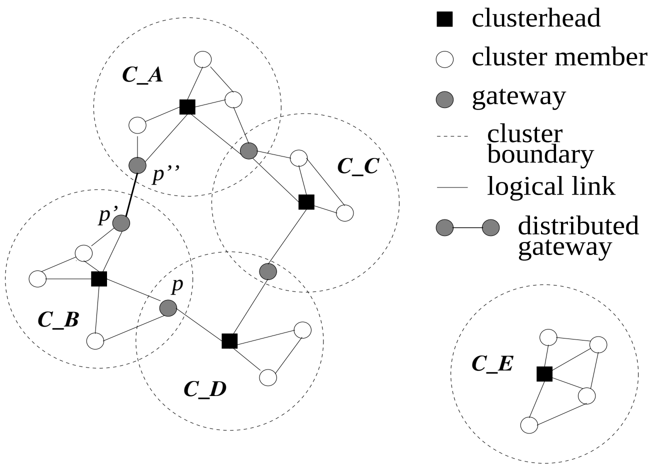

Figure 1: Example of a clustered multi-hop network.

they 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.

We consider a system where processors do not suffer from permanent failures; all types of failures are temporaneous and can be, sooner or later, repaired. This failure model is suitable to describe mobility and communication failures over wireless links. Host disconnections and network partitions, such as the case of cluster C_E in figure 1, are also tolerated under the assumption that they are sooner or later repaired. These assumptions underlie to a precise design choice addressed to reduce complexity at this level of the protocol architecture by avoiding the use of a membership protocol. Through the membership protocol, a set of hosts obtains a common agreed view of the group membership; according to our approach, the management of the membership is left, when it is required, to the higher layers where it can be performed according to the application requirements. In any case, the reliable broadcast we are describing is intended for systems of small groups of hosts (let's say, in the range [10,100] at most), in which the membership is performed elsewhere. In the event of dedicated systems, such as, for instance, the ones for emergency relief coordination, the membership is likely to cover the entire population of the stations in the system and can be defined at configuration time; otherwise, each application defines and maintains the membership over the relevant subset of hosts. In line with that, we will refer either to the more general reliable broadcast or to the reliable multicast problem. In this paper, we focus on providing the reliable broadcast service; the reliable multicast can be directly derived from the former, although it could benefit of the availability of multicast addressing facilities at the underlying levels [7, 9]. The tree infrastructures proposed in [7, 9] are promising solutions to support reliable multicast, although they seem to be more suitable when a few-to-many interaction scheme applies and when the rate of topology change is low.

3 Definitions and Assumptions

The system described in the previous section does not guarantee the message delivery to all destinations when communication failures or topology changes occur. This service can be provided by the reliable broadcast protocol. Reliable broadcast is a well known problem that has been widely studied in the literature because it represents the basic building block to construct other multicast services such as the atomic broadcast with total or causal ordering. Most reliable broadcast available solutions have been designed for wired networks [5, 3, 6]. In this realm, the problem can be informally specified as follows: given a group of n processors, if a *correct* processor broadcasts a message m then *all* the *correct* processors *deliver* m. A correct processor is a processor that does not suffer from crash or communication failures and the delivery operation describes the action of notifying a message to the higher layer. In the presence of mobility, even a correct processor can be unable to receive a message when disconnected or moving across cells. The adopted failure model allows to overwhelm this problem and leads to a slightly different problem definition (in the sequel, we will use the term *processor* as a synonymous of (mobile) host):

Definition 3.1 (Reliable Broadcast) *Given a group G of n processors in which each processor at any time can generate a broadcast message, messages are delivered by the processors so that the following properties are guaranteed:*

Validity and Agreement: *if a processor broadcasts a message m then all the processors in G eventually deliver m.*

Integrity: *for each message m, every processor delivers m at most once, and only if m has been broadcast by some processor.*

According to the given definition, the message delivery is guaranteed to *all* the members of G, and the reliability requirement is extended to *all* the generated messages. It is worth to notice that the Validity and Agreement properties guarantee an *at-least-once* delivery service semantics. This, together with the *at-most-once* delivery required by the Integrity property, yields an *exactly-once* delivery service.

We will show that, in the presence of mobility and unpredictable delays, the protocol eventually terminates. Similarly to [1], to guarantee the protocol termination, we require that the following *liveness* property is satisfied:

Property 3.1 (Liveness) *Eventually the network topology stabilizes for the time required to guarantee that:*

1. *if there are pending messages for a host p_i, then p_i receives at least one of these messages, let us say m, and it succeeds to notify the reception before the topology changes again;*

2. *a host remains clusterhead the time necessary to guarantee that the exchange of status information with the other clusterheads is successfully completed and, if there are partially diffused messages, that at least one of them is known amongst the clusterheads.*

If point (1) of this property is not guaranteed, then a processor destination of a message m could either (i) never receive m, or (ii) eventually receive m, but could never succeed in acknowledging the m's reception. In both cases, the eventual termination cannot be achieved. Similar arguments apply for point (2).

The protocol can be realized on top of the existing clustered architectures, such as those described in [10, 11]. We only assume that the rate of topology change is not so fast as to make uneffective the use of a clustered architecture or to arbitrarily delay the message stabilization. Moreover, we assume that the underlying clustering mechanism has designed to keep alive its infrastructure when the topology is slowly changing and a few nodes are moving.

4 The Protocol for Reliable Broadcast

4.1 The Service

The reliable broadcast service is visible at the interface with the user through the following non-blocking service primitives:

rbcast $\langle msg,\ group\_ID \rangle$;
deliver $\langle msg \rangle$.

The *rbcast* primitive provides a reliable transport of messages with *exactly-once* service semantics. The calling user entity remains blocked only the time needed to know that the message has been correctly received by its clusterhead. When the control is returned, the calling entity knows that the message will be eventually delivered to all the processors in *group_ID*. In the case of broadcast, the parameter *group_ID* corresponds to the clause *all*. The *deliver* primitive is the notification of *msg* reception. Upon receiving a delivery primitive, the user entity knows that all the other members in *group_ID* eventually receive the same message.

4.2 The Protocol for Static Topology

We initially describe the protocol operations in the absence of failures and mobility.

We have chosen to locate at the clusterhead stations all the information about the system state and message stability. Gateways and user mobile hosts are stateless so that the management of their mobility can be performed at a low cost. Provided that a message is uniquely identified by the pair $\langle sender\_ID, sequence\ number \rangle$, when a host in a given cluster originates a message, it is diffused to all destinations in different clusters by forwarding it through a logical tree, whose nodes are clusterheads and gateways. We indicate that tree with the term *forwarding tree* (FT). The problem here is the identification of the FT without introducing extra communication costs and to control the traffic by eliminating loops and other sources of duplicates that derive from infrastructures with multi-paths.

If the topology is stable enough, acknowledgements follow backward the logical tree and eventually arrive at the originator clusterhead. If the rate of topology change grows, there is no alternative to flood acknowledgements from each destination clusterhead to the originator clusterhead. In fact, the protocol cannot benefit of any wired backbone infrastructure, the FT is unstable and a global knowledge about it is not available. Although acknowledgements can be piggybacked onto the flowing messages or packed into multi-ack messages, they represent the hardest problem to cope with and a source of potentially unaffordable traffic load. Unfortunately, acknowledgements are an unavoidable cost to pay here. In fact, the service requirement specifies that a message is *stable* when the originator knows that all the destinations have received it. In the considered environment, negative acknowledgement scheme could be in principle adopted, under the assumption of a many-to-many interaction scheme with continuous message generation. It is not adopted in practice because it makes difficult to decide when a message is stable and, as the consequence, when a message can be deleted from internal queues.

Our protocol has a very small overhead to construct the FT and tends to preserve it, until the rate of change grows and breakes the fragile connectivity of that infrastructure; when this occurs, it switches to flooding. The management of acknowledgements is the most expensive module of the protocol and deserves further efforts in the direction of costs reduction. To these ends, we believe that significant benefits can be obtained by specializing the underlying dynamic routing algorithm in order to provide more information about the changing topology [7, 9].

Let us now consider the simple network topology composed of four clusters C_A, C_B, C_C and C_D shown in figure 2, and let us use it to describe the protocol (the case of all destinations within a single cluster is trivial). We consider a processor 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 clusterhead ch_A (remind that each host in a cluster is aware of the identity of its clusterhead). ch_A adds to m the clusterhead header, broadcasts the message within cluster C_A and waits for acknowledgements. The header specifies that m requires global diffusion because it has recipients outside cluster C_A. Upon receiving m, each destination processor p_d in C_A replies with an ack message; a gateway processor, e.g. gw_{AB}, is responsible of forwarding m to the next cluster(s) by addressing the relevant clusterhead, i.e. ch_B. gw_{AB} delays its ack message to ch_A until m has been diffused within cluster C_B (figure 2). The motivation is twofold: 1) this way ch_A knows that ch_B received m; 2) ch_A knows that it has a successor in the FT of m.

Adjacent clusterheads use gateways as agents to construct a local knowledge of the path from source to destination. As long as clusterheads are operational, this knowledge can be effectively used as follows. When ch_B broadcasts m within cluster C_B specifies that m has been arriving from ch_A; the information is sent backward to ch_A, through gateway gw_{AB}, by means of its ack message. The process is iterated, for instance, between ch_B and ch_D. ch_D can experiment a conflicting situation of the same message m coming through gw_{BD} and gw_{CD}. Under this condition, the clusterhead selects one amongst the conflicting paths, records the identifier in local data structures and piggybacks the selected identifier on the broadcast message. If ch_D

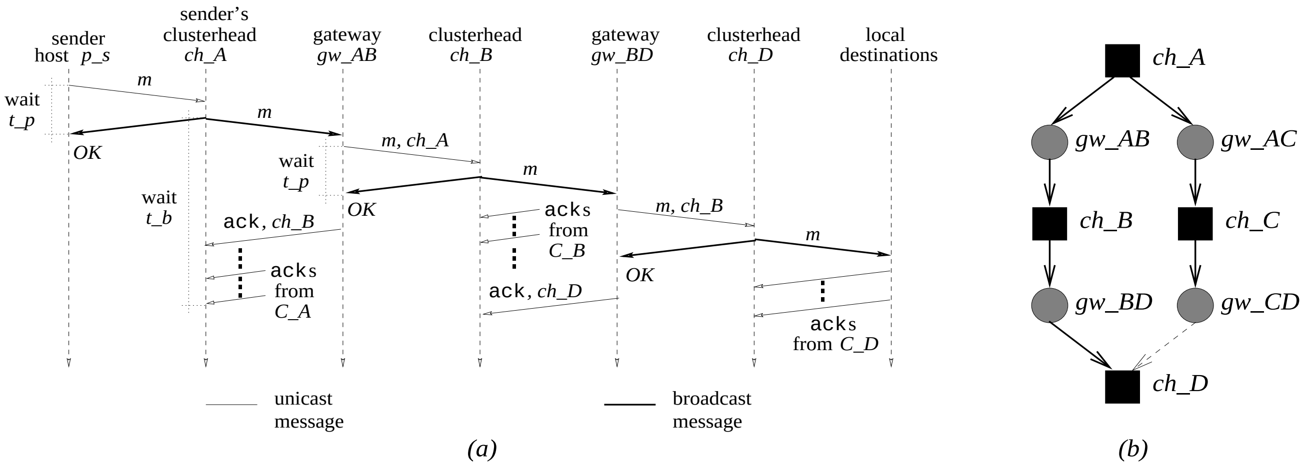

Figure 2: (a) The broadcast protocol in the case of static topology. (b) The corresponding FT.

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 for message m. Once all the acknowledgements messages in cluster C_D are received by ch_D, ch_D knows to be a leaf for message m. It is important that ch_D records its choice locally because as the consequence of message retransmission within some cluster (see section 4.3 that describes failures and mobility) and of gateway mobility, a copy of m could appear through a new route requiring consistent information about m. Under these conditions, even gateways could be involved to filter messages that have been already processed. Through this simple mechanism, the protocol cuts the loops in the message delivery, allows multi-paths to destinations, reduces message duplicates and provides the basic support to the management of acknowledgements, as described in the next section.

A sending processor, p_s, delivers the message when it receives the broadcast of the message from its clusterhead. A receiving processor p_d delivers the message upon receiving the broadcast.

4.2.1 Data Structures and Acknowledgements

Each processor p_i maintains in the data structures $stable_i$ and $unstable_i$ its local view about the message history. In $unstable_i$ messages are ordered by originator and sequence number; the ordered list contains i) the messages that p_i considers unstable and ii) the messages that are stable for p_i, but that are preceded by still unstable messages. $stable_i$ is an array of n items, with n the group cardinality. The jth entry contains the maximum sequence number of stable messages that p_j originated, such that all the messages generated by p_j up to that sequence number are stable. A message m is marked as *stable* by its originator clusterhead, let us say ch_A, when ch_A has received all the acknowledgements for m. A stable message m is deleted from the $unstable_i$ of a processor p_i only after its identifier is included in $stable_i$.

The *unstable* data structures have primary importance in providing recovery facilities in case of failures, while stability information must be recorded to eliminate duplicates. While $unstable_i$ represents the local knowledge of processor p_i about message delivery, the stability of a message is initially known by the originating clusterhead and must be quickly propagated over the entire population of processors, i.e. all processors must eventually have a consistent view of stable messages. To this purpose, a sort of full information mechanism is adopted so that the current $stable_i$ is added to the header of the exchanged messages. Upon receiving a message, a processor p_i compares the stability information it transports with local $stable_i$ and maintains the maximum.

In order to manage acknowledgements, clusterheads maintain in $unstable_i$ for each message the identity of the *predecessor* and the *successor* nodes in the message FT. In addition, they maintain two extra data structures: i) the bit map, as large as the group cardinality, to register the nodes that have validated the message and whose identity is known by the clusterhead, ii) a *hold-back* queue, in which to delay the final delivery of an acknowledgement message until some other events are verified. When messages are ready to be transmitted they enter a *delivery* queue, in which the ack messages are put together with data messages (see figure 3(a)). The *delivery* queue is a FIFO queue.

As soon as a clusterhead receives all the local acknowledgements (`ack-messages`), it prepares the `cluster-ack` message (see figure 3(b)) that resumes the validation state within the cluster. This message must be sent to the originator clusterhead and the first attempt is to route the message through the FT. By

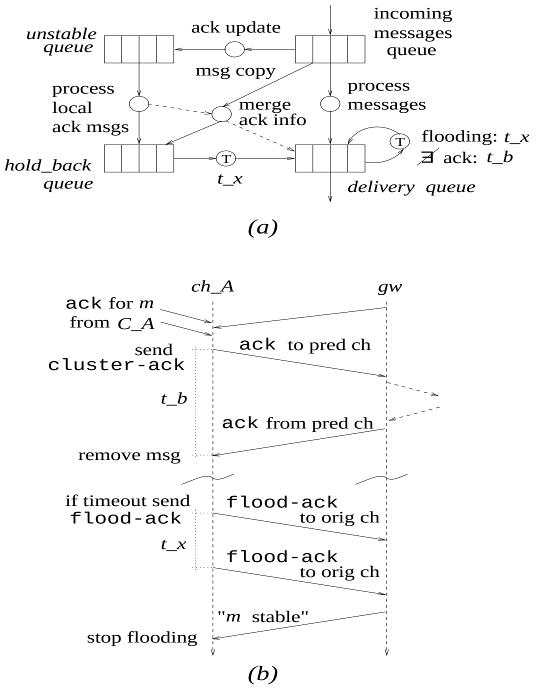

Figure 3: Ack processing.

utilizing the information in $unstable_i$, the clusterhead specifies, as a sort of source addressing, the final destination and the identity of the predecessor clusterhead (see figure 3(b)). If a gateway, in touch with the predecessor clusterhead, is still active, the message can perform backward one hop of the FT. The `cluster-ack` message requires to be validated at each hop within the time t_b. If the validation is not received, there is the possibility that the hop of the FT is no longer available. In this event, the clusterhead switches to flooding. A flooding acknowledgement (see `flood-ack` in figure 3(b)) flows to the originator clusterhead from cluster to cluster (to force a flooding message to propagate ahead, gateways can act as one-way gates). It must be retransmitted until the message it helps to validate becomes *stable*. The retransmission rate is a multiple of t_b. Often, it happens that within a time t_x (whose dimensioning could benefit of some knowledge about the network topology) after the forwarding of a data message, a clusterhead receives the acknowledgments for that message from the adjacent clusters. If the `cluster-ack` messages travel as single packets they waste bandwidth and hosts resources. The *hold-back* queue has been introduced to have some more chance to pack that messages into a single packet by merging the information they transport in the bit map. When a clusterhead prepares or receives from outside a `cluster-ack` and the delivery queue is high, the `cluster-ack` message receives a second chance for avoiding to travel alone. It enters the *hold-back* queue waiting for other `cluster-ack`'s transporting information about the same message. A deamon process is activated with a rate t_x. Its function is to move messages from the *hold-back* queue to the *delivery* queue.

4.2.2 Overall Message Format

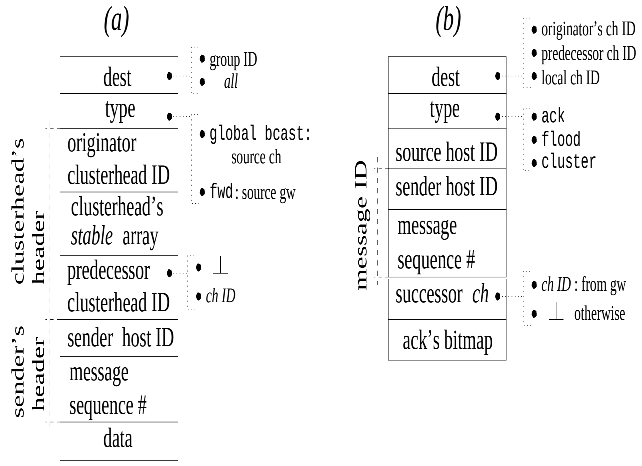

Figure 4: Layout of the message structures: (a) inter-cluster messages; (b) `ack`, `cluster-ack` and `flood-ack` messages.

In figure 4, we show the overall structure of the protocol messages: the inter-cluster and the acknowledgement messages. The former is used by clusterheads and gateways to propagate information through clusters. The second is used by mobile hosts to reply with an acknowledgement or by clusterheads to propagate cluster acknowledgements. When a host sends a data message to its clusterhead, it uses a message format similar to that of figure 4(a).

A rough evaluation of the message size can be obtained by assuming 1 byte to represent the host identifier and 1 bit to represent the *type* field. Let n be the number of message destinations and *data* the size of the data part of the message; under these assumptions the message sizes can be estimated as follows:
• $(8n + 41) + data$ bits for the inter-cluster messages (Figure $4(a)$);
• $n + 43$ bits for the ack messages (Figure $4(b)$).

4.3 The Protocol with Transient Failures and Host Mobility

In this section, we describe the mechanisms that the protocol adopts to deal with hosts mobility and with transient failures that affect communications. We still consider a static clusterhead infrastructure.

Communication failures are detected by using timeouts and recovered by means of retransmissions. Timers are used by both clusterheads and gateways which are the nodes responsible of recovering failures by means of retransmissions. A timer t_p is used in waiting an ack to unicast messages, specifically to the message a host sends to its clusterhead, while a timer t_b is used for the `ack-messages`, the `global bcast` messages and the `cluster-ack` messages; their dimensioning is outlined in section 6. Retransmissions generate message duplicates that are filtered by using the information in $stable_i$ and $unstable_i$.

Host mobility leads a processor p_i to move from a cluster, let's say C_A, to another cluster, let's say C_B. Two events are relevant to our discussion: *i*) p_i is sender of a message that is still unstable while p_i is moving, and, *ii*) p_i is the receiver of a message m, but it moves before receiving m from ch_A.

The case *i*) has three subcases: 1) ch_A never received m because of the motion of p_i or of communication failures, 2) ch_A and ch_B received m and ch_B has already diffused m in cluster C_B, 3) ch_A and ch_B received m and ch_B is still trying to diffuse m in cluster C_B. While the third case includes the problem solution, the first and the second require the execution of a simple protocol between the new clusterhead, ch_B, and the processor p_i. Once p_i enters cluster C_B, ch_B becomes aware of p_i (the information is available through the link level) and sends to it the ith entry of $stable_{ch_B}$ together with the identifiers of the p_i's messages that are still considered unstable by ch_B. p_i compares the received information with $stable_i$ and $unstable_i$ and, if it verifies that ch_B is missing some of the p_i's messages, they are sent to ch_B. This allows to solve subcases (1) and (2). In addition, p_i specifies its former role, i.e. clusterhead or node. To solve the problems when p_i is receiver, ch_B ought to be able to compare the local $unstable_{ch_B}$ data structure with $unstable_i$; to this purpose, p_i also sends to ch_B the list of the identifiers of the messages in $unstable_i$. If ch_B verifies that p_i is missing some message m, then it performs its retransmission and waits for the `ack-message` from p_i. The recovery terminates by sending to the originating clusterhead the collected `ack-messages`; they are required to stabilize m.

It is easy to observe from the above description that the more a host moves the higher is the probability of delaying the message stabilization.

4.4 The Protocol when Clusterheads Move

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's consider a cluster C_A and let's indicate with the term old_ch_A and new_ch_A respectively the former and the new clusterhead of C_A. Once new_ch_A is elected clusterhead (this responsibility pertains the underlying clustering protocol), as first public initiative in the new role, new_ch_A collects, for each host in cluster C_A, $stable_i$ and the message identifiers of the messages in $unstable_i$. That way, it obtains an updated versions of the local data structures. The implementation of this procedure can result to be costless if the exchanged information can be piggybacked onto the packets that the underlying protocol exchanges within the cluster to achieve similar objectives of consistency amongst the members of C_A.

At this point, only two cases must be considered. First, old_ch_A might have moved before having broadcasted a message m. In this event, no actions are undertaken because the sender will retransmit the message after the time t_p (see previous section). Second, new_ch_A could be unaware of the stability knowledge contained in $stable_{old\_ch_A}$. In the trivial case, old_ch_A still belongs to C_A and the problem is solved at the first step of the procedure. Otherwise, there is no alternative to go around asking for the clusterhead that currently knows old_ch_A. We have seen, in the previous section, that a clusterhead will eventually identify old_ch_A and its former role. However, provided that clusterheads do not know one another and that they are unaware of the network connectivity, two main questions arise: *i*) how to identify the cluster where old_ch_A moved and, *ii*) how to send back to new_ch_A the data structure *stable* of its clusterhead ? To give answer to these questions, a mechanism very similar to the one described to send data messages and to collect acknowledgements can be followed. For this reason we do not describe it in details. new_ch_A sends its enquiry to other clusterheads by constructing a FT. In this case, only the relevant clusterhead will respond by sending a copy of its *stable* data structure either through the FT or by flooding. The liveness property ensures that the procedure eventually terminates.

The described procedure is *not* blocking with respect to the generation of new messages, whose diffusion is started as soon as the new clusterhead has been elected.

5 Protocol Correctness

In this section, we sketch the proof of the protocol correctness; the complete proof is reported in the long version of the paper [13].

Claim 5.1 *Link failures are eventually recovered.*

This derives from the Liveness property and from the use of retransmissions. As a consequence, the protocol guarantees that, if a processor p sends a message m to a neighbour q and the network topology does not change, q eventually receives m.

By repeatedly applying the Claim 5.1 and by the protocol description, we are able to prove the following:

Claim 5.2 *In the case of static topology and transient link failures, if a clusterhead ch_A receives a message m*

then eventually m is received by both the hosts in C_A and all the clusterheads connected to ch_A via gateways.

Therefore, the following Lemma can be proved:

Lemma 5.1 *The protocol solves the Reliable Broadcast Problem in the case of interconnected static topology with transient link failures.*

Proof. The Integrity property easily follows from the protocol description. Let m be a message broadcast by p; thanks to Claim 5.1, m is eventually received by the p's clusterhead. By repeatedly applying Claims 5.1 and 5.2 we show that eventually all the hosts in the network receive m at least once. By the protocol, they all eventually deliver m, thus guaranteeing the Validity and Agreement properties. □

To prove the protocol correctness in the general case, we use the following claims.

Claim 5.3 *Let p be a host that moves throughout the system, so that the clusterhead infrastructure does not change. Let m_S a message broadcast by p and m_D a message addressed to p; let both messages be unstable at the time p moves. Then, eventually m_D is received by p and m_S is received by some clusterhead.*

Claim 5.4 *Let old_ch_A a clusterhead that is substituted by new_ch_A. Let m be an unstable message generated by a host p that was connected to old_ch_A. Then, eventually m is reliably broadcast by some clusterhead.*

To prove both these claims we exploit the Liveness property and the FIFOness of the *delivery* message queues.

The main result concerning the protocol correctness can now be easily proved.

Theorem 5.1 *The protocol solves the Reliable Broadcast problem in the considered environment.*

Proof. The Lemma 5.1 proves the theorem in the case of static topology. Claim 5.3 and the Liveness property guarantee that a message is eventually successfully broadcast by some clusterhead in the case of sender mobility. Claims 5.2 and 5.3 guarantee that the message is eventually received by all its destinations, even if they move. As a consequence, the theorem holds in the case of host movements that do not affect the clusterhead infrastructure. Claim 5.4 and the Liveness property guarantee that, if the clusterhead infrastructure changes, a message is eventually broadcast by some clusterhead and the previous arguments still apply, thus completing the proof. □

In the long version of the paper [13] we also show that, eventually, for each message m no more messages that concern m are generated and all the information about m are deleted from the hosts' memory. This is a desirable property in environments where the careful usage of the host and network resources is a critical topic.

6 Preliminary 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, $t_b \simeq 4t_r$ and $t_p \simeq 2t_r$; in the case of a WaveLAN wireless network with 2 Mbps of bandwidth the two timers may roughly last respectively 20 msec. and 10 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.

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 $O(n+2D+2)$.

7 Concluding Remarks and Future Work

We have described a reliable broadcast protocol intended for use in mobile multihop packet radio networks. The challenge here is to ensure fault tolerance despite communication failures and host mobility and, at the same time, to address efficiency by controlling resource utilization and the amount of exchanged messages. The protocol we have described is a first attempt in that direction; it provides a clear service semantics that can be exploited to construct value-added services or to provide direct support to applications according to a good use of the end-to-end argument. It can be adapted to a network topology with base stations.

We believe that this type of service is the fundamental block that will ease the design of fault tolerant distributed applications, even in a mobile environment without base stations. Nevertheless, it must be observed that a lot of work still remains to perform in different directions: i) to identify a closer cooperation between reliable broadcast and clustering protocols to obtain some more information necessary to quickly react to the topological change and dynamically construct the FT, ii) to provide multicast addressing facilities at the clustering level, iii) to tune the protocol and the flow control mechanism by continuing the simulations.

We are currently working in those directions. At the same time, we are designing a total ordering protocol on top of the reliable broadcast with the aim of experimenting the services to enforce consistency over replicated data in the frame of a distributed application.

References

[1] Acharya A., Badrinath B. R. "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, Vol. 1, pages 199–219, 1996.

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

[3] Amir Y., Moser L. E., Melliar-Smith P. M., Agarwal D. A., Ciarfella P. "The Totem Single-Ring Ordering and Membership Protocol". *ACM Trans. on Computer Systems, Vol. 13, No. 4*, pages 311–342, Nov. 1995.

[4] Bertsekas D., Gallager R. "Data Networks". Prentice-Hall, 1987.

[5] Birman K., Schiper A., Stephenson P. "Lightweight Causal and Atomic Group Multicast". *ACM Transactions on Computer Systems, Vol. 9, No. 3*, pages 272–314, Aug. 1991.

[6] Chandra T. D., Toueg S. "Time and Message Efficient Reliable Broadcast". *Proc. 4th International Workshop on Distributed Algorithms*, Vol. 486 of Lecture Notes in Computer Science:289–303, Sep. 1990. Springer, Berlin.

[7] Chiang C.-C., Gerla M. "Routing and Multicast in Multihop, Mobile Wireless Networks". *Proc. IEEE ICUPC'97*, Oct. 1997.

[8] Cho K., Birman K. P. "A Group Communication Approach for Mobile Computing". *Proc. Mobile Computing Workshop*, Santa Cruz, CA, Dec. 1994.

[9] Corson M. S., Ephremides A. "A Distributed Routing Algorithm for Mobile Wireless Networks". *Journal of Wireless Networks, Vol. 1, No. 1*, pages 61–81, Feb. 1995.

[10] Ephremides A., Wieselthier J. E., Baker D. J. "A Design Concept for Reliable Mobile Radio Networks with Frequency Hopping Signaling". *Proceedings of the IEEE, Vol. 75, No. 1*, pages 56–73, Jan. 1987.

[11] Gerla M., Tzu-Chieh Tsai J. "Multicluster, Mobile, Multimedia Radio Network". *ACM/Baltzer Journal of Wireless Networks, Vol. 1, No. 3*, pages 255–265, 1995.

[12] Kistler J. J., Satyanarayanan M. "Disconnected Operation in the Coda File System". *ACM Trans. on Computer Systems, Vol. 10, No. 1*, pages 3–25, Feb. 1992.

[13] Pagani E., Rossi G. P. "Reliable broadcast in mobile networks". *Technical Report, Dip. Scienze dell'Informazione, Università degli Studi di Milano*, 1997.